Fast Overflow Probability Estimation Tool for MPLS Networks

D. Buschiazzo, A. Ferragut, A. Vázquez, P. Belzarena
{dbusch, aferragu, alevaz, belza}@fing.edu.uy
*ARTES**
FACULTAD DE INGENIERÍA, UNIVERSIDAD DE LA REPÚBLICA
MONTEVIDEO, URUGUAY .

***Abstract*— The constant growth of internet and the variety of services provided makes the estimation of QoS parameters a fundamental need for every Internet Service Provider. The present work introduces a software tool that calculates the overflow probability on the core links of a MPLS network. The calculation is based on the statistical properties of the arriving traffic and the routing on the network. The procedure uses the results of the large deviations theory and the work of Likhanov et al. [17] for small buffer. The results obtained show high degree of accuracy as well as very short processing times. This allows the user to determine the overflow status of the network without the need to use the traditional highly time consuming simulation techniques.**

I. Introduction

Traffic Engineering (TE) is a main research area in Telecommunications and particularly in the Internet because by using TE the network can be optimized and the quality of service of the network can be improved.

Traffic Engineering maps the traffic over the network topology in order to optimize the performance, by means of an even distribution of the traffic flows over the network. Moreover another objective to be achieved is to enhance performance indexes, such as overflow probability, loss probability and end to end delay.

MPLS (MultiProtocol Label Switching)[21] is a network architecture standardized by the IETF in 1999, that enables to perform traffic engineering in IP networks. MPLS introduces the notion of Forwarding Equivalence Class (FEC), giving the network operator the possibility to split the traffic in aggregated flows according to the service model adopted by the Internet Service Provider (ISP).

This research was partially supported by PDT (Programa de Desarrollo Tecnológico), Préstamo 1293/OC-UR: S/C/OP/17/02, S/C/OP/17/03 and program FCE (Fondo Clemente Estable) 8079

*__ARTES__: Joint Research Group of the Electrical Engineering and Mathematics and Statistics Departments.**Contact:** artes@fing.edu.uy

The edge routers in an MPLS network (or LER for Label Edge Router) are responsible for establishing MPLS tunnels named LSPs (Label Switched Path) between the endpoints of the MPLS domain, and to send each arriving packet to the corresponding LSP.

Explicit Routing (ER), a typical MPLS feature, is the main function that enables Traffic Engineering (TE). Using ER the network operator can establish for each FEC one or more LSPs.

In order to guarantee end to end QoS in a MPLS network, a performance model is required. This model provides methods to analyze the end to end QoS performance parameters.

Likhanov et al. [17] developed a model that can be used for performance evaluation and traffic engineering, which is useful to analyze the overflow probability in a core network link, in the many sources and small buffer asymptotic. Based on such model, we developed a software tool that computes a core link's overflow probability. In our context, the overflow probability represents the stationary probability that the arrival rate to a given link is larger that the link's capacity, that on the small buffer (or bufferless multiplexing) context leads to losses. Under the typical hypothesis of ergodicity it can be considered as the time proportion in which the link cannot process the arriving traffic.

The work of Likhanov et al. is briefly explained in section II. In section III we present the main problems that need to be solved in order to develop a software tool that implements the main results of Likhanov's work. In section IV the solutions of the previous problems and the software architecture are explained. In section V the results obtained with our tool are validated with network simulations. Finally the conclusions are presented in section VI.

II. Network model and performance estimators

A. Introduction

Our goal is to find fast estimations of performance metrics for all links on the network by means of knowing the network's arriving traffic. We will focus on the estimation of overflow probability over a network path, and some recent results from large deviations theory will be used. This theory allows the use of use more general traffic models than traditional queuing models.

The large deviations theory applied to network performance analysis gives asymptotic results for links that are fed by a large number of traffic sources and where the link's resources (buffer size, link capacity, etc.) are scaled according to the number of sources. There are different types of asymptotic regimes useful in different kind of networks depending on the type of resource scaling used.

To model an Internet backbone the regime presented by Likhanov et al. on [17] is very useful. The main result of this work allows to analyze the overflow probability in any link of the network, with an independence hypothesis between the aggregate traffic flows feeding the network. The regime studied is called "many sources and small buffer asymptotic", the results of which we will explain briefly in the following paragraphs.

From now on we will work on discrete time. The traffic arrival rate at the edge nodes, is treated as a stationary and ergodic stochastic process from the superposition of N independent traffic sources with the same distribution. This process is represented by the variable X_m with $m = 1, \ldots, M$, being M the number of paths defined on the network. We shall define $\rho_m = \mathbf{E}(X_m)$, the mean arrival rate of the traffic m. Thus, X_m represents a typical traffic arrival on a time unit of the m flow and X_m^N is the superposition of N independent copies of X_m.

A relevant issue when working on discrete time, is the choice of sampling unit. The software will estimate the probability that a buffer is saturated on any (discrete) time unit, but the choice of this unit is not strictly arbitrary, and has to be chosen carefully. The choice of the time unit has to be adequate to the dynamics governing the sources we will be dealing with. We should choose a unit as small as possible, in order to avoid loosing any dynamic phenomena of the sources considered (i.e. we should be able to observe bursts). However, the unit can not be made as small as desired, in order to avoid the drifting from the fluid model we are working on (avoid the issued posed by packetization). If the mean rate from the source was a, and the packet size is b, then a reasonable time unit choice would be within the order of b/a.

Large deviations theory states, under certain mild hypothesis, that for an aggregated flow, the following proposition is valid:

$$\begin{aligned} -\inf_{x\in B^\circ} I_m(x) &\leq \liminf_{N\to\infty} \frac{1}{N}\log \mathbf{P}(X_m^N/N \in B) \leq \\ &\leq \limsup_{N\to\infty} \frac{1}{N}\log \mathbf{P}(X_m^N/N \in B) \leq \\ &\leq -\inf_{x\in \bar{B}} I_m(x) \end{aligned} \tag{II.1}$$

for all $B \subset \mathbb{R}$, where $I_m : \mathbb{R} \to [0, \infty]$ is the function called rate function of the traffic of type m, which is related to the traffic statistical characteristics by:

$$I_m(x) = \sup_s \{sx - \Lambda_m(s)\} \tag{II.2}$$

being $\Lambda_m(s)$ the log-laplace transform of the X_m variable:

$$\Lambda_m(s) = \log \mathbf{E}\left(e^{sX_m}\right) \tag{II.3}$$

In most cases, the latter inequality becomes an equality, which allows us to write:

$$\mathbf{P}\left(X_m^N/N \in B\right) \approx e^{-nI_m(B)}$$

being $I_m(B) = \inf_{x\in B} I_m(x)$. This is a typical result of the large deviations theory where we approximate a probability by an exponential term. In this case exponential coefficient is obtained as the result of an optimization problem over the rate function of the studied variable. The rate function I_m carries the statistical information of the aggregated flow of type X_m.

