Achieving Near-Optimal Traffic Engineering Solutions for Current OSPF/IS-IS Networks

Ashwin Sridharan[†], *Roch Guérin*[†], *Christophe Diot*[††]

Abstract— **Traffic engineering is aimed at distributing traffic so as to "optimize" a given performance criterion. The ability to carry out such an optimal distribution depends on both the routing protocol and the forwarding mechanisms in use in the network. In IP networks running the OSPF or IS-IS protocols, routing is over shortest paths, and forwarding mechanisms are constrained to distributing traffic uniformly over equal cost shortest paths. These constraints often make achieving an optimal distribution of traffic impossible. In this paper, we propose and evaluate an approach, based on manipulating the set of next hops for routing prefixes, that is capable of realizing near optimal traffic distribution without any change to existing routing protocols and forwarding mechanisms. In addition, we explore the trade-off that exists between performance and the overhead associated with the additional configuration steps that our solution requires. The paper's contributions are in formulating and evaluating an approach to traffic engineering for existing IP networks that achieves performance levels comparable to that offered when deploying other forwarding technologies such as MPLS.**

Index Terms— **Routing, Networks, Traffic Engineering, Aggregation.**

I. Introduction

As the amount and criticality of data being carried on IP networks grows, it is becoming increasingly important to manage network resources in order to ensure reliable and acceptable performance. Furthermore, it is desirable to accomplish this while minimizing or deferring costly upgrades. One of the techniques that is being evaluated by many Internet Service Providers to achieve this goal is traffic engineering. Traffic engineering aims at using information about the traffic entering and leaving the network to optimize network performance. Most often the output of traffic engineering is an "optimal" set of paths and link loads that produce the best possible performance given the available resources. This set of paths can then be used by network administrators within an autonomous system to control the flow and distribution of traffic across the network. However, explicitly setting up such paths and (optimally) assigning traffic to them, typically calls for changes to both the routing protocols and the forwarding mechanism they rely on, e.g., through the introduction of new technology such as MPLS [1].

Currently, two of the most widely used Interior Gateway routing protocols are OSPF [2] and IS-IS [3]. Hence it would be beneficial to devise solutions that allow these protocols to emulate "optimal routing," thus leveraging their widespread deployment. There are two main difficulties in doing so. The first is that these protocols use shortest path routing with destination based forwarding. The second is that when the protocols generate multiple equal cost paths for a given destination routing prefix, the underlying forwarding mechanism performs load balancing across those paths by equally splitting traffic on the corresponding set of next hops . These added constraints make it difficult or impossible to achieve optimal traffic engineering link loads. One of the first works to explore this issue was [4], where a local search heuristic was proposed for optimizing OSPF weights assuming knowledge about the traffic matrix. [4] showed that in spite of these constraints, properly selecting OSPF weights could yield significant performance improvements. However, the paper also showed that for some topologies, performance can still be substantially different from the optimal solution. Subsequently, a result from linear programming ([5][Chap. 17, Sec. 17.2]) was used in [6] to prove that *any* set of routes can be converted into a set of shortest paths based on some link weights, that matches or improves upon the performance of the original set of routes. This establishes that the shortest path limitation is in itself not a major hurdle. However, the result of [6] assumes forwarding decisions that are specific to each *source-destination* pair, and more importantly, the ability to split traffic in an arbitrary ratio over different shortest paths. Both of these assumptions are at odds with current IP forwarding mechanisms.

In this paper, we propose an approach that remedies both of these problems. It builds on [6] by taking advantage of the fact that shortest paths can be used to achieve optimal link loads, but it is compatible with both destination based forwarding and even splitting of traffic over equal cost paths. Compatibility with destination based forwarding can be achieved through a very minor extension to the result of [6], simply by taking advantage of a property of shortest paths and readjusting traffic splitting ratios accordingly. Accommodating the constraint of even splitting of traffic across multiple shortest paths is a more challenging task. The solution we propose leverages the fact that current day routers have thousands of route entries (destination routing prefixes) in their routing table. Instead of changing the forwarding mechanism responsible for distributing traffic across equal cost paths, we plan to control the actual (sub)set of shortest paths (next hops) assigned to routing prefix entries in the forwarding table(s) of a router. *In other words, for each prefix we define a set of allowable next hops, by carefully selecting this subset from the set of next hops corresponding to the shortest paths computed by*

[†] {ashwin,guerin}@ee.upenn.edu, Dept. of Elec. Eng., Univ. of Pennsylvania, Philadelphia, PA 19104
[††] cdiot@sprintlabs.com, Sprint Advanced Technology Labs, One Adrian Court,Burlingame, CA, 94010

The work of A. Sridharan and R. Guérin has been supported in part by NSF grant ANI-9902943

the routing algorithm. This allows us to control how traffic is distributed *without* modifying existing routing protocols such as OSPF or IS-IS, and without requiring changes to the data path of current routers, i.e., their forwarding mechanism. It does require some changes to the control path of routers in order to allow the selective installment of next hops in the forwarding table.

Our initial focus is on gaining a better understanding of how well the selective installment of next hops for different routing prefixes can approximate an optimal traffic allocation (set of link loads). Because the problem is NP-hard, we present a heuristic with a provable performance bound as well as two other heuristics which perform very well in our set of experiments. Even though we study these heuristics in the context of a routing problem, we believe that they are generic enough to be of potential use in other load balancing scenarios. The main finding from our investigation is that the performance achieved by this approach is essentially indistinguishable from the optimal. This being said, an obvious drawback of "hand-crafting" the set of next hops that are to be installed for each routing prefix in a router's forwarding table, is the configuration overhead it introduces. The potential magnitude (proportional to the size of the routing/forwarding table) of this overhead could make this approach impractical. As a result, our next step is to investigate a solution that can help mitigate this overhead, albeit at the cost of a possible degradation in performance. Specifically, we limit the number of routing prefixes for which we perform the proposed selective installment of next hops. Our results indicate that a significant reduction in configuration overhead can be achieved without a major loss of performance.

The rest of the paper is structured as follows. Section II introduces the linear program formulation used in [6] to generate an "optimal" set of shortest paths, and introduces our proposed modifications to make it compatible with existing IP routers. Section III presents a set of heuristics for approximating an optimal traffic distribution by manipulating the set of next hops assigned to each routing prefix. A performance bound is derived for one of the heuristics (see Appendix). Section IV presents several experiments that first establish the efficacy of the heuristics of Section III, and then explore the impact on performance of lowering configuration overhead. Section V provides a brief summary of the paper's contributions and outline directions for future work.

II. From Optimal Routing to Shortest Path Routing

In this section, we first briefly review the classic result from linear programming [5][Chap. 17, Sec. 17.2] that was cast in the context of routing in communication networks in [6] to show how optimal routing can be achieved using only shortest paths. We then discuss why this result is not directly usable in current IP networks, and finally propose solutions that allow us to implement the result under the existing paradigm.

