Thèse de Doctorat de l'université Paris VI
Pierre et Marie Curie

Spécialité

Systèmes Informatiques

présentée par

M. Ricardo Cassiano Nabhen

pour obtenir le grade de

Docteur de l'université Pierre et Marie Curie

Fuzzy logic based method for modeling queue behavior of network nodes

Soutenance prévue le 10 juillet 2009 devant le jury composé de

Rapporteur	**M. Samir TOHMÉ**	**Professeur à l'UVSQ**
Rapporteur	**M. James ROBERTS**	**Directeur unité R&D, France Telecom**
Examinateur	**M. Michel MINOUX**	**Professeur à l'Université PMC**
Examinateur	**M. Harry PERROS**	**Professeur à NCSU**
Examinateur	**M. Manoel Camillo PENNA**	**Professeur à PUCPR**
Examinateur	**M. Mauro FONSECA**	**Professeur à PUCPR**
Examinateur	**Mme. Anelise MUNARETTO**	**Professeur à l'UTFPR**
Directeur de thèse	**M. Guy PUJOLLE**	**Professeur à l'Université PMC**

Thèse de Doctorat de l'université Paris VI
Pierre et Marie Curie

Spécialité

Systèmes Informatiques

présentée par

M. Ricardo Cassiano Nabhen

pour obtenir le grade de

Docteur de l'université Pierre et Marie Curie

Fuzzy logic based method for modeling queue behavior of network nodes

Soutenance prévue le 10 juillet 2009 devant le jury composé de

Rapporteur	**M. Samir TOHMÉ**	**Professeur à l'UVSQ**
Rapporteur	**M. James ROBERTS**	**Directeur unité R&D, France Telecom**
Examinateur	**M. Michel MINOUX**	**Professeur à l'Université PMC**
Examinateur	**M. Harry PERROS**	**Professeur à NCSU**
Examinateur	**M. Manoel Camillo PENNA**	**Professeur à PUCPR**
Examinateur	**M. Mauro FONSECA**	**Professeur à PUCPR**
Examinateur	**Mme. Anelise MUNARETTO**	**Professeur à l'UTFPR**
Directeur de thèse	**M. Guy PUJOLLE**	**Professeur à l'Université PMC**

10/07/2009

To my wife, Márcia, and my children,
Jacqueline and Eduardo.

Acknowledgements

I am very grateful to Edgard Jamhour, Manoel Camillo de Oliveira Penna Neto and Guy Pujolle. They have not only provided a very valuable guidance, but also a lot of inspiration, motivation, experience sharing and tolerance. Thanks to Mauro Sérgio Pereira Fonseca for the cooperation with this research work. Also, I would like to thank Carlos Alberto Maziero for his contribution related to the simulation environment.

Finally, I want to dedicate this work to my wife, Márcia Nabhen, my daughter, Jacqueline Nabhen, and my son, Eduardo Nabhen, for their support, encouragement and tolerance.

Abstract

Capacity planning of IP-based networks is a difficult task. Ideally, in order to estimate the maximum amount of traffic that can be carried by the network, without violating QoS requirements such as end-to-end delay and packet loss, it is necessary to determine the queue length distribution of the network nodes under different traffic conditions. This relationship is strongly dependent on the incoming traffic profile and the queuing discipline adopted by the network node. Analytical models for queue length distribution are available only for relatively simple traffic patterns. The characterization of a model for the queue length distribution is a vastly studied subject, but the associated mathematical apparatus becomes more and more complicated when we have to deal with multiple priority queues, non-exponential service times and long term correlated traffic. Also, when per-flow guarantees are required, it is necessary to determine the impact of the queue behavior on the performance of individual flows. This thesis addresses the design of performance models of network nodes. We propose a generic method for modeling queue behavior aiming to provide a methodology for building the corresponding performance models. We take into account scenarios in which it is very hard to develop an analytical model. The proposed method combines non-linear programming and simulation to build a fuzzy model capable of determining the performance of a network node. Using such strategy, which is based on a general off-line method to produce the equations capable of representing outputs for the whole space of the specified input traffic parameters, it is possible to find out the optimal values that can be used for the configuration of network nodes, with important applications on traffic engineering and capacity planning. This approach does not require the derivation of an analytical model and can be applied to any type of traffic. Also, this methodology includes a training method that permits the application of any type of performance metric.

Key Words:

Fuzzy system models, non-linear programming, simulation, network capacity planning.

Table of contents

Chapter 1

Introduction

1.1 Motivation

The end-to-end data transfer in network applications may be viewed as a packet level transport activity using network services that offer Quality of Service(QoS)-enabled connectivity between endpoints. For addressing QoS needs, the DiffServ approach [1] defines that network traffic can be classified into distinct aggregated classes, according to its performance requirements. For example, VoIP traffic can be assigned to a class designed to provide low delay while data traffic can be assigned to a class designed to provide low packet loss. A DiffServ node can deal with those distinct performance requirements by employing multiple queues. In order to determine the maximum amount of traffic that can be admitted into a DiffServ domain, it is necessary to determine the relationship between the traffic load and the queue length distribution of the DiffServ nodes. This relationship is strongly dependent on the incoming traffic profile and the queuing discipline adopted by the DiffServ node. The characterization of a model for the queue length distribution is a vastly studied subject, but the associated mathematical apparatus becomes more and more complicated when we have to deal with multiple priority queues, non-exponential service times and long term correlated traffic.

QoS methodologies that support resource reservation, such as IntServ [2] and MPLS [3], permit to establish paths by selecting links with available resources, thus ensuring that the bandwidth is always available for a particular flow or flow aggregate. However, for both methodologies, it is necessary to determine the amount of resources required by a certain traffic profile in order to meet QoS requirements. Additionally, the IETF has defined strate-

gies for combining MPLS and DiffServ methodologies [4]. In these scenarios, a LSP (Label Switched Path) can be shared by distinct traffic classes or two or more LSPs with distinct QoS requirements can follow identical paths. For example, consider the illustration depicted in Figure 1.1. A MPLS domain is supposed to provide a Virtual Private Network(VPN) between LAN 1 and LAN 2 with a specific QoS requirement. Based on the support given by RSVP-TE [5] messages, a LSP can be established through a initial PATH message sent by the ingress router towards the egress router which informs the traffic description through included SENDER_TSPEC objects [6]. The LSP tunnel is completed upon the arrival of a positive RESV message which is sent backwards to the ingress router by the egress router with FLOWSPEC objects [6].

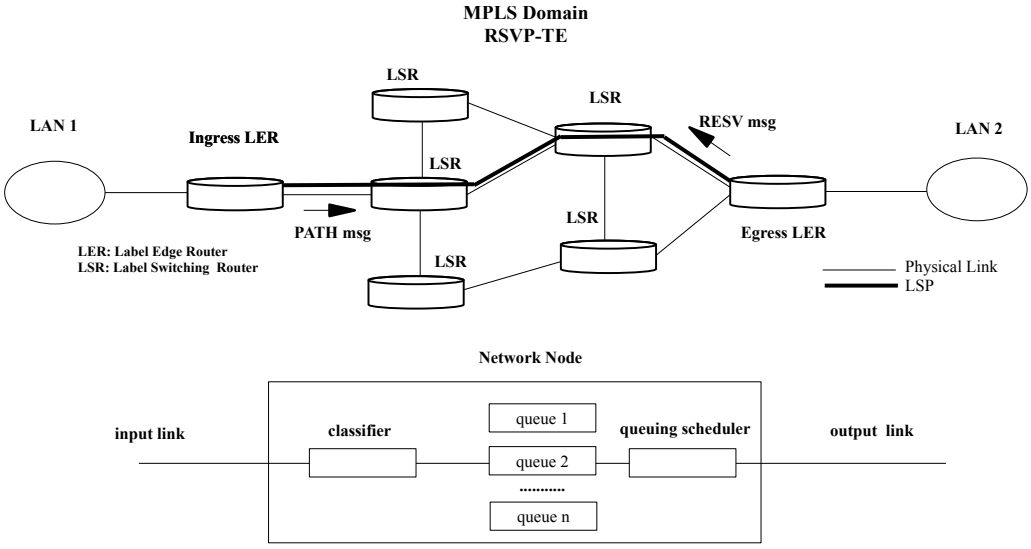

Figure 1.1: A MPLS/DIFFSERV scenario

As can be noted in this example, it is important to know in advance the performance (e.g., in terms of packet delay and loss rate) that will be experienced by incoming traffic profiles in order to obtain the optimal values for the configuration of LSP tunnels aiming to meet QoS needs. In this case, this could help to define QoS requirements of the traffic which is specified in SENDER_TSPEC objects. At the node level, these distinct requirements are satisfied by employing distinct queues, as illustrated in the bottom part of Figure 1.1. Internally, the traffic is classified and assigned to a corresponding queue by a classifier. The QoS performance of a queue can be represented in terms of the amount of loss, delay and

jitter imposed by the queue on the traffic injected to the output link. For the pure DiffServ approach or the combined DiffServ/MPLS methodology, it is necessary to determine the maximum amount of traffic that can be assigned to each queue without violating the QoS requirements. The performance model of network nodes (i.e., a multi-queue node in this example), where it is possible to establish the relationship between the traffic load at each queue and the resulting QoS performances, can help to address this issue.

Analytical models for queue length distribution are available only for relatively simple traffic profiles. An effective model needs to capture all relevant stochastic components that impact its target QoS metrics which can lead to extremely complex models [7], maybe infeasible for practical purposes. On the other hand, simpler models may not represent accurately real scenarios tending to overestimate or underestimate the actual performance [8]. Moreover, in most cases, these models apply only to the performance of the aggregated traffic. It is however pointed by some authors that under certain conditions, the delay, jitter and packet loss level experienced by individual flows can significantly diverge with respect to the performance of the aggregated traffic [9, 10]. Consequently, even when the aggregated traffic is properly dimensioned, the individual flows may not satisfy QoS requirements such as delay and packet loss. Providing per-flow guarantees poses additional difficulties because it is also necessary to take into account the fluctuations of the flows lifecycles in order to determine the impact of the queue behavior on the performance of the individual flows. For instance, consider the case of VoIP applications. Referring again to Figure 1.1, suppose that one needs to provide per-flow guarantees in terms of a maximum delay for the 99^{th} percentile of packets of each VoIP flow through a LSP tunnel that is dimensioned for a VoIP traffic aggregate. Also, another concurrent LSP tunnel is established along the same physical path in order to transport data traffic which has a specific packet loss rate as a requirement. In this case, each LSP will be assigned to distinct queues where they will compete for the same output link. This scenario poses some questions: How much traffic must be admitted in each queue in order to meet both QoS needs? Considering the DiffServ approach, is the dimensioning of the VoIP aggregated traffic enough to achieve per-flow guarantees? What is the impact of VoIP flows on the performance of the data traffic and vice-versa since they compete for the same output link? Thus, building a performance model that could take into account these issues can help service providers to implement traffic engineering policies.

1.2 Objectives and Contributions

The objective of this thesis is to develop a generic method for modeling the queue behavior of network nodes aiming to provide a methodology for building the corresponding performance models. The proposed method takes into account many of the complicating modeling issues stated before, leading to a scenario in which it is very hard to develop an analytical model. The proposed method combines non-linear programming (NLP) and simulation to build a model capable of determining the performance of a network node. Using such strategy, which is based on a general off-line method to produce the equations capable of representing outputs for the whole space of the specified input traffic parameters, it is possible to find out the optimal values that can be used for the configuration of network nodes, with important applications on traffic engineering and capacity planning. This approach does not require the derivation of an analytical model and can be applied to any type of traffic. Also, this methodology includes a training method that permits the application of any type of performance metric.

Capturing uncertainties related with such performance modeling is a key issue in devising a model that can actually represent the behavior of a system. Due to the complexity of such modeling process, our method uses fuzzy techniques in order to represent queue models. As stated in [11], "as the complexity of a system increases, our ability to make precise and yet significant statements about its behavior diminishes until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive characteristics". Fuzzy logic may be seen as a tool for dealing with uncertainty. It provides a method to deal with imprecision. Fuzzy modeling is achieving successful results on representing non-linear uncertain systems and has been used in many different investigations of network problems. Thus, our proposed method employs NLP and simulation to build a fuzzy logic based performance model of a network node.

As a direct consequence of this approach, our method can also be employed for designing DiffServ admission controllers and predictors. For example, consider the case of building a Parameter Based Admission Control(PBAC) controller for a VoIP traffic aggregate. The idea is not to permit the admission of a new flow that could violate QoS requirements of admitted traffic. Based on the results obtained from the performance model built with our method, it is possible to predict both aggregate and per-flow QoS performances and the admission control decision could easily be performed.

During the development of this research work we presented some preliminary results and

conclusions that have contributed for this study. In the first publication [12] we investigated the variation of individual flows performance with respect to the aggregate performance. In the three next [13, 14, 15], our concern was to study optimization algorithms, training methods, traffic models and simulation approaches in order to provide guarantees for individual flows and aggregate traffic. The studies showed in these four papers formed the basis for the main publication [16], which formalizes our method presentation. It introduces our fuzzy logic approach and training method to create performance models, which is detailed in Chapter 5 and validated in Chapter 6. In the following, we present a brief summary of each one of these publications.

In [12], we presented a study based on simulations that makes an analysis of the end-to-end delay observed by individual flows and how much it can diverge from that experienced by the aggregated traffic. The motivation of this evaluation is derived from the fact that SLA management approaches typically adopt provisioning strategies based on aggregated traffic in order to support end-to-end delay requirements of applications. They do not take into account individual flow needs. Several scenarios were used to evaluate this performance and some metrics were proposed to investigate empirical relations that show the end-to-end delay behavior when individual flows, the aggregated traffic and the network load are analyzed. The results show that the delay can be much higher than the one observed by the aggregated traffic, causing an important impact in the performance of network applications. Based on the previous study, in [13], we presented a study that intends to investigate the QoS deployment in DiffServ networks for delay sensitive applications considering performance guarantees established in a per-flow basis. For this purpose, we presented an evaluation methodology based on optimization methods that implements several measurement based admission control algorithms. We showed that it is possible to use the system resources efficiently and to meet the QoS requirements in a per-flow basis taking into account distinct bandwidth provisioning and performance metrics.

The work presented in [14] shows a study about the design of admission control algorithms and its impact in the performance of individual flows. Most DiffServ admission control algorithms rely on tuning parameters to help in the decision making. Tuning these parameters is a difficult task, especially when one considers the problem of assuring QoS guarantees to individual flows. In this study, we propose a method for helping the design of DiffServ AC algorithms based on non-linear programming optimization. It enables the finding of the values for the AC parameters that permits to satisfy the QoS guarantees for individual VoIP flows, while minimizing a cost function that represents the performance

goals of the service provider. This approach is used for comparing the performance of some commonly used DiffServ AC techniques and also to design a novel AC algorithm based on queue estimates. In [15], we address the issue of determining the maximum amount of traffic that can be admitted in a DiffServ network. The relationship between the traffic load and the queue length distribution of a DiffServ node is very difficult to model. In this study, we demonstrate how a non-linear programming algorithm can be employed to determine the maximum load that can be accepted by a DiffServ node without deriving an analytical model. The NLP algorithm is used to train a parameter based admission control controller using a specifically designed traffic profile. After training the PBAC for a specific network and specific statistical QoS guarantees, it can be used to provide these guarantees to distinct offered traffic loads. In this work, we evaluated such approach in a sample scenario where (aggregated on-off) VoIP traffic and (self-similar) data traffic compete for network resources.

Fuzzy logic was introduced in our work in order to carry out the modeling process of queue behavior using the studies previously developed. In [16], we propose a generic method for building a fuzzy predictor for modeling the behavior of a DiffServ node with multiple queues. The method combines non-linear programming and simulation to build a fuzzy predictor capable of determining the performance of a DiffServ node subjected to both: per-flow and aggregated performance guarantees without utilization of analytical models. In this work, we employ the fuzzy logic approach to model the behavior of a multi-queue node where (aggregated on-off) VoIP traffic and (self-similar) data traffic compete for network resources.

1.3 Organization of the thesis

The thesis is structured as follows. Chapter 2 introduces the fuzzy logic and presents key issues related to modeling based on fuzzy techniques. Also, we show an example in order to illustrate the main concepts and procedures presented. Chapter 3 presents the traffic models and related issues with respect to synthetic traffic generation and performance measurement, which are the basis for the optimization problem formulation of this research work. Our simulation approach for self-similar traffic and per-flow modeling (VoIP traffic) is also discussed. Then, we describe the simulation environment developed for the method evaluation. In Chapter 4, we first review important concepts and definitions related to optimization theory. Then, we present the optimization method employed in this research

work which is followed by an example that shows the design of a parameter based admission control mechanism using a training strategy based on simulations and the optimization method discussed.

Chapter 5 describes the proposed method. First, we show how the method can be employed for building a fuzzy predictor considering a scenario with a multi-queue node where VoIP traffic and constrained self-similar traffic compete for system resources. After that, the method is employed for building a time based fuzzy model for the MVA approximation for loss probability analytical model. We show a strategy based on normalized probability distributions in order to produce output fuzzy sets for representing the performance model of a network node in terms of loss probability. Chapter 6 provides the validation of our contributions. We consider three different scenarios, two for the fuzzy predictor evaluation and one for the MVA approximation for loss probability analytical model. Finally, Chapter 7 concludes this thesis and discusses future directions of research.

Chapter 2

Fuzzy Systems

2.1 Introduction

Fuzzy modeling is achieving successful results on representing non-linear uncertain systems and has been used in many different investigations of network problems. This is because controlling the behavior of some systems requires complex mathematical models, which is often very difficult and sometimes even impossible, making the modeling based on fuzzy techniques an appealing approach. Fuzzy logic may be seen as a tool for dealing with uncertainty, making available a method to deal with imprecision.

After the main ideas of fuzzy sets and their basic relations introduced in Zadeh′s seminal work [17], Zadeh introduces the concept of linguistic variables and fuzzy algorithms [11], that represent a key feature of the fuzzy modeling: "It has three main distinguishing features: 1) The use of so-called linguistic variables instead of or in addition to numerical variables; 2) characterization of simple relations between variables by conditional fuzzy statements; and 3) characterization of complex relations by fuzzy algorithms". Furthermore, it defines its objective: "By relying on the use of linguistic variables and fuzzy algorithms, the approach provides an approximate and yet effective means of describing the behavior of systems which are too complex or too ill-defined to admit of precise mathematical analysis. Its main applications lie in economics, management science, artificial intelligence, psychology, linguistics, information retrieval, medicine, biology, and other fields in which the dominant role is played by the animate rather than inanimate of system constituents". Since then, several modeling works based on such approach have been proposed, taking advantage of the *computation with words*, which has made it appropriate for the modeling of complex

systems, mainly when it is necessary to consider a problem with several variables. The method proposed in this thesis is directly related with this statement. In order to build a complex network model, we consider several parameters that describe the network node and those in which the target performance model is dependent, leading typically to a model with tens of variables. This chapter presents the main concepts related to fuzzy modeling and summarizes the mathematical background employed by the fuzzy logic. In the final of this chapter, we show an example in order to illustrate the main concepts and procedures presented.

2.2 Basic Concepts

2.2.1 Crisp Sets and Fuzzy Sets

The fuzzy approach is based on the idea of *fuzzy sets* [17]. In order to understand this concept, it is also important to define *crisp sets*. The fuzzy logic provides an inference structure that enables appropriate human reasoning capabilities. On the contrary, the traditional binary set theory describes crisp events, which either do or do not occur. It uses a probability theory to explain if an event will occur, measuring the chance with which a given event is expected to occur. The theory of fuzzy logic is based on the notion of relative graded membership and so are the functions of mental and cognitive processes [18]. Consider the example shown in Figure 2.1. Suppose that it is necessary, hypothetically, to represent the end-to-end one-way delay related to the perceived quality of voice due to transmissions in communication networks. According to the ITU-T recommendation G.114 [19], delays of less than $150\,ms$, most applications will experience essentially transparent interactivity, whereas delays above $400\,ms$ are unacceptable for general network planning purposes. Between both values, we can consider that there is an average range.

One can see in this example that we define three *linguistic* terms in order to classify the end-to-end one-way delay: *low*, *average* and *high*. Also, we are not interested in representing delays above $500\,ms$. So, we say that the *universe of discourse* of the delay variable is $0 \leq delay \leq 500$. A *universe of discourse* of a variable can be discrete or continuous spaces. In the first graph, any delay value belongs only to one group or *crisp set*. For example, a $212\,ms$ delay is always classified as *average*. It has a *zero* degree of membership in *low* and *high* crisp sets. However, considering the fuzzy approach, a $212\,ms$ delay belongs to two groups or *fuzzy sets*: *low* and *average*, with both having the same degree of membership:

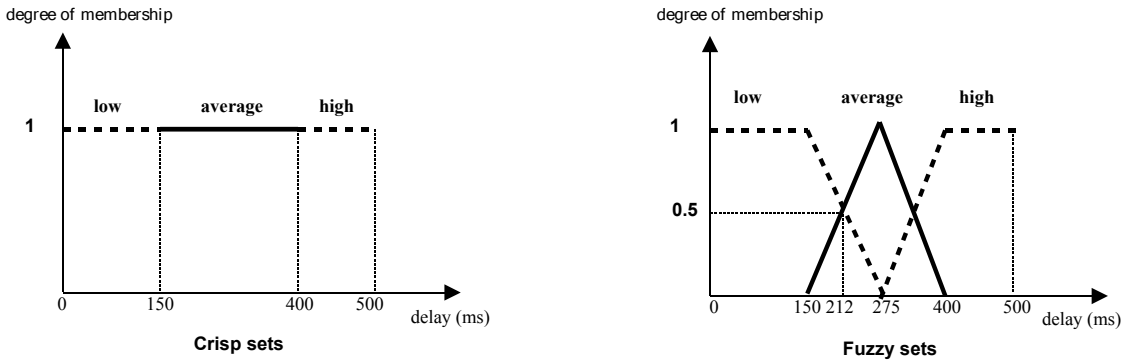

Figure 2.1: Example: Representing the end-to-end one-way delay of recommendation G.114 with Crisp and Fuzzy sets

0.5. Formally, we can now define crisp and fuzzy sets. Given an universe of discourse X, where $x \epsilon X$, a *fuzzy set* V in X is defined by a *membership function* $\mu_V(x)$ which gives, for any value of x, the degree of membership of x in V. This definition can be expressed as follows:

$$V = \{x, \mu_V(x)\} \; for \; x \in X, \; where \; 0 \leq \mu_V(x) \leq 1 \tag{2.1}$$

in the case of the *crisp set*, we have:

$$V = \{x, \mu_V(x)\} \; for \; x \in X, \; where \; \mu_V(x) = \begin{pmatrix} 1, & x & \in & V \\ 0, & x & \notin & V \end{pmatrix} \tag{2.2}$$

A universe of discourse can be mapped to one or more partially overlapped fuzzy sets. Usually, fuzzy sets are named by *linguistic values* that permit to intuitively describe their meaning. For example, the fuzzy sets *low* ($low = \{d, \mu_{low}(d)\}$), *average* ($average = \{d, \mu_{average}(d)\}$) and *high* ($high = \{d, \mu_{high}(d)\}$) could be used to describe the universe of discourse of a variable D, representing the end-to-end one-way delay. The names *low*, *average* and *high* correspond to the *linguistic values* that the *linguistic variable* D can assume.

The overlapping feature of fuzzy sets permits a *smoother transition* between groups, providing more flexibility for modeling. For example, referring again to the second graph of Figure 2.1, starting at $150\,ms$, one can easily see that higher delay values gradually lose their degree of membership in the *low fuzzy set*, while they have also an increasing degree of membership in the *average fuzzy set*. This can be seen as a more realistic approach since a delay value close to $150\,ms$, say $155\,ms$, can be neither classified entirely as *average* nor has

a zero grade as *low*. In this example, the shape and width of each fuzzy set were arbitrarily chosen. One can see that below $150\,ms$ and above $400\,ms$ degrees of membership are the same as in the case of crisp sets, indicating that delay values within these ranges strictly belong to their respective groups, *low* and *high*. However, in the intermediate range *average*, it is preferable to design a smoother transition because one can not accurately determine at what level a delay value moves completely to another group. Hence, there is an overlapping region between the fuzzy sets in this range which was modeled as a triangle. Indeed, this flexibility is achieved by the definition of the shape of membership functions, allowing to capture information in which the limits are not strictly defined.

2.2.2 Membership Functions

As could be seen in the right part of Figure 2.1, three different *membership functions* were employed to define each linguistic value: One triangular and two trapezoidal shapes. One issue to be faced during the fuzzy set modeling is to determine the shape of the sets. According to fuzzy set theory, the choice of the shape and width is subjective [20]. Clearly, many other choices for the shape of membership functions are possible and these will each provide a different meaning for the linguistic values that they quantify [21]. Figure 2.2 presents some common shapes of membership functions. In the literature, there exist other several types of membership as, for instance, Gaussian, Generalized Bell, Sigmoidal and Z shaped membership functions [20], but here we will consider only those employed in this research work.

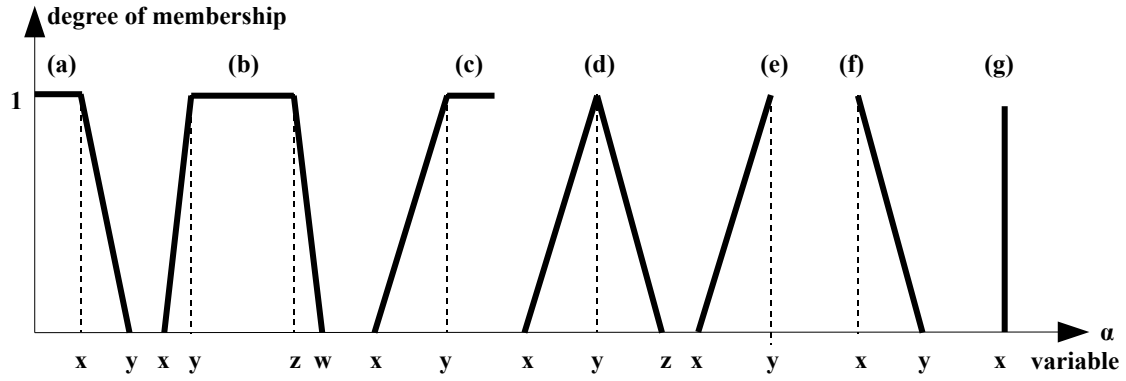

Figure 2.2: Most common shapes of membership functions: (a,b,c)-Trapezoidal shapes, (d,e,f)-Triangular shapes,(g)-Singleton shape

All fuzzy sets represented by these membership functions are classified as *normal fuzzy sets*. Normal fuzzy sets are implemented by membership functions in which there is at least one element in the universe of discourse whose value is equal to 1. However, a fuzzy set is known as a *subnormal fuzzy set* if its membership functions contains membership values less than 1 [18]. Table 2.1 shows the mathematical implementation of the membership functions presented in Figure 2.2 considering a fuzzy set V and a variable named α. The singleton shape represents one single point in the universe of discourse of the variable.

Table 2.1: Implementation of the membership functions shown in Figure 2.2

Membership Function	Mathematical representation
Trapezoidal Shape (a)	$\mu_V(\alpha) = \begin{pmatrix} 1, & \alpha < x \\ \frac{y-\alpha}{y-x}, & x \leq \alpha \leq y \\ 0, & y < \alpha \end{pmatrix}$.
Trapezoidal Shape (b)	$\mu_V(\alpha) = \begin{pmatrix} 0, & x > \alpha \\ \frac{\alpha-x}{y-x}, & x \leq \alpha \leq y \\ 1, & y \leq \alpha \leq z \\ \frac{w-\alpha}{w-z}, & z \leq \alpha \leq w \\ 0, & w < \alpha \end{pmatrix}$.
Trapezoidal Shape (c)	$\mu_V(\alpha) = \begin{pmatrix} 0, & \alpha < x \\ \frac{\alpha-x}{y-x}, & x \leq \alpha \leq y \\ 1, & \alpha > y \end{pmatrix}$.
Triangular Shape (d)	$\mu_V(\alpha) = \begin{pmatrix} 0, & x > \alpha \\ \frac{\alpha-x}{y-x}, & x \leq \alpha \leq y \\ \frac{z-\alpha}{z-y}, & y \leq \alpha \leq z \\ 0, & z < \alpha \end{pmatrix}$.
Triangular Shape (e)	$\mu_V(\alpha) = \begin{pmatrix} 0, & x > \alpha \\ \frac{\alpha-x}{y-x}, & x \leq \alpha \leq y \\ 0, & y < \alpha \end{pmatrix}$.
Triangular Shape (f)	$\mu_V(\alpha) = \begin{pmatrix} 0, & x > \alpha \\ \frac{y-\alpha}{y-x}, & x \leq \alpha \leq y \\ 0, & y < \alpha \end{pmatrix}$.
Singleton Shape (g)	$\mu_V(\alpha) = \begin{pmatrix} 0, & x > \alpha \\ 1, & \alpha = x \\ 0, & x < \alpha \end{pmatrix}$.

2.2.3 Basic Properties

Figure 2.3(a) and 2.3(b) show the basic properties of membership functions for, respectively, a normal and a subnormal fuzzy set. They can be defined as follows [22]:

$$\begin{aligned} support(V) &= \{\, x \,|\, x \in X \, and \, \mu_V(x) > 0\} \\ core(V) &= \{\, x \,|\, x \in X \, and \, \mu_V(y) \leq \mu_V(x),\ \forall y \in X\} \\ height(V) &= sup\, \mu_V(x),\ \forall x \in X \end{aligned} \tag{2.3}$$

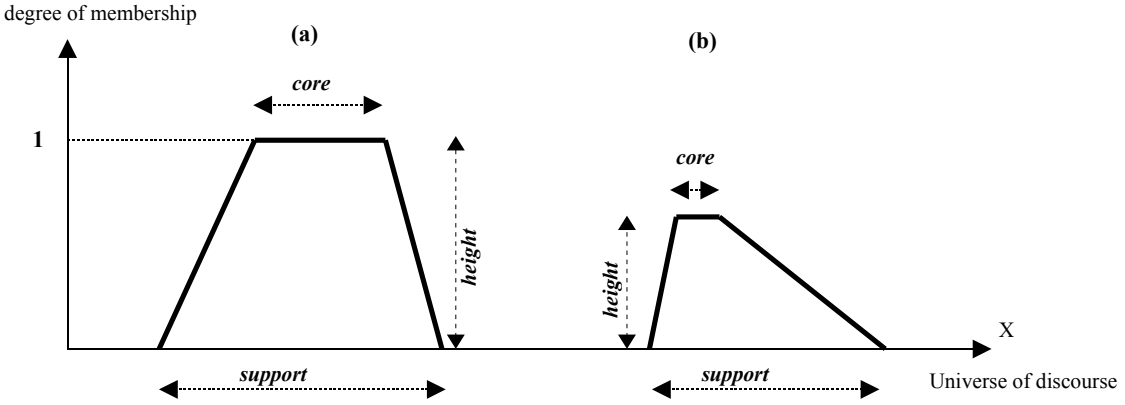

Figure 2.3: Basic properties of membership functions

Fuzzy set theory is strongly based on operations on sets. Figure 2.4 shows two important basic operations which serve as a foundation for fuzzy set operations: *Union* (a) and *Intersection* (b). These operations are also known as disjunction and conjunction, respectively. We can define the following relations according to the operations shown in Figure 2.4.

$$\begin{aligned} (a)\, Union &: W(x) = V(x) \cup Z(x) = max(\mu_V(x), \mu_Z(x)) \\ (b)\, Intersection &: W(x) = V(x) \cap Z(x) = min(\mu_V(x), \mu_Z(x)) \end{aligned} \tag{2.4}$$

where $V(x) = \mu_V(x)$, $Z(x) = \mu_Z(x)$ and $W(x) = \mu_W(x)$.

2.3 Fuzzy Logic System

In the proposed method of this thesis, it is necessary to build a performance model of a network node. As stated before, our problem is to create a system that could take into account several input parameters related to the input traffic profile, performance metrics

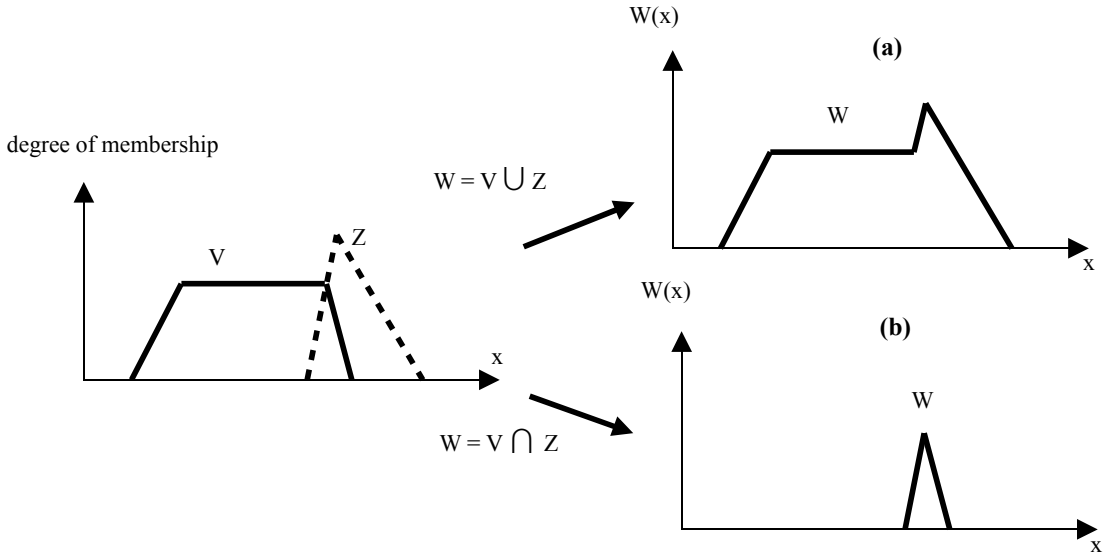

Figure 2.4: Basic operations: (a) Union (b) Intersection

and the network node in order to generate such performance model. In the context of this research work, we need to build a model considering the utilization of a *Fuzzy Logic System* (FLS). A FLS is a non-linear system capable of inferring complex non-linear relationships between input and output variables. The non-linearity property is particularly important when the underlying physical mechanism to be modeled is inherently non-linear. The system can learn the non-linear mapping by being presented a sequence of input information and desired response pairs, which are used in conjunction with an optimization algorithm to determine the values of the system parameters [23]. One interesting characteristic of a FLS is that it is independent of internal elements of the system, i.e., a FLS makes its computation collecting data from the target system and produce results based on a reasoning process without the necessity of any particular information related to the implementation of the system, e.g., simulation processes, statistical properties, etc..

Figure 2.5 presents the structure of a fuzzy logic system. Spaces I and O contain all fuzzy sets of the input and output spaces respectively. The fuzzy logic system is composed by a set of if-then rules that maps the input to the output fuzzy sets. The fuzzy approach can be used to design a logic procedure that interferes (e.g., a fuzzy controller) or not (e.g., a fuzzy predictor) on the system behavior. As illustrated in the figure, the FLS receives crisp input values and outputs crisp values. Thus, a FLS can be viewed as a non-linear

mapping between an input variable x and an output variable y, which can be expressed as $y = f(x)$. To output a result, the FLS performs three main steps: *fuzzification*, *inference* and *defuzzification*.

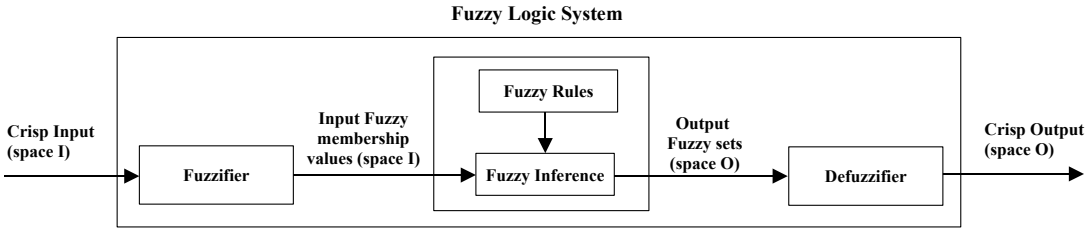

Figure 2.5: Structure of a Fuzzy Logic System

2.3.1 Fuzzification

The fuzzification is the process of computing the membership values from crisp inputs. It is a transformation process: Crisp value ⇒ Fuzzy value, where all information related to uncertainty and imprecision must be taken into account. It corresponds to the conversion of each input crisp data into its corresponding linguistic variables according to computed degrees of membership. For example, in the previous Figure 2.1, the fuzzification of the delay value $212\,ms$ is $\mu_{low}(212) = 0.5$ and $\mu_{average}(212) = 0.5$. Hence, the fuzziness is clear: a $212\,ms$ is 0.5 *low* and 0.5 *average*, i.e., we are not sure about what a $212\,ms$ delay value is precisely. In this example, we have a *singleton* fuzzification, however, it is possible to have non-singleton fuzzification [24], but this process adds more complexity to the FLS and requires higher computational resources. Thus, the *singleton* fuzzification is more widely used than non-singleton fuzzification. Throughout this thesis we will only employ the *singleton* fuzzification in the proposed method, as well as in further discussions about FLS.

2.3.2 Fuzzy rules

The reasoning process performed by the inference engine of a FLS is based on fuzzy rules. They are linguistic if-then statements involving fuzzy sets. Fuzzy rules express the knowledge of a system and assign the input variables to output variables. The rule-based form uses linguistic variables as its antecedents and consequents. The antecedents express an inference or the inequality, which should be satisfied. The consequents are those which we

can infer, and is the output if the antecedent inequality is satisfied [18]. The general form of a fuzzy rule is represented as follows:

$$IF\ antecedent\ THEN\ consequent \tag{2.5}$$

The *antecedent* is expressed in terms of input fuzzy sets, whereas the *consequent* is expressed in terms of output fuzzy sets, respectively, spaces I and O in Figure 2.5. For example, suppose we consider again the example of the end-to-end one-way delay presented in Figure 2.1 in which we are interested in mapping the end-to-end one-way delay to the perceived quality of speech. For delay values belonging to the *low* fuzzy set we know, according to the cited recommendation, that the perceived quality of speech is *good*. In this case, denoting the variable d as the end-to-end one way delay, the variable q as the perceived quality of speech, *low* as the input fuzzy set [1] and *good* as an output fuzzy set in order to classify the quality of speech, we can express this rule as follows:

$$\text{IF } d \text{ is } low \text{ THEN } q \text{ is } good, \quad \text{where } d \in I \text{ and } q \in O$$

In the literature, we have the two most important types of fuzzy rules: Mamdani fuzzy rules and Takagi-Sugeno(TS) fuzzy rules. Both types are very similar, but the main difference is related to the specification of consequents.

2.3.2.1 Mamdani FR

The Mamdani approach [25] is the most common type of fuzzy rules system. Broadly speaking, it can be seen as a rule set that describes a system by using linguistic statements. A Mamdani fuzzy rule set can be expressed as follows (adapted from [26]):

$$R^l: \ IF\ u_1\ is\ F_1^l\ and\ u_2\ is\ F_2^l\ and\ ....\ u_p\ is\ F_p^l\ THEN\ v_1\ is\ G_1^l\ ,....,\ v_q\ is\ G_q^l \tag{2.6}$$

where $l = 1, 2, ..., t$ for t denoting the number of rules, F_i^l and G_j^l are fuzzy sets in $U_i \subset \Re$, $i = 1, 2, .., p$ and $V_j \subset \Re$, $j = 1, 2, .., q$, respectively, in the input and output spaces. Also, $u_i \in U$ and $v_j \in V$. i is an index for denoting the set of input variables, whereas j is the same for output variables.

2.3.2.2 Takagi-Sugeno FR

Takagi-Sugeno(TS) fuzzy rules [27] can be seen as a representation of the behavior of a system through a linear model expressed by each rule. TS fuzzy rules change the consequent

[1] as shown in Figure 2.1

part formed by fuzzy sets with a function made of the input variables. Hence, we can see each rule as a specific linear model whose output result is aggregated with others to generate the final result. A TS fuzzy rule set can be expressed as follows:

$$R^l: \ IF\ u_1\ is\ F_1^l\ and\ u_2\ is\ F_2^l\ and\ ....\ u_p\ is\ F_p^l\ THEN\ v^l\ is\ f^l(u_1, u_2, ..., u_p) \qquad (2.7)$$

where $l = 1, 2, ..., t$ for t denoting the number of rules, F_i^l are input fuzzy sets in $U_i \subset \Re$, $i = 1, 2, .., p$, in the input spaces given by the universe of discourse $U = \{U_1, U_2, ..., U_p\}$ with $u_i \in U$. The consequent part, $v^l\ is\ f^l(u_1, u_2, ..., u_p)$, represents a function of input variables u_i. It is possible to expand the linear combination to a non-linear combination of input variables; for example, fuzzy rules which have neural networks in their consequents [28]. Typically, first order linear functions of inputs have been used for consequents. However, there exists a very typical case when we have $v^l\ is\ k$, i.e., the value assigned as a result of each rule is a *constant*. In this case, we have a *zero order TS rule*, which is quite similar to the Mamdani FR singleton output fuzzy set.

2.3.3 Fuzzy inference

Fuzzy inference is the reasoning process employed to determine the system fuzzy output. It is based on the processing of fuzzy rules taking into account the fuzzified input parameters. In order to perform this task, it is necessary to obtain the activation level for each rule, which is followed by an aggregation process of outputs of fuzzy rules. This process can be summarized as follows (inspired in [21]):

- Consider the following notation:

$$\begin{aligned}
F_1^j &= \{(u_1, \mu_{F_1^j}(u_1)) : u_1 \in U_1\} \\
F_2^k &= \{(u_2, \mu_{F_2^k}(u_2)) : u_2 \in U_2\} \\
&\cdots\cdots \\
F_n^l &= \{(u_n, \mu_{F_n^l}(u_n)) : u_n \in U_n\} \\
G_1^p &= \{(v_1, \mu_{G_1^p}(v_1)) : v_1 \in V_1\} \\
G_2^p &= \{(v_2, \mu_{G_2^p}(v_1)) : v_2 \in V_2\} \\
&\cdots\cdots \\
G_q^p &= \{(v_q, \mu_{G_q^p}(v_q)) : v_q \in V_q\}
\end{aligned}$$

  where $F_i^{j,k,l}$, $i = 1, .., n$ represents input fuzzy variables and j, k, l are indexes related to the number of linguistic values (i.e., fuzzy sets) corresponding to each linguistic

variable. G_h^p, $h = 1, .., q$ represents output fuzzy variables and p is an index related to the overall number of rules.

According to this notation, we can represent the fuzzy rule system as follows:

$$\begin{aligned} Rule_1: &\ IF\ u_1\ is\ F_1^1\ and\ u_2\ is\ F_2^1\ and\ ....\ u_n\ is\ F_n^1\ THEN\ v_1\ is\ G_1^1,\ v_2\ is\ G_2^1, ..., v_q\ is\ G_q^1 \\ Rule_2: &\ IF\ u_1\ is\ F_1^2\ and\ u_2\ is\ F_2^1\ and\ ....\ u_n\ is\ F_n^1\ THEN\ v_1\ is\ G_1^2,\ v_2\ is\ G_2^2, ..., v_q\ is\ G_q^2 \\ &\qquad\qquad\qquad\qquad ..... \\ &\qquad\qquad\qquad\qquad ..... \end{aligned}$$

As can be observed from above notation, the overall number of fuzzy rules depends on the number of linguistic variables n and their corresponding number of linguistic values.