The next step is to model the network structure. For that purpose, we will assume that the network has K links. Each link $k = 1, \ldots, K$ has a NC_k capacity that grows linearly with the number of sources (many sources hypothesis). We also assume that buffer receive traffic with FIFO policy and without priorities. The buffer size is $B_k(N)$ with $B_k(N)/N \to 0$ when $N \to \infty$ (small buffer hypothesis). Every path is an ordered group of links $\mathbf{k}_m = (k_m^{(1)}, \ldots, k_m^{(l_m)})$.

It is also necessary to assume a stability hypothesis. Defining the group of traffic types that goes through link k by $\mathcal{M}_k = \{m : k_i^m = k \text{ for any } i\}$ we shall assume that

$$\sum_{m\in\mathcal{M}_k} \rho_m < C_k \tag{II.4}$$

This is a basic hypothesis in order to apply the large deviations theory, because in case the mean work arrival rate is greater than the link's capacity it would lead to an unstable situation. In this case the fact that the link

presents losses would no longer be a rare event and therefore is out of scope of this theory.

The main contribution of Likhanov et al. en [17] is the introduction of the *transfer function* concept. Under the previous hypothesis the following proposition can be proofed:

Proposition 2.1: There exists a continuous function $g_{m,k} : \mathbb{R}^M \to \mathbb{R}$ such that the traffic arrival rate of kind m to link k, noted $X_{m,k}$ satisfies:

$$X_{m,k}/N = g_{m,k}(X_1/N, \ldots, X_M/N) + o(1) \quad \text{(II.5)}$$

That is, with a difference of an infinitesimal, the normalized traffic arrival rate at any given link, can be expressed as a continuous function of the traffic arrival rate for every traffic type entering the network. In (II.5), $o(1)$ is an error term due to the small buffers, such that $\lim_{N\to\infty} o(1) = 0$ since $\frac{B_k(N)}{N} \underset{N\to\infty}{\to} 0$.

The demonstration of the latter proposition includes the explicit construction of the function $g_{m,k}$ for feed forward networks. It is based on the composition of functions as:

$$f_m(x_1, \ldots, x_M, C_k) = \frac{x_m C_k}{\max\left\{\sum_{i=1}^{M} x_i, C_k\right\}} \quad \text{(II.6)}$$

that relate the input and output traffic arrival rates at any link. The meaning of II.6 can be depicted as follows: if at any given link, there is a superposition of different traffic aggregates with rates $x_1, \ldots, x_M$ and that link has capacity C_k (per source), then, the traffic of type m is x_m in case the capacity is greater than the total traffic arrival rate ($\sum_{i=1}^{M} x_i$). On the other hand, if the capacity is not enough to serve the total traffic arrival rate, the output of type m traffic would be a proportion of its input rate, distributed evenly over every traffic traversing the link. Composing this functions allows to determine the contribution of a certain traffic flow at each link. The infinitesimal term $o(1)$ corresponds to the data that is stored on the link's buffers. The fact that buffers are small makes their contribution less significant.

Following from the previous statements, we can deduct:

Corollary 2.1: There exists a continuous function g_k, such that, up to an infinitesimal term, allows to calculate the total traffic arrival rate to a link, using only the information of the traffic arrival rate at the edge nodes of the network. This function corresponds to the sum of the $g_{m,k}$ over every traffic that traverses the link, and will be denominated *transfer function*:

$$g_k(x) = \sum_{m \in \mathcal{M}_k} g_{m,k}(x)$$

The existence of such function allows to obtain the large deviation rate of the buffers's content, using the contraction principle. The main result of this theory can be stated as follows:

Theorem 2.1: Under the previously mentioned hypothesis, we have:

$$\lim_{N\to\infty} \frac{1}{N} \log \mathbf{P}(\text{link overflow at } k) = -\mathbf{I}_k \quad \text{(II.7)}$$

being:

$$\mathbf{I_k} = \inf\left\{\sum_{m=1}^{M} I_m(x_m) : g_k(x_1 \ldots x_M) > C_k\right\} \quad \text{(II.8)}$$

where $I_m(x)$ represents the rate function of traffic m, g_k is the transfer function to the link, and C_k is the per source capacity of the studied link.

To summarize, using only the statistic data of the flows entering the network, given by I_m and using g_k, that relates the traffic arrival rate at link k to the edge arrival rates, it is possible to estimate the link's overflow probability as $e^{-N\mathbf{I_k}}$.

To deal with the previous problem is necessary to solve an optimization problem of the form:

$$(P)\begin{cases} \min f(x) \\ \text{constrained to } g(x) \leq 0 \end{cases} \quad \text{(II.9)}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}$ y $x \in \mathbb{R}^n$

Where f is the objective function $\left(\sum_{m=1}^{M} I_m(x_m)\right)$ and g is the constraint function $(g_k(x_1 \ldots x_M))$.

On the same work [17] an improvement to the approximation of the overflow probability estimation is introduced, based on a work by M. Iltis [12] that allows to prove the following result:

Corollary 2.2: Under the previous hypothesis , and using that the traffic aggregates X_m^N are an aggregate of i.i.d sources it can be proved that:

$$\mathbf{P}(\text{link overflow } k) \underset{N\to\infty}{\sim} C_N e^{-N\mathbf{I}_k} \quad \text{(II.10)}$$

where C_N can be written as:

$$C_N = \frac{d_0 C}{\sqrt{N}} = \frac{d_0}{\sqrt{2\pi N \det(V(x^*))}|\alpha|(\det(|\alpha|(H_d(x^*))))^{1/2}}$$

where x^* is the point where the minimum of equation II.8 is reached, $\alpha = \nabla I|_{x^*}$, $V(x^*) = (\text{Hess}(I)|_{x^*})^{-1}$, and $H_d = H_h - H_f$ are the substraction of the hessians of h and f, that are local expressions of the border of the feasible region and the level curves of the function I. d_0 is an adjustment factor that is constrained by the order of the buffer.

By means of this correction it is possible to improve the results on equation II.8. Setting $d_0 = 1$, we get an estimation of the form: $\frac{C}{\sqrt{N}} e^{-N\mathbf{I}_k}$.

