HALO: Hop-by-Hop Adaptive Link-State Optimal Routing

Nithin Michael, *Student Member, IEEE*, and Ao Tang, *Senior Member, IEEE*

***Abstract*—We present HALO, the first link-state routing solution with hop-by-hop packet forwarding that minimizes the cost of carrying traffic through packet-switched networks. At each node u, for every other node t, the algorithm independently and iteratively updates the fraction of traffic destined to t that leaves u on each of its outgoing links. At each iteration, the updates are calculated based on the shortest path to each destination as determined by the marginal costs of the network's links. The marginal link costs used to find the shortest paths are in turn obtained from link-state updates that are flooded through the network after each iteration. For stationary input traffic, we prove that HALO converges to the routing assignment that minimizes the cost of the network. Furthermore, we observe that our technique is adaptive, automatically converging to the new optimal routing assignment for quasi-static network changes. We also report numerical and experimental evaluations to confirm our theoretical predictions, explore additional aspects of the solution, and outline a proof-of-concept implementation of HALO.**

***Index Terms*—IP networks, load balancing, network management, optimal routing.**

I. Introduction

OPTIMAL routing, i.e., finding routing assignments that minimize the cost of sending traffic through packet-switched networks, has been of fundamental research and practical interest since the early 1970s with the advent of ARPANET [3], the predecessor of the Internet. Yet today, we find that the different optimal routing algorithms developed over the last 40 years are seldom implemented. Instead, distributed link-state routing protocols like OSPF/IS-IS that support hop-by-hop packet forwarding are the dominant intradomain routing solutions on the Internet.

The driving force behind the widespread adoption of link-state, hop-by-hop algorithms has been their simplicity—the main idea is to centrally assign weights to links based on input traffic statistics, flood the link weights through the network, and then locally forward packets to destinations along shortest paths computed from the link weights. As our communication networks have grown rapidly in size and complexity, this simplicity has helped OSPF eclipse extant optimal routing techniques that are harder to implement.

However, the obvious tradeoff has been lost performance. For instance, due to the poor resource utilization resulting from OSPF, network administrators are forced to overprovision their networks to handle peak traffic. As a result, on average, most network links run at just 30%–40% utilization. To make matters worse, there seems to be no way around this tradeoff. In fact, given the offered traffic, finding the optimal link weights for OSPF, if they exist, has been shown to be NP-hard [4]. Furthermore, it is possible for even the best weight setting to lead to routing that deviates significantly from the optimal routing assignment [4].

Our goal in this paper is to eliminate this tradeoff between optimality and ease of implementation in routing. The result is Hop-by-hop Adaptive Link-state Optimal (HALO), a routing solution that retains the simplicity of link-state, hop-by-hop protocols while iteratively converging to the optimal routing assignment. To the best of our knowledge, this is the first optimal link-state hop-by-hop routing solution.

Not surprisingly, there are multiple challenges to overcome when designing such a solution. Before getting into them, we define the following important recurring terms for ease of exposition.

Hop-by-hop:	Each router, based on the destination address, controls only the next hop that a packet takes.
Adaptive:	The algorithm does not require the traffic demand matrix as an explicit input in order to compute link weights. Specifically, the algorithm seamlessly recognizes and adapts to changes in the network, both topology changes and traffic variations, as inferred from the network states like link flow rates.
Link-state:	Each router receives the state of all the network's links through periodically flooded link-state updates and makes routing decisions based on the link states.
Optimal:	The routing algorithm minimizes some cost function (e.g., minimize total delay) determined by the network operator. The problem of guiding network traffic through routing to minimize a given global cost function is called traffic engineering (TE).

The first design challenge stems from coordinating routers only using link states. This means that no router is aware of all the individual communicating pairs in the network or their traffic requirements. However, they still have to act independently such that the network cost is minimized. This is a very

Manuscript received July 21, 2013; revised May 29, 2014; accepted July 17, 2014; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Markopoulou. Date of publication September 10, 2014; date of current version December 15, 2015. A preliminary version of this paper appears in the Proceedings of the IEEE International Conference on Network Protocols (ICNP), Göttingen, Germany, October 7–10, 2013.

The authors are with the Department of Electrical and Computer Engineering, Cornell University, Ithaca, NY 14853 USA (e-mail: nm373@cornell.edu; atang@ece.cornell.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2014.2349905

real restriction in any large dynamic network like the Internet, where it is not possible to obtain information about each communicating pair. If the link-state requirement is set aside, optimal distance-vector routing protocols have already been developed [2]. The idea there is to iteratively converge to the optimal routing assignment by sharing estimates of average distances to destinations among neighbors. However, distance-vector protocols have not caught on for intradomain routing because of scalability issues due to their slow convergence and robustness issues like vulnerability to a single rogue router taking down the network as in the "Internet Routing Black Hole" incident of 1997 [5].

The hop-by-hop forwarding requirement presents the next challenge. As a result, a router cannot determine the entire path that traffic originating at it takes to its destination. Without this requirement, a projected gradient approach [6] can be used to yield optimal iterative link-state algorithms that can be implemented with source routing, where the path a packet takes through the network is encoded in its entirety at the source. However, the need for source routing means that these techniques are not practical given the size of modern networks.

Another challenge arises because the optimal routing assignment changes with the input traffic and the network. There are two aspects to this problem. The first aspect is that the algorithm needs sufficient time between network and traffic changes to calculate and assign optimal routes. This requirement is typically captured by the quasi-static model of routing problems described by Gallager [2]. The second aspect is that the algorithm should smoothly adapt the routes to changes when they do occur. Thus, ideally, the algorithm should avoid global inputs that require additional computation when performing routing updates. However, the algorithm also needs some way to track the network state to compute efficient routes. Link rates fill this gap because they are widely available and easily accessible in modern networks. The first aspect is modeled by studying a static network with static input traffic in between changes in the network. If the second stipulation is set aside, recently, significant progress was made in this direction with PEFT, a link-state protocol with hop-by-hop forwarding based on centralized weight calculations [7]. However, since the link weights are calculated in a centralized manner with the traffic matrix as an explicit input, PEFT is not adaptive. Nor does it always guarantee optimality as claimed in the paper.[1]

The rest of the paper is organized as follows. In Section II, we review several different solutions that have been proposed for the traffic engineering problem. Next, we formulate the quasi-static traffic engineering problem in Section III and gain new insights from it in Section IV. This intuition forms the basis for HALO and the proof of its optimality presented in Section V. Section VI discusses additional aspects of HALO, particularly those related to implementation. Numerical evaluations are used to verify our predictions and to move from our continuous-time model to a discrete-time implementation in Section VII. Section VIII presents the proof-of-concept testbed implementation of HALO, which is also used to physically verify our predictions. Finally, we conclude the paper with a summary and future research directions in Section IX.

[1]When the optimal routing solution does not use all available paths to the destination, as is the case in many networks (for example, any network with at least a loop), a set of finite optimal weights for PEFT does not exist.

II. Related Work

Over the years, due to its importance, traffic engineering has attracted a lot of research attention. We provide a brief overview of major related results from different communities such as control, optimization, and networking. Broadly, the existing work can be divided into OSPF-TE, MPLS-TE, traffic demand agnostic/oblivious routing protocol design, and optimal routing algorithms.

The work on OSPF [4], [8], [9] has concentrated on using good heuristics to improve the centralized link weight calculations. Although these techniques have been shown to improve the algorithm's performance significantly by finding better weight settings, the results are still far from optimal.

Typically, these and other centralized traffic engineering techniques also require reliable estimates or measurements of the input traffic statistics in the form of a traffic matrix. While excellent work has been done in traffic matrix estimation from link loads, even the best results have errors on the order of 20% [10], which can lead to bad traffic engineering. Another approach is to directly measure the traffic to every destination at every router. While it is possible to globally aggregate the measurements into a traffic matrix that can be fed to a traffic engineering algorithm, it is more straightforward to use local measurements locally. Also, usually it is smoother and quicker to respond to changes locally when they do occur. Thus, we are advocating a shift to relying directly on link loads and local traffic measurements instead of computing a traffic matrix for traffic engineering.

A good way to avoid traffic matrices and a popular way to implement traffic engineering today is MPLS-TE [11], [12]. The idea is to compute end-to-end tunnels for traffic demands with the available network bandwidth being assigned to new traffic demands using techniques like Constrained Shortest Path First. However, here, the performance gained over OSPF comes at the cost of establishing multiple end-to-end virtual circuits. Moreover, as the traffic changes, the end-to-end virtual circuits that were established for a particular traffic pattern become less useful, and performance degrades.

Oblivious routing has also been proposed as a way around using traffic matrices for traffic engineering. The idea is to come up with a routing assignment that performs well irrespective of the traffic demand by comparing the "oblivious performance ratio" of the routing, i.e., the worst-case performance of the routing for a given network over all possible demands. Breakthrough work in this area includes papers by Applegate and Cohen [13] that developed a linear programming method to determine the best oblivious routing solution for the special case of minimizing maximum channel utilization and Kodialam *et al.* [14] that focused on maximizing throughput for the special case of two-phase routing. However, oblivious routing solutions do not adapt well to changes in the network topology and, by not tailoring the routing to the traffic demand, still incur significant performance losses.

TABLE I
COMPARISON OF ROUTING ALGORITHMS

Algorithm	Link-state	Hop-by-hop	Optimal	Adaptive
OSPF	✓	✓	×	×
Gallager's [2]	×	✓	✓	✓
Projected Gradient [6]	✓	×	✓	✓
PEFT [7]	✓	✓	×	×
HALO	✓	✓	✓	✓

On the other hand, we have the optimal distance-vector and source routing protocols discussed earlier. Despite their implementation problems, these iterative algorithms deliver on performance without a traffic matrix by relying on knowledge of the link flow rates.

