Online OSPF weights optimization in IP networks

Josselin Vallet, Olivier Brun

Online OSPF weights optimization in IP networks

Josselin Vallet[a,b,*], Olivier Brun[a,b]

[a] *CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France*
[b] *Univ. de Toulouse, LAAS, F-31400 Toulouse, France*

Abstract

The high volatility of traffic patterns in IP networks calls for dynamic routing schemes allowing to adapt resource utilization to prevailing traffic. In this paper, we focus on the problem of link weight optimization in OSPF networks where the traffic is routed along shortest paths according to the link metrics. We propose an online approach to optimize OSPF weights, and thus the routing paths, adaptively as some changes are observed in the traffic. The approach relies on the estimation of traffic demands using SNMP link counts. Experimental results on both simulated and real traffic data show that the network congestion rate can be significantly reduced with respect to a static weight configuration.

Keywords: OSPF weights, SNMP link counts, routing optimization, traffic matrix estimation

1. Introduction

With the increasing popularity of bandwidth-hungry applications, traffic patterns are getting more and more volatile. A consequence of the high traffic variability is that it is no more credible to use a single "busy-hour" traffic matrix for traffic engineering. Indeed, such an approach can lead to poor network performances if at some point in time the actual traffic matrix deviates significantly from the one used for traffic engineering. A well-known alternative approach to handle time-varying traffic matrices is to rely on online traffic monitoring and to adapt resource utilization when changes are observed. One of the fundamental mechanisms to control the performances of a network is route optimization. It allows to make a more efficient use of network resources by tailoring routes to prevailing traffic. However, several difficulties arise when seeking to design and implement an adaptive routing scheme in IP networks.

The first difficulty is related to how traffic is routed by intra-domain routing protocols, the most prominent being Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS) [1, 2]. Each traffic flow is routed along shortest paths, splitting the flow equally at nodes where several outgoing links are on shortest paths to the destination. Although they are usually set to one, the weights of the links, and thereby the shortest path routes, can be changed by the network operator. Given a set of traffic demands between origin/destination (OD) pairs, the link weight optimization problem amounts to finding a set of link weights that optimize a given performance measure, e.g., the maximum utilization rate of the links (see [3, 4, 5, 6] and references therein).

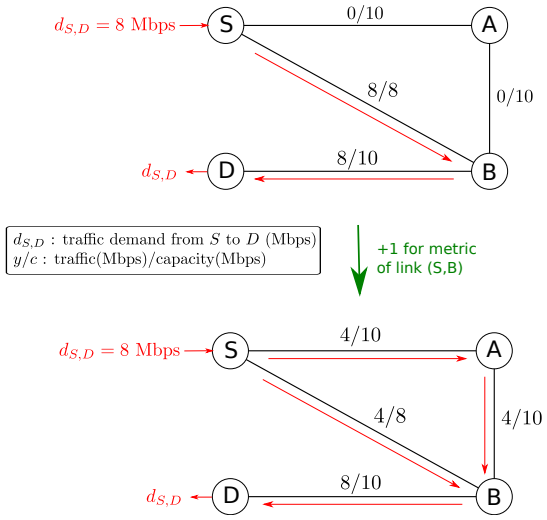

Figure 1: Metric optimization example

An illustrative example is given in Figure 1. A single flow of 8 Mbps has to be routed from node S to node D, and the capacity of all links is 10 Mbps, except link S-B whose capacity is 8 Mbps. If unit weights are used, the flow is routed along path S-B-D, leading to a maximum utilization rate equal to 100 %. However, if the weight of link S-B is increased by one, half the traffic is deviated along path S-A-B-D, resulting in a reduced maximum utilization rate of 80 % on link B-D (whereas the other utilization rates are lower than 50%). We note that the link weight optimization problem is known to be a NP-hard problem [7].

The second difficulty concerns the monitoring of OD

*Corresponding author. Telephone: +33 561 336 992
Email addresses: jvallet@laas.fr (Josselin Vallet), brun@laas.fr (Olivier Brun)

traffic demands. Traditional methods for link weight optimization have been designed for network planning purposes: they assume that a *predicted* traffic demand is known for each OD pair in the network, either exactly or with some form of uncertainty [8, 9, 10, 11]. In practice, traffic demands cannot be directly measured in large high-speed networks due to the high processing overhead and to the significant reporting traffic induced by current metrology solutions such as Netflow [12, 13]. An alternative approach is to use the link counts (amount of traffic sent on a link in a 5 min interval) provided by the SNMP protocol (Simple Network Management Protocol) to retrieve the actual traffic demands. However, since there are usually many more network flows than link load measures, this leads to an ill-posed inverse problem which cannot be solved without additional information (see, e.g., [14, 15, 16, 17, 18, 19] and references therein). We also emphasize that these observations provide some form of information on past traffic demands, whereas routing decisions have to be taken for future demands.

Finally, there are some difficulties related to operational constraints. First, the routing changes have to be limited in number, to avoid continuously changing the routes in the network, and they have to be incremental in nature, meaning that the current routing strategy has to be improved incrementally by changing one link weight after the other. Moreover, the reconfiguration decisions have to be taken in real-time. If one considers that link load observations are available every 5 minutes, and that intra-domain routing protocols need a few tens of seconds to converge to the new routes, then it means that the decision process should last only a few seconds. This clearly has an impact on the complexity of the decision algorithms to be used.

In this paper, we investigate how to dynamically reconfigure link weights so as to adapt to prevailing traffic. A similar problem was considered in [20], but, in contrast to the present work, the authors assume that information about the mean and the variance of the aggregate traffic from every source to every destination router is available periodically. Another closely related reference is [21], where the authors consider the same problem as us, but fail to propose an approach that can cope with real-time constraints.

We propose an online algorithm for dynamic reconfiguration of intra-domain routes depending on links loads in IP core networks. The algorithm uses SNMP to regularly collect link load measures, from which a set of possible traffic matrices is derived. A simple robust optimization heuristic is then used to minimize the congestion rate of the network, i.e., the utilization rate of the most loaded link. Simulation results as well as results obtained on real traffic data show that the proposed method, despite its simplicity, allows to greatly improve network performances, and has running times compatible with an online execution, even for large IP networks.

We note that the proposed online mechanism can help to mitigate the effects of route flapping in OSPF networks. If an OSPF router doesn't receive 4 consecutive hello packets from its neighbour, it will safely assume that its neighbour is unreachable/down and subsequently purges all the routes from its routing table that were once reachable via this neighbour. If a link goes down due to severe congestion (hello packets are lost due to buffer overflow), this can induce massive shift of traffic from one route to another. The original link will soon stabilize because the traffic (and possibly congestion) has moved to another link, and will comes up as available. Once it starts getting back all its traffic, it will start getting congested again. This can cause repeated traffic shifts with no apparent solution. The dynamic scheme we propose could help to avoid this phenomenon by balancing load among the routes before congestion occurs.

The paper is organized as follows. Section 2 is devoted to the mathematical formulation of the problem. The proposed algorithm and its details are described in Section 3. Results obtained on simulated and real traffic data are presented in Section 4. Some conclusions are drawn in Section 5.

2. Problem statement

The network is represented as a graph $G = (V, E)$. The set V is composed of the N nodes of the network, while the set E is composed of the M links of the network. We denote by c_l the capacity of link l, and let $K = N(N-1)$ be the number of OD pairs.

We observe the network at discrete time points $\tau = 1, 2, \ldots$ Let $\hat{\mathbf{y}}^\tau = (\hat{y}_1^\tau, \ldots, \hat{y}_M^\tau)$ be the vector of measured link traffics, where $\hat{y}_l^\tau$ gives the average traffic over link l between times $\tau - 1$ and τ. These measures provide an indirect observation on the average traffic demands in this time interval. Although conceptually traffic demands are represented in matrix form, it is more convenient to use a vector representation. Thus, we order the OD pairs and let d_k^τ be the average traffic transmitted by OD pair k in the time interval $\mathcal{I}_\tau = [\tau - 1, \tau]$. We let $s(k)$ and $t(k)$ be the source and destination nodes of demand k, respectively. We denote the vector of traffic demands by $\mathbf{d}^\tau$. We emphasize that traffic demands have to be interpreted as *offered traffic*, i.e., the traffic which would be carried if link capacities were infinite. Indeed, when evaluating the benefits that can be expected from deviating an OD flow from a link, it is more convenient to think in terms of offered traffic. We note that, as a consequence, the utilization rate of a link can be higher than 100%.