- Combine input values with rule antecedents. Recall that we will only consider singleton fuzzification in this thesis. Hence, this step is simplified by only performing the evaluation of each degree of membership, which can be expressed as follows:

$$\begin{gathered} \mu_{F_1^{j=1}}(u_1), \mu_{F_1^{j=2}}(u_1), \mu_{F_1^{j=3}}(u_1), ... \\ \mu_{F_2^{k=1}}(u_2), \mu_{F_2^{k=2}}(u_2), \mu_{F_2^{k=3}}(u_2), ... \\ ..... \\ ..... \\ \mu_{F_n^{l=1}}(u_n), \mu_{F_n^{l=2}}(u_n), \mu_{F_n^{l=3}}(u_n), ... \end{gathered}$$

- Compute the activation level of each rule. In this case, it is necessary to evaluate the expression of each rule. The class of fuzzy operators used for this purpose is called *t-norm* operator [2]. In this thesis, we will only consider conjunction connectives with the *AND* operator. The two most frequently used *t-norm* operators are:

$$\begin{aligned} Rule_i &: \ \mu_i(u_j) = \Pi_{j=1}^k \mu_{F_j^i}(u_j) \\ Rule_i &: \ \mu_i(u_j) = min(\mu_{F_j^i}(u_j)),\ for\ j = 1...k \end{aligned} \tag{2.8}$$

the former expression is an algebraic product, whereas the latter shows the *min* operator that Mamdani used in his first fuzzy control [28]. k indicates the number of terms of each rule antecedent. In this thesis, we will employ the first expression in

[2] Triangular norm (t-norm) and triangular conorm (t-conorm) are used to model the logical connectives: conjunction (AND) and disjunction (OR). In the case of the *OR* operator, we have two common implementation methods: *max* operator and the *algebraic sum* method (see [23])

2.8 in order to obtain the activation level of each rule, because it is pointed out by some authors that it can represent more precisely the results of a conjunctive association due to the product operation of all membership values instead of not taking into consideration intermediate values as it occurs with the *min* operator.

- Perform the implication procedure. In this case, we need to combine the strength of each rule and the corresponding consequent. The consequent is adjusted by using the same *t-norm* operation described above whose result is an implied fuzzy set $\widehat{G}_h^i$ expressed by a membership function as follows:

$$Rule_i: \ \widehat{G}_h^i(v_h) = \mu_{\widehat{G}_h^i}(v_h) = \mu_i(u_j) * \mu_{G_h^i}(v_h) \tag{2.9}$$

  where $u_j, j = 1, 2, ...,$ represents the set of input variables in the antecedent part; $v_h, h = 1, 2, ...,$ represents the set of output variables in the consequent part; G_h^i represents the output fuzzy set related to the h output variable in rule number i; and $\mu_i(u_j)$ is the activation level of rule i obtained in the previous step.

- Aggregate all output implied fuzzy sets. All of the implied fuzzy sets obtained in the previous step (one for each rule) are combined together to form a single fuzzy set for each output variable. This aggregation process is performed typically using *t-conorm* operations [23]. In this thesis, we will employ the *max* operation for this *t-conorm* operation. Thus, we can summarize this process as follows:

$$\widehat{G}_h(v_h) = \biguplus_{i=1}^{t} \widehat{G}_h^i(v_h) \tag{2.10}$$

  where t is the total number of rules. Using the *max* composition, the $\biguplus$ operator means that the aggregated output fuzzy set is obtained by taking the maximum value on a point basis over all $\widehat{G}_h^i(v_h)$, $for\, i = 1, .., t$.

2.3.4 Defuzzification

Defuzzification is the last step in a FLS. Actually, it is an optional procedure, being applied when it is necessary to obtain a crisp value from the aggregated implied output fuzzy set of the previous step. Indeed, when linguistic results expressed by the aggregated output fuzzy sets are enough, there is no reason for any defuzzification. Several applications in the real world need this conversion in order to use a crisp value as an input for a system because they can not deal with imprecision. For example, this is the classical case of

fuzzy controllers [20, 21, 29, 30]. In such systems, a crisp value that represents one control information (e.g., potential difference and electrical current values, movement angle values, etc..) is employed to control some device or system.

Many defuzzifiers have been proposed in the literature, but there are no scientific bases for any of them. Consequently, defuzzification is an art rather than a science [26]. We can specify this procedure as follows:

$$v_h^{crisp} = DefuzzificationMethod(\widehat{G}_h(v_h)) \tag{2.11}$$

where v_h^{crisp} is the crisp value obtained from the defuzzification of the aggregated implied output fuzzy set $\widehat{G}_h(v_h)$ according to a selected method. In engineering applications, one criterion for the choice of a defuzzifier is computational simplicity. This criterion has led to the following candidates for defuzzifiers [26]:

1. Maximum Defuzzifier
2. Mean of Maxima Defuzzifier
3. Centroid Defuzzifier
4. Center of Area

In [22], there is a study about defuzzification methods (also including those above). The authors propose criteria and classification in order to provide a basis for an evaluation of defuzzification methods. The methods are classified into three categories: *Maxima methods and derivatives*, *Distribution methods and derivatives* and *Area methods*. In the following, we present a summary of the main defuzzification methods [3] based on this work (Refer to [22] for a detailed discussion about this study).

2.3.4.1 Maxima methods and derivatives

- Random choice of maxima (RCOM): A random experiment with probabilities is performed in order to select one element $x \in core(\widehat{G}_h(v_h))$. It is assumed that there is a finite number of core elements and all of them have the same probability of occurring.

- First of maxima and last of maxima (FOM, LOM):

$$\begin{aligned} v_h^{crisp} &= FOM(\widehat{G}_h(v_h)) = max(core(\widehat{G}_h(v_h))) \\ v_h^{crisp} &= LOM(\widehat{G}_h(v_h)) = min(core(\widehat{G}_h(v_h))) \end{aligned} \tag{2.12}$$

[3] Also, it is been considered only the solution for discrete form. For the continuous case, the integral operator should be employed

- Middle of maxima (MOM): Assuming that N represents the total number of elements of $core(\widehat{G}_h(v_h))$, and C_i, $i = 1, ..., N$, is an array of these elements, we can define:

$$\begin{aligned} v_h^{crisp} &= MOM(\widehat{G}_h(v_h)) = C_{i=\frac{N+1}{2}}, \quad for\, an\, odd\, N \\ v_h^{crisp} &= MOM(\widehat{G}_h(v_h)) = C_{i=\frac{N}{2}}\ or\ C_{i=\frac{N}{2}+1}, \quad for\, an\, even\, N \end{aligned} \tag{2.13}$$

  in the case of an even N, the choice of one of the cited forms is an implementation option.

2.3.4.2 Distribution methods and derivatives

- Center of gravity (COG): This is the most common defuzzification method. In general, it computes the center of gravity of the area under the membership function:

$$v_h^{crisp} = COG(\widehat{G}_h(v_h)) = \frac{\sum_{x=v_h^{min}}^{v_h^{max}} \widehat{G}_h(x)\,.\,x}{\sum_{x=v_h^{min}}^{v_h^{max}} \widehat{G}_h(x)} \tag{2.14}$$

- Mean of maxima (MeOM): It is the mean of all elements of the core. Assuming that N represents the total number of elements of the core:

$$v_h^{crisp} = MeOM(\widehat{G}_h(v_h)) = \frac{\sum_{x \in core(\widehat{G}_h(v_h))} x}{N} \tag{2.15}$$

- Basic defuzzification distributions (BADD): In [31], the authors describe a general defuzzification process in which there is a transformation process based on probability distributions for the determination of the defuzzified output value. In fact, this method provides a conversion of fuzzy sets into probability distributions for obtaining the output crisp value. They show that the commonly used methods Center of Area (COA) and Mean of Maxima (MeOM) are only special cases of this more general method. It is expressed as follows:

$$v_h^{crisp} = BADD(\widehat{G}_h(v_h)) = \frac{\sum_{x=v_h^{min}}^{v_h^{max}} x\,.\,(\widehat{G}_h(x))^{\alpha}}{\sum_{x=v_h^{min}}^{v_h^{max}} (\widehat{G}_h(x))^{\alpha}} \tag{2.16}$$

where α is a confidence parameter. For $\alpha = 1$, we have the COA method; for $\alpha = \infty$, we have the MeOM method; and for $\alpha = 0$, we have a simple mean of the $support(\widehat{G}_h(v_h))$.

2.3.4.3 Area method

- Center of area (COA): In the COA approach, the center of gravity of the overall implied output fuzzy set, $\widehat{G}_h(v_h)$, is obtained by an approximation, taking into account the center of gravity, c_i, of the implied output fuzzy set of each rule i, $\widehat{G}_h^i(v_h)$, $for\, i = 1, ..., t$, where t is the total number of fuzzy rules (see expressions 2.9 and 2.10), with mass equivalent to the degree of membership at that point. h, as noted before, represents one output variable:

$$v_h^{crisp} = COA(\widehat{G}_h(v_h)) = \frac{\sum_{i=1}^{t} c_i \,.\, \widehat{G}_h^i(c_i)}{\sum_{i=1}^{t} \widehat{G}_h^i(c_i)} \tag{2.17}$$

2.4 Example: Building a FLS for the perceived quality of speech

In order to illustrate the use of Fuzzy Logic Systems and the main concepts discussed so far, we present an example in this section. For clarity, we show a simple scenario only considering two input variables and one output variable. Figure 2.6 depicts a FLS for a model of the perceived quality of speech with two input linguistic variables: *end-to-end one-way delay*, d, and *interactivity level*, α. The former was proposed according to information available in the ITU-T recommendation G.114 [4], while the latter we are hypothetically defining, as well as the output variable q.

According to our proposed scenario, the FLS has to output a crisp value of the perceived quality of speech, q_{crisp}, based on two input crisp values, d_{crisp} and α_{crisp}. In this context, besides the delay value, we are supposing that it is necessary to take into account the interactivity level of an application in order to classify the quality of speech in communication networks. Thus, both crisp values will be combined through fuzzy inference in order to output a supposed grade for quality of speech on a scale from 0 to 8. All variables have continuous spaces. All point values, grade scales and shapes of membership functions were

[4] as presented in section 2.2.1

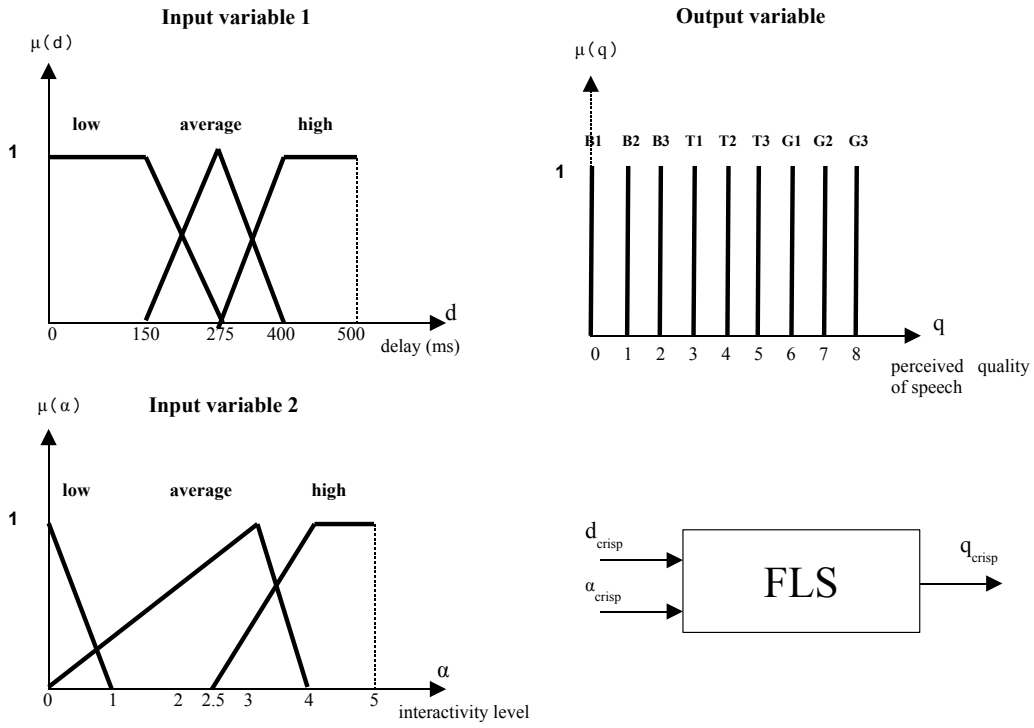

Figure 2.6: Example of a FLS for the perceived quality of speech

arbitrarily chosen except, as mentioned before, for the case of the variable d. It is interesting to note the shapes of the output membership functions; they were modeled using singletons. There are three groups of linguistic values for the linguistic variable q: *Bad*, *Tolerable* and *Good*, in which internally they are divided into three grades. For example, for *Bad* we have three linguistic values: $\{B1, B2, B3\}$. For q, $B1$ denotes the worst value, whereas $G3$ is the best in terms of quality of speech. For α, 0 denotes the lowest value, whereas 5 is the highest in terms of interactivity. Table 2.2 presents a summary of the FLS in Figure 2.6.

For the inference system, the fuzzy rule set will have nine rules due to the number of linguistic terms and variables. The fuzzy rules are of Mamdani type considering the consequent part as singletons. However, we can also have a Zero Order Takagi-Sugeno type since they have constant values in the consequent part.

Table 2.3 summarizes the 9 rules. The singleton values were assigned to each rule in a subjective way by inferring a probable behavior of the system. For instance, lower delay values provide the best results (i.e., *good* group) but the perceived quality may be worse

Table 2.2: Summary of the FLS in Figure 2.6

Variable	Type	Linguistic Terms	Universe of Discourse	Description
d	input linguistic variable	$\{low, average, high\}$	$[0, 500]$	end-to-end one-way delay, according to the ITU-T recommendation G.114
α	input linguistic variable	$\{low, average, high\}$	$[0, 5]$	grade of a network application in terms of interactivity level
q	output linguistic variable	$\{B1, B2, B3, T1, T2, T3, G1, G2, G3\}$	$[0, 8]$	grade given by a user in terms of perceived quality of speech

Table 2.3: Fuzzy Rules of the FLS in Figure 2.6

Linguistic Terms	low d	average d	high d
low α	G3	T3	B3
average α	G2	T2	B2
high α	G1	T1	B1

for higher interactivity levels. For this reason, a $(low\ d, low\ \alpha)$ pair has a better grade (i.e., G3) than a $(low\ d, high\ \alpha)$ pair (i.e, G1).

A rule in the table is read according to expression 2.6, which was already defined. For example, the first three rules are written as follows:

$$Rule_1: \ IF\ d\ is\ low\ and\ \alpha\ is\ low\ \ THEN\ \ q\ is\ G3$$
$$Rule_2: \ IF\ d\ is\ average\ and\ \alpha\ is\ low\ \ THEN\ \ q\ is\ T3$$
$$Rule_3: \ IF\ d\ is\ high\ and\ \alpha\ is\ low\ \ THEN\ \ q\ is\ B3$$

Considering the COA defuzzification method defined in expression 2.17, Figure 2.7 illustrates the behavior of the perceived quality of speech according to the variation of the end-to-end one-way delay and the interactivity level, i.e., $q_{crisp} = FLS(d_{crisp}, \alpha_{crisp})$. As expected, higher interactivity levels and delay values have lower quality levels. On the contrary, for lower interactivity levels and delay values we have higher quality grades. Also, according to the proposed modeling of the input membership functions, one can observe that the end-to-end one-way delay has a higher influence on the perceived quality of speech than the interactivity level. It is important to mention that the design of membership functions

is a crucial issue for a FLS. They must represent the actual behavior of the system as accurate as possible, capturing any non-linearity. As we will see in chapters 4 and 5, we will employ optimization techniques and simulation in order to build such models for the proposed method of this thesis.

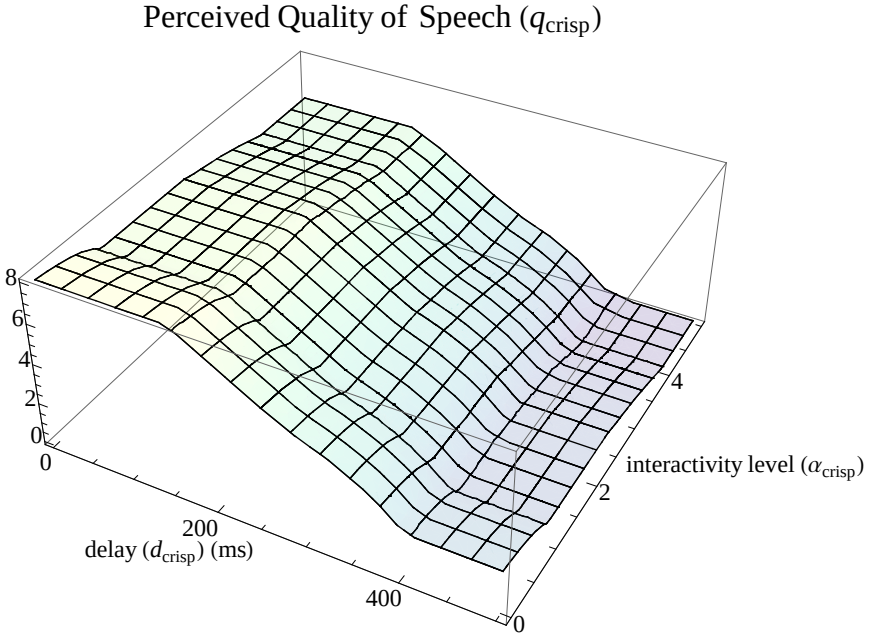

Figure 2.7: Behavior of the perceived quality of speech: $q_{crisp} = FLS(d_{crisp}, \alpha_{crisp})$

2.5 Conclusion

In this chapter, we presented the main concepts related to fuzzy systems. We showed they are based on operations that involve linguistic variables, linguistic values and sets. Fuzzy sets can be viewed as functions (i.e., membership functions) that express a "conforming level" of one object into a specific pre-defined group. This "conforming level" permits the specification of intermediate values within interval $[0, 1]$ instead of single values of membership (0 or 1) as in the case of the conventional set theory (i.e., crisp values). We showed that Fuzzy Logic Systems perform three main steps based on these fundamental elements in order to output a result: Fuzzification, Inference and Defuzzification. Fuzzification converts one crisp value into one fuzzy value. The inference process combines input fuzzy values through fuzzy rules sets in order to output a fuzzy value as a result. As could be seen,

Mamdani and Takagi-Sugeno types are the most usual fuzzy rules types, which differ with respect to the consequent part of rules. Depending on the system that is being considered, it can be necessary to convert this fuzzy value into a crisp one. This is accomplished with the defuzzification process. Several defuzzification methods were presented, however the two most common are the COG (Center of Gravity) and COA (Center of Area) methods.

We also showed that defining membership functions is a challenging task which involves the specification of their shapes, range values and number of fuzzy sets. An inaccurate definition may lead to wrong output results. In this case, such fuzzy model could not represent the real behavior of a system. In Chapter 5, we will employ optimization techniques along with simulations in order to perform this task. Finally, we presented an example of a model based on Fuzzy Logic Systems with two input variables and one output variable to illustrate an application scenario where we could see that it is a good approximation for non-linear mapping between variables.

Chapter 3

Traffic Models and Simulation Environment

3.1 Introduction

As mentioned previously, the proposed method in this research work combines non-linear programming and simulation to build a fuzzy logic based model of a network node. Simulations are used to provide input data for the optimization problem. Simulation plays a key role in this method, because it is assumed that it provides a good estimation of the actual network node behavior under the modeled traffic conditions. The simulation model includes a few network node mechanisms (e.g., leaky bucket metering and queue scheduling, and the model of the traffic offered to the network node). As these mechanisms have well known deterministic algorithms, traffic modeling becomes a critical issue [32]. As we will see in Chapter 5, the considered network node handles individual flows (e.g., VoIP traffic) and general data traffic handled by priority scheduling mechanisms with several queues.

Mathematical models and the related optimization theory is a fundamental part of several experimental techniques. Depending on the system under investigation, it is a hard task to propose an analytical model that represents correctly such system. Using simulations, researchers look for knowing the behavior of a specific system and try to validate those models. The unknown parameter values of such system can be identified by solving an optimization problem in order to find the optimal values for use in this context. In this thesis, we employ *optimization techniques* along with *simulations* in order to obtain the optimal parameters of a fuzzy logic system for modeling queue behavior.

In the traditional network traffic modeling, packet and connection arrivals were often assumed to be Poisson processes because such processes are mathematically tractable. Moreover, due to scalability reasons, most of the developed simulation modeling is target to aggregate traffic. However, in the current business networking scenario, where service providers intend to provide a larger share of resources, there is an increase in dependency on understanding whether network behavior is meeting end-user Quality of Service (QoS) goals, driving the need to detect, monitor and control individual application flows as they traverse the network. The major challenge is that the provision of services that complies with the QoS constraints while maximizing network resources. Though we can view QoS at the application or packet level, we focus on the network node performance, and QoS is defined in terms of queuing delay and packet loss probability.

In the case of VoIP traffic, it is commonly accepted that it can be adequately modeled as a two state Markov source [33], but the simulation of individual application flows poses specific issues. Also, several studies have shown that for local area (LAN) and wide area network data traffic (WAN), the distribution of packet inter-arrivals clearly differs from exponential [34, 35]. These studies convincingly argue that LAN and WAN data traffic is much better modeled using statistically self-similar processes [36].

In this Chapter, we present the traffic models and related issues with respect to synthetic traffic generation and performance measurement, which are the basis for the problem formulation in Chapter 5. Our simulation approach for self-similar traffic and per-flow modeling (VoIP traffic) is also discussed. Then, we describe the simulation environment developed for the method evaluation.

3.2 Self-Similar Traffic Modeling

Presently, the self-similar approach is widely used to characterize many different types of aggregated traffic. Self-similar traffic is modeled as a long range dependent process (LRD), as it can present longer idle and burst periods than traditional traffic models, such as those based upon Poisson arrivals. For example, sources of traffic modeled as Poisson arrivals presents a short range dependency (SRD) property, and tend to be smoothed when observed over large time scales. If we calculate the average of the occupation rate of a link subjected to a Poisson traffic using time steps, the result tends to be more homogeneous as the size of the time steps increases. As pointed by [36], this smoothing property is not so intense in real traffic. Assuming a SRD behavior for a traffic that presents self-similar properties can

considerably reduce the accuracy of the queue models as the longer burst periods can lead to increased delay and drop levels.

A LRD process happens when samples of an event observed in distinct time instants are strongly correlated. In this case, its autocovariance function, say $r(k)$, decays very slowly along the time in terms of a lag k which makes impossible to calculate its sum. Hence, according to [37], self-similar processes have LRD which can be defined as follows:

$$\sum_{k=0}^{\infty} r(k) = \infty \tag{3.1}$$

also, a process has LRD if there is a constant $C_r > 0$ such as its autocovariance function can be expressed as the following equation:

$$r(k) \sim C_r \,.\, k^{-\alpha}, \quad \alpha \in (0,1) \tag{3.2}$$

where, α is related to the Hurst parameter H, $\alpha = 2-2H$. LRD processes have $\frac{1}{2} < H < 1$, whereas processes have the SRD property for $0 < H < \frac{1}{2}$.

In this research, the data traffic follows the self-similar model presented by Norros [38], the fractional Brownian motion (fBm). It is important to note that the proposed method presented in Chapter 5 is generic and is not dependent on the assumed traffic model. In fact, we can find other models for self-similar traffic for validating our method, but we have used such model because of previous works developed so far. It is possible to include, in a future work, other self-similar traffic models.

According to this model, the accumulated aggregate arrival process is represented as follows:

$$A_t = m \,.\, t + \sqrt{a\, m} \,.\, Z_t \tag{3.3}$$

where Z_t is a normalized fractional Brownian motion (fBm), m is the mean input rate, a is a variance coefficient and H is the Hurst parameter of Z_t. The parameters a and H control the level of burstiness of the traffic. The higher these parameters, the worst-behaved the traffic is. The a parameter controls the magnitude of fluctuations around the mean rate. The H defines how sustained the bursts and idle periods can be. Usually the a and H are estimated based upon the observation of real traffic. In this thesis we have used synthetic traffic, for which the variable Z_t was implemented by using the Hosking method [39].

FBM has an incremental process known as fractional Gaussian noise (FGN), which exhibits LRD with Hurst parameter H. In order to validate our implementation for generating

FGN series to compute Z_t, consider the two FGN samples for a 10 second simulation and $H = 0.85$ depicted in Figure 3.1, where each point t corresponds to a $10ms$ interval.

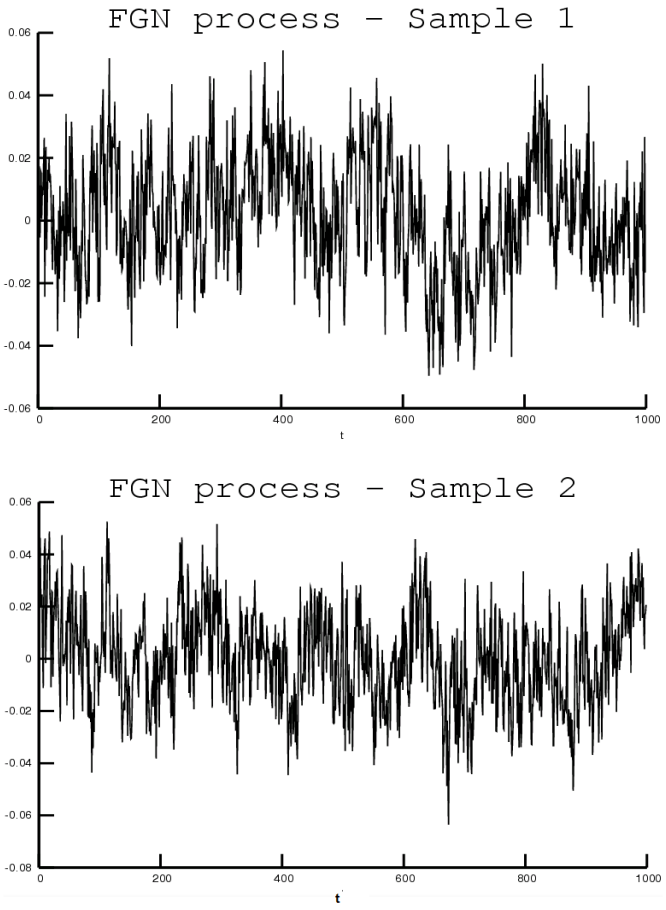

Figure 3.1: FGN process samples

The first property to be investigated is the slow decay of the autocovariance function. In this case, we can compute the autocorrelation function [40] of the FGN series. Thus, let $X = \{X_t,\, t = 1, 2, ..., N\}$ be the FGN process with mean $\overline{X}$, the autocorrelation function can be expressed as follows:

$$r_k = \frac{\sum_{i=1}^{N-k}(X_i - \overline{X}).(X_{i+k} - \overline{X})}{\sum_{i=1}^{N}(X_i - \overline{X})^2} \tag{3.4}$$

where, k is the lag, for $k = 0, 1, 2, .., N-1$. Figure 3.2 shows the plot of r_k for five samples including those illustrated in Figure 3.1. As can be observed in this Figure, we clearly see the slow decay of the statistic r_k for all samples which indicates that the FGN series present the LRD property.

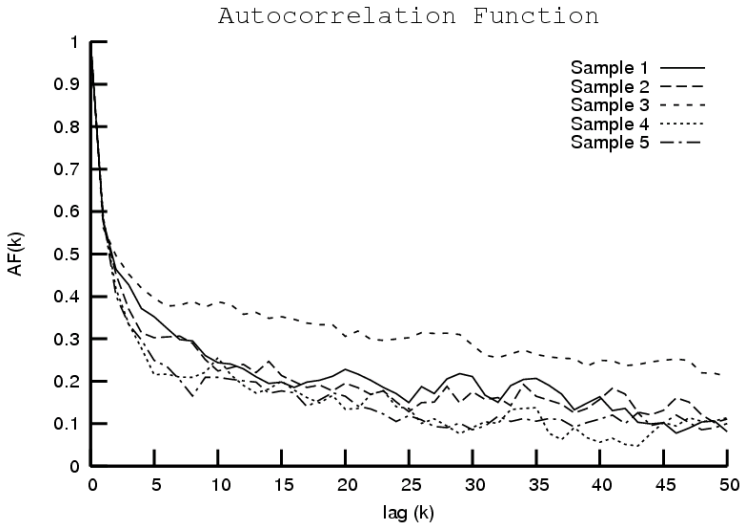

Figure 3.2: Autocorrelation function

Another common method employed to identify self-similarity is the R/S statistic [41]. It is a well-known technique for estimating the Hurst parameter. In this case, considering that our synthetic traffic was generated with $H = 0.85$, it is expected that we can estimate such H value by applying the R/S method on the FGN series. Again, let $X = \{X_t,\ t = 1, 2, ..., N\}$ be the FGN process. Given a time span n, the average $\overline{X}_n$ over n is expressed as follows:

$$\overline{X}_n = \frac{\sum_{t=1}^{n} X_t}{n} \tag{3.5}$$

The accumulated process, called A_t^n, of X_t over time span n is defined as:

$$A_t^n = \sum_{t=1}^{n} (X_t - \overline{X}_n) \tag{3.6}$$

The range of the accumulated process, denoted by R_n, is defined as follows:

$$R_n = max(A_t^n) - min(A_t^n) \tag{3.7}$$

where $0 \leq t \leq n$. The S_n statistic represents the standard deviation:

$$S_n = \sqrt{\frac{\sum_{t=1}^{n}(X_t - \overline{X}_n)^2}{n}} \tag{3.8}$$

According to this method, the paramater H can be estimated by plotting $log_{10}(n)$ and $log_{10}(R/S)$ for several values of n. Using data fitting methods, for example, the least squares, we can obtain the estimation of H which is the slope of the resulting straight line. Figure 3.3 shows the R/S statistic for all five samples. Each point in the Figure represents one R/S value for one of the five samples. Considering all points, we apply the least squares method using a equation with format $a.log_{10}(n) + b$, which gives us a straight line whose slope is $a = 0.84486$, as indicated in the Figure. This value confirms the $H = 0.85$ used in the FGN series generation.

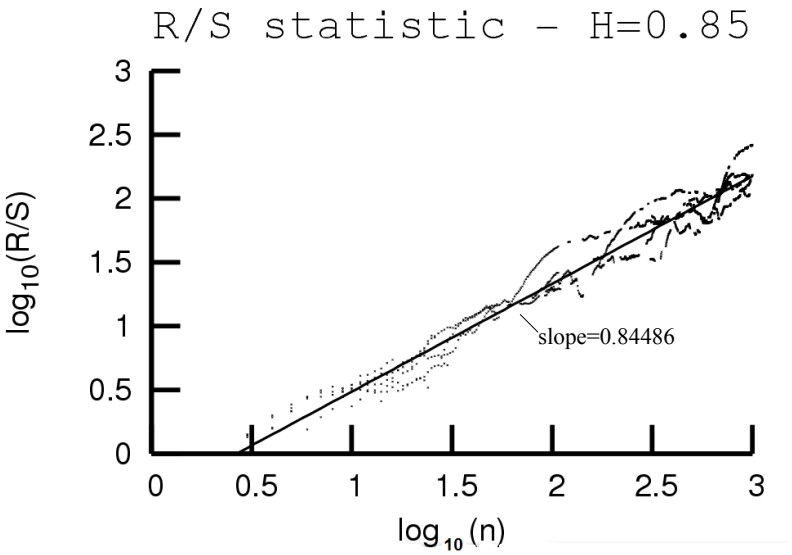

Figure 3.3: R/S statistic

The Aggregated Variance Method [42] can also be employed to estimate the parameter H. According to this method, the FGN series must be divided into blocks of length n. Then, we calculate the variance of this sample. For distinct values of n we repeat this process and plot the variance of each aggregated series versus the value of n. Using again a data fitting method, the slope of the straight line can be used to obtain H. Thus, let $X = \{X_t,\ t = 1, 2, ..., N\}$ be the FGN process. The average of X over each block is expressed

as follows:

$$\overline{X}_k^n = \frac{\sum_{t=(k-1).n+1}^{k.n} X_t}{n} \tag{3.9}$$

where, $k = 1, 2, ..., \frac{N}{n}$. The sample variance is obtained as follows:

$$\sigma_n^2 = \frac{\sum_{k=1}^{N/n} (\overline{X}_k^n - \overline{X})^2}{N/n} \tag{3.10}$$

Figure 3.4 depicts the variance time plot for all five samples. Again, the paramater H can be estimated by plotting $log_{10}(n)$ and $log_{10}(\sigma_n^2)$ for several values of n. Using the least squares fitting we can obtain the estimation of H which is given by the expression $slope = (2H - 2)$. Thus, considering that the slope of the straight line of the Figure is -0.329181 as indicated, we have $H = 0.8354095$ which is approximately the original value employed in the synthetic traffic generation.

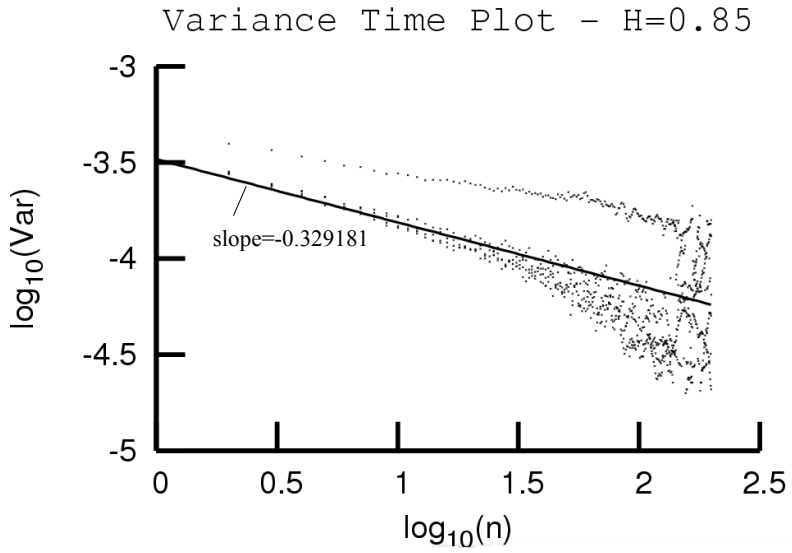

Figure 3.4: Variance Time Plot

In practice, the traffic specification is not supplied in terms of the fBm parameters. Instead, a common approach is to describe the traffic in terms of token-bucket filter parameters, which are employed as traffic conditioners at the edge of a DiffServ domain. A token-bucket mechanism is usually described in terms of a bucket rate (that determines the

sustained average of the transmission rate), a bucket size (that determines the maximum burst size in bytes) and, in some cases, a peak rate (that determines the maximum rate at which a burst can be sent). As the peak rate is usually the bandwidth of the link, it is commonly omitted. The token-bucket model allows for representation of a large variety of source types. Larger bucket sizes correspond to bad-behaved traffic, such as compressed video or data. Small bucket sizes define a traffic that approaches a CBR (constant bit rate) behavior.

3.3 VoIP Traffic Model

Traffic models typically apply only to the performance of the aggregated traffic. However, some authors have pointed out that under certain conditions the performance experienced by individual flows can significantly diverge with respect to the performance of the aggregated traffic [9, 10]. Consequently, even when the aggregated traffic is properly dimensioned, the individual flows may not satisfy QoS requirements such as delay and packet loss. In fact, it has been shown that the effect of considering flows with distinct life cycles introduces significant deviations between the individual flows percentile performance.

An individual VoIP flow corresponds to a well-behaved traffic, which could be characterized by packets of fixed size transmitted at a constant rate. The transmission rate depends on the codec used. If VAD (Voice Activity Detection or voice suppression) is used, VoIP packets are generated only when voice is detected. VoIP with VAD is usually modeled as an ON-OFF source. Many VoIP traffic modeling works assume that both ON-OFF intervals are exponentially distributed [33].

While modeling VoIP as an aggregate of ON-OFF sources is widely studied subject, the per-flow problem is not extensively addressed in the literature. Most attempts of deriving a mathematical model for the per-flow behavior assume that the life-cycle of all VoIP flows is identical, i.e., all flows start in the beginning of the evaluation and terminates simultaneously. In real world, however, the life cycle of VoIP flows is variable. This variability can be modeled by assuming two additional parameters: TBF (time between flows) and AFD (average flow duration) [43, 33]. The AFD parameter allows us to capture the impact of the load variation with respect to the performance level of a percentile of packets within each VoIP flow on a long-term basis, providing a better representation of the real scenario. Again, both intervals, AFD and TBF, are usually assumed to be exponentially distributed. Figure 3.5 illustrates the VoIP modeling.

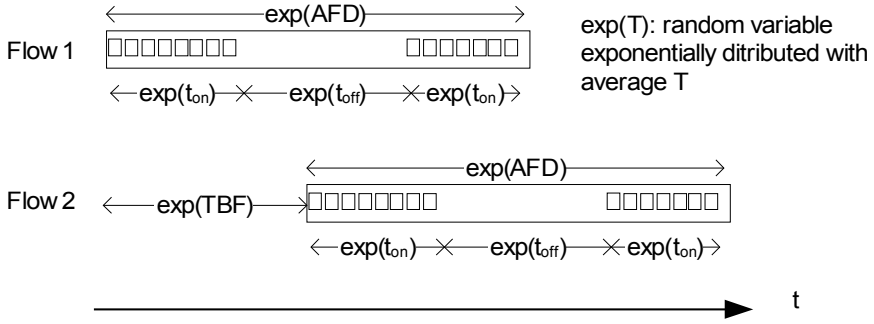

Figure 3.5: VoIP Traffic Modeling

Because the exponentially distributed parameters define variable traffic load conditions, we have assumed some default values for the average of these parameters. As suggested in [43], the AFD value is typically assumed to be between $[180\,s, 210\,s]$ in most business environments. We have assumed the value of $210\,s$ for the AFD parameter. The exponential averages were defined as $0.4\,s$ for the t_{on} period and $0.6\,s$ for the t_{off} period. Again, it is iimportant to mention that the proposed method in this research work is not dependent on the assumed traffic modeling approach for its validation. Our proposal is, given any traffic model which can be simulated, our method allows us to build a (fuzzy) mathematical model in terms of performance metrics. The assumed VoIP individual flow model employs a strategy based on two exponential distributed variables that regulates life cycle of flows, which permits a representation of more realistic scenarios. Also, it is possible to include, in a future work, other traffic models.

Figure 3.6 shows the results of a simulation that illustrates the effects of the combination of VoIP and self-similar traffic. In this scenario, the network node has two queues, one for the aggregated VoIP traffic (EF queue) and the other for self-similar traffic (AF queue) using a priority queuing scheduler. It puts in evidence how the individual flows performance can diverge with respect to the aggregate performance. In this simulation, we have a variable offered load of VoIP calls which is limited by an AC mechanism in order to keep a maximum of 60 simultaneously active VoIP flows. This number of flows was chosen intentionally high in order to illustrate how difficult it is to predict the behavior of individual flows by taken into account only the aggregated traffic behavior. We have limited the VoIP (EF) queue size to $50\,ms$, i.e., packets that exceed the $50\,ms$ delay are dropped. The AF queue size was limited to $250\,ms$. The simulation presented in this Figure has considered a link capacity of only 2 Mbps in order to avoid an excessive number of VoIP flows in Figure.

The m parameter in equation 3.3 was adjusted to represent 50% of the link capacity, $a = 275\,kbit.s$ and $H = 0.76$ as suggested in [38].

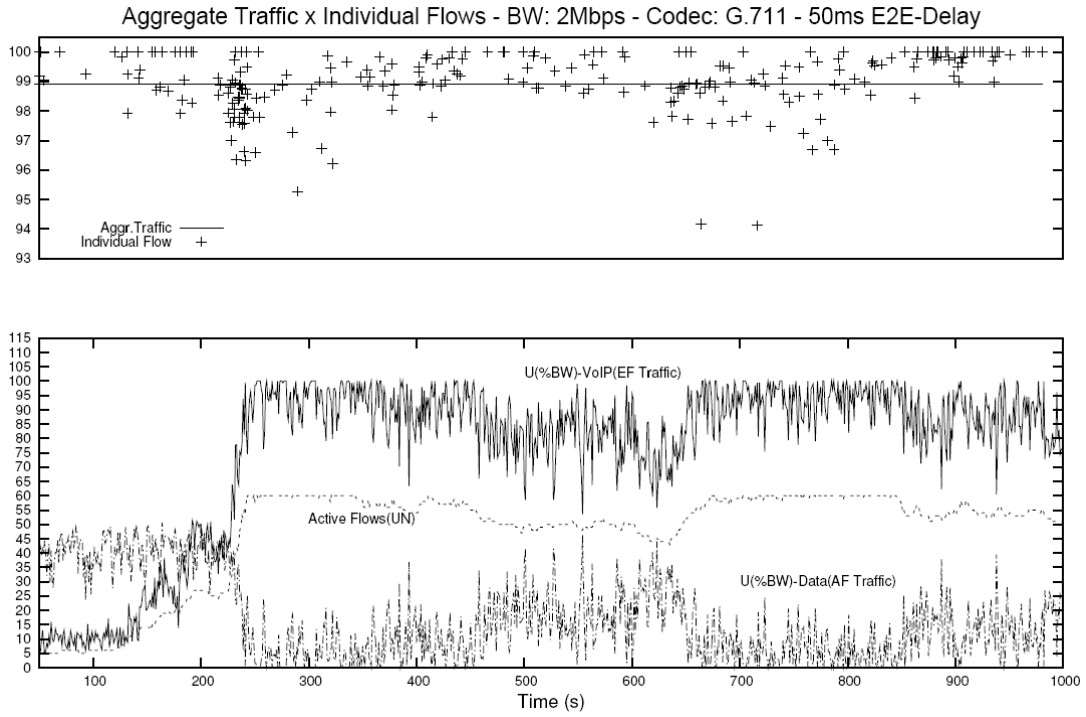

Figure 3.6: Per-flow variability illustration

The straight line in the superior part of figure indicates that, approximately, 99% of the aggregated VoIP packets have satisfied the delay bounds (i.e., they were not dropped). However, the *per-flow* quantile of packets that satisfied the delay bounds has significantly diverged with respect to the aggregate performance (each cross in the figure represents the performance of an individual flow plotted at the instant when the flow terminates). The inferior part of the figure shows the difference between the VoIP and self-similar traffic profiles. Initially, the number of active VoIP flows is less than 60 permitting a higher occupation rate of the AF traffic. However, due to the strictly priority queuing, when the number of active VoIP flows reaches 60 the occupation rate of the AF traffic diminishes. As can be noted, when the offered load in terms of VoIP flows is lower than 60 (e.g., observe the region between 500s and 600s) there is higher occupation rate of the AF traffic. Also, it is possible to see a better performance experienced by VoIP flows which were active during this period (see the corresponding superior part of the Figure). It is important to note that VoIP is not CBR even when the number of active flows is constant, due to the silence

suppression effect. In this case, providing per-flow guarantees poses additional difficulties because it is also necessary to take into account the fluctuations of the flows' lifecycles in order to determine the impact of the queue behavior on the performance of the individual flows. The proposed method in this thesis permits to create performance models taking into account QoS metrics established on a per-flow basis.

3.4 Simulation Environment

In order to carry out the simulations required for the evaluation of the proposed method, we developed a new simulation environment. Initially, simulations were performed using the NS-2 [44]. However, the NS-2 was not scalable enough for supporting the most complex simulation scenarios (usually, hundreds of simulations are required before the FP convergence). The reason is that NS-2 implements the whole IP protocol stack and includes several events and objects related to each layer of the stack, causing the utilization of many non necessary resources. In fact, NS-2 offers several resource consuming features which are not required for determining the queue behavior. For example, the use of the NS-2 is advantageous when the traffic is modeled as a bunch of TCP connections, as it offers an accurate modeling of the TCP state machine. In our approach, however, TCP connections are not individually modeled, as the self-similar model already captures the effect of aggregating TCP connections under variable congestion situations.

NS-2 and other network simulators use a discrete-event processor as their engine. Researchers have adopted several complementary approaches to improve accuracy, performance, or scaling. Some simulators augment event processing with analytic models of traffic flow or queuing behavior for better performance or accuracy [45, 46]. In our method, high simulation performance is one of the critical requirements, besides modeling flexibility and adequate programming support. Therefore, a simulation environment was developed using the Simpatica library [47] and the C language. Simpatica is a discrete-event oriented library based on the actor-message paradigm [48].

An actor is a self-contained active object that has its own control thread and communicates with other actors through asynchronous message passing [49]. In addition, an actor can create other actors, just as an object can create other objects. Every entity of our simulation implementation, such as the mechanisms of the network node, each VoIP flow traffic source, and the self-similar traffic source; is implemented as an actor. The network packets, by the other hand, are implemented as messages exchanged between actors. It

is important to mention that we have extensively tested the Simpatica library in order to assess its accuracy level, by performing many scenarios that have been compared against the NS-2 simulator, leading to identical results.