Distance-vector protocols include the ones proposed by Gallager in his classic paper [2], Stern [15], and Agnew [16]. From an optimization standpoint, they are natural and mathematically elegant algorithms since the main ideas follow directly from the decomposition of the dual of the traffic engineering optimization problem. Decompositions like this, which have been very successful for problems of this type [17], can be used to yield updating rules for primal and dual variables (split ratios and node prices in [2]) that can be shown to converge to optimal solutions. Similar node-based ideas have also been applied to the cross-layer optimization of networks [18], [19].

The source routing protocols include the flow deviation technique advocated by Fratta *et al.* [3], projection methods proposed by Bertsekas and Gafni [6], as well as proximal decomposition methods [20]. These algorithms are based on iteratively calculating a shortest path at the source for each destination and transferring varying amounts of flow from the nonshortest paths to the shortest path till the optimal routing assignment is reached. It is worth noting here that although MPLS allows the source to control the path a packet takes through the network, it may not be practical for these protocols. Their iterative nature means that a large number of label switched paths might need to be computed and set up while they converge even though several of these paths will become redundant when the traffic changes.

The preceding literature review clearly reveals an important missing link—an optimal link-state hop-by-hop routing algorithm. In this paper, we provide this missing link by introducing HALO. A comparison of representative existing solutions and ours can be seen in Table I.

III. PROBLEM FORMULATION

Under the quasi-static model, the traffic engineering problem can be cast as a Multi-Commodity Flow (MCF) problem in between topology and input traffic changes. We model the network as a directed graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ with node/router set $\mathbb{V}$ and edge/link set $\mathbb{E}$ with link capacities $c_{u,v},\ \forall (u,v) \in \mathbb{E}$. The rate required for communication from s to t is represented by $D(s,t)$. The commodities are defined in terms of their final destination t. We use $f^t_{u,v}$ to represent the flow on link (u,v) corresponding to commodity t and $f_{u,v}$ for the total flow on link (u,v). The network cost function, Φ, is typically selected to be a convex function of the link rate vector $f = \{f_{u,v}\},\ \forall (u,v) \in \mathbb{E}$. For example, if we use the M/M/1 delay formula for the cost function, then $\Phi(f) = \sum_{u,v} \Phi_{u,v}(f_{u,v}) = \sum_{u,v} f_{u,v}/(c_{u,v} - f_{u,v})$ [21]. Throughout the paper, for numerical examples, we will use this cost function unless specified otherwise. It is also assumed that $\Phi'_{u,v}(f_{u,v}) \to \infty$ when $f_{u,v} \to c_{u,v}$. This captures the common practice of not allowing links to operate too close to capacity. In this paper, given a function $\gamma(x(\tau))$, we will use γ' to represent the derivative of γ with respect to x and $\dot{\gamma}$ to represent the time (τ) derivative of γ. Using this notation, the MCF problem can be stated as

$$\begin{aligned}
&\min_{f^t_{u,v}} \ \Phi(f) \\
&\text{s.t.} \sum_{v:(s,v)\in\mathbb{E}} f^t_{s,v} - \sum_{u:(u,s)\in\mathbb{E}} f^t_{u,s} = D(s,t) \qquad \forall s \neq t \\
&f_{u,v} = \sum_{t\in\mathbb{V}} f^t_{u,v} \leq c_{u,v} \qquad \forall (u,v) \\
&f^t_{u,v} \geq 0.
\end{aligned}$$

A useful observation about the above problem is that there is a one-to-one correspondence between $f^t_{u,v}$ and the split ratios at a router u, $\alpha^t_{u,v}$, which is the fraction of traffic destined to terminal t that router u forwards to neighbor v [2]. This observation will prove useful later as we try to derive an iterative procedure to solve the MCF problem. By definition, the split ratios satisfy, $\alpha^t_{u,v} \geq 0,\ \forall (u,v) \in \mathbb{E}$ and $\sum_{v:(u,v)\in\mathbb{E}} \alpha^t_{u,v} = 1,\ \forall u \in \mathbb{V}$. In our analysis, we will also need the price of a link (u,v), $w_{u,v} = \Phi'_{u,v}(f_{u,v})$, the price of a path p, $d_p = \sum_{(u,v)\in p} w_{u,v}$, and the price from u to t

$$q^t_u = \sum_{v:(u,v)\in\mathbb{E}} \alpha^t_{u,v}[w_{u,v} + q^t_v] \tag{1}$$

where $q^t_t = 0$. We can interpret q^t_u as the average price to t from u where the average is taken over all outgoing edges of u weighted by the split ratios along those edges. If instead the average is done over all possible paths, (1) can be stated without recursion as

$$q^t_u = \sum_{p\in P_{u,t}} d_p \prod_{(i,j)\in p} \alpha^t_{i,j} \tag{2}$$

where $P_{u,t}$ is the set of all paths from u to t. For convenience, we also define the total rate of communication from s to t

$$r^t_s = \sum_{u:(u,s)\in\mathbb{E}} f^t_{u,s} + D(s,t)\,.$$

A fact about MCF is that its optimal solution generally results in multipath routing instead of single-path routing [22]. However, finding the right split ratios for each router for each commodity is a difficult task. Our starting point is to merge the link-state feature of the source-routing protocols with the hop-by-hop forwarding feature of the distance-vector schemes. Another characteristic that we borrow is the iterative nature of these algorithms. Here, each iteration is defined by the flooding of existing link states through the network followed by every router

updating its split ratios, which modifies the link states for the next iteration. In what follows, we measure time in units of iterations. With this idea in mind, in the time between network changes when the topology and the input traffic is static, we do the following.

- Iteratively adjust each router's split ratios and move traffic from one outgoing link to another. This only controls the next hop on a packet's path leading to hop-by-hop routing. If instead we controlled path rates, we would get source routing.
- Increase the split ratio to the link that is part of the shortest path at each iteration even though the average price via the next-hop router may not be the lowest. If instead we forwarded traffic via the next-hop router with the lowest average price, we get Gallager's approach, which is a distance vector solution.
- Adapt split ratios dynamically and incrementally by decreasing along links that belong to nonshortest paths while increasing along the link that is part of the shortest path at every router. If instead split ratios are set to be positive instantaneously only to the links leading to shortest paths, then we get OSPF with weights, $w_{u,v}$.

IV. Special Cases

In order to develop an intuitive understanding of why our solution takes the form that it does, it is helpful to consider a few concrete special cases first. These four cases, each of which clearly highlights the reason for including a particular factor in our solution, progressively lead us to the final algorithm. In each example, our algorithm design will exploit the fact that the KKT optimality conditions [23] of the MCF problem require that at the optimal solution the traffic rate is positive only along paths with the lowest price. The overall idea behind these examples is to design an algorithm that reduces the network cost at each iteration by moving to a routing assignment that satisfies this condition. In Section V, we will extend these ideas and show that the final algorithm that iteratively reduces the network cost will also always lead to the optimal routing assignment.

A. Finding the Right Split Dynamically

First, let us consider a very simple example illustrated in Fig. 1(a). Here, there is traffic demand of rate r with the choice of two links, l and s, to go from A to B. Assuming initially $w_l > w_s$, a simple strategy to reach optimality will be to dynamically shift traffic at some rate $\delta > 0$ from the more expensive link to the cheaper link till the prices of the two links become the same. At node A, this would be equivalent to decreasing α_l and increasing α_s at rate δ/r.

There are two ways to interpret and generalize the intuition gained from this scenario. Both give the same solution for this very simple example, but in general will lead to different dynamics (see Fig. 2) and possibly different split ratios [see Fig. 11(a)]. One interpretation, which underpins the distance-vector algorithms, is that the router should shift traffic away from neighbors with higher average price to the neighbor with the lowest average price. A different interpretation, which is the basis of our protocol, is that the router should shift traffic from links along more expensive paths to the link along the

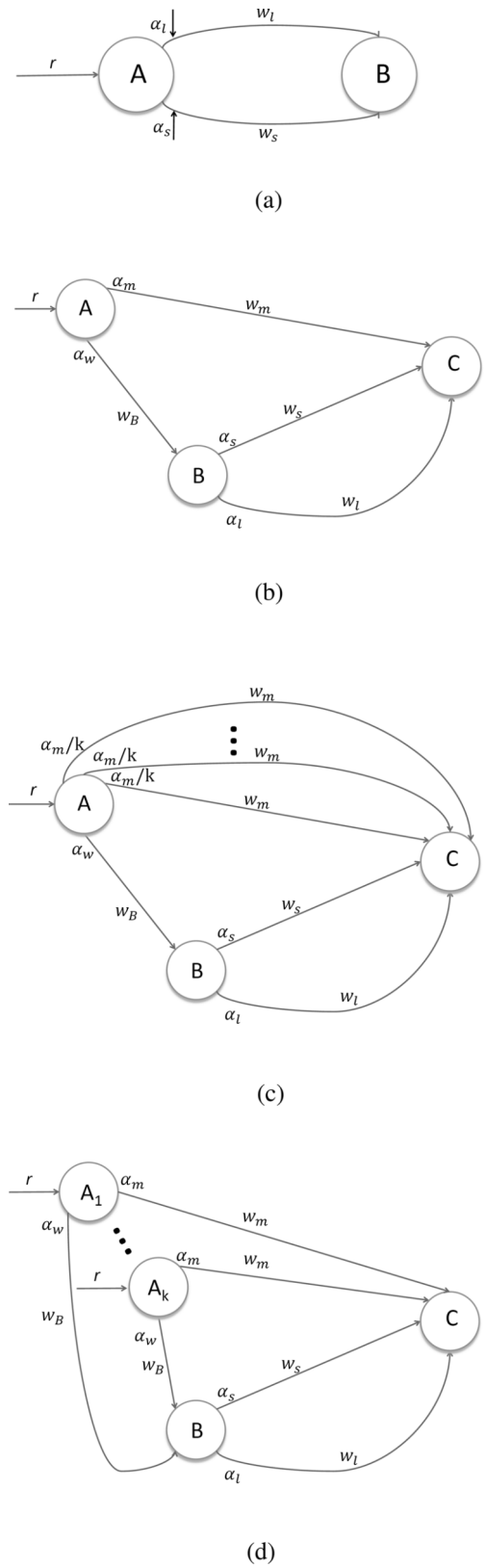


Fig. 1. Four illustrative examples. (a) Finding the right split dynamically. Suppose there is a single demand of rate r to destination B. Initially, the split ratios at A are α_l along the more expensive ("longer") link with price $w_l = \Phi_l'(\alpha_l r)$ and α_s along the cheaper ("shorter") link with price $w_s = \Phi_s'(\alpha_s r)$. (b) First test. Suppose the link weights are as shown and $w_l > w_m > w_s + w_B$. There is a single demand $D(A, C) = r$. (c) Multiple outgoing paths. Suppose the link weights are as shown and $w_l > w_m > w_s + w_B$. There is a single demand $D(A, C) = r$. (d) Multiple inputs. Suppose the link weights are as shown and $w_l > w_m > w_s + w_B$. There are k demands $D(A_i, C) = r,\ i = 1, \ldots, k$.

