# Robust Traffic Engineering using L-balanced Weight-Settings in OSPF/IS-IS

Henrik Abrahamsson
Swedish Institute of Computer Science
Box 1263, SE-164 29 Kista, Sweden
Email: henrik@sics.se

Mats Björkman
Mälardalen University
Västerås, Sweden
Email: Mats.Bjorkman@mdh.se

***Abstract*—Internet traffic volumes continue to grow at a great rate, now pushed by video and TV distribution in the networks. This brings up the need for traffic engineering mechanisms to better control the traffic. The objective of traffic engineering is to avoid congestion in the network and make good use of available resources by controlling and optimising the routing function. The challenge for traffic engineering in IP networks is to cope with the dynamics of Internet traffic demands. Today, the main alternative for intra-domain traffic engineering in IP networks is to use different methods for setting the weights in the routing protocols OSPF and IS-IS. In this paper we revisit the weight setting approach to traffic engineering but with focus on robustness. We propose *l*-balanced weight settings that route the traffic on the shortest paths possible but make sure that no link is utilised to more than a given level *l*. This gives efficient routing of traffic and controlled spare capacity to handle unpredictable changes in traffic. We present a heuristic search method for finding *l*-balanced weight settings and show that it works well in real network scenarios.**

## I. Introduction

Internet traffic volumes continue to grow at a great rate, now pushed on by video and TV distribution in the networks. Increasing traffic volumes necessitate upgrades of network equipment and new investments for operators, and keep up-to-date the question of over-dimensioning network capacity versus using traffic engineering mechanisms for better handling the traffic. In addition, as new bandwidth demanding and also delay and loss sensitive services are introduced, it is even more important for the operator to manage the traffic situation in the network.

The main challenge for traffic engineering is to cope with the dynamics of traffic demands and topology. How to best model and describe aggregated Internet traffic is still an open area of research. On short timescales up to seconds the traffic is very bursty and on long timescales there are often predictable daily and weekly cycles. In between there can be unpredictable changes and shifts in traffic demand, for instance due to hotspots and flash crowds, or because a link goes down, there are changes in the inter-domain BGP routing, or because traffic in an overlay is re-directed. For future networks more variability in traffic demands is also expected due to mobility of nodes and networks and more dynamic on-demand service level agreements (SLA:s).

The traffic variability means that, even if we could measure the current traffic situation exactly, it would not always correctly predict the near future traffic situation and this needs to be taken into account when doing traffic engineering. Network operators often handle this by relying on simple well-tried techniques (like OSPF and IS-IS routing), over-dimensioning of network capacity, and simple rules of thumb (i.e upgrade the link capacity when mean utilisation reaches 70-80%) rather than introducing complex traffic engineering techniques.

In this paper we take this need for spare capacity and simple rules of thumb as our starting point. We revisit the approach of using weight settings in OSPF/IS-IS for traffic engineering but now with focus on robustness. We propose weight settings that we call *l-balanced* where the operator, by setting the parameter $l$ (to say 80%), control the maximum utilisation level in the network and how much spare capacity is needed to handle unpredictable traffic changes. With an $l$-balanced routing the traffic takes the shortest paths possible but makes sure that no link is utilised to more than a given level $l$, if possible.

The main contributions in this paper are:

- We propose $l$-balanced weight settings in OSPF/IS-IS for robust traffic engineering.
- We present a heuristic search method for finding $l$-balanced weight settings and show that it works well in real network scenarios.
- We evaluate $l$-balanced routing and compare it with other proposed traffic engineering objectives for several real network topologies and traffic data sets.

If traffic levels continue to grow then of course network capacity needs to be added at some point. But traffic engineering with $l$-balanced routing can extend the upgrade cycle and postpone the investment, or be applied to better use the existing resources in the network until the highly utilised links have been upgraded.

The paper is organized as follows. Section II gives a short introduction to traffic engineering in IP networks and Section III discusses related work. We then present the $l$-balanced cost function in Section IV and describe the search heuristic used for finding $l$-balanced weight settings. In Section V we evaluate the proposed methods. We show that the search heuristic works well for finding $l$-balanced weight settings in real traffic scenarios. Further, we compare the robustness of different weight-setting methods and investigate what happens to link utilisations in the network if a traffic demand suddenly

*Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7184*
*http://dx.doi.org/10.4108/ICST.BROADNETS2009.7184*

increases. Finally, in Section VI we make some concluding remarks about our findings.

## II. Traffic Engineering in IP networks

The objective of traffic engineering is to avoid congestion in the network and to make better use of available network resources by adapting the routing to the current traffic situation. The traffic demands in a network changes over time and for network operators it is important to tune the network in order to accommodate more traffic and meet service level agreements (SLAs) made with their customers. This means that a network operator can not rely only on long-term network planning and dimensioning that are done when the network is first built. Robust traffic engineering mechanisms are needed that can adapt to changes in traffic demand and distribute traffic to benefit from available resources.

