Utility Maximization in Load-Balancing

Maximisation de l'Utilité Appliqué au Partage de Charge

Federico Larroca & Jean-Louis Rougier

Département Informatique et Réseaux

Institut TELECOM, TELECOM ParisTech

May 2008

Abstract. Current data network scenario makes Traffic Engineering (TE) a very challenging task. The ever growing access rates and new applications running on end-hosts result in more variable and unpredictable traffic patterns. This context makes the need of sophisticated TE techniques stronger than ever. Such mechanisms should make a good resource usage while providing users with the best possible performance. Up to now, TE was purely concentrated on the first point, typically striving to minimize a certain network cost defined in terms of the link's usage, assuming users would obtain a good performance as a consequence. In this work, we show that such procedure can result in unfairness among users and sub-optimal resource usage. We propose that the objective of TE should be to maximize a certain utility function measured at the user level instead. We define such a utility function and apply it to design a Load-Balancing scheme that is both fair and efficient. The solution to the resulting optimization problem can be obtained both by a centralized as well as a distributed algorithm, whose design we outline. We compare our proposal with prior TE mechanisms, finding that it obtains a more balanced load distribution with improved performance.

Resumé. L'évolution actuelle des applications et des réseaux pose de nouveaux défi pour l'ingénierie de trafic. L'augmentation des débits d'accès offerts aux utilisateurs et l'apparition de nouvelles applications (P2P, overlays) ont augmenté considérablement la variabilité et l'imprévisibilité du trafic. Or les techniques d'ingénierie de trafic utilisées aujourd'hui sont mal adaptés dans ces cas, car nécessitant une bonne connaissance du traffic. Des techniques d'ingénierie de trafic plus sophistiquée, notamment plus robustes face à ces variations, sont donc plus que jamais nécessaire. Ces techniques devront conduire à une bonne utilisation des ressources tout en offrant aux utilisateurs des performances adaptées à leurs services.
Le partage de charge est un outil d'ingénierie de trafic qui semble particulièrement adapté pour faire face à ces nouvelles contraintes. En revanche, jusqu'à aujourd'hui, les techniques de partage de charge existantes se sont concentrées sur l'optimisation des ressources, négligeant l'aspect performance utilisateur (en supposant implicitement que la performance perçue par les utilisateurs seraient bonne du fait de l'optimisation des ressources). Dans ce travail, nous montrons que cette façon de procéder peut provoquer des inégalités entre les utilisateurs et même une utilisation sous-optimale des ressources. Nous proposons une nouvelle technique d'ingénierie de trafic consistant à maximiser une certaine fonction d'utilité, définie au niveau des utilisateurs. Nous définissons une telle fonction et nous l'appliquons pour concevoir un mécanisme de partage de charge qui est en même temps juste et efficace. Nous présentons deux algorithmes, l'un centralisé et l'autre distribué, permettant de conduire à la solution au problème d'optimisation présenté. Nous comparons notre proposition avec des mécanismes antérieurs, et nous montrons que notre proposition conduit à une meilleure performance offerte aux utilisateurs, avec des niveaux d'utilisation des ressources plus équilibré.

1 Introduction

Network convergence is a reality. Many new services are offered in the same network, increasing the unpredictability of traffic patterns. To make matters worse, access rates had increased at such pace that the old assumption that core links capacities are several orders of magnitude bigger than access rates is no longer true. Thus, simply upgrading link capacities is now not enough, since it is no longer an economically viable solution. This means that network operators are, now more than ever, in need of Traffic Engineering (TE) mechanisms which are **efficient** (make good use of resources), but also **automated** (as much self-configured as possible), more **tolerant** with respect to network variations (changes in traffic matrix, or characteristics of transported flows) and more **robust** (in case of node/link failures).

One such TE technique is Load Balancing. If an origin-destination pair (OD) is connected by several paths, the problem is simply how should traffic be distributed among these paths. This means, what portion of traffic (traffic distribution) should be sent through each of the available paths in order to fulfill a certain objective.

The simplest approach is to find a fixed traffic distribution. If the set of possible traffic matrices (TMs) were available, one can imagine static load-balancing schemes which can transit all of them, provided there is enough capacity in the network. There exist several proposals, differing both in the considered set and the objective. Notable examples are [1] and [2]. The former considers traffic matrices belonging to a so-called *polytope* and set the traffic distribution in order to minimize the biggest load in the network for the whole set. The latter considers that all nodes in the network can send and receive at a certain maximum amount of traffic rate, and by balancing load across all other nodes before arriving to its final destination, it can assure all traffic matrices are supported with a relatively small total capacity.

Although simple, naturally stable and **tolerant** to changes in the traffic matrix within the considered set, static load-balancing presents some disadvantages. First of all, the whole set of possible traffic matrices have to be available. This data is not easy to obtain, and will probably take months to do so. Moreover, if the set is not carefully chosen and at a given moment the network faces an unforeseen TM, the resulting performance is unpredictable. Even if this is not the case, since the scheme has to work for all traffic matrices, it will usually be the case that at any given moment there are wasted resources across the network. So, there is a trade-off between the size of the considered set and the performance under a given traffic matrix. Finally, optimizing under uncertainty is much more difficult than "normal" optimization. This increased difficulty forces the use of simpler optimization criteria and tend to result in a not so good performance (for instance, it is known that minimizing the biggest link utilization in the network generally leads to the use of long paths).

In order to remedy the above mentioned shortcomings and to make the network as flexible as possible, *dynamic* load balancing has been proposed and studied in the past. In this schemes, paths are configured a priori and the portion of traffic routed through each of them depends on the current traffic matrix and on the state of the network. This makes them more **robust** and **efficient** in resource usage. If the algorithm that adapts the portion of traffic routed through each path is also distributed (in the sense that each router makes its choices independent of the rest) the resulting scheme will also be **automated**. Actually, since TMs have a time-scale in the order of some few minutes [3], it is clear that as network size increases a centralized algorithm that solves the optimization problem for the current TM will not meet the time requirements. So, in this context, a distributed algorithm is not only desirable but, if the network's size is considerable, necessary. This distribution, however, leads to the problem of oscillations. If these decentralized, and thus uncoordinated, adjustments are not carefully designed, they can give place to severe unwanted oscillations, as it already happened with the early ARPAnet routing [4].