We can control the network by changing the weights of the links. Let $\boldsymbol{\omega}^\tau = (\omega_1^\tau, \ldots, \omega_M^\tau)$ be the vector describing the link weight configuration in the time interval $\mathcal{I}_\tau$, where the metric ω_l^τ of link l is an integer value in the interval $\Omega = \left[1, 2^{16} - 1\right]$. The shortest paths resulting from the weight configuration $\boldsymbol{\omega}^\tau$ are modelled by an $M \times K$ routing matrix $\mathbf{F}(\boldsymbol{\omega}^\tau)$ whose rows represent the links of the network and columns represent the OD pairs.

Element $f_{l,k}(\boldsymbol{\omega}^\tau)$ is the fraction of demand k sent across link l. These values are readily obtained from the weight vector using any shortest path algorithm, e.g., Dijkstra's one. We emphasize that they are not constrained to be 0 or 1 due to load-balancing on equal cost paths.

OD demands and observed link traffics are then related through the following linear relation

$$\hat{\mathbf{y}}^\tau = \mathbf{F}(\boldsymbol{\omega}^\tau)\,\mathbf{d}^\tau. \tag{1}$$

By changing the weight configuration , we can control the routing matrix, and thus the resulting link loads. The main difficulty here is that we have to choose the weight configuration $\boldsymbol{\omega}^\tau$ at time τ without any means to predict the future traffic demands between times τ and $\tau+1$. Since we have no information on the future traffic, our approach is just to react to the observed network congestion by optimizing the weight configuration for the current traffic demands $\mathbf{d}^\tau$. In some sense, we proceed as if $\mathbf{d}^{\tau+1} = \mathbf{d}^\tau$. The basic idea is that, even if this modelling assumption is not fully satisfied, there is some form of stability in the traffic that allows to make the right decisions using the most recent knowledge.

The problem we address amounts to finding the weight configuration $\boldsymbol{\omega}^\tau$ that minimizes the congestion rate of the network (defined as the maximum utilization rate of the links) over the time interval $\mathcal{I}_\tau$. Formally, the problem is as follows:

$$\text{minimize } \rho(\boldsymbol{\omega}) = \max_{l\in E} \frac{y_l}{c_l} \qquad \text{(METRIC)}$$

subject to

$$\mathbf{y} = \mathbf{F}(\boldsymbol{\omega})\,\mathbf{d}^\tau,$$
$$\boldsymbol{\omega} \in \Omega^M.$$

We note that the solution of the above problem depends on the unknown traffic demands at time τ. These traffic demands have to be inferred from the available observations. We describe the proposed approach to solve this problem in the following section.

3. Online algorithm

The proposed online algorithm for dynamic reconfiguration of IP routes is described in Figure 2. This algorithm is run periodically. It first uses SNMP to collect the average traffic on each link over the last time window. These observations are then used to estimate a demand uncertainty set, over which the routing metrics have to be optimized. A robust greedy heuristic is then used to determine if some weight changes can be applied to reduce the network congestion. If no such weight changes are found, the algorithm becomes idle until the next period. Otherwise, the weight changes are applied in the network, and the algorithm waits for the new SNMP data. We describe below the main steps of the algorithm.

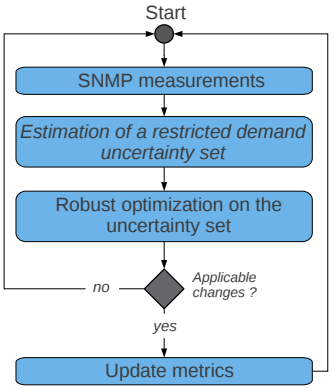

Figure 2: Online algorithm

Remark 1. *SNMP is an Internet-standard protocol for managing devices on IP networks, such as routers. The MIB variable ifOutOctets, which is used to compute the link count (number of bytes output by the interface), is directly set by the SNMP agent running on a router for each of its network interfaces. This value should thus be perfectly accurate. The time required to collect all link counts using SNMP is difficult to evaluate precisely, since it depends on many factors, but it should be in the order of a few seconds (depending on the size of the network, on the architecture of the network management tool, etc...). The resulting traffic overhead can be neglected, and, since SNMP is a low priority process as far as the CPU scheduler is concerned, the SNMP agent should not affect performance of the router.*

3.1. Traffic matrix estimation

In order to solve problem (METRIC), we have to know the traffic demands at time τ. In practice, these traffic demands are unknown and have to be inferred from the available SNMP measures. We assume that at time τ we obtain the following data regarding the traffic demands in the time interval $\mathcal{I}_\tau$:

- Traffic $\hat{y}_l^\tau$ on each link $l \in E$,
- average ingress traffic $b_n^{i,\tau}$ and egress traffic $b_n^{e,\tau}$ of each edge router n.

The vector $\mathbf{d}^\tau$ of traffic demands on the interval $\mathcal{I}_\tau$ is related to the vector $\hat{\mathbf{y}}^\tau$ of observed link traffics through equation (1). Although the routing matrix $\mathbf{F}(\boldsymbol{\omega}^\tau)$ is known at time τ, this equation does not allow in general to determine the traffic demands. Indeed, this problem is an ill-posed inverse problem because in almost any network, the number of OD pairs K is much higher than the number of links M. Nevertheless, we can define the polytope $\mathcal{D}^\tau$ of traffic matrices that comply with the observations as the set of vectors $\mathbf{d} \in \mathbb{R}_+^K$ such that

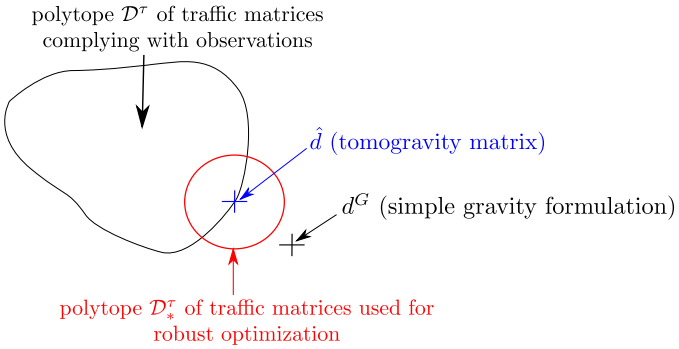

Figure 3: Matrices and uncertainty set used for optimization

$$\hat{\mathbf{y}}^\tau = \mathbf{F}(\boldsymbol{\omega}^\tau)\,\mathbf{d}, \tag{2}$$

$$\sum_{k:s(k)=n} d_k = b_n^{i,\tau}, \quad \forall n \in V, \tag{3}$$

$$\sum_{k:t(k)=n} d_k = b_n^{e,\tau}, \quad \forall n \in V. \tag{4}$$

Since we cannot determine exactly the traffic demands at time τ, we have to use additional information in order to estimate it. Existing methods for traffic matrix estimation differ in the nature of the additional information used. Starting with [22], a first generation of methods has tried to use link load covariances as this additional information (see e.g., [23, 24, 25]). These methods are based on statistical assumptions on the traffic demands whose validity is questionable. A second generation of methods use spatial or temporal prior information [26, 15, 16, 17]. The latest generation of methods use posterior spatial and temporal information. They require a calibration phase where the traffic matrix is regularly measured using Netflow over a certain period of time and then use filtering techniques to predict future traffic matrices [18, 19]. Although the accuracy of these methods is significantly superior to that of first generations methods, their drawback is the overhead induced on the network during the calibration phase. We have chosen to use the "tomogravity" method introduced in [26] because it is the simplest and fastest method of the second generation, it does not rely on statistical assumptions on traffic demands as do first generation methods, and does not introduce extra-overhead on the network as is the case for third generation methods. The basic idea of the method is to consider the traffic vector $\mathbf{d}^G$ obtained using the following simple formulation:

$$d_k^G = \frac{b_{t(k)}^{e,\tau}}{\sum_{n \neq s(k)} b_n^{e,\tau}}\, b_{s(k)}^{i,\tau}. \tag{5}$$

In other words, the traffic vector $\mathbf{d}^G$ is obtained by assuming that the total ingress traffic at a node is split between the demands originating from that node in proportion to the egress traffics of their destination nodes.