The first step in the traffic engineering process is to collect the necessary information about network topology and the current traffic situation. Most traffic engineering methods need as input a traffic matrix describing the demand between each pair of nodes in the network. The traffic matrix is then used as input to the routing optimization step, and the optimized parameters are finally used to update the current routing.

Today, the main alternative for intra-domain traffic engineering in IP networks is to use different methods for setting the weights (and so decide upon the shortest paths) in the routing protocols OSPF (Open Shortest Path First) and IS-IS (Intermediate System to Intermediate System). These are both link-state protocols and the routing decisions are based on link costs and a shortest (least-cost) path calculation. With the equal-cost multi-path (ECMP) extension to the routing protocols the traffic can also be distributed over several paths that have the same cost. These routing protocols were designed to be simple and robust rather than to optimize the resource usage. They do not by themselves consider network utilisation and do not always make good use of network resources. The traffic is routed on the shortest path through the network even if the shortest path is overloaded and there exist alternative paths. It is up to the operator to find a set of link costs (weights) that is best suited for the current traffic situation and that avoids congestion in the network.

The general problem of finding the best way to route traffic through a network can be mathematically formulated as a multi-commodity flow (MCF) optimization problem (see, e.g., [1]–[3]). The network is then modeled as a graph. The problem consists of routing the traffic, given by a demand matrix, in the graph with given link capacities while minimizing a cost function. With no limitations on how the traffic flows can be divided over the network links the MCF optimal routing problem can be formulated and efficiently solved as a linear program. Introducing integer weights and ECMP shortest paths constraints, where the traffic no longer can be split arbitrarily, makes the problem computationally much harder. For reasonably sized networks one usually has to rely on search heuristics for determining the set of weights, rather than calculating the optimal weights.

## III. Related work

Traffic engineering by finding a suitable set of weights in OSPF/IS-IS is a well studied area of research and it is described in recent textbooks in the area [3], [4]. When we now revisit the weight setting approach to traffic engineering we are most inspired by the pioneering works by Fortz and Thorup [2], [5] and Ramakrishnan and Rodrigues [6], in that we use a piece-wise linear cost function and search heuristics to find suitable weight settings.

Several studies [2], [7]–[9] have shown that even though we limit the routing of traffic to what can be achieved with weight-based ECMP shortest paths, and not necessarily the optimal weights but those found by search heuristics, it often comes close to the optimal routing for real network scenarios. How the traffic is distributed in the network very much depends on the objectives, usually expressed as a cost function, in the optimisation. An often proposed objective function is described by Fortz and Thorup [2] (and we will refer to it as the FT cost function further on). Here the sum of the cost over all links is considered and a piece-wise linear increasing cost function is applied to the flow on each link. The basic idea is that it should be cheap to use a link with small utilization while using a link that approaches 100% utilisation should be heavily penalized. The *l*-balanced cost function [1], [10] used in this paper is similar in that it uses a piecewise linear cost function to obtain desireable solutions. Additionaly, our cost function gives the operator the opportunity to set the maximum wanted link utilisation. Cost functions for traffic engineering is further investigated by Balon *et.al* [11].

This paper add to existing work on weight settings by focusing on robustness and the objective of achieving a controlled spare capacity for handling unpredictable traffic shifts. For robust traffic engineering much of the focus is on handling multiple traffic matrices and traffic scenarios [5], [12]–[16] and handling the trade-off between optimising for the common case or for the worst case. There are also several works on finding weight settings that are robust to link failures [17]–[19].

Xu *et.al* [20] describe a method to jointly solve the flow optimization and the link-weight approximation using a single formulation resulting in a more efficient computation. Their method can also direct traffic over non-shortest paths with arbitrary percentages. Their results should also be directly applicable to our problem of providing robustness to changes, by just substituting their piece-wise linear cost function with our cost function. In a continuation on that work Xu *et.al* [21] propose a new link-state routing protocol. The protocol splits traffic over multiple paths with an exponential penalty on longer paths and achieves optimal traffic engineering while retaining the simplicity of hop-by-hop forwarding.

## IV. L-balanced solutions

### A. Optimal l-balanced routing

A routing is said to be *l-balanced* if the utilisation is less than or equal to $l$ on every link in the network. For instance

Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7184
http://dx.doi.org/10.4108/ICST.BROADNETS2009.7184

a solution is (0.7)-balanced if it never uses any link to more than 70% of its capacity.

The *l*-balanced cost function, its theoretical foundation, and use in MCF optimisation is described in [1], [10]. The idea is to use a simple piece-wise linear cost function as shown in Figure 1 and apply it to the utilisation of each link in the network. The cost function consists of two linear portions where the slope of the second line segment should be large enough to penalise utilisation above $l$ and balance traffic over longer paths.

The work in [1], [10] present a formula to calculate the cost function, for a given network topology and traffic situation, that guarantees to find a *l*-balanced optimal routing (provided, of course, that such solutions exist) that takes the shortest paths possible and makes sure that no link is utilised to more than $l$.

