A Fair and Dynamic Load-Balancing Mechanism

Federico Larroca
TELECOM ParisTech
Paris, France
federico.larroca@telecom-paristech.fr

Jean-Louis Rougier
TELECOM ParisTech
Paris, France
rougier@telecom-paristech.fr

***Abstract*—The 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. By providing origin-destination (OD) pairs with several possible paths, load-balancing has proven itself an excellent tool to face this uncertainty. Most previous proposals defined the load-balancing problem as minimizing a certain network cost function of the link's usage, assuming users would obtain a good performance as a consequence. Since the network operator is interested in the communication between the OD nodes, we propose instead to state the load-balancing problem in their terms. We define a certain utility function of the OD's perceived performance and maximize the sum over all OD pairs. The solution to the resulting optimization problem can be obtained by a distributed algorithm, whose design we outline. By means of extensive simulations with real networks and traffic matrices, we show that our approach results in more available bandwidth for OD pairs and a similar or decreased maximum link utilization than previously proposed load-balancing schemes. Packet-level simulations verify the algorithm's good performance in the presence of delayed and inexact measurements.**

I. Introduction

Network convergence is a reality. Many new services such as P2P or HD-TV are offered on the same network, increasing the unpredictability of traffic patterns. To make matters worse, access rates have increased at such pace that the old assumption that core link capacities are several orders of magnitude bigger than access rates is no longer true. Thus, simply upgrading link capacities may not be an economically viable solution any longer. This means that network operators are now, more than ever, in need of Traffic Engineering (TE) mechanism which are **efficient** (make good use of resources), but also **automated** (as much self-configured as possible), more **robust** with respect to network variations (changes in traffic matrix, or characteristics of transported flows) and more **tolerant** (in case of node/link failures).

Dynamic load-balancing [1]–[3] is a TE mechanism that meets these requirements. If an origin-destination (OD) pair is connected by several paths, the problem is simply how to distribute its traffic among these paths in order to achieve a certain objective. In these dynamic schemes, paths are configured a priori and the portion of traffic routed through each of them (traffic distribution) depends on the current traffic matrix (TM) and network's condition. As long as the traffic distribution is updated frequently enough, this kind of mechanism is **robust** and their dependence on the network's condition makes them naturally **tolerant** too. Finally, if the algorithm is also distributed (in the sense that each router makes its choices independent of the others) the resulting scheme will also be **automated**.

In intra-domain TE, the network operator is interested in the communication between the OD nodes, i.e. the performance they get from their paths. The OD pairs may actually be regarded as the users of the network, sharing its resources between them. It is natural then to state the load-balancing problem (or TE in general) in their terms. An analogy can be made with the congestion control problem [4], where the users are the end-hosts and share link capacities. The user's performance (or "revenue") is the obtained rate and the objective is to maximize the sum over all users of a utility function of it. In our case the problem is different since the rate is given and we control only the portion of traffic sent through each path. In this paper we propose to measure the user's performance by the mean available bandwidth (*ABW*) the OD pair obtains in its paths, and then maximize the sum over all pairs of a utility function of this measure. We will present a distributed algorithm in which the independent adjustments made by each OD pair lead to the global optimum. Our comparison with previously proposed load-balancing schemes, using several real networks and TMs, shows that the resulting traffic distribution improves OD pairs' perceived performance and decreases maximum link utilization.

Almost all prior proposals in load-balancing (and in TE in general) define a certain link-cost function of the link's capacity and load, and minimize the total network's cost defined as the sum over all links of this function. The resulting traffic distribution will be relatively balanced, in the sense that no single link will be extremely loaded. However, it is not the situation of isolated links, but the condition on the complete path(s) connecting OD nodes that counts. Solving the problem in terms of the links is only an indirect way of proceeding which does not allow us, for instance, to prioritize a certain OD pair or to enforce fairness among the OD pairs. For example, consider the network in Fig. 1. In it, all link capacities are equal and all sources generate the same amount of traffic. However, only OD pair 1 has more than one path to choose from. It is relatively simple to verify that if the link-cost function is the same in all links, the optimum is obtained when traffic from OD pair 1 is equally distributed among paths. However, since the upper path "disturbs" two OD pairs while the lower one disturbs only one, depending

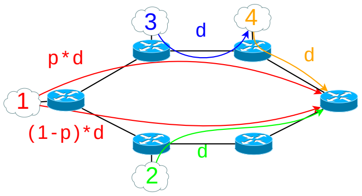


Fig. 1. An example in which if the total cost of the network is minimized the resulting optimum is unfair ($p_{opt} = 0.5$).

on our fairness definition it could make more sense to route more traffic from OD pair 1 through the lower path.