The Simpatica library maintains a list of simulation events sorted by schedule time. The environment also provides a simulation clock and an event scheduler. At a specific time t, the scheduler gets the corresponding events from the list and executes them. The simulation clock advances only when an actor performs a pause, in one of the following alternatives: (i) the actor explicitly requests a pause for a specific interval; or (ii) the actor implicitly waits for a specific message to come. This is an event-driven approach in the sense it increments the simulation clock to the next earliest scheduled event.

The event list data structure is critical with respect to simulation performance. During a simulation, it is the most frequently handled data structure and poor performance here may cause infeasible simulation times. The event list is implemented by a heap data structure, as depicted in Figure 3.7. It is a special case of a binary tree, where each event is stored in one node. Each node can have up to two children, and its event time is smaller than the event time of its children [50]. The heap can be stored in an array, allowing a faster way to obtain one node′s parent and children, when compared with the other approaches, e.g., linked-list data structures. Also, some studies [51, 52] have concluded that for larger sets, the heap data structure provides better results in respect of performance and storage economy.

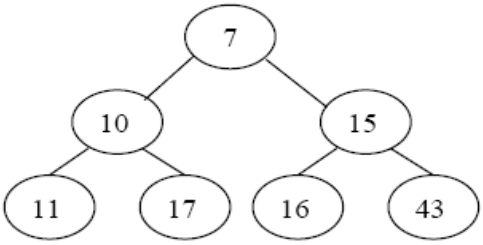

Figure 3.7: Sample heap data structure of the event list

The Simpatica library provides a set of programming primitives, the most significant are presented in Table 3.1. Considering the primitives presented in this Table, we can summarize the following states for each actor thread: RUNNING, SLEEPING, PASSIVATE_SLEEPING and MSG_WAIT. Figure 3.8 presents the state diagram for an actor thread. The transitions are represented by arrows and are caused by a primitive invocation, as indicated by

the corresponding call. Solid arrows with solid lines correspond to primitives invoked by the current actor, whereas solid arrows with dashed lines to primitives invoked by another actor. An open arrow represents a state transition due to a time-out event.

Table 3.1: Simpatica Library API

Primitive	Description
*int task_create(void *taskbody, int stackpages, void *args)*	Creates a new actor thread. The parameter **taskbody* points to the program code of the actor thread. *stackpages* indicates the number of memory pages to be allocated to the thread stack and **args* is an optional parameter used to pass arguments to the actor thread.
void task_exit()	Halts the current actor thread and releases its allocated resources.
void task_sleep(double time)	Pauses the current actor thread during *time* seconds.
void task_passivate()	Pauses the current actor thread until it is woken up by another actor thread.
void task_activate(int actor_id, double time)	Activates the actor thread identified by *actor_id* at the specified simulation *time*.
*void *msg_create(short size)*	Creates a message of size *size*.
*void msg_destroyvoid *msg)*	Destroys the message pointed by **msg*.
*void *msg_recv(double time)*	Causes the current actor thread to wait for a message to come during *time* seconds.
*void msg_send(void *msg, int actor_id)*	Sends the message pointed by **msg* to actor thread identified by *actor_id*.

An actor execution begins in the RUNNING state, after the invocation of *task_create*() by another actor. When running, the actor can invoke *task_create*(), *task_activate*(), *msg_create*(), *msg_destroy*() and *msg_send*() without a state change. On the other hand, the invocation of *task_exit*(), *task_sleep*(), *task_passivate*() and *msg_receive*() causes a state change, and the actor thread can loose execution control. The SLEEPING state is used to model the time spent by an actor in performing some activity. When an actor thread enters *SLEEPING* state it is paused and is rescheduled to the current simulation time plus the value in the primitive argument. The message passing primitives *msg_send*() and *msg_recv*(), support actors communication. They implement a semi-synchronous communication paradigm, where send is unblocking; and the receiving actor remains blocked in MSG_WAIT state until the arrival of a message sent by another actor, or the expiration of the time interval specified in the *msg_recv*() argument. The *task_passivate*() primitive allows a not busy actor to release execution control by making a transition to the PASSIVATE_SLEEPING state. Whenever in this state, the actor shall be woken up by another actor through the *task_activate*() primitive. Finally, the *task_exit*() primitive

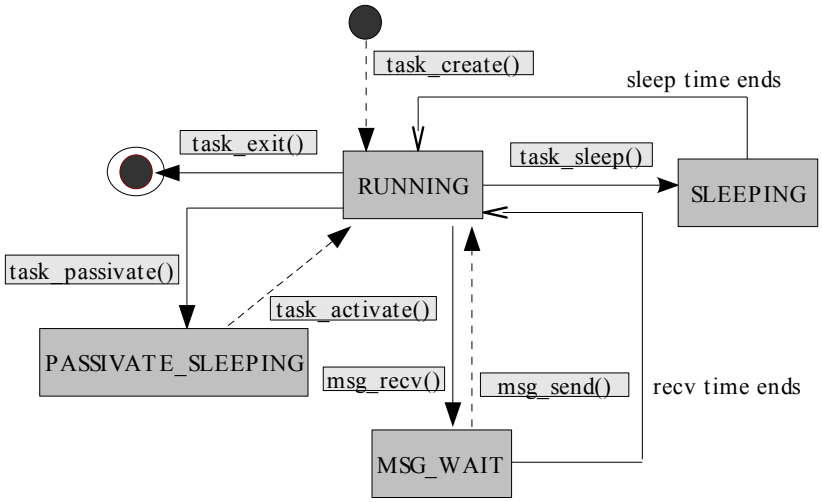

Figure 3.8: State Diagram for an Actor Thread

halts the current actor and releases the allocated resources.

The simulation environment is responsible for simulation clock and thread scheduling. Simulation events are created according to primitive invocations. The scheduler gains execution control whenever a sate change occurs. It then creates the corresponding event, advances the simulation clock, and identifies the thread that is eligible for execution given it the execution control. Figure 3.9 shows the main modules of the simulator: Traffic generators, the network node, and output link. Traffic generation includes the VoIP Flow Controller that instantiates a VoIP Traffic Generator for each individual flow and the Self-Similar Traffic Generator. As seen, the self-similar traffic is typically submitted to a conditioning mechanism, which is implemented by the leaky and token bucket modules. Each VoIP flow is simulated by one VoIP Traffic Generator. The function of the VoIP Flow Controller is to keep constant (i.e., as specified by the optimization problem) the number of VoIP Traffic Generators simultaneously active during the simulation. A unique identification number is assigned to each VoIP Traffic Generator that tags all generated packets, allowing the analysis of the results on a per-flow basis. The data traffic is specified in terms of an average load which is used to compute the accumulated arrival process, as discussed in Section 3.2. It is generated by the Self-Similar Traffic Generator and submitted to a Leaky/Token Bucket that tags the packets conforming to a committed rate as AF and the non-conforming packets as DF. Within the network node, VoIP packets are then inserted

to the EF queue and self-similar packets to AF or DF queues, according to their tags. The PRIO Scheduler implements a priority queuing mechanism with three non-preemptive finite size queues: EF, AF and DF. In this scheme, a queue is served only when the higher priority queues have no packets waiting to be served.

The network node elements, the VoIP entities and the self-similar traffic source are implemented as actors by using threads, and the packets are implemented as messages exchanged between the actors. For example, consider the case of VoIP flow traffic generation. According to the API shown in Table 3.1, during the initialization phase of the simulator, a *task_create*() call is edited to start the VoIP Flow Controller. Then, it issues several *task_create*() calls in order to instantiate as many VoIP Flow(i) Traffic Generator as required by the considered scenario. Each instantiated VoIP Flow(i) sends VoIP packets to the node entry by using *msg_send*() according to the codec specification and the VoIP Traffic Model discussed in Section 3.3. Appendix A will provide more implementation details of the aforementioned entities.

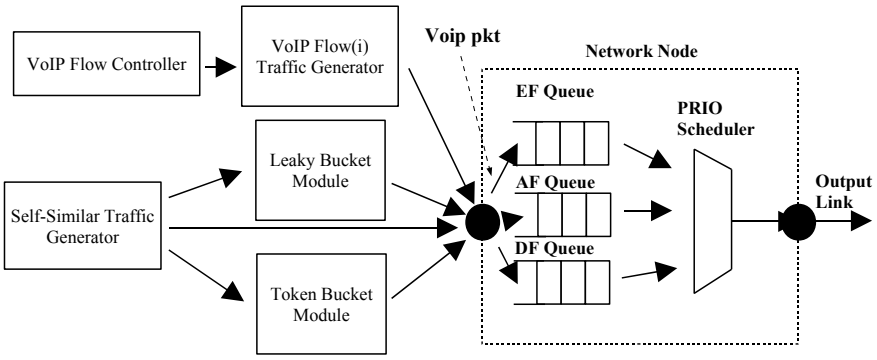

Figure 3.9: Main modules of the Simulator

3.5 Conclusion

In this Chapter, we showed the traffic models that will be employed for synthetic traffic generation in simulations for the proposed method validation. Following the classical approach, a VoIP flow is modeled as an ON-OFF source, but we considered two additional parameters, AFD and TBF, which are based on two exponential variables that regulate the life-cycle of flows, the flow duration and the inter-arrival time of flows, respectively, in order

to have more realistic simulation scenarios. On the other hand, for data traffic simulation we use the self-similar traffic model based on the accumulated aggregated process given by the fractional brownian motion as defined by Norros in his seminal paper. For validating the synthetic self-similar traffic of the simulator we employed several well known tests to verify the fundamental properties of such traffic type. The Autocorrelation function, the R/S statistic and the Variance Time plot confirmed the LRD property for the generated FGN series samples. Also, we presented the simulation environment developed in this research work which is based on the actor messages paradigm and was implemented in C language using the Simpatica API. The main primitives and the state diagram of a thread entity were discussed. Finally, the main modules of the network model for simulation were shown.

Chapter 4

Optimization Method

4.1 Introduction

CHAPTER 2 presented Fuzzy Logic Systems and showed that they can be used to create complex models due to the possibility of capturing the behavior of non-linear uncertain systems. In this case, a FLS is a good choice to perform the modeling proposed by our method, since the uncertainties and unknown parameters are evident, such as the impact in performance metrics when one considers, for example, scenarios with multiple priority queues, non-exponential service times and long term correlated traffic. A FLS is very dependent on the quality of its membership functions. Their definition (i.e., shapes, range values, etc..) must precisely represent the system behavior which sometimes is a difficult task to perform. In this context, several authors [23, 18, 21, 20] have employed *optimization techniques* in order to adjust FLS parameters according to the actual system behavior. For instance, input and output membership functions could have their coordinates defined to better represent the system.

In this Chapter, we first review important concepts and definitions related to optimization theory. Then, we present the optimization method employed in this research work. Finally, we provide an example which shows the design of a parameter based admission control mechanism using a training strategy based on simulations and the optimization method discussed.

4.2 Formulation of an Optimization Problem

Several practical problems involving decision making (or system design, analysis, and operation) can be cast in the form of a mathematical optimization problem, or some variation such as a multicriterion optimization problem [53]. Optimization is used for problems arising in network design and operation, scheduling, and many other areas. This use is motivated by the fact that analytical models sometimes can not formulate the wide range of procedures that optimization techniques can. In this case, Shortest Paths [54], Maximum Flow [55] and Assignment Problem [56] are examples of problems where optimization techniques have been widely adopted.

Optimization problems can be widely viewed as a *cost function* that maps the elements of a constraint set X. For each element of X, named x, there is a cost function $f(x)$ that specifies a value which indicates the undesirability level of choosing such a decision x. The optimal decision, x^* is such that [57]:

$$f(x^*) \leq f(x), \forall\, x \in X \tag{4.1}$$

Based on this strategy, optimization techniques can be useful if there is an objective test involving two different results. As mentioned before, these results are obtained by defining previously a cost function. Depending on the optimizing problem, one can define the criterion for obtaining the optimal values which can be based on minimizing or maximizing the cost function. The former is the most common approach and will be used in this work. Thus, a mathematical optimization problem, or just optimization problem, has the form:

$$minimize\ f(x)\ \ subject\ to\ g_i(x)\ \leq\ b_i,\ i = 1, ..., m \tag{4.2}$$

As cited in [53], x is a vector $x = (x_1, ..., x_n)$ that represents an optimization variable of a problem. The function $f : \Re^n \rightarrow \Re$ is the objective function, the functions $g_i : \Re^n \rightarrow \Re, i = 1, ..., m$, are the (inequality) constraint functions, and the constants $b_1, ..., b_m$ are the limits, or bounds, for the constraints. A vector x^* is called optimal, or a solution of the problem 4.2, if it has the smallest objective value among all vectors that satisfy the constraints: For any z with $g_1(z) \leq b_1, ..., g_m(z) \leq b_m$, we have $f(z) \geq f(x^*)$.

The problem stated in expression 4.2 is of the class known as *constrained optimization*, i.e., the optimal solution x^* is valid only if the constraint functions are matched. Conversely, there are optimization problems which there are no constraints, named *unconstrained optimization*. In this case, the problem 4.2 can be rewritten as [57]:

$$minimize\ f(x)\ \ subject\ to\ x \in \Re^n \tag{4.3}$$

4.2.1 Obtaining Optimal Values

For the optimization problems presented in expressions 4.2 and 4.3 one issue must be investigated: How can one guarantee that the optimal solution x^* is really the unique as stated in condition 4.1?. This issue can be viewed as present, or not, of *local minima*, as seen in Figure 4.1. In this Figure, the points where $x = L$ are examples of *local minima* while the point $x = G$ is one example of a *global minimum*. Using condition 4.1, we can say that a point x^* is a *local minimum* of the function $f(x)$ if there exists some $\in > 0$ such that $f(x^*) \leq f(x)\ \forall\ x$ with $|x - x^*| < \varepsilon$. On the other side, a *global minimum* is a point x^* for which $f(x^*) \leq f(x)\ \forall\ x$. Also, any *global minimum* is a *local minimum*, but a *local minimum* is not necessarily a *global minimum*. In some optimization problems, there is no *local minima*, i.e., every *local minimum* is also global. In this case, a *cost function* that matches this condition has the property of *convexity* [58].

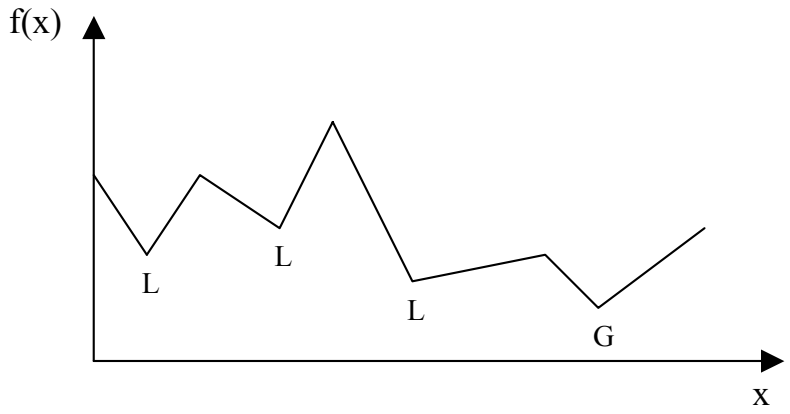

Figure 4.1: Example: Local and Global minima

4.2.2 Classification of Optimization Problems

Strategies employed in the solution of optimization problems are very dependent on the type of functions in expressions 4.2 and 4.3 for constrained and unconstrained optimization problems, respectively. In this case, optimization problems are classified in two classes:

- Linear Programming

- Non-Linear Programming

where, for the first class, all functions (objective and constraints) are linear functions, whereas they belong to the second class in other cases. As we will see in Chapter 5, the optimization problem considered in this thesis is suitable for the Non-Linear Programming class because of the complex modeling proposed and the non-linear behavior of performance metrics when one considers as input several distinct non-correlated parameters.

4.2.3 Non-Linear Programming (NLP)

In the literature, we can find several methods for the solution of Non-Linear optimization problems. Basically, these methods are based on three main concepts [59, 60]:

1. Linearization Methods
   This solution is based on an adaptation of a NLP by using linear techniques. In fact, a starting point is selected, and the non-linear model and constraints are linearized about this point to obtain a linear problem which can be solved by linear optimization methods (e.g., with the Simplex Method [61]). The point from the linear programming solution can be used as a new point to linearize the non-linear problem, and this can be continued until a criterion is met [62]. Such linearization methods present some problems, mainly errors generated by first order approximators when the objective and constraint functions are strongly non-linear, which may compromise convergence properties.

2. Penalization Methods
   In this case, the idea is to convert a NLP problem with constraints into a set of NLP problems without constraints by changing the objective function with the add of penalization terms. In order to illustrate such method, consider the d'Uzawa method [61]. According to it, the original optimization problem P is replaced by a set of dual problems $\{P_d^0, P_d^1, ..., P_d^p\}$ with no constraints. Each P_d^p is formulated with the substitution of the original objective function f by a dual objective function f_d. f_d is composed by the original function f along with terms of the constraints g_i (see expression 4.2) and Lagrange multipliers λ_i. Thus, the p^{th} dual problem P_d^p can be written as follows, considering we want to find the optimal solution x_d^p:

$$x_d^p \in \Re,\ f_d(x_d^p) = inf_{x \in \Re^n} \{f_d(x)\} \tag{4.4}$$

for,

$$f_d(x_d^p) = f(x_d^p) + \sum_{k=0}^{nc-1} \lambda_k^p \,.\, g_k(x_d^p) \tag{4.5}$$

where nc is the number of constraints of the NLP problem. In this expression, Lagrange multipliers are iteratively computed with the gradient method [61] being obtained according to:

$$\lambda_k^p = max\{\lambda_k^{p-1} + \alpha.g(x_d^{(p-1)})\},\ k = 0, ..., nc-1 \tag{4.6}$$

where α is a constant value that is arbitrary chosen. The d'Uzawa method consists of computing iteratively equations 4.6 and 4.4. One difficulty observed in this method is that it requires the solution of a high number of NLP problems without constraints which normally tends to lead to slower procedures, mainly when there exists a high dimension vector as input of the optimization problem.

3. Projection Methods
   These methods are of the gradient class. They convert infeasible solutions generated during the optimization process to a feasible one. Based on the projection concept, given one vector $b \in \Re$ it is necessary to find the vector x_p close to b that belongs to the space of admissible solutions of X. x_p is known as the projection of b, and can be expressed as follows [61]:

$$x_p \in X \, and \, \|x_p - b\| = inf_{x \in X} \|x - b\| \tag{4.7}$$

   In this case, we can define a *projection operator* $P : \Re^n \Rightarrow X$ that determines x_p using b, $x_p = P\{b\}$. Finally, we can express the utilization of the operator as follows:

$$x_{k+1} = P\{x_k - \alpha \,.\, \nabla f(x_k)\},\ k \geq 0 \tag{4.8}$$

   where x_0 is an arbitrary chosen value; $\nabla f(x_k)$ is the gradient of function f at point x_k; and α is a parameter indicating the searching step gain.

In Chapter 5, the formulation of the NLP optimization problem will be stated as an unconstrained problem. However, our method employs the penalization concept discussed above in order to penalize intermediate infeasible solutions, allowing the optimization process to search optimal solutions in the correct search direction.

4.3 The Flexible Polyhedron (FP) Optimization Method

Optimization methods can be broadly classified into *Zero-Order*, *First-Order* and *Second-Order* categories. One advantage of *Zero-Order* methods is that they do not require any derivatives of the objective function. Because of that, they also named $non-derivative$ methods. On the contrary, the last two are based on derivatives and are also classified as *derivative* methods. There are several methods proposed in these categories. The Complex method presented in [63] and the Genetic algorithms in [64] are examples of $non-derivative$ methods. The Newton's Method [57] and the Gauss-Newton algorithm [65] are examples of *derivative* methods. In this thesis, the optimization processes are based on non-derivative methods. Indeed, the chosen method is based on the principle of *flexible polyhedron search* [66]. It is a *direct search method* for solving non-linear programming problems without constraints. The motivation for using this method is twofold: First, it does not need a derivative of the objective functions. Second, it can handle several parameters in a very flexible manner. As will be shown in next Chapter, using such $non-derivative$ method, objective functions can be easily used as results of complex computer simulations. One drawback of using $non-derivative$ methods is that they typically require too much computations due to the necessity of evaluation of the objective function. However, their simplicity and flexibility for reaching the proposed solution of this thesis are far more significant than this problem.

4.3.1 Method Description

As mentioned before, FP is a direct search method of the non-linear programming class, i.e., consecutive evaluations of the objective function are performed in order to obtain the optimal solution. Its main idea is to calculate the search direction using a set formed by the best points obtained until previous iterations. A geometric figure named *polyhedron* is formed by a set of vertexes, where each vertex corresponds to a solution $v \in \Re^n$, and n is the number of variables of the optimization problem. The number of vertexes of the polyhedron is $n+1$. For each iteration a new solution is computed, and the worst vertex is replaced. Considering successive iterations, the polyhedron tends to adjust itself around the optimal solution and reduce its dimension (distance among vertexes) until a convergence criterion is achieved. The FP method can generate infeasible solutions during the search process, raising difficulty for treating constrained optimization problems. This drawback can be solved by making the cost function to strongly penalize infeasible solutions.

Figure 4.2 presents an example of a two variable optimization problem. There is a three dimension polyhedron. The variables are x and y and the vertexes f_1, f_2 and f_3 are the results of the cost function evaluation, $f(x, y)$, for, respectively, $f(x_1, y_1)$, $f(x_2, y_2)$ and $f(x_3, y_3)$. During each algorithm iteration one new solution is generated, when the worst value is discarded. Thus, only the best $f(x_1, y_1)$, $f(x_2, y_2)$ and $f(x_3, y_3)$ are kept in the polyhedron. Considering these successive iterations until matching the convergence criterion, the polyhedron will be in the solution domain of the specific optimization problem given by the objective function $f(x, y)$.

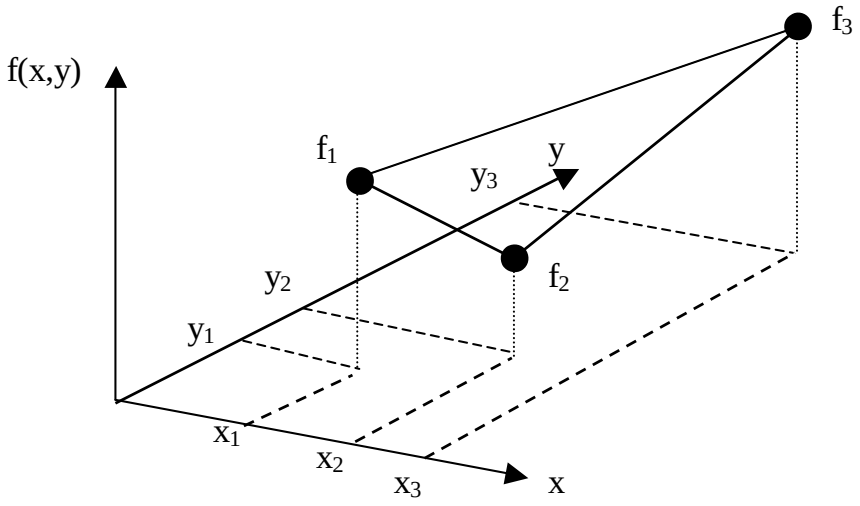

Figure 4.2: Example: A two variable three dimension Polyhedron

There are some variations of the FP algorithm according to the problem being addressed, especially in order to reduce the risk of convergence towards a local minimum. For this reason, we have employed a modified version of the FP method that rebuilds the polyhedron around the optimal solution found in the previous convergence, which will be discussed later.

In order to present the FP algorithm, consider a optimization problem P_v with n variables. The parameter v is a vector $v = \{x_1, x_2, ..., x_n\}$ which contains n values. The solution of this optimization problem is to find the optimal vector v^* where $P_{v^*} \leq P_v \; \forall \; v$. The algorithm uses $n + 1$ vectors v_i for $1 \leq i \leq n + 1$ to determine the search direction. It is important to note that each vector v_i corresponds to one polyhedron vertex. Below, we summarize the FP algorithm.

1. The first step is to obtain the initial polyhedron. If the interval of applicable values

of each variable is known, a straightforward manner to implement this procedure is to initialize the vectors v_i using the combination of the maximum and minimal values of these intervals. Of course, the number of possible combinations is higher than the number of required vertexes. In fact, if there is an optimization problem with n variables it is required $n+1$ vertexes but the number of combinations of maximum and minimal values is 2^n, forcing to discard $2^n - (n+1)$ combinations.

Another strategy for initialization is presented in [59]. In this case, for optimization problems where it is difficult to identify such initialization intervals, [59] suggests the use of coefficients in order to calculate the distance of initial vertexes based on only one estimated vertex. Thus, a parameter that characterizes the initial dimension of the polyhedron is used to calculate the values the others. Inspired by these strategies, we will employ the following approach:

Given the initial solution v_1, initialize the polyhedron vertexes as linearly independent vectors as indicated in 4.9. The coefficients d_i^+ and d_i^- represent, respectively, a large and a small distance with respect to the elements of the initial vector v_1. The choice of these parameters relates to the initial search space. In this work we have employed $d_i^+ = v_{1,i}$ and $d_i^- = -\frac{v_{1,i}}{2}$.

$$\begin{aligned} v_2 &= v_1 + [d_1^+ \, d_2^- \, d_3^- \, ... \, d_n^- \,] \\ v_3 &= v_1 + [d_1^- \, d_2^+ \, d_3^- \, ... \, d_n^- \,] \\ &..... \\ v_{n+1} &= v_1 + [d_1^- \, d_2^- \, d_3^- \, ... \, d_n^+ \,] \end{aligned} \tag{4.9}$$

2. Compute the cost function $f_c(v)$ for each vertex, ordering and labeling the vertexes according to their cost function as follows:

$$f_c(v_1) \leq f_c(v_2) \leq f_c(v_3) \leq .... \leq f_c(v_{n+1}) \tag{4.10}$$

3. Compute the centroid excluding the worst vertex as follows:

$$\overline{v} = \frac{1}{n} \left[\sum_{i=1}^{n} v_i \right] \tag{4.11}$$

According to the principle of the FP search, it is necessary to find a new and better vector v_{new} (i.e., a vector which has a lower f_c value) in the direction of the line that connects $\overline{v}$ and v_{n+1} for each algorithm iteration. v_{new} is used together with the

worst vertex v_{n+1} in order to determine the search direction. Hence, the polyhedron is rebuilt in each iteration until the final convergence. Having the $\overline{v}$ point as a starting reference, we define two search directions: *primary search direction* and *secondary search direction*, as illustrated in Figure 4.3 [60].

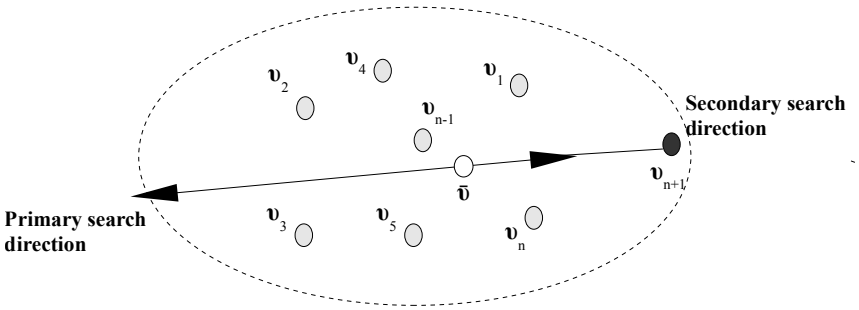

Figure 4.3: Search directions according to the FP principle

4. Determine new vertex v_{new}. Initially, the primary search direction is chosen. If v_{new} is worse than v_{n+1}, i.e., $f_c(v_{new}) > f_c(v_{n+1})$, then the search direction changes to the secondary direction. Actually, the primary search direction corresponds to the opposite direction in which a worse value was found. In this context, the first operation to be performed is the *reflection*. Denoting v_r as the new vertex obtained in the reflection operation, it can be defined as follows:

$$v_r = \overline{v} - \alpha\,(v_{n+1} - \overline{v}) \tag{4.12}$$

where α represents the reflection search step.

If $f_c(v_1) \leq f_c(v_r) < f_c(v_n)$, replace v_{n+1} by v_r, i.e., $v_{new} = v_r$, and go to step 9. If $f_c(v_r) < f_c(v_1)$, go to step 5. If $f_c(v_n) \leq f_c(v_r) < f_c(v_{n+1})$, go to step 6. Otherwise, if $f_c(v_r) \geq f_c(v_{n+1})$, go to step 7.

5. Compute v_e using the *expansion operation*:

$$v_e = \overline{v} - \beta\,(\overline{v} - v_r) \tag{4.13}$$

where β represents the expansion search step. if $f_c(v_e) < f_c(v_r)$, replace v_{n+1} by v_e, i.e., $v_{new} = v_e$, and go to step 9. Otherwise, replace v_{n+1} by v_r, i.e., $v_{new} = v_r$ and go to step 9.

6. Compute v_c using the *outside contraction operation*:

$$v_c = \overline{v} - \gamma\,(\overline{v} - v_r) \tag{4.14}$$

where γ represents the contraction search step. if $f_c(v_c) \leq f_c(v_r)$, replace v_{n+1} by v_c, i.e., $v_{new} = v_c$, and go to step 9. Otherwise, go to step 8.

7. Compute v_c using the *inside contraction operation*:

$$v_c = \overline{v} - \gamma\,(\overline{v} - v_{n+1}) \tag{4.15}$$

where γ represents the contraction search step. if $f_c(v_c) < f_c(v_{n+1})$, replace v_{n+1} by v_c, i.e., $v_{new} = v_c$, and go to step 9. Otherwise, go to step 8.

8. Perform a *shrink operation* by recalculating all vertexes excluding the best vertex [57]. This operation is performed when previous operations have not generated a better vertex, as follows:

$$v_i = v_1 - \sigma\,(v_1 - v_i),\ for\, i = 2, 3, ..., n+1 \tag{4.16}$$

where σ represents the shrink factor.

9. Test the below condition for rebuilding the polyhedron. If the condition is not satisfied, return to step 2. Otherwise, go to step 10. In this case, for every optimization problem, it is necessary to establish a tolerance parameter, say ε, in order to verify if the size of the polyhedron is sufficient small for the convergence around the optimal zone. This verification can be done using the distance from each vertex to the best vertex:

$$\sum_{i=1}^{n+1} \|v_i - v_1\| \leq \varepsilon_1 \tag{4.17}$$

10. If v_1^{old} (i.e., the vector of the previous convergence) has not been initialized yet, set $v_1^{old} = v_1$ and return to step 1. Otherwise, test the following convergence criterion. If the condition is satisfied, terminate the algorithm and output v_1 as a result. Otherwise, set $v_1^{old} = v_1$ and return to step 1.

$$\|v_1^{old} - v_1\| \leq \varepsilon_2 \tag{4.18}$$

In our optimization problems, we have selected the following typical values: $\alpha = 1$, $\beta = 2$, $\gamma = 0.5$ and $\sigma = 0.5$ [59], which are recommended as "default" values in all references considered in this study. The ε_1 and ε_2 were both selected as 0.01. Observed results were sufficient for the solution of the optimization problems. In terms of convergence properties, [67] gave a family of strictly convex functions and a class of initial simplices in two dimensions for which all vertices of the working simplex converge to a nonminimizing point. [68] contains several convergence results in one and two dimensions for strictly convex functions with bounded level sets. Rigorous analysis of the Nelder-Mead method seems to be a very hard mathematical problem. By design, the shape of the working polyhedron can almost degenerate while "adapting itself to the local landscape", and the method uses only simple decrease of function values at the vertices to transform the polyhedron. Hence, very little is known about the convergence properties. Almost nothing is known about the behavior of the method for less smooth or discontinuous functions [69]. The proof of its convergence is not available because it is based on heuristics in which convergence theorems are not applicable [60]. In order to face the problem of convergence to local minima, we have adopted a modified version of the FP method that rebuilds the polyhedron around the solution found in the previous convergence, as seen in step 10 and in test 4.18.

Figure 4.4 depicts one illustration with respect to the polyhedron operations. The first operation is the *reflection*. As seen in Figure 4.4(b), the reflection vertex moves the polyhedron to the opposite direction of the worst vertex (i.e., primary search direction), rebuilding the polyhedron around the optimal space. Despite that this procedure is enough, an additional operation towards the same direction (*expansion*) is verified before the choice of the new vertex. The goal of this operation is to go further through the optimal space, as shown in Figure 4.4(c). When the *reflection* operation gives a worse value for the cost function, a new operation, named *contraction*, is performed aiming to bring the polyhedron back to the optimal region. Figure 4.4(d) shows an example of an *outside contraction* (above) and an *inside contraction* (below).

4.3.1.1 Algorithm

Below, Algorithm 1 summarizes the main steps of the FP method. The optimization problem has n variables. All vectors v_{new}, v_1, v_{n+1}, ε, v_1^{old}, v_r, v_e and v_c have a n dimension. Each element of these vectors corresponds to one value assigned to each variable.

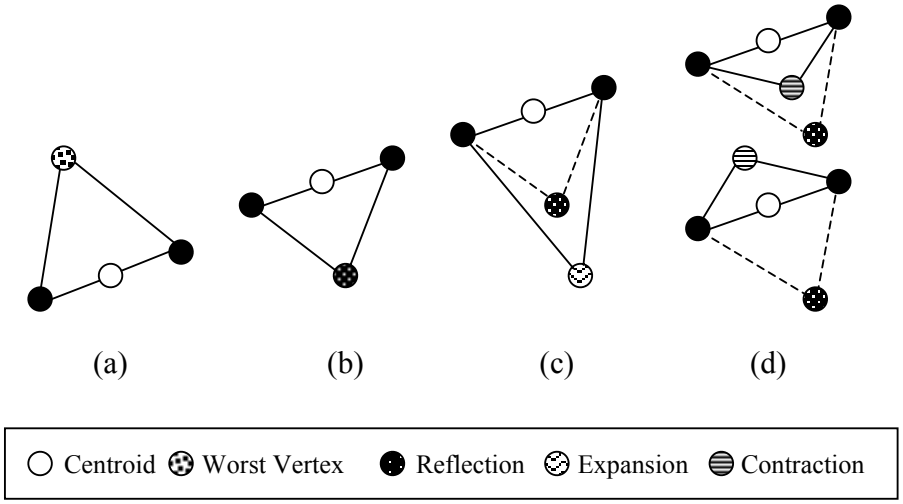

Figure 4.4: FP Operations: (a) Initial Polyhedron (b) Reflection (c) Expansion (d) Contractions

```
1.1  n ← number of variables of the optimization problem;
1.2  initialize the new vector v_new = {0, 0, .., 0} and convergence test vectors ε_1 and ε_2;
1.3  repeat
1.4      initialize v_i for i = 1, ..., n + 1 according to expression 4.9 ;
1.5      repeat
1.6          sort the v_i vertexes according to expression 4.10;
1.7          calculate the centroid vertex v̄ according to expression 4.11;
1.8          calculate v_r according to expression 4.12;
1.9          if f_c(v_1) > f_c(v_r) then
1.10             calculate v_e according to expression 4.13;
1.11             if f_c(v_e) < f_c(v_r) then
1.12                 v_new ← v_e;
1.13             else
1.14                 v_new ← v_r;
1.15             end
1.16         else
1.17             if f_c(v_1) ≤ f_c(v_r) < f_c(v_n) then
1.18                 v_new ← v_r;
1.19             else
1.20                 if f_c(v_r) ≥ f_c(v_{n+1}) then
1.21                     calculate v_c according to expression 4.15;
1.22                     if f_c(v_c) < f_c(v_{n+1}) then
1.23                         v_new ← v_c;
1.24                     end
1.25                 else
1.26                     calculate v_c according to expression 4.14;
1.27                     if f_c(v_c) ≤ f_c(v_r) then
1.28                         v_new ← v_c;
1.29                     end
1.30                 end
1.31             end
1.32         end
1.33         if ! updated (v_new) then
1.34             rebuild the polyhedron according to expression 4.16;
1.35         else
1.36             v_{n+1} ← v_new;
1.37         end
1.38     until exp. 4.17 == TRUE ;
1.39     if empty (v_1^old) then
1.40         v_1^old ← v_1;
1.41         continue;
1.42     end
1.43 until exp. 4.18 == TRUE ;
1.44 accept the vector v_1 as the solution of the optimization problem;
```

Algorithm 1: The FP Method

4.3.1.2 Final remarks

Parameters α and β in expressions 4.12 and 4.13, respectively, can lead to infeasible values in vectors v_r and v_e. In this case, one must consider the following expressions in order to obtain these values:

$$\alpha = \begin{pmatrix} 1, & if\,(2\,.\,\overline{v}^i - v^i_{n+1}) > 0 \;\; for\, i = 1, ..., n \\ \delta\,.\,min\,(\frac{\overline{v}^i}{v^i_{n+1} - \overline{v}^i}), & if\,\exists\, i\; for\,(2\,.\,\overline{v}^i - v^i_{n+1}) \leq 0 \end{pmatrix} \tag{4.19}$$

$$\beta = \begin{pmatrix} 2, & if\,(2\,.\,v^i_{new} - \overline{v}^i) > 0 \;\; for\, i = 1, ..., n \\ \delta\,.\,min\,(\frac{\overline{v}^i}{\overline{v}^i - v^i_{new}}), & if\,\exists\, i\; for\,(2\,.\,v^i_{n+1} - \overline{v}^i) \leq 0 \end{pmatrix} \tag{4.20}$$

where i is the i^{th} element in arrays v^i_{new}, $\overline{v}^i$ and v^i_{n+1}. The parameter $0 < \delta < 1$ is a constant that is chosen in order to avoid negative values in arrays v_r and v_e. In this thesis, we selected $\delta = 0.95$ as the default value.

4.4 Example: DiffServ PBAC Design with Optimization Method

4.4.1 Introduction

This section shows the design of a parameter based admission control mechanism using a training strategy based on simulations, the FP optimization method and traffic models [15] presented in previous sections.

In a DiffServ network, the traffic can be classified into distinct aggregated classes, according to its performance requirements. For example, VoIP traffic can be assigned to a class designed to provide low delay and data traffic can be assigned to a class designed to provide low packet loss. Because of the incertitude on the traffic nature, characterizing a model for the queue length distribution is a difficult task. In the literature, it is possible to find several proposals for designing admission control (AC) algorithms for DiffServ networks. These AC algorithms can be divided into two large categories: PBAC (parameter-based access control) and MBAC (measured-based access control). The PBAC approach defines the AC parameters based only on the theoretical traffic behavior, the queuing disciplines and the network capacity. Usually, the PBAC proposals assume simplifications on the traffic behavior in order to provide an asymptotic bound for the queue tail probability. The MBAC approach, by the other hand, measures the real network traffic in order to dynamically adjust the AC parameters. In fact, MBAC approaches enable to design controllers

that are more robust with respect to the accuracy of the traffic model. However, MBAC controllers are also dependent on some sort of parameter tuning, and a setting that gives excellent performance under one scenario, may give a very pessimistic or too optimistic performance in another scenario [10, 70].

As we mentioned previously, the proposed method of this thesis also permits to build performance models taking into account QoS requirements on a per-flow basis. Even though several techniques have been proposed for designing AC algorithms for providing QoS guarantees to the aggregated traffic, designing AC algorithms for providing QoS guarantees to individual flows is an issue far less addressed. Providing individual flow guarantees poses additional difficulties on the DiffServ AC design.

Here, we address this problem using an alternative approach. We employ the FP algorithm to determine the maximum load that can be accepted by a DiffServ node without deriving an analytical model. The FP algorithm is used to "train" a parameter based admission controller (PBAC) by using a specifically designed traffic profile. It is important to note that it is an off-line training process. After training the PBAC for a specific network and specific statistical QoS guarantees, it can be used to provide these guarantees to distinct offered traffic loads. Because the NLP is a multi-dimensional optimization process, this approach can be applied to, virtually, any AC technique (PBAC or MBAC) with one or more tuning parameters. This hybrid simulation-optimization approach is also justified because even AC algorithms whose design is based on analytical models require parameter optimization. To illustrate the proposed approach, a sample scenario where (aggregated on-off) VoIP traffic and (self-similar) data traffic compete for the network resources was simulated and evaluated, whose models were also presented in Chapter 3.

4.4.2 Related works

A common approach for designing a DiffServ AC algorithm is to determine the maximum amount of traffic that can be aggregated without leading to an excessive buffer overflow. A single-link analysis of several statistical PBAC algorithms that follows this approach is provided by Knightly and Shroff in [9]. Also, surveys comparing the performance of both, PBAC and MBAC algorithms have been presented in [10, 70]. The works discussed on these surveys assume that satisfying the QoS requirements for the aggregated traffic is sufficient to satisfy the QoS requirements for the individual flows. Some works have shown, however, that the end-to-end delay and the packet loss level of individual flows can substantially

fluctuate around the average aggregate performance.

A study presented by Siripongwutikorn and Banerjee [71] evaluated the per flow delay performance with respect to the traffic aggregates. The authors consider a scenario with a single node, where heterogeneous flows are aggregated into a single class. Distinct queuing disciplines have been considered, such as FIFO, static priority, waiting time priority and weighed fair queuing. The authors observed, by simulation, that the traffic heterogeneity, the load condition and the scheduling discipline affect the per-flow delay performance. Notably, the simulation results indicate that it may not be able to achieve delay guarantees for some individual flows based solely on the class delay guarantees when the flows are heterogeneous in a high load condition.

The work presented by Xu and Guerin [72] explored the differences that can exist between individual and aggregate loss guarantees in an environment that enforces guarantees only at the aggregate level. The work develops analytical models that enable to estimate the individual loss probabilities in such conditions. A bufferless single hop scenario with distinct traffic sources have been considered: ON-OFF, constant bit rate periodic and real traffic video sources. The authors points that in order to avoid significant deviations across individual and aggregate loss probabilities, one should avoid multiplexing flows with significantly different rates into aggregates with a small number of sources. They also observed that the per-flow deviation decreases with the number of aggregated sources. The number of sources required to reduce the deviation across flows is significantly higher when the ON-OFF sources have rather different peak and mean rate.

Spliter and Lee [73] have presented an optimization method for configuring a node level-CAC controller for a bufferless statistical multiplexer. The problem assumes a single class of traffic, which must provide QoS expressed in terms of packet-loss constraints. The arriving call-requests are assumed to follow a Poisson distribution and the call durations are assumed to be general. The flow corresponding to each call is modeled as an ON-OFF process. The traffic during the ON period is assumed to be constant. The idea of the CAC controller was to minimize the call blocking probability subject to the packet loss constraints. Because the authors assume a bufferless approach, the optimization process could be modeled in terms of a linear programming problem. The packet loss ratio constraint was imposed to the aggregated traffic instead of individual flows. The authors then present a CAC policy where the decision about the admission of a call is based only on the number of calls in progress. In spite of using an optimization approach, this work differs from our proposal in several points. First, we assume a multi-class scheduler with buffer capabilities. Second, our

AC algorithm is designed to provide per-flow guarantees. Finally, our method is based on non-linear programming approach and does not require a mathematical model to describe the constraints imposed to the problem, which enables to employ the method to complex traffic profiles.

4.4.3 Scenario and Proposal

We consider a sample scenario where (aggregated on-off) VoIP traffic and (self-similar) data traffic compete for the network resources. The VoIP traffic is controlled by an AC algorithm which limits the number of simultaneous active VoIP flows. Also, homogeneous VoIP flows aggregated into a single EF class. The self-similar data traffic is not controlled by the AC algorithm. Instead, it is submitted to a leaky bucket classifier, which determines which packets will receive an assured forwarded (AF) treatment (i.e., packets within the limits of an accorded service rate) and which packets will be treated as best effort (BE). The AC strategy must provide statistical guarantees that each VoIP flow will respect a percentile limit imposed on the end-to-end delay and packet loss performance. This scheme must also provide delay and packet loss performance guarantees for the AF data traffic.

The PBAC algorithm must be capable of answering the following question: what is the maximum number of simultaneous VoIP flows that can be served without violating the performance guarantees? As discussed in section 4.4.2, the per-flow problem is not extensively addressed in the literature. Also, most attempts of deriving a mathematical model for the per-flow behavior assume that the life-cycle of all VoIP flows is identical, i.e., all flows start in the beginning of the evaluation and terminate simultaneously. In this work, the VoIP traffic model follows the model presented in section 3.3. Our work shows that the effect of considering flows with distinct life cycles introduces significant deviations among the individual flows percentile performance, requiring a "per-flow" tuning of the AC method in order to respect the QoS requirements.