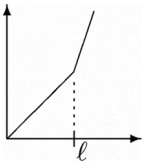


Fig. 1. The link cost function.

### B. Search for l-balanced weight settings

To apply the *l*-balanced routing in real OSPF/IS-IS networks we need to find *l*-balanced weight settings. For weight settings we dont have the guarantee to find an *l*-balanced routing in the same way as described for optimal routing above. But we want to use the *l*-balanced cost function to find weights settings that achieve the same effect of taking the shortest paths possible while routing the traffic so that no link is utilised to more than a given level $l$.

The problem of finding the optimal weight setting is NP-hard [2], [3]; and so the optimal weights are often too computationally hard and time consuming to calculate for real networks and traffic scenarios. Instead we use a problem specific local search heuristic to determine the set of weights. An overview of local search methods can be found in [22]. Our search method can be placed under the Tabu search meta-heuristic in that we allow cost-increasing solutions to direct the search away from local minima, and use a tabu list to prevent from looping back to old solutions. A solution is a vector $w = \{w_1, .., w_n\}$ of weights, with one weight per directed link in the network. We have a solution space $W$ where each weight can take integer values between 1 and 65535. We generate a neighboring solution $i \in N(w)$ by increasing one weight in the current solution $w$ to divert traffic from the most utilised link $(s,t)$ or change weights to create paths with the same cost to get ECMP routing of traffic over several links from $s$. We use a $l$-balanced cost function (as described in the previous section) calculated for the given topology, traffic matrix and required utilisation level $l$. The cost $f(w)$ for a given weight vector is determined by calculating the shortest paths routing with these weights using Dijkstra's algorithm, adding the traffic matrix, and applying the cost function to the resulting link loads. The starting point is to set all weights to the same value, for instance $w_i = 10$. The search terminates either when we find a solution with utilisation under the threshold $l$ or it stops after a fixed number of iterations.

At the core of our search method is a simple descent search [22] where we:

1) choose an initial weight vector $i \in W$
2) find the neighbor $j \in N(i)$ with lowest cost i.e. $f(j) <= f(k)$ for any $k \in N(i)$.
3) If $f(j) >= f(i)$ then stop. Else set $i = j$ and go to step 2.

This type of search may stop at a local minimum. We therefore allow the search to continue by doing new descents starting from weight sets with higher cost. We use information that becomes available during the search to build a candidate list of weight sets that are used as starting points, and a tabu list of weight sets are used to avoid cycling.

We start by setting all weights to the same value. This gives the shortest paths in number of hops which probably is a good starting point for most real networks; if the link capacities are uniform and the network was built with OSPF/IS-IS routing in mind. Given the network topology, traffic matrix and initial weights, we calculate the ECMP shortest paths, add the traffic matrix, and find the most loaded link $(s,t)$ in the network. If the utilisation is less than $l$ then we are done. We have a routing that takes the shortest paths possible and makes sure that no link is utilised to more than the limit $l$. If the link is utilised to more than $l$ we start searching for a better weight setting using two strategies:

- the first search strategy is to increase the weight on the overloaded link in controlled steps so to divert more and more demands (or part of demands) from the link. See details in IV-C.
- the second search strategy is to find weights to get ECMP routing from $s$ for the demands over $(s,t)$, and so balance the traffic over the outgoing links from $s$. See details in IV-D.

In each iteration of a descent we have a number of neighbor weight settings that we evaluate (one for each weight step and ECMP set described above). If a neighbor weight setting gives a lower cost than the current best (in this iteration) it is saved and used as the starting point in the next iteration. If a candidate weight setting gives a routing with a higher cost than the current best but with a different link than $(s,t)$ as most utilised, then that weight-setting is saved in the candidate list and used as a starting point for another descent search later on.

### C. How to determine weight increments for a link?

If a link $(s,t)$ is over-utilised we want to increase the weight on the link in controlled steps so to divert more and more traffic demands from the link.

To decide the steps in which to increase the weight on $(s,t)$ we first determine the current total weight-cost for each

Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7184
http://dx.doi.org/10.4108/ICST.BROADNETS2009.7184

demand routed over $(s,t)$. We then temporarily take away the link $(s,t)$ from our representation of the topology and calculate a new shortest-path routing. For all demands that before were routed over $(s,t)$ we then check how much the weight cost have increased and use this for determining the steps with which to increase the weight on $(s,t)$.