The rest of the paper is organized as follows. The following section discusses related work. Section III defines the network model and associated notation, followed by the presentation of the utility objective function. In Sec. IV we address the resolution of the problem. We present some flow-level simulations in Sec. V, where we show the performance of the distributed algorithm and the advantages of our scheme over other TE techniques. In Sec. VI we discuss some implementation issues, and present some packet-level simulations. We conclude the paper in Sec. VII.

II. Related Work

Load-Balancing can be seen as a particular case of Multi-Path routing. Many papers fall under this category, but we will highlight only those closely related to our work, or which inspired us for our proposal. We will further classify research in this topic into two sub-categories depending on the time-scale under consideration.

A. Long Time-Scale

Here we consider the minute or hour time-scale of routing. A TM is assumed to exist and is used as an input to the problem. At this time-scale congestion control is not considered, i.e. the TM is assumed independent of the current condition of the network. However, it is clear that the TM is not static and changes over time [5]. Furthermore, some network operators offer their customers VPN services in which they only have to specify the total maximum rate each node can receive or send [6]. Load-Balancing has proved itself a very effective tool to cope with this TM uncertainty, and research in this area differs mainly in choosing which uncertainty set contains the real TM at all times. Robust routing considers a well-defined set, while dynamic load-balancing only the current TM.

The objective in robust routing is to find a unique static routing configuration that fulfills a certain criteria, generally the one that minimizes the maximum link utilization over all TMs of the corresponding uncertainty set. The set can be for instance the TMs seen the previous day, or the same day the previous week [7]. A very used option is the polytope proposed in [8] which allows for easier and faster optimization. In any case, since a single traffic distribution that works for all TMs is used, resources will be wasted for any specific TM. Shrinking the uncertainty set results in improved performance, and there are some papers in this direction [9], [10]. This shrinking should be carefully done though, because if the selected set is too small and the network faces an unforeseen TM, the resulting performance is unpredictable. Finally, optimizing under uncertainty is more difficult than “normal” optimization. This increased difficulty forces the use of simpler optimization criteria which can lead to a not so good performance (e.g. it is known that minimizing the biggest link utilization generally results in the use of longer paths).

In dynamic load-balancing, each origin node estimates its entries in the TM and, based on feedback from the network, adapts the portion of traffic it sends through each path. After some iterations, and if the TM does not change in the meantime, a global optimum is achieved. The two most well-known proposals in this area are MATE and TeXCP. In MATE [1], a convex link cost function is defined, which depends on the link’s capacity and load. The objective is to minimize the total network cost, for which a simple gradient descent method is proposed. TeXCP [2] proposes a somewhat simpler objective: minimize the biggest utilization (ρ_l/c_l) each OD pair sees in its paths. A rough description of the algorithm is that origin nodes iteratively increase the portion of traffic sent through the path with the smallest utilization. Another load-balancing scheme which has the same objective but a relatively different mechanism is REPLEX [3].

B. Short Time-Scale

This short time-scale refers to the congestion control time-scale. Possible adaptations of TCP to the multi-path case (MP-TCP) have been extensively studied, where the utility each user perceives is now a function of the total rate he obtains from all his paths. Several propositions exist in this direction. For instance, in [11]–[13] the user is responsible of calculating his total rate and how much he should send through each path. In [14], the user only calculates the total sending rate and the routers distribute traffic among paths.

A different but related problem is a user downloading the same file from different sites or hosts (as in Bittorrent). Currently, greedy policies are used where users change a path only if they obtain a better performance on the new one. In [15] the authors show that if current TCP flavors are used in such schemes, the resulting allocation can be both unfair and inefficient, and that a mechanism similar to MP-TCP should be used instead.

In [16] the objective is to adapt the sending rates to maximize the total users’ utility minus a network cost. The idea is that users should also take into account the utilization of the links and leave a margin for future arrivals. We believe that this is not the best criteria. Congestion control should enable users to consume *all* their fair-share of the path. At this time-scale, saving a little bandwidth for future arrivals is, in our opinion, a waste of resources.

Although MP-TCP constitutes a very interesting long-term objective, no actual implementations of it exists. Allowing end-hosts to choose their paths, or even making them aware that several possibilities exist, presents several technical difficulties in current Internet architectures.

III. Source-Level Utility Maximization

A. Network Model

We represent the network as L unidirectional links, indexed by l, whose capacities are given by the column vector $c = [c_1 \dots c_L]^T$. We will reference OD pairs by index $s = 1 \dots S$. By abuse of notation we will also reference by s its *source node*, defined as the router through which its traffic ingress the network. This node will be in charge of distributing this traffic among paths, and in the sequel we will use the terms source and OD pair without differentiation. Each source s has n_s possible paths towards its destination, indexed by i. R_s is a $L \times n_s$ matrix whose li entry is 1 only if source s uses link l in its i-th path, and 0 otherwise. 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 suppose that flows arrive to source s as a Poisson process of intensity λ_s. Each of these flows consist of a random arbitrarily distributed workload (with mean ω_s) they want to transfer, generating a demand $d_s = \lambda_s \omega_s$. Each flow is routed through path P_{si} with probability p_{si}, and it uses it throughout its lifetime. It is worth noting that we consider a dynamic context, in which flows appear and have a finite lifetime. 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) may have a negative impact on TCP performance due to packet reordering on the receiver's side.