Observation 2.1: Setting $d_0 = 1$ corresponds to asume we are estimating the parameters on a bufferless network. The result by Likhanov shows that the approximation is good for networks where the buffers are kept "small" compared to the link capacity. In networks with larger buffer sizes the overflow probability will be smaller meaning that d_0 should be less than 1. Selecting another value for this parameter is not posible at this time as it depends in a complex way of buffer sizes across the network, and there are no results on this subject.

To conclude, the necessary steps to find the overflow probability in a network link are the following:

- Find the large deviation rate function for each traffic aggregate that feeds the network $I_j(x)$. This function must be estimated from the traffic traces.
- For each link k, the transfer function $g_k(x)$ must be calculated.
- Solve the optimization problem II.9 to find the link's overflow probability.
- Calculate the correction constant C, which depends of the functions I_j and g on a neighborhood of the solution point.
- Estimate the link overflow probability as:

$$\mathbf{P}(\text{overflow}) \approx \frac{1}{\sqrt{N}} C e^{-N\mathbf{I}} \qquad \text{(II.11)}$$

  where N is the number of sources the feeds the link's buffer.

On the following sections we will discus the different steps mentioned on the previous paragraph.

III. Problems to solve

A. Find the relevant LSPs

As mentioned on the last section, the first problem we have to deal with is to determine which LSPs contribute in any way to the traffic arriving to the studied node. We will take into consideration not only those LSPs that actually pass through the link, but also any other one that shares resources anywhere on the network with them. It is really important to accurately establish the number of LSPs that affect each link, because this number will be the dimension of the optimization problem to be solved. The dimension of this problem will influence directly on calculation times, and so will affect the tool's performance.

- Determine which LSPs go through the link in the specified direction. Eventhough all the links that defined on the network topology are full duplex, we will treat them as two separate links, with traffic flowing in opposite directions, just for calculation matters.
- Add the LSP found on the last step to the list of important LSPs.
- For each of this LSPs, find the previous link:
  - If the current link is the first of the LSP, (it has no previous link) the algorithm finishes for this LSP and it goes on with the others.
  - If there is a previous link, the relevant LSPs for it are found using this same algorithm, and added to the list of important ones.

When working with feed-forward networks, the fact that we go through in a recursive way all the LSPs, that at some point share resources with the one we are studying, guarantees us that the algorithm will converge, finding the relevant LSPs.

B. Computation of the transfer function

One of the objectives previously defined, was to be able to carry out performance estimations over any kind of network. This arises two restrictions when dealing with the implementation of the transfer function: it has to be robust, and topology independent. More specifically, the procedure described on this section, calculates the traffic rate that arrives to a given link (the one that invokes it). This transfer function (g), was presented on section II and represents the constrain of the optimization problem.

To calculate the total traffic rate that flows to the studied link, we need to compute the contribution of each of the LSPs that traverses it. The following (per LSP) recursive algorithm was developed to accomplish this:

- Determine the previous link of the considered LSP.
  - If the studied link is an edge link, that is, it has no previous link, the LSP's contribution is the traffic rate at the edge.
  - In case there is a previous LSP, its contribution is computed using the shaping function, II.6.
- The shaping function takes as a parameter each previous link's contribution. This is where the recursion appears, as the same function is invoked again, over each previous link on the path.

As this function is based on the algorithm used to determine the relevant LSPs, we can assure it will converge to a result, when dealing with feed-forward networks.

C. Optimization algorithm

In order yo solve the minimization problem presented on section II, the constrained problem (P) was transformed into a non-constrained one using a penalty method. Furthermore, to solve the latter a steepest descent method was used, with inexact linear searches, which used step lengths given by the Wolfe's rule.

On figure 1 a flow diagram of the algorithm used is presented.

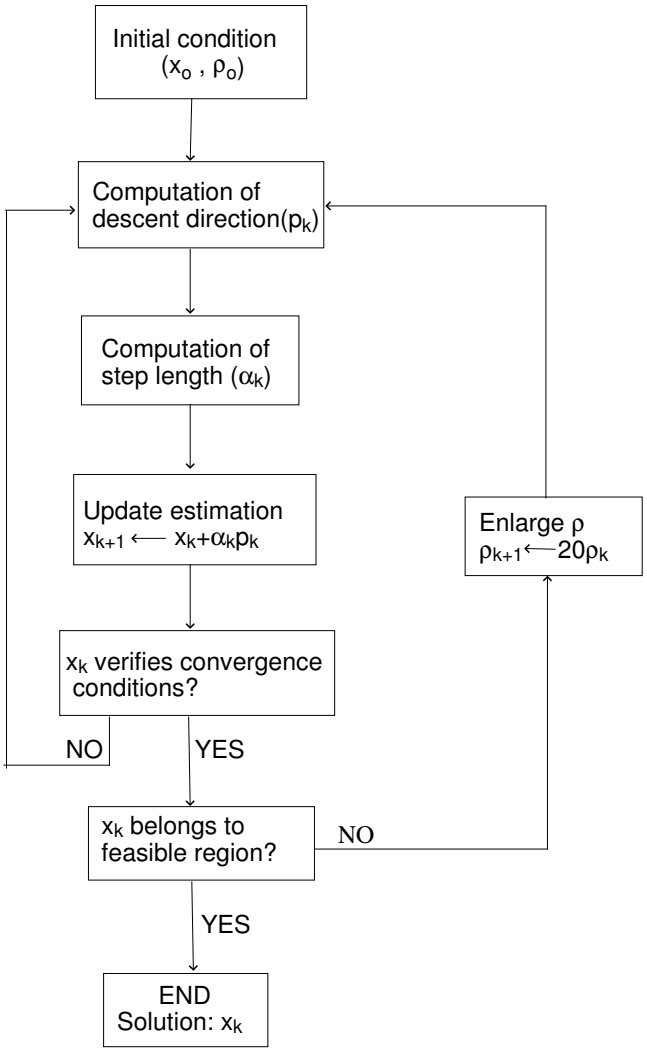

Fig. 1. Flow diagram of the iterative algorithm

On each step of the iteration, an auxiliar problem P_ρ is solved

$$(P_\rho)\begin{cases} \min \phi_\rho = f(x) + \rho\left(g_m(x)\right)^2 \\ x \in \mathbb{R}^n \end{cases}$$

for a given value of ρ, where $g_m = \max\{g(x), 0\}$.