In the example in Figure 2, we assume that the two demands

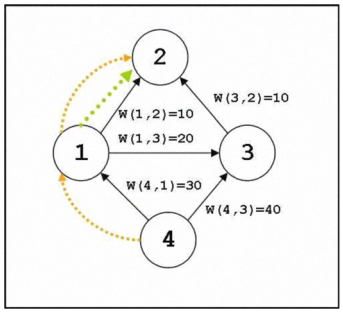


Fig. 2. Example with an overloaded link (1,2) where traffic can be diverted to other paths by increasing the weight on (1,2) in controlled steps w(1,2)= 20, 25, 30 and 31. With the first increment w(1,2)=20 we divert half of demand D(4,2) by ECMP. The next increment w(1,2)=25 diverts all of D(4,2), and with w(1,2)=30 we route also half of D(1,2) on another path. Finally, w(1,2)=31 diverts all traffic from (1,2).

$D(1,2)$ and $D(4,2)$ overload the link $(1,2)$. We thus want to divert traffic from the link $(1,2)$ by increasing the weight $w(1,2)$.

We start by determining the increase steps in which to increase the weight $w(1,2)$:

The total weight costs for $D(1,2)$ and $D(4,2)$ are 10 and 40, respectively. If we take away the link $(1,2)$, we get total weight costs of 30 and 50, an increase by 20 and 10 units respectively. From this we decide on the increase steps 10, 15 (mid-point between 10 and 20), 20 and 21 units. We add this to the original $w(1,2) = 10$ and get the candidate weights $w(1,2)$= 20, 25, 30 and 31 to evaluate.

With the first increment $w(1,2) = 20$ we divert half of demand $D(4,2)$ by ECMP while the other half of $D(4,2)$ and all of demand $D(1,2)$ is still routed on $(1,2)$. The next increment $w(1,2) = 25$ diverts all of $D(4,2)$ but keeps all of $D(1,2)$. With $w(1,2) = 30$ we also route half of $D(1,2)$ on another path and with $w(1,2) = 31$ we divert all traffic from $(1,2)$.

### D. How to determine ECMP weight settings?

If we have a weight set that results in an overloaded link $(s,t)$ then we want to also evaluate neighbor weight settings where we split traffic demands evenly over the outgoing links from $s$ using ECMP. In order to split a traffic demand ECMP the total weight for each path from $s$ to the demand destination $d$ need to be the same.

Consider, as in Figure 3, a node $s$, the next hops $t_i$, and the shortest path $P_i$ from each $t_i$ to the destination $d$. Also consider the corresponding weights $w(s,t_i)$ and total weight cost $w(P_i)$ for a path $P_i$ from $t_i$ to $d$. One way to

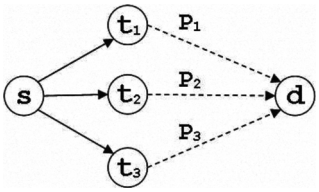


Fig. 3. Determining ECMP weights

achieve ECMP weights is to adjust the weights $w(s,t_i)$ on the outgoing links from $s$ such that:

$$w(s,t_i) = 1 + max_{j=1,..,n}\{w(P_j)\} - w(P_i)$$

This gives the same total cost for each path from s to d.

A possible extension to this is to not always spread the traffic over all possible links but also evaluate different subsets of ECMP weights setting with varying number of outgoing links from $s$.

### E. Increment weight on a less utilised link in a path

With high traffic load in the network, link weights can become sensitive to change after some iterations in the search. For instance if we on an overloaded link already have adjusted the weight to split a large demand with ECMP then we can not easily increase the link weight to divert yet another flow without disturbing the existing load balancing.

In order to divert traffic demands to other paths but without disturbing existing splits on the most utilised link we extend the neighborhood in the search. We evaluate weight sets where we instead of changing the weight on the overloaded link $(s,t)$ increment the link weight some step away closer to the demand destination. In the example in Figure 4, assume that the link

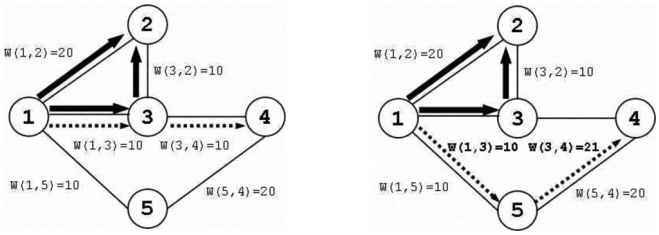


Fig. 4. Example with an overloaded link (1,3). With an extended neighborhood in the search the demand D(1,4) can be diverted by increasing the weight w(3,4) instead of w(1,3), and avoid disturbing the other flows on the overloaded link (1,3).

(1,3) is overloaded. With our search (as described in IV-C) we would in this example evaluate a weight setting where the demand $D(1,4)$ is diverted to the path 1-5-4 by increasing the weight $w(1,3)$ to 21. But increasing the weight $w(1,3)$ will also send all of $D(1,2)$ on the link $(1,2)$, possibly creating overload on that link and a higher cost solution.

With the extended neighborhood we also evaluate alternative weight settings where we increase the weight on other links in the path (not only on the most utilised link). In this example for demand $D(1,4)$ we increase the weight $w(3,4)$ which

*Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7184*
*http://dx.doi.org/10.4108/ICST.BROADNETS2009.7184*

diverts the demand $D(1,4)$ from the overloaded link $(1,3)$ while keeping the needed ECMP split of demand $D(1,2)$.