In order to train our PBAC controller we have selected a variable VoIP offered load template which follows a "peaked call arrival pattern" [43], as defined in Figure 4.5. This approach is necessary in order to design a PBAC controller which is robust with respect to the fluctuations of the offered load. According to the figure, the evaluation time is divided into ten identical periods. Each period corresponds to a distinct offered load, which is determined by adjusting the TBF parameter. The AFD parameter depends on the average user behavior, and it is kept constant among all periods. The offered load is computed in

terms of an estimation of the blocking probability, by using the Erlang-B functions [50]. In this case, the equivalent number of VoIP lines for a given link capacity is "virtually" defined by the AC method, which imposes a limit to the number of simultaneously active VoIP connections. There is a discrete universe of discourse for the offered load profile. Despite the training including regions with high blockage levels (e.g., 0.9), an operator should use regions with appropriate blocking probabilities according to the ISP policy. Distinct offered load templates could be defined in order to have lower blockages values, also, according to the ISP policy.

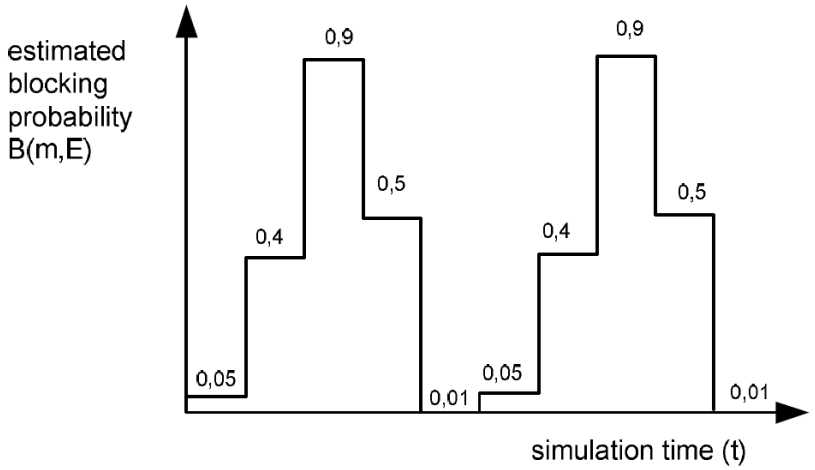

Figure 4.5: Variable offered load template

The data traffic used to train the PBAC algorithm follows the self-similar model, which was presented in section 3.2. In this work, we have adopted $a = 275\,kbit.s$ and $H = 0.76$, as used by Norros in the sample simulation presented in his seminal paper. For each $\triangle t$ window we compute the arrival process and generate a packet series $A_{\triangle t}$ assuming a MTU size (1500 bytes) and a minimum packet size of 46 bytes. The packet series is then submitted to a leaky bucket metering scheme, controlled by two parameters: the bucket size and the transmit rate. The $A_{\triangle t}$ packets within the bucket limits are tagged as AF PHB, and the non-conforming $A_{\triangle t}$ packets (i.e., packets that could not be inserted into the bucket) are tagged as BE. In this work, we have adopted $\triangle t = 10\,ms$. The scheduling discipline of the DiffServ node is a priority queuing mechanism with three non-preemptive finite size queues: EF, AF and BE. In this scheme, a queue is served only when the higher priority queues have no packets waiting to be served. The AF and BE queue sizes were limited to $250\,ms$.

For VoIP traffic, each flow was modeled with a $200\,byte$ packet size, already included

the payload and the protocol headers. During the ON period, the arrival interval between packets is $20\,ms$, which represents a peak rate of $80\,kbps$ and an average rate of $32\,kbps$ for each individual flow. We have limited the VoIP (EF) queue size to $50\,ms$, i.e., packets that exceed the $50\,ms$ delay are dropped.

We assume that the PBAC algorithm is capable of determining the exact instant when each VoIP flow starts and terminates. Therefore, the only tuning parameter of this controller is the maximum number of simultaneously active flows.

4.4.4 Optimization Problem

In order to determine the cost function (i.e., the objective function of optimization problem) to be applied in the FP method, it is necessary to define the per-flow VoIP and the AF traffic requirements. Because this work considers a single node AC, these requirements are expressed in terms of the delay and the packet loss caused by the node. The per-flow VoIP requirement is expressed as the percentile of packets that can exceed a delay bound or be dropped in a single flow. For example, a QoS requirement (97_{th},$50\,ms$) defines that a maximum delay of $50\,ms$ must be observed by 97% of the packets of each individual flow, i.e., only 3% of the packets can violate the QoS requirements by exceeding the $50\,ms$ limit or being dropped. The AF traffic requirement is imposed in terms of the overall packet-loss ratio. We have limited the AF queue to a size corresponding to the maximum admissible delay assigned to the AF packets (i.e, the single node delay contribution), so all packets that exceed this delay bound are dropped.

The main idea of the cost function is to induce the optimization process to find a solution for the admission controller that maximizes the number of flows served without violating the QoS requirements of both, VoIP and the AF data traffic. Considering an evaluation period, let F_a, $\overline{F}_a$, $\widehat{F}_a$ and F_v be, respectively, the absolute number of accepted flows, the average number of active flows, the estimated maximum number of active flows and the absolute number of admitted flows that violate the QoS requirements. We also define AF_d as the percentage AF packets dropped with respect to the total number of packets marked as AF. The cost function can, then, be represented as:

$$f_c(x) = \left[\ 1 - \frac{\overline{F}_a}{\widehat{F}_a}\ \right] + \left[\ \varphi\,\frac{\overline{F}_v}{F_a}\ \right] + \left[\ \phi\,AF_d\ \right] \tag{4.21}$$

The x vector represents the AC variables that must be optimized. Because we have assumed a simple PBAC controller, in this case, the variable is the maximum number of

simultaneously active flows. The $\overline{F}_a$ term is not required to be precise, as it works only as a normalization factor. It can be computed using the techniques suggested by [74]. The second component introduces a penalization for violating the QoS requirements. The φ (undimensional) factor defines the "penalty" for admitting a violated flow. The third term defines a penalization for dropping the AF traffic. The penalization weight is controlled by the ϕ (undimensional) factor. Both φ and ϕ are inputs to the optimization process and depend on the service provider business policy. Extremely high occupation rates that induce too many VoIP or AF violations will be penalized by the cost function. Therefore, no special treatment is required for avoiding infeasible solutions.

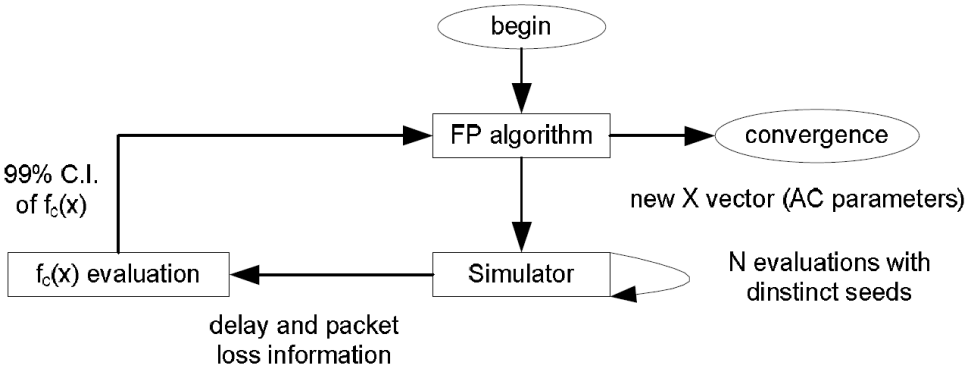

Figure 4.6: Optimization Flow for the Proposed PBAC controller

Figure 4.6 illustrates the complete optimization flow, where a simulator and the FP method are combined. The simulator is activated for computing the cost function of each new vertex of the polyhedron. In order to reduce the seed influence, N simulations with distinct seeds are computed for each new vertex generated by the FP method. During a simulation, the AC parameters are tested against a variable load scenario that follows a "peaked call arrival pattern" shared with the self-similar data traffic, as described in section 4.4.3. We have employed N=10. The cost function (f_c) is computed for each seed, and a 99% confidence interval for the (f_c) average is determined by assuming a t-distribution [50]. The cost assigned to a PBAC solution corresponds to the worst bound in this interval (i.e., the highest cost function). The AC parameters obtained using the optimization approach are valid for a specific scenario defined by the link capacity, the per-flow delay/packet-loss bound, the leaky bucket parameters, the per-flow VoIP percentile performance and the ϕ and φ parameters.

4.4.5 Simulation and Results

In this section, we have evaluated our approach by optimizing the PBAC method for distinct link capacities and distinct leaky bucket parameters. In all scenarios, the VoIP offered load followed the template defined in Figure 4.5, but the conditions of the data traffic were modified by adjusting the leaky bucket parameters: service rate and bucket size. Figures 4.7, 4.8 and 4.9 have considered a scenario that imposed a per-flow delay bound of $50\,ms$ for the 97^{th} percentile of the VoIP packets and a delay bound of $250\,ms$ for the aggregate of AF packets. The performance parameters were adjusted respectively to $\phi = \varphi = 10$. This represents a strong penalization for violating VoIP or AF guarantees. As a consequence, in all scenarios, there was no violation of VoIP flows or AF traffic. Therefore, this information was omitted in the figures.

Both, the self-similar and the VoIP traffic can significantly vary according to the seed values. To avoid misinterpretation, the results presented in this section correspond to the average of 30 simulations with distinct seeds, presented with a 99% confidence interval, i.e., after obtaining the optimal values for the PBAC controller, we perform such evaluation considering 30 simulations with distinct seeds. Evaluating the approach with so many distinct seeds assures that the simulated results achieved by the controller will be robust enough to be reproduced in the real world.

The left side of Figure 4.7 illustrates the effect of the link capacity on the share of bandwidth occupied by the AF and the VoIP traffic. The right side of the figure illustrates the effect of the link capacity on the BE traffic drop rate and the average number of active flows per Mbps. One can observe in the figure that the number of VoIP flows per Mbps increases with the link capacity. As a result, the share of the link bandwidth occupied by the VoIP traffic also increases with the link capacity. This effect is justified because a longer VoIP queue can be tolerated in terms of packets when a fixed maximum delay is imposed to a node with higher service rate. This effect is also observed for the AF traffic. Also, note the smoothing effect obtained by aggregating more VoIP flows, as observed by the reduced dispersion around the active VoIP flows average.

Figure 4.8 illustrates the effect of modifying the leaky bucket rate (expressed as a % of the link capacity). In this scenario, the link capacity was fixed in 8 Mbps. Note that increasing the leaky bucket rate increases the offered load of AF packets. One can observe that the higher priority VoIP traffic is limited by the PBAC controller in order to accommodate the additional AF traffic. In all evaluations, the average bandwidth of the

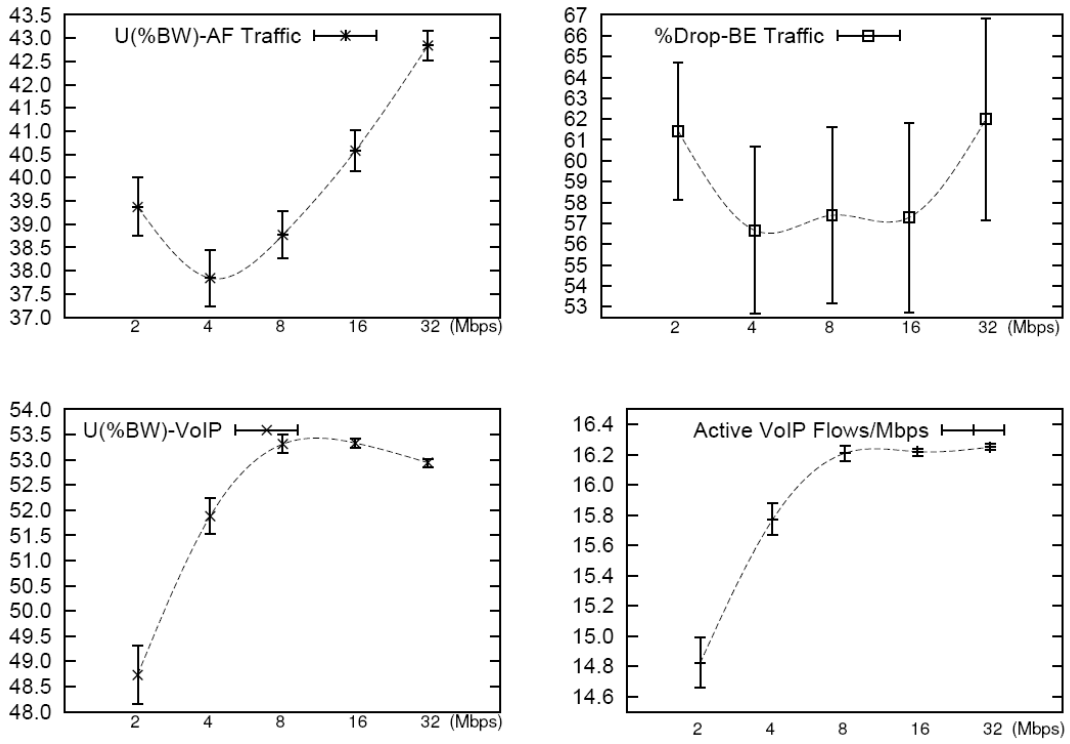

Figure 4.7: Proposed PBAC controller: Effect of the Link Capacity

self-similar traffic (i.e., offered load before marking) was fixed in 50% of the link capacity (m parameter). However, increasing the bucket rate above 50% still affects the traffic, due the fBm variation.

Figure 4.9 illustrates the effect of modifying the leaky bucket size. Again, in this scenario, the link capacity was fixed in 8 Mbps. Note, in Figure 4.7, that the share of bandwidth occupied by the AF traffic was always less than 50% (the leaky bucket rate). This is the effect of the small bucket size adopted in this scenario (only 2 MTU). Increasing the bucket size enables the accommodation of the self-similar traffic variation, increasing the AF traffic rate. Consequently, the AF traffic shaped by the leaky bucket is not as well-behaved as the traffic in Figure 4.7. In this case, the PBAC had to reduce the number of VoIP flows per Mbps accordingly.

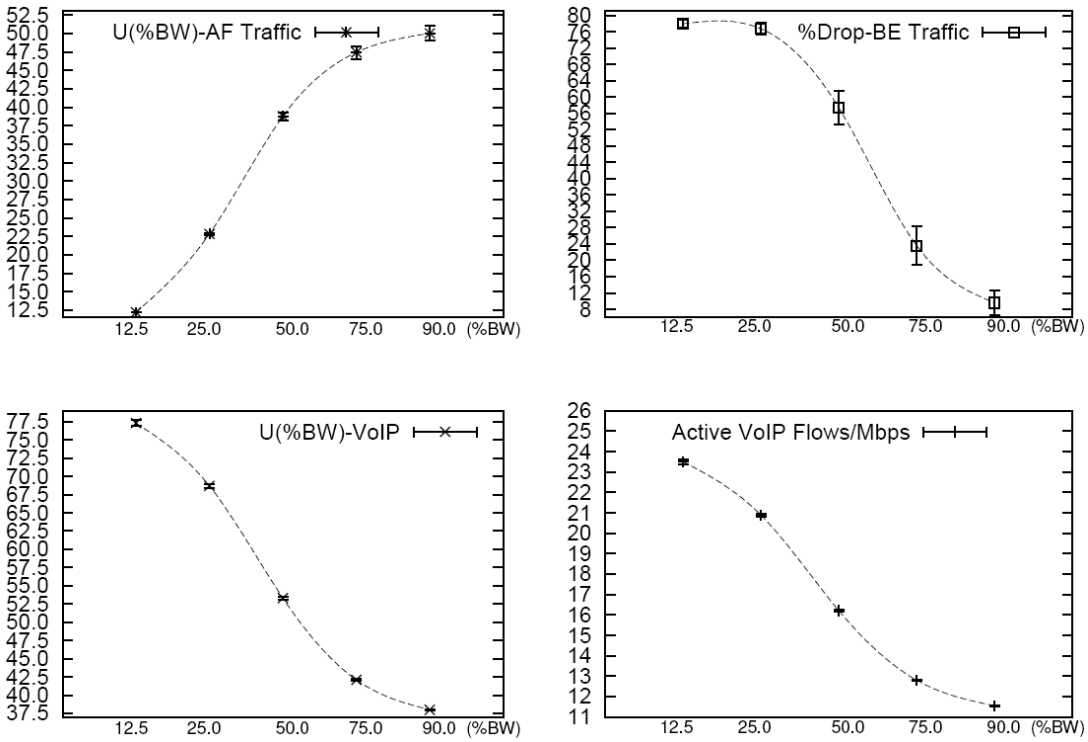

Figure 4.8: Proposed PBAC controller: Effect of Leaky Bucket Rate

4.5 Conclusion

In this Chapter, we reviewed the main concepts of optimization theory in order to introduce the method that will be employed for solving optimization problems in this thesis. As we saw, our problem belongs to the non-linear programming category. Several techniques are available for solving NLP problems. In this case, we presented the Flexible Polyhedron method which is a zero-order direct search method that does not require derivatives of the cost function. This feature allows us to create optimization problems that handle several parameters in a very flexible manner, where objective functions can be easily used as results of complex computer simulations. During each iteration, a new solution is computed by applying arithmetic operations on the Polyhedron vertexes, and the worst vertex is replaced. Successive iterations are performed leading to a reduction of the Polyhedron dimension until a convergence criterion is achieved. In order to face the problem of local minima, we will employ a modified version of the FP algorithm. According to this modification, the polyhedron is rebuilt around the convergence vertex. The final convergence criterion will

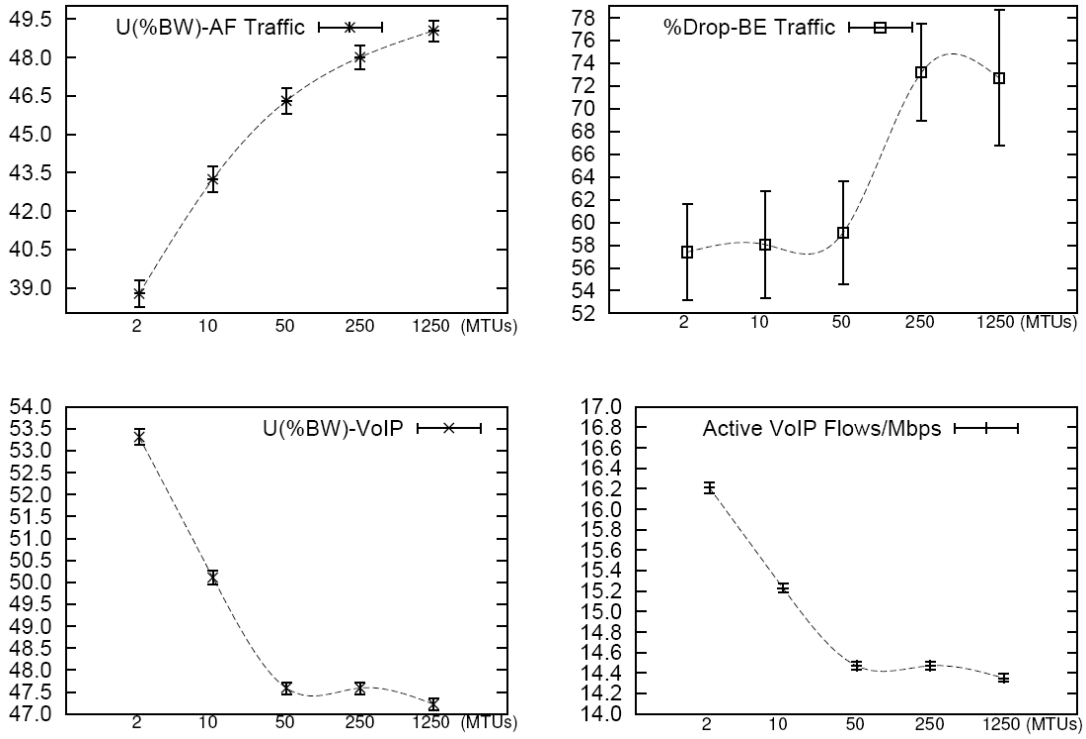

Figure 4.9: Proposed PBAC controller: Effect of Leaky Bucket Size

be achieved only if the current convergence minimally differs (according to a specific error) from the previous convergence.

In order to illustrate the use of the FP method, we also showed an optimization approach that helps the design of DiffServ AC algorithms capable of providing per-flow delay packet-loss guarantees. The approach was illustrated considering a scenario where a node link was shared by distinct types of traffic: VoIP and self-similar traffic marked by a leaky bucket algorithm. A typical DiffServ node capable of scheduling only large classes of traffic (i.e., flow-unaware) was studied. Because most results obtained in the literature target only the performance of aggregated traffic, there is no accurate analytical model capable of treating a per-flow scenario with this type of scheduler.

The strategy described here was capable of achieving the required performance results in all scenarios, including distinct link capacities and variations on the level of self-similar packets marked as assured forwarding. Also, the controller is robust with respect to the offered load variation.

Chapter 5

A New Method for Modeling Queue Behavior

5.1 Introduction

THIS Chapter presents the proposal of a new generic method for modeling queue behavior of network nodes aiming to provide a methodology for building the corresponding performance models. These models (e.g., expressed in terms of packet delay and loss rate) represent the performance experienced by incoming traffic profiles where it is possible to establish the relationship between the traffic load in queues of network nodes and the resulting QoS performances. The proposed method takes into account many complex issues faced in typical deployment scenarios: Multiple priority queues, non-exponential service times and long term correlated traffic. Also, we address the problem of providing per-flow guarantees, which poses additional difficulties because it is also necessary to take into account the fluctuations of the flows lifecycles in order to determine the impact of the queue behavior on the performance of the individual flows. Such statements leads to a scenario in which it is very hard to develop an analytical model.

The proposed method combines non-linear programming (NLP) and simulation to build a fuzzy logic based model capable of determining the performance of a network node. Using such strategy, which is based on a general off-line method to produce the equations capable of representing outputs for the whole space of the specified input traffic parameters, it is possible to find out the optimal values that can be used for the configuration of network nodes, with important applications on traffic engineering and capacity planning. This ap-

proach does not require the derivation of an analytical model and can be applied to any type of traffic. Also, this methodology includes a training method that permits the application of any type of performance metric.

In order to clarify the contributions of this generic method and its distinct approaches employed for fuzzy modeling, we divided its presentation into two sections according to the strategy used for modeling the consequents of fuzzy rules and the corresponding applications of results. After reviewing the state of art in section 5.2, section 5.3 presents our method considering an application scenario involving a fuzzy predictor. As we will see, the consequents of fuzzy rules are modeled using singletons and the results are expressed in terms of crisp values considering the corresponding performance metrics. Section 5.4 presents a more generic approach. In this case, our method is employed to provide an output fuzzy set as a result instead of crisp values. We use a strategy based on normalized probability distributions in order to build a performance model of a network node in which a queue is fed by a self-similar traffic source. We propose a time based fuzzy model for the MVA approximation for loss probability analytical model.

5.2 Related works

There are two main research domains on fuzzy logic related with our work. The first domain focuses on extending existing queue models to provide performance measures expressed by membership functions. The second domain employs fuzzy models to build active network traffic controllers.

The application of fuzzy techniques to model queuing systems has been initially reported by Li and Lee [75]. Based on the Zadeh′s extension principle, on the concept of possibility and on fuzzy Markov chains, they proposed a general approach for fuzzy queue analysis. According to their approach, the (precise) original queue model parameters are considered imprecise and their possibility distribution is derived by applying Zadeh′s extension principle. The result is a fuzzy queue model with uncertain parameters defined according to their membership functions. Considering the complexity of deriving these membership functions, some techniques based on parametric non-linear programming have been investigated, which obtain the fuzzy parameters from their alpha-cuts. This approach has been applied by Chen [76] to model queues with bulk service and by Ke and Lin [77] to analyze queue systems with unreliable servers. More recently, Pardo and de la Fuente [78] derived two fuzzy models for priority queues with exponential arrivals characterized by fuzzy pa-

rameters. The first model assumed a fuzzy exponential service time and the second model assumed a fuzzy deterministic service time.

The method proposed in this thesis can be compared to the fuzzy queue modeling approach, but differently from the previously discussed works, its goal is not to determine membership functions for the uncertain parameters of existing queuing models, but to derive a fuzzy system to model the relationship between the crisp input and output variables using training techniques. Also, we use these training techniques to build a fuzzy system that outputs a fuzzy set as a result considering crisp input variables. Although the aforementioned models have all been derived from relatively complex queue models, in realistic networking scenarios, as the multi-queue DiffServ node considered in our study, the complexity goes far beyond. For example, the use of Poisson process to model the input traffic can be viewed as a major simplification for the elastic data traffic. In fact, since long range dependency and self-similar characteristics have been identified in LAN [36], WAN [34], and in the WWW [35] traffic, the self-similar model is presently considered a more accurate model. Another example of traffic model complexity is to consider the input traffic to be constrained by a token-bucket mechanism. This is indeed a broadly employed technique in IP networks that surely complicates the analytical model. A last example of traffic modeling complexity is to include in the model the ability to handle per flow parameters. As discussed previously, most of queue models are suitable for the aggregate traffic and are unable to take into account per flow parameters.

The application of fuzzy techniques to build network traffic controllers addresses a different context. The goal is to find control laws that depend on the predicted behavior of the controlled system. The output predictions are computed according to a process model. In order to achieve the controlling goals, the system input is sampled, and a model that captures the dependence of the system's output on the current measured variables and the current and future inputs is derived. The model should be able to predict the future behavior which, in general, includes non-linear relations between the involved variables. Fuzzy modeling techniques have also been successfully applied in the construction of such models, specifically in the control the network traffic with non-linear characteristics. Chen at al. [79] have used a fuzzy adaptive prediction technique in order to overcome the difficulties in congestion control design due to non-linear time varying characteristics of network traffic. In their study, the traffic flow from controlled sources in ATM networks is described by a fuzzy ARMAX (Autoregressive Moving Average eXogenous) model, which is translated to a fuzzy predictive traffic model. The parameters of the predictive model are estimated by

a normalized least mean squares (NLMS) algorithm in real time, allowing the controller to be constructed based on this recursive parameter estimate.

A self-adapted wavelet-based fuzzy traffic predictor has been proposed by Gou and Hu [80]. They use the Mallat wavelet transform to decompose the input traffic into components at multiple time scales. After rebuilding the time-series, traffic components at each time scales are predicted independently and then combined to form the predicted traffic. A fuzzy logic is then constructed, with the corresponding membership functions parameters being derived from a clustering algorithm. In order to address the construction of the logic for fuzzy controllers, Hsiao and Su [81] have proposed a general approach to generate the fuzzy rules for an ARMA (Autoregressive Moving Average) predictor model by a two phase process: the first phase generates the fuzzy rules from on-line data, whereas the second updates the fuzzy systems parameters by means of an on-line learning phase. The algorithm for generating the fuzzy rules first defines the domain intervals for input and output variables, divide the input and output spaces into fuzzy regions and select the parameters for Gaussian membership functions. A new rule is generated if the difference between the actual output and the predicted output is greater than a desired error. Then, conflicts between rules are eliminated and redundant rules are discarded. In the learning phase, the premise part of the rules is updated according to an adaptation equation, looking for the reduction of the prediction error. Again, if the error overcomes a pre-defined bound, a new rule is generated.

Finally, the study of Din and Fisal [82] investigates fuzzy traffic control in a scenario very close to the one we use to demonstrate our method in Section 5.3. They propose a fuzzy logic token bucket bandwidth predictor for assured forwarding traffic aggregates at DiffServ nodes. The DiffServ traffic is accommodated in three queues that are constrained by a token bucket mechanism: EF for real time, AF assured forwarding, and BE for best effort, in an arrangement that is quite similar to ours. However, their traffic model is quite different: they model the EF traffic as constant bit rate, the AF traffic by a Pareto distribution, and the BE by an exponential distribution. They discuss the fuzzy predictor only for the AF queue, where two input variables are considered, the average rate to peak rate ratio and the available bandwidth ratio, both computed from a rate estimation mechanism. The fuzzy predictor outputs a prediction factor, which is added to the ratio between the average and the peak rate and multiplied to the current token bucket rate, providing a new token bucket rate prediction, used to feed back the token bucket mechanism and the admission control mechanism.

Similarities between our work and the previously discussed model based on predictive control can be identified. We have used a similar method to calibrate our fuzzy logic as the one used by Hsiao and Su [81], that is, both studies try to minimize the prediction error. Also, the scenario considered in Din and Fisal [82] paper is almost the same of the one considered in our study. However, our approach is quite dissimilar in both cases. Although we can envisage the use of our predictor in on-line traffic control, our work proposes a general off-line method to produce the equations capable to predict outputs for the whole input space, with important applications on traffic engineering and capacity planning.

5.3 Building a Fuzzy Predictor

5.3.1 Problem Statement

5.3.1.1 DiffServ and Service Classes

Differentiated Services is a general architecture that may be used to implement a variety of services. Even though the DiffServ methodology is supposed to be generic, the IETF has supplied some guidelines suggesting how to configure DiffServ nodes according to the service classes supported by the network [83]. Service class definitions are based on the distinct traffic characteristics and required performance of application/services. Application/services with similar traffic characteristics and performance requirements are mapped into the same service class. Table 5.1 presents the main service classes proposed by the RFC 4594 [83] and their respective performance requirements.

Table 5.1: IETF Service Class Guidelines

Service Class	DSCP	Conditioning at Edge	Tolerance to			Queuing
			Loss	Delay	Jitter	
Telephony	EF	none for trusted traffic	very low	very low	very low	priority
Multimedia Conferencing	AF4(1-3)	two-rate three color marker	low/med	very low	yes	rate
Multimedia Streaming,	AF3(1-3)		low/med	medium	yes	rate
Low Latency Data,	AF2(1-3)		low	low/med	yes	
High-Throughput Data	AF1(1-3)		low	med/high	yes	
Standard (Best Effort)	DF					rate

In the network core, the service classes are identified by the DSCP (Differentiated Ser-

vices Code Point) field, present in the header of the IP packets. The IETF has defined a set of standard DSCP values and proposed a mapping between the DSCP values and the service classes, as shown in Table 5.1. The DSCP values are also mapped to a forwarding behavior that must be implemented by the DiffServ node in order to meet the respective QoS performance. Three basic forwarding behaviors have been defined: Expedited Forwarding (EF), Assured Forwarding (AF) and Default Forwarding (DF). The EF behavior is recommended for real-time (inelastic) traffic. The AF behavior is recommended for rate adaptative traffic, i.e., when the receiver provides feedback to the sender about loss or delay variation, and the sender adjusts its transmission rate in order to approximate the estimated network capacity (e.g., TCP data traffic). The AF traffic should be more elastic by nature. However, the RFC 4594 has assigned as AF the Multimedia Conferencing service class, which has an inelastic delay requirement. The DF behavior is only recommended for elastic traffic with almost no guarantees, and corresponds to the basic best effort service.

As shown in Table 5.1, presently, the IETF has chosen one EF class, four AF classes and one DF class. The AF classes are supposed to be conditioned at the edge by a token-bucked policing mechanism called two-rate three color marker. The mechanism meters an IP packet stream and marks its packets based on two rates, Peak Information Rate (PIR) and Committed Information Rate (CIR), and their associated burst sizes. For the AF4 class, for example, packets are marked AF41 (green) if they are below the CIR, AF42 (yellow) if they are between CIR and PIR and AF43 (red) if they are above PIR. A DiffServ node uses the color information in order to decide which packets should be discarded during congestion conditions. By choice of the operator, a DiffServ node could not be able to differentiate all three colors in a service class. In this case, the unsupported colors should be forwarded using the standard service class.

In order to provide the required behavior for the distinct service classes, a DiffServ node employs multiple queues, as illustrated in Figure 5.1. The incoming traffic can arrive to a DiffServ node from one or more links. Internally, it is classified and assigned to a corresponding queue by a classifier based on the DSCP value. There are many possible approaches for mapping the service classes into the queues. One possible approach would be to create a distinct queue for each service class indicated in Table 5.1. It is also possible, however, to assign multiple service classes into the same queue, but the individual performance requirements of each service class must still be respected (i.e., the queue must satisfy the service class with the strictest performance requirements). The QoS performance of a queue can be represented in terms of the amount of loss, delay and jitter imposed by the

queue on the traffic injected to the output link. The QoS performance is strongly dependent on the amount and profile of the incoming traffic. It is also dependent on the scheduler algorithm adopted by the node. For example, in a priority queuing system, packets awaiting transmission in a queue are served only if the highest priority queues are empty. In a rate queuing system, a specified rate is assigned to each queue, avoiding the starvation of the lowest priority queues. Note in table that only the EF service class is recommended as priority queuing.

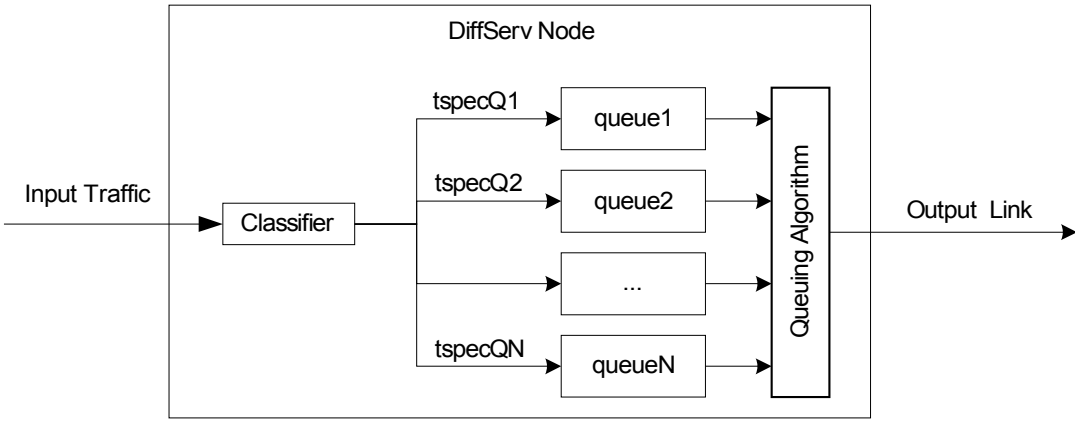

Figure 5.1: Multi-Queue Node Representation

5.3.1.2 Training Method

In this section, our method is employed for building a fuzzy system capable of predicting the QoS performance of each queue in a DiffServ node. By using the fuzzy predictor, it is possible to determine how much traffic could be accepted without compromising the performance objective of the service classes assigned to the queue. The fuzzy predictor takes the traffic specification assigned to each queue as input (noted in Figures 5.1 and 5.2 as tspec) and outputs the predicted performance of each queue in terms of loss, delay and jitter. An important step in designing a fuzzy predictor is to properly define its membership functions in order to capture any non-linearity in the queues behavior under distinct traffic load conditions. In order to accomplish that, our method proposes a training approach based on multiple simulations under distinct traffic load conditions. The input of the training problem is the universe of discourse of the tspec parameters. For example, the traffic assigned to the AF queues can be characterized as a certain source of traffic constrained by

a token bucket mechanism defined by parameters such as bucket size and bucket rate. In this case, the universe of discourse of the input parameters would be the range of acceptable source types, bucket sizes and rates that the fuzzy predictor should be able to interpret. Similarly, telephony traffic could be characterized as the number of simultaneous calls using a certain type of codec. In this case, the universe of discourse could be the range of acceptable values for the number of calls and codec types.

The training method is formulated as two-simultaneous optimization problems, with opposite objectives, as illustrated in Figure 5.2. The first optimization problem, called *Membership Optimization* consists of adjusting the membership functions for the fuzzy predictor in order to minimize the prediction error (i.e., the difference between the performance predicted by the fuzzy system and the one determined by simulation). The second optimization problem, called *Simulation Optimization* consists of finding the set of load traffic conditions that maximizes the prediction error. Note that, without the *Simulation Optimization* algorithm, we should have to exhaustedly explore the multi-dimensional space defined by tspec parameters in order to assure that the non-linearities in the queue behavior have been properly modeled by the fuzzy system.

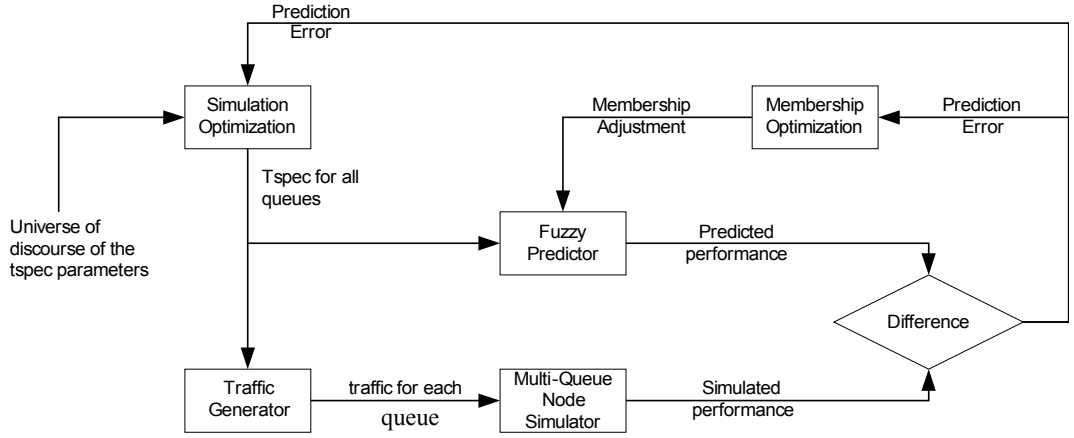

Figure 5.2: Off-Line Dual-Optimization Fuzzy Predictor Training Method

Although the method illustrated in Figure 5.2 can be considered generic, it is impractical to generate a single model capable of representing any possible implementation of a DiffServ node, as the number of membership functions necessary to model the fuzzy predictor increases with the number of tspec parameters. The model is also dependent on the number of queues, the queuing algorithms (including active queue management algorithms,

such as preventive drop algorithms) and the bandwidth available at the output node link. By using our approach, it is only practical to generate a prediction model for a specific combination of number of queues, traffic profiles, queuing algorithms and link capacities. Gladly, the DiffServ methodology recommends adopting a static configuration for the core routers, by defining a standard set of service classes and corresponding per-hop behaviors. This is fairly suitable to our approach. DiffServ core routers with similar queuing configuration and link bandwidths could be represented by the same fuzzy predictor. Dissimilar nodes, however, should be represented by distinct fuzzy systems.

As it will be presented in Chapter 6, in order to illustrate and evaluate our proposal, we have considered a sample scenario described in Table 5.2. As recommended by IETF, we have assigned only VoIP traffic to the EF queue. It corresponds to most scenarios employed in practice. However, we have gone beyond to most studies in this field by imposing that the EF queue must satisfy per-flow performance requirements instead of just aggregated performance. We have assigned all assured forwarding service classes to a unique queue, designated as AF. To aggregate multiple service classes in a single queue is also a common practice. In this case, we are interested in predicting the aggregated performance of the queue. Finally, all traffic exceeding the single rate conditioning at the edge is treated as best effort. All queues are treated by a strict priority scheduler using reasonable queue size limits. It is important to note that the fuzzy predictor will be used in order to define the actual behavior of the queue. For example, if most of the AF traffic is real-time in nature, then the traffic engineer must use the fuzzy predictor in order to define how much traffic can be assigned to the node without leading to an excessive delay. If the traffic is elastic than the traffic engineer must avoid excessive drops.

It is also important to note that the result obtained by the fuzzy predictor does not represent an end-to-end performance prediction. In fact, the performance prediction of all nodes along a path must be taken into account in order to determine the end-to-end performance. As each node along a path will possibly receive distinct amounts of traffic, it will operate with distinct levels of QoS performance. As the fuzzy predictor can predict the performance for any value of the tspec parameters within the universe of discourse, once the maximum load assigned to each DiffServ node in the network is known, it is possible to predict the end-to-end performance for any path in the network. The main application of the fuzzy predictor is traffic engineering of MPLS-DiffServ networks as described in [4]. Although the predictor is not an AC algorithm, its output can be employed to build an on-line AC strategy, being embedded into the network nodes in order to help in the decision

making of signalization protocols such as RSVP-TE. With this application in mind, we have defined the fuzzy predictor to be as simple as possible, to minimize real-time calculations.

Table 5.2: DiffServ Node Configuration

Type of Traffic	Queue	Conditioning at Edge	Predicted QoS Performance Metric	Maximum Queue Size	Queuing
VoIP	EF	none = trusted traffic	per-flow delay per-flow drop	50 ms	priority
Token-bucket conditioned self-similar traffic	AF	single-rate two-color marker	aggregated delay aggregated drop	250 ms	priority
Standard (Best Effort)	DF	exceeding AF traffic	aggregated drop		

Generically, the EF queue can receive several aggregates of homogenous VoIP flows as indicated in equation 5.1:

$$tspec_{EF} = (\sum_i N_{T1,i}\,, \sum_i N_{T2,i}\,, \sum_i N_{T3,i}\,, ...) \tag{5.1}$$

where: $N_{Tj,i}$ is the number of the VoIP flows of the i^{th} aggregate using codec type T_j.

In the general case, the EF class can receive several aggregates of VoIP traffic using distinct codecs. The input for the fuzzy system is a merged tspec described in terms of the maximum number of simultaneously active VoIP flows using each type of codec. Note that because we are interested in a per-flow prediction of the EF performance, we could not supply the traffic specification in terms of total required bandwidth. Fortunately, there are not many distinct VoIP codec types employed in practice, as shown in [84]. The VoIP aggregate traffic is not conditioned via any mechanism. In fact, as stated by the IETF document [83], the policing is optional for packet flows from trusted sources whose behavior is ensured via other means (e.g., administrative controls on those systems). In our case, the training method keeps the control of the number of simultaneous VoIP flows.

It is important to mention the application scenario we are envisioning for these aggregates of homogenous VoIP flows in terms of per-flow requirements: Considering such aggregate traffic, it is necessary to be more conservative with respect to resource reservations if one needs to guarantee performance levels for all individual flows that form this

aggregate. In this case, how conservative should it be necessary? Instead of a simple over-provisioning approach, our approach is to permit estimating such value by allowing us to establish performance metrics in terms of individual flows. This strategy will allow a more conservative provisioning for aggregate traffic which is intended to better achieve per-flow requirements.

For the AF traffic, the AF queue can receive several aggregates of homogenous token bucket conditioned self-similar traffic as indicated in equation 5.2:

$$tspec_{AF} = \sum_{i} (tbsize_i, tbrate_i) = (\sum_{i} tbsize_i \, , \sum_{i} tbrate_i \,) \tag{5.2}$$

where, $tbsize_i$ is the bucket size for the i^{th} aggregate and $tbrate_i$ is the rate for the i^{th} aggregate.

Here, the token bucket mechanism is according to the description given in Annex A.4, where we are interested in taking into account the effect of the bucket size, as it can be noted in equation 5.2. It is important to note that in Chapter 4 we presented in section "Example" a preliminary work, which employs a more specific scenario using a simple leaky bucket mechanism, as described in Annex A.3.

Equation 5.2 denotes a "merged" *tspec* used as input to the fuzzy system, which is represented by only two parameters. A total bucket rate and a total bucket size. This additive property of token buckets is an approach suggested by IETF in the RFC 2211 [85]. Following this IETF recommendation, our proposal considers token bucket parameters for the aggregate traffic, which is calculated as a merged value formed by a sum of individual users token bucket parameters.

5.3.2 Fuzzy Predictor

The Fuzzy Logic System of the proposed predictor for modeling the performance of multi-queue network nodes is shown in Figure 5.3. As illustrated, the predictor receives the traffic specification assigned to each queue in the node as input variables and outputs the QoS performance prediction (usually expected delay or drop) for only one of the queues in the node, i.e., we generate an independent fuzzy model for each queue in the node. By the same reason, we also generate a distinct fuzzy logic for each metric evaluated in the queue.