It can be easily proven that the minimum of P_ρ is always less or equal than minimum of P, our original problem. Furthermore, the equality will only be reached on a feasible point, that is, when the constrain is met. So, after each auxiliar problem is solved, the solution, $x(\rho)$ is tested to be a feasible point. In case it is feasible, the whole iteration ends and the optimum $x^* = x(\rho)$ is found[1]. On the other hand, when $g_m\left(x(\rho)\right) > 0$ the value of ρ is increased (on our particular case, $\rho_{k+1} = 20\rho_k$) and $P_{\rho_{k+1}}$ is solved. Making the value of ρ larger after each auxiliary problem is solved, determines that the values $x(\rho)$ are increasing on every iteration, and each time closest to x^*. The iteration stops when $x(\rho)$ is a feasible point, that is, a minimum of the original problem P.

D. Estimation of the rate function

One of the main issues to take into account in order to estimate the overflow probability is to know the rate function of each flow, using traffic observations. This brings up the need to develop estimators for the rate function of a stationary process using different samples of it. As may be noticed on the definition (equation II.2), the rate function is closely related with the effective bandwidth function [13], thus estimating the latter may lead us to the desired estimators.

We will adopt the following notation:

$$M(s) = \mathbf{E}\left(e^{sX}\right) \tag{III.1}$$

$$\Lambda(s) = \log \mathbf{E}\left(e^{sX}\right) \tag{III.2}$$

$$I(x) = \sup_s \{sx - \Lambda(s)\} \tag{III.3}$$

On the previous equations, X represents a typical work arrival, and as the process is stationary, the t parameter does not appear at all.

Our goal is to use a traffic trace (one single run of the process X_t) to develop an estimator for function I (that is, the value $I(x)$ for each x).

More specifically, we have samples $\{x_t : 0 \leq t \leq n\}$ and we will obtain, for each x, an estimation $I_n(x)$, that varies depending on this samples.

The main problem concerning the estimation, is that I represents a complex transformation of $\Lambda(s)$, and its estimation is not direct using the samples. Because of this, we propose to follow an indirect path, estimating Λ as $\Lambda_n(s)$ and apply the transformation in equation II.3 to it:

$$I_n(x) = \sup_s \{sx - \Lambda_n(s)\} \tag{III.4}$$

Moreover, the value of the overflow rate, (**I**) for a given buffer, which comes from the equation II.8, will be

[1]And x^* is the minimum of (P)

estimated by a certain $\mathbf{I}_n$, which is the result of solving the same equation, substituting $I(x)$ by its estimations:

$$\mathbf{I}_n = \inf\left\{\sum_{m=1}^{M} I_n^{(m)}(x_m) : g(x_1,\ldots,x_M) > C\right\} \quad \text{(III.5)}$$

where $I_n^{(m)}$ represents the estimation of I_m from the data, g the transfer function, and C the link's capacity.

This gives place to several subproblems, which will be addresses one at a time on the section:

- Find a suitable estimator for the functions $M(s)$ and $\Lambda(s)$, that is, the effective bandwidth, using the input data. We will also estimate the order of the estimation error.
- Find under what circumstances the estimator, $I_n(x)$ of equation III.4 converges to the real value $I(x)$ as well as the error in this operation.
- Determine under what circumstances it is possible to estimate the value of $\mathbf{I}$, solving equation II.8.

Effective bandwidth estimation (or either $M(s)$ or $\Lambda(s)$), can be dealt with using two different approaches. On the first place, one can be tempted to search suitable estimators that can be used for several traffic models, without assuming any restrictive hypothesis. This approach is referred to as *non-parametric*

The non-parametric approach consists in using the usual hope estimator:

$$M_n(s) = \frac{1}{n}\sum_{i=1}^{n} e^{sx_i} \quad \text{(III.6)}$$

Secondly, when working with parametric models (e.g. Markov ON/OFF fluid sources [14]), it is possible to develop a parametric estimation of the magnitudes of interest. This is based on the fact that some explicit formulas exist to compute this magnitudes using the parameters of the Markov chain modulating the process. This leads to *plug-in* estimations where we estimate the parameters of the model in first place, and later substitute them on the expressions to estimate the effective bandwidth. Further reference about this estimation can be found on [19].

The next step is to find an estimator for the rate function $I(x)$. The way to do so was presented on equation III.4, and briefly consists in finding the maximum in II.2 substituting $\Lambda(s)$ with an estimation. The accuracy and validity of this estimator is not trivial, on [10] this issue is studied thoroughly.

If an estimator of the derivative of Λ is available, (Λ'_n), the problem can be dividen in two steps. Firstly s^* is estimated as:

$$s_n^* : x - \Lambda'_n(s_n^*) = 0 \quad \text{(III.7)}$$

and following, $I(x)$ is estimated as:

$$I_n = s_n^* x - \Lambda(s_n^*) \quad \text{(III.8)}$$

There are several alternatives to estimate the derivative. If we have an estimator for Λ, Λ' could be approximated with a incremental quotient. On the non-parametric case, the derivative can be estimated, taking into consideration that:

$$\Lambda'(s) = \left(\log\left(\mathbf{E}\left(e^{sX}\right)\right)\right)' = \frac{\mathbf{E}\left(Xe^{sX}\right)}{\mathbf{E}\left(e^{sX}\right)}$$

so, Λ' can be estimated by an average quotient:

$$\Lambda'_n(s) = \frac{\sum_{i=1}^{n} x_i e^{sx_i}}{\sum_{i=1}^{n} e^{sx_i}} \quad \text{(III.9)}$$

To solve equation III.7, the secant method for nonlinear equations was used, [7].

Finally, having $I_n(x)$ as the estimator for $I(x)$, we estimate $\mathbf{I}$, the overflow rate of a link using equation II.8. The demonstration that this procedure is valid, is similar to the one used to prove that I_n estimates I.

IV. Software architecture

On the present section we describe ARCA (Analizador de Redes de CAminos Virtuales meaning virtual path networks analyzer) the software developed to solve the theoretical problems presented on the previous section,

This software tool was developed following the guidelines to allow code reusability, and the possibility to be complemented by auxiliar software packages in order to add new functionalities without major modifications. In order to comply with the previous requisites we programmed using an object oriented language, JAVA. On this section we will describe thoroughly on the implementation of the two main aspects of the tool: network representation and performance parameters estimation.

A. Network Representation

According to the guidelines previously defined, we focused on identifying on a network those elements that represent a concept either because of their operating or physic characteristics. Later we would associate them a java class describing their properties and implementing their functionalities. With the interaction of this classes we came to a network architecture that will be described in brief, and provides the physical support to every process that occurs on the network.

Our network model has three levels of abstraction:

- One describing the network topology, implemented using the classes *Nodo* and *Enlace*.
- A second level, to define virtual paths over the topology, represented by the class *Lsp*.
- A last level, describing the properties of the flow traversing the network, done with the class *Flujo*, from which other classes will inherit, describing each one different traffic models.