### F. Comments on the search method

As described above several different techniques are needed to get an efficient search method to find $l$-balanced solutions.

When designing and implementing our search method we were in part inspired by the works of Ramakrishnan and Rodrigues [6] and Fortz and Thorup [2]. From the first we borrowed the idea of temporarily taking away the overloaded link from the representation of the topology, and calculate a new shortest-path routing, to find the weight increments for the link. But apart from this idea our approaches are different. Fortz and Thorup [2] use a Tabu local search heuristic to find appropriate link weights, and from here we also borrowed the idea on how to find ECMP weight-settings over many links.

But, for efficiency, we wanted a more problem-specific search heuristic rather than a generic Tabu search. Instead of searching at random, we start with the shortest paths possible and directly look at the most loaded link. If the utilisation is less than $l$, then we are done and no search is needed. If the link is utilised to more than $l$, then we start to divert traffic from there.

The higher the traffic level the more difficult it is to find a weight setting, that not only balances the traffic, but actually keeps it under a specified level $l$. We combined the existing techniques described above: weight increments and ECMP traffic splits at the most utilised link. But with our direct approach and at high traffic loads, it turned out not to be enough to find $l$-balanced routings.

Therefore, we also added our ideas with candidate lists and extended neighborhoods. For the candidate list, we choose weight settings with a higher cost but where the overload has moved to another link, in order to diverse the search. And for extended neighborhoods, we increment the weight on a less utilised link in a path in order to not disturb the weight composition in sensitive, highly loaded areas.

## V. Evaluation

### A. Method

In order to evaluate the $l$-balanced routing and our search method for finding $l$-balanced weights we use real network topologies and traffic matrix data that we scale up to get high loads in the networks. First in Section V-B we evaluate that the search method works well for finding $l$-balanced weight setting in these scenarios and compare the resulting network loads with optimal $l$-balanced routing and routing with other traffic engineering objectives. The main objective of $l$-balanced routing is to give a controlled amount of spare capacity to handle traffic changes. In Section V-C we investigate how different weight settings handle hotspots where one traffic matrix entry increases.

For the evaluation we here use two different data sets that include network topologies and traffic matrix data from the Geant network [23], and from the American sub-network of a global IP network.

- Network I: the Geant network with 23 nodes, 74 links and 506 demands.
- Network II: the American network with 24 nodes, 110 links and 552 demands.

The details of the global IP-network, the subnetwork topologies and traffic demands, are described in [24]. For the Geant network we set all link capacities to 10 Gb and scaled up the traffic data to create high loads in the network.

### B. Static scenario: Evaluating the search method

The evaluation shows that the $l$-balanced objective and our search method for finding l-balanced weight settings work well. Figure 5 shows comparisons of optimal and weight-based $l$-balanced routing (with $l$=80%) for increasing levels of traffic demand in the Geant network (Network I) and the American network (Network II). The $l$-balanced routing sends the traffic on the shortest paths as long as the utilisation is low in the network. The shape of the curves shows that when we scale up the traffic demand the $l$-balanced method tries to keep the utilisation under $l$=*0.8*. The figures also show that the weight-based routing is close to the optimal routing which validates that our search method for setting the weights works well. Note that optimal routing minimises the total cost when the

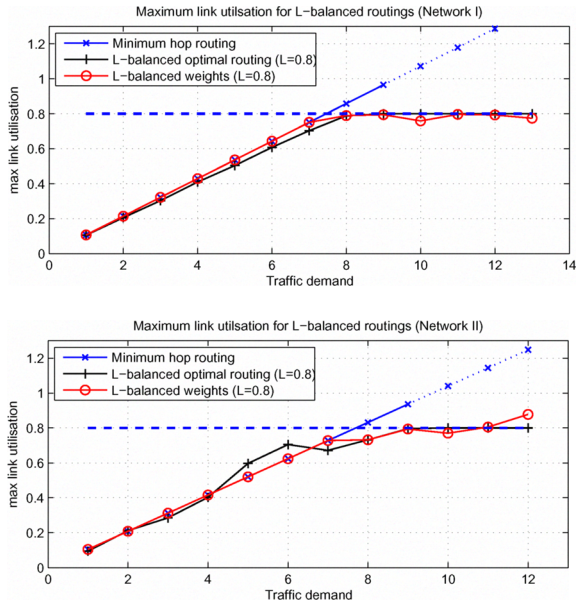


Fig. 5. Comparison of maximum link utilisations for optimal and weight-based L-balanced routing for different scaled traffic demands in the Geant network (top) and the American network (bottom). The utilisation is kept under the chosen limit $l$ and the weights found by the search heuristic gives a routing close to optimal.

