Short-Sighted Routing or When Less is More

José Legatheaux Martins and Nelson Campos

NOVA LINCS, and Department of Informatics, Faculty of Sciences and Technology,
NOVA Lisbon University, 2829–516 Caparica, Portugal
Email: jose.legatheaux@fct.unl.pt, nac19324@campus.fct.unl.pt

***Abstract*—Due to the high costs of wide area links, traffic engineering is frequently used to optimize wide area networks performance. In recent years, architectures based on Software Defined Networking (SDN) approaches are being proposed as better ways of implementing network control and management. However, network optimization requires timely acquisition of statistics on network traffic demands by SDN controllers, as well as timely reconfiguration of network paths. Both goals may only be achieved with a scalable control plane.**

In this paper, we present an approach which is meant to trade complexity and scalability of the control plane for routing optimality. Our experiments suggest that the performance of provider networks is nearly optimal, even when load balancing parameters have been computed using a slightly outdated vision of traffic demands. We named this approach *Short-Sighted Routing* (SSR). Our results also show that performance optimality in many provider networks, that use load balancing over several paths, is not affected in a significant way by relaxing the timeliness of network traffic demands acquisition and the corresponding load balancing reconfiguration.

This allowed us to conclude that a dynamic centralized network optimization and management, in accordance with a SDN approach, is also quite realistic in service provider networks.

***Keywords*—Routing, Traffic Engineering, Network Optimization, SDN, Unpredictable traffic, Service Provider Networks**

I. Introduction

Due to the high costs associated with wide area links, traffic engineering is frequently used to optimize wide area networks performance. Network performance maximization may be treated as a multi-commodity flow optimization problem, which is NP-complete for integer flows, but may be solved by linear programming when such flows are divided as needed [1]. In both cases, the input of the problem is the network definition and a set of demands, *i.e.*, a traffic matrix.

Multi-path routing can be optimized by splitting arbitrary fractions of each demand over several paths, which may be implemented as tunnels. In this context, the significant number of required tunnels may be seen as a drawback since it increases network management complexity.

Since providers require the usage of tunnels for many reasons other than optimization, like providing virtual private networks (VPNs) and virtual links, or isolating different classes of services or customers, most multi-path routing solutions adopted in service provider networks are based on tunnels. This technique is known as explicit routing [2, 3, 4].

To be more effective, network optimization requires demands to be dynamically balanced across different paths, in a continuous on-line adaptation process. However, in practical terms, demands are overestimated and many links exhibit an important fraction of spare capacity for security reasons [3]. In fact, demands estimations are hard to obtain, they may quickly vary in short periods [5], and faults may modify the network configuration [6].

Additionally, traditional network routing architectures lie on a distributed set of semi-autonomous routing and forwarding nodes (*e.g.*, switches, routers), and make use of distributed algorithms for coordination. For example, MPLS-TE [2] solutions with the auto-bandwidth feature, periodically estimate trunk[1] load, and may execute a Constrained-Based Shortest-Path First Routing algorithm [7] to recompute the paths used. The resulting system is complex and requires expensive devices, but it also introduces some instability and latency inflation of the lower priority trunks during the distributed trunk rearrangement phases [7].

The introduction of simpler logically centralized solutions for managing networks is being pursued under the umbrella of the SDN movement [8]. For example, [9, 10] describe experiences focused on network optimization in data center environments, flow by flow, using an SDN approach. However, with service provider networks, in addition to higher latencies between switching nodes and the controller, there is an unpredictable and limitless number of heterogeneous packet flows. Due to scalability issues, their aggregation is mandatory. Packet flow aggregates load may vary in time, requiring the gathering of statistics and dynamic updates of paths and load balancing weights in order to optimize the network.

Several SDN-based network optimization architectures are also being used [3], or proposed [4], with the aim of controlling data center interconnection networks. These are a kind of private networks[2] where the operator may shape network demands, by means of traffic shapers and application schedulers (*i.e.*, to distribute in time backups and other voluminous data movements between different data centers). This enables a smoother transition between sets of demands than what would be expectable in the general case. Therefore, these solutions may not be suited for general purpose provider networks, in which user demands are harder to schedule and should not be shaped.

In a general purpose service provider network using a SDN control plane, these limitations can be mitigated by making

[1] In the context of MPLS, a trunk is equivalent to a Forwarding Equivalence Class. It is a subset of packet flows with the same ingress and egress nodes that are forwarded using the same Label Switching Path.