The demand on path si is then $p_{si} d_s = d_{si}$. The traffic distribution is defined simply as $d = [d_{11} \dots d_{1n_1} \dots d_{S1} \dots d_{Sn_S}]^T$, and 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 [17], meaning that the network supports the given traffic distribution.

B. The Utility Function

Since we consider the OD pairs as the users of the network, a single performance indicator per pair should be used. Even if we considered end-hosts as the users, a single indicator per OD pair is also adequate since traffic belonging to a given OD pair is composed of many flows generated by several end-hosts.

Our proposal defines first a revenue function $u_s(d)$ which indicates the performance perceived by source s when the traffic distribution is d. The question is how this revenue should be distributed among sources. We could for instance maximize the average or the smallest of them. Drawing on the work on congestion control [4], we 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:

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

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

We multiply each utility by d_s to give more weight to those nodes generating more traffic. The constraints 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 $U(x)$ is the utility function that leads to the so-called α-fairness [18]:

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

Throughout our simulations we will use $\alpha = 1$, which results in proportional fairness [19].

Probably the most delicate part of the problem is defining $u_s(d)$. A relatively simple path performance measure is its available bandwidth (*ABW*). The *ABW* of path si is defined as $ABW_{si} = \min_{l \in si} \{c_l - \rho_l\}$. The meaning of this indicator is twofold. On the one hand, it is a rough estimator of the throughput TCP flows will obtain from the path [20], [21]. On the other hand, a path with a big *ABW* is a "healthy" path, in the sense that it 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, which presents a good balance between current good conditions and prudence. Substituting u_s in (1) results in:

$$\underset{d}{\text{maximize}} \sum_{s=1}^{S} d_s U_s \left(\sum_{i=1}^{n_s} p_{si} \min_{l \in si} \{c_l - \rho_l\} \right) \tag{3}$$

This version of problem (1) is very important for the elastic traffic case. Although TCP takes care of path's resource sharing, routing constitutes a degree of freedom in the obtained rate that may be taken into account. Since the mean obtained rate depends on the amount of traffic the flow is sharing its path with, this obtained rate may be indirectly controlled through routing. Let us assume that this relation is simply that TCP flows traversing path si achieve a mean rate equal to ABW_{si}. Then problem 3 is very similar to the multi-path TCP one (see Eq. 4 in [15]) where each OD pair is seen as serving d_s MP-TCP flows. Notable differences are that the decision variable is the *portion* of traffic sent through each path (and not the *amount* of traffic), and that the mean of the flow rate is used. However, by using standard TCP and changing the ingress nodes only (not all end-hosts), users can now be regarded *as if* they were using MP-TCP, with all the advantages that this means (better performance and more supported demands).

IV. Solving the Problem: Distributed Algorithm

As the network's size increases, a centralized algorithm that solves (3) for the current TM does not scale well. In this context, a distributed algorithm is not only desirable but, if the network's size is considerable, necessary. In this section we present a distributed solution for our problem.

Let us rewrite (3) introducing the auxiliary variable t_{si}:

$$\begin{aligned} \underset{d,t}{\text{maximize}} \quad & \sum_{s=1}^{S} d_s U_s\left(\sum_{i=1}^{n_s} p_{si} t_{si}\right) \qquad (4)\\ \text{subject to} \quad & t_{si} \le c_l - \rho_l \quad \forall s,i\ \forall l : l\in si \\ & t_{si} > 0\ ,\ d \ge 0 \text{ and } \sum_{i=1}^{n_s} d_{si} = d_s \end{aligned}$$

Although all constraints are affine, the objective function is not concave, meaning that methods solving the dual problem will only find a lower bound for the optimum. How tight is this lower bound (i.e. how small is the duality gap) is closely related to the lack of concavity of the function [22]. Our estimations indicate that in this case the lack of concavity is relatively small and decreases with the number of paths. In view of these results, we applied the well-known Arrow-Hurwicz method [23] and confirmed that the resulting traffic distribution is a very tight approximation to the optimum.

This method is iterative and 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:

$$\begin{aligned} L(p,t,\theta) = -\sum_{s=1}^{S} d_s U_s\left(\sum_{i=1}^{n_s} p_{si} t_{si}\right) + & \\ \sum_{s=1}^{S}\sum_{i=1}^{n_s}\sum_{l:l\in si} \theta_{sil}(t_{si} - c_l + \rho_l) & \qquad (5) \end{aligned}$$

Paths with zero *ABW* will not be used by the algorithm and the conditions on p_{si} will be necessarily true (normalization should be carefully done, though). Since the constraints are enforced by the algorithm (presented below), we omitted the Lagrange multipliers associated with the positiveness of p_{si} and t_{si}, and the normalization condition on p_{si}. However, θ_{sil} plays a very important role since it represents the cost of link l generated by source s in its path i, resulting in a total cost of $\widehat{\theta_l} = \sum_{s=1}^{S}\sum_{i:l\in si}\theta_{sil}$.

