An efficient decentralized on-line traffic engineering algorithm for MPLS networks*

François Blanchy[a], Laurent Mélon[a†] and Guy Leduc[a]

[a]Research Unit in Networking, EECS department, University of Liège, Belgium
{blanchy,melon,leduc}@run.montefiore.ulg.ac.be

Multi-Protocol Label Switching (MPLS) provides ways to control the Label Switched Paths (LSPs) followed by traffic trunks in a network and thereby to better traffic engineer it. In this context, we look at the problem of organizing the mapping of LSPs in an optimal way throughout the network on the basis of a given objective function. This problem is highly combinatorial and makes dynamic and real-time features a difficult issue for any LSP routing scheme. For this reason, we propose a computationally efficient, though approximate, on-line scheme adapted to an incremental optimization of the network state. It is then applied to a seldom mentioned traffic engineering problem : the compromise between load-balancing and traffic minimization. It is expected that clever routing strategies to balance the network load will sometimes favor longer paths in order to avoid congestion, leading to an increase of the overall network utilization. This reasoning is confirmed by our study, and we show that an improvement in network management can be made by appropriately tuning this compromise.

1. Introduction

Routing in an IP network is conceptually rigid and straightforward from a traffic engineering point of view : any packet roughly follows the shortest path at its disposal. For this reason, attempts have been made to allow Internet Service Providers (ISPs) to gain a measure of control and engineer their traffic somewhat. For instance, it is now made possible in OSPF by tuning the weights assigned to the links (as illustrated in [7]). With MPLS though, the notion of packet can be abstracted to that of traffic trunk carried in LSPs, giving us the possibility to achieve a much more effective traffic engineering when setting up a route for such a traffic trunk. Indeed, it is a common belief that one of the most significant interest of MPLS is traffic engineering of a network, allowing a better use of its resources and making provisioning a bit easier.

The existence of explicit routing mechanisms in MPLS makes possible an overall optimization of the routes taken by all flows throughout the network. However, this requires that a knowledge is available on the amount of resources used by each LSP. This knowledge can come from active monitoring of the network or from a preceding request asking

*This work has been carried out in the IST-1999-20675 ATRIUM project funded by the European commission.

†Research Fellow of the Belgian National Fund for Scientific Research (F.N.R.S.)

for a certain level of leased (and guaranteed) resources. In this paper, we will assume this kind of data is made available one way or another and concentrate on the path optimization process. Quite some work has already been done on this topic. There are two main categories of schemes : *off-line* and *on-line*. The difference is based on the type of data used in the best paths computation : the first method requires a complete knowledge of all LSPs going through the network, whereas the second needs only a synthetic image of each link state in the topology. Any scheme can be implemented following two approaches : *centralized* (the computation is done by a server) and *decentralized* (the computation is done at an ingress router).

An *off-line* procedure is generally meant for a centralized implementation. A central route server periodically recomputes the best organization of the LSP map, thanks to a generally costly optimization procedure that may take a lot of time. This way, the response time of the network may fall down a long way in comparison with classical IP routing (unless new LSPs are established using a classical shortest path, waiting for reoptimization). Some papers however propose quicker specific schemes thanks to approximations or given traffic properties ([12] among others). Two persistent problems of this approach are that the network depends on very few points of failure (two if there is only one backup server), and the potential rerouting of lots of LSPs once a new optimal mapping has been computed. Indeed, the optimal solution has the property of being very sensitive.

The *on-line* approach, compatible with both a centralized and a decentralized implementation (provided the computational cost is not too high for a router), is favored by a number of papers ([3],[10], [9], [1], the last being clearly centralized). Generally, approximate algorithms are used to compute the path for any new requested LSP. Once the computation is done, the LSP is established immediately, using RSVP-TE [2]. This kind of scheme offers the advantage of being simpler, more robust, and with a generally shorter response time.