The network is modeled as a directed graph $G = (V, E)$ with $m = \| V \|$ nodes and $n = \| E \|$ directed links. We assume the existence of a traffic matrix T where entry $T(s_r, t_r) = d_r$ denotes the average intensity of traffic from ingress node s to egress node t for commodity $r \in \mathcal{R}$. A good reference on how to construct such a traffic matrix can be found in [7]. Assume that an optimal allocation based on some network wide cost function yields a set of paths $\mathcal{P}_r$ for each commodity r, so that the total bandwidth consumed by these paths[1] on link (i, j) is $\tilde{c}_{ij}$. It can be shown that the same performance, in terms of the bandwidth consumed on each link, can be achieved with a set of shortest paths by formulating and solving a linear program and its dual. The linear program can be formulated as ([6])

$$\begin{aligned}
&\min \sum_{(i,j)\in E} \sum_{r\in\mathcal{R}} d_r X_{ij}^r \\
&\text{subject to} \\
&\quad \sum_{j:(j,i)\in E} X_{ij}^r - \sum_{j:(i,j)\in E} X_{ij}^r = 0\,, i \neq s_r, t_r\, r \in \mathcal{R} \\
&\quad \sum_{j:(j,i)\in E} X_{ji}^r - \sum_{j:(i,j)\in E} X_{ij}^r = 1\,, i = t_r\, r \in \mathcal{R} \\
&\quad \sum_{r\in R} d_r X_{ij}^r \leq \tilde{c}_{ij} \quad (i,j) \in E \\
&\quad 0 \leq X_{i,j}^r \leq 1 (i,j) \in E, \quad r \in \mathcal{R}, \qquad (1)
\end{aligned}$$

where X_{ij}^r is the fraction of traffic for commodity r that flows through link (i, j). Solving the linear program gives a traffic allocation $\{\tilde{X}_{i,j}^r\}$ that consumes no more than $\tilde{c}_{i,j}$ amount of bandwidth on any link (i, j). In order to obtain link weights for shortest path routing, the dual of the linear program as formulated in [6] needs to be solved:

$$\begin{aligned}
&\max \sum_{r\in R,\, t\in V} U_{t_r}^r - \sum_{(i,j)\in E} \tilde{c_{ij}} W_{ij} \\
&\text{subject to} \\
&\quad U_j^r - U_i^r \leq W_{ij} + 1 \quad \forall\, r \in \mathcal{R},\, (i,j) \in E \\
&\quad W_{ij} \geq 0 \\
&\quad U_{s_r}^r = 0. \qquad (2)
\end{aligned}$$

The dual gives a set of link weights $\{\tilde{W}_{i,j}\}$, from which a set of shortest paths can be constructed that are consistent with the traffic allocation variables $\{\tilde{X}_{i,j}^r\}$.

It is, however, important to understand that although routing can now be done over shortest paths, this is still quite different from the *forwarding* paradigm currently deployed in existing OSPF and IS-IS networks. The reasons are two-fold and both can be traced to the output of the primal LP, namely, the traffic allocation $\{\tilde{X}_{i,j}^r\}$. Firstly, observe that the traffic allocation is for each *commodity* or source-destination *pair*. This means that the routing protocol could possibly generate different set of next hops for each source-destination pair *on which traffic is to be forwarded.* This impacts the forwarding mechanism on the data path, as it now needs to make decisions on the basis of both source and destination addresses.

The second problem relates to the fact that current forwarding mechanisms support only equal splitting of traffic on the set of equal cost next hops. The linear program yields a

[1]The bandwidth consumed on a link is assumed to be the sum of the traffic on all paths that use the link

traffic allocation that is not guaranteed to obey this constraint. Modifying the forwarding engine to support unequal splitting (see, for example, [8]) of traffic would involve significant and expensive changes. The function used to select the next hop on which to send a packet would have to be modified, and additional information stored in the forwarding entries in order to achieve the desired split ratios. This change is all the more difficult since it impacts the data path. In the next two sub-sections we suggest methods that overcome both these problems.

A. Destination Based Aggregation of Traffic

The first problem of translating a traffic allocation that distinguishes between source-destination *pairs* into one that only depends on destinations is relatively straightforward. It can be achieved simply by transforming the individual splitting ratios of source-destination pairs that share a common destination into a possibly different splitting ratio for the aggregate traffic associated with the common destination. The reason this is possible is because all routes are shortest paths. Shortest paths have the property that segments of shortest paths are also shortest paths, so that once two flows headed to the same destination meet at a common node they will subsequently follow the same set of shortest paths. This means that we need not distinguish these packets based on their source address, and can make splitting and forwarding decisions simply based on their destination address. The new splitting ratios that are to be used on the aggregate traffic in order to achieve the same traffic distribution can be computed as follows.

Let the traffic heading for destination t at node $i \neq t$ be denoted as

$$f_i^t = \sum_{j:(j,i)\in E} \sum_{s:(s,t)\in \mathcal{R}} d_r X_{i,j}^r.$$

The fraction of destination t traffic that is forwarded on link (i,j) is then

$$\alpha_{ij}^t = \frac{\sum_{s:(s,t)\in \mathcal{R}} d_r X_{i,j}^r}{f_i^t}.$$

It can be readily seen that using α_{ij}^t as the fraction of the overall traffic headed toward destination t and sent on link (i,j) will maintain the optimal traffic profile.

B. Approximating Unequal Split of Traffic

In the previous sub-section, we saw that solving the problem of source-destination based forwarding decisions was relatively straightforward. Unfortunately, the same does not hold for the uneven splitting issue, and as mentioned earlier providing such a capability is a significant departure from current operations. Our proposal to overcome this problem is to take advantage of the fact that today's routing tables are relatively large, with multiple routing prefixes associated with the same egress router. By controlling the (sub)set of next hops that each routing prefix is allowed to use, we can control the traffic headed toward a particular egress router(destination). In other words, instead of the current operation that has all routing prefixes use the full set of next hops, we propose to selectively control this choice based on the amount of traffic associated with each routing prefix and the desired link loads for an optimal traffic allocation.

The following example illustrates the idea behind the approach. Assume that at some node, there are four routing prefixes, r_1, r_2, r_3 and r_4 that map to a common destination d and have traffic intensities $t(r_1) = 2, t(r_2) = 5, t(r_3) = 8$ and $t(r_4) = 4$. Let there be three shortest paths associated with destination d, so that the routing table has 3 possible next hops to d, and assume that the *optimal* distribution of traffic to the three next hops is $f_1 = 6, f_2 = 4$ and $f_3 = 9$. We can then intuitively match this traffic distribution by the following next hop assignment:

$$\begin{aligned} r_1 &\rightarrow \text{Hops 1, 3} \\ r_2 &\rightarrow \text{Hop 1} \\ r_3 &\rightarrow \text{Hop 3} \\ r_4 &\rightarrow \text{Hop 2} \end{aligned}$$

The resulting traffic distribution is $f'_1 = (2/2 + 5/1) = 6, f'_2 = (4/1) = 4, f'_3 = (2/2 + 8/1) = 9$, which matches the optimal allocation.

The advantage of the above approach is that the forwarding mechanism on the data path remains unchanged, as packets are still distributed evenly over the set of next hops assigned to a routing prefix. This means that a close approximation of an optimal traffic engineering solution might be feasible even in the context of existing routing and forwarding technologies. There are, however, a number of challenges that first need to be addressed. The first is the need for traffic information at the granularity of a routing prefix entry instead of a destination (egress router). This in itself is not an insurmountable task as most of the techniques currently used to gather traffic data, e.g., router mechanisms' like Cisco's Netflow or Juniper's cflowd, can be readily adapted to yield information at the granularity of a routing prefix.