To output a result, the predictor performs three main steps: fuzzification, inference and defuzzification, as presented in Chapter 2. Each fuzzy logic subsystem illustrated in Figure 5.3 requires the design of the membership functions for all parameters of the EF

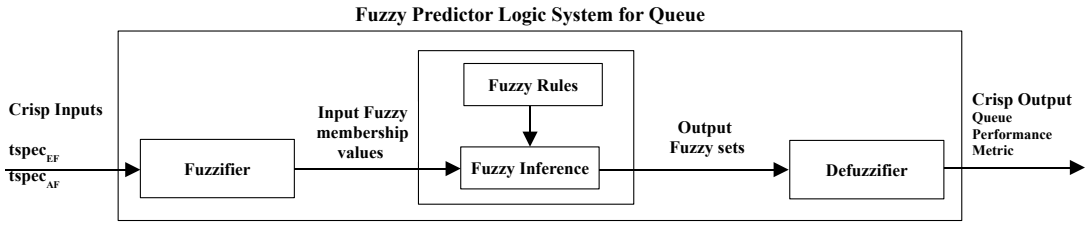

Figure 5.3: Fuzzy Predictor Logic System

and AF traffic specification. As shown in expression 5.1, the merged traffic specification for the EF queue corresponds to the number of active VoIP flows (N_i) of each codec type(T_i) aggregated in the EF class. The term N_i means the number of simultaneous VoIP flows which is kept constant along the simulation run. In this case, following the VoIP model presented in Chapter 3 Section 3.3, each VoIP flow has different duration. Thus, when one flow ends another one is activated in order to keep the N_i amount. The training method controls the number of flows according to the amount required by the optimization phase. There is no admission control. Also, because of the same reason, there is no variable offered load as used in the example in Chapter 4. The merged traffic specification for the AF queue is represented by a merged token-bucket rate (tbrate) and a merged token-bucket size (tbsize). Therefore, the number of parameters that define the EF load corresponds to the number of distinct codec types aggregated in the EF class. In contrast, the number of parameters that defines the AF load is always two.

Each traffic specification parameter must be modeled by an input membership function in the fuzzy system. Several choices are required in order to define these membership functions. Despite the fact that some related issues are subjective, there are some common recommendations that can be considered as a starting point [86, 20]:

- Symmetrically distribute the fuzzy set across the defined universe of discourse;
- Overlap adjacent fuzzy sets to ensure that no crisp value fails to correspond to any set and to help in ensuring that more than one rule is involved in determining the output;
- Use triangular or trapezoidal membership functions, as these require less computation time than other types;

- Start with triangular sets. All membership functions for a particular input or output should be symmetrical triangles of the same width. The leftmost and the rightmost should be shouldered ramps;

- Overlap at least 50%. The widths should initially be chosen so that each value of the universe is a member of at least two sets, except possibly for elements at the extreme ends.

Despite the strategy employed for the basic definitions, the FP optimization method will provide the final configuration for the membership functions. In the following section, we present the proposed design for one of the two subsystems that will be presented in Chapter 6. The short description of the scenario provided here is intentional; it is only intended for a better understanding of method procedures that will be presented in next sections. We consider the design of the fuzzy predictor for the AF Drop subsystem. In this case, we have three input variables: $tspec_{EF} = N_{G711}$, $tspec_{AF} = (tbrate, tbsize)$; and one output crisp variable related to the AF Drop performance metric. VoIP traffic aggregate of codec type G711 and self-similar traffic constrained by a token bucket mechanism compete for the node output link.

Figure 5.4 illustrates a possible definition for the input membership functions of the fuzzy prediction AF aggregate drop subsystem considering a single codec type (e.g., G711). In figure, the variable x corresponds to the number of VoIP flows (N_{G711}), the variable y corresponds to merged token bucket rate ($tbrate$) and z corresponds to the merged token bucket size ($tbsize$).

As shown in Figure, the EF load is mapped to four fuzzy sets: *Low*, *Average*, *High* and *Very High*. The letters x_1 and x_{10} correspond to fixed points and define the universe of discourse for the x variable. The AF traffic specification (tbrate and tbsize) is mapped to three fuzzy sets: *Low*, *Average* and *High*. Similarly, the fixed points y_1 and y_7, and z_1 and z_7, define the universe of discourse for y and z variables, respectively. The choice of the number of fuzzy sets associated to each input variable is important. If too many fuzzy sets are employed, there would be too many variables to be defined by the optimization process, raising the risk of converging to a local minimum. If the number of fuzzy sets is too small, it may not be possible to model important non-linearities in the queue behavior. The importance of these choices will be illustrated in the examples discussed in Chapter 6.

Table 5.3 summarizes the 36 rules corresponding to the AF aggregate drop subsystem. Each rule in the table reads as follows: IF $N_{G711}\,(x)$ is *Low* AND $tbrate\,(y)$ is *Low*

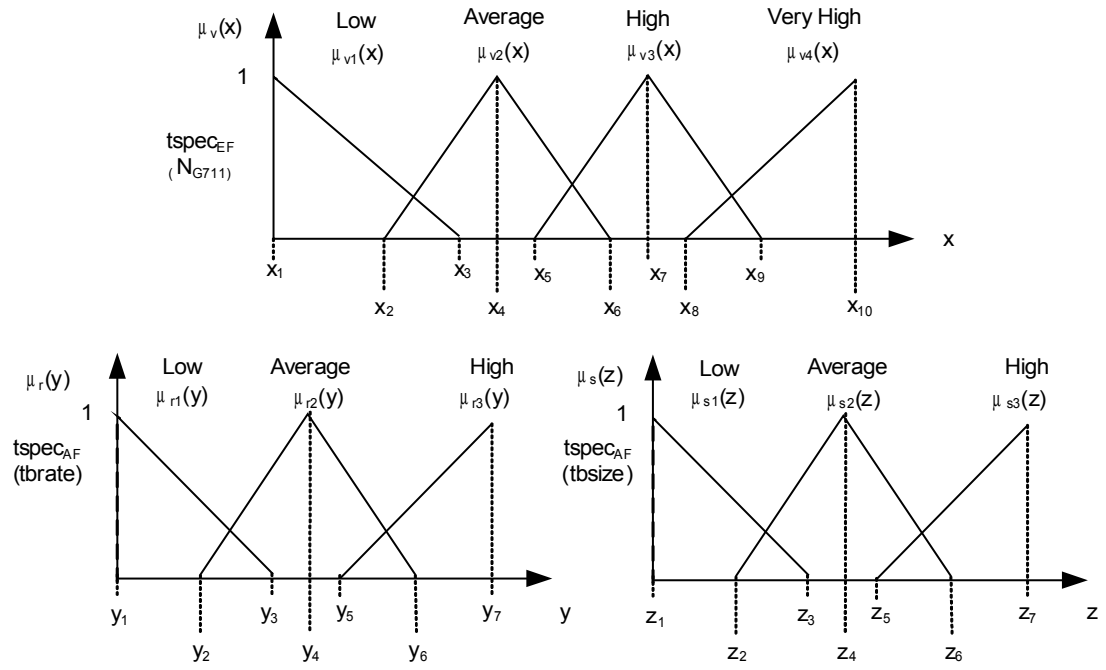

Figure 5.4: Configuration for the AF aggregate drop subsystem

AND $tbsize\,(z)$ is *Low* THEN $fuzzyoutput$ is o_{111}. The symbols o_{xyz} correspond to the fuzzy output variables. Because the membership functions of the input variables are flexible enough to map the queue behavior of the network node, we have defined that the membership functions for all output variables are singletons, which are also input variables for the optimization problem.

As seen in Chapter 2, there are many defuzzification methods discussed in the fuzzy literature. Here, we have selected the COG method(section 2.3.4.2) whose operation is simplified due to the use of singletons as consequents. Thus, given the input variables (x, y, z), representing the EF load and the AF load, the performance prediction is computed by using the centroid method as follows:

$$drop_{AF}^{crisp} = \frac{\sum_{i=1}^{4}\sum_{j=1}^{3}\sum_{k=1}^{3} \mu_{vi}(x) \,.\, \mu_{rj}(y) \,.\, \mu_{sk}(z) \,.\, o_{ijk}}{\sum_{i=1}^{4}\sum_{j=1}^{3}\sum_{k=1}^{3} \mu_{vi}(x) \,.\, \mu_{rj}(y) \,.\, \mu_{sk}(z)} \tag{5.3}$$

Table 5.3: Output membership functions for the AF aggregate drop subsystem

y ⟶	Low			Average			High		
z ⟶ / x	Low	Avg	High	Low	Avg	High	Low	Avg	High
Low	o_{111}	o_{112}	o_{113}	o_{121}	o_{122}	o_{123}	o_{131}	o_{132}	o_{133}
Avg	o_{211}	o_{212}	o_{213}	o_{221}	o_{222}	o_{223}	o_{231}	o_{232}	o_{233}
High	o_{311}	o_{312}	o_{313}	o_{321}	o_{322}	o_{323}	o_{331}	o_{332}	o_{333}
Very High	o_{411}	o_{412}	o_{413}	o_{421}	o_{422}	o_{423}	o_{431}	o_{432}	o_{433}

5.3.3 Formulation of the Optimization Problem

In order to employ the FP algorithm (see Chapter 4) to tune the fuzzy predictor parameters, the following elements must be defined: the cost function, the variables being optimized and the constraints defining "admissible solutions", i.e., the admissible limits for the variables being optimized. The objective of the optimization problem is to minimize the fuzzy logic prediction error, which can be expressed as follows:

$$e(s,v) = \widehat{d}_{AF}^{crisp}(s,v) - d(s) \tag{5.4}$$

where, v: a vector formed by the membership function parameters; s: a vector formed by the input variables describing the traffic conditions; $\widehat{d}_{AF}^{crisp}(s,v)$: the performance estimated by the fuzzy predictor; $d(s)$: the actual performance determined by simulation.

The vector s depends on the input variables considered for the fuzzy predictor. Considering the example of the membership functions for the AF aggregated drop performance subsystem in Figure 5.4, the vector s is composed by the variables x, y and z, representing, respectively, the number of VoIP flows (N_{G711}), the merged token bucket rate ($tbrate$) and the merged token bucket size ($tbsize$).

The me metric defined in 5.5 expresses the quality of a fuzzy prediction. It is defined by two components: the absolute estimation error and a penalization for under-estimating the drop(i.e., the negative part of the error). The penalization weight is controlled by the η parameter. The penalization is necessary because under-provisioning it would have a very negative impact on the QoS guarantees, making the predictor useless. In our study we have adopted $\eta = 10$.

$$me(s,v) = |e(s,v)| + (e(s,v) < 0\,?\,(\eta\,.\,|e(s,v)|) : 0) \tag{5.5}$$

The v vector is formed by the variables to be optimized. It corresponds to the parameters that define the input membership functions and the output singletons. Considering Figure 5.4 and Table 5.3, the v vector is defined as follows:

$$v = [\,x_2 \ldots x_9 \;\; y_2 \ldots y_6 \;\; z_2 \ldots z_6 \;\; o_{111} \ldots o_{433}\,] \tag{5.6}$$

In order to employ 5.5 to formulate an optimization problem, it is necessary to eliminate its dependency on the traffic conditions s. Our strategy to eliminate such dependency is to define two metrics for the overall prediction error: $me_{max}(v)$ and $me_{avg}(v)$, the first corresponds to the maximum error performed by the fuzzy system and the second corresponds to the average error.

The $me_{max}(v)$ is the most important metric, as it defines the maximum error performed by the fuzzy predictor at a specific traffic condition s^*. Because it corresponds to a single point, it can be determined as an optimization process, avoiding an exhaustive search of the load space. The $me_{avg}(v)$ metric, on the other hand, offers no contribution to the fuzzy logic prediction accuracy, but it permits to differentiate solutions with similar maximum errors. The $me_{avg}(v)$ metric is very hard to compute, as it requires scanning the multi-dimensional traffic space. Because its impact is less important to the final result, we replace this metric by an estimate easier to be computed. The estimation $\widehat{me}_{avg}(v)$ computes the average error using just some selected points of the load space.

We define the cost function for the fuzzy system as a combination of $me_{max}(v)$ and $\widehat{me}_{avg}(v)$ metrics as follows:

$$f_c(v) = me_{max}(v) + \varphi\,.\,\widehat{me}_{avg}(v) \tag{5.7}$$

The $\varphi < 1$ parameter controls the weight of the $\widehat{me}_{avg}(v)$ metric. Considering the cost function expressed in 5.7, the problem of defining an optimal fuzzy predictor consists of finding the optimal v^* vector such as:

$$v^* \in V \; and \; f_c(v^*) = inf_{v \in V}\, f_c(v) \tag{5.8}$$

When we deal with optimization problems, we have typically infeasible values that should be discarded in order to avoid wrong convergence results. Normally, optimization

methods allow the specification of constraints to face such issue. However, as discussed in Chapter 4, the FP method does not allow constraints. In this case, we introduce a penalization term, which is added to the final value of the cost function, to generate a worse cost for such points with infeasible values, making them to be discarded by the optimization algorithm. So, the optimization problem under constraints 5.8 must be replaced by another one without constraints by introducing a penalization term $g(v)$ to avoid infeasible solutions, expressed as follows:

$$v^* \in V \; and \; f_c(v^*) = inf_{v \in V} \left(f_c(v) + \delta \, . \, g(v) \right) \tag{5.9}$$

An infeasible solution corresponds to a malformed membership function (e.g., a membership function with one or more discontinuity regions) or negative output singletons. The parameter δ is used to weight the penalization term with respect to $f_c(v)$. It must be large in order to avoid the convergence to an infeasible solution. In our work we have employed $\delta = 10$.

5.3.4 Fuzzy Design Procedure

The fuzzy design procedure consists of determining the v vector by solving the optimization problem defined in 5.9. To solve this problem, the FP method requires computing the cost function $f_c(v)$(5.7) for every new candidate vertex v generated by the algorithm. As said before, the main component of $f_c(v)$ is $me_{max}(v)$, i.e., the maximum prediction error performed by the fuzzy logic defined by v. To determine $me_{max}(v)$, it is necessary to define the maximum prediction error at a specific traffic condition by solving another optimization problem, which can also be solved by the same PF method. In this case, the variables to be optimized correspond to the traffic conditions and the resulting performance, i.e., $s \rightarrow d(s)$. To simplify the explanation that follows, we introduce the following notation:

- MFP (Membership FP): denotes the polyhedron used to determine the v^* vector corresponding to the optimal fuzzy predictor parameters;
- SFP (Simulation FP): denotes the polyhedron used to determine the traffic conditions $s^* \rightarrow d(s^*)$ that correspond to the maximum prediction error for a given candidate solution v generated by the MFP.

Note that the SFP is executed till the convergence for every new candidate solution v generated by the MFP. Every SFP candidate solution s requires to run a simulation in

order to determine the actual performance $d(s)$. Without simplifications, the number of simulations can be estimated as follows:

$$ns = avgCS_{MFP} \,.\, avgCS_{SFP} \,.\, S \tag{5.10}$$

Where, $avgCS_{MFP}$ is the average number of candidate solutions generated by the MFP, $avgCS_{SFP}$ is the average number of candidate solutions generated by the SFP, and S is the number of simulations performed to reduce the seed influence. These numbers depend on the parameters ε_1 and ε_2 (see expressions 4.17 and 4.18 in Chapter 4) used in the convergence criteria expressions. The MFP needs to optimize the v vector, with tens of elements, and the SFP needs to optimize the s vector with only a few elements, which depends on the traffic specification. Therefore, the number of candidate solutions generated by the MFP is higher (hundreds) than the SFP (tens).

Running a simulation is the most expensive procedure of our method. Therefore, some simplifications have been introduced in order to reduce the overall number of simulations performed during the optimization process. The simplification consists of using a set of load conditions to evaluate all candidate solutions generated by MFP, instead of running the SFP for every new candidate. The load conditions test set is denoted as S_{LC}.

By using this method, the solution v^* obtained by the MFP is a local minimum, as it only minimizes the error corresponding to some points of the load condition space. After the MFP convergence, the SFP is executed in order to determine the *true* worst load condition corresponding to v^*. If the load condition obtained by the SFP is not included in the S_{LC} set, it is included in the set and the MFP is repeated. The MFP optimizations are repeated until no worst point could be found by the SFP.

The number of simulations required using this simplification can be estimated by 5.11. The variable $avgCR_{MFP}$ is the average number of MFP convergence repetitions until the overall convergence criteria is achieved. Typically, $avgCS_{MFP} \geq 100 \,.\, avgCR_{MFP}$. Therefore, the reduction in the number of simulations ns is significant.

$$ns = avgCR_{MFP} \,.\, avgCS_{SFP} \,.\, S \tag{5.11}$$

Algorithm 2 presents the dual optimization approach including the simplifications.

2.1 *initialize the MFP repetition index* $i = 0$;
2.2 *initialize the vector* v_1 *considering arbitrary admissible values*;
2.3 *initialize* S_{LC} *with one or more arbitrarily chosen test points (* s_0 *):*
$S_{LC} = \{s_0 \rightarrow d(s_0)\}$;
2.4 *Initializes the MFP polyhedron around* v_1 *as defined in equation 4.9 (Chapter 4)*;
2.5 *Determine the* v^* *vector corresponding to the* i^{th} *iteration* (v_i^*) *by performing the MFP until the convergence*;
2.6 *Perform the SFP and determine the worst point* $s_i \rightarrow d(s_i)$ *for the current solution* v_i^*;
2.7 **if** $s_i \in S_{LC}$ [a] **then**
2.8 *Terminate*;
2.9 **else**
2.10 *Include the new point* $s_i \rightarrow d(s_i)$ *into the* S_{LC} *set*;
2.11 *Set* $v_1 = v_i^*$;
2.12 *Increment the repetition index:* $i = i + 1$;
2.13 *Go to step 2.3*;
2.14 **end**

Algorithm 2: Dual Optimization approach: MFP and SFP

[a] A point is considered included if its norm distance with respect to any other vector is less than 1%.

The following modification in step 2.7 reduces the total number of simulations: the norm distance in step 2.7 is initially set to 10%, when no point is included the norm distance is reduced to 5%, and finally, when no point is included the norm distance is reduced to 1%.

5.4 Building a Time Based Fuzzy Model for MVA approximation for Loss Pr.

5.4.1 Problem Statement

The self-similar traffic model employed in this thesis follows the fractional Brownian Motion (fBm) model proposed by Norros [38](see Section 3.2 in Chapter 3). Besides presenting the traffic model, Norros also develops an equation for estimating the probability of buffer overflow θ of a queue subjected to the fBm traffic. The equation 5.12 is defined in terms of the capacity of the output link "C", the buffer limit "x" and the fBm parameters "H", "a" and "m", already defined in Section 3.2 in Chapter 3.

$$\theta = exp\left(-\frac{(C-m)^{2H} \cdot x^{2-2H}}{2k(H)^2\, a \cdot m} \right) \tag{5.12}$$

where $k(H) = H^H (1-H)^{1-H}$.

This equation defines the fraction of time in which an infinite queue spends above level "x". It can not be used to predict the loss probability. Instead, it defines the probability of a queue within an infinite buffer system to exceed a given buffer size "x". However, Kim and Shroff [87] have proposed the equation 5.13, which is an adaptation of the equation 5.12 in order to address finite queuing systems.

$$\varepsilon = \alpha \,.\, exp\left(-\frac{(C-m)^{2H} \,.\, x^{2-2H}}{2k(H)^2\, a \,.\, m} \right) \tag{5.13}$$

where ε is the loss probability and α is the MVA [1] approximation for loss probability that adapts equation 5.12 for finite buffer queuing systems, which is given by the following equation:

$$\alpha = \frac{1}{m\sqrt{2\pi}} \,.\, \left[exp\,\frac{(C-m)^2}{2\,.\,m\,.\,a} \,.\, \int_C^{\infty} (r-C) \,.\, exp\,(-\frac{(r-m)^2}{2\,.\,m\,.\,a}) \,.\, dr \right] \tag{5.14}$$

The study shown by Kim and Shroff is motivated by the fact that, asymptotically, the loss probability for finite buffer systems and the tail probability curves for infinite buffers are quite similar. Figures 5.5 and 5.6 depict an example of the behavior of the loss probability considering the use of equations 5.13 and 5.14. For each graph we consider the variation of one parameter (x axis) while the others are fixed (indicated on the top). The parameters a and H control the level of burstiness of the traffic. The higher these parameters are, the worst-behaved the traffic is, what is denoted by corresponding higher loss probability values in these Figures. As expected, we can see in Figure 5.6(a) that when the m parameter is close to C, the loss probability increases exponentially towards an instability region. Also, one can see in Figure 5.5(b) that the effect of the Hurst parameter in the loss probability is more significant for higher buffer sizes, obviously because for lower buffer sizes we have higher loss rates regardless the value of H.

Inspite of being a good approximation for the actual behavior of a finite queue fed by a fBm source, by using equation 5.13, we have *asymptotic values* for loss probability, i.e., this equation provides a loss probability value for the long-term. However, suppose that one service provider needs to obtain such value for the short-term. In this context, it is expected that in the short-term [2] there exists a variability for loss probability values. This

[1] Maximum Variance Asymptotic

[2] The concept of long-term and short-term is vague. Here, we consider the short-term a time period in which the system presents a variation of loss probability (among different simulations) higher than a specific limit.

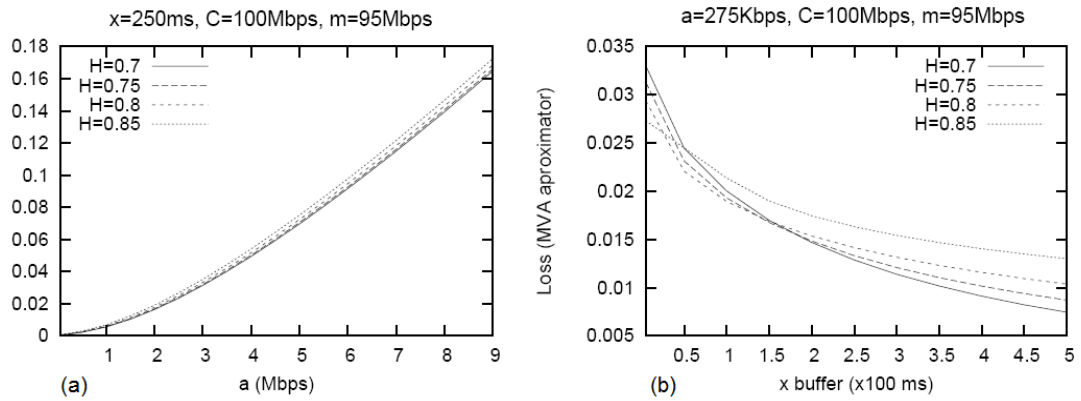

Figure 5.5: Example: Loss probability according to the MVA approximation analytical model. (a) Variance coefficient a (b) Buffer Size x

scenario is illustrated in Figure 5.7, which is based on the analysis of simulation data considering several points with distinct seeds. The straight line represents the crisp value ε obtained from the analytical model. As it can be seen, the accumulated loss probability variability is higher during the "short-term", which is denoted by the large widths of ranges (d_{min}, d_{max}). During this time period, one can not assume the value ε as a precise result for loss probability. On the contrary, when the system is close to the "asymptotic" region, around the time t_N, the variability is lower and the accumulated loss probability values converge to ε.

Taking this statement into consideration, this thesis proposes a time based fuzzy model for the MVA approximation for loss probability, i.e., we want to build a fuzzy model that can be viewed as a "fuzzy adjusting coefficient" for equation 5.13 for the short-term. As it will be shown later, using a similar training strategy presented for the fuzzy predictor, we employ simulations for building this time based fuzzy model, but, on the contrary, the result is a *fuzzy output* that represents this *adjusting coefficient* instead of an equation for crisp values. Thus, we can define our problem as follows:

$$\widehat{\varepsilon} = \widehat{F}(T, H) \,.\, \varepsilon \tag{5.15}$$

The ε is the crisp MVA approximation for loss probability defined in equation 5.13. $\widehat{F}(T, H)$ represents the proposed fuzzy model (whose result is a fuzzy set) for the adjusting coefficient. As seen, it depends on the time T and the Hurst parameter H. As a result, one can obtain a "fuzzy version" of ε, denoted by $\widehat{\varepsilon}$. For the definition of $\widehat{F}$ we could have taken

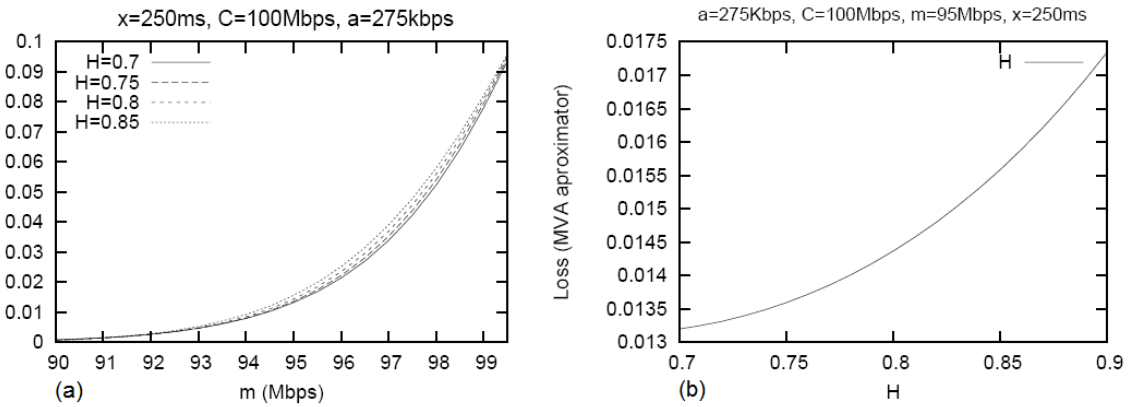

Figure 5.6: Example: Loss probability according to the MVA approximation analytical model. (a) Mean input rate m (b) Hurst parameter H

into consideration the other parameters of 5.13, but we intentionally selected the Hurst parameter, H, which is expected to have a considerable impact on the loss probability, leaving the investigation of which other parameters should be included or not for a future work.

5.4.2 Training Method

As defined in the previous section, here our method is employed for building a fuzzy system capable of capturing the variability of loss probability values observed during the "short-term" period in a queue fed by a fBm traffic source. So, taking the fuzzy coefficient $\widehat{F}(T, H)$, it is possible to infer about the loss probability behavior by obtaining $\widehat{\varepsilon}$. In this first proposal, considering the traffic specification described in terms of fBm parameters, the fuzzy system takes as input the values of the Hurst parameter H and the time instant $t \in T$ and outputs the expected loss probability expressed in terms of a normalized probability distribution, i.e., given a specific pair (T, H), the proposed fuzzy system outputs a normalized probability distribution which represents the loss probability behavior for such pair.

As in the case of the fuzzy predictor, a training approach based on multiple simulations will be employed again for obtaining the optimal $\widehat{F}(T, H)$. The input of the training problem is the universe of discourse of T and H. The training method is now formulated as one optimization problem, as defined before, the Membership Optimization (MFP), as shown in Figure 5.8. Initially, all simulation points are generated considering the universe of discourse

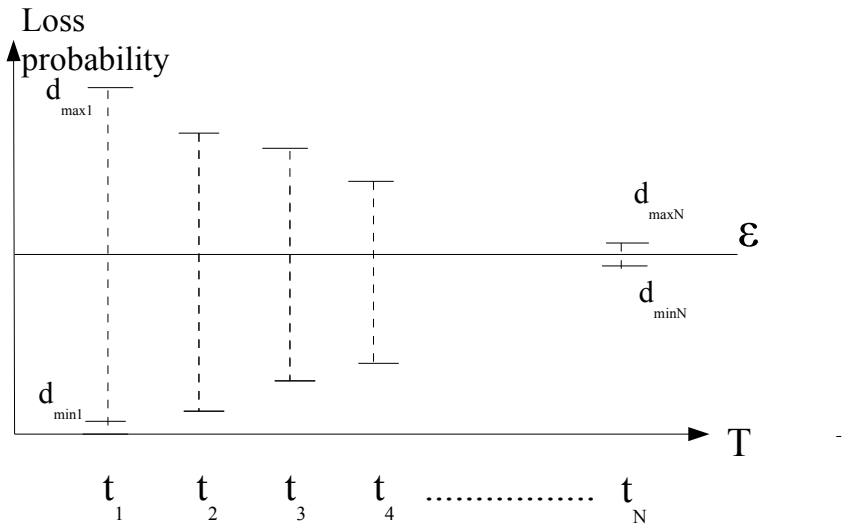

Figure 5.7: Example: Accumulated loss probability variability illustration

of T and H. Then, the normalized probability distribution from fuzzy inference is obtained for each point (T,H) which is compared with the corresponding histogram computed from simulations in order to calculate the overall error.

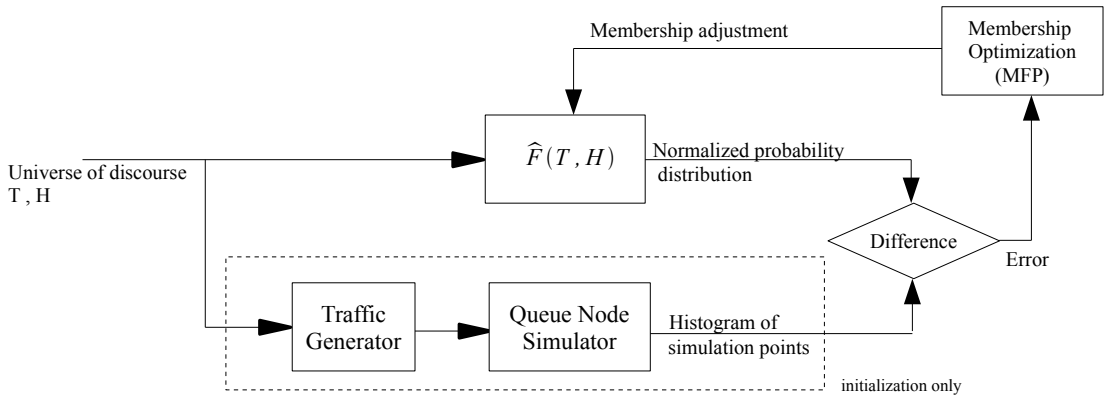

Figure 5.8: Optimization Training Method for the optimal $\widehat{F}$

5.4.3 The Fuzzy Coefficient $\widehat{F}$

The Fuzzy Coefficient $\widehat{F}$ represents an output fuzzy set that is generated by the fuzzy logic system for the MVA approximation for Loss Probability. As it will be seen later, this output

fuzzy set is used to obtain a normalized probability distribution expressed in terms of loss probability values, which represents the output of the fuzzy system. The proposed FLS is illustrated in Figure 5.9. As shown, the FLS receives as input variables a specific time instant $t \in T$ and the value of the Hurst parameter H. It outputs a *normalized* fuzzy set for the Loss Probability for this pair (t, H). In this case, *normalized* means that loss probability values, $d_1, d_2, d_3, ...$, are expressed as a ratio of ε, as denoted in equation 5.15. It is important to note the advantage of having an output fuzzy set as a result. Considering a future utilization of such results, the choice of the defuzzification method in order to have a crisp value for loss probability is an option of the specific application scenario. As shown in Chapter 2 section 2.3.4, there are several defuzzification methods, which differ in terms of complexity and accuracy.

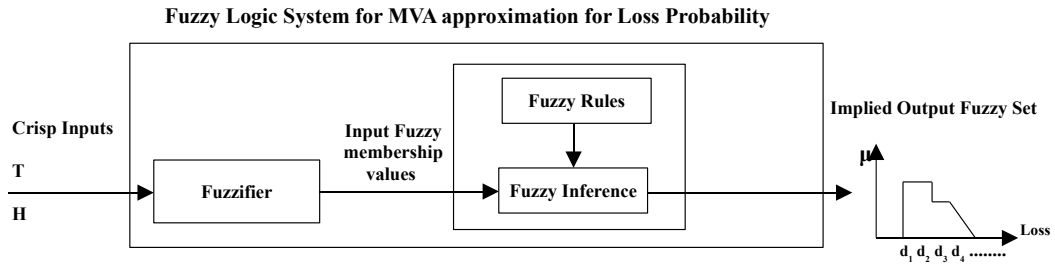

Figure 5.9: FLS for the MVA approximation for Loss Probability

Our strategy for obtaining $\widehat{F}$ is based on the specification of a normalized probability distribution as employed by the BADD method (see section 2.3.4.2). In [31, 88], Filev and Yager show that the difference between two common defuzzification methods, COA and MOM (see Section 2.3.4 of Chapter 2), lies in the procedure used to obtain the probability distributions from the fuzzy sets. They also show that with the help of the BADD transformation it is possible to obtain these more specific methods. In order to understand this concept, consider the following example: Suppose a fuzzy set Ω and a discrete variable x whose universe of discourse is $\{1, 2, 3, 4, 5\}$, where $\Omega(x)$ denotes the degree of membership of x in Ω. Also, consider the following definition of $\Omega(x)$:

$$\Omega(x) = \{\frac{0.2}{1}, \frac{0.4}{2}, \frac{0}{3}, \frac{0}{4}, \frac{1}{5}\}$$

we have, for instance, $\Omega(1) = 0.2$.

The normalized probability distribution of $\Omega(x)$ can be expressed as follows [88]:

$$P(x_i) = \frac{\Omega(x_i)}{\sum_i \Omega(x_i)} \tag{5.16}$$

which result is:
$P(1) = 0.125$, $P(2) = 0.25$, $P(3) = P(4) = 0$, $P(5) = 0.625$

Based on such concept for generating normalized probability distributions using fuzzy sets, our proposed FLS employs *ranges* expressed in terms of loss probability values for modeling the output variable. We obtain a specific normalized probability distribution for each pair (t, H), where the weight of each range is determined by fuzzy inference for this pair. Thus, for a given time $t \in T$ and a Hurst parameter H it is possible to obtain a normalized probability distribution, which allows one to infer about the loss probability behavior for the pair (t, H). In order to illustrate our approach, consider the example [3] shown in Figure 5.10. The input variable T (5.10(a)) has three membership functions: $range_1$, $range_2$ and $range_3$, where each one corresponds to a range within the universe of discourse of the output variable loss probability d (5.10(c)). These input membership functions are used to determine the weight of the corresponding loss probability ranges for each point t. The input variable H has only one membership function, the singleton H_1(5.10(b)). It is important to note that the loss probability ranges are defined specifically for such singleton, as it is illustrated in the indexes of the values d_{i,H_1}.

The resulting output fuzzy set for the pair (t_1, H_1) is depicted in Figure 5.10(d). As can be seen in Figure 5.10(a), t_1 has the following degrees of membership for the considered membership functions: $range_1(t_1) = 1$, $range_2(t_1) = 0.3$ and $range_3(t_1) = 0$. According to the presented approach, the resulting normalized output fuzzy set has the following values for each range: $p_1 = 0.77$, $p_2 = 0.23$ and $p_3 = 0$, which means that at time instant t_1 there is a 0.77 probability for a loss probability value to be within (d_1, d_2), whereas a 0.23 value to be within (d_2, d_3). Also, there is a zero probability assigned to $range_3$.

Figure 5.11 illustrates a possible definition for the input and output membership functions for the $\widehat{F}$ fuzzy logic system. The input variable H is modeled using singletons(5.11(b)). As shown in Figure 5.11(a), the input variable T is mapped to five fuzzy

[3] This example is a simplified version of the proposed modeling in order to illustrate the fuzzy inference process and the corresponding resulting output fuzzy set.

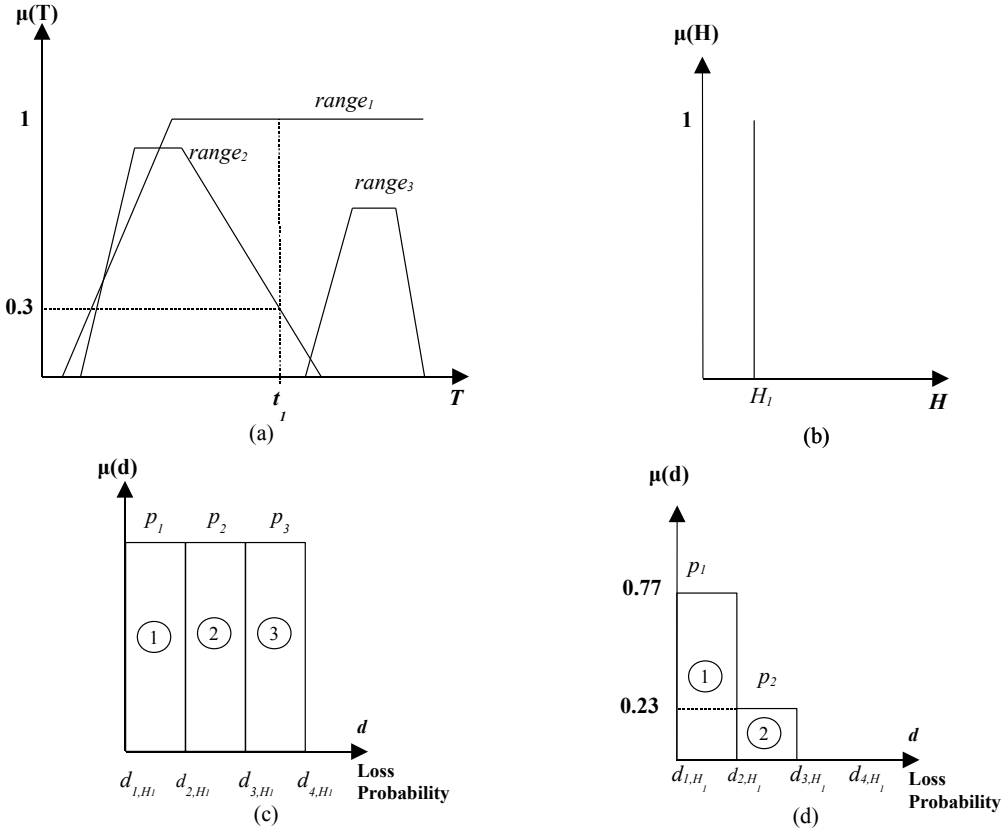

Figure 5.10: Example: Illustration of the fuzzy inference process for generating the normalized probability distribution for loss probability. Input Membership Functions: (a) T (b) H. Output Membership Functions: (c) Loss Probability Ranges (d) Resulting Implied Output Fuzzy Set.

sets: $range_1$, $range_2$, $range_3$, $range_4$ and $range_5$. The position, width and height of each one are merely illustrative, since the values will be optimized. The membership function $range_3$ is modeled with the half-trapezoidal shape, whereas the others have the trapezoidal shape. This is because the ranges are defined based on the dispersion of loss probability values along the time T. We define $range_3$ as the *asymptotic range*, i.e., it is expected that all loss probability values will be within this range when the system achieves the stability region, which includes, obviously, the crisp value ε provided by the analytical model MVA for loss probability. Around this range, $range_2$ and $range_4$ represent two intermediate ranges for regions that have an average degree of dispersion and, finally, the last two, $range_1$ and

$range_5$, for the cases of high degree of dispersion. The membership function $range_3$ has a different shape because we naturally consider that this range is the most expected to appear, i.e., this range is always active most of the time, while the other ranges gradually lose their importance in terms of the number of expected values.

The output variable d (loss probability) is specified in terms of loss probability *ranges* (5.11(c)), as discussed previously. For this first study, we have five ranges: $\{[d_{1,i}, d_{2,i}), [d_{2,i}, d_{3,i}), [d_{3,i}, d_{4,i}), [d_{4,i}, d_{5,i}), [d_{5,i}, d_{6,i}]\}$. The variable i represents an index for the number of H values. Here, we assume four different H values, so we have $i = \{1, 2, 3, 4\}$. For each loss probability range we have the terms $\{p_{1,i}, p_{2,i}, p_{3,i}, p_{4,i}, p_{5,i}\}$, which represent the output values generated by fuzzy inference. After having such values, we apply equation 5.16 in order to obtain a normalized probability distribution to calculate the correct weight of each range. Table 5.4 presents the 20 rules corresponding to the $\widehat{F}$ fuzzy logic system.

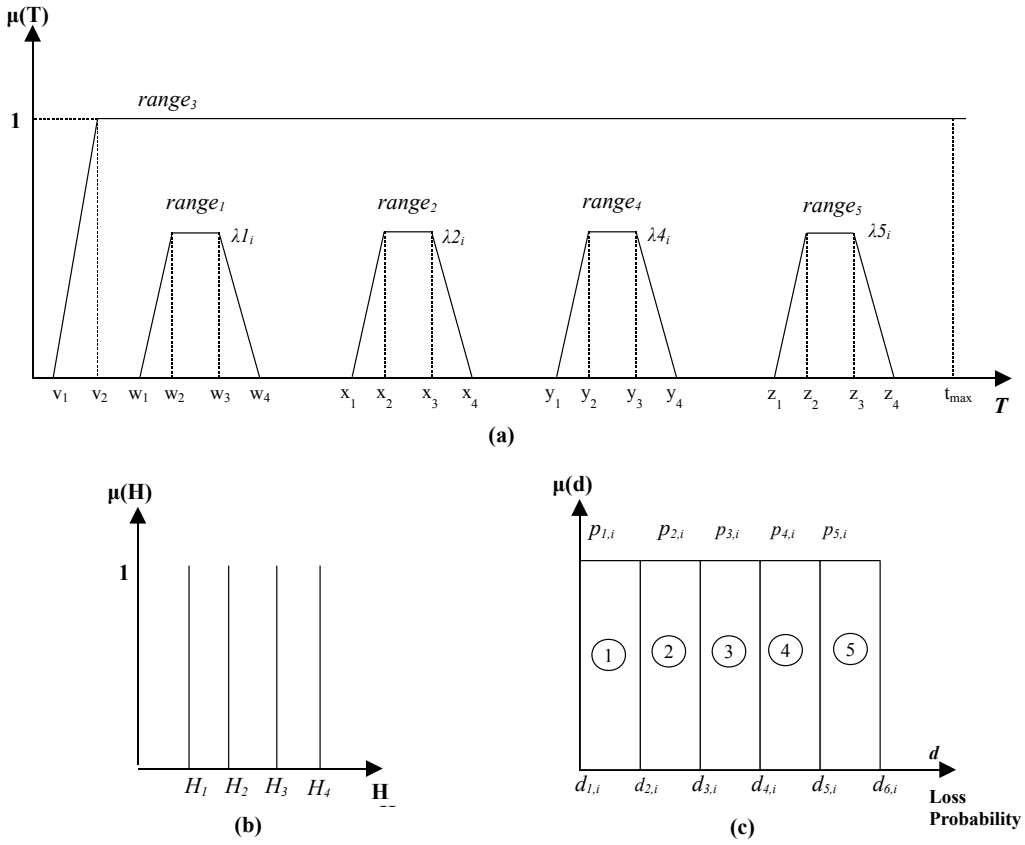

Figure 5.11: Input and Output Membership Functions for the $\widehat{F}$ coefficient

Table 5.4: Rule Set for the $\widehat{F}$ fuzzy logic system

Range $\longrightarrow$ H	$Range_1$	$Range_2$	$Range_3$	$Range_4$	$Range_5$
H_1	$(d_{1,1}, d_{2,1})$	$(d_{2,1}, d_{3,1})$	$(d_{3,1}, d_{4,1})$	$(d_{4,1}, d_{5,1})$	$(d_{5,1}, d_{6,1})$
H_2	$(d_{1,2}, d_{2,2})$	$(d_{2,2}, d_{3,2})$	$(d_{3,2}, d_{4,2})$	$(d_{4,2}, d_{5,2})$	$(d_{5,2}, d_{6,2})$
H_3	$(d_{1,3}, d_{2,3})$	$(d_{2,3}, d_{3,3})$	$(d_{3,3}, d_{4,3})$	$(d_{4,3}, d_{5,3})$	$(d_{5,3}, d_{6,3})$
H_4	$(d_{1,4}, d_{2,4})$	$(d_{2,4}, d_{3,4})$	$(d_{3,4}, d_{4,4})$	$(d_{4,4}, d_{5,4})$	$(d_{5,4}, d_{6,4})$

A key issue of this proposal is the flexibility given for the optimization algorithm to define the membership functions. In this case, $\forall t \in T$, the optimization algorithm must find an optimal configuration for membership functions in order to provide the best results in terms of normalized probability distributions of loss probability d, according to expression 5.15. Initially, we can not infer about the degree of membership values of t in the respective ranges, i.e., given a $t \in T$ we want to know the corresponding distribution of d within the ranges. Each t point can have loss probability values distributed in different ranges. This is the main issue of the optimization problem. Hence, the membership function points of the input variable T, $\{v_1, v_2, w_1, w_2, w_3, w_4, x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4, z_1, z_2, z_3, z_4\}$ can assume any value within the universe of discourse of T [4]. Figure 5.12 shows an example of one possible result of membership functions of T after optimization phase. As shown, we have a high degree of overlapping, and different time points for each one. For example, $range_1$ starts at t_2 and finishes at t_5. As shown in this Figure, after t_8 all points have a 100% degree of membership in $range_3$, i.e., $\forall t > t_8$ we have $range_3(t) = 1$ and 0 for the other ranges. This is the case of t_b. This means that, according to the fuzzy model, it is expected that $\forall t > t_8$, all loss probability values are within the $range_3$. For t_a, we have $range_3(t_a) = 1$, $range_4(t_a) = 0.65$ and $range_5(t_a) = 0.18$, and 0 for the other ranges. Applying equation 5.16 in order to obtain the corresponding normalized probability distribution, it is expected the following distribution of loss probability among ranges for t_a according to the fuzzy model: $range_3 = 54.64\%$, $range_4 = 35.52\%$ and $range_5 = 9.84\%$. Thus, one can infer about the queue behavior in terms of loss probability.