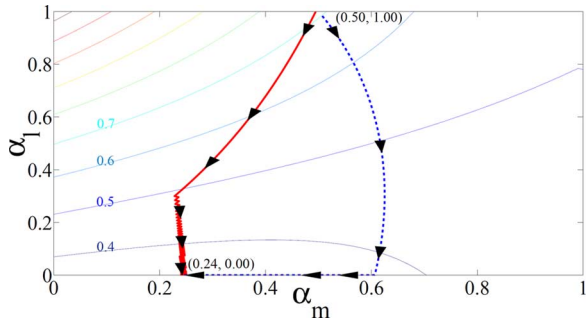


Fig. 2. Trajectories taken by Gallager's algorithm (dashed line) and HALO (solid line) to converge to the optimal solution. Cost values are shown for some contour lines.

path with the lowest price. Mathematically, we reach the following update rule for the split ratios:

$$\dot{\alpha}^t_{u,v} = -\frac{\delta}{r^t_u} \tag{3}$$

where $(u, v) \in \mathbb{E}$ but is not on the shortest path from u to t.

B. First Test

However, as a potential counterexample to this interpretation, it is possible to suggest some version of the scenario described in Fig. 1(b). Here, there is traffic demand of rate r from router A to router C. The initial splits at router A are α_m along an intermediate price link with price w_m and α_w along the more expensive route with price $w_B + w_l$, assuming $\alpha_l = 1$ initially. The relationship between the initial link prices are assumed to be $w_l > w_m > w_s + w_B$, i.e., link (A, B) is along the shortest path from A to C, but B also has the most expensive way to reach C. The concern is that router A shifting traffic from the intermediate price link to the link with price w_B might result in the cost increasing as router B initially routes traffic only through the most expensive link ($\alpha_l = 1$). However, because router B decreases α_l and increases α_s (in conjunction with the changes at router A), the total cost does in fact decrease. More precisely, the cost derivative can be calculated as follows:

$$\begin{aligned}\dot{\Phi} =& -r \times \frac{\delta}{r} \times w_m + r \times \frac{\delta}{r} \times (w_B + w_l) \\ & - r_B \times \frac{\delta}{r_B} \times w_l + r_B \times \frac{\delta}{r_B} \times w_s \\ =& -\delta(w_m - w_B - w_s) \leq 0\end{aligned}$$

where r_B is the incoming rate to C at B (superscript dropped for convenience since C is the only destination) and the inequality follows from the relationship between the prices.

This particular example can also be used to illustrate the difference between our approach and Gallager's technique, which arises from the fact that the link leading to the neighbor with the lowest average price (path A-C with price w_m) may not lead to the cheapest path (path A-B-C with price $w_B + w_s$). Fig. 2 shows the trajectories taken by the two different algorithms to converge to the optimal solution for this topology. In order to simulate the long link between nodes B and C, an intermediate dummy node D is introduced that splits the bottom link between B and C into two identical links. The capacities used were $(A, B) = 5, (B, C) = 10, (A, C) = (B, D) = (D, C) = 3$. The rate $r = 1$, and initially $\alpha_w = \alpha_m = 0.5$ and $\alpha_l = 1$. Here, we take only one split ratio at each node because the value of that split ratio automatically defines the value of the other at each node. Initially, as can be seen, Gallager's algorithm, following the lowest average price path to the destination (A, C), increases the value of α_m. Also, as expected from theory, the trajectory of the algorithm (gradient descent) is perpendicular to the objective function contour curves. On the other hand, using HALO, both split ratios are decreased simultaneously. It turns out that HALO's trajectory is usually not perpendicular to the contour curves. However, it still goes along a descent direction and drives the total cost down.

C. Multiple Outgoing Paths

The above case study might lead us to ask whether the simple rule developed thus far (3) is sufficient to guarantee decreasing network cost along its trajectory. In order to see why it is not, consider the situation presented in Fig. 1(c). Now there are k intermediate price links from router A to router C each of which gets α_m/k fraction of the demand. The relationship between the link prices is the same as in the previous example. Now the concern is that shifting traffic in an unrestricted fashion from the intermediate price links to router B with $\alpha_l = 1$ might result in an increase in the cost, and it is a valid concern as illustrated by the following calculation:

$$\begin{aligned}\dot{\Phi} =& -k \times r \times \frac{\delta}{r} \times w_m + k \times r \times \frac{\delta}{r} \times (w_B + w_l) \\ & - r_B \times \frac{\delta}{r_B} \times w_l + r_B \times \frac{\delta}{r_B} \times w_s \\ =& -k\delta w_m + \delta(kw_B + w_s) + (k-1)\delta w_l\end{aligned}$$

which may be positive for $k > 1$. However, this problem can be surmounted by modifying the update rule followed by the split ratios by adding a weighting factor of the split ratio itself. Mathematically, we have

$$\dot{\alpha}^t_{u,v} = -\frac{\alpha^t_{u,v}\delta}{r^t_u} \tag{4}$$

where $(u, v) \in \mathbb{E}$ but is not on the shortest path from u to t. With this new rule, the cost derivative can be evaluated as

$$\begin{aligned}\dot{\Phi} =& -k \times r \times \frac{\delta\alpha_m}{rk} \times w_m + kr \times \frac{\delta\alpha_m}{rk} \times (w_B + w_l) \\ & - r_B \times \frac{\delta}{r_B} \times w_l + r_B \times \frac{\delta}{r_B} \times w_s \\ =& -\delta[\alpha_m w_m + (1-\alpha_m)(w_B + w_l)] + \delta(w_B + w_l) \\ & - \delta w_l + \delta w_s \\ =& -\delta[\alpha_m w_m + (1-\alpha_m)(w_B + w_l)] + \delta(w_B + w_s) \\ \leq& 0\end{aligned}$$

where the last inequality follows from the fact that the average price from router A to C, which is $\alpha_m w_m + (1-\alpha_m)(w_B + w_l)$, has to be at least as large as the price of the shortest path from A to C, which is $w_B + w_s$.

D. Multiple Inputs

Does (4) ensure that total cost decreases along its trajectory? Another case worth considering is illustrated in Fig. 1(d). This time there are k sources $A_1, \ldots, A_k$ that each have data rate r to send to router C. Now the concern is that shifting traffic in an unrestricted manner from all the sources to router B with $\alpha_l = 1$

could cause the total cost to increase as shown by the following calculations:

$$\begin{aligned}\dot{\Phi} =& -k \times r \times \frac{\delta\alpha_m}{r} \times w_m + k \times r \times \frac{\delta\alpha_m}{r} \times (w_B + w_l) \\ & - r_B \times \frac{\delta}{r_B} \times w_l + r_B \times \frac{\delta}{r_B} \times w_s \\ =& -k\delta[\alpha_m w_m + (1-\alpha_m)(w_B + w_l)] + (k-1)\delta w_l \\ & + \delta(k w_B + w_s)\end{aligned}$$

which may be positive for $k > 1$. Formally, the above discussion leads us to further modify the update rule in (4) by introducing a new factor denoted by η_u^t

$$\dot{\alpha}_{u,v}^t = -\frac{\alpha_{u,v}^t \delta}{\eta_u^t r_u^t} \tag{5}$$

where $(u,v) \in \mathbb{E}$ but is not on the shortest path from u to t. For now, using $\eta_u^t = k$ leads to the same cost derivative as in Section IV-C, which is always no more than zero. In Section V, we will show how to calculate η_u^t in general and that for any network, this update rule for the split ratios (5) makes the total cost of the network always decrease, resulting in the split ratios converging to a set where every element of the set achieves the global optimum to the MCF problem, and therefore achieves optimal traffic engineering.

V. General Solution

We begin by defining η_u^t, the *branch cardinality*, as the product of the number of branches encountered in traversing the shortest path tree rooted at t from t to u. It makes sure that routers on the tree that are farther away from the destination shift traffic to the shortest path more conservatively than routers that are closer to the destination. At every iteration due to link-state flooding, each node u has the link-state information to run Dijkstra's algorithm to compute the shortest path tree to destination t. Here, additional care is required because every node has to locally arrive at the same shortest path tree to ensure that the algorithm proceeds as expected. Therefore, at any stage, while running Dijkstra's algorithm locally, if there is ambiguity as to which node should be added next, tie-breaking based on node index is used. In other words, if at any iteration there are multiple shortest paths to choose from, tie-breaking is used to ensure that all routers arrive at the same shortest path tree. The calculation of η_u^t proceeds as shown in Algorithm 1. For an illustration of how η_u^t is calculated, please refer to the example following the proof of Theorem 1.