Although there are several proposals in this direction, we shall highlight the two most representative ones. In TeXCP [5], the objective is to minimize the biggest load in each path. A simplified version of the algorithm is that when a router sees less load in one of its paths, he increases the portion of traffic he routes through it. Such method can be seen as selfish load-balancing, where each router acts in its own interest. This kind of mechanisms can result in inefficient resource usage [6], and it is natural to believe that in the context of intra-domain TE routers should collaborate and not compete between them. In MATE [7] a certain function $\phi_l(\rho_l)$ is defined for each link l, which represents the cost of the link as a function of its current load ρ_l. The objective is then to minimize the total cost in the network defined as the sum of all link's costs. If $\phi_l(\rho_l)$ is convex, the resulting optimization problem is convex and can be solved in a distributed fashion. The specific cost function used is $\phi_l(\rho_l) = 1/(c_l - \rho_l)$, where c_l is the capacity of the link.

Although it is true that the total load traversing the links is very related with perceived performance, simply minimizing the total network's cost is not enough. For instance, consider the example in Fig. 1. In it, only the router serving source 1 has more than one path to choose from, link capacities are all the same and the demand generated by each source is also the same. It is relatively simple to show that for both of the above mentioned schemes, the optimum is that the traffic from source 1 is equally distributed among both paths. However, since the upper path "disturbs" two sources (source 3 and 4) while the lower one disturbs only one (source 2), it makes more sense from a fairness point of view to route more traffic from source 1 through the lower path.

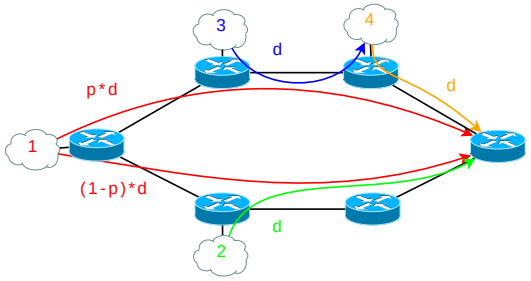

Fig. 1. An example in which if the total cost of the network is minimized the resulting optimum is unfair ($p_{opt} = 0.5$).

Inspired on TCP congestion control [8], particularly in the multi-path case [9], we propose that load-balancing (and more generally Traffic Engineering) should not have as an objective to minimize a certain *link-level cost*, but rather to maximize a *source-level utility*. If this utility function is carefully chosen, it will lead not only to good resource utilization, but also to fairness among sources. This paper is a first step towards defining such utility function and solving the resulting optimization problem.

The rest of the paper is organized as follows. The following subsection defines the network model and associated notation. In section 2.2 we present the utility objective function. In section 3 we address the resolution of the problem. Two approaches are presented. In section 3.1 we solve it in a centralized fashion, and in section 3.2 we outline a distributed algorithm. We present some flow-level simulations in section 4, where we show the advantages of the scheme over other TE techniques and the performance of the distributed algorithm. In section 5 we discuss some implementation issues, and present some packet-level simulations. We finish the paper on section 6.

2 Source-Level Utility Maximization

2.1 Network Model

In order to minimize the added complexity in the network, we will assume that the router through which traffic of a certain OD ingress the network is the one in charge of distributing this traffic among paths. We shall call this node the OD's *source node*, and in the sequel we will reference an OD pair by its source node.

We will represent the network as L unidirectional links, indexed by $l = 1 \dots L$. The capacity of each of these links is given by the column vector $c = [c_1 \dots c_L]^T$. There are a total of S sources in the network, indexed by $s = 1 \dots S$. Each source s has n_s possible paths to its destination, indexed by $i = 1 \dots n_s$. R_{si} is a column vector of length L, whose l-th entry is 1 only if source s uses link l in its i-th path, and 0 otherwise. Finally, we define $R_s = [R_{s1} \dots R_{sn_s}]$ and the incidence matrix is then $R = [R_1 \dots R_S]$.

All traffic in the network is assumed to be elastic (i.e. controlled by TCP). We will suppose flows arrive to source s as a Poisson process of intensity λ_s, each of them trying to transmit a certain workload (which has an arbitrary distribution of mean ω_s). This means that each source s generates a total demand $d_s = \lambda_s \omega_s$. Incoming flows will be routed through path si with probability p_{si}. Once their path is chosen, they will remain there until they are finished with their transmission. The demand in path si is then $p_{si} d_s = d_{si}$. The traffic distribution is defined simply as $d = [d_{11} \dots d_{1n_1} \dots d_{S1} \dots d_{Sn_S}]^T$, meaning that the total load on link l (ρ_l) can be easily calculated as the l-th entry of $R \times d$. Under these assumptions, if ρ_l is strictly smaller than c_l for all l, the number of flows in the network will not go to infinity [10], meaning that the network supports the given traffic distribution.

It is worth noting that we consider a dynamic context, in which flows appear and have a finite lifetime. This consideration is not always made in TE related research (see [11] for instance), but we consider it more realistic in the timescale (minutes) at which TE is interested. It is also important to highlight that we are enforcing flow-level load-balancing. Packet-level load-balancing (where packets from the same flow can take different paths) can have a negative impact on TCP performance due to packet reordering on the receiver's side.

2.2 The Utility Function

We shall consider that all traffic generated by a source as a unit. This means that its perceived performance is the performance perceived by the source from all its paths "as a whole". This assumption is true if, for example, we are interested in the communication between the OD routers, if we are doing packet-level load-balancing (which we are not) or if we take into account that each flow does not represent a user, since any of such flows is part of a bigger session.

Our proposal is to first define a certain revenue function $u_s(d)$ which measures the performance perceived by source s with traffic distribution d. The question is how this revenue should be distributed among sources. We could for instance maximize the average, or the smallest of them. A more general approach is to define a concave non-decreasing utility function $U_s(u_s)$ that represents the satisfaction source s has with its revenue $u_s(d)$, and maximize the sum over all sources. The problem in this most general version reads like this:

$$\underset{d}{\text{maximize}} \sum_{s=1}^{S} d_s U_s(u_s(d)) \tag{1}$$

$$\text{restricted to } Rd < c,\ d \geq 0 \text{ and } \sum_{i=1}^{n_s} d_{si} = d_s$$

We multiply by d_s each utility to give more weight to those nodes generating more traffic. The restrictions assure that the number of living flows is finite, that there are no negative demands and that all traffic is routed.