[2] Inter-data center customers traffic crosses the internet.

use of spare capacity or a very tight dynamic load distribution rearrangement. The former is a simple but expensive solution. The latter is hard to implement and to scale since it requires timely acquisition of demands statistics by the controller, and a smooth rearrangement of load distribution across paths. Scalability of this type of architecture is still an open issue [11].

However, if whenever the network experiences significant traffic demands change, the impact of delaying load distribution rearrangements had a low impact on the network performance, we could be closer to a solution to the above challenge. Such process could allow a less stringent demand estimation as well as wider intervals between load rearrangement phases. We call this approach *Short-Sighted Routing*, (SSR), since it is based on a blurred vision of the network state. It is a solution that stands between a fully periodic off-line and a fully on-line dynamic optimization.

As the number of required paths to spread the load may be very large and our goal is to increase scalability as much as possible, in order to speed up the computations and ease the deployment in the network, SSR load balancing may be restricted to a set of predefined and a priori computed set of ingress to egress nodes paths of order $O(n^2 \times k)$, where k is a small constant (*e.g.*, < 6). This simplification is popular [3, 4] and has been shown to imply a small sacrifice in terms of optimality [12]. The path sets are computed a priori with the goal of maximizing link usage diversity without significant end-to-end path latency increase [12]. Moreover, for the same scalability issues, load balancing rearrangement is only performed by edge switches, where the tunnels start [2, 3, 4].

This article presents a study that shows that a control plane along the lines introduced above, is realistic in several operators backbones, even when network demands are quite far from those that were used to compute optimal load distributions. The results show that a centralized dynamic approach to maximize network performance can be quite scalable. They also show that the use of a closed-loop control strategy to trigger the tuning of load balancing weights is realistic. In fact, the controller may estimate the intermediate levels of link load aggravation, and solely trigger the reconfigurations, if the expected link usage increase exceeds a safeguard level.

In the following section we present the network architecture, explaining where SSR can be used in conjunction with an SDN-based approach. Section III is devoted to the model and type of demand deviations that have been used to evaluate the proposal. In section IV the results of the tests performed are presented. A related work analysis is the content of section V. The article ends with our conclusions and perspectives for future work.

II. Network Architecture for SSR

SSR can be used in a SDN setting where a central controller determines the way a set of switches forwards packets, see Figure 1. Network switches are divided among a set of *n edge switches* and a set of *core switches*. The set of core switches may be empty. This happens when all switches receive and deliver traffic to customers or other networks. A traffic matrix is a $n \times n$ matrix where each pair (x, y) represents all traffic that enters the network at ingress node x and leaves the network at egress node y. In what follows, this traffic will be called a flow or a demand. The role of each component of the architecture is explained below.

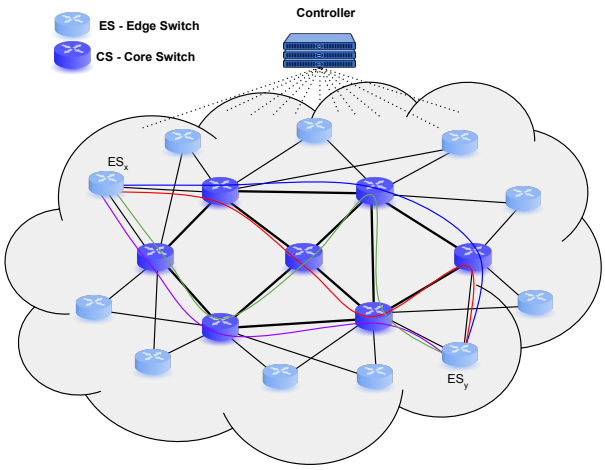

Figure 1: Network architecture including the controller, core switches, edge switches and an example of four tunnels among edge switches ES_x and ES_y

Core switches statically implement k paths between each pair of edge switches via tunnels (IP over IP, MPLS, VLANs, ...). These paths are computed a priori and independently of network load. Ways of dealing with transient faults are discussed in section V.

Edge switches receive ingress traffic, classify it in flows, and load balances them over the available tunnels to the corresponding egress switches. The load balancing parameters are computed by the controller by making use of aggregated statistics of each line of the traffic matrix, *i.e.*, all entries of the form $(x, -)$, collected by each edge switch. Edge switches also forward the egress traffic of their tunnels to providers, peers and customers.