Algorithm 1: Algorithm to calculate $\eta_u^t\{w_e, \forall e \in \mathbb{E}\}$

1: Compute shortest path tree for destination t using Dijkstra's algorithm with tie-breaking based on node index.
2: Traverse the tree from t to u.
3: Initialize $\eta_u^t \leftarrow 1$.
4: At every junction do $\eta_u^t \leftarrow \eta_u^t b$ where b is the number of branches from that junction.

We are now in a position to describe HALO. At each router u, it controls the evolution of $\alpha_{u,v}^t$, $\forall t \in \mathbb{V}$, $(u,v) \in \mathbb{E}$. Let $(u,\bar{v}) \in \mathbb{E}$ be on the shortest path from u to t. Then, HALO updates the split ratios as follows:

$$\text{If } r_u^t > 0, \qquad \dot{\alpha}_{u,v}^t = -\frac{\alpha_{u,v}^t \delta}{\eta_u^t r_u^t}, \qquad v \neq \bar{v} \tag{6}$$

$$\dot{\alpha}_{u,\bar{v}}^t = -\sum_{v:(u,v)\in\mathbb{E},\ v\neq\bar{v}} \dot{\alpha}_{u,v}^t \tag{7}$$

$$\text{else if } r_u^t = 0, \qquad \alpha_{u,v}^t = 0, \qquad v \neq \bar{v} \tag{8}$$

$$\alpha_{u,\bar{v}}^t = 1. \tag{9}$$

To prove the optimality of the above link-state hop-by-hop algorithm, we will need the following two lemmas. The first one, originally derived by Gallager [2], relates the node prices to the link weights for each destination t.

Lemma 1: $\sum_{u\in\mathbb{V}} D(u,t) q_u^t = \sum_{(u,v)\in\mathbb{E}} f_{u,v}^t w_{u,v}$.

It analytically states the intuitive idea that the total price of sending traffic to meet the demand in the network, as defined by the sum of the products of the traffic demand rate and the node price for each source node, is equal to the sum over all links of the price of sending traffic through each link. The next lemma describes how to calculate the rate of change of network cost [18].

Lemma 2:

$$\sum_{(u,v)\in\mathbb{E}} \dot{f}_{u,v}^t w_{u,v} = \sum_{u\in\mathbb{V}} \sum_{(u,v)\in\mathbb{E}} r_u^t \dot{\alpha}_{u,v}^t [w_{u,v} + q_v^t].$$

The above expression captures the fact that the change in network cost can either be expressed in terms of the change in the link flow rates, i.e., how each link affects the network cost or in terms of the change in the split ratios at each node, i.e., how each node affects the network cost. Now we are finally in a position to prove the main result of the paper, which is summarized in the following theorem.

Theorem 1: In a network, at every node u, for every destination t, let the evolution of the split ratios be defined by (6)–(9). Then, starting from any initial conditions, we have the following.

Convergence: α converges to the largest invariant set in $\{\alpha \mid \dot{\Phi}(f) = 0\}$.

Optimality: Any element of this set yields an optimal solution to the MCF problem.

Proof: We will prove the result in three steps. First, we will show that using HALO, $\dot{\Phi}(f) \leq 0$, i.e., the network cost decreases at each iteration. This is the key step of the whole proof. Then, we will use this result to invoke LaSalle's Invariance Principle for hybrid systems [24], i.e., systems that exhibit both discrete and continuous changes, to argue that α converges to the largest invariant set in $\{\alpha \mid \dot{\Phi}(f) = 0\}$. Lastly, we will establish that any element of this set is an optimal solution to the MCF problem.

Step 1 (Monotonicity): Note that

$$\dot{\Phi}(f) = \sum_{t\in\mathbb{V}} \sum_{(u,v)\in\mathbb{E}} \dot{f}_{u,v}^t w_{u,v} = \sum_{t\in\mathbb{V}} \dot{\Phi}^t(f)$$

where $\dot{\Phi}^t(f) = \sum_{(u,v)\in\mathbb{E}} \dot{f}^t_{u,v} w_{u,v}$ is the rate of change of the network cost as the flows to destination t change at each iteration. Consequently, if we show that $\dot{\Phi}^t(f) \leq 0$ for each destination t, then we have that $\dot{\Phi}(f) \leq 0$. From Lemma 2

$$\dot{\Phi}^t(f) = \sum_{(u,v)\in\mathbb{E}} \dot{f}^t_{u,v} w_{u,v} = \sum_{u\in\mathbb{V}} \sum_{(u,v)\in\mathbb{E}} r^t_u \dot{\alpha}^t_{u,v}[w_{u,v} + q^t_v].$$

The crux of Step 1 is to decompose the change in cost to each t after each iteration into separate components, each of which we can show to be at most 0. The idea is to group the terms from the summation derived in Lemma 2, using the "branches" of the shortest path tree rooted at t.

More precisely, we define a *branch* ($\mathcal{B}$) as the set of nodes on the path from a leaf node on the shortest path tree to the destination node t. Given the definition, it is easy to see that some intermediate nodes will be shared among multiple branches. Thus, the change in cost contributed by these nodes has to be appropriately divided among the different branches that pass through them.

We use the branch cardinality of the nodes to help us do this. Basically, when grouping terms, for a particular branch passing through an intermediate node u, we only take a fraction, $1/\pi^{\mathcal{B}}_u$, of the change in cost contributed by u, to be summed with that branch. For a given branch $\mathcal{B}$ and intermediate node u, we calculate $\pi^{\mathcal{B}}_u$ by requiring $\pi^{\mathcal{B}}_u \eta^t_u$ to be the same as the branch cardinality of the leaf router that defines $\mathcal{B}$. Consequently, $\pi^{\mathcal{B}}_u \eta^t_u$ will be the same for all $u \in \mathcal{B}$. One can check $\sum_{\mathcal{B}} \frac{1}{\pi^{\mathcal{B}}_u} = 1$, which confirms that the total contribution from node u is distributed over the different branches that pass through it. Formally, we have

$$\sum_{u\in\mathbb{V}} \sum_{(u,v)\in\mathbb{E}} r^t_u \dot{\alpha}^t_{u,v}[w_{u,v} + q^t_v] = \sum_{\forall\mathcal{B}} \sum_{u\in\mathcal{B}} \frac{1}{\pi^{\mathcal{B}}_u} \sum_{(u,v)\in\mathbb{E}} r^t_u \dot{\alpha}^t_{u,v}[w_{u,v} + q^t_v].$$

We continue the proof by restricting attention to an arbitrary branch $\mathcal{B}$, with n nodes numbered $1, \ldots, n$ from the leaf node to the destination and showing that, at each iteration, the change in cost for this branch is at most 0. For ease of notation, in what follows, we will use η to represent $\pi^{\mathcal{B}}_u \eta^t_u$, $\forall u \in \mathcal{B}$. Then, for any $u \in \{1, 2, \ldots, n-1\}$, we have the following important equation:

$$\frac{1}{\pi^{\mathcal{B}}_u} \sum_{(u,v)\in\mathbb{E}} r^t_u \dot{\alpha}^t_{u,v}[w_{u,v} + q^t_v] = -\frac{\delta}{\eta}(q^t_u - w_{u,u+1} - q^t_{u+1}). \tag{10}$$

Note if $r^t_u = 0$, following (8) and (9), the left-hand side of (10) is zero because $\dot{\alpha}^t_{u,v} = 0$. The right-hand side of (10) is also zero because $\alpha^t_{u,u+1} = 1$. If $r^t_u > 0$, (10) is still valid because

$$\begin{aligned}
&\frac{1}{\pi^{\mathcal{B}}_u} \sum_{(u,v)\in\mathbb{E}} r^t_u \dot{\alpha}^t_{u,v}[w_{u,v} + q^t_v] \\
&= -\frac{\delta}{\eta}\left(\sum_{(u,v)\in\mathbb{E}} \alpha^t_{u,v}[w_{u,v} + q^t_v] - \sum_{(u,v)\in\mathbb{E}} \alpha^t_{u,v}[w_{u,u+1} + q^t_{u+1}] \right) \\
&= -\frac{\delta}{\eta}\left(q^t_u - w_{u,u+1} - q^t_{u+1}\right).
\end{aligned}$$

Therefore

$$\begin{aligned}
&\sum_{u\in\mathcal{B}} \frac{1}{\pi^{\mathcal{B}}_u} \sum_{(u,v)\in\mathbb{E}} r^t_u \dot{\alpha}^t_{u,v}[w_{u,v} + q^t_v] \\
&= \sum_{u=1}^{n-1} -\frac{\delta}{\eta}\left(q^t_u - w_{u,u+1} - q^t_{u+1}\right) \\
&= -\frac{\delta}{\eta}[q^t_1 - w_{1,2} - \cdots - w_{n-1,n}] \\
&\leq 0.
\end{aligned}$$

The last inequality follows from the fact that the average price from the leaf router (node 1) to the destination (node n) that can be thought of as an average over paths from (2) has to be no less than the price of the shortest path. Note that this relationship holds with equality only when the node price of the leaf node is the same as the price of the shortest path, which means that all the traffic from every node in the branch to the destination is along shortest paths to the destination.

Having established that for any branch the change in cost is at most zero, we can say that the change in cost for the shortest path tree that is composed of multiple such branches is at most zero as well. The total change in cost is just the sum of the changes in cost over all destinations which will also be negative. That is, we then have

$$\dot{\Phi} = \sum_t \dot{\Phi}^t(f) = \sum_{(u,v)\in\mathbb{E}} \dot{f}^t_{u,v} \Phi'(f_{u,v}) \leq 0. \tag{11}$$

Step 2 (Convergence): Given the control laws, we have established that $\dot{\Phi}(f) \leq 0$. In order to show convergence, we will rely on the language of hybrid automata [24] to model the dynamics of our system. Specifically, our system is an example of a nonblocking, deterministic, and continuous hybrid automaton. Consequently, a generalization of LaSalle's Invariance Principle to hybrid automata [24] can be invoked to show that the set of split ratios converges to the largest invariant set within $\{\alpha \mid \dot{\Phi}(f) = 0\}$.