In general, the traffic demands d_k^G are not consistent with the observations, i.e., $\mathbf{d}^G \notin \mathcal{D}^\tau$. To obtain traffic demands complying with the observations, the tomogravity method computes the projection of the traffic vector $\mathbf{d}^G$ on the polytope $\mathcal{D}^\tau$ (see Figure 3). The estimated traffic vector $\hat{\mathbf{d}}$ is thus the vector of $\mathcal{D}^\tau$ minimizing the distance to $\mathbf{d}^G$. Choosing the infinity norm distance, the traffic vector $\hat{\mathbf{d}}$ is therefore the solution of the following linear programming problem:

$$\begin{array}{lll} \text{minimize} & \alpha & \\ \text{subject to} & & \\ & \alpha \geq |\hat{d}_k - d_k^G| & k = 1, \ldots, K \\ & \hat{y}_l^\tau = \sum_{k=1}^{K} f_{l,k}(\boldsymbol{\omega}^\tau)\hat{d}_k, & \forall l \in E \\ & \sum_{k:s(k)=n} \hat{d}_k = b_n^{i,\tau}, & \forall n \in V \\ & \sum_{k:t(k)=n} \hat{d}_k = b_n^{e,\tau}, & \forall n \in V \end{array}$$

Note that the minimal value of α has to be interpreted as the distance between the tomogravity traffic vector $\mathbf{d}^G$ and the polytope of traffic matrices complying with SNMP measures. Therefore, it does not give the distance to the actual traffic matrix of the network.

Our online algorithm uses the CPLEX library (C++) [27] to compute the traffic vector $\hat{\mathbf{d}}$ once the SNMP data are available. The estimated traffic demands $\hat{d}_k$ can be obtained very quickly, in a fraction of seconds for small networks, and in just a few seconds for very large ones. They are then used to optimize the link weights for the time interval $\mathcal{I}_{\tau+1}$.

If a topology modification (e.g., a link going down or up) occurs between the time instants $\tau-1$ and τ, the above minimization problem can become infeasible. In such a case, to avoid taking wrong decisions due to inconsistent information, no weight change is applied and the algorithm waits for the next consistent SNMP measures.

3.2. Robust optimization of link weights

The vector $\hat{\mathbf{d}}$ of estimated traffic demands can be used to obtain lower and upper bounds on the demand of each traffic flow in the time interval $\mathcal{I}_\tau$:

$$(1-\gamma)\,\hat{d}_k \leq d_k^\tau \leq (1+\gamma)\,\hat{d}_k, \quad \forall k = 1, \ldots, K, \tag{6}$$

where the parameter γ has to be adjusted to take into account the estimation error. Since we know exactly the ingress and egress traffics of each edge router in the time interval $\mathcal{I}_\tau$, we also require that

$$\sum_{k:s(k)=n} d_k^\tau = b_n^{i,\tau}, \quad \forall n \in V, \tag{7}$$

$$\sum_{k:t(k)=n} d_k^\tau = b_n^{e,\tau}, \quad \forall n \in V. \tag{8}$$

Require: $\boldsymbol{\omega}^\tau$ and $\mathcal{D}_*^\tau$

1: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega}^\tau$; $\boldsymbol{\omega}^* \leftarrow \boldsymbol{\omega}$

2: **while** $n < N_{max}$ **and** $q < Q_{max}$ **do**

3: Choose $\ell \in \arg\max_{l \in E} \max_{\mathbf{d} \in \mathcal{D}_*^\tau} \frac{y_l(\boldsymbol{\omega}, \mathbf{d})}{c_l}$

4: $\Delta_\ell \leftarrow \arg\min_{\Delta \geq 1} \left[\sum_k f_{l,k}(\boldsymbol{\omega} + \Delta\, \mathbf{e}_\ell) < \sum_k f_{l,k}(\boldsymbol{\omega}) \right]$

5: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \Delta_\ell \mathbf{e}_\ell$

6: Compute $\rho(\boldsymbol{\omega})$

7: **if** $\rho(\boldsymbol{\omega}) \leq \rho(\boldsymbol{\omega}^*)$ **then**

8: $\boldsymbol{\omega}^* \leftarrow \boldsymbol{\omega}$; $q \leftarrow 0$

9: **else**

10: $q \leftarrow q + 1$

11: **end if**

12: $n \leftarrow n + 1$

13: **end while**

Figure 4: Greedy heuristic for robust weight optimization.

We denote by $\mathcal{D}_*^\tau$ the set of traffic vectors $\mathbf{d}$ satisfying (6)-(8). The set $\mathcal{D}_*^\tau$ describes the uncertainty on the traffic demands at time τ. As will be explained in Section 3.2.2, the main advantage of using the polytope $\mathcal{D}_*^\tau$ instead of $\mathcal{D}^\tau$ is that it greatly simplifies the computation of worst-case link loads. Defining $y_l(\boldsymbol{\omega}, \mathbf{d}) = \sum_k f_{l,k}(\boldsymbol{\omega})\, d_k$ as the traffic on link l under the weight configuration $\boldsymbol{\omega}$ when the traffic vector is $\mathbf{d}$, and noting that

$$\begin{aligned} \rho(\boldsymbol{\omega}) &= \max_{\mathbf{d} \in \mathcal{D}_*^\tau} \max_{l \in E} \frac{1}{c_l} y_l(\boldsymbol{\omega}, \mathbf{d}), \\ &= \max_{l \in E} \max_{\mathbf{d} \in \mathcal{D}_*^\tau} \frac{1}{c_l} y_l(\boldsymbol{\omega}, \mathbf{d}), \end{aligned} \tag{9}$$

we obtain the following equivalent formulation of problem (METRIC)

$$\begin{aligned} &\text{minimize } \rho(\boldsymbol{\omega}) = \max_{l \in E} \max_{\mathbf{d} \in \mathcal{D}_*^\tau} \frac{1}{c_l} y_l(\boldsymbol{\omega}, \mathbf{d}) \\ &\text{subject to } \boldsymbol{\omega} \in \Omega^M \end{aligned}$$

This new formulation lends itself better to optimization because, as explained in Section 3.2.2, it allows to use minimum cost flow algorithms to compute the worst-case utilization rate of a link. Our online algorithm uses a greedy heuristic to incrementally solve this problem. This greedy heuristic is described in Figure 4.

The greedy heuristic first initializes the best weight configuration $\boldsymbol{\omega}^*$ and the current weight configuration $\boldsymbol{\omega}$ to $\boldsymbol{\omega}^\tau$. At each iteration of the optimization loop, it arbitrarily selects a link ℓ among those with the maximum worst-case utilization rate (line 3), and computes the minimum metric increment Δ_ℓ to deviate at least one OD flow (in whole or in part) from this link (line 4). The current weight configuration $\boldsymbol{\omega}$ is then set to $\boldsymbol{\omega} + \Delta_\ell \mathbf{e}_\ell$, where $\mathbf{e}_\ell$ is the M-vector with 1 in position ℓ and 0 elsewhere, and the network congestion rate under weight configuration $\boldsymbol{\omega}$ is evaluated (line 6). If it is reduced with respect to the best weight configuration $\boldsymbol{\omega}^*$, then $\boldsymbol{\omega}^*$ is updated (line 8). To avoid getting stuck in a local minimum, the heuristic also allows the network congestion rate to increase for a limited number of iterations $q \leq Q_{max}$. The convergence is reached when either $q = Q_{max}$ or when the maximum number of iterations N_{max} is attained (line 2). We describe in more detail the main steps of the heuristic below.

3.2.1. Minimum metric increment Δ_ℓ

The basic idea of the greedy heuristic is to deviate traffic from a link ℓ having the maximum worst-case load. This is done by increasing the weight of that link by the minimum quantity Δ_ℓ such that at least one OD flow k sent through link ℓ is deviated from that link. Formally, Δ_ℓ is computed using the formula given in line 4 of Figure 4.

We emphasize that, although ℓ is the link with the maximum worst-case utilization rate, the quantity Δ_ℓ is independent of the traffic demands. We first observe that if there exists an OD flow k such that $0 < f_{l,k}(\boldsymbol{\omega}) < 1$, then this flow is split among several shortest paths. In this case, a metric increment $\Delta_\ell = 1$ is sufficient to deviate flow k from link ℓ. Otherwise, we use the technique proposed in [4]. The basic idea is to remove link ℓ from the network and to analyze how the distances between source and destination nodes increase for those flow k such that $f_{l,k}(\boldsymbol{\omega}) = 1$. Choosing Δ_ℓ as the minimum increase in shortest path lengths introduces load-sharing for at least one of these OD flows, thus deviating in part this flow from link ℓ. We refer to [4] for further details.