A typical example of U_s is the utility function that leads to the so-called α-fairness [12]:

$$U_s(x) = \begin{cases} (1-\alpha)^{-1} x^{1-\alpha}, & \alpha \neq 1 \\ \log x, & \alpha = 1 \end{cases} \tag{2}$$

The parameter α sets the fairness of the optimum. For $\alpha = 0$, it maximizes the sum of u_s, and for $\alpha \to \infty$ it results in max-min fairness.

Probably the most delicate part of the problem is to define $u_s(d)$. We can think of many possibilities, but a "good" u_s should have the following characteristics:

- Versatility in the sense that a low value should indicate bad performance for various criteria
- Easily measured and/or calculated by the node
- Mathematical properties that makes it amenable to optimization

A relatively simple path performance measure is its available bandwidth (ABW). The ABW of path si is defined as: $ABW_{si} = \min_{l \in si} \{c_l - \rho_l\}$. This indicator is twofold. On the one hand, is a rough estimator of the throughput TCP flows will obtain from the path [13, 14]. On the other hand, a path with a big ABW is a "healthy" path, in the sense that he can accommodate future unexpected increases in traffic. Our definition for u_s will be the average ABW seen by source s in all its paths. Substituting u_s in (1) results then in:

$$\begin{aligned} \underset{d}{\text{maximize}} \ & \sum_{s=1}^{S} d_s U_s \left(\sum_{i=1}^{n_s} p_{si} \min_{l \in si} \{c_l - \rho_l\} \right) \\ \text{restricted to } & Rd < c,\ d \geq 0 \text{ and } \sum_{i=1}^{n_s} d_{si} = d_s \end{aligned} \tag{3}$$

If we consider as true that TCP flows traversing path si will achieve a mean rate equal to ABW_{si}, the above problem is very similar to the multi-path TCP problem (see eq. 4 in [9]). In the rest of the paper we address its resolution, we outline an approximative distributed algorithm, and show its performance on some scenarios.

3 Solving the Problem

3.1 Centralized approach

It has been said that "*the great watershed in optimization isn't between linearity and non-linearity, but convexity and nonconvexity*" [15]. So, the first obvious approach in solving an optimization problem is to try and find an equivalent convex problem. This task has however proved impossible, for we could not find a convex problem with the same optimum as (3). What we could find was a similar convex problem that when combined with simple search heuristics yields a very good approximation to the optimum.

Let us take a look at the following problem:

$$\begin{aligned} \underset{d}{\text{maximize}} \ & \sum_{s=1}^{S} d_s U_s \left(\sqrt{\sum_{i=1}^{n_s} p_{si} \min_{l \in si} \{c_l - \rho_l\}} \right) \\ \text{restricted to } & Rd < c,\ d \geq 0 \text{ and } \sum_{i=1}^{n_s} d_{si} = d_s \end{aligned} \tag{4}$$

If $U_s(x)$ was like in (2), a simple change in the value of α in the exponent for this last problem would make its optimum and the one of (3) the same. On the other hand, a simple application of the Cauchy-Schwartz inequality yields that $\sqrt{\sum_i x_i} \geq \sum_i \sqrt{x_i}/\sqrt{n}$. Actually, as long as the elements of the sum are relatively similar, the upper bound is a good approximation. Substituting it in (4) results in:

$$\begin{aligned} \underset{d}{\text{maximize}} \ & \sum_{s=1}^{S} d_s U_s \left(\frac{1}{\sqrt{n_s}} \sum_{i=1}^{n_s} \sqrt{p_{si} \min_{l \in si} \{c_l - \rho_l\}} \right) \\ \text{restricted to } & Rd < c,\ d \geq 0 \text{ and } \sum_{i=1}^{n_s} d_{si} = d_s \end{aligned} \tag{5}$$

This approximated problem can be transformed into an equivalent convex problem. Since U_s as well as the square root are non-decreasing functions, we can introduce the auxiliary variable t_{si}:

$$\begin{aligned} \underset{d}{\text{maximize}} \ & \sum_{s=1}^{S} d_s U_s \left(\frac{1}{\sqrt{n_s}} \sum_{i=1}^{n_s} t_{si} \right) \\ \text{restricted to} \quad & t_{si} \leq \sqrt{p_{si}(c_l - d_l)} \Rightarrow \frac{t_{si}^2}{p_{si}} \leq c_l - \rho_l \quad \forall s, i \ \forall l : l \in si \\ & Rd < c \ ,\ d \geq 0 \text{ and } \sum_{i=1}^{n_s} d_{si} = d_s \end{aligned} \tag{6}$$

The objective is concave in t_{si}, the first restriction is convex (the quadratic over lineal function is convex in both variables [16]) and the rest of the restrictions are affine. This means that the optimum can be rapidly calculated by any standard convex solver.

The solution of (6) yields a very good estimation of the optimum of (3) (which we shall note d^*) as long as the corresponding $p^*_{si} ABW^*_{si}$ do not differ very much between paths of the same source. We already mentioned that the alternative objective function is a good approximation of the original one as long as all the elements in the sum are relatively similar. If d^* does not verify this condition the objective functions will be very different at it, thus the optimum of (6) will not result in d^*.

However, we will assume it a good starting point to search for the real optimum. Our simulations indicate that this is actually an excellent starting point, and that a simple steepest ascend method results in the optimum (for those simple cases that can be analytically calculated), or points where much more sophisticated methods obtain only marginal improvements.

This centralized solution is actually not a bad one, and can be useful for relatively small, not very dynamic networks. In such cases one can imagine a scheme where measurements of the traffic matrix are gathered periodically by a central entity, which performs the optimization and send every source node the new optimal traffic distribution. This is actually how most networks are managed today, with the period being days or months. We will see the advantages of our proposal over link-level cost functions in the simulation section.