Step 3 (Optimality): The only way to have $\dot{\Phi}(f) = 0$ is if each $\dot{\Phi}^t(f) = 0$, which implies that the change in cost along each branch

$$\sum_{u\in\mathcal{B}} \frac{1}{\pi^{\mathcal{B}}_u} \sum_{\substack{(u,v)\in\mathbb{E} \\ \text{such that } u\in\mathcal{B}}} r^t_u \dot{\alpha}^t_{u,v}[w_{u,v} + q^t_v] = 0$$

for every t. From the preceding analysis, the change in cost along a branch $\mathcal{B}$ is zero only when all the traffic from the nodes that belong to the branch is being routed to the destination through shortest paths with respect to the link prices. Since this is a necessary and sufficient condition for optimality in MCF [21], the proof is complete. ■

Next, as an illustrative example to help understand the first step of the above proof, we consider a sample shortest path tree and perform the corresponding cost change calculations explicitly. Consider the shortest path tree of Fig. 3. The number of branches that we divide the tree into is determined by the number of leaf nodes. In this example, the shortest path tree rooted at t has 12 leaf routers, and consequently we will divide the summation into 12 branches. Following the algorithm for the

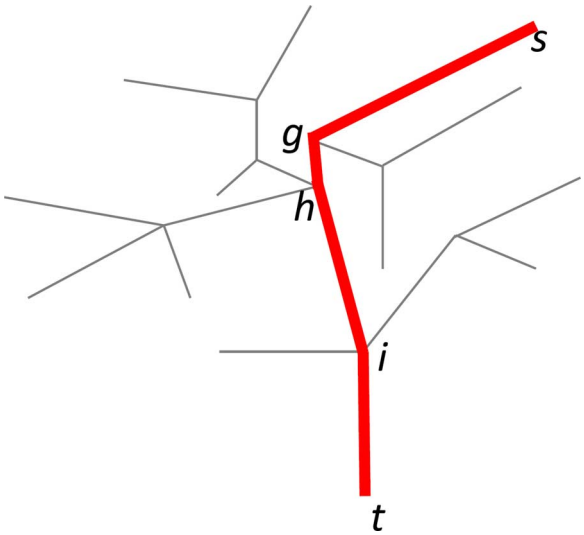


Fig. 3. Shortest path tree. Only the links in the shortest path tree for terminal t is shown, with the other links in the network not shown for ease of exposition.

calculation of η, we find $\eta_i^t = 1$, $\eta_h^t = 3$, $\eta_g^t = 9$, and $\eta_s^t = 18$. As noted in the proof, the change in the cost function over an iteration can be calculated using Lemma 2. In order to evaluate it, we further divide the terms in the summation and group them per branch. Recall from the proof that for the routers that are downstream to a leaf router in a branch, only a fraction of the change in the cost contributed by the downstream router is selected where the fraction is determined by the need to have the same η for all routers in the summation for a branch. The contribution to the change in the cost by the routers for the highlighted branch can be calculated as follows:

$$\begin{aligned}
&\sum_{u\in\mathcal{B}} \frac{1}{\pi_u^{\mathcal{B}}} \sum_{(u,v)\in\mathbb{E}} r_u^t \dot{\alpha}_{u,v}^t [w_{u,v} + q_v^t] \\
&\quad = -r_s^t \sum_{(s,v)\in\mathbb{E}} \frac{\alpha_{s,v}^t \delta}{\eta_s^t r_s^t} [w_{s,v} + q_v^t] + r_s^t \sum_{(s,v)\in\mathbb{E}} \frac{\alpha_{s,v}^t \delta}{\eta_s^t r_s^t} [w_{s,g} + q_g^t] \\
&\qquad - r_g^t \sum_{(g,v)\in\mathbb{E}} \frac{\alpha_{g,v}^t \delta}{2\eta_g^t r_g^t} [w_{g,v} + q_v^t] + r_g^t \sum_{(g,v)\in\mathbb{E}} \frac{\alpha_{g,v}^t \delta}{2\eta_g^t r_g^t} [w_{g,h} + q_h^t] \\
&\qquad - r_h^t \sum_{(h,v)\in\mathbb{E}} \frac{\alpha_{h,v}^t \delta}{6\eta_h^t r_h^t} [w_{h,v} + q_v^t] + r_h^t \sum_{(h,v)\in\mathbb{E}} \frac{\alpha_{h,v}^t \delta}{6\eta_h^t r_h^t} [w_{h,i} + q_i^t] \\
&\qquad - r_i^t \sum_{(i,v)\in\mathbb{E}} \frac{\alpha_{i,v}^t \delta}{18\eta_i^t r_i^t} [w_{i,v} + q_v^t] + r_i^t \sum_{(i,v)\in\mathbb{E}} \frac{\alpha_{i,v}^t \delta}{18\eta_h^t r_i^t} [w_{i,t}] \\
&\quad = -\frac{\delta}{\eta_s^t} [q_s^t - w_{s,g} - w_{g,h} - w_{h,i} - w_{i,t}] \leq 0.
\end{aligned}$$

VI. HALO Implementation

So far, we have described HALO and proved its convergence and optimality based on a fluid model. We have also assumed that "time" (τ) is measured in units of iterations, where each iteration itself requires link states to be flooded throughout the network, routers to update their split ratios, and traffic to converge to the new link rates. In this section, we discuss how to translate these assumptions into a practical implementation.

A. Discrete Implementation

Typically, once the analysis of the fluid model is available, with a small enough step-size, similar results should hold in a discrete implementation as well. Actually, in our model, because we measure time in terms of iterations of the algorithm, the physical time it takes to execute an iteration does not affect our results. What is important, however, is that in between iterations, the split ratio updates have to be made with a small enough step-size. This means that, in practice, our calculations are valid if the routers are synchronized and wait long enough between updates to see the changes reflected in the link rates.

Our description of HALO does not include a stopping criterion. Instead, in the fluid model, we rely on LaSalle's Invariance Principle for hybrid systems to prove that the dynamics converge to the optimal routing assignment. In order to approximate this result, diminishing step-sizes are required as we approach the optimal routing assignment. Barring that, our evaluations in Section VII show that with a small enough but constant step-size, the routes stabilize close to the optimal routing assignment. HALO also exhibits hybrid dynamics where sometimes, using a small enough step-size is not sufficient [25]. Fortunately, once again, our extensive numerical and experimental evaluations in Sections VII and VIII indicate that this is not the case and that the algorithm does in fact converge to the optimal solution even in a discrete implementation.

B. High-Frequency Link-State Updates

The physical time needed to complete an iteration directly impacts the actual time that the algorithm takes to find the optimal solution. In fact, the need to converge to the optimal routing assignment before the traffic changes means that routers are restricted in how long they have for each iteration. Fortunately, in many networks, it does not take very long to flood link states across them or to update the routes according to our calculations. Typically, end-to-end latencies range from hundreds of microseconds for data center networks [26] to tens of milliseconds for wide-area networks spanning the continental US. Combined with the fact that, depending on the step-size, several iterations might be needed to converge to the optimal solution, this means that HALO requires high-frequency link-state updates. The exact frequency depends on the number of iterations needed by HALO to find the optimal solution. Analytically bounding the required number of iterations remains open. Instead, we use the evaluations in Section VII to help us gauge the time HALO takes to converge for reasonable step-sizes and find that for our test cases, a couple of hundred iterations is sufficient to reach the optimal routing assignment. Also, in modern networks, the communication overhead of the updates is negligible. This is because, at each iteration, there are $|V|$ link-state updates that need to traverse $|E|$ edges, and the length of each message is at most a couple of hundred bytes.

Lastly, it is important to note that, traditionally, high frequency link-state updates have been avoided as they can cause existing routing algorithms to produce route oscillations. HALO, and the other iterative algorithms discussed earlier, avoid this problem by being conservative about the fraction of traffic they shift at each iteration. Specifically, when there are thousands of flows, route flapping can be avoided by shifting

different flows at each iteration to approximate the calculated split ratios. Thus, given enough flows, most flows can have enough time to complete before they are shifted again.

C. Splitting Traffic

In the fluid model, we assume that we can arbitrarily split traffic to reach the optimal routing solution. By appropriately selecting the step-sizes used to update the split ratios, in practice, we can approximate the fluid model pretty closely. However, this introduces undesirable packet-reordering to the network unless we are careful not to split individual flows. In practice, in a network with tens of thousands to hundreds of thousands of flows, we can still approximate the split ratios that we calculate by using them to divide the incoming flows among different output ports while keeping individual flows together.

D. Interaction With Single-Path Routing

Initial implementations of HALO might see it coexist with single-path routing schemes like OSPF. Assuming protocol-specific link weights, the routes change much less frequently at the OSPF routers compared to the HALO routers. This notion of time–scale separation means that the subset of routers running HALO executes the algorithm in between slower route changes due to OSPF. Hence, the "single-path" routers have a pruning effect on the network from the perspective of the HALO routers, i.e., the outgoing links that are not used by them are effectively not a part of the network topology. The HALO routers base their calculations on this reduced network to reach the optimal routing assignment for it, i.e., increasing the routers implementing HALO essentially increases the search space for finding a better routing assignment. See Section VII-E for numerical evidence.