The **controller** receives aggregate statistics computed by edge switches, periodically executes an optimization program and, when needed, informs the ingress switches about the new load balancing splitting weights. These reconfigurations can be strictly periodic, or be triggered only when special conditions occur (*e.g.*, the expected load variations are above network dependent thresholds). In a large network, local controllers, coordinated by a central one, may be in charge of aggregating traffic statistics and closely controlling nearby switches [3].

This architecture and SSR will be suitable if the network optimization is maximized and no flow aggregate is penalized behind an acceptable threshold (even if the controller at times, and not continuously, adapts load distribution). For each reconfiguration, the traffic matrix collected at that time is used for the optimization computation. By hypothesis, this traffic matrix is almost exact. Thus, the controller is able to compute an optimal load distribution that maximizes network performance for this traffic matrix. After each reconfiguration, the traffic dynamically changes but load distribution is still based on

the previous traffic matrix. Therefore, network optimization may decay, since load distribution parameters are still those previously computed. SSR suitability depends on the level of this degradation: if important demands deviations imply small degradations, SSR may be adopted as a scalable solution. A more rigorous formulation of both the problem and answer to these questions are presented in the following sections.

In this study we adopt some simplifications that are frequently used in network optimization studies (*e.g.*, see [5, 6]). It is difficult to implement an exact match of the computed load splitting weights, with real load distributions, in real switches, since different packet sizes and flows durations may prevent an exact load splitting [9]. In provider networks with a very large number of packet flows, the impact of this simplification is less significant. We also consider that all statistics used at the beginning of each period were collected instantaneously. This is unlikely to happen with most networks, specially wide-span networks with a large number of switches. However, these latencies can be ignored when compared with the intervals of each traffic load splitting rearrangement. Finally, we admit that it is possible to transit smoothly from one load distribution to another without any major negative effects on the current flows. This transition can itself be planned as an optimization problem [4].

III. Experiments Methodology

To study SSR resilience, we have used graphs of 7 networks: 3 publicly available (B4, Geánt and NTT), and 4 other inferred by project RocketFuel [13] (Above, Att, Sprint and Tiscali), see Table I. In each network, POPs (Point Of Presence cities) are mapped to nodes, and links are mapped to edges. All POPs were considered to correspond to edge nodes. All networks have been stripped of nodes of degree 1 since these would not contribute to the goals of our study.

Network	# Nodes $\|V\|$	# Links $\|E\|$	Characterization
Abovenet	15	30	USA backbone
ATT	35	68	USA backbone
B4 (Google)	12	19	Worldwide Inter-data center network
Géant	32	49	European backbone
NTT	27	63	Worldwide backbone
Sprint	32	64	USA backbone
Tiscali	30	76	European backbone

Table I: Characterization of the networks used in tests

We generated synthetic traffic matrices using an adaptation of the gravity model [14]. Computed demands are proportional to POPs populations. To this end, nodes were classified into three different categories. For the generation of an initial reference matrix (see below) the parameters were: the network graph, the POPs classifications and the matrix total demands sum. As in most networks, we don't have access to the real capacities of links to generate traffic matrices. Thus, during the tests, we used a fixed value for all links. As we shall soon discuss, given how the experiments were conducted, this simplification didn't affect the results significance or invalidated our conclusions. To get the desired number of matrices and to simulate the dynamic evolution of the traffic, we made use of distortion functions of the reference matrices that seek to recreate some known traffic variation patterns. These were:

Random distortion - It aims to recreate a random evolution of a traffic matrix. It has as parameter the distortion interval or depth. The higher this value, the more distant the distorted matrix is from the original one. Each matrix point is randomly distorted using a linear distribution of the distortions over the interval, that is, the depth of the distortion. Total traffic is not changed.

DDoS attack - This distortion aims to represent the occurrence of a DDoS attack. A node is chosen as the victim of the attack and the intensity or depth of the attack is the second parameter. The intensity of the attack is the factor by which all the elements of the column of the traffic matrix corresponding to the attacked victim are multiplied.

***Flash-crowd* event -** It recreates the occurrence of an event that leads to an accessible content from a particular POP to be highly demanded (*e.g.*, news website on a tragedy, stream of a football game, *etc* ...). The function is similar to the DDoS attack, with the exception that, instead of multiplying a chosen column by a factor, it multiplies a line by such factor.