All this classes group together on a package named *Topología*, which also contains a class *Minim* whose main role is to implement mathematical tools that carry out computations on the topology. The UML[2] diagram on figure 2 allows a better comprehension of the relations between the classes that belong to this package. Following we present a brief description of each class.

1) Class Nodo *(Node):* The node represents on our network architecture the connecting element. A non predefined number of links may originate or converge to a node, which will later form the paths that will go across the network. This characteristic defines nodes as an absolutely necessary element in order to determine the way the flows traverse the network.

2) Class Enlace *(Link):* Whenever data transmission occur on a network, the link appears as the key element. This topological element concentrates most of the parameters relative to this task, e.g. data processing capacity, buffer size, etc.

Each instance of *Enlace* represents a bidirectional connection between two instances of *Nodo*, that are given during its definition. The nodes are used to determine the way the LSP passes throw the link, which is useful to find out which flows share resources at a given link, and thus are relevant when estimating the overflow probability.

3) Class Lsp*:* This class is used to model the virtual paths which can be defined over MPLS networks, and allow to direct an traffic aggregated flow. This class is the first that does not represent any physical element from the network, and can be associated with the second level of abstraction in our model. Each one of this LSPs is formed by an ordered concatenation of links, in the way they are traversed. Moreover, as the LSPs define the way an aggregated flow will follow, they will be linked to an instance of the class *Flujo*. This class represents the characteristics of the traffic passing throw the LSP.

4) Class Flujo *(Flow):* As said, this class is used to represent the characteristics and requirement of the different kind of flows that enter the studied network. It stores the statistic properties of a representative flow from the aggregate passing throw a path.

[2]Unified Modeling Language

The objective of this class is to provide a framework defining the main functionalities any flow has. Later other classes will be build that inherit from this one, defining a unique interface presented to the rest od the package. For those traffic models that, due to their specific characteristics allow a simpler and/or more effective way to compute the methods associated to a flow, additional classes will be defined, which inherit from *Flujo*.

The most general way an instance of "Flow" can be defined is using a traffic trace, this is why this class provides methods to read from an archive and store, as well as perform statistic calculation with the data.

5) Class FlujoONOFF *(ON-OFF Flow):* A kind of traffic that is of particular interest on modern networks is Exponential ON-OFF model, also known as Markovian ON-OFF.

In order to be able to represent this model on our tool, a special class was developed, *FlujoONOFF*, that inherits from *Flujo* and redefines the methods that can be compute on a more efficient and exact way taking into consideration the Exponential ON-OFF flow hypothesis.

6) Class Minim*:* The main task of this class is to implement the minimization algorithm described on section III-C, as well as some complementary tools for performance estimation computations. The decision to create a specific class for this purpose, was taken based on the **expert** pattern from UML, [15]. This algorithm shall be controlled by a class conceptually above the rest of the network, and with access to all the necessary information.

B. Performance parameter estimation

The main goal of the software tool is estimate the overflow probability at any given link of the network. In sections II and III we described both theoretical and practical implementation to achieve this goal. On the previous sections we focused on the classes involved in this implementation. In the present section we will describe the interaction process between classes to implement the desired estimations.

The general use of the tool is splitted in two different phases. In the first one, the user defines the network topology basically represented by its physical elements, nodes and links (classes *Nodo* and *Enlace*). Later, over the physical elements the user defines the paths for the traffic using *LSP* and *Flujo* classes.

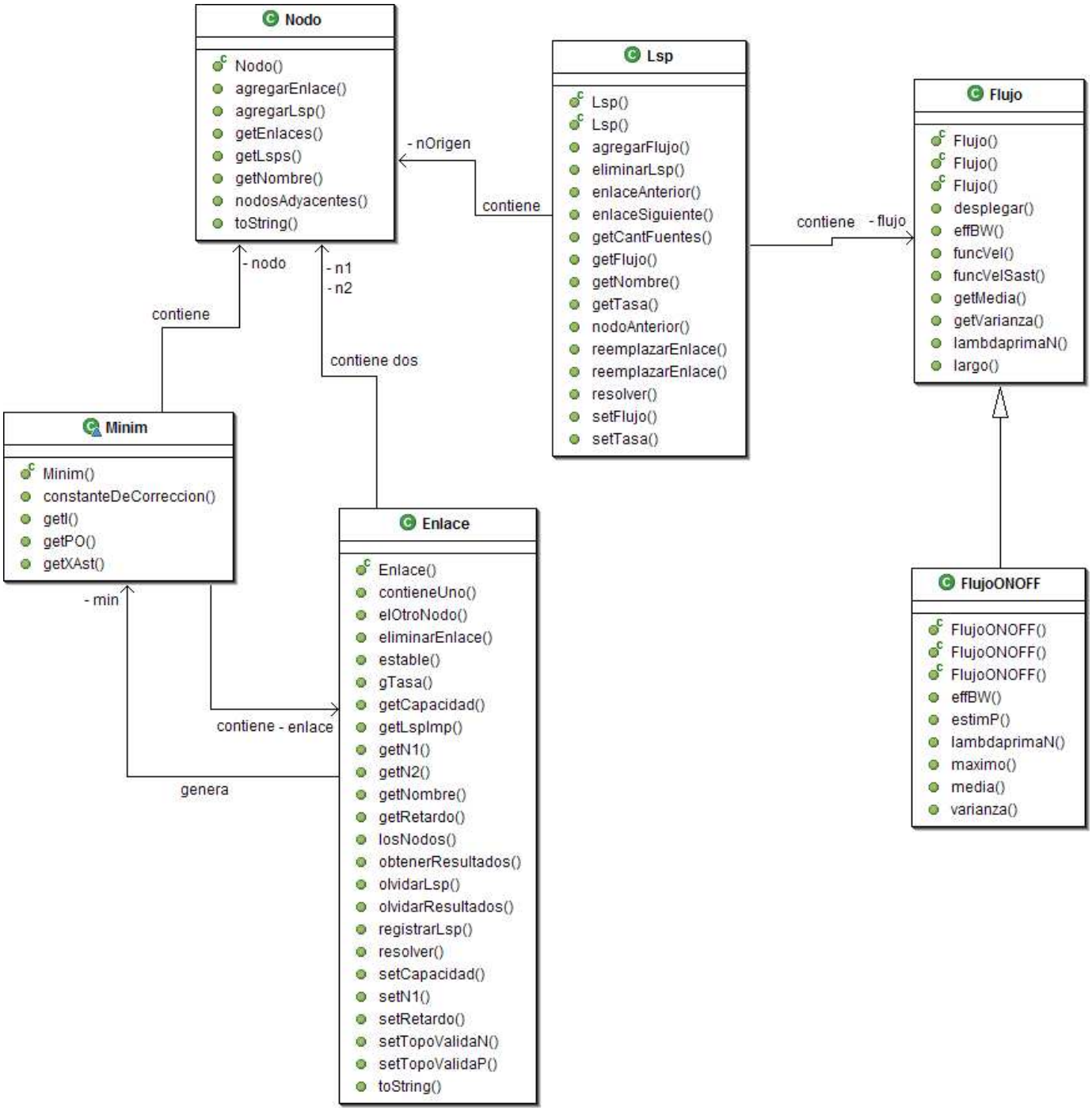

Fig. 2. UML diagram of package “Topologia”

In the second phase we procede to estimate the overflow probability over the previously defined topology. The user is able now to request the tool to solve a certain Link or LSP[3]. The resolution of a link implies to estimate it's overflow probability. The user invoques commands which allows him/her to solve one or more links by using the method **resolver** from the *Link* class.