VII. Numerical Evaluation

In this section, we study numerical evaluations of the performance of HALO from the point of view of optimality and rate of convergence to the optimal solution. We also present evidence of the adaptivity of the algorithm as the traffic changes as well as studies of the performance of HALO in asynchronous environments and its interaction with single-path routing protocols. The evaluations are primarily performed on three network topologies—the benchmark Abilene network (Fig. 4), a 4×4 Mesh network, and a two-level hierarchical 50 node network [4]. The 4×4 Mesh network is selected to study the effects of intermediate routing loops on the optimality of the algorithm as this topology is particularly prone to such loops, while the hierarchical network is selected to mimic larger networks with high-capacity backbone links and lower-capacity local links. An additional test is performed on an even larger randomly generated 100-node network in order to confirm that the algorithm converges quickly for larger networks [Fig. 9(c)]. Randomly generated traffic demands are used for the mesh network and the hierarchical network, while for the Abilene network, uniform traffic demand is used. In order to study the algorithms' performance, in all three cases, the demand is scaled up until at least one link in the network is close to saturation at the optimal solution. In our evaluations, we capture network demand through the average link utilization resulting from routing the input traffic.

Fig. 4. Abilene network.

A. Convergence

As expected, the speed of convergence depends on the step-size. In general, smaller step-sizes guarantee convergence of the algorithm to the optimal solution at the expense of speed of convergence. This is demonstrated to be the case in Fig. 5. However, as can be seen in Fig. 5(a) and (c), larger step-sizes quickly approach the optimal solution although they can be prone to oscillations that prevent convergence to optimality. Often, it is sufficient to come to some neighborhood of the optimal solution, and in such cases, exact convergence ceases to be an issue as small oscillations around the optimal solution are acceptable. In such situations, a larger step-size may be used. It is encouraging to note that in all our test cases, including for the larger 100-node network [Fig. 9(c)], the algorithm was fairly quick, converging to a small neighborhood of the optimal solution within a few hundred iterations. In general, from our evaluations we observe that the rate of convergence is faster farther away from the solution and slower as the routes approach the optimal solution.

Another factor that affects the rate of convergence of the algorithm is the load on the network. By increasing the input traffic rate, we pushed the average link utilization for the Abilene network to 59.5%, for the mesh network to 65.4% and for the hierarchical network to 27.2%. These values indicate the point at which further scaling up the demand for the given traffic pattern would exceed the capacity of at least one link in the network, even with optimal routing. From Fig. 6, we can see that the algorithm takes more iterations to converge to the optimal solution for more heavily loaded networks. Promisingly, even in such limiting cases, HALO converges to the optimal solution on the order of a thousand iterations. Given that today, link-state advertisements can be broadcast on the order of milliseconds [27], our evaluations indicate the possibility of convergence times of less than a second to a few seconds for the protocol on networks where transmission/propagation delay of the link-state advertisements is not a limiting factor.

B. Performance

In order to verify that the algorithm does in fact achieve the optimal solution, the optimal solution was calculated for the test networks by solving the corresponding MCF problem using cvx [28] for different network loads. The objective value obtained by using HALO matched the optimal solution for each test case as can be seen from Fig. 7(a)–(c). Also, as expected from theory, the intermediate routing loops produced while determining the optimal solution for the mesh network did not affect the optimality of the algorithm.

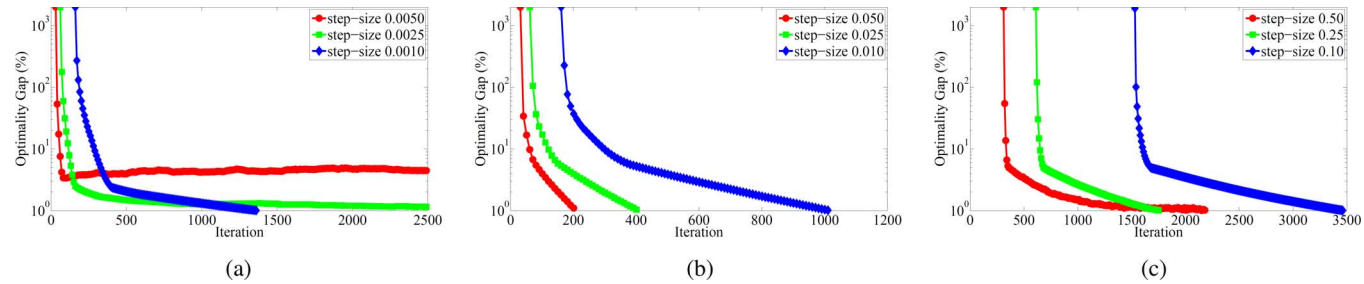


Fig. 5. (a) Evolution of the optimality gap for the Abilene network as the number of iterations increases with varying step-sizes (average link utilization $=$ 59.5%). (b) Evolution of the optimality gap for the 4×4 Mesh network as the number of iterations increases with varying step-sizes (average link utilization $=$ 60.8%). (c) Evolution of the optimality gap for the Hierarchical 50-node network as the number of iterations increases with varying step-sizes (average link utilization $=$ 27.2%).

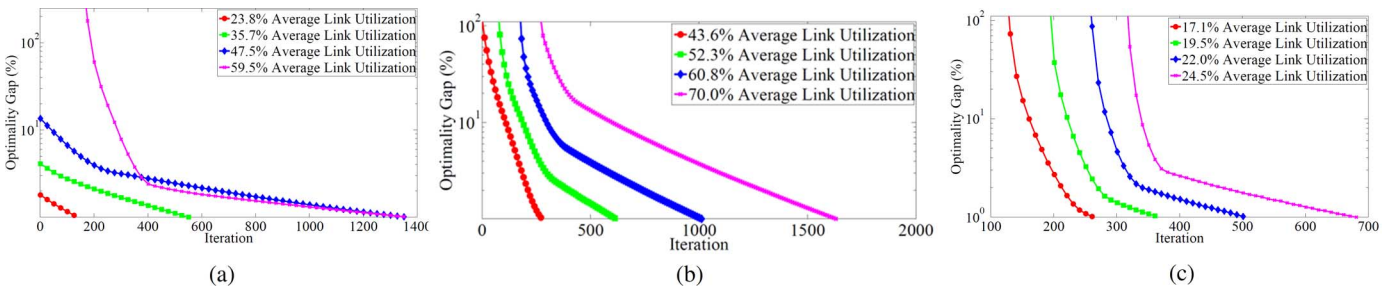


Fig. 6. (a) Evolution of the optimality gap for the Abilene network as the number of iterations increases with different network loads (step-size $=$ 0.001). (b) Evolution of the optimality gap for the 4×4 Mesh network as the number of iterations increases with different network loads (step-size $=$ 0.01). (c) Evolution of the optimality gap for the Hierarchical 50-node network as the number of iterations increases with different network loads (step-size $=$ 0.4).

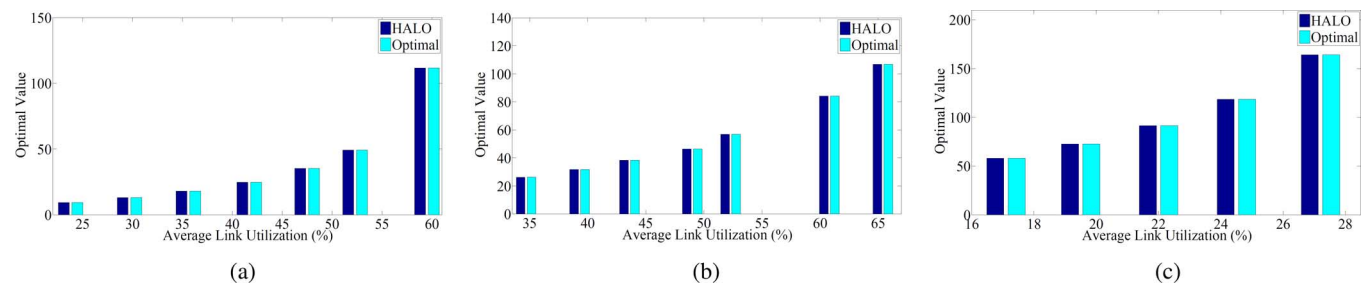


Fig. 7. (a) Abilene network optimal performance. (b) 4×4 Mesh network optimal performance. (c) Hierarchical 50-node network optimal performance. Centralized optimal value matched by HALO.

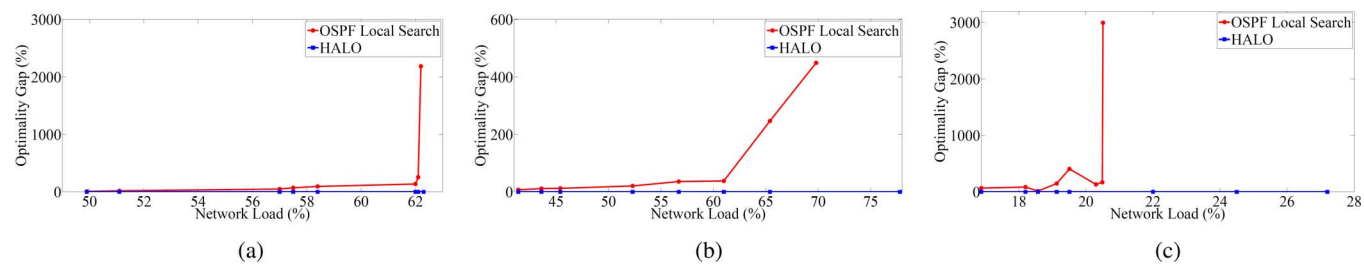


Fig. 8. (a) Abilene network algorithm comparison. (b) 4×4 Mesh network algorithm comparison. (c) Hierarchical 50-node network algorithm comparison. Relative performance of different algorithms for different network loads

The major advantage that HALO offers is the significant performance improvement that an optimal solution offers over techniques like OSPF-TE. In Fig. 8, we compare the performance of HALO to OSPF boosted by better weight settings obtained from the algorithms of the TOTEM toolbox [29] for demand matrices that placed increasing loads on the test networks. The local search algorithm used by TOTEM minimizes a piecewise-linear approximation of our convex cost function. The power of optimality is demonstrated by the performance improvements on the order of 1000% as the load on the network increases.