Our work is clearly part of the *on-line* decentralized approach. We propose a very effective approximation ($O(NM)$ with N the number of nodes, M the number of links) able to resist a very quick change of the network where plenty of LSPs are set up and destroyed in a short span of time. This is a first contribution of this paper. Indeed, we feel that most other schemes are not quick enough, would not scale very well and might put a heavy burden on a router in terms of computational power (except perhaps [9] but it focuses mainly on restoration). Our approach also offers the advantage of being very simple and generic, leaving aside, in a first time, the question of the objectives and the constraints of the optimization procedure. While most mentioned methods use specific and elaborate techniques, we think our scheme is potentially adequate (and generally computationally more efficient) to solve a large field of traffic engineering problems. This approximation procedure will be presented in section 2. In section 3, we will illustrate how this scheme can be used to tackle a basic traffic engineering problem : the compromise between balancing the available resources throughout the network and minimizing the overall use of the resources. Indeed, it is obvious that load balancing will favor paths a bit longer than a minimal hop routing.

2. Optimization point of view : the CSPF approximation

The goal of this section is to introduce the problem of optimizing the LSP mapping throughout a given topology. We wish to confront an optimal *off-line* scheme, using a branch-and-bound algorithm, with a simple *on-line* approximation obtained thanks to an adapted Constrained Shortest Path First (CSPF) procedure. Of course, we already know that the latter is by far the most effective (quickest) method, but we can wonder how good an approximation we can reach with such a scheme.

2.1. General presentation of the problem addressed

We are given

- A set of LSPs $\mathcal{L}$, ranged over by l, each characterized by a set of generic parameters $\lambda(l)$, including the source and destination nodes, bandwidth reservation, whatever else is needed for specific traffic engineering constraints.
- A topology, under the form of a finite connected graph $\mathcal{G} = (\mathcal{X}, \mathcal{U})$ where $\mathcal{X}$ is a set of nodes (vertices) and $\mathcal{U}$ a set of links (directed arcs) between these nodes. Each link is characterized by a set of generic parameters $\omega(i,j)$ with $(i,j) \in \mathcal{U}$ including for instance its capacity as well as any feature needed for traffic engineering. It is assumed no link of the kind (i,i) with $i \in \mathcal{X}$ is included in $\mathcal{U}$.
- A set of constraints to be respected by the LSP mapping on each link independently. That is, a conjunction of propositions,

$$\bigwedge_{(i,j)\in\mathcal{U}} \mathcal{P}(state^{\mathcal{L}}_{(i,j)}, \omega(i,j)) \quad with \quad state^{\mathcal{L}}_{(i,j)} = \{\lambda(l) | l \in \mathcal{L} \wedge (i,j) \in path(l)\}$$

  that must be verified. It can specify, among others, that the capacity threshold must not be exceeded.
- Two particular constraints allowed for each LSP $l \in \mathcal{L}$. The first is a *flow* constraint such that $path(l)$ must be a *path* in $\mathcal{G}$ from source node to destination node as specified by the graph theory vocabulary (a path in a directed graph is any sequence of arcs where the final vertex of one is the initial vertex of the next one). The second is a *no-loop* constraint such that $path(l)$ must be an *elementary* path in $\mathcal{G}$ (each node is used at most once). No other constraints are allowed on a LSP, which means we cannot ensure path-wide properties for the solution (for example, we cannot impose a maximal threshold on the number of hops for a LSP).
- An objective function $\mathcal{F}(path(\mathcal{L}), \lambda(\mathcal{L}), \omega(\mathcal{U}))$ to minimize.

We look for a mapping from $\mathcal{L}$ to $\mathcal{G}$ under the form of a function

$$path(.) : \mathcal{L} \longrightarrow (2^{\mathcal{U}}, order^{3})$$

specifying the path taken by a LSP from source to destination through $\mathcal{G}$, minimizing $\mathcal{F}$ and respecting the given constraints.