The second issue concerns the configuration overhead involved in communicating to each router the subset of next hops to be used for each routing prefix,. This can clearly represent a substantial amount of configuration data, as routing tables are large and the information that needs to be conveyed is typically different for each router. The approach we propose and study is to identify a small set of prefixes for which careful allocation of next hops is done and rely on default behavior for the remaining prefixes. The trade-off will then be in terms of how close one can get to an optimal traffic distribution, while configuring the smallest possible number of routing prefixes. We investigate this trade-off in Section IV, where we find that near optimum performance can often be achieved by configuring only a small number of routes.

The third and last challenge is to actually formulate a method for determining which subset of next hops to choose for each routing prefix in order to approximate an optimal allocation. The goal of any solution will be to minimize some metric that measures discrepancy between the optimal traffic allocation and the one achieved under equal-splitting constraints on any hop. We explored two metrics: the maximum gap between the optimal traffic and the allocated traffic on any hop, and the maximum *load* on any hop, where the *load* on a hop is the ratio of the allocated traffic and the optimal traffic. We note that this allocation problem is a generalized version of scheduling unsplittable tasks on a set of processors with speed-up, and hence is NP-hard [9]. In the next section, we propose some simple heuristics that are both fast and still give reasonably good performance.

III. Heuristics for Traffic Splitting

Ideally, one should consider the problem of selective next-hop allocation at the global level, that is, do a *concurrent* optimal assignment of next hops for *each* routing prefix at *each* node. However, since even the single node allocation problem is computationally difficult, we propose heuristics that perform independent computations for each routing prefix at each node. These computations are based only on the incoming traffic at the node and the desired outgoing traffic profile. A potential problem with this approach is that the traffic arriving at a node may not match the optimal profile due to the heuristic decisions at some upstream node. Consequently, the profile of the outgoing traffic from the node in question, could further deviate from the desired one. However, as we shall see, the heuristics perform excellently and hence incoming traffic seen at any node and the resultant outgoing traffic have a near-optimal profile. We propose three heuristics that are greedy in nature and try to minimize one of the two metrics mentioned in Section II-B.

In the following discussion, we focus on a given egress point, and use the words "stream" and "traffic intensity of a routing prefix associated with the egress point" interchangeably. The three heuristics we propose work broadly in the following fashion. When performing computation at an arbitrary node

1) Order routing prefixes destined to a particular egress router in decreasing order of traffic intensity,
2) Sequentially assign each routing prefix to a subset of next hops so as to minimize a given metric.

For clarity we use the following notation in our subsequent discussion:
At an arbitrary node n, when assigning routing prefixes associated with an arbitrary egress router (destination) m to next hops:

1) Denote the set of next hops to egress router m by $\mathcal{K} = \{1, 2, \ldots, K\}, \quad \| \mathcal{K} \| = K$.
2) Denote the desired (optimal) traffic load (for egress router m) on hop $k \in \mathcal{K}$ by f_k.
3) Denote the traffic intensity of routing prefix i by x_i. Denote the collective set of the routing prefixes (at n for m) that need to be assigned to next hops by $\mathcal{X}_{n,m}$.
4) Denote the traffic load on hop k after heuristic H has assigned i routing prefixes by l_k^i. Assume $l_k^i = 0 \quad \text{for} \quad i \leq 0$.

We now describe the three heuristics we investigate in the paper.

<u>MAX-MIN RESIDUAL CAPACITY</u> : This heuristic tries to assign each routing prefix so that the minimum gap between the optimal and desired traffic on any hop is maximized. Although this may seem to be at odds with the goal of matching the optimal profile, the intuition behind such an assignment is to always keep enough residual capacity (difference between optimal and assigned traffic) so as to be able to accommodate subsequent routing prefixes. Since all routing prefixes *must* be allocated a set of next hops, by keeping enough residual capacity we try to ensure that an allocation does not "overflow".

Algorithm MAX-MIN RESIDUAL-CAPACITY:

1) Sort the set of prefixes $\mathcal{X}_{n,m}$ in descending order of traffic intensity.
2) For each prefix $i \in \mathcal{X}_{n,m}$ choose a subset of next hops $\tilde{\mathcal{M}} \in \mathcal{K}$, with cardinality $\| \tilde{\mathcal{M}} \|$ which maximizes

$$\min_{k \in \mathcal{K}} (f_k - l_k^{i-1} + \frac{x_i}{\| \tilde{\mathcal{M}} \|}) \quad .$$

Note that Step 2) can be easily achieved by simply sorting all the next hops in decreasing order of their residual capacity $f_k - l_k^{i-1}$, indexing them in that order, going through an increasing sequence of $\mathcal{M} = \{d \leq k : d \in \mathcal{K}\}, \quad k = 1, 2, \ldots, K$ assignments over $\| \mathcal{M} \|$ hops and choosing the best one, ie., one with maximum min gap.

<u>MIN-MAX GAP</u> : This heuristic tries to assign each routing prefix so as to minimize the maximum gap between the optimal and desired traffic on any hop. Observe that even though, the metric used by this heuristic is the opposite of that used by heuristic MAX-MIN RESIDUAL CAPACITY, both essentially try to achieve the same goal. This is because both heuristics must obey the conservation constraint of assigning all routing prefixes.

Algorithm MIN-MAX GAP:

1) Sort the set of prefixes $\mathcal{X}_{n,m}$ in descending order of traffic intensity.
2) For each prefix $i \in \mathcal{X}_{n,m}$, choose a subset of next hops $\tilde{\mathcal{M}} \in \mathcal{K}$, with cardinality $\| \tilde{\mathcal{M}} \|$ which minimizes

$$\max_{k \in \mathcal{K}} (f_k - l_k^{i-1} + \frac{x_i}{\| \tilde{\mathcal{M}} \|}) \quad .$$

Again, note that this step can be executed in the same fashion as for MAX-MIN RESIDUAL CAPACITY.

<u>MIN-MAX LOAD</u> :The Min-Max Load heuristic is similar to a work conserving scheduling algorithm which tries to minimize the maximum load on any processor (the maximum makespan). Heuristic MIN-MAX LOAD tries to minimize the maximum ratio of assigned traffic to the optimal traffic load over all hops. The difference now is that each task (stream) can be split equally among multiple processors (next hops) and the processors (next hops) can have different speeds (optimal traffic loads).

Algorithm MIN-MAX LOAD:

1) Sort the set of prefixes $\mathcal{X}_{n,m}$ in descending order of traffic intensity.

2) For each prefix $i \in \mathcal{X}_{n,m}$ choose a subset of next hops $\tilde{\mathcal{M}} \in \mathcal{K}$, with cardinality $\parallel \tilde{\mathcal{M}} \parallel$ which minimizes

$$\max_{k \in \mathcal{K}} (\frac{l_k^{i-1} + \frac{x_i}{\|\mathcal{M}\|}}{f_k})$$

Step 2) can be achieved in two stages. First, for each index $p = 1, 2, \ldots, K$, do a *virtual* assignment of routing prefix i to a set of p hops which yields the smallest maximum. This can be done by simply sorting the set $\{\frac{l_k^{i-1} + x_i/p}{f_k}\}, \quad k \in \mathcal{K}$ in increasing order, re indexing them and *virtually* assigning i only to the first p hops.

Second, from all the K such possible assignments, choose the one with the smallest maximum for an *actual* assignment. In case of a tie, choose a lexicographically smaller assignment.