$l$-balanced cost function is applied to the utilisation of each link in the network. The utilisation for an individual link (and so the maximum link utilisation) can be higher in the optimal solution if it finds a shorter path that still keeps the utilisation below $l$.

*Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7184*
*http://dx.doi.org/10.4108/ICST.BROADNETS2009.7184*

TABLE I
PERFORMANCE OF THE SEARCH HEURISTIC FOR NETWORK I

| load-level | Min. hop routing links >L | max util | L-balanced routing (L=0.8) time | max util | descents | sets |
|---|---|---|---|---|---|---|
| 7 | 0 | 0.751 | 0.1 s | 0.751 | 0 | 0 |
| 8 | 2 | 0.858 | 0.3 s | 0.784 | 1 | 63 |
| 9 | 3 | 0.965 | 0.2 s | 0.797 | 1 | 51 |
| 10 | 3 | 1.072 | 0.4 s | 0.780 | 1 | 82 |
| 11 | 4 | 1.179 | 0.4 s | 0.795 | 1 | 91 |
| 12 | 5 | 1.287 | 0.4 s | 0.794 | 1 | 123 |
| 13 | 6 | 1.394 | 56.8 s | 0.790 | 262 | 21451 |

TABLE II
PERFORMANCE OF THE SEARCH HEURISTIC FOR NETWORK II

| load-level | Min. hop routing links >L | max util | L-balanced routing (L=0.8) time | max util | descents | sets |
|---|---|---|---|---|---|---|
| 7 | 0 | 0.728 | 0.4 s | 0.728 | 0 | 0 |
| 8 | 1 | 0.832 | 0.4 s | 0.732 | 1 | 41 |
| 9 | 2 | 0.936 | 0.6 s | 0.766 | 1 | 95 |
| 10 | 5 | 1.040 | 2.2 s | 0.732 | 7 | 671 |
| 11 | 5 | 1.144 | 477.5 s | 0.801 | 4390 | 200037 |

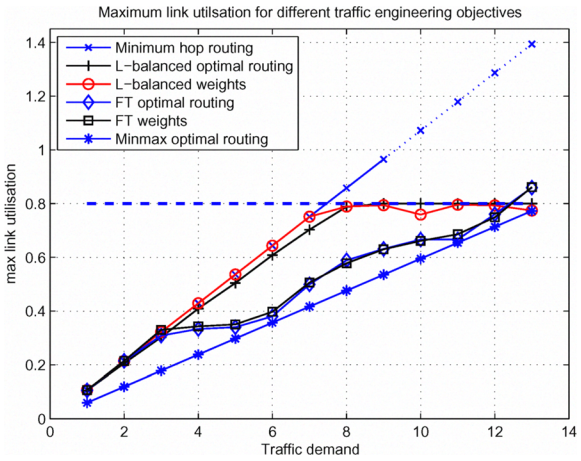


Fig. 6. Comparison of maximum link utilisations for different traffic engineering objectives in the Geant network.

Tables I and II describe the performance of our search method and show that our search heuristic is fast. The left-hand side of the tables describes the load situation in the networks. The increasing load levels (shown in the first column) come from multiplying each entry in the traffic matrix with a higher and higher constant value. For both networks it holds that, up to level 7, no search is needed since the start weights (all set to 10) and the resulting minimum-hop routing give a maximum link utilisation of less than $l = 0.8$.

The second and third columns show the number of links that are loaded to more the level $l = 0.8$ and the maximum link utilisation, when all weights are set to 10. This is the state from which the search start.

The right-hand side of the tables shows the performance of our search method. The first column shows how long time it takes to find an $l$-balanced solution for different levels of network load. The table also shows the number of search descents (number of new starts) and the total number of neighbor weight sets that were evaluated.

As an example, for Network I in Table I, at scale 8 there are two links that are utilized to more than $l = 0.80$. The search heuristic investigate 63 different weight settings to find an $l$-balanced solution with a maximum link utilisation of 0.784. This search took only 0.3 seconds on a standard laptop with a 1.6GHz Intel Core 2 Duo CPU and 2 GB of memory.

At scale 13 there are 6 links utilized to more than $l = 0.80$ and with a maximum utilisation of 1.394. The search needs to find a weight setting that diverts traffic and simultaneously pushes down all six link utilisations under $l = 0.8$ (and without increasing any other link to more than $l$, of course). Our search heuristic evaluates 21451 weight settings and finds an $l$-balanced solution at this level in less than a minute.

Figure 6 shows a comparison between the $l$-balanced routing and other traffic engineering objectives. The minimum-hop routing (with all weights set to 10), where no attempt is done to adapt the weight setting to the current traffic demand, quickly leads to overload in the network when the traffic demands are increased. The $l$-balanced method sends the traffic on the shortest paths as long as the utilisation is less than the chosen value $l$=0.8. With a low utilisation of the network there is no reason to split the traffic over several paths. The FT cost function used in [2], pushes down the maximum link utilisation already at lower traffic levels. This piece-wise linear cost function consists of several segments which is reflected in the shape of the curve with plateaus where the maximum link utilisation is pushed down. With minmax routing the objective is to minimise the maximum link utilisation in the network. This routing always balance the load over the network to keep the highest link utilisation down to a minimum. The optimal minmax routing gives a lower bound on how much it is possible to keep down the maximum link utilisation.