3.2 Distributed Approach

Let us take another look at (3), using once again the auxiliary variable t_{si} and substituting the constraint on the load on each link by an equivalent restriction on t_{si}:

$$\begin{aligned} \underset{d}{\text{maximize}} \ & \sum_{s=1}^{S} d_s U_s \left(\sum_{i=1}^{n_s} p_{si} t_{si} \right) \\ \text{restricted to} \quad & t_{si} \leq c_l - \rho_l \quad \forall s, i \ \forall l : l \in si \\ & t_{si} > 0 \ , \ d \geq 0 \text{ and } \sum_{i=1}^{n_s} d_{si} = d_s \end{aligned} \tag{7}$$

Although all restrictions are affine, the objective function is not concave, meaning that methods solving the dual problem will only find a lower bound of the optimum. How near is this lower bound (i.e. how small is the duality gap) is closely related to the lack of concavity of the function [17]. Intuitively, the more concave the objective function, the smaller the duality gap. Anyway, calculating this lack of concavity can be very difficult. We estimated it by monte-carlo simulation for several possible network configurations, and found that it is relatively small and decreases with the number of paths. In the view of these results, we applied the well-known Arrow-Hurwicz method (see for instance [18]) and confirmed that the resulting traffic distribution is a very tight approximation of the optimum.

The method is an iterative one that at each step updates the value of the dual (primal) variables moving them in the direction of (opposite to) the gradient of the Lagrangian function. In this case, the Lagrangian function is:

$$\begin{aligned} L(p, t, \overline{\mu}, \underline{\mu}, \gamma, \lambda, \theta) = & - \sum_{s=1}^{S} d_s U_s \left(\sum_{i=1}^{n_s} p_{si} t_{si} \right) + \sum_{s=1}^{S} \mu_s \left(\sum_{i=1}^{n_s} p_{si} - 1 \right) - \\ & - \sum_{s=1}^{S} \sum_{i=1}^{n_s} (p_{si} \gamma_{si} + t_{si} \lambda_{si}) + \sum_{s=1}^{S} \sum_{i=1}^{n_s} \sum_{l : l \in si} \theta_{sil} (t_{si} - c_l + \rho_l) \end{aligned} \tag{8}$$

The Lagrange multipliers that enforces that p_{si} and t_{si} are positive (γ_{si} and λ_{si}), as well as the one that makes the addition of the p_{si} equal to one (μ_s), are not actually very important. Paths with zero ABW will not be used, and the conditions on p_{si} will at all stages of the algorithm necessarily be true (normalization should be carefully done, though). However, the multiplier θ_{sil} plays a very important role in the algorithm since it represents the cost of the link l generated by source s in its path i, resulting in a total cost $\widehat{\theta}_l = \sum_{s=1}^{S} \sum_{i : l \in si} \theta_{sil}$. The derivatives of (8) with respect to $p_{s_0 i_0}$ and $\theta_{s_0 i_0 l_0}$ are:

$$\frac{\partial L}{\partial p_{s_0 i_0}} = -d_{s_0} U'_{s_0} \left(\sum_{i=1}^{n_{s_0}} p_{s_0 i} t_{s_0 i} \right) t_{s_0 i_0} + \sum_{l \in s_0 i_0} \sum_{s=1}^{S} \sum_{i : l \in si} d_{s_0} \theta_{sil}$$

$$\frac{\partial L}{\partial \theta_{s_0 i_0 l_0}} = t_{s_0 i_0} - c_{l_0} + \rho_{l_0}$$

The auxiliary variable t_{si} does not have much physical meaning, except that for any given p its optimal value is the ABW in path si. The derivative on θ_{sil} does not tell us much then, except that it should decrease when l is not the bottleneck of si (which means that in such case its value should tend to zero). This forces us to estimate the value of θ_{sil}. Before discussing possible estimations, we will present the distributed algorithm:

- *Link's algorithm.* At times $t = 1, 2, \ldots$ link l:
  1. Receives path demands $d_{si}(t)$ from all sources using it, and estimates its total load $\rho_l(t)$.
  2. Computes its cost for each path $\theta_{sil}(t)$ and its total cost $\widehat{\theta_l}(t)$.
  3. Communicates this last value and its ABW to all sources traversing him.
- *Source's algorithm.* At times $t = 1, \ldots$ source s:
  1. Receives from the network the cost of all links he uses ($\widehat{\theta_l}(t)$) and their ABW.
  2. Computes the available bandwidth in each of its paths ($ABW_{si}(t)$).
  3. For each of its paths, he calculates the number:

$$\Delta_{si}(t) = d_s(t)U_s'\left(\sum_{j=1}^{n_s} p_{sj}(t)ABW_{sj}(t)\right)ABW_{si}(t) - d_s(t)\sum_{l\in si}\widehat{\theta_l}(t)$$

  4. He finds the path i_{max} with the biggest $\Delta_{si}(t)$ ($\Delta_s^{max}(t)$). He then updates each p_{si} in the following manner:

$$p_{si}(t+1) = [p_{si}(t) + \gamma(\Delta_{si}(t) - \Delta_s^{max}(t))]^+$$
$$p_{si_{max}}(t+1) = 1 - \sum_{\substack{i=1\ldots n_s \\ i\neq i_{max}}} p_{si}(t+1)$$

  where γ is a small constant.

If in step 4 the source finds that there is more than one path with maximum $\Delta_{si}(t)$, he distributes the "remaining" probabilities evenly between them.

What is left is finding a good estimation of θ_{sil}. We have several guidelines as to what this cost should look like:

- It should be zero unless link l is the bottleneck of path si.
- Should avoid as much as possible the use of information not local to the link.
- Big values should indicate that this particular link is very important to path si, and so sources should, if possible, avoid using it.