We outline our implementation of the heuristics in the pseudo-code below :

```
procedure **Selective Hop Allocation**
Input ← (*Link Weights* $\{\tilde{W}_{ij}\}$, *optimal traffic allocation* $\{\tilde{f}^t_{ij}\}$, *Traffic Matrix* $T$ )
For each *destination node* $m$ do
  Run Dijkstra's algorithm with weights $\{\tilde{W}_{ij}\}$
  For each *node* $n \neq m$ *in order of decreasing distance from* $m$ do
    Apply the heuristic to the set of routing prefixes $\mathcal{X}_{n,m}$ to determine, for each routing prefix $i$, the set of next hops $\mathcal{K}_i$
    For each  routing prefix $i \in \mathcal{X}_{n,m}$ do
      Update the intensity of the corresponding routing prefix at each node $j \in \mathcal{K}_i$
    done
  done
done
```

In the appendix, we analyze the MIN-MAX LOAD heuristic and show that the load ratio achieved is within a factor of $(1 + \ln K/2)$ of the ratio achieved by an optimum allocation (under the equal splitting constraint). We have as yet not been able to provide tight bounds for the other two heuristics, although as one will see in Section IV, all three heuristics appear to give very similar results.

IV. Experiments

In order to evaluate the effectiveness of our approach, we conducted two sets of experiments on artificially generated topologies as well as on an actual ISP topology, namely the Sprint Backbone [2]. In the first set we studied the performance of our heuristics when compared against optimal routing. In the second set of experiments we studied the trade-off between performance and configuration overhead by varying the number of routing prefixes for which we controlled the set of next hops they were assigned.

For purposes of comparison, we solved a linear multi-commodity flow routing problem with the same piecewise linear cost function used in [4]. The only constraint in the routing problem is flow conservation and consequently it provides a lower bound on the performance of any routing scheme, for the same metric. Hence forth, we shall refer to this problem as the "*optimal routing problem*" and its solution as the "*optimal routing solution*". The solution to this problem is a set of paths (traffic allocation) for each commodity which yields $\tilde{c}_{ij}$, the bandwidth consumed on each link.

We reproduce the optimal allocation problem with regard to this cost function below for completeness. Let the flow of commodity r on link (i,j) be denoted by $y^r_{i,j}$. Let the total flow on link (i,j) is $f_{i,j} = \sum_{r \in \mathcal{R}} y^r_{i,j}$ and the capacity is $C_{i,j}$. Denote the cost of link (i,j) by $\Phi_{i,j}(f_{i,j}, C_{i,j})$, which is a piecewise linear function that approximates an exponentially growing curve (the exact pieces of the function are presented in Equation 4 - 9). The cost grows as the traffic on the link increases and the rate of growth accelerates with increasing utilzation. Evolution of the cost function with link load is shown in Figure 1. The problem may then be formulated as:

$$\min \sum_{(i,j) \in E} \Phi_{i,j}(f_{i,j}, C_{i,j})$$

subject to

$$\sum_{j:(j,i) \in E} y^r_{j,i} - \sum_{j:(i,j) \in E} y^r_{i,j} = \begin{array}{ll} d_r & \text{if } i = s_r \\ -d_r & \text{if } i = t_r \\ 0 & \text{otherwise} \end{array} \quad (3)$$

$$\forall\, i \in V,\, r \in \mathcal{R}$$

$$f_{i,j} = \sum_{r \in R} y^r_{i,j}$$

$$u_{i,j} = f_{i,j}/C_{i,j}, \quad \forall (i,j) \in \mathcal{E}$$

$$\Phi_{i,j} = f_{i,j}, \;\; u_{i,j} \le 1/3 \quad (4)$$

$$\Phi_{i,j} = 3f_{i,j} - \frac{2}{3}C_{i,j}, \;\; 1/3 \le u_{i,j} \le 2/3 \quad (5)$$

$$\Phi_{i,j} = 10f_{i,j} - \frac{16}{3}C_{i,j}, \;\; 2/3 \le u_{i,j} \le 9/10 \quad (6)$$

$$\Phi_{i,j} = 70f_{i,j} - \frac{178}{3}C_{i,j}, \;\; 9/10 \le u_{i,j} \le 1 \quad (7)$$

$$\Phi_{i,j} = 500f_{i,j} - \frac{1468}{3}C_{i,j}, \;\; 1 \le u_{i,j} \le 11/10 \quad (8)$$

$$\Phi_{i,j} = 5000f_{i,j} - \frac{19468}{3}C_{i,j}, \;\; 11/10 \le u_{i,j} \quad (9)$$

$$(10)$$

Equation 3 imposes flow conservation constraints, and Equations 4 - 9 describe the cost function. Note that our approach (like [6] and [4]) is not limited to any particular cost function. We simply chose this cost function as an example. Also note that the cost function in the Linear Program (1) tries to avoid long paths while trying to meet bandwidth constraints. In the rest of the section, we explain our experimental set up and discuss our observations regarding performance and complexity trade-off.

A. Experimental Set Up

For our experiments, the artificial topologies were generated using the Georgia Tech [10] and BRITE [11] topology generators[3]. The topologies generated using both generators were

[2]www.sprint.net

[3]BRITE allows several options for generating topologies: AS Level, Hierarchical and router level. We chose the router level option.

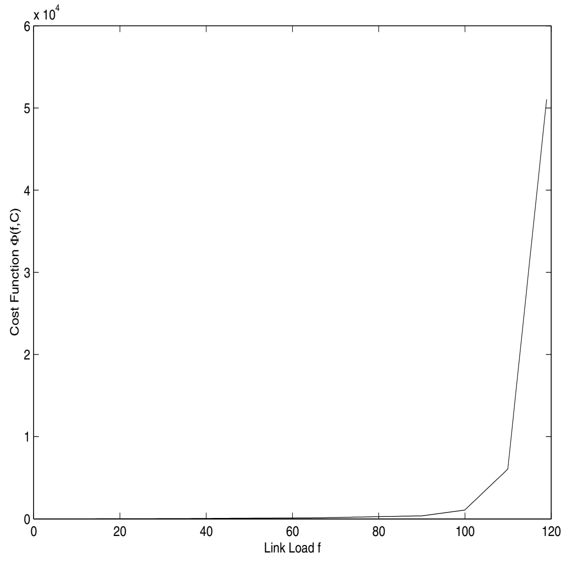

Fig. 1. Evolution of cost function $\Phi_{i,j}(f_{i,j}, C_{i,j})$ as a function of link load $f_{i,j}$.

random graphs constructed by choosing points uniformly on a grid. In all instances of simulated topologies, the link capacities were set to 500 Mbps. Actual physical link capacities were used for the topology based on the Sprint backbone.

For the artificially generated topologies, random traffic matrices were generated by picking the traffic intensity of each routing prefix from a Pareto distribution. The choice of a Pareto distribution was motivated by measurements taken from several routers on the Sprint backbone (see [12] for details). We also experimented with other distributions, i.e., uniform, bimodal, Gaussian, and exponential in other experiments, but do not include them as the results were similar to those obtained with the Pareto distribution. The Sprint traffic matrix was based on actual traffic traces downloaded from access links to two of the Sprint backbone routers. The traces was measured at the granularity of the routing table entries[4] and gives us two rows of the traffic matrix. The routing prefix intensities in the remaining rows were generated artificially using a Pareto distribution. The other parameter of importance is the number of routing prefix associated with each egress router. For this, we used both a uniform and a Pareto distribution, as it gives a reasonable coverage for the possible difference in the number of available routing prefixes to a given egress router.