BGP policy change - Some of the traffic that crosses the network does not have its origin or destination in direct network customers and has to be forwarded to other ASs (Autonomous Systems). This distortion function aims to recreate a situation where a proportion of the traffic leaving the network through node x is redirected to node y, due to a change in BGP policy. Both nodes and the fraction of traffic - the depth of the distortion - are used as parameters.

For each network, a set of k paths between each ordered pair of nodes was computed, using an algorithm that chooses, among all available paths, the ones that maximize edge diversity. This search only considers as eligible paths those that do not increase latency and length over those of the shortest paths. We used $k = 4$ since, with greater values, no significant modifications of preliminary results were detected.

To determine the load distribution of a traffic matrix over these paths that maximizes network optimization, we used an algorithm, which we named Optimal MultiPath (OMP), rather than a generic solver. Motivation came from the fact that it is possible to obtain an approximate good result in an efficient way. The algorithm is explained in the sequel.

Let $G = (V,E)$ be a simple graph (without self-loops or parallel edges), directed, connected and weighted, being the weight of each edge strictly positive. G models a network topology, where V is the set of all nodes, E is the set of all edges, and $n = \#V$ is the total number of nodes in the graph. The weight of each edge $e \in E$, $c(e)$, represents the link capacity. To obtain the directed graph from network descriptions we considered all links as full-duplex and mapped them in two simplex links. This property ensures that for each edge there is a corresponding symmetrical one.

A traffic matrix D is an $n \times n$ matrix, where each element D_{xy} corresponds to an ordered pair of origin-destination nodes (OD pair) that represents the demand between the origin node x and destination node y. For each OD pair, there exists a

set of k possible paths as referred above. P is the set of all available paths, and P_{xy} is the set of all available paths for the OD pair (x,y). All paths are simple, *i.e.*, all nodes of a path are different (there are no cycles). The demand D_{xy} is divided along the paths P_{xy}. Additionally, in order to describe the algorithm, we define Link Utilization (LU) as the fraction of the capacity of a link used to route traffic. Given a traffic matrix and a routing in a network, its maximum is designated Maximum Link Utilization (MLU) as usual [5, 15]. The goal of the algorithm is to minimize MLU by minimizing the LU of all links of the network.

Given a networks and a traffic matrix, the algorithm intends to find a routing that minimizes MLU. Informally, it divides the traffic matrix D into a number of identical matrices d_i, each representing an identical fraction of the overall network traffic, and whose sum is equal to the original matrix ($D = \sum d_i$). Then, the algorithm iteratively determines the single path for the demand of each OD pair $d_i(x,y)$ that minimizes the LU of the links of the paths P_{xy}, and adds $d_i(x,y)$ to the current load of all the edges of the selected path. Finally, the algorithm computes the splitting fractions of D_{xy} over paths P_{xy} that approximately minimize the LU of the different links, and computes their maximum. The result depends on the order in which the different elements of each matrix d_i are processed. However, if the number of iterations is appropriate, the fraction of traffic processed at each iteration is as small as required, and thus the order is indifferent. The end result will be approximately optimal, *i.e.*, the lowest MLU possible.

The final result of the algorithm is a routing solution defined by a set of values $f = \{f_{xy}(p) \mid x,y \in V,\ p \in P_{xy}\}$, where $f_{xy}(p)$ specifies the fraction of the demand from x to y that is routed over the path p. Let $p_{xy}(i,j) = \{p \in P_{xy} \mid (i,j) \in p\}$ be the set of all paths from x to y that cross the link (i,j). The total demand that is routed over the link (i,j) is given by:

$$l(i,j) = \sum_{x,y} \sum_{p \in p_{xy}(i,j)} D_{xy} \times f_{xy}(p). \quad (1)$$

The MLU of a routing f on a traffic matrix D is defined as the maximum LU of the network links induced by that routing:

$$MLU(f,D) = \max_{(i,j) \in E} \frac{l(i,j)}{c(i,j)}. \quad (2)$$

The algorithm OMP, see Algorithm 1, iteratively routes a demand fraction (an element of each elementary matrix d_i) by the available path with higher residual capacity, *i.e.*, the lowest LU. Using small demand fractions, no link is overloaded, and the end result is a more balanced distribution. With infinitely small fractions, the distribution converges to a solution that minimizes the MLU.