3.2.2. Worst-case evaluation of link loads

One of the key operations of the greedy heuristic is the evaluation of the worst-case loads of the links. This operation is required when choosing the link ℓ (line 3), and when computing the network congestion rate (line 6) after the metric of that link has been increased. We first note that we do not need to recompute the worst-case load for all links, but only for those links for which the parameters $f_{l,k}$ have been updated. Let l be one of these links. Noting that

$$\sum_n b_n^{e,\tau} - \sum_k d_k^\tau = 0, \tag{10}$$

the worst-case load of link l can be written as follows:

$$\begin{aligned} \max_{\mathbf{d} \in \mathcal{D}_*^\tau} y_l(\boldsymbol{\omega}, \mathbf{d}) &= \sum_n b_n^{e,\tau} + \max_{\mathbf{d} \in \mathcal{D}_*^\tau} \sum_k (f_{l,k}(\boldsymbol{\omega}) - 1) d_k, \\ &= \sum_n b_n^{e,\tau} - \min_{\mathbf{d} \in \mathcal{D}_*^\tau} \sum_k (1 - f_{l,k}(\boldsymbol{\omega})) d_k. \end{aligned}$$

The computation of $\max_{\mathbf{d} \in \mathcal{D}_*^\tau} y_l(\boldsymbol{\omega}, \mathbf{d})$ therefore reduces to solving the following minimization problem:

$$\begin{cases} \min_{\mathbf{d}\in\mathbb{R}_+^K} & \sum_k \left(1 - f_{l,k}(\boldsymbol{\omega})\right) d_k \\ \text{s.t.} & \\ & (1-\gamma)\hat{d}_k \leq d_k \leq (1+\gamma)\hat{d}_k \quad ,\forall k \\ & \sum_{k:s(k)=n} d_k = b_n^{i,\tau} \quad , \; n \in V \\ & \sum_{k:t(k)=n} d_k = b_n^{e,\tau} \quad , \; n \in V \end{cases}$$

The structure of the above linear problem is that of a standard minimum cost flow problem on a bipartite graph. It can be solved very efficiently using a dedicated algorithm [28]. The advantage of using the polytope $\mathcal{D}_*^\tau$ instead of the polytope $\mathcal{D}^\tau$ is precisely here: it allows to drastically reduce the computing times by solving a minimum cost flow problem instead of a general linear programming problem. We have used the software library LEMON (C++) [29] to solve this problem.

3.3. Reduction of the number of metric changes

Recall that, to avoid being trapped in local minima, some weight changes can lead to an increase of the network congestion rate. We therefore regroup the weight changes proposed by the greedy heuristic into groups, such that when all weight changes of the same group are applied, the network congestion rate is strictly improved.

After some changes in the weight configuration, the intra-domain routing protocol needs some time before converging to the new routes, and the convergence time depends on the number of changes. It is therefore sensible to restrict the number of weight changes to those that are really needed. The following rules are used to reduce the number of metric changes:

- the greedy heuristic stops if the weights of more than MAX_METRIC distinct links have been changed,
- the latest group of weight changes reducing the network congestion rate by less than a certain threshold $G_{threshold}$ is removed.

Figure 5 provides an illustration of these rules with $MAX\_METRIC = 10$. Each step corresponds to the production of a new group of weight changes. The cumulative number of links whose weight has to be changed is shown in the figure at each step (note that it does not increase from steps 3 to 4 because the weight changes concern already modified links). In this example, the heuristic proposes 6 groups of weight changes. We keep only the first four because : (a) group 6 exceeds the maximum number of allowed metric changes, and (b) group 5 provides a gain below the threshold $G_{threshold}$.

4. Results

In this section, we analyse how the network congestion rate evolves in time using the proposed online algorithm.

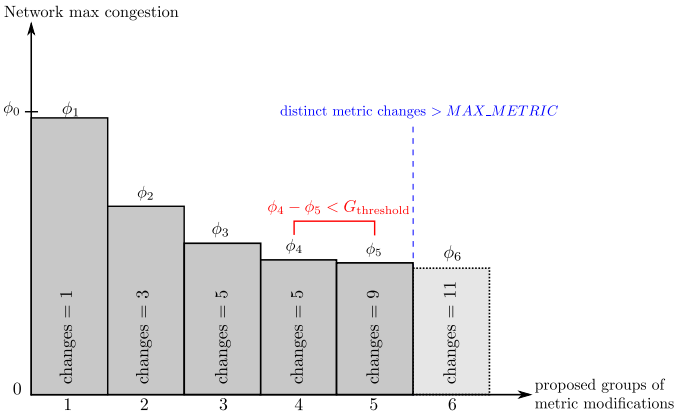

Figure 5: Restricting the number of metric changes

Table 1: Topologies characteristics : number of nodes (N) and links (M)

Topology	N	M
ABOVENET	19	68
ARPANET	24	100
BHVAC	19	46
EON	19	74
METRO	11	84
NSF	8	20
PACBELL	15	42
VNSL	9	22

The analysis is done for simulated traffic data and real traffic data.

All results presented were obtained using the following parameters: $N_{max} = 100$, $Q_{max} = 10$, $G_{threshold} = 2\%$, $MAX\_METRIC = 10$ and various values of parameter γ. The C++ implementation code was compiled using the GCC compilator, using the -O3 level of optimization. All simulations have been performed on a Intel Core i5-2430M processor at 2.4 GHz, running under Linux with 4 GB of available memory.

4.1. Simulated traffic data

Simulations are performed on 8 real network topologies (see Table 1). Their characteristics have been found in IEEE literature (bhvac, pacbell, eon, metro, arpanet, nsf) or from the Rocketfuel project [30] for abovenet and vnsl topologies. For each topology, the initial weight of each link is set to 1.

For each network, a random traffic matrix is generated at time $\tau = 0$. We remind that the number of OD pairs is $K = N(N-1)$, where N is the number of nodes of the network topology. Each minute, the traffic matrix is updated by adding a white gaussian noise to each traffic demand:

$$d_k^t = d_k^{t-1\,minute} + \mathcal{N}(0, \sigma^2) \quad k = 1, \ldots, K. \tag{11}$$

The standard deviation of this noise is chosen so that 99.7% of the traffic demands varies by at most ±8.5% per minute (we enforce these bounds for the remaining 0.3%). As a consequence, each traffic demand can vary by at most ±50% over an interval of 5 minutes:

$$0.5d_k^{\tau-1} \leq d_k^{\tau} \leq 1.5d_k^{\tau-1} \tag{12}$$

Despite its simplicity, this model allows significant traffic variation (+-50% on five minutes). In addition, real traffic data observed on the ABILENE network (see section 4.2 for details) gives some credits to our simple model: real measured traffic exhibit bursty traffic, but on time scales of several minutes, and the maximum observed variations are also similar to the one used in our simulation (max +- 50% each five minutes). We emphasize that our method is not intended to handle large traffic variations on short time scales (less than 5 minutes), which would require truly dynamic routing algorithm, but rather to adapt routes as traffic fluctuates over larger time scales due to the behavior of users.

All our simulations on simulated traffic data consider a total time period of 250 minutes.

4.1.1. Time-average network congestion rates

To investigate the performances obtained using the online algorithm, we decide to study the time-average network congestion rate over the complete time period. To compute this value, the network congestion rate is observed at each minute. The time-average value is then defined as follows:

$$\bar{\phi} = \frac{1}{n}\sum_{t=1}^{n} \max_{l \in E} \frac{y_l^t}{c_l} = \frac{1}{n}\sum_{t=1}^{n} \max_{l \in E} \left(\frac{\sum_{k=1}^{K} f_{l,k}(\boldsymbol{\omega}^t) d_k^t}{c_l} \right) \tag{13}$$

with in our case, $n = 250$.

Three different static weight configurations are considered. The first two ones are commonly used in real networks, and correspond to Unit metrics ($\boldsymbol{\omega}_l = 1, \forall l \in E$) and CISCO metrics ($\boldsymbol{\omega}_l = 10^8/c_l, \forall l \in E$), respectively. The third considered configuration is hypothetical, and uses optimized weights. To obtain these optimized weights, we assume to know future traffics demands for the entire period. We compute the average traffic matrix over the whole temporal period of 250 minutes, and then use the local search algorithm proposed in [21] to optimize the weights for this average matrix. Finally, these optimized weights are used to define an optimized static metric configuration for the whole 250 min simulation, that is the weights are not changed during the simulation.