### C. Dynamic scenario: Evaluation of robustness

The main purpose with $l$-balanced routing is to give a controlled traffic level and spare capacity to handle uncertainties and sudden changes in the traffic situation. To confirm that the $l$-balanced weight settings fulfil this, we added hotspot traffic (in a magnitude that the $l$-balanced routing should be able to handle) and investigated the resulting link utilisations. Figure 7 shows the maximum link utilisations for minimum hop routing, $l$-balanced and FT weight-settings under assumed hotspot traffic in the Geant network scenario. After determining the weights and the routing for a given traffic matrix each of the 506 demands was increased one at a time by 20% of the link capacity.

The minimum hop routing, without any traffic engineering, gives link overload for all hotspot traffic at this demand level. The FT routing sometimes results in overloaded links when the hotspot traffic is added. The $l$-balanced routing (with $l$=0.8) on the other hand gives 20% spare capacity and so handle the increase for any of the demands.

*Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7184*
*http://dx.doi.org/10.4108/ICST.BROADNETS2009.7184*

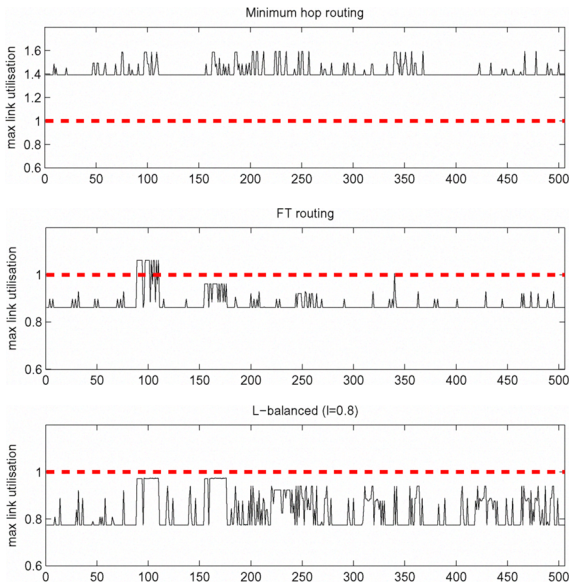


Fig. 7. Hotspot traffic scenario in the Geant network. Comparison of maximum link utilisations for three weight setting strategies. Minimum hop routing and FT routing exceeds the link capacity while $l$-balanced routing can avoid overload.

## VI. Conclusions

In this paper we propose $l$-balanced routing with OSPF/IS-IS for robust traffic engineering. We present a heuristic search method for finding $l$-balanced weight settings and show that the search and the resulting weight settings work well in real network scenarios.

L-balanced weight settings give the operator possibility to apply simple rules of thumb for controlling the maximum link utilisation and control the amount of spare capacity needed to handle sudden traffic variations. It gives more controlled traffic levels than other cost functions and more efficient routing for low traffic loads when there is no need to spread traffic over longer paths.

Our local search method can be placed under the Tabu search meta-heuristic in that we allow cost-increasing solutions to direct the search away from local minima, and use a tabu list to prevent from looping back to old solutions. But for efficiency, rather than using a generic Tabu search, we implement a search heuristic specific for the problem of finding $l$-balanced weight settings. We start with minimum-hop routing and investigate the most loaded link. If the utilisation is less than $l$, then we are done and no search is needed. If the link is utilised to more than $l$, then we start the search from there, and we use several different weight strategies for diverting traffic to other paths.

The higher the traffic level the more difficult it is to find a weight setting, that not only balances the traffic, but actually keeps it under a specified level $l$. We combine controlled weight increments and ECMP traffic splits to divert traffic from the most utilised link. We also introduce candidate lists and extended neighborhoods. Promising weight settings that move the overload to other links are saved in the candidate list to be starting points for further search. Extended neighborhoods means that, when diverting a traffic flow from an overloaded link, we do not only try to increase the weight on the overloaded link. We also evaluate weight settings where we increment the weight on a less utilised link further down the path. This is done in order to not disturb already achieved traffic splits in highly loaded areas.

We evaluate our search heuristic in several real network scenarios and show that the search is fast and that it finds $l$-balanced weight-settings in seconds or minutes depending on the traffic level.

## Acknowledgment

The authors would like to thank Anders Gunnar for providing the traffic data used in the evaluation. We also would like to thank Adam Dunkels and Bengt Ahlgren for inspiring and helpful discussions.