We chose this kind of structure to handle the link calculations in order to implement a distributed architecture where each link is capable to solve itself and store the results. This structure also allows us to avoid implementing a global scope class to handle the estimations for each link. That implementation would have lead to an unnecessary exchange of data and overflow of such a class.

Every result of the overflow probability is stored in such a way that is accessible all the time for each link.

In the next subsection we describe the method **resolver** and its interaction with the rest of the architecture, once the user request a certain link to be solved.

It is important to remark that for the first phase the user has two options to define the network topology. The tool has a user friendly graphical interface to define the nodes, links, paths and flows. On the other hand, the same interaction could be achieved by providing a text file with a preestablished format. This last alternative allows the user to automatically (script) create multiple scenarios, solve them and store the results.

C. ***Resolver*** *method*

As it was said before this section attempts to describe the processes that take place once the user requests to solve a link. Since we are working with bi-directional links, the first thing to do is select the direction of interest. In order to do so, method **resolver** receives as a parameter an instance of *Nodo*, to decide which direction to take into account. At the same time, the method checks the coherence of the request by verifying the existence of the given node.

In most cases, the method shall run the minimization algorithm described in III-C, but in order to avoid unnecessary processing several checks are performed before that. This improves dramatically the software's performance.

The first task the method carries out is to find the relevant LSPs, as described before. If it finds none, the method ends indicating that no traffic is carried in the specified direction. In this case clearly the overflow probability is zero.

[3]Solving an LSP is merely to solve each one of the it it has

In case the method identifies a group of relevant LSP, the link must be checked to see whether the incoming traffic is able to overflow it. If the maximum arrival rate from the previous links on each relevant LSP, is less than the capacity of the link, it will never lose work under any circumstances so it is overdimensioned. In this particular case, the solution is trivial and the overflow probability is zero. The software also provides additional information about the remaining capacity of the link. The maximum arrival rate of a link, is the result of maximizing the input traffic rate of each LSP that arrive to the link, and turning zero every other LSP's rate[4].

The next step will depend on the amount of relevant LSPs for the link. In case there is only one relevant LSP, the minimization problem is trivial and the solution is $I_m(C)$ where m is the flow associated to the LSP and C is the link's capacity. In this 1 dimensional problem the correction coefficient is calculated using Bahadur-Rao's theorem [8].

Finally, in case our problem's dimension is greater than 1 (more than one relevant LSP), the algorithm creates an instance of the *Minim* class to solve the minimization problem. No references to the Minim instance are stored, this characteristic optimizes memory usage and avoids unnecessary redundancy. The correction coefficient for this case is calculated using Iltis theorem [12].

In the previous cases, the link calls the getI method of the Minim class to handle with the processing. Once the results are ready the resolver method recovers from the *Minim* instance the most relevant values like the overflow probability, the overflow rate (**I**), the coordinates of the minimum (x^*) and the correction coefficient.

With all the numerical results, the link creates an instance of the ResolverEnlace class to store all the values together with the particular observations that came up on the estimation.

V. Results Validation

Some representative network topologies were selected to test the accuracy of our tool, estimating the overflow probability. In these cases the probability of overflow given by our tool was compared with the obtained by means of simulations in NS-2[5] varying the number of sources that enter the network.

Our validation scheme brought up the need to estimate the overflow probability from simulations in different links of the network. With this purpose buffers were

[4]as they will reduce the rate arriving to the studied link

[5]Network Simulator 2: network simulator developed by the University of Berkeley

monitored and the overflow probability was estimated as follows: [6]:

$$P_{\text{exp}}(overflow) = \\ = \frac{\text{time units where ths system lose data}}{\text{total time of simulation in time units}}$$

A. General hypothesis for the simulations

In order to evaluate the buffers overflow state, we worked in discrete time with the same sampling rate that we used to sample the traffic traces for our application. The simulation time in all cases was large enough to assure the estimator's convergence. Additionally, each topology was simulated in 5 occasions, and the average of the obtained overflow probabilities was considered the real value to reduce the effects of the variance on each simulation.

In each simullation the size of the buffer grew along with the number of sources, using a constant b, that changed on each simulation ($B(N) = b\sqrt{N}$). Finally we chose to simulate in all the cases with markovian flows ON-OFF with different work cycles and arrival rates.

In order to measure the accuracy of the estimations the distance in logarithmic scale was chosen as magnitude of comparison:

$$E = \log_{10}\{\mathbf{P}_{estim}(overflow)\} - \log_{10}\{\mathbf{P}_{exp}(overflow)\} \quad \text{(V.1)}$$

This magnitude is suitable to measure the precision in the order of the estimation. It is often more worthy to consider the order of the overflow probability and not its exact value. In terms of the chosen measure, a unit represents an order of magnitude of difference.

B. Two links case

Figure 3 shows our first studied topology, that consists of two links and three LSPs. One LSP goes through both links and the rest share resources with previous one just in one link.

This simple example will provide a qualitative approach to the accuracy and validity of estimations provided by our software tool

On figure 4 the results obtained from the overflow probability estimation are shown and on figure 5 the resulting error from this estimation appears. Both graphs show the results varying for several number of sources. The reason to do so is because as our results are asymptotic, we expect to get better results as N grows.

[6]Based on the ergodicity hypothesis on the arrival process

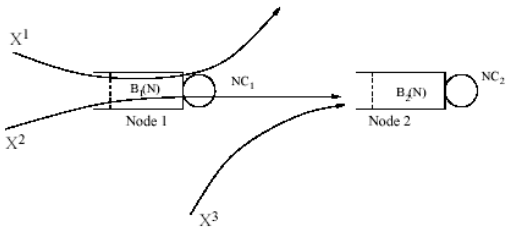

Fig. 3. Two links case topology

Accuracy reached in this case is reflected in the low error obtained that can be seen in the following graph.