These three static weight configurations are compared with the proposed online algorithm. As the γ parameter (defined in section 3.2) affects the evaluation of worst case link loads, it can change the routing decisions. To better understand the influence of this parameter, a set of simulations are performed with $\gamma \in \{0, 0.25, 0.4\}$. The results are shown in Table 2.

Several comments are in order. First, the results show that significant benefits can be expected from link weight optimization. Indeed, the proposed online optimization clearly outperforms the static configurations Unit and CISCO, for all γ values different from zero, except for ABOVENET, ARPANET and VNSL topologies, where no acceptable modification was proposed. We also note the proximity between the results obtained with the proposed online algorithm and the static configuration with optimized weights. We emphasize that this last static configuration is purely theoretical in that it requires the knowledge of future traffic demands over the whole time period, which is not the case in reality. Moreover, the length of the considered time period is relatively short (250 min), which means that the traffic demands remain "close" to their average values over the time period. The larger the period of time used to compute the optimized static configuration, the greater the probability that the actual traffic matrix deviates at some point in time from the average matrix over that period, and the worse the time-average network congestion rate obtained with the static configuration with respect to those obtained with the dynamic reconfiguration algorithm.

Finally, we note the benefit of using some uncertainty on estimated traffic demands. Indeed, the congestion rates obtained with $\gamma = 0$ (considering only the tomogravity matrix, with no uncertainty on traffic demands) are higher than those obtained with $\gamma = 0.25$ or $\gamma = 0.4$. In fact, most of the time with $\gamma = 0$, except for the PACBELL topology, the obtained congestion rates are equal to the ones given by the Unit static configuration. This confirms the interest to consider a demand uncertainty set around the tomogravity matrix, to avoid bad decisions due to matrix estimation errors. Finally, due to the heuristic nature of the algorithm and also to unpredictable estimation error, it is difficult to choose the best γ value, as the results are very close between $\gamma = 0.25$ and $\gamma = 0.4$. In what follows, we will consider the value $\gamma = 0.25$, which seems to be a good compromise.

The total number of weight changes is shown in Table 3. It remains quite limited. Table 4 gives the number of time instants where reconfigurations are done. The changes are applied at just a few time instants, thus avoiding to continuously change the routes in the network. The BHVAC topology is an exception here, with more instants applying modifications than for the other networks.

4.1.2. Temporal evolution of network congestion rates

In this section, we are interested in the temporal evolution of the network congestion. We also want to study how the proposed metrics changes are applied, and their consequences on the network congestion. In this perspective,

Table 2: Time-average network congestion rate (%) for 250 minutes

Topology	**Static metrics**			**Online Algorithm**		
	Unit	**CISCO**	**Optimized static weights**	$\gamma = 0$	$\gamma = 0.25$	$\gamma = 0.4$
ABOVENET	81.00	101.20	55.69	81.00	81.00	81.00
ARPANET	81.57	96.35	75.84	81.57	81.57	81.57
BHVAC	81.62	75.13	67.30	81.62	64.80	65.62
EON	91.21	103.49	54.86	91.21	50.95	47.63
METRO	63.08	47.83	44.13	63.08	37.84	37.93
NSF	43.78	43.77	41.52	43.78	42.16	43.52
PACBELL	137.97	40.04	37.56	46.15	38.14	39.47
VNSL	59.28	59.28	58.43	59.28	59.28	59.28
Average	79.94	70.89	54.42	68.46	56.97	57.00

Table 3: Total number of weight changes for 250 minutes

Topology	$\gamma = 0$	$\gamma = 0.25$	$\gamma = 0.4$
ABOVENET	0	0	0
ARPANET	0	0	0
BHVAC	0	34	49
EON	0	28	42
METRO	0	25	28
NSF	0	6	12
PACBELL	1	9	9
VNSL	0	0	0

Table 4: Number of epochs where weight changes are applied, for 250 minutes

Topology	$\gamma = 0$	$\gamma = 0.25$	$\gamma = 0.4$
ABOVENET	0	0	0
ARPANET	0	0	0
BHVAC	0	16	17
EON	0	4	6
METRO	0	5	6
NSF	0	2	3
PACBELL	1	2	1
VNSL	0	0	0

we decide to plot the network congestion and the weight changes during the considered time period.

To evaluate the gain of using an adaptive routing scheme, we compare with the static weight configuration where all metrics are set to 1 (Unit metrics). To assess the loss in performance due to the uncertainty on traffic demands, we also compare with the results obtained by the greedy heuristic in the case where the traffic demands are known. Finally, we also wish to measure the gap with respect to the network congestion rates that would be obtained using the optimal weight configuration at each time step if the traffic demands were known. Since there is no exact optimization algorithm to solve the weight optimization problem, we compute instead a lower bound on the optimal network congestion rate by solving the following multipath routing problem:

$$\min\ z \qquad \text{(MULTIPATH)}$$

s.t.

$$\sum_k f_{l,k}\, d_k^{\tau+1} \le c_l\, z,\ \forall l \in E, \tag{14}$$

$$\sum_{l\in\delta^+(n)} f_{l,k} - \sum_{l\in\delta^-(n)} f_{l,k} = h_k^n,\ \forall k, \forall n \in V, \tag{15}$$

$$0 \le f_{l,k} \le 1,\ \forall l \in E,\ \forall k \tag{16}$$

where $\delta^+(n)$ (resp. $\delta^-(n)$) denotes the set of incoming (resp. outgoing) arcs at node n, while h_k^n is 1 if $v = s(k)$, -1 if $v = t(k)$ and 0 otherwise. Note that inequalities (16) ensure that there is no restriction on the routing scheme. Note also that it is assumed that future traffic demands $d_k^{\tau+1}$ are known at time τ. We emphasize that this lower bound is potentially much lower than the network congestion rate obtained under the optimal weight configuration.

Figures 6, 7 and 8 present the results obtained for the topologies BHVAC, EON and METRO, respectively (similar results are obtained for the other topologies). These

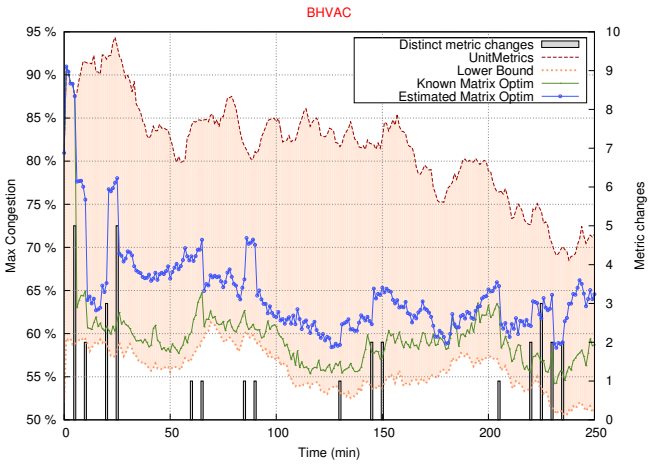

Figure 6: Congestion for BHVAC topology

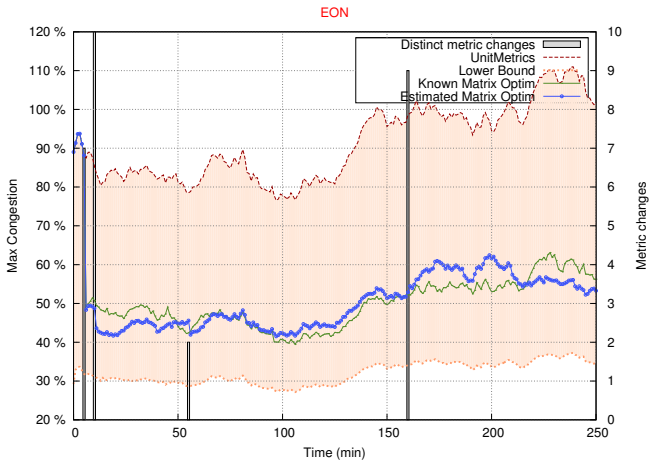

Figure 7: Congestion for EON topology

figures show the evolution of the network congestion rate. The x-axis represents the time τ. Two vertical axes are present : the left one is the measured congestion rate of the network. The right one gives the number of weight changes made by the online algorithm. Four curves are plotted on each graph:

- Lowerbound : the lower bound obtained by solving problem (MULTIPATH),
- UnitMetrics : congestion rate obtained with unitary metrics,
- KnowMatrixOptim : congestion rate obtained with the greedy heuristic if the traffic matrix was known,
- EstimatedMatrixOptim : congestion rate obtained with the proposed online algorithm.