In a given network, the result of the OMP execution on a traffic matrix D is a distribution of demand ratios which approximately maximizes network utilization. Let us call this distribution the routing $f = OMP(D)$. To each f and D is associated a MLU, denoted by $MLU_f(D)$.

We conducted several tests using the networks of Table I. For each one, the MLU obtained with 20 different random

Input: a graph $G = (V,E)$, a set of paths P, and a traffic matrix D
Output: a set of values $f = \{f_{xy}(p) \mid x,y \in V,\ p \in P_{xy}\}$
begin
  Define: $p.available() = \min_{e \in p} (1 - \frac{e.assigned}{e.capacity})$
  Initialize: $\forall\ e \in E,\ e.assigned = 0$
  Initialize: $\forall\ e = (x,y) \in E,\ x,y \in V,\ e.capacity = c(e)$
  Initialize: $\forall\ (x,y \in V,\ p \in P_{xy}),\ f_{xy}(p) = 0$
  for *i = 1* **to** *iterations* **do**
    foreach (x,y) **do**
      $p = \arg\max_{p \in P_{xy}}\ p.available()$;
      foreach $e \in p$ **do**
        $e.assigned = e.assigned + \frac{D_{xy}}{iterations}$
      end
      $f_{xy}(p) = f_{xy}(p) + \frac{1}{iterations}$
    end
  end
end

Algorithm 1: *Optimal Multipath*

matrices was computed with the number of iterations ranging from 1, 2, ... up to 100. In all, but the network Géant, the computed MLUs after 15 iterations and 100 iterations differed by less than 1%. The same difference was observed in the Géant network after 35 iterations. As a precaution, all presented results in section IV were computed with $iterations = 100$.

The complexity class of the algorithm is $O(iterations \times n^2 \times k \times l)$, where *iterations* is the number of iterations, n is the number of nodes, k is the number of different paths used to connect each OD pair, and l is the average path length. As already stated, $iterations = 100$ and $k = 4$. The number of nodes depends on the network. Finally, in the tested networks, the highest average path length was 5.09 with Géant.

The algorithm was implemented in Java and executed in an Intel 2 core CPU T5750@2.00GHz processor, with 3 G bytes of RAM and running Linux Ubuntu 12.04. With the above parameters, it took 42 milliseconds to compute paths for a network with 12 nodes (B4), and 1906 milliseconds for a network with 35 nodes (AT&T).

SSR uses OMP to compute $f = OMP(D)$ from a matrix D, and controls the network to distribute demands in accordance to f. Then, until a new matrix is used to update the routing, any different traffic matrix D' continues to be routed according to the distribution f, with the corresponding $MLU_f(D')$.

To evaluate the SSR quality, the test procedure is similar between the several trials. A reference traffic matrix D is fixed and a routing $f = OMP(D)$ is computed using the gravity model. Then, several distorted traffic matrices D' are generated using a distortion function, and, for each one, $f' = OMP(D')$, $MLU_{SSR} = MLU_f(D')$, and $MLU_{OPT} = MLU_{f'}(D')$ are computed. Having MLU_{SSR} and MLU_{OPT}, we can calculate the ratio $\frac{MLU_{SSR}}{MLU_{OPT}} - 1$ for each distorted traffic matrix. Its maximum indicates how far MLU_{SSR} is from MLU_{OMP}.

As the comparison of the link loads is performed using linear functions and the result of the tests is a ratio, the actual capacity of the links is not relevant as long as we consider that all links have the same capacity. Ultimately, a MLU can

even be greater than 1. Also, it is easy to recognize that there is a linear relation among the total traffic of a matrix and the computed MLU. Thus, the distortions characterized by multiplying a traffic matrix by a factor do not need to be tested.

IV. Results

The performance of SSR and its ability to absorb traffic variations was extensively evaluated for the selected networks. In each experiment, we made several trials with different traffic matrices. For trials with random distortion, we made use of 100 different traffic matrices. For trials of types DDoS attack and flash crowd events, the tests performed used each of the network nodes as victims. In the case of BGP policy change, as we must choose two nodes in each trial, covering all possibilities would imply a very high number of tests. As a consequence, we chose 10 POPs that have the largest population, and covered all combinations of these 10 nodes (90 trials). For each type of distortion, we did trials with three values of the parameter depth of distortion: 25%, 50%, 75%.