One possible cost function that presents almost all of the above mentioned characteristics (specially the third one) is the derivative of the total utility with respect to t_{si}. If the KKT conditions [16] where applicable in this problem, at optimality the derivative of (8) with respect to t_{si} should be zero. This means that at optimality for all $t_{s_0i_0} > 0$:

$$-d_{s_0}U_{s_0}'\left(\sum_{i=1}^{n_{s_0}} p_{s_0i}t_{s_0i}\right)p_{s_0i_0} + \sum_{l:l\in s_0i_0}\theta_{s_0i_0l} = 0$$

If path si would have only one bottleneck, there would only be one θ_{sil} not zero in the addition. This suggest the following estimation:

$$\theta_{sil} = \begin{cases} d_{si}U_s'\left(\sum_{i=1}^{n_s} p_{si}ABW_{si}\right) & \text{if } l = \underset{l\in si}{\operatorname{argmin}}\{c_l - \rho_l\} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

However, the link does not know the source's mean ABW, nor the utility function he uses. The latter makes necessary the assumption that all sources use the same utility function, known by all entities in the network. The former will force the link to make an estimation in order to maintain communication between elements in the network as restricted as possible. He will

assume all links in the network are as loaded as he is, in which case source s would have a mean ABW equal to his ABW. Then the link's estimation of θ_{sil} will finally be:

$$\theta_{sil} = \begin{cases} d_{si}U'(c_l - \rho_l) & \text{if } l = \underset{l \in si}{\operatorname{argmin}}\{c_l - \rho_l\} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Consider the case of $U(x) = \log(x)$. If the ABW was the service rate of a PS queue, the estimation can be seen as the average number of living flows in the system.

This is the cost function we will use, thus finishing the specification of the algorithm. As we will see in the next section, it yields a very good approximation of the optimum.

4 Fluid-Level Simulations

4.1 Centralized - The benefits of Maximizing the Utility

Before presenting simulations of the distributed algorithm, we will see some examples where the advantages of maximizing source-level utility over minimizing link-level cost can be appreciated.

We will begin by the case scenario in Fig. 1. In it, all links have a capacity equal to 5.0 and all sources generate the same demand d, except for source 1 who generates d_1. Finally, the utility function is $U(x) = \log(x)$. In Fig. 2 we can see the optimum value of p as a function of d_1 for different values of d. As can be appreciated in the graphs, while d_1 is relatively small compared to d and the ABW on the lower path is high enough, source 1 will only use this path. When any of this two conditions is not true, p will rapidly go towards 0.5, but always privileging better conditions on the upper path. Both MATE and TeXCP will have its optimum at $p = 0.5$ for all configurations.

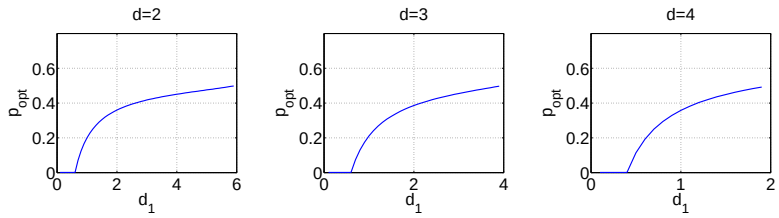

Fig. 2. The optimum value of p for different values of d in the network of Fig. 1

We will now make a comparison in a more complicated and neutral case. We will use Abilene, an academic network which consists of 12 nodes and 15 bidirectional links (see Fig. 3) all with a capacity of 100. The network is then rather small both in the number of nodes and links, and results obtained from it should not be considered as conclusive. However, it is one of those few real networks whose topology and several TMs are publicly available, and as such is an obvious first example to be studied. The paths we used were constructed by hand, trying to give sources as much path diversity as possible (i.e. presenting the source with disjoint paths when possible). The optimization for MATE was made using CVX [19], a MATLAB convex solver. For our proposition (UM from now on, as in Utility Maximization) we used the distributed algorithm. As we mentioned, when the number of sources is relatively big, the results obtained by the distributed algorithm are very good, even better than the ones obtained by the centralized heuristic we presented. As in the previous example, all source's utility function was $U_s = \log(x)$.

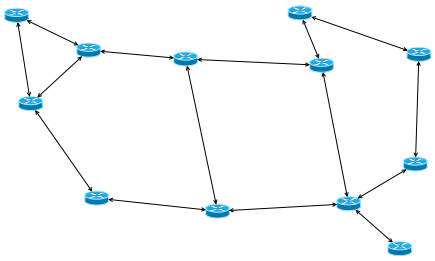

Fig. 3. The Abilene network topology

We will compare our proposition with MATE in terms of two indicators: mean ABW perceived by sources (u_s) and link load. To give a precise picture, for any given TM we will measure the mean, the 25% (75%) quantile and the minimum (maximum) u_s (link load). The comparison will be made by dividing between the value obtained by UM and the one obtained by MATE for each TM (gain of UM over MATE).

Most of the real TMs available do not load the network significantly, so a re-scaling is necessary. We used dataset X11 from [20] but with all its entries divided by 2.5×10^6, which yield a mean link load of around 25. In Fig. 4 we can see the boxplots of the u_s and link load indicators. Let us first turn our attention to the mean ABW indicators. The mean u_s is always bigger in UM than in MATE, although the difference is generally not very important, being mostly between 1-2%. However, there are some TMs for which the gain can be relatively significant, achieving values as big as 8%. The gain on the minimum u_s is more significant, being mostly between 3% and 12%. This is because generally the source with the smallest u_s is the source with the biggest d_s, thus it is the most "important" node in the optimization. Let us now consider the link load indicators. In this case the mean and 75% quantile link load are generally bigger in UM than MATE. However, the maximum link load is most of the times smaller for UM than MATE (as much as 13% smaller). This means that link load values in UM are concentrated while they are more dispersed in MATE, making UM's load distribution more balanced.

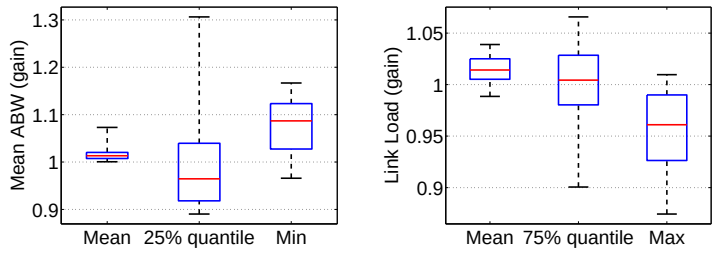

Fig. 4. Boxplot of the gain of UM over MATE for the u_s and link load indicators using real TMs

To further explore the relative performance for this topology we generated random TMs. In particular, we are interested in the effect of the number of active OD pairs. This study was done using the following simple model: an origin node will send traffic to a certain destination node with probability p, and the entry in the TM (i.e. the corresponding d_s) will be a random log-normally distributed number.