Vertical bars represent the number of link weights modified by the online algorithm. No vertical bar means that the algorithm does not change the routing.

We first observe that the use of an adaptive routing scheme allows to significantly reduce the network congestion rate with respect to a static weight configuration such as that with unitary metrics. We also observe that for all topologies there is no significant loss in performance due to the uncertainty on traffic demands. In practice, the estimation of the traffic matrix seems sufficiently accurate to perform interesting optimization. The robust optimization algorithm even performs better than if the traffic matrix was known in some cases: this can be explained by the heuristic nature of the algorithm. We note that for the BHVAC and EON topologies, the network congestion rate obtained with the online algorithm is often fairly close to the lower bound, indicating that this algorithm provides a near-optimal weight configuration. Finally, we also observe that the number of metric changes is quite limited, thus avoiding to continuously change the routes in the network.

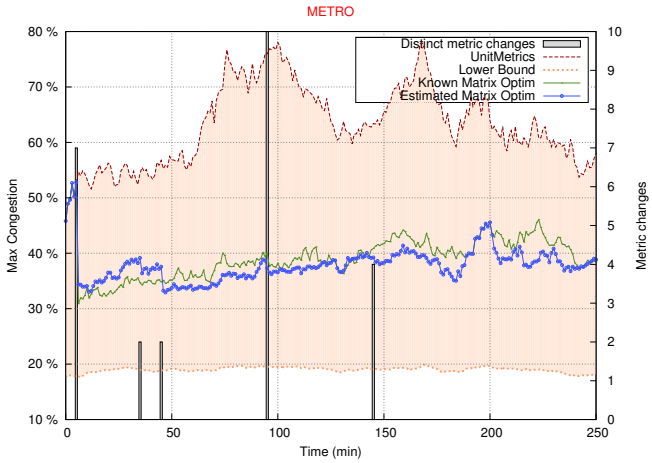

Figure 8: Congestion for METRO topology

4.1.3. Execution time

The execution time of the online algorithm at each time step is critical in this study. The worst per-step execution times are presented in Table 5. For all the networks, the routing decisions can be taken in a few fractions of seconds, which is clearly compatible with online processing. An experiment on two larger network topologies, which we call BRITE1(50 nodes, 200 links) and BRITE2 (100 nodes, 400 links), generated using the BRITE tool [31], has shown that execution times can increase very fast as the size of the topology grows. The worst execution time was equal to 4.6 seconds for BRITE1 and around 24 seconds for BRITE2. This is certainly due to the size of the problem we consider, which also grows very fast with network size (for example, the routing fractions $f_{l,k}$ is composed of $N * N * M$ values which needs to be computed and used). For example, on the BRITE2 topology experiment, the traffic matrix estimation took around 4.6 seconds, and the repeated calculation of the worst cost evalution (cf section 3.2.2) problem took around 5.5 seconds to compute in total. The remaining time is spend computing the routes and

Table 5: Worst per-step execution times (s)

Topology	Time
ABOVENET	0.15
ARPANET	0.56
BHVAC	0.17
EON	0.21
METRO	0.07
NSF	0.04
PACBELL	0.10
VNSL	0.04

especially the corresponding routing fractions $f_{l,k}$. Unfortunately, ISPs generally regard their router-level topologies as confidential and, as a result, real topologies are not publicly available. Available public data (see Table 1 and the ABILENE network in Section 4.2) suggest however that real core networks have far fewer nodes and links than the BRITE2 topology

4.2. Real traffic data

We now present the results obtained with real traffic data recorded in 2004 on the ABILENE network (12 nodes and 30 links) [32]. Unfortunately, we were not able to obtain more recent traffic data because such data are not publicly available. A recent traffic matrix would probably have higher traffic levels, although it is difficult to say what it would look like since there is probably not a single representative traffic matrix.

The ABILENE traffic data were collected every 5 minutes during 24 weeks (6 months). We needed to choose one week of traffic to work on. The motivation for choosing a specific week of traffic was three-fold. First, we wanted a week with a clear day-to-day traffic variability. Second, we wanted a week exhibiting significant traffic variations on short time-scales. Finally, we wanted a week with a significant gap between the average and the maximum link utilizations, indicating room for optimization. Week 4 was one the weeks complying to all criteria, and this is why only the fourth week is used to carry out our experiments.

Assuming unitary metrics, the evolution of the network congestion rate is plotted in Figure 9. The average utilization rate of the links is also plotted. In contrast to the network congestion rate, variations in average utilization rate of the links are very slight, indicating that there is a true potential for a better load distribution in the network.

We analyse in more detail three phases of one hour traffic (see Figure 9): phase 1 corresponds to a period of low congestion ($\approx 10\%$), phase 2 corresponds to a period when load increases and with important traffic variations (see Figure 10), and phase 3 is a period of high congestion ($\approx 55\%$). For the three phases, we compute the average traffic demand of each OD flow over the one hour period, sort them in the order of decreasing traffic demands and then retain only the first OD flows that represent 90% of the total traffic (those "elephants" that are really meaningful for traffic engineering purposes). Table 6 gives statistical data regarding the relative variation (in absolute value) of these large flows.

Table 6: Maximum traffic variations (%) for large flows

Percentile (%)	Phase 1	Phase 2	Phase 3
100	83	142395	87
99	41	74	48
98	33	43	35
95	21	31	26
90	16	25	18

We note that 98% of the flows have relative variations lower than 43%, so that the constraint (12) is satisfied for almost all traffic demands k. The large value observed for the 100th percentile of phase 2 is caused by OD demands which were very close to zero at some measurement epoch, and became non-negligible at the next one.

The same study is performed for ingress/egress traffics for all the nodes in the network. Results obtained for ingress and egress are similar, and Table 7 gives statistical data for ingress traffics. We note that 90% of ingress traffics have relative variations lower than 19% for phase 2 and lower than 12 % for phase 1 and 3. These variations are not completely insignificant, but the obtained results show that it does not affect the optimization quality.

Table 7: Maximum ingress traffic variations (%) for all nodes

Percentile (%)	Phase 1	Phase 2	Phase 3
100	49	1062	36
95	15	28	14
90	10	19	12

The optimization results obtained for the ABILENE topology, with $\gamma = 0.25$ and the 4th week traffic data are shown in Figure 11.

The proposed dynamic routing algorithm allows a significant reduction of the network congestion rate when the traffic is high: the maximum congestion rate of the network is decreased from 55% with unitary metrics to about 45%.

Using the static optimized metrics configuration defined in 4.1.1, the time-average network congestion rate for the whole 4th week of ABILENE is visible in Table 8. As expected, the online algorithm provides slightly better results than the optimized static weights configuration ones. Furthermore, with both configuration, when the traffic is high, the congestion rate is decreased to about 45%. This proximity is a good result, considering that the optimized static weight is hypothetical, as we can not know future