In the remainder of this section all results are presented using figures exhibiting plots of the cumulative distribution functions, and representing the probability that a trial resulted in a loss of quality of x (abscissa axis) or less, relative to the optimal result. Due to lack of space, we only present the worst results for each network.

For networks Sprint, Att, Above and Tiscali, the worst results were the random distortion tests. Sprint, presented in Figure 2, and Tiscali, presented in Figure 3, represent, respectively, the best and the worst networks of this group.

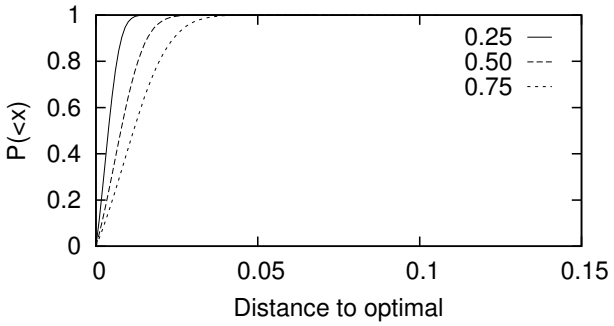

Figure 2: Random distortion results for network Sprint

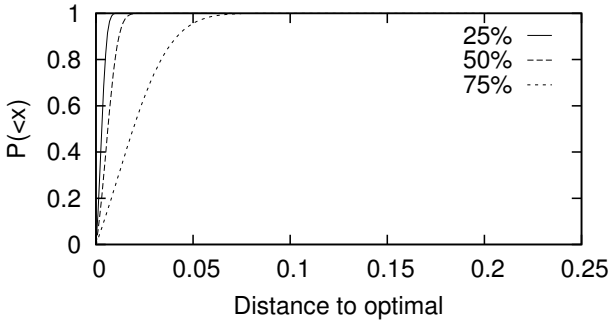

Figure 3: Random distortion results for network Tiscali

Networks Sprint, Att, Above and Tiscali all performed very well and exhibited an MLU below 2% far from the optimal with a random distortion of depth 25%, and below 8% with a random distortion of depth 75%. The other types of distortions were better absorbed by these networks. The main conclusion is that SSR is very resilient and absorbs almost all distortions in these networks.

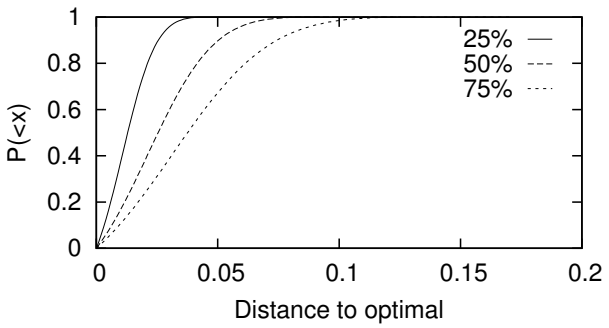

Figure 4: Random distortion results for network B4

Network B4 shows an MLU below 5% far from the optimal with a random distortion of depth 25%, and below 12% with a random distortion of depth 75%. It absorbed in a similar way the DDoS attack distortion, and tolerated more smoothly the remaining types of distortions.

With networks NTT and Géant the worst performant tests were the Flash-crowd events, see figures 5 and 6.

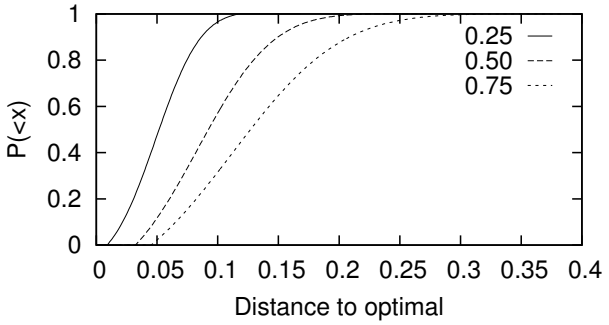

Figure 5: Flash-crowd event distortion results for network Géant

In the NTT and GÉANT networks, for the flash-crowd events, the tests with distortions of depth 25% increased MLU_{SSR} at most 12%. However, with higher distortions, the results were slightly worse. A new cycle of flow redistribution weights computation and reconfiguration in edge switches should be triggered before distortions of such depth occur. In fact, the controller can use intermediate traffic samples to quickly evaluate (*e.g.*, by making use of a smaller number of iterations) the MLU_{SSR}, and dynamically trigger the adjustment of traffic splitting parameters in switches depending on the computed MLU.