The derivatives of (5) with respect to $p_{s_0 i_0}$ and $\theta_{s_0 i_0 l_0}$ are:

$$\begin{aligned} \frac{\partial L}{\partial p_{s_0 i_0}} &= -d_{s_0} U'_{s_0}\left(\sum_{i=1}^{n_{s_0}} p_{s_0 i} t_{s_0 i}\right) t_{s_0 i_0} + d_{s_0} \sum_{l\in s_0 i_0} \widehat{\theta_l} \\ \frac{\partial L}{\partial \theta_{s_0 i_0 l_0}} &= t_{s_0 i_0} - c_{l_0} + \rho_{l_0} \end{aligned}$$

The auxiliary variable t_{si} does not have much physical meaning, except that for any given p its optimal value is ABW_{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 (meaning 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 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 traversing sources.

Source algorithm. At times $t = 1, 2, \ldots$ source s:

1) Estimates its current demand $d_s(t)$.
2) Receives from the network the cost $\widehat{\theta_l}(t)$ of all links it uses and their *ABW*.
3) Computes the available bandwidth of its paths ($ABW_{si}(t)$) and its mean *ABW* ($u_s(t)$).
4) For each of its paths, it calculates the number:
$$\Delta_{si}(t) = d_s(t) U'_s\left(u_s(t)\right) ABW_{si}(t) - d_s(t)\sum_{l\in si}\widehat{\theta_l}(t)$$
5) It finds the path i_{max} with the biggest $\Delta_{si}(t)$ ($\Delta_s^{max}(t)$). It then updates each p_{si} in the following manner (where γ is a small constant):
$$\begin{aligned} p_{si}(t+1) &= \left[p_{si}(t) + \gamma(\Delta_{si}(t) - \Delta_s^{max}(t))\right]^{+} \\ p_{si_{max}}(t+1) &= 1 - \sum_{\substack{i=1\ldots n_s \\ i\neq i_{max}}} p_{si}(t+1) \end{aligned}$$

We will now discuss possible estimations of θ_{sil}. The Karush-Kuhn-Tucker (KKT) conditions [23] state that the derivative of (5) with respect to t_{si} evaluated at the optimum should necessarily be zero. This means that at optimality:

$$\frac{\partial L}{\partial t_{s_0 i_0}} = -d_{s_0} U'_{s_0}\left(\sum_{i=1}^{n_{s_0}} p_{s_0 i} t_{s_0 i}\right) p_{s_0 i_0} + \sum_{l:l\in s_0 i_0} \theta_{s_0 i_0 l} = 0$$

If path si only has one bottleneck, there would only be one nonzero $\theta_{s_0 i_0 l}$ in the addition, a fact that may be used to make a first estimation of θ_{sil}. However, the link does not know the source's mean *ABW*. To maintain communications between elements in the network as restricted as possible, links will assume that all the sources that use them have a mean *ABW* equal to their *ABW*. The link's estimation of θ_{sil} will then be:

$$\theta_{sil} = \begin{cases} d_{si} U'\left(c_l - \rho_l\right) & \text{if } l = \underset{l\in si}{\text{argmin}}\{c_l - \rho_l\} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

We have assumed, for simplicity's sake, that all sources use the same known utility function. This is the cost function we will use, thus finishing the specification of the algorithm. As we will see in Sec. V-A, the consequences of this approximation are not significant and the algorithm yields a very good estimation of the optimum. Details on how to implement this algorithm in a real network are discussed in Sec. VI.

V. Fluid-Level Simulations

A. Distributed Algorithm Performance

In this section we shall present some simple examples to gain some insight into the proposed framework and to verify that the resulting traffic distribution of the distributed algorithm is not far from the actual optimum. We first present fluid-level simulations to verify its behavior in an idealized

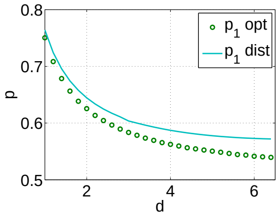


Fig. 2. Portion of traffic through the widest path in a two path single source scenario

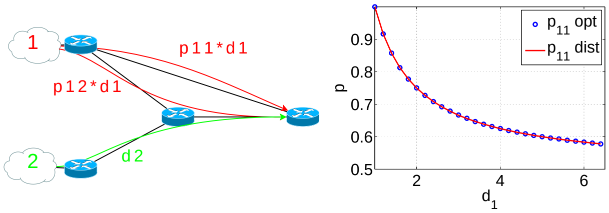


Fig. 3. The example topology, and its optimum traffic distribution as a function of d.

context. We have also included some packet-level simulations to analyze the effect of imprecise 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. The two paths have a capacity of 3.0 and 4.0 respectively. In Fig. 2 we can see the value of p_1 (the portion of traffic routed through the path with the biggest capacity) obtained by the distributed algorithm and the actual optimum, as a function of the demand generated by the source.