The simulations were made under the following conditions.

- **Flow type** ON-OFF Markovian Flow, with mean ON time = mean OFF time = 10 time units and a maximum arrival rate of 5 packets per unit time.
- **Links' capacity:** $c_1 = 7$, $c_2 = 5.5$ This magnitudes are capacities per source and are measured in packets per time unit.
- **Buffer's constant** $b_1 = b_2 = 2$
- **Number of sources:** $N = 20 \ldots 450$

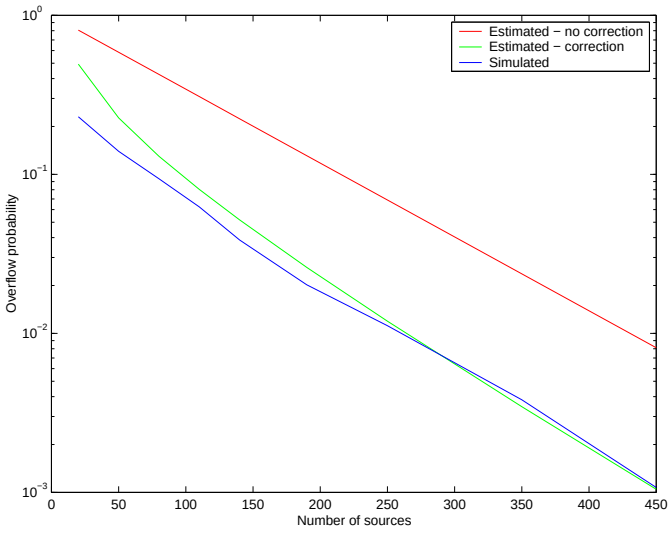

Fig. 4. Results for the two link case

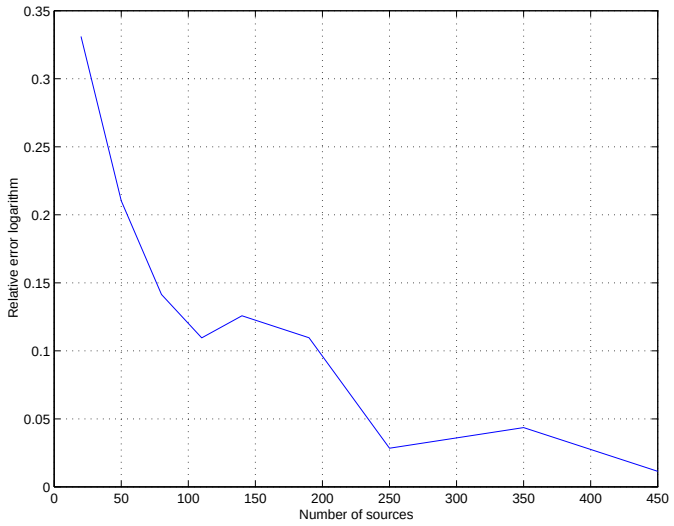

Fig. 5. Error results for the two link case

C. Tree topology network

This case attempts to study the behavior of those links in which a high concentration of traffic occurs. The correct design of this kind of links is crucial to guarantee end to end QoS of the network. In this kind of situations a tool like the one developed turn out to be really useful.

The aforementioned topology generated with our tool can be seen on figure 6.

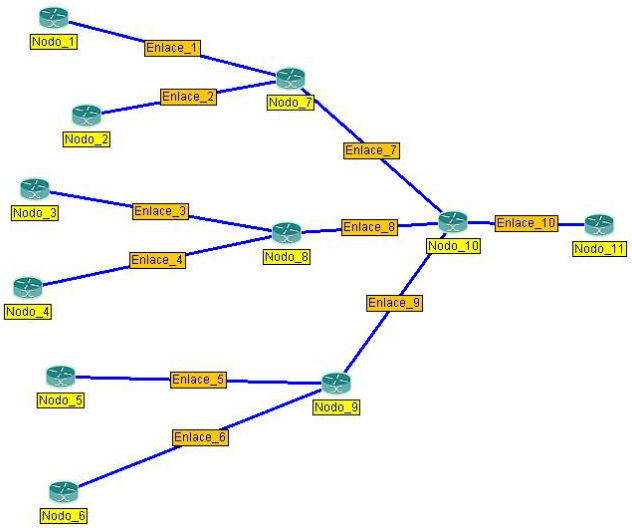

Fig. 6. Tree topology

Over this topology 6 LSP were defined from the different “leaves” of the tree to the “root” node. The simulations were made under the following conditions.

- **Flow type** ON-OFF Markovian Flow, with mean ON time = mean OFF time = 10 time units and a maximum arrival rate of 5 packets per unit time.
- **Links capacity:** The higher capacity per source is at the ”leaves”nodes and decrease to the ”root” node.
- **Buffer’s constant:** $b_i = 2$ with $i = 1 \dots n$
- **Number of sources:** $N = 10 \dots 220$

The results are shown on picture 7

As it can be seen, the graphs show the effect caused of the approximation of d_0 by 1 over the estimations. As in the previous case our tool properly captures the asymptotic slope and the error never exceeds the order of magnitude.