[3] the nodes in this set are ordered by precedence in the path

Any instance of this problem can be formulated as a non-linear (at worst, depending on the objective function $\mathcal{F}$) mixed integer problem with non-linear constraints (again, at worst, depending on the per-link constraints). However, even the simplest formulation (linear mixed integer problem with linear constraints) leads to an at least *NP-complete* problem [8] commonly solved with a computationally (very) expensive algorithm called *branch-and-bound*. That is why an off-line approach (*i.e.* a powerful server computes all routes) can be seen as the most direct way to tackle this problem, though it is very slow and highly static. In a practical use (day-to-day management of a network), an additional constraint comes from the fact that the requests for LSPs occur at different points in time, requiring a constant update of the mapping. An off-line approach has difficulties to cope with this particular feature. It is therefore appropriate to look for polynomial approximate schemes following an incremental approach, that is, adding the LSPs one by one, in the order of occurrence of the requests. This is the scope of the next section.

2.2. Double approximation using a CSPF

With an incremental scheme, the same as with any other approximation, we can only hope to reach a local optimum among the available LSP mappings, whereas the *branch-and-bound* finds the global optimum. But even if we look only at the problem of adding a single LSP to a given topology (with or without any other LSPs running), computing the optimum (global for this reduced problem) is again at least *NP-complete*. Looking for a polynomial scheme, we have to settle for a second approximation, yielding a local optimum in our search for the best route for a new LSP.

Basically, we limit the exploration of the routes available to those examined by a constrained shortest path algorithm, a CSPF, the costs on the links being the *variation* on the objective function caused by bringing the link into the path for the LSP. A classical *Dijkstra* algorithm could almost do it except that those costs might be negative, depending on the objective function $\mathcal{F}$: nothing ensures that adding an LSP always makes the objective increase. We will then use the well-known *Bellman-Kalaba* scheme ([4]) (sometimes called *Bellman-Ford*) and adapt its formulation to our needs.

On the basis of the hypothesis of section 2.1, we have a set $\mathcal{L}$ of LSPs already established through the $path(.)$ function. We want to add a new LSP l^{new} with parameters $\lambda(l^{new})$. Let $\mathcal{L}^{new} = \mathcal{L} \cup l^{new}$. We define

- a predicate that tells us if (i,j) with $i, j \in \mathcal{X}$ is a valid link for l^{new},

$$p(i,j) = (i,j) \in \mathcal{U} \quad \wedge \quad \mathcal{P}(state^{\mathcal{L}}_{(i,j)} \cup \{\lambda(l^{new})\}, \omega(i,j))$$

- a *no-loop* clause applied to a potential partial route $\{i_0, i_1, .., i_{n-1}\}$ for l^{new} and a node i_n, such that

$$nl(i_n, \{i_0, i_1, .., i_{n-1}\}) = \neg(i_n \in \{i_0, i_1, .., i_{n-1}\}) \quad i_k \in \mathcal{X}, k \in [0, n]$$

- a score function applied to a potential partial route $\{i_0, i_1, .., i_n\}$ for l^{new}, such that

$$s(\{i_0, i_1, .., i_n\}) = \begin{cases} \mathcal{F}(path(\mathcal{L}^{new}), \lambda(\mathcal{L}^{new}), \omega(\mathcal{U})) \\ \qquad if \quad (i_k, i_{k+1}) \in \mathcal{U} \quad \forall k \in [0, n-1] \\ \infty \quad otherwise \end{cases}$$

$$with \quad path(l^{new}) = \{i_0, i_1, .., i_n\} \quad i_k \in \mathcal{X}, k \in [0, n]$$

- a link-cost function

$$v(i,j) = \begin{cases} \infty \quad if \quad \neg p(i,j) \vee \neg nl(j, \phi(i)) \\ \Delta(i,j) = s(\phi(i) \cup \{j\}) - s(\phi(i)) \\ \quad otherwise \end{cases}$$

$$\forall i, j \in X$$

with $\phi(.) : \mathcal{X} \longrightarrow (2^{\mathcal{U}}, order^4)$, yielding the current partial path from the source to the specified node already computed in the *Bellman-Kalaba* procedure. Of course, it is preferable that the evaluation of $\Delta(i,j)$ can be done without recomputing the whole route scores. A suitable property of the score function (*i.e.* the objective function $\mathcal{F}$) is that it can be updated effectively, in an incremental way, when adding a link to the path.