In Fig. 5 we can see the boxplots of u_s and the load indicators for 200 random TMs with $p = 0.05$ and mean d_s equal to 30. The mean u_s is mostly 1%-5% bigger in UM than in

MATE, although it can be as big as 16%. The minimum u_s shows better results, the gain being generally between 1% and 10% but achieving a maximum of 34%. On the other hand, similar to the previous example, the mean and 75% quantile load is most of the times bigger in UM than MATE, while the maximum load is generally smaller for UM than MATE.

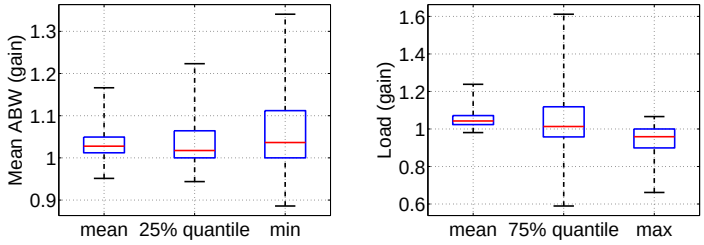

Fig. 5. Boxplot of the gain of UM over MATE for the u_s and load indicators for $p = 0.05$ and mean d_s equal to 30

Fig. 6 presents the results for 200 random TMs with $p = 0.25$ and mean d_s equal to 3. The gain obtained by UM over MATE for the u_s is smaller than in the previous case. However, the load tendency is maintained. The maximum load in the network is significantly smaller for UM than MATE, the difference being mostly between 5% and more than 20%.

From these two results we can conclude that as the number of active OD pairs augments, the two schemes show similar results. Intuitively, if all nodes communicate with each other, a scheme that looks at the paths' condition and one that considers the links' will not differ substantially.

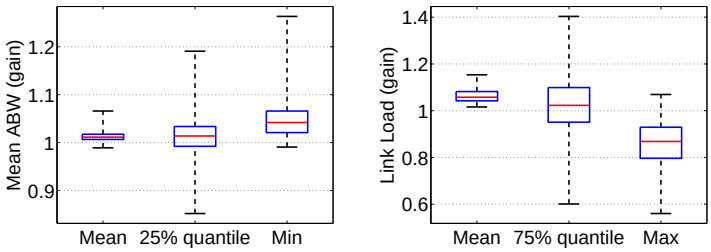

Fig. 6. Boxplot of the gain of UM over MATE for the u_s and load indicators for $p = 0.25$ and mean d_s equal to 3

As a conclusion from these experiments we may say that, for this topology, the mean u_s is almost always bigger with UM than MATE, although this increase is generally not very big. However, there are cases where the improvement can be important. On the other hand, the load is more evenly distributed among links in UM than MATE, resulting in a significantly smaller maximum link load.

4.2 Distributed

In this section we shall present some simple examples to see how the distributed algorithm converges, and how far from the optimum is the resulting traffic distribution. We shall first

present some fluid-level simulations to see how it behaves in an idealized context. We have also included some packet-level simulations to see the effect of unprecise and delayed measurements, which are presented in the next section.

The first example we will consider is the simplest one: a single source has two possible paths to choose from. Each of these paths have a capacity of 3.0 and 4.0 respectively. In Fig. 7 we can see the values of p_1 (the portion of traffic routed through the path with the smallest capacity) and p_2, both for the distributed and the centralized algorithm, as a function of the demand generated by the source.

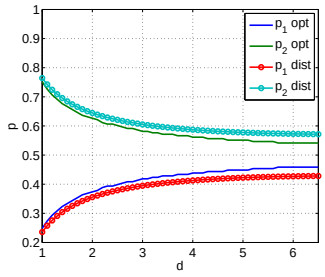

Fig. 7. p_1 and p_2, both for the centralized and distributed algorithm, in a two paths single source scenario

The first remarkable thing is that the distributed algorithm approximates very well the optimum, being the biggest difference less than 0.05. The second aspect that is worth noting is that the distributed algorithm always tends to "over-use" the widest path. This can be explained by the approximation we made from (9) to (10). Since $U(x)$ is concave, $U'(x)$ is a non-increasing function, meaning that if $x_1 > x_2$ then $U'(x_1) \leq U'(x_2)$. So, when a link has an ABW bigger than the source's average, its estimation of the price will be smaller than it originally would be. In this case, it means that link 1 will calculate a smaller price than he should, resulting in the source sending more traffic through it than the optimum.

Consider now the example in Fig. 8. In it, all links have a capacity of 4.0. Source 2 generates a total demand of 1.0 (d_2), and we analyze what is the optimum traffic distribution while varying d_1. Notice how the ABW on the lower path is the same as in the last example, but how source 1 concentrates more of its traffic in the wider path than before. It is also worth noting that in this case the distributed algorithm approximates even more the global optimum, so much that it is very difficult to tell them apart in the graph.

In Fig. 9 we present the evolution of the traffic distribution with the network of Fig. 1. In this particular example, the demand generated by source 1 is $d_1 = 2$, while the rest of the sources generate $d = 3$. We used $\gamma = 1.5 \times 10^{-3}$. We can see two things from this example. First, and as can be seen comparing the converged probability and Fig. 2 (b), the global optimum and the probability resulting from the distributed algorithm are very similar. Secondly, the initial traffic distribution is not supported by the network, which does not prevent the algorithm from rapidly moving to a supported distribution, slowly converging towards the optimum afterwards.

5 Implementation Issues

So far we have described the algorithm theoretically, without discussing practical issues. The first clear necessity of the algorithm is that border routers should be able to send arbitrary portions of traffic through the different paths. Secondly, in order to measure d_{si}, interior routers should distinguish between traffic belonging to a given path that is traversing its links.

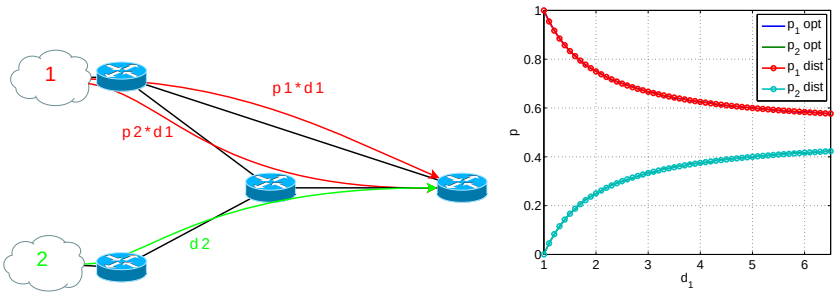

Fig. 8. The example topology, and its optimum traffic distribution as function of d both for the centralized and distributed algorithm.