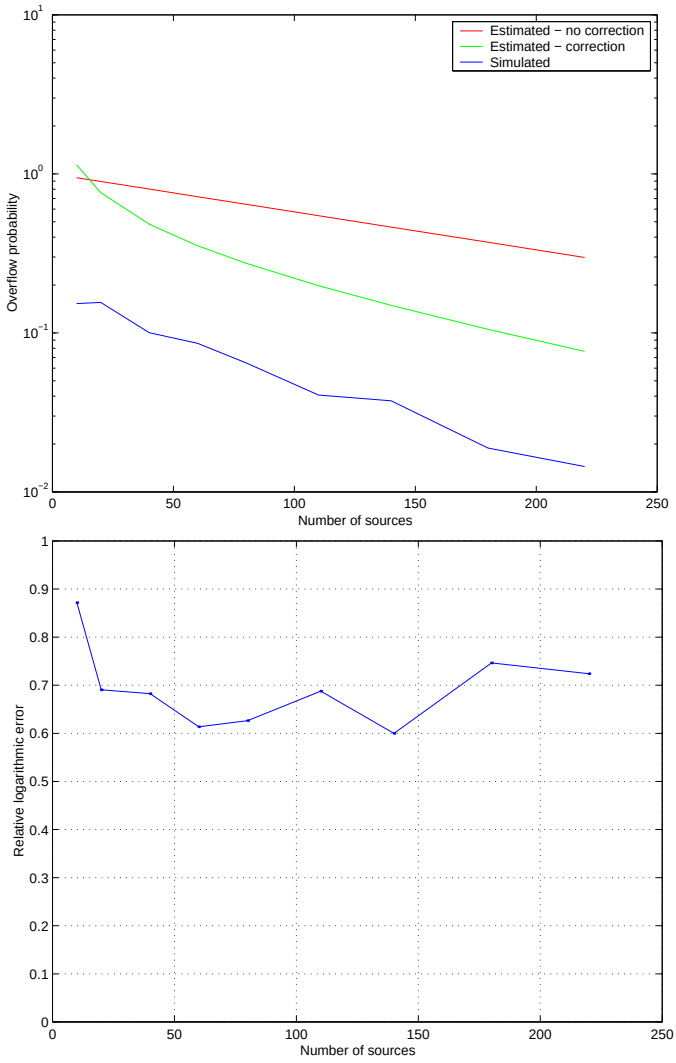

Fig. 7. Results for the tree topology case

D. Results’ analysis

The most significant result we can conclude from the previous situations, is that our estimations are close to the real[7]. This was confirmed with the different graphs presented. In spite there is a noticeable error,

[7]By real we are meaning the obtained on the simulations

the method captures very accurately the behavior of the slope (overflow rate).

Unfortunately, for some particular cases the results are not that good because the model seems not to match completely to reality. This problem is noticed especially when the behavior of the network is strongly influenced by the buffer's state. Nowadays there are no results on the field that provide better estimations. This assumption was checked in simple examples where we could easily estimate the theoretical probability in another way. This allowed us to verify that the estimation reaching its accuracy threshold, and the committed error was caused by the model used.

Considering the generality of the technique and the short processing times it important to remark that the obtained estimations constitute an excellent approximation of the overflow probability. It is relevant to point out that for all the cases we worked with, our results always were on the safety side, since in none of them losses are underestimated. Additionally, the error obtained in each case, is never greater than the order of magnitude, that in many cases is the accuracy degree needed to design.

VI. Conclusions

The software implements a theoretical set of tools in a practical methodology that allows to carry out the calculation of theoretical estimators. The developed application allows to evaluate the operation of a given network from samples of its traffic by providing the overflow probability in each link of such network.

We developed a library of classes that allows to define a network of arbitrary topology and provide numerical results about the defined network's performance. This library was designed in such a way that could be reusable by other applications using different performance estimators.

The processing time in the final version of the tool widely surpassed the expectations. The final performance of the algorithms allows to think about applying these techniques for on-line traffic engineering. This processing efficiency is based on the development of a specific algorithm for the complex problem of optimization presented. With a similar objective, we defined a model to approximate the statistical characteristics of the traffic that significantly reduced processing times.

Another contribution constitutes the developed base language to use the libraries, and the set of classes that implements the user's interface with the tool. We also developed a graphical interface to simplify the usage of the tool, allowing to visualize the topologies in study and to allow modifications easily. This allows to analyze the impact in the performance produced by the variation of the different parameters of the network, which is very useful in the design stage.

References

[1] L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas, "LDP Specification", Internet Engeneeirng Task Force RFC 3036, Enero 2001.

[2] L. Aspirot; "Procesos condicionalmente débilmente dependientes y su aplicación a la estimación de anchos de banda efectivos", Monografía de Lic. en Matemática, Fac. de Ciencias, 2003.

[3] P. Bermolen; "Ancho de banda efectivo para flujos markovianos", Monografía de Lic. en Matemática, Fac. de Ciencias, 2003.

[4] C.S. Chang; *Performance Guarantees in Communication Networks*, Springer, 2000.

[5] C. Courcoubetis, R. Weber; "Buffer overflow asymptotics for a buffer handling many traffic sources", *Journal of Applied Probability* 33, pp. 886-903, 1996.

[6] C. Courcoubetis, V. Siris; "Procedures and tools for analysis of network traffic measurements", Performance Evaluation, 48, pp. 5-23, 2002.

[7] G. Dahlquist, B. Björck, I. Andersson; "Numerical Methods", Prentice Hall, 1974

[8] A. Dembo, O. Zeitouni; *Large Deviations Techniques and its Applications*, Bartlett and Jones, 1991.

[9] B. Eckel; *Thniking in Java* 2nd. Ed., Prentice-Hall, 2001.

[10] Grupo ARTES; "Quality of service parameters and link operating point estimation based on effective bandwidths", *Proceedings of the 3rd Confernce on Hetergeneous Networks (HetNet03)*, 2003.

[11] IEC Trillium, "Multiprotocol Label Swiching".

[12] M. Iltis; "Sharp asymptotics of large deviationss in $\mathbb{R}^d$. *Journal of Theoretical Probability* **8**(3), pp. 501-524, 1995.

[13] F. Kelly; "Notes on Efective Bandwiths" in *Stochastic Networks: Theory and Applications*, pp. 141-168. Oxford, Oxford University Press, 1996.

[14] G. Kesidis, J. Walrand, C.S. Chang; "Effective bandwidths for multiclass Markov fluids and other ATM sources", IEEE/ACM Trans. Networking, No.1, pp. 424-428, 1993.

[15] C. Larman; "UML y Patrones: "Una introducción al anàlisis y diseño orientado a objetos y al proceso unificado", Editorial: Prentice Hall, 2001.

[16] N. Likhanov, R. Mazumdar; "Cell loss asymptotics in bufferws fed with a large number of independent stationary sources", *Journal of Applied Probability*, **36**(1), pp. 86-96, 1999.

[17] N. Likhanov, R. Mazumdar,R., O. Özturk; "Many sources Asymptotics for Networks with Small Buffers", Queueing Systems (QUESTA), **46** (1-2), pp. 129-147, 2004.

[18] J.M. Ortega, W.C. Rheinboldt; *Iterative solution of nonlinear equations in several variables*, Academic Press New York, 1970.

[19] J. Pechiar, G. Perera, M. Simon; "Effective Bandwidth estimation and testing for Markov sources", Performance Evaluation 48, pp. 257-175, 2002.

[20] Rabinovitch, P. (2000) "Statistical estimation of effective bandwidth", M.Sc.thesis, University of Cambridge.

[21] E. Rosen, A. Viswanathan,R. Callon, "Multiprotocol Label Switching Architecture", Internet Engeneering Task Force RFC 3031, Enero 2001.

[22] Wischik, D. The Output of a switch, or, effective bandwidths for networks. *Queuing Systems* 32, pp. 383-396, 1999.