Each experiment was conducted in the following fashion :

1) For each network topology, we generated random traffic matrices, varying both the total number of routing prefixes and distribution[5] from which the ingress traffic intensity of each routing prefix was picked.
2) Hot spots were introduced in the traffic matrix by randomly selecting elements from the traffic matrix and scaling them to create several instances of the traffic matrix. We tested cases where only some of the traffic elements were chosen and also cases where all entries were chosen. In the latter case, this involves scaling the entire traffic matrix.
3) The "*optimal routing problem*" (10) was then solved for each such instance (topology and traffic matrix).
4) The linear program (1), with the optimal link bandwidths from the "*optimal routing solution*" as input, was solved to obtain the traffic allocation (which was aggregated based on destination, ref. Section II-A) and the set of link weights.
5) Finally, the three heuristics were run over the network with the link weights and traffic flows from the previous step (*please refer to pseudo-code*).

In most of our trials, the link weights turned out to be integers in the range $1-20$. In a few experiments however, the weights were not integers. In such cases, we rounded the link weights to within 5 digit accuracy, which was found to be sufficient in all cases. We used ILOG CPLEX to solve the *optimal routing problem* and the linear program (1). On a Dell 2500 1 Ghz machine it took about 2 hours to solve the *optimal routing problem* and 30 minutes and less than 10 minutes for the LP (1) and our heuristics respectively, on the largest networks.

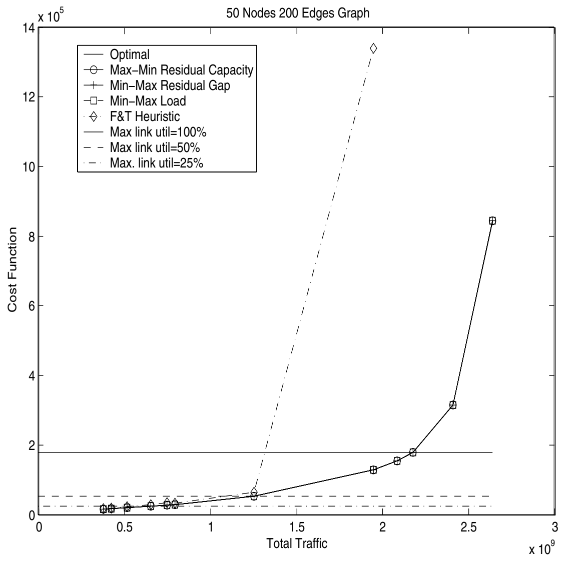

Fig. 2. BRITE 50 Node 200 Edge graph: Performance of the 3 heuristics with an average of 26500 routing prefixes per node.

B. Performance Comparison against Optimal Routing

We now present and discuss the results of our experiments. In Figure 2, we plot cost vs total traffic demand for all 3 heuristics and optimal routing on a 50 Node 200 Edge graph with a granularity of 26500 routing prefixes per node. This number was chosen simply as an approximation of the number of routing prefixes in a backbone router. We have conducted experiments with upto 100,000 routing prefixes and as few as 500 routing prefixes without any significant change in performance. The graph was generated using the BRITE generator. The horizontal lines represent various levels of maximum average link utilization over all links for optimal

[4]The routing prefixes are averages over 10 hrs.

[5]Except in the case of the Sprint traffic matrix.

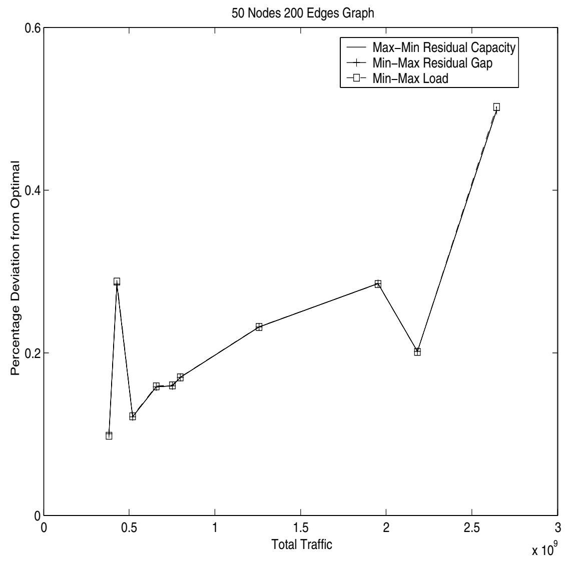

Fig. 3. BRITE 50 Node 200 Edge graph:% Deviation of the 3 heuristics from the optimal with an average of 26500 routing prefixes per node.

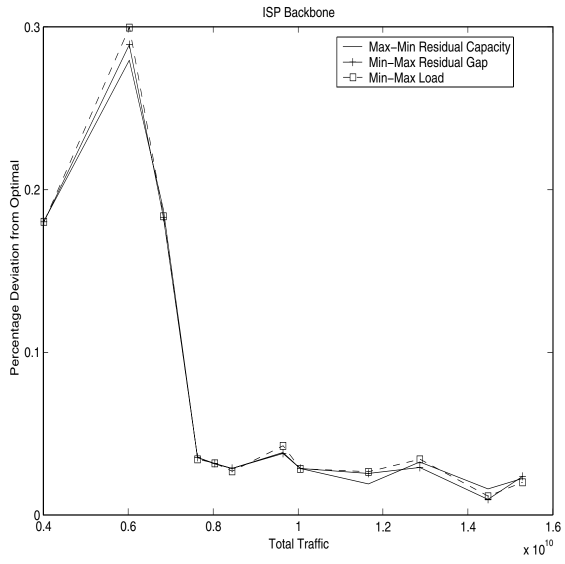

Fig. 5. Sprint Backbone:% Deviation of the 3 heuristics from the optimal.

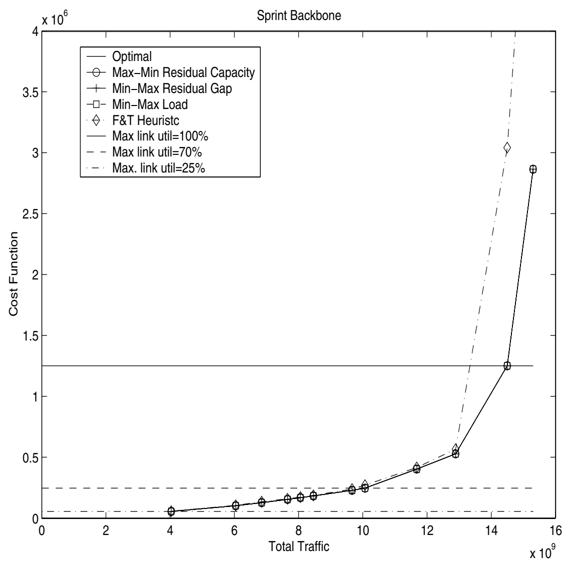

Fig. 4. Sprint Backbone: Performance of the 3 heuristics.

routing. The traffic matrix for this experiment was scaled by selecting 70% of the traffic elements as hotspots. From the figure, we see that in all the cases, the heuristics are very near the optimal solution indicating that they are able to match the optimal traffic split very closely. Moreover, all three heuristics perform equally well in all instances. For comparison, we have also shown the performance of standard OSPF routing with weights computed using our implementation of the heuristic proposed in [4] (denoted by "F&T Heuristic" in the graph). In Figure 3 we plot the % deviation from the optimal for the three heuristics. The low percentage deviation $(0.2\% - 1\%)$ from the optimal value highlights how effective the heuristics are.