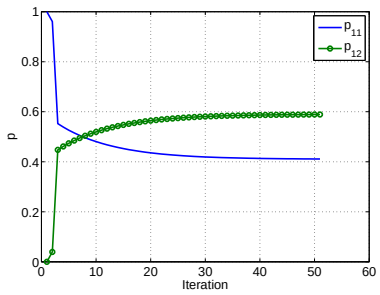

Fig. 9. Probability of sending through the upper path (p_{11}) and the lower one (p_{12}) for the example in Fig. 1

These requirements are accomplished for instance by MPLS. Hashing can be used in order to load-balance traffic with an arbitrary distribution. Packets belonging to a given si can be identified by its label header. Interior routers should then keep a counter indicating the number of bytes they have routed of a given label. They periodically calculate the corresponding d_{si} that had traversed them by dividing this counter between the measurement period time, after which they reset the counter. In order to avoid measurements that are too noisy, some filtering should be applied. In our simulations, a simple exponential filter is enough. The total load ρ_l is then calculated as the sum of all d_{si} that use the link. Available Bandwidth of link l can be easily calculated as the difference between the total capacity and this value. However, and in order to avoid numerical problems, all calculations that includes it should use the maximum between the ABW and a relatively small value (for instance $c_l/100$).

Another important aspect is the communication between link and source. The source communicates with the link implicitly, since communication in that sense is simply how much traffic source s is sending through link l. It is true that what actually reaches the link will always be smaller or equal than originally at the source node, but such an approximation is not too important.

The most challenging communication is from the link towards the source. We will use the same approach as TeXCP and use probe packets, which in our case will contain the path's ABW and total cost ($\sum_{l:l\in si} \widehat{\theta}_l$). Periodically, the source sends probe packets, initially indicating as the path's ABW and cost ∞ and 0.0 respectively. As this packet advances from the source towards the destination, each interior router sees which is the ABW indicated in it. If he sees that the packet has a smaller value than the one of the outgoing link, he will overwrite it and put this new value instead (and "remember" it). When the probe packet reaches the destination,

it goes back to the source exactly in the opposite direction. As it is going back to the source, each interior router will check whether the final ABW indicated on the packet is the one the router had when it first passed. If so, it means that he is the bottleneck of the particular path. He then calculates θ_{sil} accordingly, update the link's total cost $\widehat{\theta}_l$, and add this value to the total path cost indicated on the packet. As a result of this, the source will finally receive the path's ABW and total cost.

5.1 A Simulation

We implemented the algorithm in an ns-2 [21] simulation script to see its performance in a realistic context. In Fig. 10 we can see a simple case scenario. There are two sources and each of them can use two paths, one of which is shared with the other. All links have a capacity of 1.0 Mbps, except for the "access" ones which have 2.0 Mbps. Traffic consists of elastic flows with an exponentially distributed size whose mean is 20kB, arriving as a Poisson process of the corresponding intensity so as to generate a given demand. The chosen utility function is the natural logarithm. The exponential filter's parameter is set to $\alpha = 0.7$, and γ is set to 5×10^{-9}. The initial probabilities are set on both sources so that the shared link is not used. This will maximize the likeness of oscillations. Load measurement periods should always be smaller (several times smaller) than the probability update period. It is important that the source receives measurements as updated as possible in order to see the effects of the previous probability update. Probabilities are updated every 50 seconds for both sources and the load measurements are made every 10 seconds. Sources are however not coordinated between them and update their probabilities at different moments. Both sources generate the same demand, approximately 1.1 Mbps, which the network cannot initially support. The optimal distribution is then that both sources send a third of their traffic through the shared path.

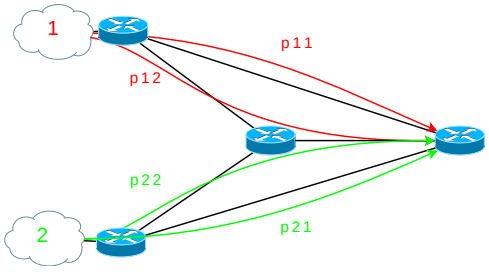

Fig. 10. The example topology

As can be seen in Fig. 10, both sources at first rapidly change their probability to start using the middle path. It take them a little while to realize that another source is also using this path, and start augmenting the direct path probability, but slower than before, since the price difference between them is not so big now. After a little longer the probabilities finally converge to the optimum. This whole process takes approximately 15 minutes, not too much considering the time-scale and that it is an extreme case. It is important to notice that load measurements need not be very precise, and that the algorithm supports a little bit of noise. This suggest that maybe some of the measured variables (e.g. d_{si}) could be inferred from more "global" and readily available measurements (e.g. ρ_l) without affecting the algorithm's performance.

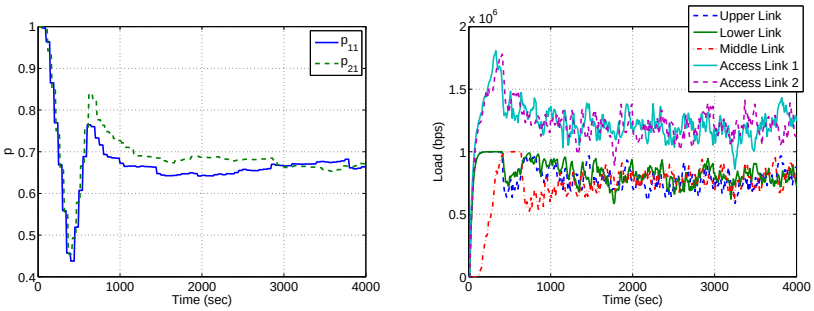

Fig. 11. The evolution of p and link load over time

6 Conclusions

In this work we presented a TE mechanism that takes into account the needs of both the network operator (uncongested links) and the users (uncongested paths, leading to faster transfers). We achieved this by defining an objective function that is not a cost at a *link level*, but a utility at a *user level* (actually, at the border router level, the source). This lead to an optimization formulation very similar in spirit to Multi-Path TCP [9], where we maximize the sum of an utility function whose argument is the average ABW each source sees in its path. The resulting optimization problem was not convex. Two kinds of resolution algorithms were presented: a centralized and a distributed one. The centralized one can be applied to relatively small, not too dynamic networks. As the network increases in size, the centralized algorithm is no longer a solution. For this case, a distributed algorithm was outlined which, although based on a resolution method for convex problems, finds very tight approximations to the optimum in a relatively short time.