We first remark 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 in (6). 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 should. In this example, it means that link 1 will calculate a smaller price, which results in the source sending more traffic through it than at optimality.

Consider now the example in Fig. 3. In it, all links have a capacity of 4.0. Source 2 generates a total demand $d_2 = 1.0$, and we analyze the optimum traffic distribution while varying d_1 (the demand generated by source 1). Notice how, even if the *ABW* source 1 sees on the lower path is the same as in the last example, it concentrates more traffic in the wider path than before. The presence of source 2 makes the lower path more “expensive”. Also note that in this case the distributed algorithm approximates even better the global optimum.

Finally, we analyze some examples in the network of Fig. 1. In particular, we will consider the case of $c_l = 5.0 \forall l$ where source 1 generates a demand d_1 and the rest a demand d. The two graphs in Fig. 4 shows the optimum traffic distribution

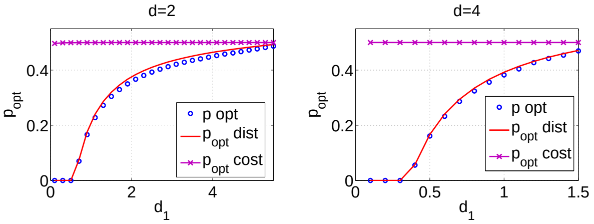


Fig. 4. Some case-scenarios for the network in Fig. 1

for $d = 2, 4$ as a function of d_1. The three curves represent the actual optimum, the value obtained by the distributed algorithm, and the one obtained by both MATE and TeXCP. We can see that while d_1 is relatively small and the *ABW* is enough, source 1 uses only the lower path. If any of these conditions is not true, p will rapidly go to 0.5, but always privileging better conditions on the upper path.

B. The Benefits of Utility Maximization

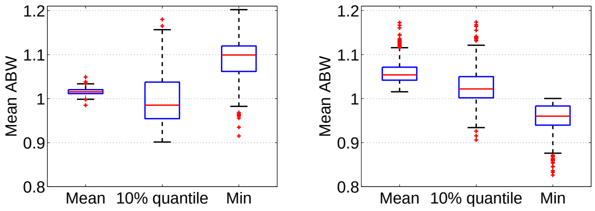
In this section we will assess the performance gain achieved by our proposal (UM from now on, as in Utility Maximization) over two well-known load-balancing techniques: MATE and TeXCP, which we already presented in Sec. II. It is important to highlight that the results shown in this section were obtained by the distributed algorithm. We tried some centralized numerical optimization methods and obtained very similar results, verifying that approximations we did had little effect.

The comparison will be made in two real networks, with several real TMs, calculating for each of these demands two performance indicators: mean *ABW* perceived by sources (u_s) and link utilization (ρ_l/c_l), whose importance we have already discussed. We could consider other performance indicators, such as queueing delay or path propagation delay. However, calculation of the former depends heavily on the assumed traffic model, and we shall suppose that the latter has already been taken into account by the operator in the choice of paths.

For each TM we measured a weighted mean u_s, where the corresponding weight is $d_s/\sum d_s$. This average provides us with a rough idea of the performance as perceived by traffic. A good value of this average indicator could however hide some pathological cases where some OD pairs obtain a bad performance. That is why we also measured the 10% quantile and the minimum u_s. The comparison will be made by dividing the value obtained by UM by the one obtained by the other load-balancing technique in each case.

For each TM we also calculated the mean, 90% quantile and maximum utilization on the network for each of the mechanisms. The difference between the TeXCP indicators and the other mechanisms is presented.

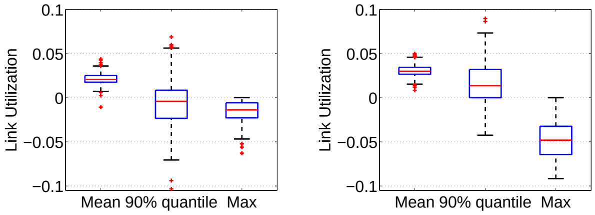
1) Comparison in Abilene: Our first case study is Abilene [24], a well-known academic network which consists of 12 nodes and 15 bidirectional links all with the same capacity. The topology comes as an example in the TOTEM toolbox [25] and we used 388 demands (spanning a complete week) of dataset X11 from [26]. The paths we used in this case