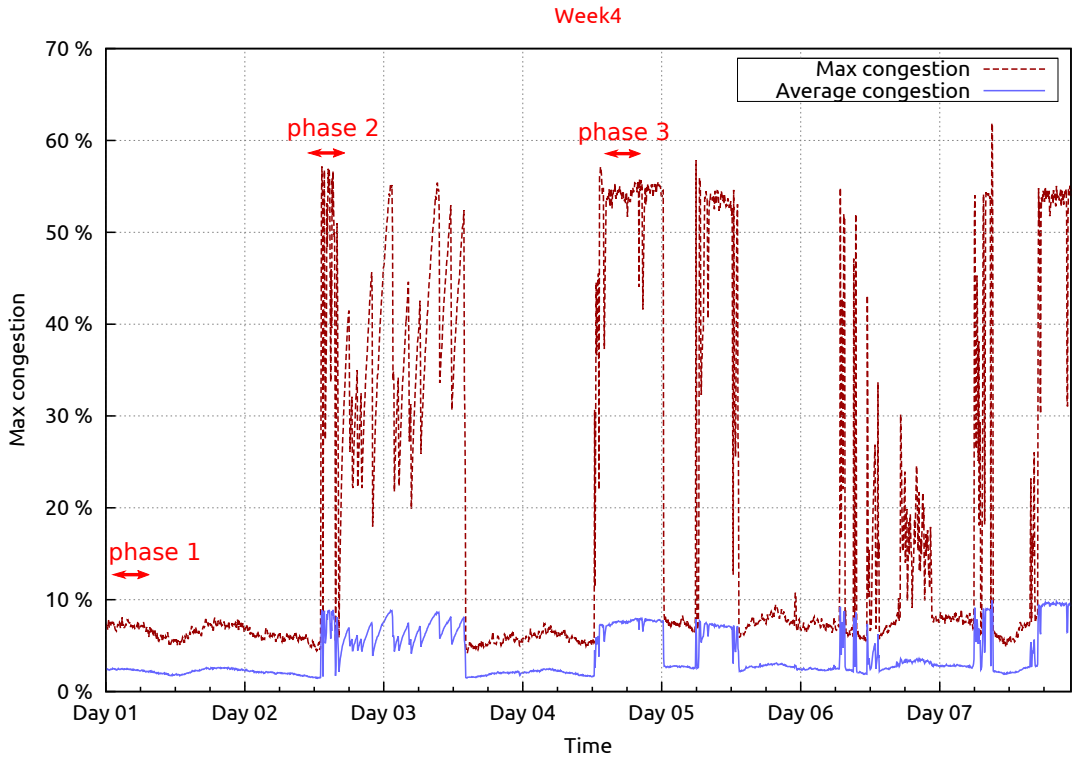

Figure 9: Link congestion rates for 4th week on ABILENE, using unit metrics

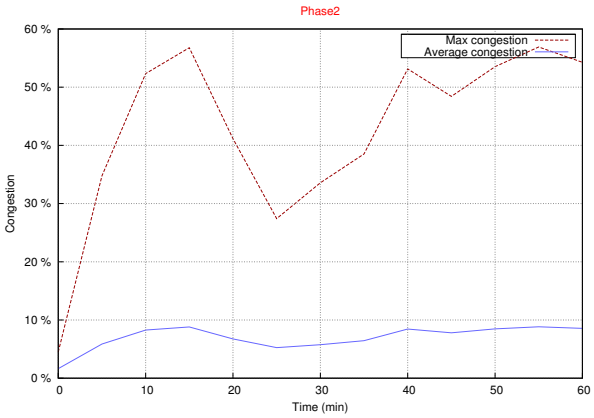

Figure 10: Zoom on phase 2

traffic.

Concerning execution times, they remain below 0.07 seconds, which is fully compatible with an online execution. The total number of modifications is also very limited. For $\gamma = 0.25$, there are only 17 time instants where routes are reconfigured for the whole week (which is composed of around 2000 instants), and the total number of applied metric modifications for the whole week is 27.

4.3. Link failure

In this section, we study the results obtained with the algorithm when a link failure occurs in the network. More precisely, we compare the network congestion obtained with unitary metrics and with the proposed online algorithm when a link goes down at an arbitrary time instant T_{down}, and then goes up at time $T_{up} > T_{down}$. We use the same simulation protocol as that used in section 4.1. The same link is cut in the different configurations: it is one of the links with the greatest utilization rate under unitary weights at the time of failure. Three examples of results are presented in Figures 12, 13 and 14, all obtained with $T_{down} = 97^{th}$ minute, $T_{up} = 157^{th}$ minute and $\gamma = 0.25$. The same kind of results is obtained for the others topologies.

As explained at the end of Section 3.1, the demand estimation problem can become infeasible when the link goes down, and when it reappears. In these two cases, the algorithm stays idle until the next SNMP measures become available. For $T_{down} = 97^{th}$ minute and $T_{up} = 157^{th}$ minute, it means that the algorithm does not propose any

Table 8: Time-average network congestion rate (%) for the whole ABILENE 4th week

Configuration	**Time-average congestion**
Unit metrics	19.37 %
Optimized static weights	17.84 %
Online algorithm (γ=0.25)	16.18 %
Online algorithm (γ=0.4)	16.42 %

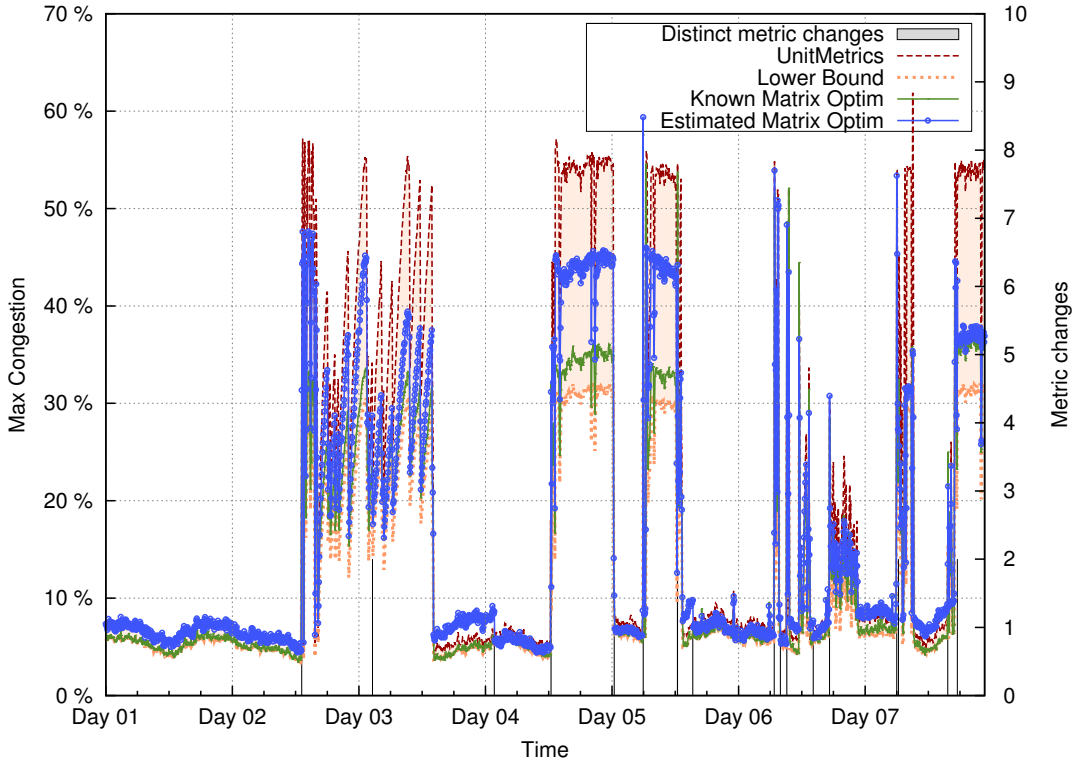

Figure 11: Congestion for ABILENE topology during the 4th week

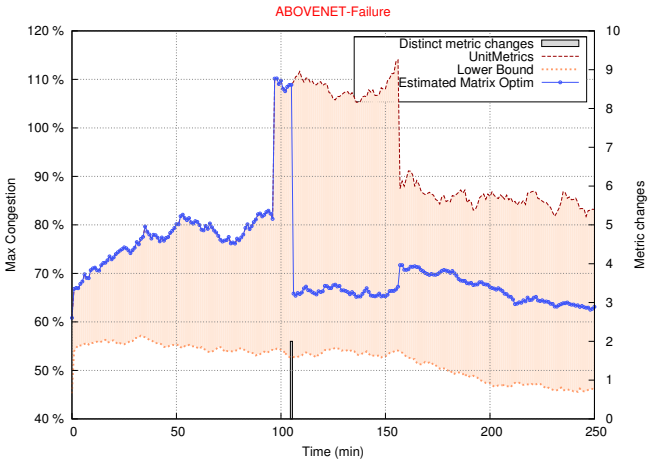

Figure 12: Congestion for ABOVENET topology with link failure

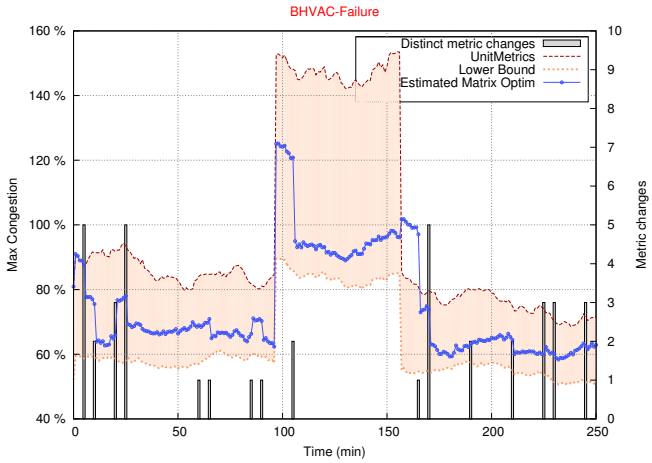

Figure 13: Congestion for BHVAC topology with link failure

modification at the 100^{th} and 160^{th} minutes.