When compared with other load-balancing mechanism, the obtained distribution improves perceived performance with a more balanced load distribution. The obtained results were promising but we tested our proposition in only one relatively complicated network. This means that these results are not conclusive, and more topologies and TMs must be explored, including real as well as randomly generated ones.

There are many more things left to do. We considered that only elastic traffic is present in the network, and the utility function was designed with this in mind. However, streaming traffic does exist, and its requirements are different from elastic one. Although our scheme indirectly avoids paths with big queueing delays and losses, the propagation delay should also be taken into account. If for instance a source has two paths to choose from, he could prefer the shorter one if the ABW is enough in it, which would actually benefit both streaming and elastic traffic. Maybe a small change in the utility function is enough. We are currently investigating this possibility.

With respect to the distributed algorithm, there are also many things to investigate. A convergence analysis should be made, and the stability of the algorithm analyzed. Taking into account that potential oscillations are what make operators most uneasy when offered with this kind of adaptive schemes, this study is very important.

When applying the distributed algorithm, one rapidly realizes that the value of γ (the adaptation step) is very important. This value indicates how fast the probabilities adapt. A very big value makes the algorithm unstable, while a very small one makes it unresponsive. The problem is that a "good" choice of γ depends on the network topology, but also on the current load. A value that works when the network is too congested, may make the network unresponsive when the network is lightly loaded. In this last case one may think that is not really important what the traffic distribution is, or at least is not very urgent to change it to

the optimum. However, and in order to make the algorithm as self-configured as possible, we are exploring a possible alternative mechanism, where probabilities are always augmented in the same quantity, independent of the current load.

Another practical aspect to be explored is the need of all the measurements. Our simulations indicate (see Fig. 11) that loads are not required to be measured too precisely for the algorithm to work. Maybe link prices can be inferred from the total's link load. If this can be done, we may be able to remove the need of MPLS, making the algorithm less technologically specific. We will still need a way to send arbitrary portions of traffic through specific paths. Maybe an alternative to MPLS could be Muti-Topology Routing [22].

7 Acknowledgements

We would like to thank Paola Bermolen, Irène Charon, Olivier Hudry, Jim Roberts and Sara Oueslati for the fruitful discussions that enabled this report to be written.

References

1. Ben-Ameur, W., Kerivin, H.: Routing of uncertain traffic demands. Optimization and Engineering **6**(3) (september 2005) 283–313
2. Zhang-Shen, R., McKeown, N.: Designing a predictable internet backbone with valiant load-balancing. In: IWQoS. (2005) 178–192
3. Uhlig, S., Quoitin, B., Lepropre, J., Balon, S.: Providing public intradomain traffic matrices to the research community. SIGCOMM Comput. Commun. Rev. **36**(1) (2006) 83–86
4. Khanna, A., Zinky, J.: The revised arpanet routing metric. SIGCOMM Comput. Commun. Rev. **19**(4) (1989) 45–56
5. Kandula, S., Katabi, D., Davie, B., Charny, A.: Walking the tightrope: responsive yet stable traffic engineering. In: ACM SIGCOMM '05. (2005) 253–264
6. Roughgarden, T., Éva Tardos: How bad is selfish routing? J. ACM **49**(2) (2002) 236–259
7. Elwalid, A., Jin, C., Low, S., Widjaja, I.: MATE: MPLS adaptive traffic engineering. INFOCOM 2001 **3** (2001) 1300–1309
8. Kelly, F., Maulloo, A., Tan, D.: Rate control in communication networks: shadow prices, proportional fairness and stability. In: Journal of the Operational Research Society. Volume 49. (1998)
9. Key, P., Massoulie, L., Towsley, D.: Path selection and multipath congestion control. In: IEEE INFOCOM 2007. (May 2007) 143–151
10. Bonald, T., Massoulié, L.: Impact of fairness on internet performance. In: ACM SIGMETRICS '01. (2001) 82–91
11. He, J., Bresler, M., Chiang, M., Rexford, J.: Towards robust multi-layer traffic engineering: Optimization of congestion control and routing. Selected Areas in Communications, IEEE Journal on **25**(5) (June 2007) 868–880
12. Mo, J., Walrand, J.: Fair end-to-end window-based congestion control. IEEE/ACM Trans. Netw. **8**(5) (2000) 556–567
13. Bonald, T., Massoulié, L., Proutière, A., Virtamo, J.: A queueing analysis of max-min fairness, proportional fairness and balanced fairness. Queueing Syst. Theory Appl. **53**(1-2) (2006) 65–84
14. Bonald, T., Proutière, A.: On performance bounds for balanced fairness. Perform. Eval. **55**(1-2) (2004) 25–50
15. Rockafellar, R.T.: Lagrange multipliers and optimality. SIAM Rev. **35**(2) (1993) 183–238
16. S. Boyd and L. Vandenberghe: Convex Optimization. Cambridge University Press (2004)
17. Pappalardo, M.: On the duality gap in nonconvex optimization. Math. Oper. Res. **11**(1) (1986)
18. M. Minoux: Programmation Mathématique: théorie et algorithmes. Dunod (1983)
19. Grant, M., Boyd, S.: CVX: Matlab software for disciplined convex programming (March 2008)
20. Yin Zhang: "Abilene Dataset" http://www.cs.utexas.edu/∼yzhang/research/AbileneTM/.
21. : The Network Simulator - ns http://nsnam.isi.edu/nsnam/index.php/Main_Page.
22. Kvalbein, A., Lysne, O.: How can multi-topology routing be used for intradomain traffic engineering? In: INM '07: Proceedings of the 2007 SIGCOMM workshop on Internet network management. (2007)