(a) UM vs MATE

(b) UM vs TeXCP

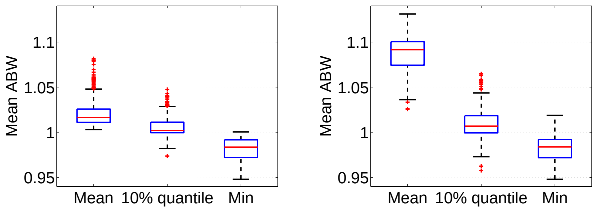
Fig. 5. u_s for UM, MATE and TeXCP in the Abilene network



(a) TeXCP vs UM

(b) TeXCP vs MATE

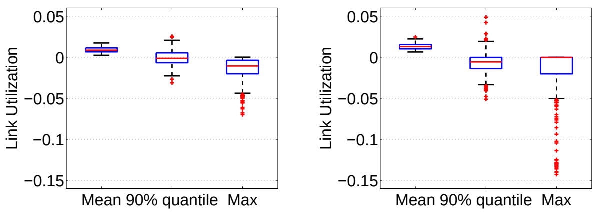
Fig. 6. Link utilization for TeXCP, UM and MATE in the Abilene network



(a) UM vs MATE

(b) UM vs TeXCP

Fig. 7. u_s for UM, MATE and TeXCP in the Géant network



(a) TeXCP vs UM

(b) TeXCP vs MATE

Fig. 8. Link utilization for TeXCP, UM and MATE in the Géant network

were constructed by hand, trying to give sources as much path diversity as possible, but limiting the hop count.

In Fig. 5 we can see the boxplots of the u_s indicators. We first note that the weighted mean u_s is always bigger in UM than in MATE, being generally between 1-2% and at most 4%. On the other hand, TeXCP obtains a much smaller mean u_s, generally between 4-7% and as much as 12% smaller. No conclusive results can be obtained from the quantile u_s. In the minimum u_s, UM achieves a minimum u_s that is generally between 6-12% (and can be as big as 20%) bigger than MATE. As expected, TeXCP obtains the best results in this aspect, although its gain over UM is not so large.

Fig. 6 shows the results on link utilization. Both the mean and the quantile do not present any substantial difference between the three mechanisms (except for a relatively bigger mean utilization for TeXCP). It is in the maximum utilization that we can see a clearer distinction between them, where as expected TeXCP always obtains the best results. However, and in concordance with the u_s indicators, its gain over UM is smaller than over MATE, the former being generally 1-2% and the latter between 3-7%.

2) Comparison in Géant: The second case scenario is Géant [27]. This European academic network connects 23 nodes using 74 unidirectional links, with capacities that range from 155 Mbps to 10 Gbps. The topology and TMs (477 in total, covering a three week period) were obtained from TOTEM's webpage [5], [25]. In this case paths were constructed by a shortest path algorithm, where we used the inverse of the capacity as the link's weight. For each OD pair we computed two paths. The first is simply the shortest path, we then prune the network of the links this path uses, and compute the second shortest path.

Results for the u_s in this case can be seen in Fig. 7. This time, results of both UM and MATE are more similar, where the mean and quantile u_s are somewhat bigger for UM than MATE (although in some cases the difference easily exceeds 5%), and the minimum is relatively bigger for MATE than UM. However, the results of the comparison between UM and TeXCP are clearly in favor of the former. The mean ABW_P is generally 7-10% bigger, going as high as 13%. With respect to the minimum ABW_P, the results are logically better for TeXCP, but the difference is not significant.

Fig. 8 shows that the results for the link utilization are also very similar between UM and MATE. The difference between the two and TeXCP is not very significant, specially in the mean and quantile. With respect to the maximum utilization, TeXCP obtains only subtle improvements over the rest. However, there are some cases where the difference with MATE is more than 10%.

VI. Implementation Issues

In this section we will discuss some practical issues of the distributed algorithm. Its first clear requirement is that border routers have to 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.

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. A counter for each of them should be kept by interior routers, indicating the number of routed bytes belonging to a given label. Periodically, each router calculates the corresponding d_{si} by dividing its counter by the measurement interval, after which they reset it. In order to avoid noisy measurements, some filtering should be applied. In our simulations a simple exponential filter was sufficient. The total load ρ_l is then calculated as the sum of all d_{si} that use the link. The *ABW* of link l can be easily calculated as the difference between the total capacity and this value. However, in order to avoid numerical problems, the maximum between the *ABW* and a relatively small value (for instance $c_l/100$) should be used.

Another important aspect is the communication between link and source. Explicit messages from the source to the link are not necessary, 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, but this approximation does not affect the algorithm's performance.