Using these notations as well as those of section 2.1, the formulation of the *Bellman-Kalaba* algorithm is almost unchanged from the original. Let $d_k(i)$ be the cost of the best computed route from source node 1 to node i at step k and note that in graph theory, the $\Gamma(.)$ function returns the "destination set" of a node, following link orientation.

The algorithm follows these steps :

$$1) \qquad \begin{cases} d_k(1) = 0, \forall k \in \mathcal{N} \\ d_1(i) = v(1,i), \forall i \in \mathcal{X} \end{cases}$$

$$k+1) \quad d_{k+1}(i) = \min_{j \in \Gamma^{-1}(i)} \{d_k(j) + v(j,i)\} \quad \forall i \in \mathcal{X}$$

Note that $v(j,i)$ above dynamically changes with the path used to reach j. It stops if $d_k(i) = d_{k+1}(i)\ \forall i \in \mathcal{X}$, $d_k(i)$ is then the minimal cost for a path from 1 to i and the approximation to the optimal route for the new LSP is given by $\phi(destination)$. The convergence is ensured in at worst $|\mathcal{X}|$ steps, unless there is some negative circuit (which is made impossible by the particular *no-loop* constraint presented in section 2.1). Note that this algorithm is of complexity $O(NM)$ with N the number of nodes, M the number of links, which makes it quite efficient. The *flow* constraint is of course ensured by the procedure.

Why is the solution approximate? The procedure progresses from the source towards the destination always using the link which minimizes the variation of the objective function. However, these variations depend on the partial path used to reach the link. Which means that, to find the true optimal route, we may have to choose a link whose variation is not minimal, but such that further variations will be smaller, and the final objective value lower. This can be compared to the well known problem of continuous optimization of a function whose slope is unknown. In that situation, we can reach a local minimum by following down the slope of the curve, but nothing tells us that there is no better minimum in another "valley", unreachable except by climbing a bit. But how do you know when to climb? This is what makes our problem *NP-complete*. There is no way to circumvent

[4] again, the nodes in this set are ordered by precedence in the path

this except by requiring certain properties from the objective function. For instance, in the continuous optimization problem just mentioned, a convex objective function makes sure any minimum we find is global. In general, a careful choice of the objective function can be sufficient to make sure the solution is not that much suboptimal as it will be the case with the load-balancing objective of section 3.

In order to assess the consistency of this approximation, we have confronted it to an exact algorithm. In [6] we presented a mixed-integer formulation for a basic instance of the problem introduced in section 2.1, and used it to draw a lower bound (a relaxed solution) as a point of comparison. The conclusion was that the approximation is very close to the optimal solution provided the mean LSP size is small enough compared to the mean link capacity.

3. Traffic engineering point of view

The purpose of this section is not the presentation of elaborated techniques to engineer the traffic of a network. Rather, we simply want to validate the approximate approach of section 2.2 by illustrating how it can be used to control the flows passing through a network, to minimize the resources used, the blocking probability, etc. This will be done simply by an appropriate choice of the objective function. Remember that, aside from the source and destination, a LSP $l \in \mathcal{L}$ is only characterized by its requested bandwidth $\lambda^{bw}(l)$. Each link of $(i,j) \in \mathcal{U}$ has a specific bandwidth capacity $\omega^{cap}(i,j)$. The only obligation we have is to respect the capacity on a link while routing the LSPs.

We will first present the objective functions in use, before laying down some results and observations obtained by simulations.

3.1. Objective functions for traffic engineering

In the following, we denote $L_{(i,j)}$ the load on link $(i,j) \in \mathcal{U}$.