Another important issue related with our strategy is the adjusting of the height of

[4] Obviously, we also consider the restrictions involving the membership shapes. For example, for $range_1$ we have $w_1 < w_2 < w_3 < w_4$.

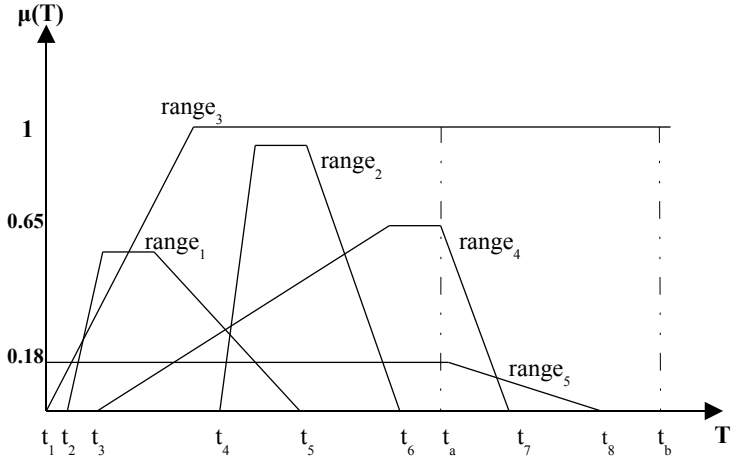

Figure 5.12: Example: Illustration of membership functions of T after optimization phase

the membership functions $range_1$, $range_2$, $range_4$, and $range_5$. Despite the possibility of optimizing the points in the T axis, allowing the FP algorithm to change the height of membership functions permits a fine-grain adjustment, which contributes to the convergence process. In this case, only the membership function $range_3$ is modeled as a normal fuzzy set, while the others can be seen as subnormal fuzzy sets, because they may have their maximum degree of membership lower than 1. Each one has multiplier coefficients, which we define as $\{\lambda 1_i, \lambda 2_i, \lambda 4_i, \lambda 5_i\}$. The coefficient λR_i, $0 \leq \lambda R_i \leq 1$, represents a multiplier for the corresponding membership functions. R denotes the number of the range and i is the index that represents the corresponding H value. For example, for $range_1$ and H_1, we have $(\lambda 1_1.range_1(t))$ to express the degree of membership of any t in the fuzzy set $range_1$ for $0 \leq t \leq t_{max}$, where the latter is the universe of discourse of $t \in T$. As it can be noted, there is no $\lambda 3_i$ for $range_3$, because of the strategy employed for defining the ranges, as discussed above.

5.4.4 Formulation of the Optimization Problem

The objective function of the optimization problem is to minimize the error between the histogram of loss probability values obtained from simulations and the corresponding normalized probability distribution obtained from the fuzzy inference for $\forall t \in T$, both considering the same T ranges $range_k$. In order to formalize our proposal consider the following definitions: Let $\triangle t$ to be the time step for t_j within the universe of discourse of T, $0 \leq t \leq t_{max}$.

In this case, we have $\{t_j, t_{j+1}, ..., t_N\}$ where, $t_0 = 0$, $t_N = t_{max}$ and $t_j = t_{j-1} + \triangle t$ for $1 \leq j \leq N$. Also, let M and R denote the number of Hurst parameter values and the number of ranges, respectively. So, the objective of the optimization problem is to minimize the following expression, which also represents the cost function:

$$f_c(v) = \sum_{p=1}^{M} \sum_{k=1}^{R} \sum_{j=1}^{N} |\Omega_k^p(t_j, v) - \Psi_k^p(t_j)| \tag{5.17}$$

where: v is a vector formed by the membership function parameters; $\Psi_k^p(t_j)$ corresponds to a normalized value obtained from the histogram given by simulation point t_j for range k and H value p. $\Omega_k^p(t_j, v)$ corresponds to the output value obtained from the normalized probability distribution given by fuzzy inference using $\widehat{F}$ for the same point t_j, range k and H value p, and membership function parameters v. The v vector is formed by the variables to be optimized. It corresponds to the parameters that define the input membership functions of T. Considering Figure 5.11, the v vector is defined as follows:

$$v = [\, v_1\, v_2 \;\; w_1 \ldots w_4 \;\; x_1 \ldots x_4 \quad y_1 \ldots y_4 \;\; z_1 \ldots z_4 \;\; \lambda 1_p\, \lambda 2_p\, \lambda 4_p\, \lambda 5_p \;\;] \tag{5.18}$$

where p was defined in equation 5.17.

Note that the output membership parameters in Figure 5.11, i.e., loss probability range values, as well as H membership functions parameters are not optimized; they are only employed as inputs to optimize v. Also, observe that the number of λ multipliers depends on the number of H values of the optimization problem.

The problem of defining an optimal $\widehat{F}$ consists of finding the optimal v^* vector such as:

$$v^* \in V \; and \; f_c(v^*) = inf_{v \in V} \, f_c(v) \tag{5.19}$$

Due to the FP algorithm, as in the case of the fuzzy predictor, 5.19 must be replaced by another one without constraints by introducing a penalization term $g(v)$ to avoid infeasible solutions, expressed as follows:

$$v^* \in V \; and \; f_c(v^*) = inf_{v \in V} \, (f_c(v) + \delta \, . \, g(v)) \tag{5.20}$$

Again, the parameter δ is used to weigh the penalization term with respect to $f_c(v)$. It must be large in order to avoid the convergence to an infeasible solution. In this work we have employed $\delta = 10$.

5.4.5 Fuzzy Design Procedure

The fuzzy design procedure consists of determining the v vector by solving the optimization problem defined in 5.20. To solve this problem, the FP method requires computing the cost function $f_c(v)$(5.17) for every new candidate vertex v generated by the algorithm. Different from the case of the fuzzy predictor, we have now only one optimization problem, which will be performed by the MFP ($Membership\ FP$), which denotes, as before, the polyhedron used to determine the v^* vector corresponding to the optimal $\widehat{F}$. The number of simulations can be expressed as follows:

$$ns = S.M \tag{5.21}$$

where, S is the number of simulations with distinct seeds [5] and M is the number of H values. Fortunately, the simulations cited in equation 5.21 are not repeated according to the number of candidate solutions generated by the MFP, as it occurs in the case of the fuzzy predictor, because here we don't have the SFP procedure. Thus, before starting the MFP procedure, we obtain these data points, denoted as $D_{j,p,g}$ [6], where $g = 1..S$ represents the index for the number of simulations performed with distinct seeds. For each pair (p, g), we compute the accumulated loss rate at each time t_j along an unique simulation run whose duration is the universe of discourse of T.

Algorithm 3 summarizes the optimization procedure for obtaining the optimal $\widehat{F}$.

3.1 *initialize the vector* v_1 *considering arbitrary admissible values*;
3.2 *initialize the vector* $D_{j,p,g}$ *with simulation data points for* $1 \leq j \leq N$, $1 \leq p \leq M$ *and* $1 \leq g \leq S$ *where* N, M *and* S *are defined as the number of* t *points, the number of* H *values and the number of simulations performed, respectively*;
3.3 *Initializes the MFP polyhedron around* v_1 *as defined in equation 4.9 (Chapter 4)*;
3.4 *Determine the* v^* *vector by performing the MFP until the convergence*;

Algorithm 3: Optimization approach for obtaining the optimal $\widehat{F}$

5.5 Conclusion

In this Chapter, we presented the proposal of a new method for modeling queue behavior of network nodes aiming to provide a methodology for building the corresponding performance

[5] In this case, the proposed fuzzy model will capture the loss probability variability using S distinct seeds
[6] j and p were defined in equation 5.17

models. As we saw, one of the contributions of this method is to take into account many complex issues faced in typical network deployment scenarios, where it is typically difficult to develop an analytical model. Based on a general off-line procedure, the proposed method combines non-linear programming (NLP) and simulation to build a fuzzy logic based model capable of determining the performance of a network node. Also, it introduces a training methodology that allows the utilization of any type of performance metric. In the literature, we can find two main research domains on fuzzy logic related with our work. In the first domain, the focus is typically on extending existing queue models to provide performance measures expressed by membership functions, whereas the other fuzzy models are used to building active network traffic controllers.

Taking into account the approach employed for modeling the consequents of fuzzy rules and the corresponding application scenarios, we divided the presentation of our method into two different parts. In the first, we proposed a fuzzy predictor which is based on a dual off-line optimization training method. Such training approach is performed by solving two simultaneous flexible polyhedron optimization problems. We showed that the proposed fuzzy predictor can be employed to determine optimal values that could be used for the configuration of network nodes, with important applications on traffic engineering and capacity planning. For example, we showed an application scenario considering a multi-queue node where it is necessary to determine how much traffic could be accepted without compromising the performance objective of the service classes assigned to the queue. In the second part, we presented the proposal of a time based fuzzy model for the MVA approximation for loss probability analytical model. As we saw, such analytical model is a good approximation for finite buffers by providing an asymptotic value for loss probability. In this case, we presented a fuzzy logic model that could be viewed as a fuzzy adjustment coefficient in order to capture the loss probability variability observed before the system achieves the stability region. We proposed a training strategy based on normalized probability distributions obtained from fuzzy inference and histograms calculated from self-similar traffic simulations in order to solve a flexible polyhedron optimization problem for the proposed time based fuzzy model.

Chapter 6

Evaluation

6.1 Introduction

As shown in Chapter 5, the method proposed in this study combines non-linear programming and simulation to build a fuzzy based model of a network node. In this Chapter, we present the validation of the proposed method considering the implementation of application scenarios according to both strategies presented in Chapter 5. Section 6.2 illustrates how the method described can be employed to define a fuzzy-based performance model for a multi-queue node considering two subsystems, AF Drop Subsystem and EF Delay Subsystem. Section 6.3 shows the implementation of the proposed fuzzy model for the MVA approximation for loss probability analytical model.

6.2 Validation of the Fuzzy Predictor

Figure 6.1 illustrates the sample scenario for the Fuzzy Predictor evaluation. This scenario is employed for the evaluation of two different subsystems. We consider the performance modeling of a DiffServ node composed by three queues, named EF, AF and DF. The EF queue receives only VoIP traffic, and the AF and DF queues receive self-similar traffic. The AF queue receives a portion of these self-similar traffic which conforms to a token-bucket filter conditioner. The DF queue receives the traffic that exceeds the token-bucket limits. In this scenario we are interested in evaluating the EF queue performance expressed in terms of the maximum per-flow delay and the AF queue performance expressed in terms of the aggregated drop. Training the fuzzy predictor for each of the metrics corresponds to an

independent problem, and distinct membership functions are employed.

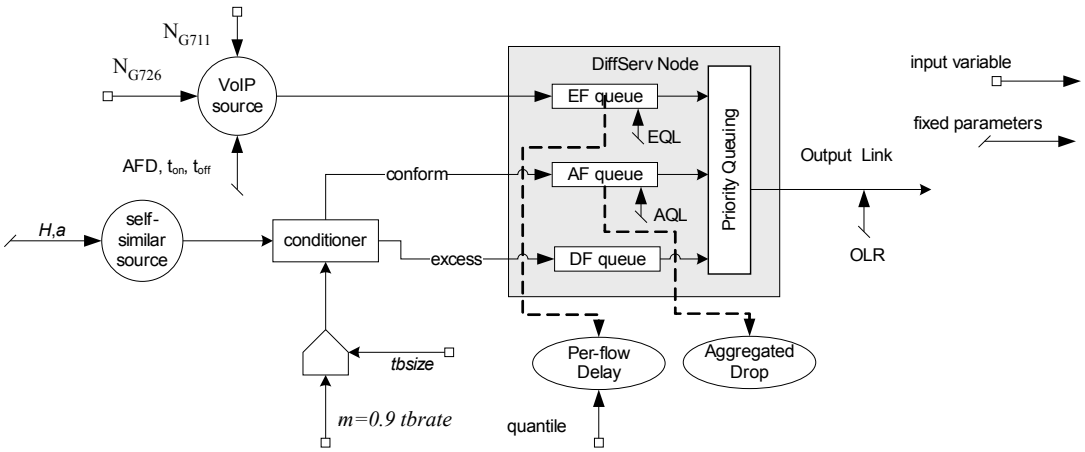

Figure 6.1: Evaluation scenario for the Fuzzy Predictor

6.2.1 AF Drop Subsystem

In this first example, we are interested in evaluating how much traffic can be accepted by a node without leading to an excessive drop level in the AF queue. In this scenario, we consider only one type of codec ($G711$) assigned to the EF queue and a token bucket conditioned traffic assigned to the AF queue. As the EF queue is a high priority queue, the drop level experienced by the AF queue is strongly influenced by both, the VoIP traffic level and the AF traffic level. The drop is represented in terms of the ratio between the packets accepted in the AF queue and the packets dropped by exceeding the 250 ms limit. Because the training method is based on simulation, the drop level is computed considering several simulations with distinct seeds (in this example, five simulations). It is in fact calculated by considering the maximum drop with a confidence interval of 97.5%.

Because the token bucket filter permits the conditioning of traffic with different levels of burstiness, this fuzzy predictor would be quite generic, being capable of representing a wide range of applications. As explained in the Chapter 3, the self-similar source is defined by three parameters: m, a and H. The a and H parameters are usually defined by observing the traffic behavior in a real network. In this work, we have adopted $a = 275\,kbit.sec$ and $H = 0.76$, as used by Norros in the sample simulation [38]. In order to eliminate the dependence of the m parameter during the training process of the fuzzy predictor, we have (arbitrarily) fixed the following relation: $m = 0.9\,tbrate$. Note that assuming m is much

greater than the rate can lead to a CBR traffic without burst periods. Choosing m when it is much smaller than the rate will not properly represent the amount of traffic defined by the token-bucket. Indeed, this parameter is uncontrollable. It depends on the user behavior.

The fuzzy predictor will be trained with the following input variables:

- Number of G711 VoIP flows (voip): universe of discourse (0 to 3000);
- Token bucket rate (tbrate): universe of discourse (0 to 1) (Normalized by the output link rate);
- Token bucket size (tbsize): universe of discourse (0 to 500).

Additionally, this scenario considers the following fixed parameters:

- Output link rate (OLR): 100 Mbps
- Maximum AF queue-length (AQL): 250 ms
- Maximum EF queue-length (EQL): 50 ms
- Average Flow Duration (AFD): 210 s
- VAD effect: ton = 0.4s and toff = 0.6 s
- G711 VoIP flow [43]: PDU Length=200 bytes, Packet rate=50 packets/s.

Each G711 VoIP flow is implemented according to [43].

Figure 6.2 shows the convergence results obtained for the membership functions representing the input variables.

The graph representing the relationship between input variables, and the corresponding output is a four dimensional graph, i.e., {tbsize X tbrate X voip} $\rightarrow$ drop. In order to illustrate the fuzzy prediction results, we have selected some three-dimensional cuts of the prediction space, as illustrated in Figures 6.3 and 6.4. Figure 6.3 illustrates the effect of the token bucket rate and number of VoIP flows for two distinct token bucket sizes: 500 MTU and 0 MTU. As expected, a larger token bucket size leads to an increased drop level, i.e., it is possible to accept more traffic when the token bucket size is smaller.

Figure 6.4 illustrates the effect of the token bucket parameters for two distinct levels of VoIP load: 1000 and 2000 flows. Again, in both figures, the effect of the token bucket size is very clear (please note the distinct ranges in the drop axis). It can also be observed that the number of fuzzy sets for the input variable *token bucket size* should be increased, as the effect of this input variable is represented by rough steps, instead of being smooth.

During the training process, 149 test points were generated. These test points correspond to the maximum prediction errors determined by the SFP algorithm during the

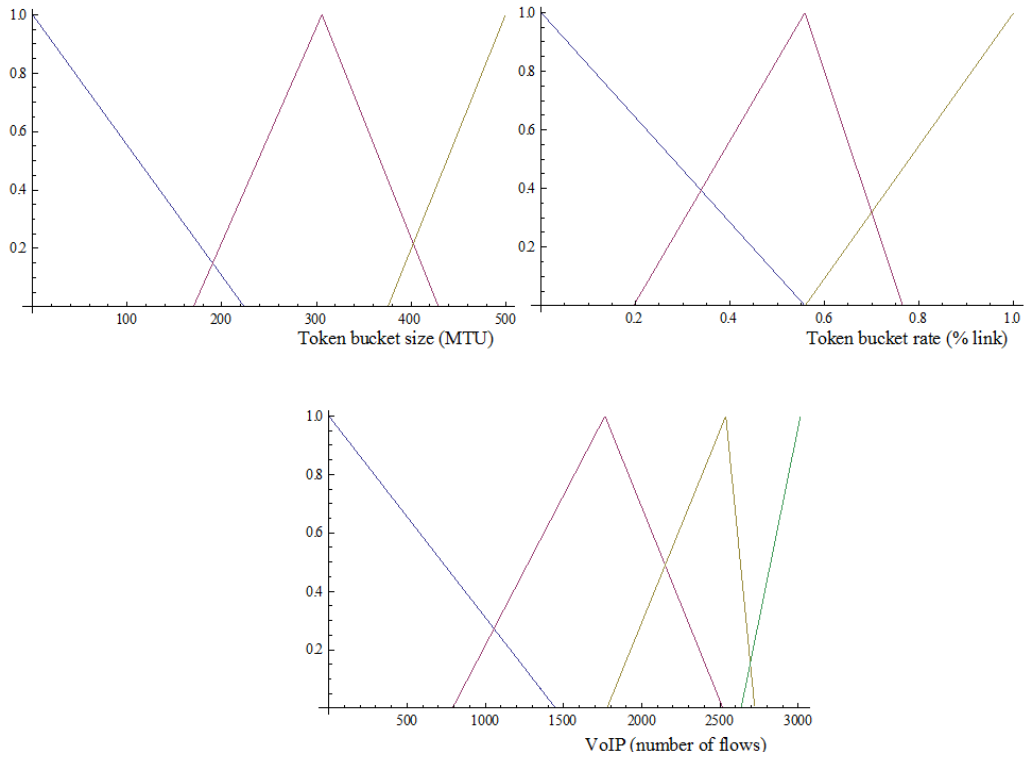

Figure 6.2: AF drop subsystem input membership functions

training process. The Figure 6.5 (left side) illustrates the histogram representing the error prediction distribution for the test points. One thing to note here is that the maximum prediction error was about 20%, and most of the test points presented a error inferior to 2%. It is important to observe that the fuzzy predictor was trained to never present an under-estimate, i.e., the traffic engineer can expect that the drop level be inferior to the prediction with a confidence level of 97.5%. Considering the traffic variability among successive simulations (captured by using distinct seeds in the traffic modeling), to achieve a zero error is virtually impossible. However, it is possible to improve the results, by choosing a more flexible set of membership functions. The right side of Figure 6.5 illustrates the distribution of the test points in the solution space.

6.2.2 EF Delay Subsystem

In this second example, we are interested in evaluating how much traffic can be accepted by a node without leading to an excessive delay in the EF queue. We consider two types of

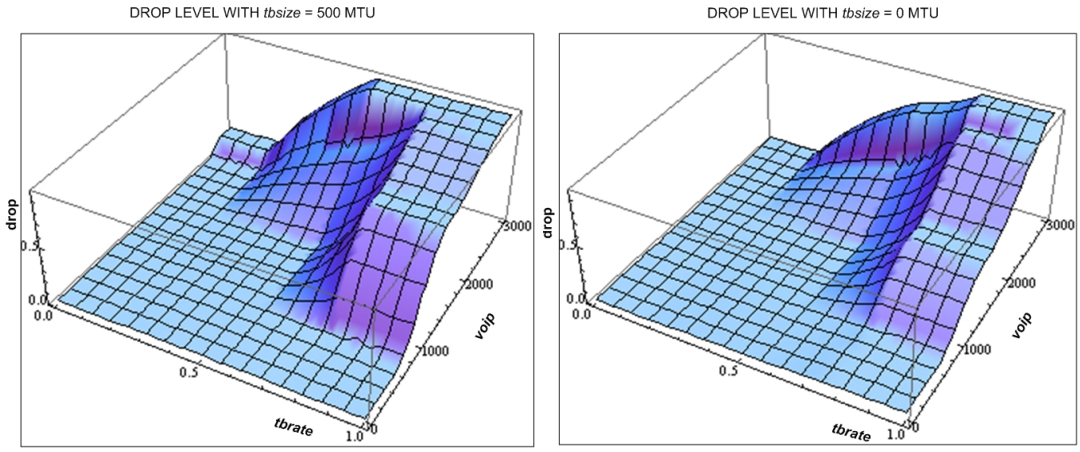

Figure 6.3: Influence of the token bucket rate and number of VoIP flows on the AF drop

codecs (G711 and G726) assigned to the EF queue. Because the EF queue is served with strict priority, there is no significant influence of the AF traffic on the EF delay. In fact, if the AF influence is ignored during the training process, the maximum prediction error introduced is only 0.12 ms (for an output link with a rate of 100 Mbps and AF packets limited by a MTU of 1500 bytes). Therefore, the AF traffic is not considered in this example. As discussed in Chapter 3, the delay experienced by individual VoIP flows can diverge with respect to the delay experienced by the aggregated traffic. In order to permit the traffic engineer to define how much traffic can be accepted by a node, the fuzzy predictor will output the maximum per-flow delay experienced by any VoIP flow in the EF aggregate. In most network agreements, the per-flow delay is computed for a quantile of packets in a flow. The higher the quantile, the closer the maximum per-flow delay will approach the maximum aggregated delay.

Considering this scenario, the fuzzy predictor will be trained with the following input variables:

- Average EF load: universe of discourse (0.9 to 1);
- Proportion of VoIP flows using G711 codec: universe of discourse (0 to 1);
- Per-flow delay quantile: (0.97 to 1).

The *Average EF load* range was selected in order to make evident the region in which the resulting performance is useful. As it will be shown later, for EF load lower than 0.9, the EF delay in terms of per-flow requirements is negligible, making no sense to optimize membership functions for representing a region where the values could be easily estimated.

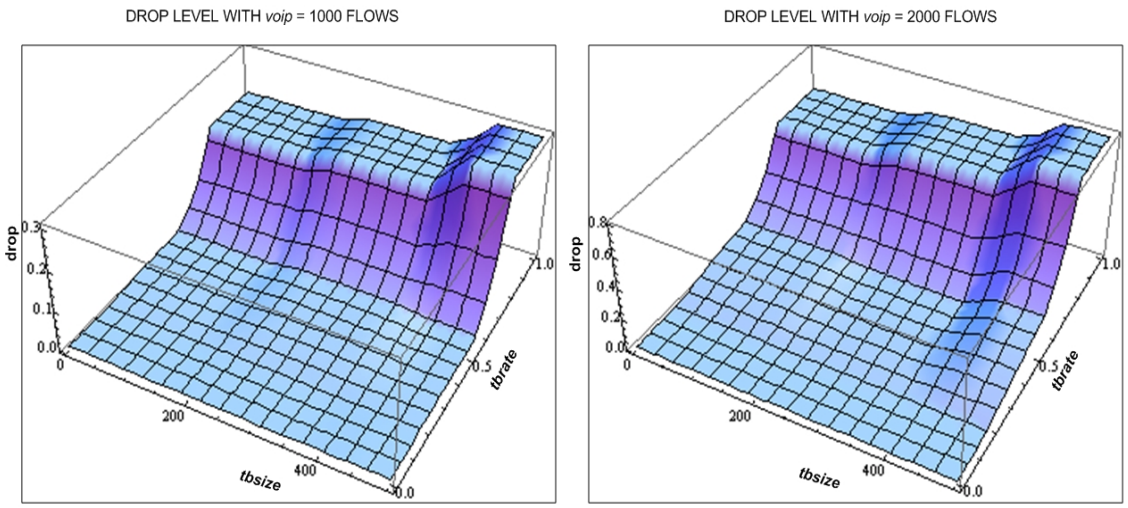

Figure 6.4: Influence of the token bucket rate and token bucket size on the AF drop

Because we have a VoIP aggregate formed by flows of two codec types, the EF load represents the aggregate load considering VoIP flows from both codecs. The number of flows of each codec is captured by the performance model using the metric *G711 proportion*, which is equal to "1" when this aggregate is formed only by G711 flows, "0" is formed only by G726 flows and proportional values for the other cases.

Additionally, this scenario considers the following fixed parameters:

- Output link rate (OLR): 100 Mbps;
- Maximum EF queue-length (EQL): 50 ms;
- Average Flow Duration (AFD): 210s;
- VAD effect: ton = 0.4s and toff = 0.6s.
- G711 VoIP flow [43]: PDU Length=200 bytes, Packet rate=50 packets/s;
- G726 VoIP flow [89]: PDU Length=120 bytes, Packet rate=50 packets/s.

Note that we have not employed the number of flows (N_{G711} and N_{G726}) as input variables, because it could lead to situations where the link capacity is largely overloaded (i.e., when the number of VoIP flows using both codecs is simultaneously large). Instead, we have used two artificial variables to represent the VoIP traffic load. The first input variable, EF load, is calculated considering the average bandwidth occupied by the aggregate VoIP stream. The second variable, G711 proportion, represents the proportion of flows using the G711 codec (i.e., when the proportion is 1, all flows are G711, and when the proportion

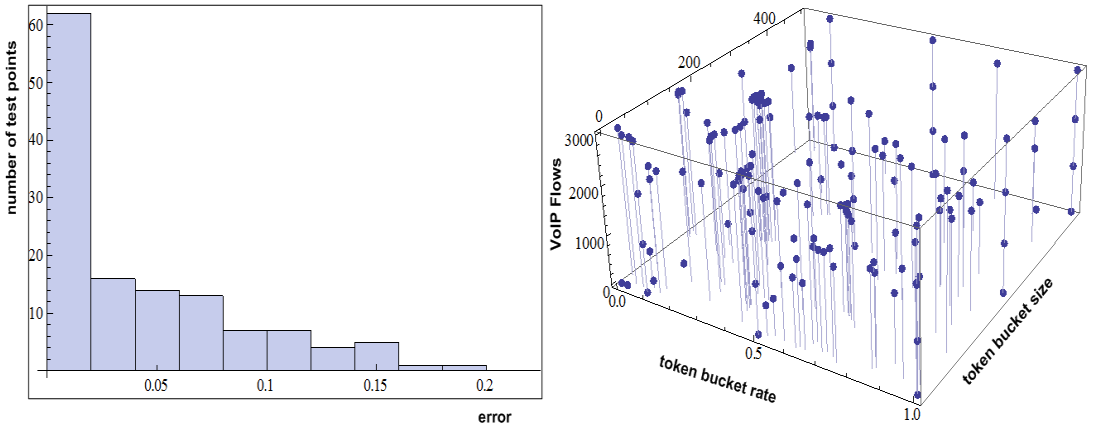

Figure 6.5: Error prediction histogram for the AF drop test points

is 0, all flows are G726). The third variable, Per-flow quantile, defines the quantile to be considered when observing the per-flow delay. Figure 6.6 shows the convergence results obtained for the membership functions that represent the input variables.

Again, the graph representing the relationship between input variables, and the corresponding output is a four dimensional graph, i.e., {EF load X G711 proportion X Per-flow quantile} $\rightarrow$ delay. In order to illustrate the fuzzy prediction results, we have selected some three-dimensional cuts of the prediction space, as illustrated in Figures 6.7 and 6.8. Figure 6.7 illustrates the effect of the EF load and the codec type on the per-flow delay, for two quantile levels: 0.97 and 1.0. One can observe in the figure that the type of codec has an important influence on the number of flows that can be accepted without leading to an excessive delay. For the same link occupation, the G726 codec clearly presents less delay than the G711 codec. The effect is more significative for lower quantiles. Note that when considering a 1.0 quantile, the per-flow delay is actually equivalent to the aggregated delay, as it represents the maximum delay experienced by any packet in the EF aggregated.

Figure 6.8 illustrates the effect of the per-flow quantile and the proportion of codec type for two levels of link occupation: 0.92 and 0.96. Again, one can observe that when the per-flow quantile is considered, it is possible to accept more traffic without leading to an excessive delay.

During the training process, 187 test points were generated. These test points correspond to the maximum prediction errors determined by the SFP algorithm during the training process. Figure 6.9 (left side) illustrates the histogram representing the error prediction distribution for the test points. One notes that the maximum prediction error was

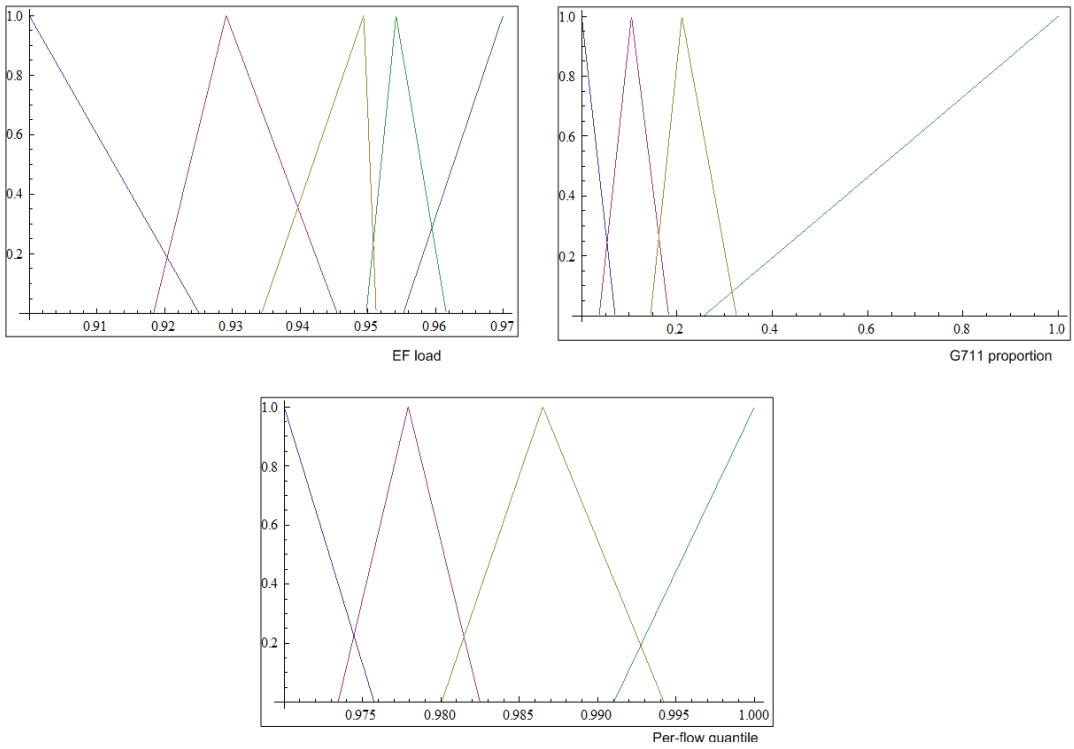

Figure 6.6: EF per-flow delay input membership functions

about 8 ms. Again, it is important to observe that the fuzzy predictor was trained to never present an under-estimate, i.e., the actual delay is expected to be inferior to the predicted ones in 97.5% of the cases. The right side of Figure 6.9 illustrates the distribution of the test points in the solution space.

6.2.3 Comparison with Theoretical Models

As discussed in Chapter 5, to the extent of our knowledge, there is no work in the literature that has addressed the problem of building fuzzy models for predicting queue performance using an off-line training process, the closer matches being those works that develops a fuzzy model to represent existing analytical queue models. However, the queue models addressed in these works are not applicable in our examples. For this reason, we compare in this section the results obtained with our approach with respect to existing analytical queue models that are closer to our examples. We have considered two types of traffic: self-similar traffic conditioned by a token-bucket filter and ON-OFF traffic (i.e., VoIP with VAD).

The self-similar traffic model was presented in Chapter 3. Unfortunately, equation 5.12

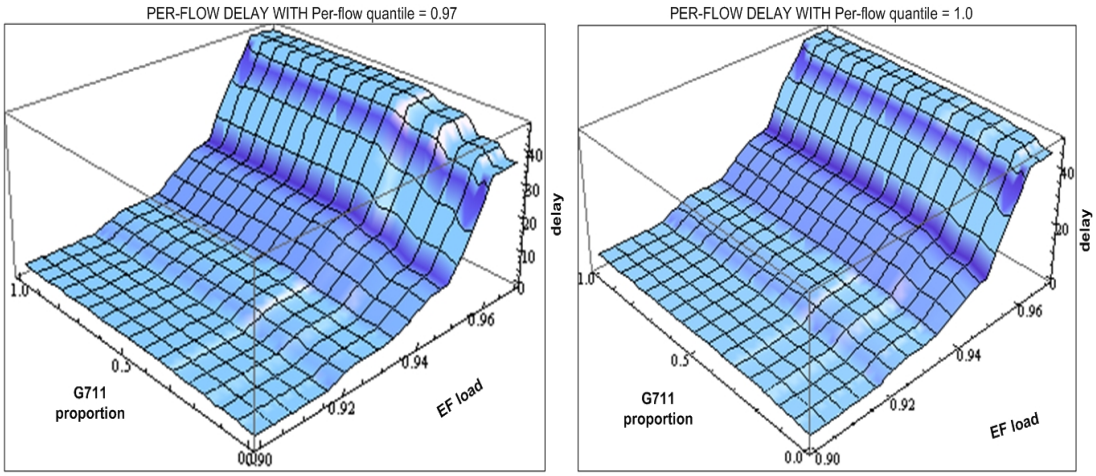

Figure 6.7: Influence of the link occupation and codec type on the EF delay

in Chapter 5 does not apply for the traffic conditioned by a token bucket filter. However, according to [90], it is possible to determine how much traffic is affected by the token bucket filter, by replacing the parameters C by the token-bucket rate (tbrate) and x by the token bucket size (tbsize). As we have fixed the relation $m = 0.9\,tbrate$, we obtained the curve illustrated in Figure 6.10. According to the curve, the self-similar traffic is barely conditioned for high values of tbsize and tbrate. In this region, the token bucket is expected to have a negligible effect, and all self-similar traffic is injected into the network. The example in section 6.2.1 has confirmed this effect, and justifies our choice of the universe of discourse for the variable tbsize. By selecting points in this region, we are able to proceed with the comparison.

Figure 6.11 illustrates a comparison of the prediction error obtained by the MVA Approximation for loss probability theoretical model, as defined in equations 5.13 and 5.14 of Chapter 5, and the fuzzy model, considering the test points obtained in the scenario of the AF Drop Subsystem. Because these equations are only applicable to a "pure" self-similar scenario, we have subtracted the average bandwidth obtained by the VoIP traffic from the link capacity for the test points where VoIP traffic was present. Also, to remain within the validity limits of the analytical model, we have selected only the test points that satisfy two conditions: $m < C$ and tbsize and tbrate values such that the token bucket effect is negligible (tbsize and tbrate values for which the buffer overflow probability under 1% in Figure 6.10). This resulted in 60 out of 149 tests points being considered in Figure

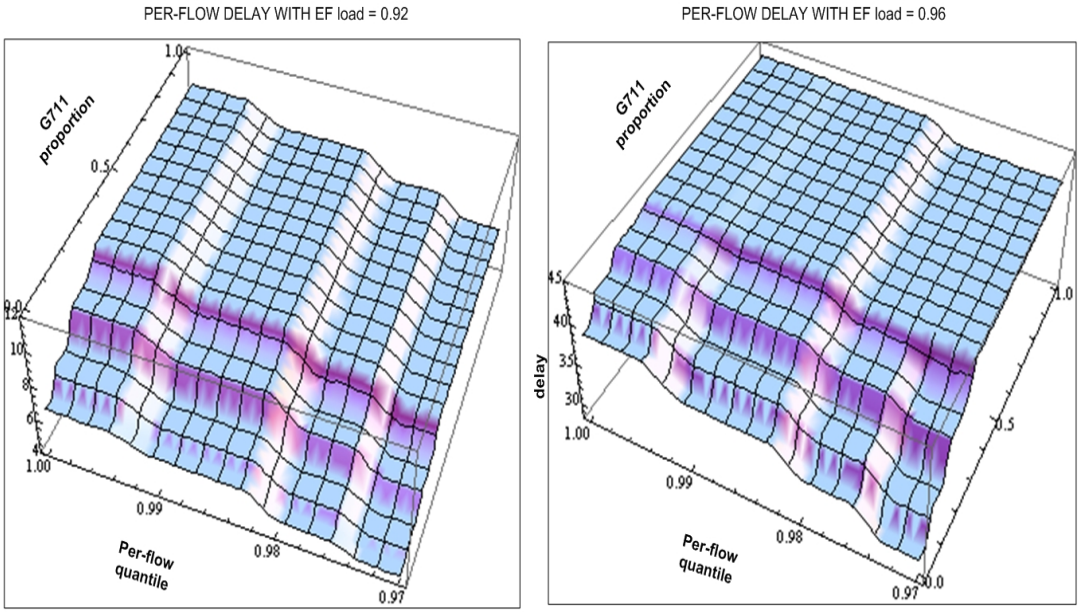

Figure 6.8: Influence of codec distribution and per-flow quantile on the EF delay

6.11. The figure shows that, for this restricted zone, the prediction error for both models is quite similar. The comparison also indicates a bit higher inaccuracy for low drop probability predictions performed by the fuzzy model. It is important to note that this effect can be reduced by increasing the number of fuzzy sets.

In the example of the EF Delay Subsystem, the EF queue is submitted to a traffic composed by the combination of ON-OFF aggregates. The commonest packet loss model for an ON-OFF input traffic comes from equivalent bandwidth studies, which is the characterization of the minimum required bandwidth for this input traffic, such that the packet loss requirement is met. In the following, we compare the results of our predictor with the equivalent bandwidth model proposed by [91]. According to this model, the equivalent bandwidth is calculated as a minimum of two expressions, the first derived as an approximation of the stationary bit rate distribution and the other derived by the fluid-flow model approach. Considering the following parameter for the input traffic, buffer overflow probability ε for a buffer size b, mean m and standard deviation σ of the aggregate bit rate, individual flow peak rate R, and active (t_{on}) and idle (t_{off}) time of individual ON-OFF flows, the equivalent bandwidth is given by:

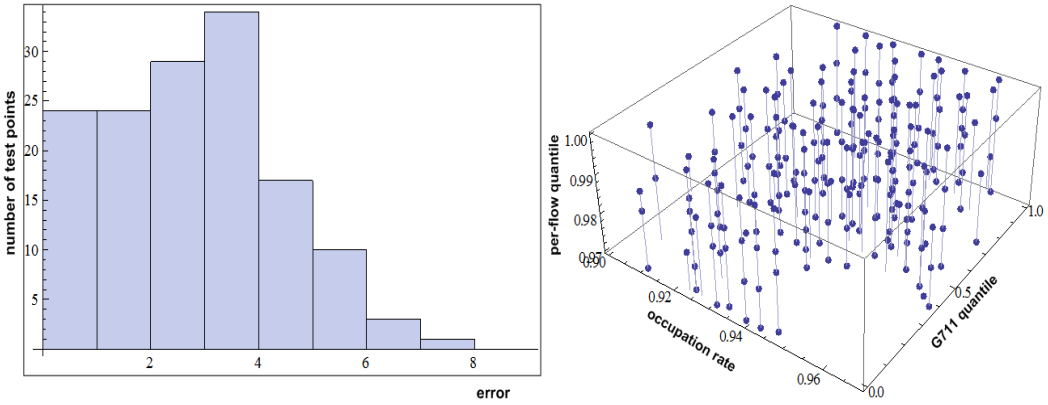

Figure 6.9: Error prediction histogram for the EF delay test points

$$EB = Min\left(m + \sqrt{-2.ln(\varepsilon) - ln(2.\pi)}\,.\,\sigma\,, \sum_{i=1}^{N} \widehat{c_i} \right) \tag{6.1}$$

where:

$$\widehat{c_i} = \frac{\alpha.t_{on}.(1-\rho).R - b + \sqrt{[\alpha.t_{on}.(1-\rho).R - b]^2 + 4.b.\alpha.t_{on}.(1-\rho).R}}{2.\alpha.t_{on}.(1-\rho)} \tag{6.2}$$

with:

$$\alpha = \frac{1}{\varepsilon} \quad , \quad \rho = \frac{t_{on}}{t_{on} + t_{off}} \tag{6.3}$$

We can use equation 6.1 to estimate the delay experienced by a percentage of the VoIP traffic by determining the buffer size that satisfies the inequality $EB \leq link\,capacity$. This equation can be solved only when the average bandwidth occupied by the VoIP traffic is inferior to the link capacity, which corresponds to the universe of discourse adopted in the EF delay subsystem example. Therefore, we can consider the totality of the test points employed in this example. Figure 6.12 illustrates the error prediction obtained by the corresponding test points. In this case, the theoretical model overestimates and underestimates the actual delay determined by simulation. The underestimates are not surprising, as the metric determined by simulation was the per-flow delay, which is, in most cases, higher than the aggregated delay determined by equations 5.13 and 5.14 in Chapter 5. The overestimates seems to be related to the failure of the theoretical model in capturing

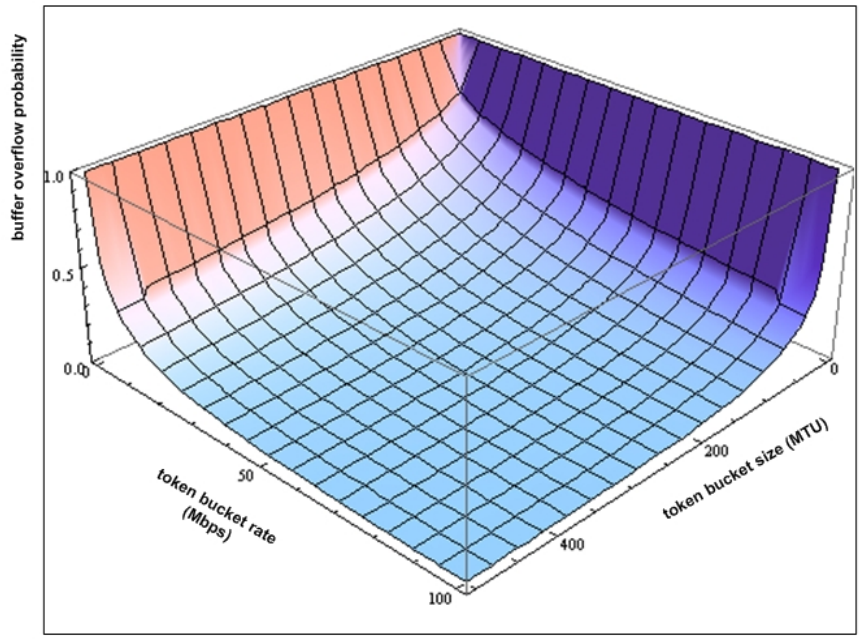

Figure 6.10: Probability of the token bucket filter to constrain the self-similar traffic when m=0.9 C

simultaneously the effects of multiplexing gain and buffering, which is a known problem in the literature.