The obtained network congestion rates are most of the time below that obtained with unitary weights. We can observe that after each event, some modifications are applied, reducing the congestion in a very efficient way. We also note that for each event, the network congestion increases less when we use the online algorithm, and the number of applied modifications remains limited. It seems that balancing the traffic load among the network links before the failure event occurs, helps to reduce its impact on network congestion.

5. Conclusion

The key idea of the proposed approach is to deviate traffic from the most loaded links. To evaluate to which extent this is interesting, we need some information on the traffic in the network. This information is obtained using a simple estimation of the current traffic from the SNMP link counts. Robust optimization is used to compensate for the estimation errors. The results obtained on simulated and real traffic data show that a significant reduction of the network congestion rate can be obtained

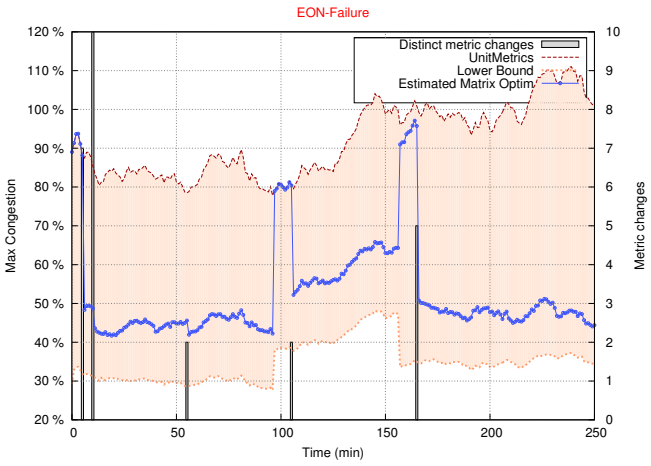

Figure 14: Congestion for EON topology with link failure

with respect to a static weight configuration, even when the traffic has significant variations in time. In the ABILENE record, the maximum congestion has been reduced to around 45% with the algorithm, while it was around 55% with the static weight configuration. It also shows that the dynamic algorithm provides a more robust configuration, giving the ability to react to traffic variation. The running times are compatible with online execution, as long as we do not consider very large network topologies. Another interesting conclusion is that, by balancing the traffic load among the network links, the proposed online mechanism helps reducing the impact of failures on network congestion when they occurs. Future work will consider the implementation of the algorithm in a real network to demonstrate its practical feasibility.

Acknowledgement

This work was supported by French FUI project NEC.

References

[1] Networking Working Group, OSPF version 2, Technical Report, Internet Engineering Task Force, 1994.

[2] J. Moy, OSPF, Anatomy of an Internet Routing Protocol, Addison-Wesley, 1998.

[3] B. Fortz, M. Thorup, Increasing internet capacity using local search., Computational Optimization and Applications 29 (2004) 13–48.

[4] C. Fortuny, O. Brun, J. M. Garcia, Metric optimization in ip networks, in: 19th International Teletraffic Congress, Beijing, China, 2005, pp. 1225–1234.

[5] H. Ümit, A column generation approach for igp weight setting problem, in: CoNEXT, Toulouse, France, 2005, pp. 294–295.

[6] B. Fortz, H. Ümit, Efficient techniques and tools for intra-domain traffic engineering, Technical Report 583, ULB Computer Science Departement, 2007.

[7] B. Fortz, M. Thorup, Internet traffic engineering by optimizing ospf weights, in: Proc. 19th IEEE Conf. on Computer Communications (INFOCOM), 2000.

[8] E. Mulyana, U. Killat, Optimizing ip networks for uncertain demands using outbound traffic constraints, in: INOC'2005, 2005, pp. 695–701.

[9] A. Altin, P. Belotti, M. Pinar, Ospf routing with optimal oblivious performance ratio under polyhedral demand uncertainty, Optimization and Engineering (2009).

[10] A. Altin, B. Fortz, H. Ümit, Oblivious ospf routing with weight optimization under polyhedral demand uncertainty, in: International Network Optimization Conference (INOC 2009), Pisa, Italy, 2009.

[11] J. Chu, C. T. Lea, Optimal link weights for ip-based networks supporting hose-model vpns, IEEE/ACM Transactions on Networking 17 (2009) 778–786.

[12] Cisco Systems, Cisco ios netflow, http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html, 2004.

[13] Cisco Systems, Sampled NetFlow, http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/12s_sanf.html, 2001.

[14] Y. Zhang, M. Roughan, N. Duffield, A. Greenberg, Fast accurate computation of large-scale ip traffic matrices from link loads, in: ACM SIGMETRICS, San Diego, USA, 2003. URL: citeseer.ist.psu.edu/article/zhang03fast.html.

[15] A. Soule, A. Lakhina, N. Taft, K. Papagiannaki, K. Salamatian, A. Nucci, M. Crovella, C. Diot, Traffic matrices: balancing measurements, inference and modeling, SIGMETRICS Perform. Eval. Rev. 33 (2005) 362–373.

[16] A. Nucci, R. Cruz, N. Taft, C. Diot, Design of IGP link weight changes for estimation of traffic matrices, in: IEEE Infocom, Hong Kong, 2004.

[17] C. Fortuny, O. Brun, J. M. Garcia, Fanout inference from link counts, in: 4th European Conference on Universal Multiservice Networks (ECUMN 2007), 2007, pp. 190–199.

[18] A. Soule, K. Salamatian, A. Nucci, N. Taft, Traffic matrix tracking using kalman filters, SIGMETRICS Perform. Eval. Rev. 33 (2005) 24–31.

[19] K. Papagiannaki, N. Taft, A. Lakhina, A distributed approach to measure IP traffic matrices, in: Internet Measurement Conference, 2004, pp. 161–174. URL: http://doi.acm.org/10.1145/1028808.

[20] T. Ye, H. T. Kaur, S. Kalyanaraman, K. S. Vastola, S. Yadav, Dynamic optimization of ospf weights using online simulation, in: IEEE INFOCOM, 2002.

[21] O. Brun, J. M. Garcia, Dynamic igp weight optimization in ip networks, in: First International Symposium on Network Cloud Computing and Applications (NCCA), 2011.

[22] Y. Vardi, Bayesian inference on network traffic using link count data, Journal of the American Statistical Association 93 (1998) 573–576.

[23] A. Medina, N. Taft, S. Battacharya, C. Diot, K. Salamatian, Traffic matrix estimation: Existing techniques compared and new directions, in: SIGCOMM, Pittsburgh, 2002. URL: citeseer.ist.psu.edu/medina02traffic.html.

[24] J. Cao, D. Davis, S. Wiel, B. Yu, Time-Varying Network Tomography : Router Link Data, Technical Report, Bell Labs, 2000.

[25] I.Juva, S.Vaton, J.Virtamo, Quick traffic matrix estimation based on link count covariances, 2006 IEEE International Conference on Communications (ICC 2006),Istanbul (2006).

[26] Y. Zhang, M. Roughan, N. Duffield, A. Greenberg, Fast accurate computation of large-scale ip traffic matrices from link loads, in: In ACM SIGMETRICS, 2003, pp. 206–217.

[27] ILOG CPLEX, http://www.ilog.com/products/cplex/, 2012.

[28] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, Network Flows, Prentice Hall, 1993.

[29] Library for Efficient Modeling and Optimization in Networks (LEMON), https://lemon.cs.elte.hu/trac/lemon, 2012.

[30] N. Spring, R. Mahajan, D. Wetherall, T. Anderson, Measuring isp topologies with rocketfuel, Networking, IEEE/ACM Transactions on 12 (2004) 2 – 16.

[31] Boston University Representative Internet Topology Generator (BRITE), http://www.cs.bu.edu/brite/, 2001.

[32] Y. Zhang, Abilene traffic matrices, http://www.cs.utexas.edu/~yzhang/research/AbileneTM/, 2004.