3.1.1. Load balancing objective function

$$\sum_{(i,j)\in\mathcal{U}} \left(\frac{L_{(i,j)}}{\omega^{cap}(i,j)} - \overline{\frac{L}{\omega^{cap}}} \right)^2 \quad with \quad \overline{\frac{L}{\omega^{cap}}} = \frac{1}{|\mathcal{U}|} \sum_{(i,j)\in\mathcal{U}} \frac{L_{(i,j)}}{\omega^{cap}(i,j)}$$

the mean of the relative link load throughout the network. This function is then the variance on the relative link load and, as such, represents the deviation from the optimal load balancing situation. In a perfectly balanced network, this deviation would be zero for all links so that all would be occupied in exactly the same proportions. Note that an incremental update of the objective when adding a link is possible after some development (more details can be found in [6]). This is a quadratic function for two reasons:

- Obviously, the deviation should be a positive notion.
- As the deviation increases, it is penalized with a score that rises more than linearly. Indeed, it is conceivable that an optimization procedure might prefer a very poor balancing on a single link along with a perfect one for all others, against a situation

where all links are very well balanced (though not perfectly). Using a quadratic function prevents this to some extent.

3.1.2. Load balancing limitation (traffic minimization)

The main problem with the load-balancing function presented above is that the only thing it tries to do is to flatten the relative load throughout the network. It will not matter if some of the paths go a long way around in order to achieve a better load-balancing. This means that LSPs may be characterized by very long delays, and that the network load is susceptible to rise very quickly. We must then try to limit the length of the paths chosen for the LSPs by adding a kind of "shortest path length" term to the objective function. But of course, this limitation must be carefully chosen so as to let the load-balancing term operate when needed, both terms must scale in a similar manner.

After a careful study of the variation of the load balancing function, we decided to use as our "traffic minimization" term the sum of the square relative link loads. As a consequence, the compromise between load balancing and traffic minimization can be expressed as follows

$$\sum_{(i,j)\in\mathcal{U}} \left(\frac{L_{(i,j)}}{\omega^{cap}(i,j)} - \overline{\frac{L}{\omega^{cap}}} \right)^2 + \alpha \sum_{(i,j)\in\mathcal{U}} \left(\frac{L_{(i,j)}}{\omega^{cap}(i,j)} \right)^2$$

Why is this interesting? The (weighted) combination of both terms will give more importance to the load-balancing term if the deviation is high enough to justify the detour, else it will let the "shortest path" term minimize the resources used. The weighting factor α allows to give more importance to one aspect or the other.

3.1.3. Variant - Average delay objective function

$$\sum_{(i,j)\in\mathcal{U}} \frac{1}{\omega^{cap}(i,j) - L_{(i,j)}}$$

Since, for a packet size P, $\frac{P}{\omega^{cap}(i,j)-L_{(i,j)}}$ approximates the queuing and transmission delay on link (i,j), optimizing this function will strive to minimize the average delay throughout the network. This will naturally lead to load-balance the network. Moreover, this objective has a built-in traffic minimization feature in the sense that long paths would degrade the average delay and are therefore discouraged.

3.2. "Load balancing versus traffic minimization" compromise at work

Before presenting some simulations, let us consider what we could expect. Load balancing the network should ideally produce a network with an homogeneous blocking probability by source-destination pair. Hopefully, it could also lower the overall blocking probability in comparison with a classical shortest path (in number of hops). On the other hand, using such traffic engineering techniques sometimes favor detours and it might logically increase further the load inside the network (again compared with a minimum hop routing). Clearly there must be a compromise between load-balancing and traffic minimization. The purpose here is to experiment how the objective functions described in the preceding section, along with the CSPF approximation of section 2.2 allow us to

play on this compromise. More importantly, we want to know if some advantage can be drawn in the field of network management.