6.2.4 Algorithm Performance

The training method performs a dual-optimization procedure to obtain the parameters of the fuzzy predictor. The MFP that adjusts the fuzzy logic parameters to minimize the prediction error and the SFP that finds the set of load conditions that maximize the prediction error. As pointed out, the simulation is the most expensive procedure of the algorithm. Considering the definitions supplied in Chapter 5 section 5.3.4, the overall convergence time can be estimated by the following equation:

$$t_c = avgCR_{MFP} \,.\, (tavgC_{MFP} + tavgC_{SFP}) \tag{6.4}$$

Recall that the variable $avgCR_{MFP}$ is the average number of MFP convergence repetitions until the overall convergence criteria are achieved. The variable $tavgC_{MFP}$ is the average time for obtaining a convergence for a given set of test points S_{LC}. It represents

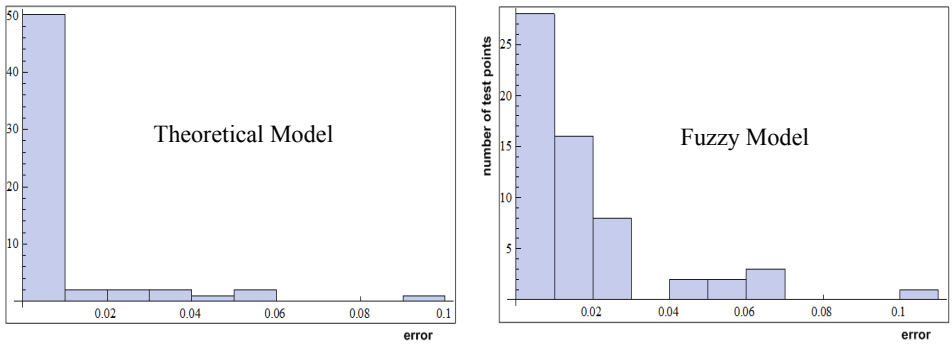

Figure 6.11: Comparison of the error prediction histograms for AF Drop Subsystem

the time for adjusting the membership functions in order to minimize the prediction error corresponding to previously computed test points. This variable is strongly dependent on the number variables required to represent the fuzzy sets in the fuzzy system (v vector dimension). The variable $tavgC_{SFP}$ is the average time for determining a new test point. It is strongly dependent on the number of input variables for the fuzzy system (s vector dimension) and how non-linear is the relationship between these input variables and the metric being evaluated.

Both examples presented in sections 6.2.1 and 6.2.2 have the same complexity in SFP terms, as the dimension of the vector s is three. However, the AF Drop Subsystem presented in section 6.2.1 is less complex than the example of the EF Delay Subsystem discussed in section 6.2.2, in terms of the number of variables necessary to represent the fuzzy sets. The first requires 54 variables, 18 for the input membership functions and 36 for the singletons, whereas the second requires 107 variables, 27 for the input membership functions and 80 for the singletons.

Table 6.1 presents the numbers related to the performance of the MFP procedure. As already stated, the number of repetitions depends on the non-linearity of the involved model, which is more important in the first scenario. On the other hand, the mean time for one convergence is greater in the second case, due to the greater number of involved variables. Please note that the convergence time in Table 6.1 does not consider the SFP time.

Table 6.2 presents the numbers related to the performance of the SFP procedure. The dimension of s vector is the same in both examples. The SFP convergence time is highly dependent on the simulation time. It can be observed that the time spent in one simulation

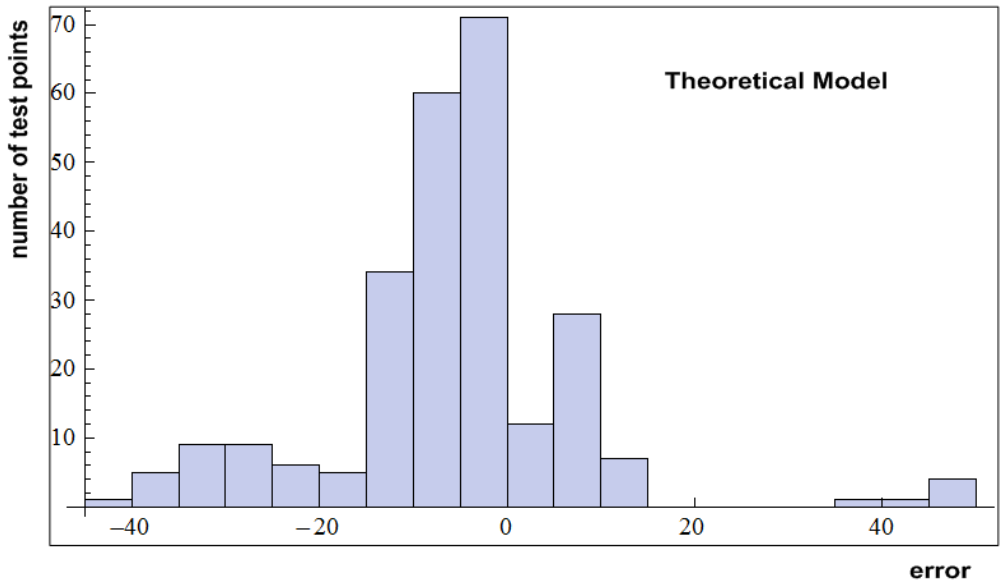

Figure 6.12: Error prediction obtained for test points in the EF Delay Subsystem

Table 6.1: Performance of MFP procedure

MFP procedure	AF Drop Subsystem	EF Delay Subsystem	Percent of Increase
Dimension of v vector	54	107	98%
MFP convergence repetitions: $avgCR_{MFP}$	113	107	-5%
Mean time for one MFP convergence (minutes): $tavgC_{MFP}$	0.28	0.83	196%
Total MFP convergence time (minutes): $avgCR_{MFP}.tavgC_{MFP}$	31.64	88.81	181%

is lower in the AF Drop Subsystem than in EF Delay Subsystem. This is because of the need for per-flow classification in order to compute the performance in terms of packet quantiles in the second. Also, for both examples, the total convergence time is just a bit greater than the total simulation time, which makes the weight of simulation evident in the computing effort. Please note that Total SFP simulation time = (Mean time for one simulation . Total number of simulations) $\div S$. This is because we have taken advantage of the multi processor architecture [1] of the host environment and performed the five simulations required to seed variation in parallel ($S = 5$).

[1] Configuration of the computer used in this study: dual processor 3.16 GHz QuadCore CPU, 32 Gbytes RAM and Linux

Table 6.2: Performance of SFP procedure

SFP procedure	AF Drop Subsystem	EF Delay Subsystem	Percent of Increase
Dimension of s vector	3	3	0%
Mean time for one SFP convergence (minutes): $tavgC_{SFP}$	16.65	23.72	42%
MFP convergence repetitions: $avgCR_{MFP}$	113	107	-5%
Total number of simulations: $avgCR_{MFP} . avgC_{SFP} . S$	9785	9495	-3%
Mean time for one simulation (minutes):	0.96	1.32	38%
Total SFP simulation time (hours):	31.31	41.78	33%
Total SFP convergence time (hours): $avgCR_{MFP} . tavgC_{SFP}$	31.36	42.30	35%

As expected, the computing effort spent in MFP is less important than that spent in SFP. The simplifications introduced in the algorithm (as explained in Chapter 5) lead to feasible off-line solution time: 31.9 hours for the AF Drop Subsystem and 43.8 hours for the EF Delay Subsystem.

6.3 Validation of the Time Based Fuzzy Model for MVA App. for Loss Probability

Figure 6.13 illustrates our sample scenario for the validation of the Time Based Fuzzy Model for MVA Approximation for Loss Probability proposed in Chapter 5. As seen, we consider the performance modeling of a network node composed by one queue that receives self-similar traffic. In this scenario, we are interested in evaluating the queue performance expressed in terms of loss probability. Such simulation scenario is employed for obtaining the simulation data points $D_{j,p,g}$ defined in Algorithm 3 (Chapter 5), for $1 \leq j \leq N$, $1 \leq p \leq M$ and $1 \leq g \leq S$ where N, M and S are defined as the number of t points within universe of discourse of input variable T, the number of H values and the number of simulations performed with distinct seeds, respectively. In this case, before starting the FP algorithm, we compute the accumulated loss rate for each point (j, p, g) in order to initialize $D_{j,p,g}$.

Considering the equation 5.15, $\widehat{\varepsilon} = \widehat{F}(T, H) . \varepsilon$, the goal is to capture the fuzzy queue behavior during "short-term" running in terms of loss probability by obtaining the optimal fuzzy coefficient $\widehat{F}(T, H)$ that gives the fuzzy model $\widehat{\varepsilon}$. Thus, we have to define what a

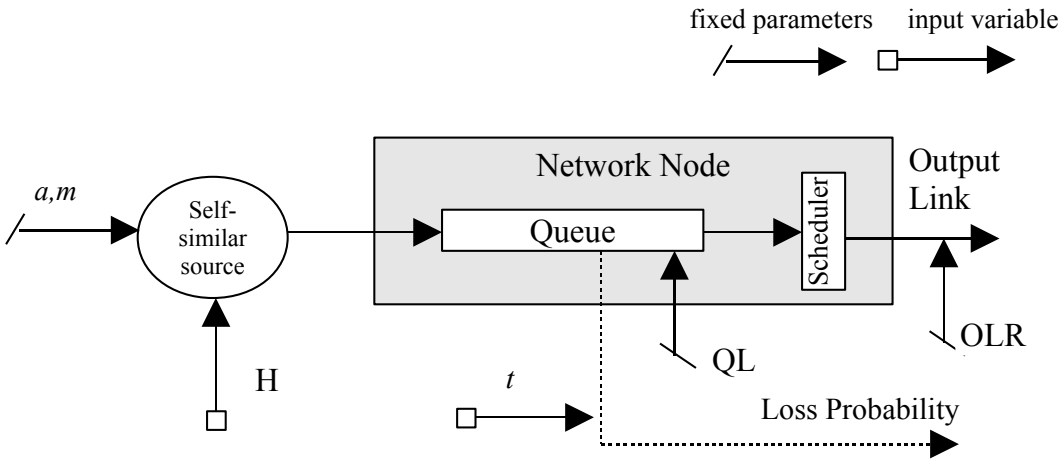

Figure 6.13: Evaluation scenario for the Time Based Fuzzy Model for MVA Approximation for Loss Probability

"short-term" period is as well as the range intervals that will be employed to build $\widehat{F}(T,H)$ (see Figure 5.11 in Chapter 5). The "short-term" period length is related to the real scenario that is being considered. However, we can establish a criterion based on variance estimation [50] in order to determine the stop time. According to this approach, because we have independent simulations with distinct seeds (the variable S defined in Section 5.4.5), we can employ the following method to compute the short-term period length:

$$\overline{x}_i = \frac{1}{n} \sum_{j=n_0+1}^{n_0+n} x_{i,j}, \quad i = 1, 2, ..., S \tag{6.5}$$

where, $\overline{x}_i$ is the mean for each simulation; $n + n_0$ is the length of each simulation; n_0 is the number of discarded observations for transient removal.

The overall mean for all simulations with distinct seeds is obtained as follows:

$$\overline{\overline{x}} = \frac{1}{S} \sum_{i=1}^{S} \overline{x}_i \tag{6.6}$$

Hence, we can calculate the variance of replicate means:

$$Var(\overline{x}) = \frac{1}{S-1} \sum_{i=1}^{S} (\overline{x}_i - \overline{\overline{x}})^2 \tag{6.7}$$

We can calculate a normalized value as follows:

$$\eta = \frac{z_{1-\frac{\alpha}{2}} \cdot \sqrt{Var(\overline{x})}}{\overline{\overline{x}}} \tag{6.8}$$

where, the value $z_{1-\frac{\alpha}{2}}$ is obtained from the standard normal distribution [2]. Thus, range intervals and the universe of discourse of T are defined based on distinct values of η, for which we have the possibility of capturing the variability of loss probability values expressed in terms of range intervals.

The loss probability is represented in terms of the ratio between the packets accepted in the queue and the packets dropped by exceeding the queue size. Because the training method is based on simulation, the loss rate is computed considering several simulations with distinct seeds, in this case, we select $S = 30$.

Below, we present a summary containing the values employed in this evaluation:

- $H = \{0.7, 0.75, 0.8, 0.85\}$;
- $\triangle t = 10ms$ [3]. It gives the step for computing the loss rate and, consequently, the number of simulation points within the universe of discourse of the input variable T;
- $n_0 = 1s$. The first 100 loss rate values are discarded for transient removal [50];
- Output link rate (OLR) = 100 Mbps;
- m = 95 Mbps;
- Queue-length (QL) = 50 ms;
- a = 275 kbit.s.

Considering above values, Figure 6.14 depicts the behavior of the loss probability in terms of the metric η defined in equation 6.8 for a 95% confidence interval. As expected, for higher H values we see higher variability levels for loss probability values. Also, we select $1 \leq t \leq 3000s$ as the universe of discourse of the input variable T. The lower bound is related to the transient removal and the upper bound was defined based on the analysis of simulation logs, where one can observe only a smooth decrease of η for $t >> 3000$.

One key issue related to our approach is the definition of range intervals. We consider the variability of loss probability values along the universe of discourse of T to define them. Taking a time t as a reference, say t_{range}, the range interval defined at this point is specified in such a way where all loss probability values observed *after* t_{range} [4] are within this range

[2] for $S \geq 30$ or from the t-student distribution in other cases [50].

[3] defined in section 5.4.5

[4] Also including t_{range}.

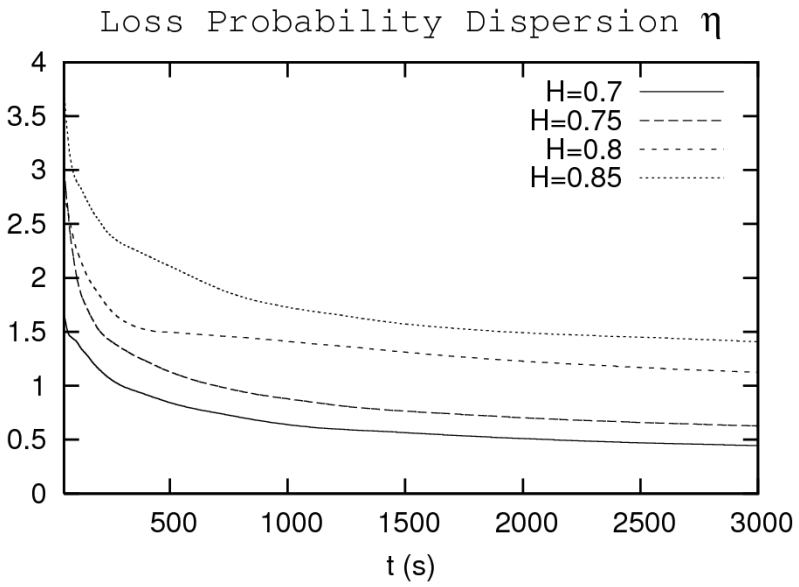

Figure 6.14: Loss probability dispersion η

interval. Thus, each range is expressed in terms of a pair (d_{min}, d_{max}), where d_{min} and d_{max} are, respectively, the minimum and maximum loss rate values observed for $\forall\, t \geq t_{range}$.

The loss probability dispersion metric, η, permits us to establish a criterion to define the time t to be used as a reference for these range intervals specification. Recall the assumptions discussed in Chapter 5 section 5.4.3 about range intervals. As seen, $range_3$ is the range related to the stability region, which will include the crisp value ε provided by the analytical model MVA for loss probability. Around this range, $range_2$ and $range_4$ represent two intermediate ranges for regions that have an average degree of dispersion and, finally, the last two, $range_1$ and $range_5$, for the cases of high degree of dispersion.

Let $\eta_{asymptotic}$ to be the value of η at a time t in the stability region and $\eta_{average}$ to be the value of η where we have a time t with an average dispersion. From Figure 6.14, we obtain $t_{asymptotic} = 3000s$ as a reference for defining $range_3$. We assume $\eta_{average} = 1.25.\eta_{asymptotic}$ for defining the $t_{average}$ value in order to obtain $range_2$ and $range_4$, which represents a point with an increase of 25% in the degree of dispersion. Remaining points belong to $range_1$ and $range_5$. Thus, the upper and lower bounds of $range_3$ are defined in order to include all loss probability values observed at $t_{asymptotic}$; the upper and lower bounds of $range_2$ and $range_4$ are defined in order to include all loss probability values observed at $t_{average}$ and beyond; and the upper and lower bounds of $range_1$ and $range_5$ are

defined in order to include all loss probability values.

Table 6.3 presents the normalized values for range intervals taking as reference the ε value obtained from the analytical model. One can observe an increase of range widths due to fluctuations of loss rate values generated by a worse behavior of traffic for higher H values. Also, as expected, the value 1, which represents the ε, is included in the $range_3$.

Table 6.3: Normalized range values for the universe of discourse of T

H	MVA L.Pr.(ε)	$range_1$	$range_2$	$range_3$	$range_4$	$range_5$
0.70	$7.60E^{-3}$	[0.0,0.3742)	[0.3742,0.5376)	[0.5376,1.3354)	[1.3354,1.7634)	[1.7634,4.0289]
0.75	$8.83E^{-3}$	[0.0,0.1785)	[0.1785,0.3146)	[0.3146,1.0945)	[1.0945,1.3897)	[1.3897,4.9875]
0.80	$1.05E^{-2}$	[0.0,0.0803)	[0.0803,0.1556)	[0.1556,1.2096)	[1.2096,1.7543)	[1.7543,6.2015]
0.85	$1.26E^{-2}$	[0.0,0.0102)	[0.0102,0.3276)	[0.3276,1.2488)	[1.2488,1.9989)	[1.9989,10.294]

Figure 6.15 shows the convergence results after FP optimization obtained for the membership functions of the input variable T. For clarity, this Figure only shows the $range_1$, $range_2$, $range_4$ and $range_5$ membership functions for the optimized multipliers λ for $H = 0.7$. It is important to remember that these membership functions are unique for H=$\{0.7, 0.75, 0.8, 0.85\}$; only the optimized multipliers λ are distinct for each pair (range,H), as denoted by the vector 5.18 in Chapter 5. As expected, we have a high level of variability during initial phase of simulations, which is represented by the presence of all ranges with similar degree of membership. Around $500\,s$, loss rate values within $range_1$ and $range_5$ are almost negligible. The system converges to $range_3$ when close to $3000\,s$.

Using the optimal input membership function of T showed in Figure 6.15, we can obtain the normalized output fuzzy set in order to have the corresponding normalized probability distribution for any pair (T, H). For example, consider the time $t = 500s$ and $H = 0.7$ as illustrated in Figure 6.16. Figure 6.16(a) shows the normalized values after applying equation 5.16 (Chapter 5). Considering such values, we have the corresponding normalized output fuzzy set for this pair, which is shown in Figure 6.16(b). The range intervals depicted in this Figure were already shown in Table 6.3. The ε corresponds to $d = 1$.

Following the model presented in Figure 6.16(a), Figures 6.17, 6.18, 6.19 and 6.20 present the normalized values obtained from fuzzy inference taking the optimized membership functions shown in Figure 6.15 and the optimized multipliers λ for all $t \in T$. Also, in

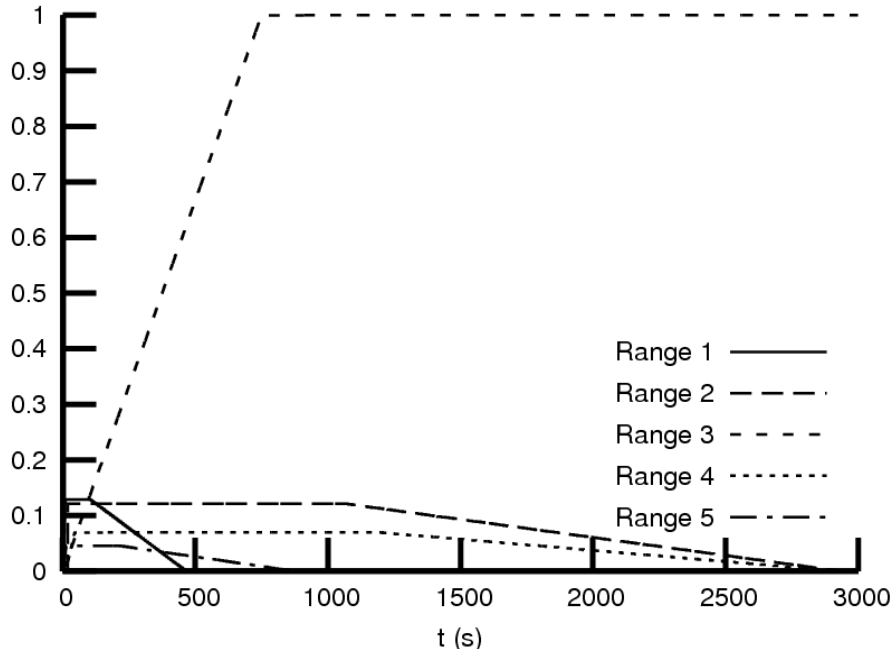

Figure 6.15: $\widehat{F}$ input membership functions for the universe of discourse of T

order to make a comparison, we plot the results obtained directly from the corresponding simulation points, which are calculated from histograms using the same range intervals. For each t point we have normalized values that indicate the degree of membership of t in the corresponding range. The total sum is always 1 for each point t.

Despite the use of the same optimized membership functions of T for all H values, it is interesting to note the effect of the multipliers λ, whose optimized values are shown in Table 6.4. For example, we clearly see that the height of the membership function $range_2$ for $H = 0.7$ in Figure 6.17 is greater than for $H = 0.8$ in Figure 6.19. This is because $range_2$ has a higher weight in terms of the number of simulation points for $H = 0.7$, forcing the FP optimization algorithm to compensate using such factor. According to these curves, the fuzzy model $\widehat{F}(T, H)$ can be seen as a good approximation for the queue behavior in terms of loss probability. However, in order to precisely evaluate this statement, Figure 6.21 shows the Chi-Square Test [50] results for assessing the fitting level between each pair.

The Chi-Square test allows us to calculate the fitting level using the following equation:

$$D = \sum_{i=1}^{5} \frac{(hist_i - fuzzy_i)^2}{hist_i} \tag{6.9}$$

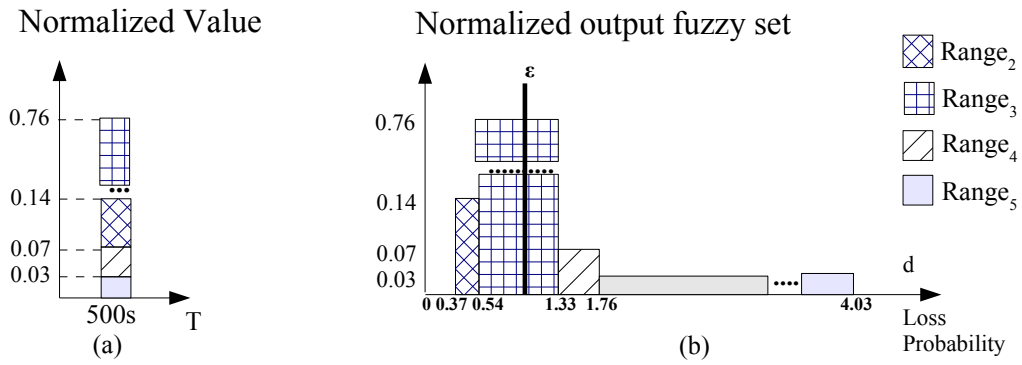

Figure 6.16: Example: Normalized output fuzzy set for $t = 500s$ and $H = 0.7$. (a) Normalized value from input membership functions T and H. (b) Resulting Normalized Output Fuzzy Set for Loss Probability for the pair (500,0.7)

Table 6.4: Optimal λ multipliers for height adjustment of membership functions of T

H	$range_1$	$range_2$	$range_4$	$range_5$
0.70	0.144	0.188	0.108	0.039
0.75	0.128	0.121	0.069	0.045
0.80	0.086	0.029	0.049	0.027
0.85	0.101	0.041	0.039	0.009

where, $hist_i$ denotes a normalized value obtained from the histogram of simulation points for the range i; $fuzzy_i$ is a normalized value obtained from the inference with $\widehat{F}(T, H)$ for the range i. We compute this fitting metric D for each t point. Based on this metric, we obtain the confidence level from the Chi-Square distribution [50], which is plotted in the Figure. A confidence level 1 indicates a 100% fitting level, whereas 0 indicates a zero fitting level. Considering that we are using $S = 30$ distinct seeds for simulations and taking into account these 30 values for each point in the universe of discourse of T with increments of $\triangle t = 10ms$, the results are acceptable, except for some points during the initial period of the universe of discourse of T.

Recall that the fuzzy model $\widehat{F}(T, H)$ is intended to serve as an "adjusting" coefficient for the short-term of the MVA for Loss Probability analytical model. It allows us to observe the queue behavior in terms of loss probability by capturing the variability observed during

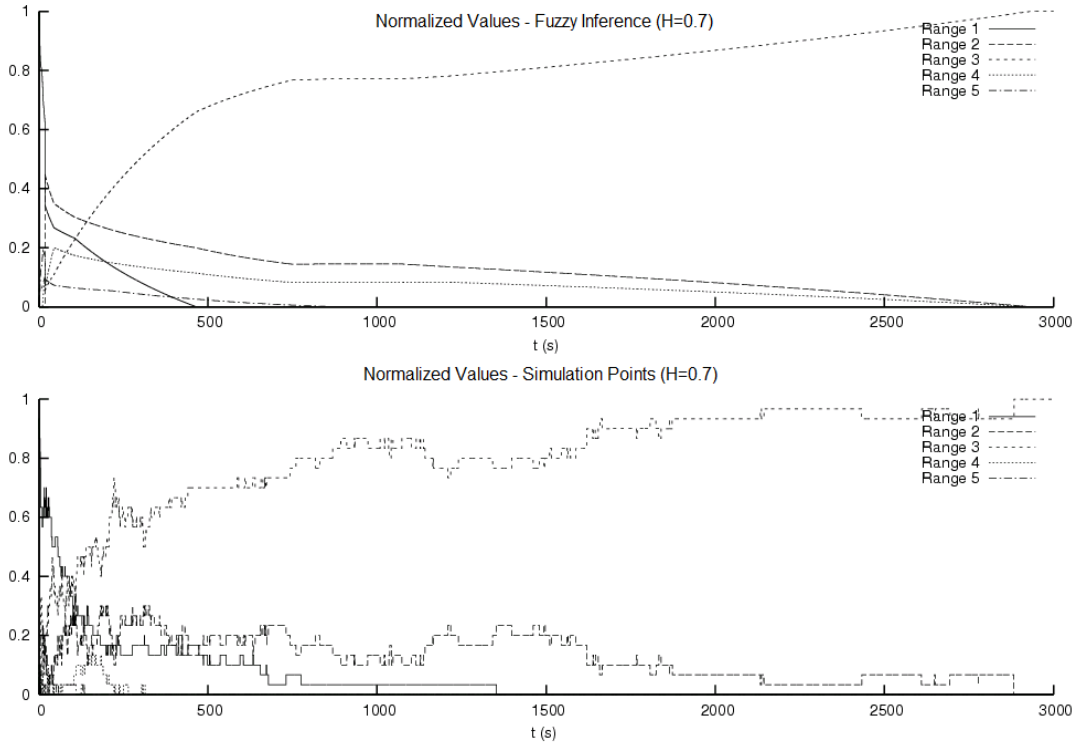

Figure 6.17: Normalized Values from Fuzzy Inference *versus* Normalized Values from Histograms of Simulation Points - H=0.7

such period. In order to illustrate how the fuzzy model $\widehat{F}(T,H)$ can be employed for such investigation, Figures 6.22 and 6.23 show the normalized output fuzzy sets of $\widehat{F}(T,H)$ obtained through fuzzy inference for several values of t for $H = 0.85$. The widths of each bar are merely illustrative.

In these Figures, the bars are expressed in terms of the ranges defined in Table 6.3. As can be observed, for $t = 1s$ all drop values are within $range_1$. For $t = \{10, 50, 100\}$, we can see a high loss variability level, which is indicated by the presence of several ranges. At $t = 500s$, there are no more values in $range_1$ and for the $range_5$ they are almost negligible. For $t = 1000s$, the "asymptotic" range $range_3$ shows a high strength, which is 100% at $t = 3000s$. Considering such results, one can infer about the queue behavior in terms of loss probability for a specific pair (T,H). Also, defuzzification methods could be applied in order to obtain a crisp value for loss probability, if required. Table 6.5 shows the corresponding crisp values for the loss probability of the t points considered in Figures 6.22 and 6.23 according to different defuzzification methods discussed in Chapter 2. The

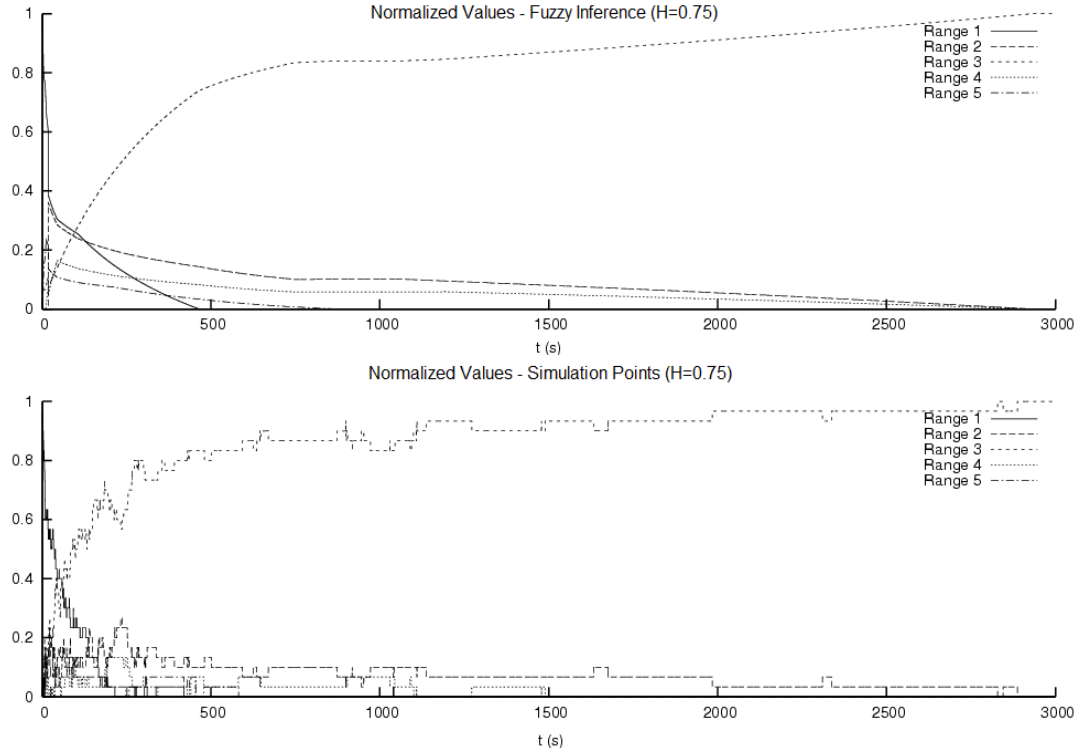

Figure 6.18: Normalized Values from Fuzzy Inference *versus* Normalized Values from Histograms of Simulation Points - H=0.75

values are normalized by the loss probability value obtained from the analytical model (i.e., $\varepsilon = 1.26E^{-2}$)

As can be noted, crisp values may diverge according to the defuzzification method chosen. Also, they are very dependent on the range limits and depending on the approach employed by the defuzzification method, they can affect differently the final crisp value. For example, consider the case of the COA and COG defuzzification methods for $t = 10s$. As seen in Figure 6.22, only $range_1$, $range_3$ and $range_5$ are active. In spite of having a relative small weight, $range_5$ has a large width, as can be seen in Table 6.3. For the COA method, we take the average point within the range in the defuzzification process, whereas for the COG method we calculate the corresponding area. For this reason, the COG value is higher than the COA value. It is also interesting to note that all methods converge to similar values for higher t values. One can see in Figure 6.23 that we have only active the ranges $range_2$, $range_3$ and $range_4$, which have smaller widths as can be seen in Table 6.3, diminishing the influence of the defuzzification method in the final result. Finally, it is

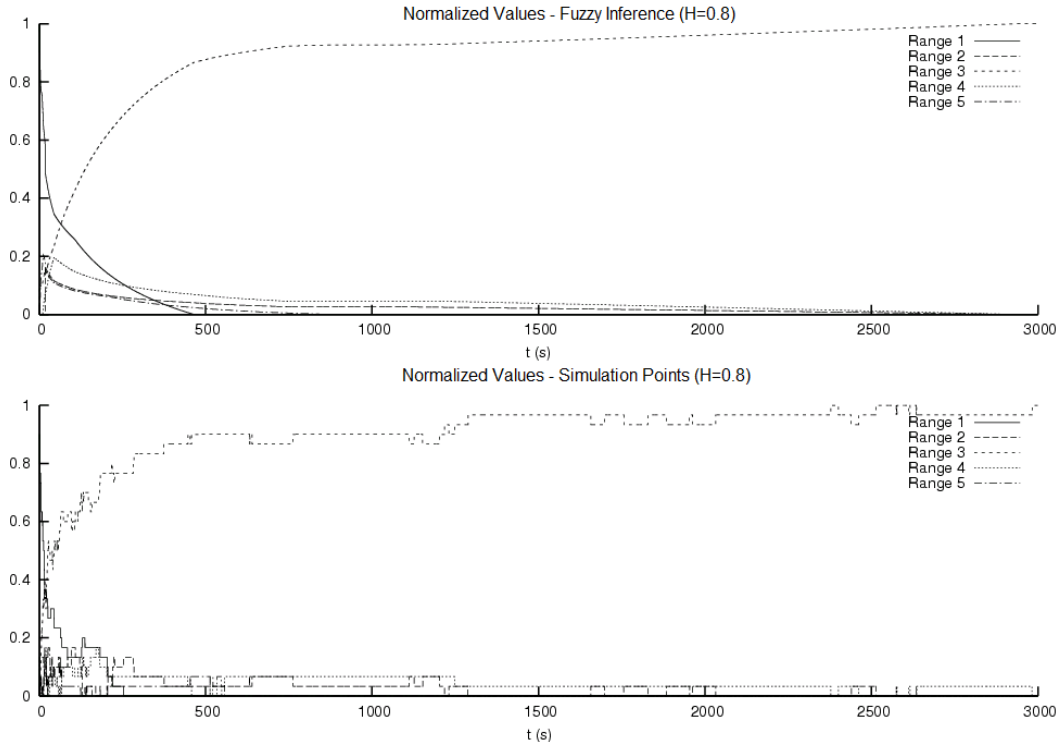

Figure 6.19: Normalized Values from Fuzzy Inference *versus* Normalized Values from Histograms of Simulation Points - H=0.8

also important to observe the crisp values close to the "asymptotic" region. As stated by the analytical model, the MVA approximation for loss probability is intended to serve as an upper bound for loss probability in finite buffers. In this context, we can see that the results shown in Table 6.5 are according to this statement, since in this region all values are less than 1 which represents the reference value from the analytical model.

6.3.1 Algorithm Performance

The training method performs one optimization procedure to obtain the parameters of the optimal $\widehat{F}(T, H)$, i.e., only the MFP procedure that adjusts the fuzzy logic parameters to minimize the overall error given by the difference between the normalized values obtained from fuzzy inference and the corresponding normalized values computed from histograms of simulation points for all $t \in T$ and all H values. As discussed previously, the simulation is the most expensive procedure of the algorithm. Considering the definitions supplied in Chapter 5 section 5.4.5, the number of simulations can be expressed as follows:

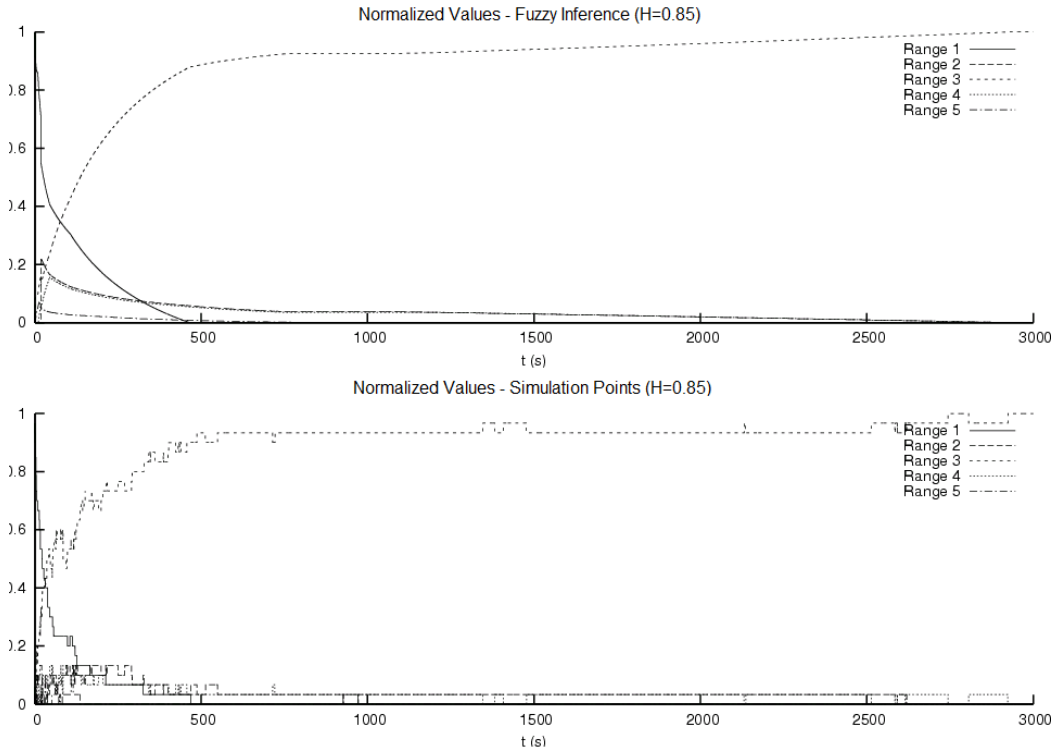

Figure 6.20: Normalized Values from Fuzzy Inference *versus* Normalized Values from Histograms of Simulation Points - H=0.85

$$ns = S \,.\, M \tag{6.10}$$

where, S is the number of simulations performed with distinct seeds and M is the number of H values. As discussed previously, simulations cited in equation 6.10 are not repeated according to the number of candidate solutions generated by the MFP procedure, because all simulation data are generated before starting the MFP and we don't have the SFP procedure. Also, for each pair given by a seed value in S and H value in M, we have only one simulation run whose duration is the universe of discourse of T. In this case, we obtain the accumulated loss rate at each time t_j along this period.

Thus, the overall convergence time can be estimated by the following equation:

$$t_c = t_{ns} + tCR_{MFP} \tag{6.11}$$

The variable tCR_{MFP} is the time spent for the MFP repetitions until the overall convergence criteria are achieved. It is important to mention that there is only one MFP

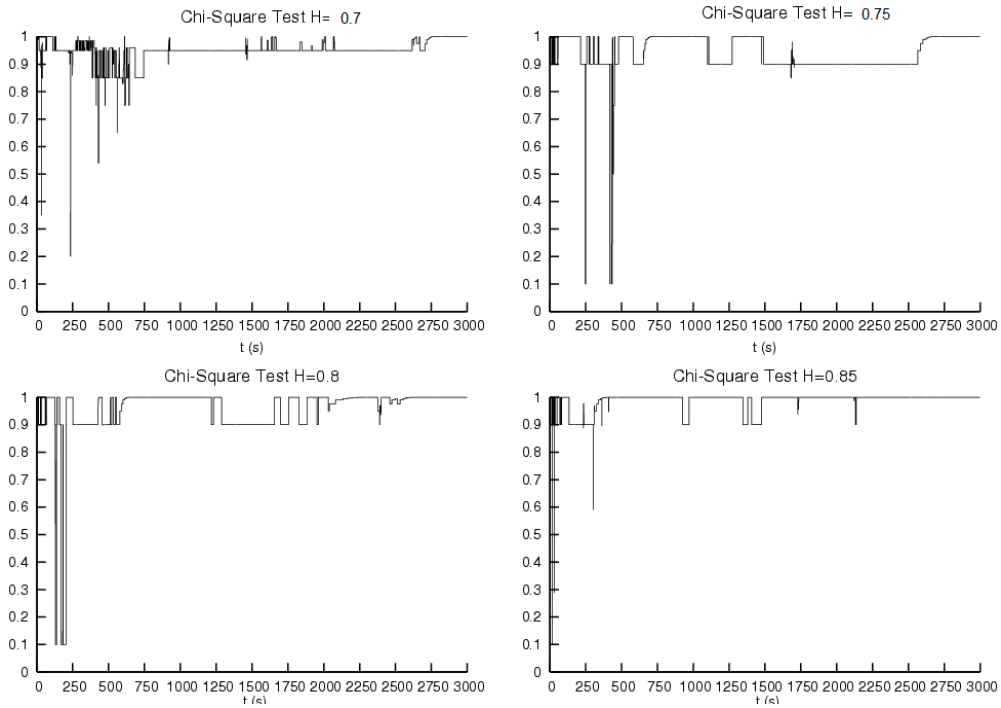

Figure 6.21: Chi-Square Fitting Test: $\widehat{F}(T,H)$ and normalized values obtained from histograms of simulation points

procedure that takes into into account all points within the universe of discourse of the input variables, as seen in equation 5.17 (Chapter 5). It represents the time for adjusting the membership functions in order to minimize the overall error. This variable is strongly dependent on the number variables required to represent the fuzzy sets in the fuzzy system (v vector dimension). The variable t_{ns} is the overall time for generating the simulation data.

Table 6.6 presents the numbers related to the performance of the MFP procedure and the generation of simulation data for the optimization problem. As in the case of the fuzzy predictor, we have employed the same machine with a multi processor architecture, in which we can perform five concurrent simulations to reduce t_c.

6.4 Conclusion

In this Chapter, we showed numerical results and the evaluation of the proposed method presented in Chapter 5. Our evaluations show that the Flexible Polyhedron (FP) algorithm

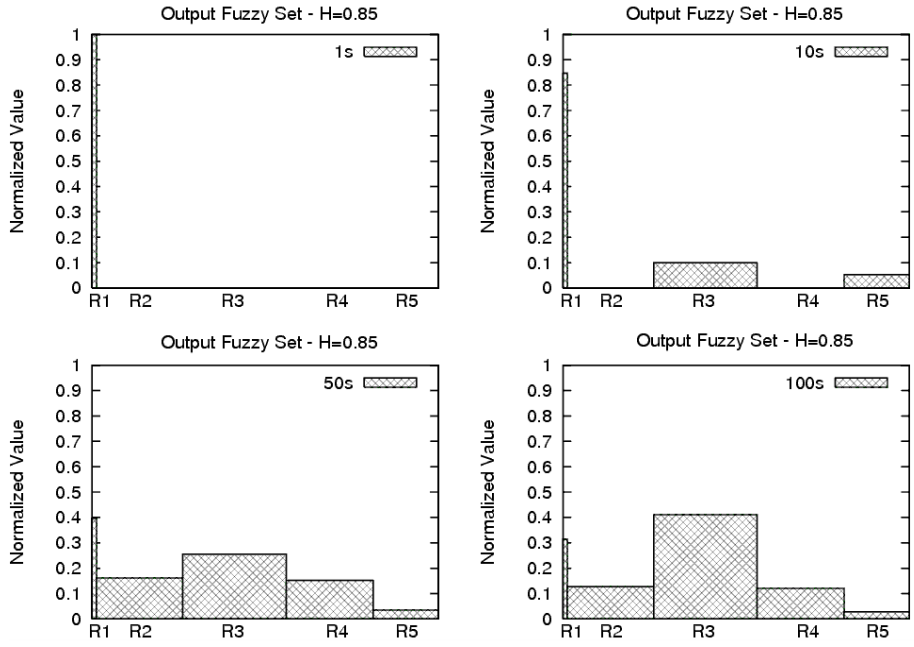

Figure 6.22: Output Fuzzy Sets for $t = \{1, 10, 50, 100\}$

behaves surprisingly well, even when a large number of parameters are simultaneously optimized, as illustrated in the EF Subsystem of the Fuzzy Predictor, where 107 variables were required to represent the fuzzy sets by the optimization procedure. In this case, the method is barely limited by the complexity of the fuzzy system. The training process, however, can be somewhat complex when a large number of input variables are considered. Therefore, it is convenient to limit the choice of the input variables to those that could be controlled by the traffic engineering.