These results show that there is room to introduce a closed-loop control scheme, performing relaxed network reconfigu-

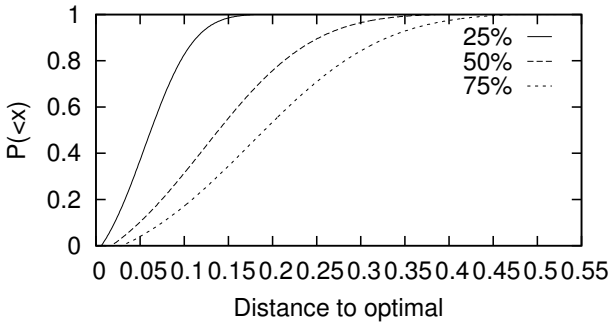

Figure 6: Flash-crowd event distortion results for network NTT

rations, since multi-path load distribution absorbs quite well traffic variations.

We also compared the performance of the SSR with hop-by-hop shortest path routing. As expected, in all tests, the behavior of routing using only the shortest path leads to a much higher MLU than that obtained using SSR. For example, with network Sprint, the network which performs better with SSR, with shortest path routing and a random distortion of depth 75%, all matrix demands increased MLU above 80%, and some close to 100%. This compares badly with SSR that, in the same conditions, only increased MLU below 4%.

V. Related Work

On-line traffic engineering has been intensively researched during this century. An extensive overview of such research can be found in [1]. However, most operational networks performing on-line adaptive traffic-engineering, make use of an MPLS-based solution [2] complemented with the auto-bandwidth feature [7]. This approach requires complex and expensive equipments, that execute distributed algorithms, are hard to manage and less flexible than desirable [3]. This state of affairs is being challenged by the SDN approach.

SDN-based approaches of on-line traffic engineering in data center networks are quite common (*e.g.*, [9, 10]) and show promise. In this environment, there is a closer integration between the network management systems and hosts using it. Also, traffic demands seem more predictable, at least in what concerns the worst case. Finally, controller(s) and switches are close to each another.

Papers [3, 4] present SDN-based architectures, that rely on centralized algorithms to successfully dynamically optimize the performance of private inter data center networks. These algorithms and approaches cannot be adopted as is in general-purpose provider networks, where shapers and application scheduling cannot be used.

Service provider networks always span a large area, customers are less constrained, and traffic variations may be quite significant in short periods. These characteristics bring together different scalability issues, whose discussion is quite frequent [11]. In this work we have researched how scalability can be traded for control plane precision. The final result seems optimistic, since, among several networks, precision can be sacrificed for scalability, without having a significant impact on optimality.

The difficulties of network precise demands acquisition and dynamic network optimization, have promoted the so called hose model and the oblivious or two phase routing [5, 16] approaches. These are off-line traffic engineering methods, that make use of static load-balancing across several paths, for sets of demands whose routing is feasible. They sacrifice some degree of optimality [5], as does SSR. In general purpose networks such solutions require a number of paths of order $O(n^3)$, since they are inspired by Valiant Load Balancing [17]. Moreover, they expand latency, and do not necessarily perform better than SSR in provider networks [5]. They seem better suited for data center environments where ECMP (Equal Cost Multi-Path) routing may be used.

One of the main challenges of tunnel-based traffic engineering is related with network faults. When faults arise, some tunnels become unavailable, and the traffic they carried must be redistributed among alternative operational tunnels. When a fault model based on previous statistics is known, the best sets of paths and load distribution weights can be computed a priori for each fault scenario, and statically programmed in the edge switches to be used when faults occur, see [6]. As the authors point, another alternative is to redistribute demands, using the same proportions, through the still operational paths. In a recent paper [18], another technique is proposed for a private network. This technique consists of computing a traffic distribution, that also trades some optimality (by reserving some spare network capacity) for its ability to still route the same demands, without congestion, even when up to k faults occur. The method represents a new way of computing the splitting of demands. The authors also show that the method may be used in service provider networks that do not use customer traffic shaping. The above methods could be adapted to be integrated in SSR.