In Figures 4 and 5, we plot the performance of the heuristics on the Sprint Backbone. The entire traffic matrix was scaled for the experiments involving the Sprint backbone. We can see again that the heuristics clearly perform very well (well within 1% of the optimal). We observed this kind of performance in a number of other experiments that we conducted but have not shown here due to their similar nature. For results regarding a topology generated using the Georgia Tech topology generator please refer [12].

C. Lowering Configuration Overhead

Our other goal was to investigate the trade-off between configuration overhead and performance. Recall that in the original approach the heuristics decide the *subset* of next hops assigned to *every* routing prefix. However, it has been observed that in practice ([13]), a large fraction of the traffic is distributed over a relatively small number of routing prefixes. Our analysis of the backbone traces obtained from the Sprint router show that 95% of the total traffic was accounted for by only 10% of the routing prefixes, confirming the results reported in [13]. Figure 6 highlights this observation, where we have plotted the cumulative traffic intensity as a function of the number of routing prefixes sorted in decreasing order of their traffic intensities. We can potentially exploit such a phenomenon by configuring the set of next hops for only a few selective routing prefixes that carry most of the traffic and allowing the default assignment of all next hops for the remaining routing prefixes. This has the advantage of lowering configuration overhead, but raises the question of how it impacts performance.

We carried out a systematic study of such a trade-off on all the previous topologies. In each instance, we configured the set of next hops at each node for only a certain set of routing prefixes that were selected based on the amount of traffic they carried. The remaining routing prefixes were split equally over the entire set of next hops as would happen with default OSPF/IS-IS behavior. The set of configured routing prefixes was then progressively increased in each experiment to determine the evolution of the impact on performance. In all cases the MIN-MAX LOAD heuristic was used when configuring the set of next hops.

The resulting performance curves for the 50 Node 200 Edge graph are shown in Figure 7 and the number of configured routing prefixes are shown in Table I. Each curve on the plot is referenced by the amount of traffic that was accounted for by the configured routing prefixes. This can be cross-referenced from the table against the number of routing prefixes that were configured. We observe that on an average, by configuring about 165 routing prefixes per router we get good performance till about 50% maximum link utilization. If we configure next hops for about 17 % of all routing prefixes, or 4500 entries, at a router, we account for approximately 75% of the traffic and the resulting performance is quite close to that of optimal routing.

Experiments conducted on the Sprint Backbone (Figure 8, Table II) yield similarly encouraging results. We get good performance up to approximately 50-60% maximum link utilization, by configuring only 200 routing prefixes per router and up to more than 70% link utilization if we configure 600 routing prefixes per router.

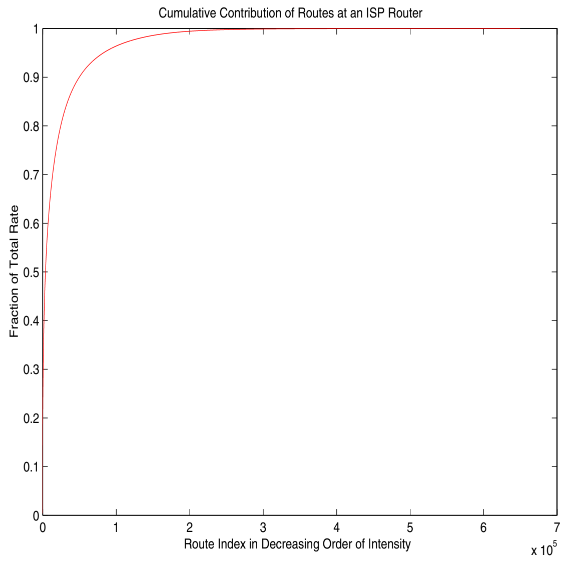

Fig. 6. Cumulative contribution of routing prefixes at a Sprint Router sorted in decreasing order of intensity.

Prefixes Configured	Total No. of Prefixes	% Allocated Fraction	% of Traffic
75	26500	0.3%	10 %
165	26500	0.62%	20 %
1252	26500	4.7 %	50 %
4500	26500	17.0 %	75 %
11747	26500	44.32 %	90 %

TABLE I

CONFIGURATION OVERHEAD: 50 NODE 200 EDGE GRAPH, ALL ENTRIES ARE PER NODE

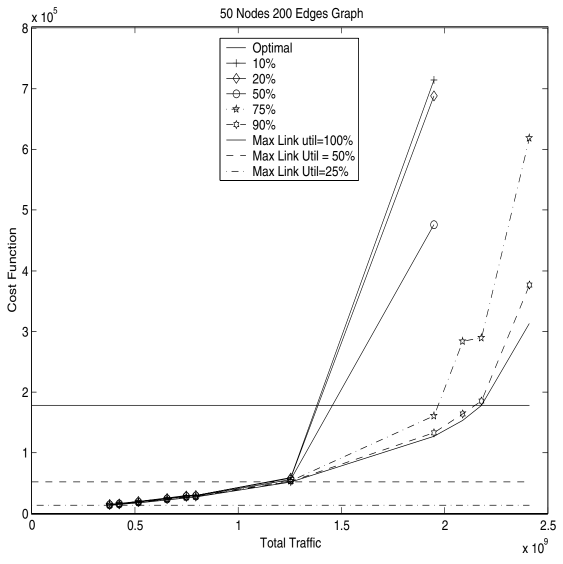

Fig. 7. BRITE 50 Node 200 Edge graph : Performance as a function of configuration overhead

Prefixes configured	Total No. of Prefixes	% Allocated Fraction	% of Traffic
160	30700	0.5%	10 %
200	30700	0.6%	20 %
620	30700	2 %	50 %
1750	30700	6 %	75 %
4180	30700	14 %	90 %

TABLE II

CONFIGURATION OVERHEAD: SPRINT BACKBONE, ALL ENTRIES ARE PER NODE

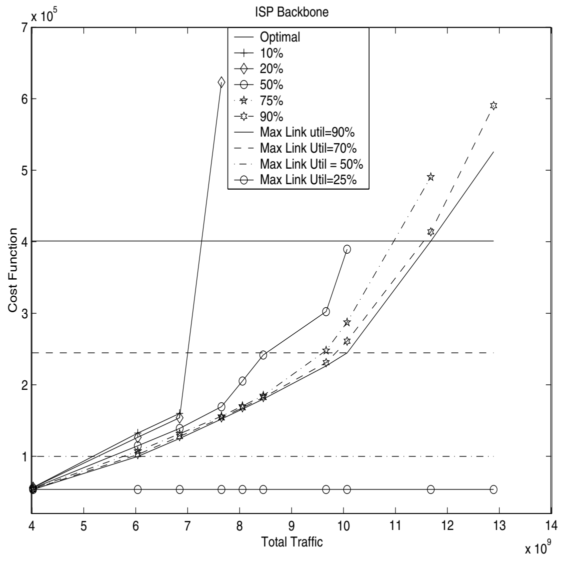

Fig. 8. Sprint Backbone: Performance as a function of configuration overhead.

V. Conclusion

In this paper, we have described and evaluated an approach that has the potential for providing the benefits of traffic engineering to existing IP networks, i.e., without requiring changes to either the routing protocols or the forwarding mechanisms.