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 a probe packet, initially indicating as the path's *ABW* and cost ∞ and 0.0 respectively. As the probe advances towards the destination node, each interior router checks the *ABW* indicated in it. If this value is bigger than that of the outgoing link, the router overwrites it with the link's *ABW* (and "remembers" it). When the destination node receives the probe packet, he sends it back to the source through the same path but in the opposite direction. As it is going back, each interior router checks whether the final *ABW* indicated on the probe packet is the one its link had when the packet first passed. If this is the case, it means that he is the bottleneck of the particular path. He then calculates θ_{sil} accordingly, updates the link's total cost $\widehat{\theta_l}$, and adds this value to the total path cost indicated on the packet. Finally, the source receives the path's *ABW* and total cost, and updates his load distribution.

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 it is not very urgent to change the traffic distribution to the optimum. Research on this direction will be the object of future work.

As a final remark we will emphasize the importance of load measurement periods being smaller (several times smaller) than the inter-probe period. This way, the source can clearly appreciate the effects of the last load distribution update. If this is not the case, the distributed algorithm will either be too unstable or unresponsive.

A. Packet-Level Simulations

In order to verify the correct operation of the algorithm in a realistic context (i.e. delayed and unprecise measurements), we implemented it in an ns-2 [28] simulation script. The first example we will present is again the one of Fig. 1. This time, all links have a capacity of 1.0 Mbps, except for the "access" ones which have 2.0 Mbps, and all transmission delays are 20 ms. Traffic consists of elastic flows with an exponentially distributed size with mean 20 kB, arriving as a Poisson process of the corresponding intensity so as to generate a demand $d = 400$ kbps for all sources. The exponential filter's parameter is set to $\beta = 0.7$, and γ is set to 250×10^{-9} (although it may seem like a small value, we measured everything in bps). Probabilities are updated every 60 seconds and load measurements are made every 20 seconds. Fig. 9 shows the evolution of p (the upper path's portion of source 1 traffic) over time for several initial values. Two important aspects of the algorithm are highlighted in this example. First, the initial condition has virtually no effect on the stationary behavior of p. Secondly, the distributed algorithm converges to the optimum very fast (less than 15 iterations). Finally, the simulations indicate that, although they traverse a more congested link, flows on the lower path are transferred faster than those in the upper one. This can be explained by the bigger queueing delay in the upper path. The second router on the lower path has no queueing delay, because packet size is constant, and their interarrival time is shaped by the previous router. This means that, by preferring less shared links, the algorithm avoids the use of links with big queueing delays.

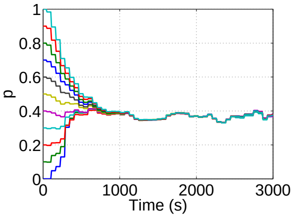


Fig. 9. Evolution of p over time in the example of Fig. 1

We will now present an example where two UM load-balancing algorithms interact. 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. Links as well as the traffic characteristics are the same as in the previous example. The initial probabilities are set on both sources so that the shared link is not used, maximizing the likelihood of oscillations. Probabilities are updated every 50 seconds ($\gamma = 5\times10^{-9}$) and the load measurements are made every 10 seconds. Sources are however not coordinated 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.

In Fig. 10 we see that both sources at first rapidly change their probability to start using the middle path. It takes them a little while to realize that another source is using this path, and start augmenting the direct path probability, but slower than before, since the price difference between them is not big now. The probabilities finally converge to the optimum after some few minutes. This whole process takes approximately 15 min. (only 20 iterations). Notice that load measurements need not be very precise, and that the algorithm supports some noise.

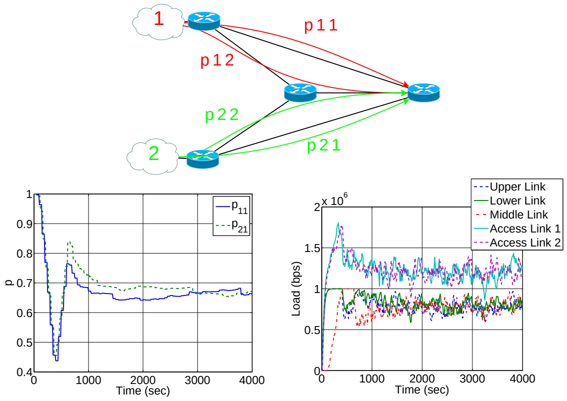


Fig. 10. The example topology, and the traffic distribution and links' load as a function of time

VII. Concluding Remarks

In this work we presented a load-balancing mechanism that takes into account the needs of both the network operator and the users. We achieved this by defining an objective function that is not a cost at the *link level*, but a utility at the *OD pair level*. This lead to an optimization formulation very similar in spirit to Multi-Path TCP [15], where we maximize the sum of a utility function whose argument is the average *ABW* each OD pair sees in its path. Although the resulting optimization problem was not convex, a distributed algorithm was outlined which finds very tight approximations to the optimum in a relatively short time.