In the case of the fuzzy predictor, the results are valid for a specific transmission rate and the queuing discipline. Even though it is possible, it is not worthy to consider these parameters as input variables for the fuzzy system, as they are well known parameters, and are not supposed to change. The fuzzy predictor can also be sensitive to other traffic parameters such as the fBm parameters and VoIP modeling parameters. In practice, however, both VoIP parameters and fBm parameters are not controllable by the traffic engineering and must be determined by observing the real traffic. However, it is possible, by adjusting the training process, to generate more conservative predictions in order to accommodate bad estimations on these parameters.

For the case of the Time Based Fuzzy Model for MVA Approximation for Loss Prob-

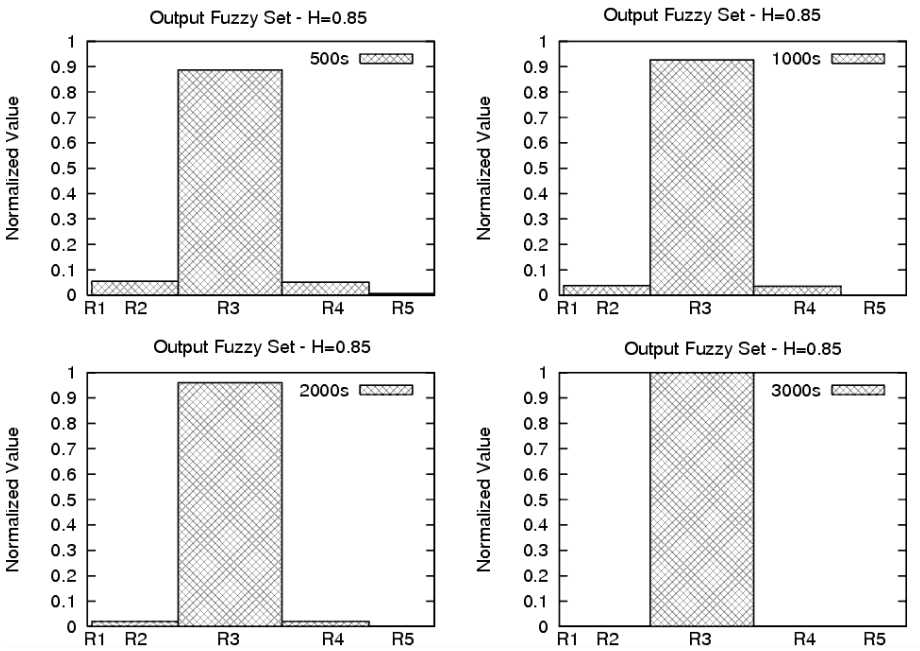

Figure 6.23: Output Fuzzy Sets for $t = \{500, 1000, 2000, 3000\}$

ability analytical model, we showed how the model can be used for observing the queue behavior in terms of loss probability during the short-term period. Our results confirmed that the analytical is a good approximation for loss probability in finite queues, where all crisp values obtained from several defuzzification methods respected this upper bound close to the "asymptotic" region. Also, we saw that the optimized fuzzy coefficient $\widehat{F}(T, H)$ is a good model for representing the queue behavior in terms of loss probability, which presented a good fitting level when compared with loss rate values obtained from simulation data.

Table 6.5: Normalized Crisp Values for $t = \{1, 10, 50, 100, 500, 1000, 2000, 3000\}$ and $H = 0.85$

t(s)	COA	COG	Badd ($\alpha = 2$)	Badd ($\alpha = 30$)
1.00	0.0051	0.0051	0.0051	0.0051
10.00	0.4055	5.1366	0.0387	0.0051
50.00	0.6929	3.1185	0.3727	0.0051
100.00	0.7127	2.5184	0.5499	0.7879
500.00	0.8337	1.1357	0.7889	0.7882
1000.00	0.7945	0.8049	0.7884	0.7882
2000.00	0.7926	0.7977	0.7883	0.7882
3000.00	0.7882	0.7882	0.7882	0.7882

Table 6.6: Performance for the Time Based Fuzzy Model for MVA aproximation for Loss Probability

MFP procedure	Time Based Fuzzy Model
Dimension of v vector	34
S	30
M	4
t_{ns} (minutes)	37.44
tCR_{MFP} (minutes)	1.83
t_c (minutes)	39.27

Chapter 7

Conclusion and Future Work

This thesis addresses the design of performance models of network nodes. In this case, we proposed a new method for modeling queue behavior where such models can be expressed in terms of common performance metrics, like the amount of loss and delay imposed by the queue on the traffic injected to the output link, permitting to establish the relationship between the traffic load in queues of network nodes and the resulting QoS performances. Based on a general off-line method, the proposed method combines non-linear programming and simulation to build a fuzzy logic based model capable of determining the performance of a network node without the derivation of an analytical model. The proposed methodology can be applied to any type of traffic and includes a training method that permits the application of any type of performance metric. Also, it is possible to find out the optimal values that can be used for the configuration of network nodes, with important applications on traffic engineering and capacity planning.

Despite the characterization of a model for the queue length distribution is a very discussed issue in the literature, our study has been motivated by the fact that devising analytical models sometimes becomes a challenging task due to the associated complexity when it is necessary to deal with common resources employed in network deployment scenarios, like multiple priority queues, non-exponential service times and long term correlated traffic. In fact, even in scenarios where there exist analytical models they may not represent accurately the system tending to overestimate or underestimate the actual performance when it is typically necessary to perform optimization procedures for an adjustment of model parameters. Furthermore, these analytical models usually apply only to the performance of the aggregated traffic, but some authors have pointed out that under certain conditions

the performance experienced by individual flows can significantly diverge with respect to the performance of the aggregated traffic. One of the main contributions of this method is to take into account many of these complex issues, where it is typically difficult to develop an analytical model.

Aiming to carry out the modeling proposed by the method, our approach employs fuzzy logic systems for capturing the queue behavior, which is clearly non-linear, making the fuzzy approach a very useful tool for such modeling. So, in Chapter 2 we discussed the main concepts related to fuzzy systems. Fuzzification, Inference and Defuzzification steps were presented, which have evidenced the importance of membership function modeling in order to have an accurate representation of system behavior. Considering the utilization of simulations, Chapter 3 presented the traffic models employed for synthetic traffic generation for the evaluation of the proposed method. VoIP flows were modeled as ON-OFF sources including two additional parameters based on exponential variables in order to regulate the life-cycle of flows for more realistic simulation scenarios. Data traffic simulation is based on self-similar traffic model which was generated as a fractional brownian motion process. We performed the validation of the synthetic self-similar traffic using several well-known tests which confirmed the self-similar property. Also, the simulation environment developed in this research work was discussed along with its main entities.

Optimization methods represent an important tool for the definition of membership function. In this case, fuzzy model parameters can be fine tuned in order to improve its accuracy, since the modeling proposed in this thesis is based on input and output variables whose functional relationship is not known. Thus, Chapter 4 presented the main concepts related to optimization theory along with the Flexible Polyhedron method that was employed for solving optimization problems in this research work, which is a zero-order direct search method that does not require derivatives of the cost function. As we saw, this feature allows us to create optimization problems that handle several parameters in a very flexible manner, where objective functions can be easily used as results of complex computer simulations. In order to avoid the problem of local minima, we employ a modified version of the FP algorithm based on successive rebuildings of the polyhedron after intermediate convergences. Our results show that the FP algorithm behaves well, even when a large number of parameters are simultaneously optimized.

In Chapter 5, we presented the new method for modeling queue behavior considering two different approaches related to the modeling of fuzzy rule consequents and the corresponding application of results. Firstly, we showed how our method can be employed to build a fuzzy

predictor. In this case, the consequents of fuzzy rules were modeled using singletons and the results are expressed in terms of crisp values considering the corresponding performance metrics. Taking into account input variables defined in terms of t_{espec} parameters with traffic specification and output variables described in terms of common performance metrics (e.g., loss rate and delay), we proposed the fuzzy predictor considering a dual off-line optimization training method. Such training approach is performed by solving two simultaneous flexible polyhedron optimization problems. We showed that the proposed fuzzy predictor can be employed to determine optimal values that could be used for the configuration of network nodes, with important applications on traffic engineering and capacity planning. Secondly, we presented a more generic approach, where our method is employed to provide an implied output fuzzy set as a result instead of crisp values. We use a strategy based on normalized probability distributions in order to build a performance model of a network node in which a queue is fed by a self-similar traffic source for building a time based fuzzy model for the MVA approximation for loss probability analytical model. Such fuzzy logic model could be viewed as a fuzzy adjustment coefficient which aims to capture the loss probability variability observed before the system achieves the stability region.

Chapter 6 showed the validation of the proposed method considering three different scenarios. The first two are related to the fuzzy predictor evaluation whereas the third presented the results of the time based fuzzy model for the MVA analytical model. Firstly, we presented the results of the fuzzy predictor for the AF Drop subsystem. In this case, the optimized fuzzy model produces four dimension graphs because of the utilization of three input variables, number of VoIP flows, bucket size and bucket rate, and one output variable, the drop rate of self-similar traffic. Considering a multi-queue node, the results show the impact of the EF load and token-bucket parameters on the drop rate of self-similar traffic. Secondly, the results of the fuzzy predictor for the EF delay subsystem were shown. The optimized fuzzy model also produces four dimension graphs due to three input variables, EF load, VoIP codec proportion and per-flow quantile, and one output variable, the delay of VoIP traffic. In this case, our results show the effect on the EF delay of having two different VoIP codecs considering a per-flow quantile basis. In both cases, some three dimensional cuts were presented and discussed, where one could observe the behavior of the considered QoS performance metrics in several perspectives. Also, a comparison with theoretical models and an analysis of the method performance were shown. Finally, we showed the resulting fuzzy model for the MVA approximation for loss probability analytical model. Considering the time T and the Hurst parameter H as input variables, and the

loss probability as a fuzzy output variable, the resulting normalized output fuzzy sets were discussed, which represent the loss probability variability within the universe of discourse of T and H. The results and fitting tests show that the optimized fuzzy model is a good model for representing the queue behavior in terms of loss probability when it was compared with loss rate values obtained from simulation data.

As an ongoing perspective, there are open issues and important directions of future work. Considering the case of fuzzy predictors, there is still much work to be done with respect to the choice of the membership functions to be trained. Simpler membership functions are easier to compute, but tends to generate models that accommodates non-linearities as small steps, instead of smooth curves. Also, the method can be easily adapted to generate fuzzy models for existing queue systems, in order to create expressions that could be evaluated in real time. In this case, instead of using simulations, the SFP algorithm would generate inputs for a theoretical model. It seems also a possibility to use the training process by simulation to improve or adjust existing queuing models. For example, we could generate a fuzzy model to adjust the output of existing aggregated VoIP model in order to generate per-flow estimates. The introduction of some sort of feedback could also be useful in order to provide less conservative estimates and would be an important issue to be addressed in our future research. For the case of the time based fuzzy model for MVA approximation for loss probability, one important study is to evaluate the necessity of including other parameters of the analytical model in the fuzzy model in order to have more general results. Also, the investigation of the use of other membership functions instead of singletons for modeling the Hurst paramater H would be a natural improvement. Another issue is related to the specification of range intervals employed for modeling normalized output fuzzy sets. A more flexible approach permitting the choice of the number of ranges and their inclusion as variables for the optimization problems should be investigated in order to have an automated definition of ranges based, for instance, on the minimization of the difference observed among crisp values obtained from different defuzzification methods.

Appendix A

Simulation Environment: Pseudo Code

In this appendix, we provide the pseudo code related to the main entities that were implemented in the C simulator developed for the evaluation of the proposed method. The primitives shown in Chapter 3 section 3.4 are cited in the following codes.

A.1 VoIP Flow Traffic Generator

Each VoIP Traffic Generator is implemented by a SIMPATICA actor thread and simulates an individual VoIP flow, modeled as an ON-OFF source, where both ON-OFF intervals are exponentially distributed. The flow generates packets at the specific codec rate during the ON state and keeps silent during the OFF state. The life cycle variability of VoIP flows has been modeled by assuming one additional parameter, AFD (average flow duration), represented by the average of an exponentially distributed random variable. The AFD parameter allow us to capture the impact of the load variation with respect to the performance level of a percentile of packets within each VoIP flow on a long-term basis, providing a better representation of the real scenario.

```
t_DUR = T^i_DUR;
j = 0;
seq = 0;
while j < t_DUR do
    k = 0;
    t_ON = exponential(rng, T^i_ON);
    t_OFF = exponential(rng, T^i_OFF);
    while k < t_ON do
        pkt = (pkt*)msg_create(sizeof(pkt));
        pkt-> birth = time_now();
        pkt-> seq = seq++;
        pkt-> bytes = PKT_SIZE;
        pkt-> flowId = i;
        pkt-> queue = EF;
        msg_send(pkt, nodeID);
        task_sleep(1/PKT_RATE);
        k = k + 1/PKT_RATE;
    end
    task_sleep(t_OFF);
    j = j + t_ON + t_OFF;
end
```

Algorithm 4: VoIP Flow Traffic Generator

where:

T^i_{DUR}: Duration of flow i, which is obtained from paramater AFD.

T^i_{ON}: Mean value of the ON time of flow i.

T^i_{OFF}: Mean value of the OFF time of flow i.

rng: Random number generator.

$pkt->queue$: Queue which the packet will be inserted.

$exponential()$: Returns a value according to the exponential distribution.

$nodeID$: Number that represents the network node.

$time\_now()$: Returns the current simulation time.

PKT_RATE: Number of packets per second according to the codec of the simulation scenario.

PKT_SIZE: Packet size according to the codec of the simulation scenario.

A.2 Self-Similar Traffic Generator

The data traffic is modeled as a fractional Brownian motion, as presented in Chapter 3, where the accumulated aggregate arrival process is represented by the following equation:

$$A_t = m \,.\, t + \sqrt{a\, m} \,.\, Z_t \tag{A.1}$$

where Z_t is a normalized fractional Brownian motion (fBm), m is the mean input rate, $a > 0$ is a variance coefficient and H is the Hurst parameter of Z_t. As aforementioned, the function Z_t was implemented using the Hosking method. For each $\triangle t$ window we compute the arrival process and generate a packet series $A_{\triangle t}$ assuming a MTU size (1500 bytes) and a minimum packet size of 46 bytes. Thus, we generate $A_{\triangle t}$ bytes for each time interval $\triangle t = [t-1, t]$ according to equation:

$$A_{\triangle t} = max\,\{\, m.\triangle t + \sqrt{a\, m} \,.\, FGN_{\triangle t}\,,\, 0\} \tag{A.2}$$

where $FGN_{\triangle t}$ is the fractional Gaussian noise, the incremental variation of two consecutive values of $Z(t)$, that is, $FGN_{\triangle t} = Z(t)$ - $Z(t-1)$. The data traffic model also captures the packaging phenomenon by accommodating the $A_{\triangle t}$ bytes in packets with size equals the MTU (Maximum Transmit Unit), except for the last one whose size can vary between the minimum packet size and MTU. Figure A.1 illustrates the traffic generation process by plotting $Z(t)$, $FGN(t)$ and the number of packets for an one second simulation, for $\triangle t = 10ms$, $H = 0.76$, $a = 275kbit.s$ and $m = 8Mbps$.

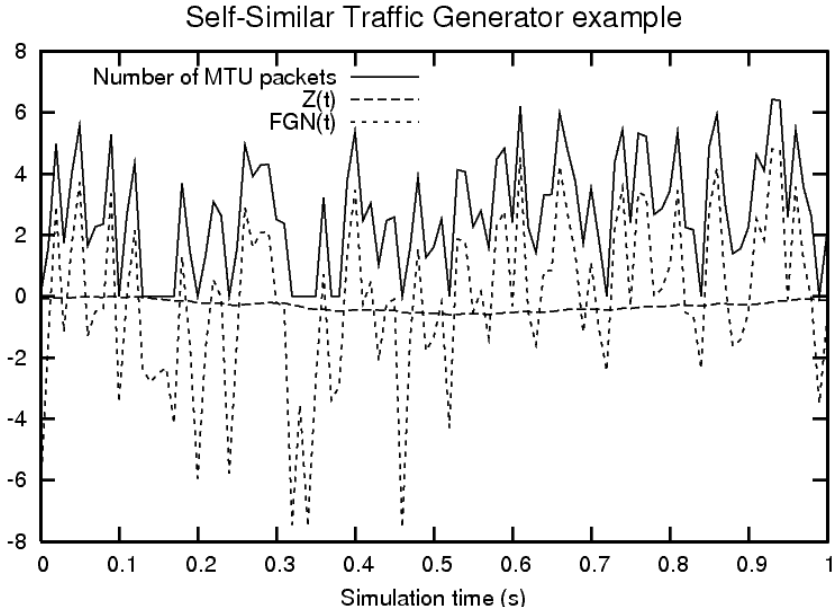

Figure A.1: Example: Self-Similar Traffic generation for an one second simulation

```
i = 1;
seq = 0;
Z_old = 0;
while i < t_SIM do
    FGN_△t = Z(i) − Z_old;
    A_△t = m.△t + √(m.a).FGN_△t;
    Z_old = Z(i);
    N_pkts = floor(A_△t / MTU);
    remainder = fmod(A_△t, MTU);
    t_pkt = △t / (N_pkts + 1);
    timeac = 0;
    if remainder == 0 then
        t_pkt − = 1;
    end
    while N_pkts ≥ 1 do
        pkt = (pkt*)msg_create(sizeof(pkt));
        pkt− > birth = time_now();
        pkt− > seq = seq + +;
        pkt− > bytes = MTU;
        pkt− > flowId = SELF_SIMILAR_ID;
        msg_send(pkt, destID);
        task_sleep(t_pkt);
        timeac+ = t_pkt;
        N_pkts − −;
    end
    if remainder > 0 then
        if remainder < MIN_PKT_SIZE then
            remainder = MIN_PKT_SIZE;
        end
        pkt = (pkt*)msg_create(sizeof(pkt));
        pkt− > birth = time_now();
        pkt− > seq = seq + +;
        pkt− > bytes = remainder;
        pkt− > flowId = SELF_SIMILAR_ID;
        msg_send(pkt, destID);
    end
    task_sleep(△t − timeac);
    i + +;
end
```

Algorithm 5: Self Similar Traffic Generator

where:

t_{SIM}: Total simulation time.

$Z(i)$: Normalized fractional brownian motion value for $t = i$ calculated via the Hosking method.

N_{pkts}: Number of packets.

MTU: MTU size.

t_{pkt}: Time between packets.

$timeac$: Accumulated time.

$destID$: Number that indicates the destination entity of the packet. It can assume: $nodeID$, $leakyID$ or $tokenID$ for, respectively, the network node, the leaky bucket or the token bucket.

A.3 Leaky Bucket Module

The Leaky Bucket is controlled by two parameters: the bucket size b and the transmit rate r, as illustrated in Figure A.2a. The bucket has a queue with size b to store the packets containing the self similar traffic, which are served according to a committed rate r. Whenever a packet arrives and there is no room for it in the queue it is classified as a non-conforming packet and tagged as BE, whereas the packets in the queue are classified as conforming and are tagged as AF. Figure A.2b presents an example of the Leaky Bucket metering and marking scheme. In this figure, we considered the average load of self-similar traffic $m = 8Mbps$, the bucket rate $r = 8Mbps$ and the bucket size $b = 10MTU$ packets. The figure presents the results of the metering/marking processes in terms of the number of packets, where each point indicates a 10ms sample of the simulation.

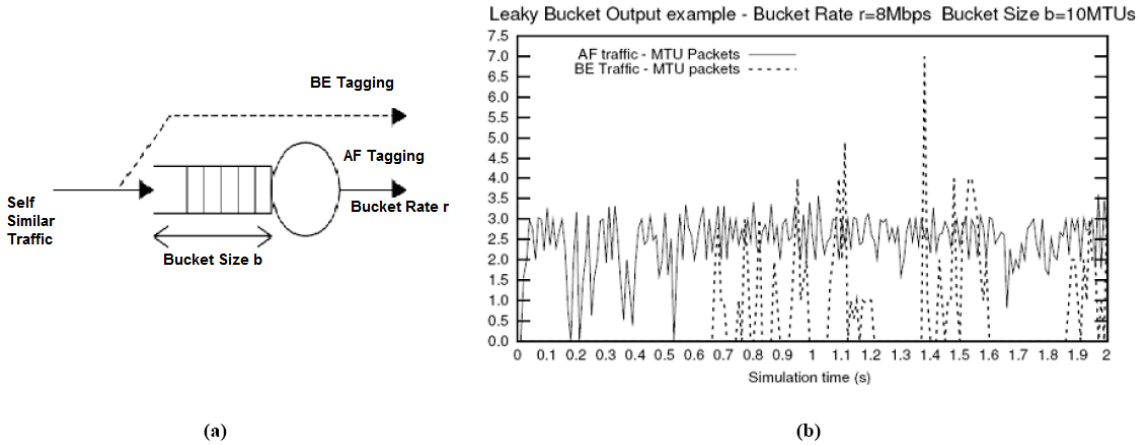

Figure A.2: Leaky Bucket Module

```
while TRUE do
    pkt = (pkt*)msg_recv() ;
    if (queueCurrentSize + pkt->size) ≤ b then
        queueCurrentSize += pkt->size ;
        pkt->queue = AF ;
        putLeakyQueue(pkt);
        task_activate(leakyService, 0);
    else
        pkt->queue = BE ;
        msg_send(pkt, nodeID);
    end
end
```

Algorithm 6: Leaky Bucket Module: Marker

where:

queueCurrentSize: Current leaky bucket queue size.

AF: AF tagging code.

BE: BE tagging code.

putLeakyQueue(): Inserts a packet into the leaky bucket queue.

leakyService: Entity responsible (i.e., Dispatcher) for serving the packets of the leaky bucket queue.

```
while TRUE do
    if queueCurrentSize > 0 then
        pkt = (pkt*)getLeakyQueue() ;
        queueCurrentSize -= pkt->size;
        time = (pkt->bytes)/r;
        task_sleep(time);
        msg_send(pkt, nodeID);
    else
        task_passivate();
    end
end
```

Algorithm 7: Leaky Bucket Module: Dispatcher

where:

getLeakyQueue(): Remove and obtain a packet from the leaky bucket queue.

A.4 Token Bucket Module

The Token Bucket is also controlled by two parameters: the bucket size b and the token rate r. When a packet arrives and there exists enough tokens in the bucket it is tagged as

AF. Conversely, whenever a packet arrives and there is no enough tokens in the bucket it is classified as a non-conforming packet and tagged as BE. We have two entities (threads) that implement the token bucket module: The first, called $tokenControl$, is responsible for incrementing the token counter, whereas the second, called $tokenBucket$, is the main entity that is responsible for packet classification and dispatching.

```
while TRUE do
    pkt = (pkt*)msg_recv() ;
    if currentTokenNumber ≥ pkt->size then
        currentTokenNumber -= pkt->size ;
        pkt->queue = AF ;
    else
        pkt->queue = BE ;
    end
    msg_send(pkt, nodeID);
end
```

Algorithm 8: Token Bucket Module: tokenBucket marker and dispatcher

where:
$currentTokenNumber$: Current number of tokens in the bucket.

```
while TRUE do
    if (currentTokenNumber + MTU) < b then
        currentTokenNumber+ = MTU ;
    else
        currentTokenNumber = b ;
    end
    task_sleep(MTU/r);
end
```

Algorithm 9: Token Bucket Module: token control

A.5 Network Node

The Network Node is implemented considering two active entities: The first thread, called $classifier$ receives the packet and inserts it into the corresponding queue. The second, called $prio\_sch$, is the link scheduler that implements the strictly priority queue scheduling policy.

```
while TRUE do
    pkt = (pkt*)msg_recv() ;
    if (queueCurrentSize_{pkt->queue} + pkt->size) ≤ queueLength_{pkt->queue}
    then
        queueCurrentSize_{pkt->queue} += pkt->size;
        putQueue_{pkt->queue}(pkt);
        pkt->status = RECEIVED;
        task_activate(prio_sch);
    else
        pkt->status = DROPPED;
    end
end
```

Algorithm 10: Network Node: classifier

where:

$queueCurrentSize$: Gives the current size of the corresponding queue $pkt->queue$.

$putQueue()$: Inserts the packet into the correponding queue $pkt->queue$.

$RECEIVED$: Code for a received packet by the network node.

$DROPPED$: Code for a dropped packet by the network node.

```
queuePriority = 0;
while TRUE do
    if queueCurrentSize_{queuePriority} > 0 then
        pkt = (pkt*)getQueue_{queuePriority}() ;
        queueCurrentSize_{queuePriority} -= pkt->size;
        task_sleep(pkt->size / C);
        msg_send(pkt, logger);
    end
    queuePriority++;
    if queuePriority > QUEUES then
        queuePriority = 0;
        task_passivate();
    end
    queuePriority = queueVerify();
end
```

Algorithm 11: Network Node: prio scheduler

where:

$queuePriority$: A number that indicates the queue priority: 0=EF, 1=AF and 2=BE.

$queueCurrentSize$: Gives the current size of the corresponding queue $queuePriority$.

$getQueue()$: Removes and obtains a packet from the corresponding queue $queuePriority$.

C: Capacity of the output link.

$QUEUES$: Total number of queues in the network node.

logger: Thread that logs all packet information for processing.

$queueVerify$: Gives a number corresponding to the highest priority queue which has $queueCurrentSize > 0$.

Publications

[Nabhen et al. 2006] Nabhen, R., Jamhour, E., Penna, M. C., and Fonseca, M. S. (2006). Analysis of individual flows performance for delay sensitive applications. *19TH IFIP World Computer Congress/TC6,5th International Conference on Network Control and Engineering NETCON, 2006*, 1:143–156.

[Nabhen et al. 2007a] Nabhen, R., Jamhour, E., Penna, M. C., Fonseca, M. S., and Pujolle, G. (2007a). Avaliando estratégias de controle de admissão para implementação de garantias de qos de fluxos individuais em redes diffserv usando métodos de otimização. *SBRC - Simpósio Brasileiro de Redes de Computadores, 2007, Belém. SBRC 2007 - 25 Simpósio Brasileiro de Redes de Computadores, 2007*, 1:811–824.

[Nabhen et al. 2007b] Nabhen, R., Jamhour, E., Penna, M. C., Fonseca, M. S., and Pujolle, G. (2007b). Diffserv pbac design with optimization method. *7th IEEE International Workshop on IP Operations and Management, IPOM, 2007*, 1:73–84.

[Nabhen et al. 2007c] Nabhen, R., Jamhour, E., Penna, M. C., Fonseca, M. S., and Pujolle, G. (2007c). Optimal diffserv ac design using non-linear programming. *32nd IEEE Conference on Local Computer Networks, LCN, 2007*, 1:1–8.

[Nabhen et al. 2008] Nabhen, R., Jamhour, E., Penna, M. C., and Pujolle, G. (2008). Modeling a multi-queue network node with a fuzzy predictor. *Journal: Fuzzy Sets and Systems. doi:10.1016/j.fss.2008.12.004.*

[Nabhen et al. 2009] Nabhen, R., Jamhour, E., Penna, M. C., and Pujolle, G. (2009). A time based fuzzy model for mva approximation for loss probability. *Fuzzy Sets and Systems. (Under Revision).*

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," *IETF RFC 2475*, Dec. 1998.

[2] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (RSVP) – version 1 functional specification," *IETF RFC 2205*, Sept. 1997.

[3] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," *IETF RFC 3031*, Jan. 2001.

[4] F. L. Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, and J. Heinanen, "Multi-Protocol Label Switching (MPLS) Support of Differentiated Services," *IETF RFC 3270*, May 2002.

[5] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels," *IETF RFC 3209*, Dec 2001.

[6] J. Wroclawski, "The Use of RSVP with IETF Integrated Services," *IETF RFC 2210*, Sep 1997.

[7] M. Zukerman, T. D. Neame, and R. G. Addie, "Internet traffic modeling and future technology implications," *Proceedings of IEEE INFOCOM*, Mar 2003.

[8] A. Adas, "Traffic models in broadband networks," *IEEE Communications Magazine*, vol. 35, pp. 82–89, Jul 1997.

[9] E. W. Knightly and N. B. Shroff, "Admission control for statistical qos: theory and practice," *Network, IEEE*, vol. 13, no. 2, no. 2, pp. 20–29, 1999.

[10] L. Breslau, S. Jamin, and S. Shenker, "Comments on the performance of measurement-based admission control algorithms," *INFOCOM (3)-19th Annual Joint Conference of the IEEE Computer and Communications Societies*, pp. 1233–1242, 2000.

[11] L. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. SMC-3, pp. 28–44, 1973.

[12] R. Nabhen, E. Jamhour, M. C. Penna, and M. S. Fonseca, "Analysis of individual flows performance for delay sensitive applications," *19TH IFIP World Computer Congress/TC6,5th International Conference on Network Control and Engineering NETCON, 2006*, vol. 1, pp. 143–156, Aug 2006.

[13] R. Nabhen, E. Jamhour, M. C. Penna, M. S. Fonseca, and G. Pujolle, "Avaliando estratégias de controle de admissão para implementação de garantias de qos de fluxos individuais em redes diffserv usando métodos de otimização," *SBRC - Simpósio Brasileiro de Redes de Computadores, 2007, Belém. SBRC 2007 - 25 Simpósio Brasileiro de Redes de Computadores, 2007*, vol. 1, pp. 811–824, May 2007.

[14] R. Nabhen, E. Jamhour, M. C. Penna, M. S. Fonseca, and G. Pujolle, "Optimal diffserv ac design using non-linear programming," *32nd IEEE Conference on Local Computer Networks, LCN, 2007*, vol. 1, pp. 1–8, Oct 2007.

[15] R. Nabhen, E. Jamhour, M. C. Penna, M. S. Fonseca, and G. Pujolle, "Diffserv pbac design with optimization method," *7th IEEE International Workshop on IP Operations and Management, IPOM, 2007*, vol. 1, pp. 73–84, Nov 2007.

[16] R. Nabhen, E. Jamhour, M. C. Penna, and G. Pujolle, "Modeling a multi-queue network node with a fuzzy predictor," *Journal: Fuzzy Sets and Systems. doi:10.1016/j.fss.2008.12.004*, Dec 2008.

[17] L. A. Zadeh, "Fuzzy sets," *Information Control*, vol. 8, pp. 338–353, 1965.

[18] S. N. Sivanandam, S. Sumathi, and S. N. Deepa, "Introduction to fuzzy logic using matlab," *Springer-Verlag New York, Inc.*, Oct 2006.

[19] I. T. Union, "ITU-T recommendation G.114," *Series G: Transmission Systems and Media, Digital Systems and Networks*, May 2003.

[20] J. Jantzen, "Design of Fuzzy Controllers," *Technical University of Denmark: Dept. of Automation*, 1998.

[21] K. M. Passino and S. Yurkovich, "Fuzzy control," *Addison-Wesley Longman Publishing Co., Inc.*, 1997.

[22] W. V. Leekwijck and E. E. Kerre, "Defuzzification: criteria and classification," *Fuzzy Sets and Systems*, vol. 108, no. 2, no. 2, 1999.

[23] J. Mendel and G. Mouzouris, "Designing Fuzzy Logic Systems," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 44, pp. 885–895, Nov 1997.

[24] G. Mouzouris and J. A. Mendel, "Non-singleton fuzzy logic systems: theory and applications," *IEEE Trans. on Fuzzy Systems*, vol. 5, pp. 56–71, Feb 1997.

[25] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *Int. J. Hum.-Comput. Stud.*, vol. 51, no. 2, no. 2, 1999.

[26] J. M. Mendel, "Fuzzy logic systems for engineering: a tutorial," *Proceedings of the IEEE on Signal & Image Process. Inst., Univ. of Southern California, Los Angeles, CA*, vol. 83, Mar 1995.

[27] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Transaction on Systems, Man. and Cybernetics*, vol. 15, pp. 115–132, 1985.

[28] H. Takagi, "Introduction to fuzzy systems, neural networks, and genetic algorithms," *In: Ruan, D. ed. Intelligent hybrid systems. Fuzzy logic, neural networks, and genetic algorithms. 1st ed. Kluwer Academic Publishers*, pp. 3–33, 1997.

[29] J. Cortajarena, J. D. Marcos, P. Alvarez, F. Vicandi, and P. Alkorta, "Indirect vector controlled induction motor with four hybrid p+fuzzy pi controllers," *IEEE International Symposium on Industrial Electronics*, vol. 4, no. 7, pp. 197–202, Jun 2007.

[30] A. V. Patel, "Simplest fuzzy pi controllers under various defuzzification methods," *International Journal of Computational Cognition*, vol. 3, no. 1, Mar 2005.

[31] D. P. Filev and R. R. Yager, "A Generalized Defuzzification Method via bad Distributions," *International Journal of Intelligent Systems*, vol. 6, pp. 687–697, 1991.

[32] Z. Sahinoglu and S. Tekinay, "On multimedia networks: self-similar traffic and network performance," *IEEE Communications Magazine*, vol. 37, pp. 48–52, Jan 1999.

[33] C. Boutremans, G. Iannaccone, and C. Diot, "Impact of link failures on voip performance," *International workshop on network and operating systems support for digital audio and video*, pp. 63–71, 2002.

[34] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, pp. 226–244, Jun 1995.

[35] M. Crovella and A. Bestavros, "Wide-area traffic: The failure of poisson modeling," *Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes*, vol. 5, pp. 835–846, Dec 1997.

[36] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking*, vol. 2, pp. 1–15, Feb 1994.

[37] W. Willinger and K. Park, "Self-similar network traffic and performance evaluation," *John Wiley & Sons*, 2000.

[38] I. Norros, "On the use of fractional Brownian motion in the theory of connectionless networks, journal = IEEE Journal on Selected Areas in Communications, volume = 16 , issue = 6, pages =,"

[39] J. R. M. Hosking, "Modeling persistence in hydrological time series using fractional brownian differencing," *Water Resources Research*, vol. 20, 1984.

[40] G. Box, G. Jenkins, and G. G. Reinsel, "Time series analysis: Forecasting and control," *Prentice Hall*, Feb 1994.

[41] J. Feder, "Fractals," *Plenum Press*, 1988.

[42] M. Taqqu, V. Teverovsky, and W. Willinger, "Estimators for long-range dependence: An empirical study," *Fractals*, vol. 3, pp. 785–798, 1995.

[43] C. Systems, "Traffic analysis for voice over ip," *Cisco Document Server*, Sep 2002.

[44] "The Network Simulator ns-2 (v2.1b8a)." http://www.isi.edu/nsnam/ns/, Oct 2001.

[45] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, "Advances in network simulation," *IEEE Computer*, vol. 33, no. 5, pp. 59–67, May 2000.

[46] G. Kesidis and J. Walrand, "Quick simulation of atm buffers with on-off multiples markov fluid sources," *ACM Trans. Modeling and Computer Simulations*, vol. 3, no. 3, pp. 269–276, Jul 1993.

[47] C. Maziero and R. Nabhen, "Simpatica discrete event simulator," *http://www.ppgia.pucpr.br/~maziero/doku.php/software:simulation*, 2007.

[48] G. Agha, "Actors: A model of concurrent computation in distributed systems. doctoral dissertation," *MIT Press*, Jun 1985.

[49] G. Agha, I. Mason, S. Smith, and C. Talcott, "A foundation for actor computation," *Journal of Functional Programming*, vol. 7, no. 1, pp. 1–69, Jan 1997.

[50] R. Jain, "The art of computer systems performance analysis," *John wiley and Sons*, 1991.

[51] W. M. McCormack and R. G. Sargent, "Comparison of future event set algorithms for simulations of closed queuing systems," *Current issues in Computer Simulation, Academic Press*, 1979.

[52] C. M. Reeves, "Complexity analyses of event set algorithms," *The Computer Journal*, vol. 27, 1984.

[53] S. Boyd and L. Vandenberghe, "Convex optimization," *Cambridge University Press*, 2004.

[54] J. Mitchell, "Geometric shortest paths and network optimization," *Elsevier Science*, 1998.

[55] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *J. ACM*, vol. 35, no. 4, no. 4, pp. 921–940, 1988.

[56] D. A. Grundel, P. Krokhmal, C. A. Oliveira, and P. M. Pardalos, "On the average case behavior of the multidimensional assignment problem," *Pacific Journal of Optimization*, vol. 1, no. 1, no. 1, pp. 39–57, 2005.

[57] P. D. Bertsekas, "Nonlinear programming," *Athena Scientific*, 1999.

[58] P. D. Bertsekas, A. Nedic, and A. E. Ozdaglar, "Convex analysis and optimization," *Athena Scientific*, 2003.

[59] D. M. Himmelblau, "Applied nonlinear programming," *McGrawHill*, 1972.

[60] E. Jamhour, "Commande douce de systèmes mécaniques: Optimisation de trajectories sous diverses contraintes," *L'U. F. R. des Sciences et des Techniques de L'Université de Franche-Comté*, vol. 1, 1994.

[61] P. Ciarlet, "Introduction à l'analyse numérique matricielle et à l'optimisation," *Ed. Masson, Paris*, 1990.

[62] R. Bhowmik, "Building design optimization using sequential linear programming," *IEEE SoutheastCon*, vol. 22, no. 25, pp. "498–502", Mar 2007.

[63] J. M. Box, "A new method of constraint optimization and a comparison with other methods," *Computer Journal*, vol. 8, pp. 42–52, 1965.

[64] J. H. Holland, "Adaptation in natural and artificial systems, an introductory analysis with application to biology, control and artificial intelligence," *The University of Michigan Press*, 1975.

[65] M. Avriel, "Nonlinear programming: Analysis and methods," *Dover Publishing*, 2003.

[66] J. A. Nelder and R. Mead, "A simplex method for function minimization," vol. 7, pp. 308–313, 1965.

[67] K. I. M. McKinnon, "Convergence of the nelder–mead simplex method to a nonstationary point," *SIAM J. on Optimization*, vol. 9, no. 1, no. 1, pp. 148–158, 1998.

[68] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence properties of the Nelder-Mead Simplex method in low dimensions.," *SIAM J. Optimization*, vol. 9, no. 1, no. 1, pp. 112–147, 1998.

[69] C. Audet and J. E. D. Jr., "Analysis of generalized pattern searches," *SIAM J. Optimization*, vol. 13, pp. 889–903, 2003.

[70] A. W. Moore, "An implementation-based comparison of measurement-based admission control algorithms," *Journal of High Speed Networks*, vol. 13, pp. 87–102, 2004.

[71] P. Siripongwutikorn and S. Banerjee, "Per-flow delay performance in traffic aggregates," *Proceedings of IEEE Globecom 2002*, vol. 3, pp. 2634–2638, Nov 2002.

[72] Y. Xu and R. Guerin, "Individual qos versus aggregate qos: A loss performance study," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 370–383, Apr 2005.

[73] S. Spitler and D. C. Lee, "Optimization of call admission control for a statistical multiplexer allocating link bandwidth," *IEEE Transactions on Automatic Control*, vol. 48, pp. 1830–1836, Oct 2003.

[74] R. Guerin, H. Ahmadi, and M. Naghshineh, "Equivalent capacity and its application to bandwidth allocation in high-speed networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 968–981, Sep 1991.

[75] R. Li and E. Lee, "Analysis of fuzzy queues," *Computer and Mathematics with Applications*, vol. 17, pp. 1143–1147, 1989.

[76] S. Chen, "Parametric nonlinear programming approach to fuzzy queues with bulk service," *European Journal of Operational Research*, vol. 163, pp. 434–444, 2005.

[77] J. Ke and C. Lin, "Fuzzy analysis of queue systems with an unreliable service: a nonlinear programming approach," *Applied Mathematics and Computation*, vol. 175, pp. 330–346, 2006.

[78] M. J. Pardo and D. la Fuente, "Optimizing a priority discipline queuing model using fuzzy set theory," *Computer Mathematics with Applications*, vol. 54, pp. 267–281, 2007.

[79] B. Chen, Y. Yang, B. Lee, and T. Lee, "Fuzzy adaptive predictive flow control of atm network traffic," *IEEE Transactions on Fuzzy Systems*, vol. 11, pp. 568–581, 2003.

[80] W. Guo and H. Hu, "A self-adapted wavelet-based fuzzy predictor of network traffic," *IEEE International Conference on Computing*, pp. 367–372, Nov 2006.

[81] C. Hsiao and S. Su, "An on-line fuzzy predictor from real-time data," *IEEE International Fuzzy Systems Conference*, pp. 1–5, Jul 2007.

[82] N. M. Din and N. Fisal, "Fuzzy logic token bucket bandwidth predictor for assured forwarding traffic in a diffserv-aware mpls internet," *IEEE Asia International Conference on Modeling and Simulation*, pp. 247–252, Mar 2007.

[83] J. Babiarz, K. Chan, and F. Baker, "Configuration guidelines for diffserv service classes," *IETF - RFC 4594*, 2006.

[84] J. Seger, "Modeling approach for VoIP traffic aggregations for transferring tele-traffic trunks in a QoS enabled IP-backbone environment," *International Workshop on Inter-domain Performance and Simulation*, Feb 2003.

[85] J. Wroclawski, "Specification of the Controlled-Load Network Element," *IETF RFC 2211*, Sep 1997.

[86] P. Kumar and R. Garg, "Fuzzy model of corona current signals for ehv lines," *Institution of Engineers of India, Electrical Engineering Division*, vol. 86, Mar 2006.

[87] H. Kim and N. Shroff, "Loss probability calculations and asymptotic analysis for finite buffer multiplexers," *IEEE/ACM Trans. Networking*, vol. 9, no. 6, no. 6, pp. 755–768, 2001.

[88] D. P. Filev and R. R. Yager, "Defuzzification Under Constraints and Forbidden Zones," *Kybernetes*, vol. 23, pp. 43–57, 1994.

[89] G. Vaudreuil and G. Parsons, "Toll quality voice-32 kbit/s adpcm mime sub-type registration," *IETF-RFC 2422*, Sep 1998.

[90] G. Procissi, A. Garg, M. Gerla, and M. Y. Sanadid, "Token bucket characterization of long range dependent traffic," *Computer Communications*, vol. 25, Jan. 2002.

[91] R. Guerin and M. Naghshine, "Equivalent capacity and its application to bandwidth allocation in high speed networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 7, no. 7, pp. 968–981, 1991.

List of acronyms

AC	*Admission Control*
AF	*Assured Forwarding*
AFD	*Average Flow Duration*
API	*Application Programming Interface*
AS	*Autonomous System*
BE	*Best Effort*
CBR	*Constant Bit Rate*
CIR	*Committed Information Rate*
DF	*Default Forwarding*
DiffServ	*Differentied Services*
DSCP	*Differentiated Services Code Point*
EF	*Expedited Forwarding*
FBM	*Fractional Brownian Motion*
FBt	*Fractional Brownian Motion Traffic*
FLS	*Fuzzy Logic System*
FP	*Flexible Polyhedron*
IETF	*Internet Enginneering Task Force*
IntServ	*Integrated Services*
IP	*Internet Protocol*
ISP	*Internet Service Provider*
ITU-T	*International Telecommunication Union - Telecommunication Standardization Sector*
LAN	*Local Area Network*
LER	*Label Edge Router*
LIP6	Laboratoire d'Informatique de Paris VI

LP	*Linear Programming*
LRD	*Long Range Dependence*
LSP	*Label Switched Path*
LSR	*Label Switching Router*
MBAC	*Measurement Based Admission Control*
MF	*Membership Function*
MFP	*Membership Flexible Polyhedron*
MPLS	*Multiprotocol Label Switching*
MTU	*Maximum Transfer Unit*
MVA	*Maximum Variance Asymptotic*
NLP	*Non-Linear Programming*
NS-2	*Network Simulator 2*
PBAC	*Parameter Based Admission Control*
PIR	*Peak Information Rate*
RFC	*Request For Comments*
RSVP	*Resource Reservation Protocol*
RSVP-TE	*Resource Reservation Protocol - Traffic Engineering*
SFP	*Simulation Flexible Polyhedron*
SLA	*Service Level Agreement*
SRD	*Short Range Dependence*
TBF	*Time Between Flows*
T-conorm	*Triangular conorm*
TCP	*Transmission Control Protocol*
Tspec	*Traffic Specification*
T-norm	*Triangular norm*
TOS	*Type of Service*
TS	*Takagi-Sugeno*
VAD	*Voice Activity Detection*
VoIP	*Voice over IP*
VPN	*Virtual Private Network*

List of figures

List of tables