The results presented correspond to a generated topology with 20 nodes. 10 ingress nodes have been selected and a source-destination probability distribution as well as a LSP-bandwidth requested probability density have been randomly created. These probabilities are used and respected by a random LSP request (removal) generator. The ratio between the mean link capacity and the mean LSP size has been set to 50.In all simulations, LSPs are added up to a certain number or up to a certain network load (a kind of rise in power). Then LSPs are added and removed in such a way as to keep a stationnary state (in terms of the number of LSPs or the network load). The idea is to simulate a continuous use of the topology, lots of paths being set up and destroyed for some time while we track the blocking probability, the number of failed establishment, the network load, the number of LSPs established, and the relative load deviation (image of the load-balancing). Note that the blocking probability is an exact value, not an estimation, and was computed with a simple scheme described in [6].

The topology used has been "perfectly engineered" thanks to a *Generalized maximum concurrent flow* algorithm. By "perfectly engineered", we mean that a load of 100% is reachable throughout the network if the source-destination probabilities are respected and the LSP-bandwidth are infinitesimal. This has been done because we realized it was useless to try to engineer the traffic on a network with engineering inconsistencies such as huge links following very small ones (they can only reach a very small relative load). And indeed, the *Generalized maximum concurrent flow* approximation we used is close to the behaviour used by some network engineers we have met (*e.g. "double up the link capacity when it reaches 50% of load"*). Note that we have applied the same simulations to a completely random topology, the results were presented in [6] and are very similar, though in a less marked way.

The results are expressed in terms of the mean overall blocking probability, the mean number of LSPs and the mean network load observed throughout the simulations. Figure 1 illustrates our compromise in the case of a network load frozen at 80%, while figure 2 shows the results with a fixed number of LSPs (550). The shortest path in terms of hop counts is used as a reference.

For a constant network load, we can see on figure 1 that as the importance of load-balancing increases (lower α), the blocking probability decreases. However, the number of LSPs established goes down as well and falls very low for $\alpha = 0$. In this case, it would seem that $\alpha = 0.25$ is an optimal "load-balancing - traffic minimization" compromise, since it has a low blocking probability for a relatively high number of LSPs (3 times less chance of LSP blocking than the shortest path, for an equivalent number of LSPs). We can see that the compromise achieved by the delay objective is not quite as good, but it has the advantage of being self-contained, without any need for tuning, besides it is still much better than the shortest path. It is interesting to notice that the number of LSPs established by a constrained shortest path in the hop sense is slightly lower than what can be obtained with the superior values of α, though the network load is the same. By always choosing the shortest possible path no matter the consequences, we end up doing more detours to get to the destination as some links become saturated.

For a fixed number of LSPs, as shown on figure 2, the conclusions remain globally the

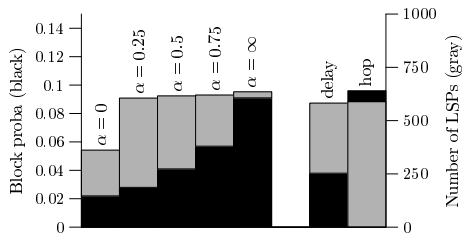

Figure 1. fixed network load (80%)

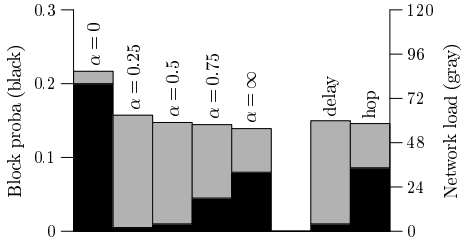

Figure 2. fixed number of LSPs (550)

same. The network load logically increases for a decreasing α favoring load balancing. The blocking probability, influenced by the network load, shows a minimum for $\alpha = 0.25$. The right compromise is achieved for $\alpha = 0.25$ or $\alpha = 0.5$. The untunable delay objective gives good results, but the shortest path is once again left behind with a very high blocking probability for a network load not that much lower.

By playing on this "load-balancing versus traffic minimization" compromise in real topologies, we can hope to gain much. Some providers we have met use to double the link capacity when the load reaches 50%, and indeed a shortest path routing strategy is then close to its limits concerning blocking probabilities. With a more sophisticated scheme such as this one, we can hope to make a better use of the resources by allowing higher network load in normal use.