C. *Adaptivity*

Another attraction of HALO is that it dynamically adapts to changes in the traffic on the network. In Fig. 9(a), we plot the evolution of the optimality gap as the traffic matrix undergoes

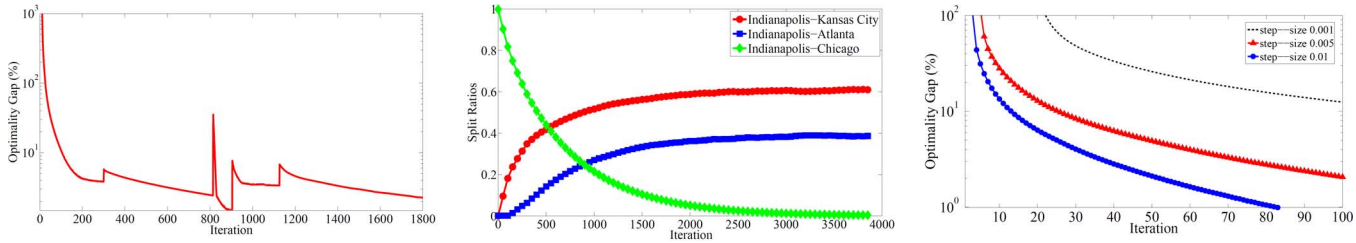


Fig. 9. (a) Evolution of the optimality gap for the Abilene Network as the number of iterations increases with varying demand matrices. (b) Evolution of the split ratios to Chicago, Kansas City, and Atlanta for traffic destined to LA at the Indianapolis node on the Abilene network. (c) Evolution of the optimality gap for a randomly generated 100-node network with varying step-sizes.

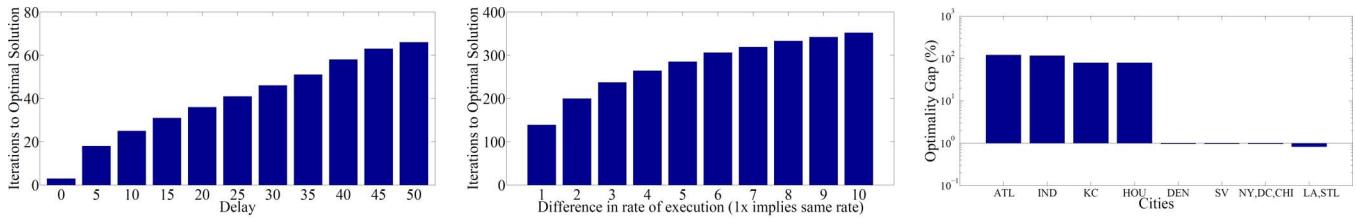


Fig. 10. (a) Asynchronous link-state updates. Iterations required to converge increase with increasing delay (step-size $= 0.1$). (b) Asynchronous execution. Iterations required to converge increase with increasing difference in rate of execution (step-size $= 0.001$). (c) Partial implementation. Results of partially implementing HALO on the Abilene network. Increasing number of nodes running HALO improves performance.

changes for the Abilene network under different network load conditions. In this example, after around 300 iterations, the network load is changed by changing 20% of the flows in the network. As can be seen, the algorithm quickly adapts, and the optimality gap increases very little before beginning to converge to the new optimal solution. The traffic pattern is again changed by varying 50% of the flows in the network after 800 iterations. This time, the change in the optimality gap is greater, but the convergence to the new optimal value is seen to be quicker. The traffic pattern in the network is changed two more times, and as can be observed from the figure, in both cases the algorithm quickly converges to thep new optimal solution.

A closely related concept to the adaptivity of the algorithm is the evolution of the split ratios at individual routers. Additionally, it serves as a visualization of HALO in action. We pick the Indianapolis node for the Abilene network and plot the evolution of the split ratios to Los Angeles in Fig. 9(b). For our test traffic, the initial suboptimal allocation of split ratios is quickly corrected as HALO reduces traffic sent to Chicago and increases traffic sent to Kansas City and Atlanta.

D. Asynchronous Implementation

In dynamic network environments, random delays can affect the time it takes for link-state information to reach every node in the network as required by the algorithm. Note that without synchronized link-state updates, facets of HALO like calculating the shortest path tree and η_u^t are affected. There are two ways to approach this problem. The first is to allow enough time between successive iterations of the algorithm so that every node has access to the most up-to-date link-state information. The second is to let the nodes execute HALO despite asynchronous link-state updates. It is also possible for asynchronous behavior to arise despite synchronized link-state updates due to some subset of the nodes executing the algorithm faster than the other nodes. We separately tested the performance of HALO, for asynchronous link-state updates and asynchronous executions, using uniform traffic on the Abilene network. The results of our evaluation, studying both type of asynchronous behavior, are presented in Fig. 10(a) and (b).

In both cases, in order to simulate asynchronous behavior, the nodes in the network were numbered and divided into two groups. For asynchronous link-state updates, at every iteration, the even-numbered nodes received link states without any delay, while the odd-numbered nodes received link states from the even-numbered nodes after a fixed delay. Consequently, at each execution of the algorithm, the two sets of nodes had different views of the network link states. The fixed delay was then varied to generate the results reported in Fig. 10(a). For asynchronous execution of HALO, the odd-numbered nodes were forced to execute the algorithm slower than the even-numbered nodes. The difference in the rate of execution was varied in order to obtain the results reported in Fig. 10(b). Note that different step-sizes had to be used to prevent oscillations in the two cases.

As can be seen, HALO still converged to within 1% of the optimal solution, which was used as a stopping criterion for our evaluation. Additionally, it is interesting to note that there is a steady increase in the number of iterations required by HALO as the delay in propagating the link states or the difference in the rate of executing HALO increases. While the results of this particular experiment are promising, more research is required to establish whether the observations noted above are a more general property of HALO.

E. Coexistence With Single-Path Protocol

We also used numerical evaluations to study how HALO works in conjunction with a single-path routing protocol. The same setup as described in Section VI is studied using a randomly generated traffic pattern on the Abilene network.

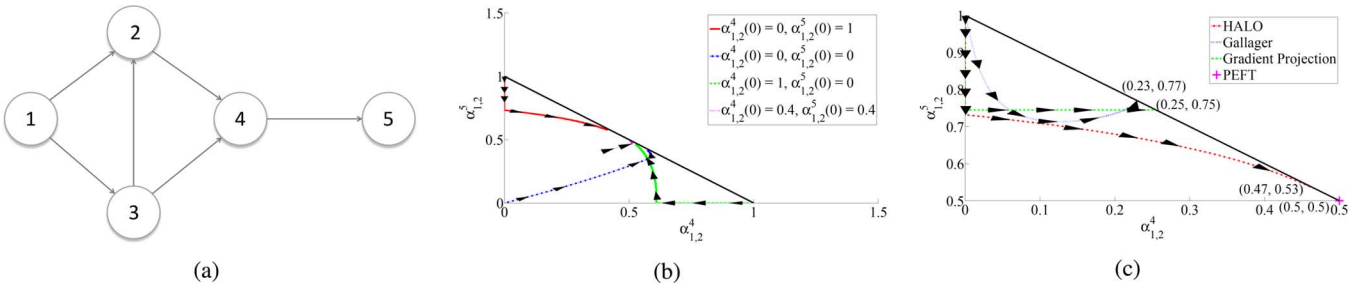


Fig. 11. (a) Network used to study optimal solutions reached by different algorithms and HALO's dependence on initial conditions [Solid line in (b) and (c) is $\alpha^4_{1,2} + \alpha^5_{1,2} = 1$]. (b) Initial split ratios affect both the number of iterations to converge to the optimal solution as well as the optimal solution. Parentheses in the legend indicate time 0. (c) Solutions and trajectories computed by different routing algorithms. PEFT does not have a trajectory since it is calculated centrally.

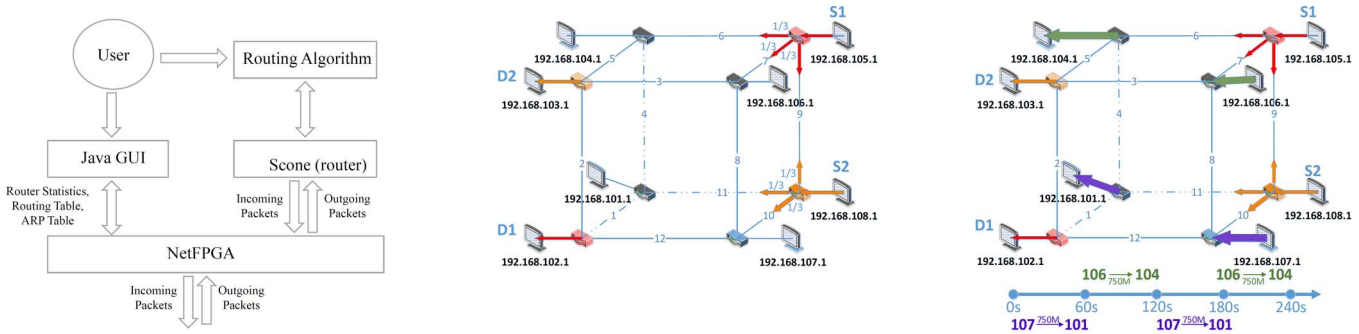


Fig. 12. NetFPGA experimental setup. (a) System diagram (b) Topology of the hardware testbed. Each link is bidirectional and can be replaced with two unidirectional links to get a directed graph. (c) Time-varying traffic.