Our contribution is three-fold. First, we propose a solution whereby we can closely approximate the optimal link loads without changing current forwarding mechanisms, namely, by carefully controlling the set of next hops for each prefix. Second, we propose a heuristic with a provable performance bound (see Appendix) as well as two other simple heuristics. All three were shown to give excellent and similar performance through experiments. We believe the heuristics are general enough to be potentially useful in their own right. Finally, we showed, using actual traffic traces, that configuration overhead can be vastly reduced without significant loss of performance. Specifically, by only configuring next hops for a small set of prefixes, we were able to obtain near-optimal performance for link loads of up to 70%. This is obviously an important aspect for the practical deployment of our traffic engineering solution.

There are clearly many other aspects that need to be addressed in order to formulate a fully operational solution to traffic engineering. One topic we are currently investigating is that of making our traffic engineering solution robust to unexpected changes in network topology, e.g., link or router failures. This is obviously an important aspect and one of the areas that traffic engineering solutions that rely on new forwarding technologies, e.g., MPLS, have focused on ([14]). The general direction of the solution we are pursuing, is to compute a set of link weights that are robust enough to give good performance in normal or failure scenarios. In case of a failure, the assignment of next hops, as proposed in the paper, would either remain the same, or default to the standard assignment for routing prefixes that are going over an entirely different set of next hops after the failure. We are currently investigating the performance and robustness of such an approach ([15]).

Another issue we are investigating, is that in some cases we encountered, link weights were not integer. Although this did not happen often, it is still problematic since the dual of the LP (2) does not guarantee integer link weights as required by OSPF/IS-IS. In our experiments, we rounded off the link weights to 5 digit accuracy. However such a solution may not always be feasible due to the limited field length for weights in current OSPF and IS-IS protocols, which limits scaling. Instead, we are investigating methods that will allows us to obtain integer weights consistent with the original weights. We believe that a solution that accounts for both problems is feasible, and that this work represents another argument in favor of evolving the current infrastructure to support traffic engineering, if and when needed, rather than embark on a migration to a rather different technology. There may be justifications for such a migration, e.g., better support for policies or VPNs, but traffic engineering does not appear to be one of them, and we hope that the results of this paper can help clarify this issue.

References

[1] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," Internet Engineering Task Force, Request For Comments (Standards Track) RFC 3031, January 2001.

[2] J. Moy, "OSPF Version 2," Internet Engineering Task Force, Request For Comments (Standard) RFC 2328, April 1998.

[3] R. Callon, "Use of OSI IS-IS for routing in TCP/IP and dual environments," Internet Engineering Task Force, Request For Comments (Standard) RFC 1195, December 1990.

[4] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proceedings of INFOCOM'2000*, Tel Aviv, Israel, March 2000.

[5] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows, Chapter 17, Section 17.2*. Prentice-Hall Inc., 1990.

[6] Z. Wang, Y. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying," in *Proceedings of INFOCOM'2001*, Anchorage, Alaska, April 2001.

[7] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational IP networks: Methodology and experience," *IEEE/ACM Transactions on Networking*, vol. 9, 2001.

[8] C. Villamizar, "MPLS optimized multipath MPLS-OMP," INTERNET-DRAFT, `draft-villamizar-mpls-omp-01.txt`, February 1999, (work in progress).

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[10] E. W. Zegura, "GT-ITM: Georgia tech internetwork topology models (software)," Georgia Tech," http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm/gt-itm/tar.gz, 1996.

[11] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: Boston university representative internet topology generator," Boston University," http://cs-www.bu.edu/brite, April 2001.

[12] R. G. Ashwin Sridharan and C. Diot, "Achieving Near-Optimal Traffic Engineering Solutions for Current OSPF/IS-IS Networks," University of Pennsylvania, Tech. Rep., May 2002, Available at http://einstein.seas.upenn.edu/publications.html.

[13] S. Bhattacharya, C. Diot, J. Jetcheva, and N. Taft, "Pop-level and access-link level traffic dynamics in a tier-1 pop," *Proceedings of ACM SIGCOMM Internet Measurement Workshop (IMW 2001)*, November 2001.

[14] M. Kodialam and T. Lakshman, "Dynamic routing of bandwidth guaranteed tunnels with restoration," in *PROCEEDINGS of INFOCOM*, Tel Aviv, Israel, March 2000.

[15] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *Journal on Selected Areas in Communication*, February 2002.

[16] R. L. Graham, "Bounds on multiprocessing timing anamolies," *SIAM Journal on Applied Mathematics*, vol. 17, March 1969.

Appendix

Our analysis of the MIN-MAX LOAD heuristic consists of two steps. We first give a performance bound when the set of routes is unordered. We then demonstrate an improved bound when the set of routes is ordered according to their traffic intensity.

Proposition 1: Heuristic MIN-MAX LOAD achieves a load that is no more than $(1 + \ln K)$ times that of the maximum load with an optimal algorithm, where K is the number of next hops (for a given destination).

Proof: The proof proceeds in two steps. First we identify a key property of the MAX-MIN LOAD heuristic, namely that for each *virtual* assignment, we can associate a distinct hop. Next we use this property to establish the main result.

We start by establishing some notation. Denote the maximum load achieved by Heuristic MIN-MAX LOAD as $\gamma(\mathcal{X}_{n,m}, \mathcal{K})$. Let hop $t \in \mathcal{K}$ achieve this load and the last prefix assigned to hop t be prefix j. For simplicity, we denote the intensity on hop k *before* j is assigned by l_k. By definition, we have

$$\gamma(\mathcal{X}_{n,m}, \mathcal{K}) \geq \frac{l_k}{f_k} \quad \forall \quad k \in \mathcal{K}.$$

Let $\gamma_o(\mathcal{X}_{n,m}, K)$ denote the maximum load achieved by an optimum algorithm that satisfies the equal subset splitting constraint, i.e., splits traffic equally across the subset of next hops that have been assigned to a route. Let

$$L = \sum_{i=1}^{N} x_i$$
$$F = \sum_{k \in \mathcal{K}}^{K} f_k.$$

Note that $\gamma_o(\mathcal{X}_{n,m}, \mathcal{K}) \geq \frac{L}{F}$ where $\frac{L}{F}$ is the optimum attained if arbitrary splitting of routes was allowed at the node.

First recall how the second step of heuristic MIN-MAX LOAD works. The heuristic does a *virtual* assignment of the streams over an increasing sequence of hops, $\mathcal{M} = \{d \leq k : d \in \mathcal{K}\}, \quad k = 1, 2, \ldots, K$ as described in the algorithm and chooses the arrangement with the smallest maximum for an *actual* assignment. We make the following observations:

1) The smallest maximum among all *virtual* assignments of prefix j must be $\gamma(\mathcal{X}_{n,m}, \mathcal{K})$. If there were a smaller maximum, it would contradict our assumption that j is the last prefix assigned to hop t.
2) In a *virtual* assignment of prefix j over p hops, let r_p denote the index of the hop with the maximum load *among* the *virtually* assigned hops (to which a portion x_j/p of the route was *virtually assigned*). Then r_p must be maximal over *all* hops, for that *virtual* assignment. If it were not, let there exists a hop k (which must belong to the set of *virtually* unassigned hops) such that

$$\frac{l_k}{f_k} > \frac{l_{r_p} + x_j/p}{f_{r_p}}. \qquad (11)$$

Since all hops satisfy the property $\frac{l_k}{f_k} \leq \gamma(\mathcal{X}_{n,m}, \mathcal{K})$, we have an assignment of prefix j which yields a lower load ratio than $\gamma(\mathcal{X}_{n,m}, \mathcal{K})$. This contradicts our initial assumption that j was the last prefix assigned to hop t such that its load was $\gamma(\mathcal{X}_{n,m}, \mathcal{K})$. Hence r_p must be maximal.