4. Conclusion

We believe the kind of scheme presented in section 2.2 is well-adapted to solve traffic engineering problems of the category described in section 2.1 and some others as well. Though alternatives exist, whether *off-line* or *on-line*, *centralized* or *decentralized*, our scheme seems an excellent compromise between computational efficiency and optimality. It appeared in [6] that for a small enough LSP mean size compared to link capacities, the accuracy of the solution is high. This approach is also easily upgradable, in the sense that notions such as preemption [5], restoration [11]can be integrated in a straightforward manner. Moreover, the distinct underlying three-fold organization of the scheme (objective or score function, predicate, computation procedure) leaves the door wide open for refinement : we can define new objectives, add new clauses to the predicate and improve the approximate scheme independently. Last, due to the computational efficiency of the approximation, this method is highly dynamic, almost compatible with some kind of "peer-to-peer session" LSP establishment, and has a very short response time. In fact, the time needed to establish an LSP is practically reduced to that required to announce the path through the network.

Though section 3 was meant as a basic illustration of what we intend with this scheme, we think it has the interest of showing practically a compromise that is seldom mentioned : load-balancing against traffic minimization. It is obvious that clever traffic engineering techniques, favoring detours (even small) for the LSP paths may speed-up the growth of resources utilization and end up giving a worse result than a simple minimum hop routing.

We also offer a practical way to tune this compromise.

Finally, our scheme can be implemented quite easily. We only need an extension of an existing link state protocol (OSPF-TE [13] is an interesting possibility). We also need a way to establish the computed LSP path with the necessary bandwidth reservations throughout the network. This could be done using the extended RSVP-TE defined in [2].

References

[1] P. Aukia, M. Kodialam, P. Koppol, T. Lakshman, H. Sarin, and B. Suter. RATES: A server for MPLS traffic engineering. RATES: A server for MPLS traffic engineering, IEEE Network Magazine, pp. 34–41, March/April 2000.

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

[3] Gargi Banerjee and Deepinder Sidhu. Comparative analysis of path computation techniques for MPLS traffic engineering. In *Computer Networks 40*, pages 149–165, 2002.

[4] R. Bellman and R. Kalaba. Shortest paths through networks. Dynamic Programming and Modern Control Theory, pp. 50-54, Academic Press, 1965.

[5] F. Blanchy, L. Mélon, and G. Leduc. Routing in a MPLS network featuring preemption mechanisms. ICT, February 2003.

[6] F. Blanchy, L. Mélon, and G. Leduc. Assessment of protocols and algorithms for intra-domain traffic engineering. Deliverable 3.2 for the ATRIUM project funded by the European Commission, Available at http://run.montefiore.ulg.ac.be/~blanchy, January 2003.

[7] Bernard Fortz and Mikkel Thorup. Internet traffic engineering by optimizing OSPF weights. In *INFOCOM (2)*, pages 519–528, 2000.

[8] M.R. Garey and D.S. Johnson. Computers and intractability : a guide to the theory of NP-completeness. Freeman,New-York, 1979.

[9] Murali S. Kodialam and T. V. Lakshman. Dynamic routing of bandwidth guaranteed tunnels with restoration. In *INFOCOM (2)*, pages 902–911, 2000.

[10] Murali S. Kodialam and T. V. Lakshman. Minimum interference routing with applications to MPLS traffic engineering. In *INFOCOM (2)*, pages 884–893, 2000.

[11] L. Mélon, F. Blanchy, and G. Leduc. Decentralized local backup LSP calculation with efficient bandwidth sharing. ICT, February 2003.

[12] S. Salsano, F. Ricciato, M. Listanti, and A Belmonte. Offline configuration of a MPLS over WDM network under time-varying offered traffic. INFOCOM, June 2002.

[13] P. Srisuresh and P. Joseph. TE LSAs to extend OSPF for traffic engineering. Internet-Draft : draft-srisuresh-ospf-te-02.txt, Jan 2002.