VI. Conclusions and Future Work

In this article, we presented a network performance enhancement strategy, dubbed SSR, leveraging the SDN philosophy to support on-line dynamic traffic engineering for provider networks. The most interesting conclusion of this study is that multi-path routing over a set of a priori computed set of paths is able to absorb a very important fraction of demands variation without load distribution rearrangement.

In fact, and surprisingly, this study shows that in many service provider networks, SSR behaves very well. Indeed, in several networks, distortions of depth 75% cause a MLU increase of less than 2.5% far from the optimal. In the worst performing tested networks, deviations of depth 25% brought always a MLU increase below 12%.

The tests performed show that the MLU increase vis-à-vis the optimal, is low enough to allow the enlargement of the period used to update the load balancing weights of demands. In fact, even with traffic matrices far from those used to compute optimal weights, MLU is a small percentage above optimal.

These results show that a SDN approach to maximize network performance can be quite scalable. They also show that the use of a closed-loop control strategy to trigger the tuning of load balancing weights seems realistic. Looking

back, making use of more complex solutions is probably less effective, in real networks, than simpler ones.

Future work on SSR includes the development of strategies to deal with faults, as well as the development of a real testbed to monitor its performance in an environment closer to a production one.

VII. Acknowledgements

Nelson Campos was partially supported by NOVA LINCS on grant FCT - NOVA LINCS UID/CEC/04516/2013. We are grateful to Margarida Mamede and João Horta for their valuable contribution for the development of the algorithms used to compute network paths and to Inês Legatheaux for her help in revising the manuscript.

References

[1] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys and Tutorials*, vol. 10, no. 1, pp. 36–56, 2008.

[2] X. Xiao, A. Hannan, B. Bailey, and L. Ni, "Traffic engineering with mpls in the internet," *Network, IEEE*, vol. 14, no. 2, pp. 28–33, Mar 2000.

[3] S. Jain and et al., "B4: Experience with a globally-deployed software defined wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Aug. 2013.

[4] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *ACM SIGCOMM 2013*. New York, NY, USA: ACM, 2013, pp. 15–26.

[5] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, "Cope: Traffic engineering in dynamic networks," in *SIGCOMM'2006*. New York, NY, USA: ACM, 2006, pp. 99–110.

[6] M. Suchara, D. Xu, R. Doverspike, D. Johnson, and J. Rexford, "Network architecture for joint failure recovery and traffic engineering," *SIGMETRICS Performance Evaluation Review-Measurement and Evaluation*, vol. 39, no. 1, p. 97, 2011.

[7] A. Pathak, M. Zhang, Y. C. Hu, R. Mahajan, and D. Maltz, "Latency inflation with mpls-based traffic engineering," in *Proceedings of ACM IMC'2011*. New York, NY, USA: ACM, 2011, pp. 463–472.

[8] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, attributes, and use cases: A compass for sdn," *Communications Magazine, IEEE*, vol. 52, no. 6, pp. 210–217, June 2014.

[9] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of USENIX NSDI'2010*. Berkeley, CA, USA: USENIX Association, 2010, pp. 19–19.

[10] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[11] S. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 136–141, February 2013.

[12] O. M. Heckmann, *The Competitive Internet Service Provider*, 1st ed., ser. Wiley Series in Communications Networking & Distributed Systems. Chichester, UK: Wiley-Interscience, April 2006.

[13] N. Spring, R. Mahajan, and T. Anderson, "Quantifying the causes of path inflation," in *Proceedings of SIGCOMM'2003*. ACM, 2003.

[14] M. R. Paul Tune, "Traffic matrices: A primer," *in H. Haddadi, O. Bonaventure (Eds.), Recent Advances in Networking, ACM SIGCOMM eBook*, pp. 108–163, 2013.

[15] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," *Networking, IEEE/ACM Transactions on*, vol. 19, no. 6, pp. 1717–1730, Dec 2011.

[16] M. Kodialam, T. V. Lakshman, and S. Sengupta, "Traffic-oblivious routing in the hose model," *IEEE/ACM Trans. Netw.*, vol. 19, no. 3, pp. 774–787, jun 2011.

[17] H. Liu and R. Zhang-Shen, "On direct routing in the valiant load-balancing architecture," in *GLOBECOM '2005. IEEE*, vol. 2, Dec. 2005, pp. 6 pp.–726.

[18] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *Proceedings of ACM SIGCOMM'2014*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 527–538.