From Observation 1, we have

$$\frac{l_{r_p} + x_j/p}{f_{r_p}} \geq \gamma(\mathcal{X}_{n,m}, \mathcal{K}). \qquad (12)$$

Clearly, the following relation holds for all the $K-p$ *virtually* unassigned hops

$$\frac{l_k + x_j/p}{f_k} \geq \frac{l_{r_p} + x_j/p}{f_{r_p}} \qquad (13)$$

since only the p smallest hops are chosen in the *virtual* assignment. We then have from Equations (12) and (13) that the following relation must hold for r_p and the remaining (if any) $K - p$ *virtually* unassigned hops :

$$\frac{l_k + x_j/p}{f_k} \geq \gamma(\mathcal{X}_{n,m}, \mathcal{K}). \qquad (14)$$

Using these 2 observations we make the following claim.
Claim : For every *virtual* assignment over $p = 1 \ldots K$ hops of prefix j, we can identify a distinct hop $k(p)$ that satisfies

$$\frac{l_{k(p)} + x_j/p}{f_{k(p)}} \geq \gamma(\mathcal{X}_{n,m}, \mathcal{K}). \qquad (15)$$

Proof: The proof is by induction. For a *virtual* assignment by the heuristic over p hops, let $\mathcal{A}_p$ denote the set of hops such that:

$$\mathcal{A}_p = \{k : \frac{l_k + x_j/p}{f_k} \geq \gamma(\mathcal{X}_{n,m}, \mathcal{K})\}.$$

Note from Observation 2 and Equation (14) that $\mathcal{A}_p$ comprises of at least r_p and the $K-p$ unassigned hops. Hence

$$\parallel \mathcal{A}_p \parallel \geq K - p + 1.$$

Also note that $\mathcal{A}_p$ is a non-decreasing sequence for $p = K, K-1, \ldots, 1$, because if some hop $k \in \mathcal{A}_n$, then $k \in \mathcal{A}_{n-1}, \quad n \geq 1$. The claim certainly holds for $p = K$ since by Observation 1, there is at least 1 hop which has a load of $\gamma(\mathcal{X}_{n,m}, \mathcal{K})$. Let the claim hold for all $p = K, K-1, \ldots, n$. Then by the non-decreasing property, $\mathcal{A}_n$ contains all the $K - n + 1$ distinct hops. Let us look at a virtual assignment over $n - 1$ hops. We know that

$$\parallel \mathcal{A}_{n-1} \parallel \geq K - n + 2.$$

Since only $K - n + 1$ hops have been associated (with virtual assignments $p = K$ to $p = n$) and by the non-decreasing property, $\mathcal{A}_{n-1}$, contains all these hops, there is at least 1 hop which has not yet been used and hence can be associated with a virtual assignment over $n-1$ hops. This completes the proof.
We are now in a position to prove **Proposition 1**.
By our previous result, for each possible virtual assignment of prefix j over $p = 1, 2, \ldots, K$ hops, we have a distinct hop $k(p)$ which satisfies Equation (15). Summing Equation (15) over this set of K distinct hops, we have

$$\sum_{p=1}^{K} l_{k(p)} \geq \sum_{p=1}^{K} f_k \, \gamma(\mathcal{X}_{n,m}, \mathcal{K}) - \sum_{p=1}^{K} \frac{x_j}{p},$$
$$\sum_{p=1}^{K} l_{k(p)} \geq F \, \gamma(\mathcal{X}_{n,m}, \mathcal{K}) - \sum_{p=1}^{K} \frac{x_j}{p},$$
$$L - x_j \geq F \, \gamma(\mathcal{X}_{n,m}, \mathcal{K}) - x_j(\ln K + 1), \qquad (16)$$
$$\gamma(\mathcal{X}_{n,m}, \mathcal{K}) \leq \frac{L}{F} + \frac{x_j}{F}(\ln K) \leq \frac{L}{F}(1 + \ln K),$$
$$\gamma(\mathcal{X}_{n,m}, \mathcal{K}) \leq \gamma_o(\mathcal{X}_{n,m}, \mathcal{K})(1 + \ln K), \qquad (17)$$

where the LHS (16) of follows from the fact that the total assigned load is not more than $L - x_j$. This proves **Proposition 1**.

The above analysis holds for an arbitrary ordering of the prefixes. If the prefixes are ordered in decreasing order as is the case in MAX-MIN LOAD, the bound can be improved as can be the performance of the algorithm. Our proof for this result draws on the method used in [16].

Proposition 2: If the prefixes are assigned in decreasing order of their traffic intensities, then,

$$\frac{\gamma(\mathcal{X}_{n,m}, \mathcal{K})}{\gamma_o(\mathcal{X}_{n,m}, \mathcal{K})} \leq (1 + \frac{\ln K}{2}).$$

Proof: The proof is by contradiction. Assume that the above result does not hold for some ordered set of prefixes with intensities $\mathcal{I} = \{x_1, x_2, \ldots, x_N\}$, where

$$x_1 \geq x_2 \geq \ldots \geq x_N \,.$$

Without loss of generality assume x_N is the intensity of the last prefix(N) assigned to the hop, which achieves the maximum load under heuristic MIN-MAX LOAD. If it is not, we can truncate the sequence up to the prefix last assigned to a hop which achieves the maximum load without affecting the maximum achieved by MIN-MAX LOAD. If the optimum for the truncated sequence is $\gamma_o'(\mathcal{X}, \mathcal{K})$, then $\gamma_o'(\mathcal{X}, \mathcal{K}) \leq \gamma_o(\mathcal{X}_{n,m}, \mathcal{K})$ and our assumption still holds. Let as before,

$$L = \sum_{i=1}^{N} x_i \quad \text{and} \quad F = \sum_{k=1}^{k} f_k.$$

Following the exact same analysis as for the arbitrary ordering we have

$$\begin{aligned}
\gamma(\mathcal{X}_{n,m}, \mathcal{K}) &\leq \frac{L}{F} + \frac{x_N}{F} \ln K, \\
&\leq \gamma_o(\mathcal{X}_{n,m}, \mathcal{K}) + \frac{x_N}{F} \ln K, \\
\frac{\gamma(\mathcal{X}_{n,m}, \mathcal{K})}{\gamma_o(\mathcal{X}_{n,m}, \mathcal{K})} &\leq 1 + \frac{x_N}{\gamma_o(x, f) \cdot F} \ln K.
\end{aligned}$$

By our assumption, we have

$$\begin{aligned}
1 + \frac{x_N}{\gamma_o(\mathcal{X}_{n,m}, \mathcal{K}) \cdot F} \ln K &> 1 + \frac{\ln K}{2}, \\
\text{or} \quad \frac{x_N}{F} &> \frac{\gamma_o(\mathcal{X}_{n,m}, \mathcal{K})}{2}.
\end{aligned}$$

Note that $\frac{x_N}{F}$ is the smallest achievable load under *any* split. Hence, if the above inequality, and our assumption, is to hold, we can have only one route (destination prefix) in $\mathcal{X}$. However it is clear from the algorithm itself that MIN-MAX LOAD achieves an optimal allocation when there is only one stream(prefix). This proves **Proposition 2**.