Along with our proposal (noted as UM), we considered two previously proposed dynamic load-balancing mechanisms: MATE [1] and TeXCP [2]. From our study, conducted over two real networks along with several real traffic demands, some conclusions can be drawn. Firstly, performance as perceived by traffic (measured as the mean *ABW*) is always better in UM than both MATE and TeXCP. More specifically, the improvement over MATE is generally not very big for the mean value, but can be important, specially in the worst *ABW*. This difference comes from the implicit unfairness among OD pairs of the social cost function of MATE. With respect to TeXCP, UM obtains a significantly better performance, specially when the link capacities are not similar. Secondly, results on link utilization are very similar for UM and TeXCP. MATE obtains similar results in the mean and quantile link utilization. However, maximum link utilization can be significantly bigger in MATE than the other two mechanisms. All in all, UM is the most balanced mechanism, in the sense that it generally outperforms the rest (though in some cases the difference may not be large), and when it does not there is only a small difference.

Much remains to be done. For instance, we have considered only elastic traffic but the performance obtained by streaming traffic should be studied. Moreover, the stability of the algorithm should be analyzed, specially considering that several approximations were used.

References

[1] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," *INFOCOM 2001*, vol. 3, pp. 1300–1309, 2001.

[2] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: responsive yet stable traffic engineering," in *ACM SIGCOMM '05*, 2005, pp. 253–264.

[3] S. Fischer, N. Kammenhuber, and A. Feldmann, "Replex: dynamic traffic engineering based on wardrop routing policies," in *CoNEXT 2006*, pp. 1–12.

[4] R. Srikant, *The Mathematics of Internet Congestion Control*. Birkhäuser Boston, 2003.

[5] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 83–86, 2006.

[6] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe, "Resource management with hoses: point-to-cloud services for virtual private networks," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 679–692, 2002.

[7] C. Zhang, J. Kurose, D. Towsley, Z. Ge, and Y. Liu, "Optimal routing with multiple traffic matrices tradeoff between average and worst case performance," in *ICNP '05*, November 2005.

[8] W. Ben-Ameur and H. Kerivin, "Routing of uncertain traffic demands," *Optimization and Engineering*, vol. 6, no. 3, pp. 283–313, september 2005.

[9] I. Juva, "Robust load balancing," *IEEE GLOBECOM '07*, pp. 2708–2713, 26-30 Nov. 2007.

[10] P. Casas, L. Fillatre, and S. Vaton, "Robust and Reactive Traffic Engineering for Dynamic Traffic Demands," *NGI 2008*, April 2008.

[11] W.-H. Wang, M. Palaniswami, and S. H. Low, "Optimal flow control and routing in multi-path networks," *Perform. Eval.*, vol. 52, no. 2-3, pp. 119–132, 2003.

[12] F. Kelly and T. Voice, "Stability of end-to-end algorithms for joint routing and rate control," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 5–12, 2005.

[13] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley, "Multi-path tcp: a joint congestion control and routing scheme to exploit path diversity in the internet," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1260–1271, 2006.

[14] F. Paganini, "Congestion control with adaptive multipath routing based on optimization," in *40th Annual Conference on Information Sciences and Systems*, March 2006.

[15] P. Key, L. Massoulie, and D. Towsley, "Path selection and multipath congestion control," in *IEEE INFOCOM 2007*, May 2007, pp. 143–151.

[16] J. He, M. Bresler, M. Chiang, and J. Rexford, "Towards robust multi-layer traffic engineering: Optimization of congestion control and routing," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 5, pp. 868–880, June 2007.

[17] T. Bonald and L. Massoulié, "Impact of fairness on internet performance," in *ACM SIGMETRICS '01*, 2001, pp. 82–91.

[18] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Trans. Netw.*, vol. 8, no. 5, pp. 556–567, 2000.

[19] F. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability," in *Journal of the Operational Research Society*, vol. 49, 1998.

[20] T. Bonald, L. Massoulié, A. Proutière, and J. Virtamo, "A queueing analysis of max-min fairness, proportional fairness and balanced fairness," *Queueing Syst. Theory Appl.*, vol. 53, no. 1-2, pp. 65–84, 2006.

[21] T. Bonald and A. Proutière, "On performance bounds for balanced fairness," *Perform. Eval.*, vol. 55, no. 1-2, pp. 25–50, 2004.

[22] M. Pappalardo, "On the duality gap in nonconvex optimization," *Math. Oper. Res.*, vol. 11, no. 1, 1986.

[23] M. Minoux, *Programmation Mathématique: théorie et algorithmes*. Dunod, 1983.

[24] "The Abilene Network," http://www.internet2.edu/network/.

[25] "TOTEM: TOolbox for Traffic Engineering Methods," http://totem.info.ucl.ac.be/.

[26] Yin Zhang, "Abilene Dataset," http://www.cs.utexas.edu/∼yzhang/research/AbileneTM/.

[27] "Géant Topology Map," http://www.geant.net.

[28] "The Network Simulator - ns," http://nsnam.isi.edu/nsnam/index.php/Main_Page.