## References

[1] H. Abrahamsson, J. Alonso, B. Ahlgren, A. Andersson, and P. Kreuger, "A multi path routing algorithm for IP networks based on flow optimisation," in *Proceedings of the Third International Workshop on Quality of Future Internet Services (QoFIS)*, Zürich, Switzerland, October 2002.

[2] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proceedings IEEE INFOCOM 2000*, Israel, March 2000, pp. 519–528. [Online]. Available: http://www.ieee-infocom.org/2000/program.html

[3] M. Pióro and D. Medhi, *Routing, Flow, and Capacity Design in Commmunication and Computer Networks*. Morgan Kaufmann, 2004.

[4] J. Rexford, "Route optimization in IP networks," in *Handbook of Optimization in Telecommunications*, M. G. Resende and P. M. Pardalos, Eds. Springer Science+Business Media, 2006.

[5] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 4, pp. 756–767, May 2002.

[6] K. Ramakrishnan and M. Rodrigues, "Optimal routing in shortest path data networks," *Lucent Bell Labs Technical Journal*, vol. 6, no. 1, 2001.

[7] A. Gunnar, H. Abrahamsson, and M. Söderqvist, "Performance of Traffic Engineering in Operational IP-Networks: An Experimental Study," in *Proceedings of 5th IEEE International Workshop on IP Operations and Management*, Barcelona, Spain, October 2005.

[8] A. Sridharan, R. Guérin, and C. Diot, "Achieving Near-Optimal Traffic Engineering Solutions for Current OSPF/IS-IS Networks," in *IEEE Infocom*, San Francisco, March 2003. [Online]. Available: http://ipmon.sprintlabs.com

[9] D. Applegate and E. Cohen, "Making Intra-Domain Routing Robust to Changing and Uncertain Traffic Demands: Understanding Fundamental Tradeoffs," in *Proceedings of ACM SIGCOMM*, Karlsruhe, Germany, August 2003.

[10] J. Alonso, H. Abrahamsson, B. Ahlgren, A. Andersson, and P. Kreuger, "Objective functions for balance in traffic engineering," SICS – Swedish Institute of Computer Science, Tech. Rep. T2002:05, May 2002.

[11] S. Balon, F. Skivee, and G. Leduc, "How Well Do Traffic Engineering Objective Functions Meet TE Requirements?" in *Proceedings of IFIP International Networking Conference*, Coimbra, Portugal, May 2006.

[12] C. Zhang, Z. Ge, J. Kurose, Y. Liu, and D. Townsley, "Optimal Routing with Multiple Traffic Matrices: Tradeoffs between Average Case and Worst Case Performance," in *Proceedings of ICNP 2005*, Boston, USA, November 2005.

[13] C. Zhang, Y. Liu, W. Gong, J. Kurose, R. Moll, and D. Townsley, "On Optimal Routing with Multiple Traffic Matrices ," in *Proceedings of Infocom 2005*, Miami, USA, March 2005.

[14] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, "Cope: Traffic engineering in dynamic networks," in *Proceedings of ACM SIGCOMM*, Pisa, Italy, September 2006.

*Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7184*
*http://dx.doi.org/10.4108/ICST.BROADNETS2009.7184*

[15] A. Gunnar and M. Johansson, "Robust routing under bgp reroutes," in *Proceedings of Globecom 2007*, Washington, DC, USA, 2007.
[16] M. Menth, R. Martin, and J. Charzinski, "Capacity Overprovisioning for Networks with Resilience Requirements," in *Proceedings of the ACM SIGCOMM'06*, Pisa, Italy, September 2006.
[17] A. Nucci, S. Bhattacharyya, N. Taft, and C. Diot, "IGP Link Weight Assignment for Operational Tier-1 Backbones," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, August 2007.
[18] M. Menth, M. Hartmann, and R. Martin, "Robust IP Link Costs for Multilayer Resilience," in *Proceedings of the 6th International IFIP Networking Conference*, Atlanta, Georgia, USA, May 2007.
[19] A. Sridharan and R. Guérin, "Making IGP Routing Robust to Link Failures," in *Proceedings of the 4th International IFIP Networking Conference*, Waterloo, Ontario, Canada, May 2005.
[20] D. Xu, M. Chiang, and J. Rexford, "DEFT: Distributed exponentially-weighted flow splitting," in *IEEE Infocom*, Anchorage, Alaska, USA, May 6–12, 2007.
[21] ——, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," in *IEEE Infocom*, Phoenix, Arizona, USA, April 2008.
[22] E. Aarts and J. K. Lenstra, Eds., *Local search in combinatorial optimization*. Princeton University Press, 2003.
[23] *The Geant network*, http://www.geant.net.
[24] A. Gunnar, M. Johansson, and T. Telkamp, "Traffic Matrix Estimation on a Large IP Backbone - a Comparison on Real Data," in *Proceedings of ACM Internet Measurement Conference*, Taormina, Sicily, Italy, October 2004.

*Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7184*
*http://dx.doi.org/10.4108/ICST.BROADNETS2009.7184*