The nodes were first ranked according to their degree and then added in that order to the set implementing the HALO protocol to obtain the results seen in Fig. 10(c). The degree-two nodes on the East and the West Coasts were added to the set together since by that point most of the performance gain had already been achieved. As can be seen, for each set implementing HALO, the algorithm converges to the optimal solution for the case where the remaining nodes are constrained to have only one outgoing link per destination. As expected, as the number of cities/nodes that run the algorithm increase, the performance of the network improves. However, it is also interesting how adding the first few higher-degree nodes to the set implementing HALO results in the bulk of the performance improvement.

F. Dependence on Initial Conditions

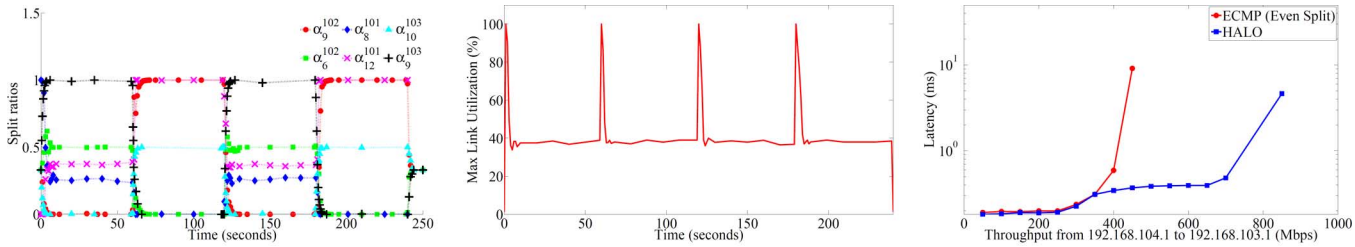
Next, using the network in Fig. 11(a), we explore how the initial split ratios influence the rate of the convergence of HALO to the optimal solution and the solution itself. Each link has capacity 5 units, and there are two demands $D(1,4) = D(1,5) = 2$ units. We start with initial routes (1–3–2–4, 1–2–4–5), (1–3–2–4, 1–3–4–5), (1–2–4, 1–3–2–4–5), and 60% through (1–3–4, 1–3–4–5), 40% through (1–2–4, 1–2–4–5). As can be seen from Fig. 11(b), each set of initial split ratios generates a different optimal solution, all of which satisfy $\alpha^4_{1,2} + \alpha^5_{1,2} = 1$, an optimality condition that follows from the fact that at optimum $f_{3,2} = 0$ and the resulting symmetry of the problem. Also, we find that each set of initial conditions needs a different number of iterations to converge to the optimal solution. With a step-size of 0.01 for HALO, listed in the same order as the initial routes, the numbers of iterations required to come within 0.1% of the optimal solution are 574, 347, 1004, and 179.

G. Different Algorithms Can End up With Different Split Ratios

As shown in Fig. 2, HALO follows a different trajectory from Gallager's algorithm in searching for an optimal solution. However, in that case, both algorithms converged to the same optimal solution. In general, because the MCF problem is strictly convex in link rates $(f_{u,v})$ and only convex in flow rates $(f^t_{u,v})$, there can be multiple optimal solutions in terms of the flow rates. We demonstrate this with the following example. Again, we use the topology in Fig. 11(a) with the same link capacities and demands used to evaluate the dependence of HALO on the initial split ratios. The initial routes supplied to the different algorithms are (1–3–2–4) and (1–2–4–5), i.e., $\alpha^4_{1,2} = 0$ and $\alpha^5_{1,2} = 1$. As can be seen from Fig. 11(c), each algorithm generates a different optimal solution, all of which again satisfy $\alpha^4_{1,2} + \alpha^5_{1,2} = 1$.

VIII. Experimental Evaluation

The results of the numerical evaluation, particularly those showing convergence with asynchronous link-state updates and executions, led us to test HALO on a hardware testbed. As shown in Fig. 12(b), we built a simple cube network of eight Dell PCs connected to NetFPGA 1G boards programmed to act as routers. We performed experiments to verify optimality, to evaluate performance as input traffic varied, as well as to measure network latency compared to Equal-Cost Multipath (ECMP) routing. Before describing our results, we give a brief overview of how we implemented HALO.



(a) (b) (c)

Fig. 13. NetFPGA experimental results. (a) Converging to optimal routing for time-varying traffic. (b) Max link utilization quickly reduced to optimal values. (c) Significant improvement in latency and maximum sustainable load over ECMP.

A. Implementing HALO on NetFPGA

We implemented HALO by modifying the Reference Router [30] project that converts the NetFPGA board to a single-path router. First, we altered the link-state update packets to carry link rates and set the link-states to broadcast every 250 ms for quick convergence. The link rates and the per-destination flow rates were obtained from the appropriate registers. Then, we programmed HALO to calculate split ratios that were stored in a modified routing table. We accomplished multipath routing by implementing a random number generator that selected the appropriate output port according to these split ratios. For the network cost function, we used $\sum_{u,v\in\mathbb{E}} f_{u,v}^2$, which gave us $2f_{u,v}$ as the price of each link.

A notable difference between the NetFPGA implementation and our model is the lack of synchronization between the routers. Thus, the NetFPGA testbed is used to provide physical verification of the numerical observations of the convergence of HALO in asynchronous environments.

B. Verifying Optimality

As illustrated in Fig. 12(b), we sent two video streams from $S1$ to $D1$ and $S2$ to $D2$. From symmetry, we expected the traffic to split equally over the three outgoing links at $S1$ and $S2$. Similarly, we calculated the optimal split ratios at the intermediate routers as well. Similar to the numerical evaluation, despite the lack of synchronization between the routers, the network converged to the optimal routing assignment about a second after we introduced the traffic streams.

C. Performance With Varying Input Traffic

Next, we added time-varying traffic to the preceding experiment using iPerf as shown in Fig. 12(c). HALO quickly adapted to the input traffic changes as can be seen in Fig. 13(a). Also, it distributed the larger time-varying traffic input more evenly over the network as captured by the reduction in the maximum link utilization in the network in Fig. 13(b).

D. Latency Compared to ECMP

In this experiment, we sent increasing traffic from 192.168.104.1 to 192.168.103.1, in addition to the video traffic from $S1$ to $D1$, using both HALO and Equal-Cost Multipath with equal traffic splits (ECMP). We also used a rate limiter to reduce the capacity of the NetFPGA boards to 475 Mb/s since otherwise the PC network interface cards (NICs), with a capacity of around 800 Mb/s, would be the bottleneck to sending extra traffic. Using ECMP, one of the paths from $S1$ to $D1$ shared link 5 with the direct path between 192.168.104.1 and 192.168.103.1. Consequently, the maximum traffic that we were able to send with ECMP between 192.168.104.1 and 192.168.103.1 was around 430 Mb/s after accounting for the video traffic. On the other hand, HALO adjusted the routes as needed to satisfy the increasing traffic demand based on link-state feedback, and we were able to send as much traffic as the NIC at 192.168.104.1 could support. The results of measuring the latency between 192.168.104.1 and 192.168.103.1 are plotted in Fig. 13(c). As can be seen, HALO not only supports more traffic than ECMP, but also offers much better latency over the larger range of traffic that it supports.

IX. Summary

In this paper, we developed HALO, the first link-state, hop-by-hop routing algorithm that optimally solves the traffic engineering problem for intradomain routing on the Internet. Furthermore, we showed that based on feedback from the link-state updates, the protocol automatically adapts to input traffic and topology changes by adjusting router split ratios. We also provided guidelines on implementing HALO by translating the theoretical model to a discrete implementation for numerical evaluations and then to a physical testbed built on NetFPGA boards. Importantly, although they did not satisfy the theoretical assumptions about continuous split ratio updates and synchronization between the routers, the numerical and experimental evaluations backed up our theoretical predictions about the performance and adaptivity of HALO. In terms of future directions, there are still interesting areas to be explored. For instance, the convergence rate of the algorithm needs to be analyzed. Another direction involves developing the theory behind the performance of algorithm in the absence of synchronous link-state updates and executions.

Acknowledgment

The authors thank Dr. D. Xu of AT&T for helpful discussions on PEFT and helping generate some simulation results, and P. Sharma, A. Sinha, N. Wu, and C.-H. Yu of Cornell University for helping obtain testbed results.

References

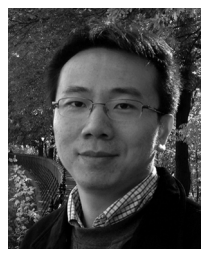
[1] N. Michael, A. Tang, and D. Xu, "Optimal link-state hop-by-hop routing," in *Proc. IEEE ICNP*, 2013, pp. 1–10.
[2] R. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Trans. Commun.*, vol. COM-25, no. 1, pp. 73–85, Jan. 1977.

[3] L. Fratta, M. Gerla, and L. Kleinrock, "The flow deviation method: An approach to store-and-forward communication network design," *Networks*, vol. 3, no. 2, pp. 97–133, 1973.
[4] B. Fortz and M. Thorup, "Increasing internet capacity using local search," *Comput. Optim. Appl.*, vol. 29, no. 1, pp. 13–48, Oct. 2004.
[5] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach, 5/E*. New York, NY, USA: Addison-Wesley, 2010.
[6] D. Bertsekas and E. Gafni, "Projected newton methods and optimization of multicommodity flows," *IEEE Trans. Autom. Control*, vol. AC-28, no. 12, pp. 1090–1096, Dec. 1983.
[7] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," *IEEE/ACM Trans. Netw.*, vol. 19, no. 6, pp. 1717–1730, Dec. 2011.
[8] A. Sridharan, R. Guerin, and C. Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," *IEEE/ACM Trans. Netw.*, vol. 13, no. 2, pp. 234–247, Apr. 2005.
[9] S. Srivastava, G. Agrawal, M. Pioro, and D. Medhi, "Determining link weight system under various objectives for OSPF networks using a lagrangian relaxation-based approach," *IEEE Trans. Netw. Service Manag.*, vol. 2, no. 1, pp. 9–18, Nov. 2005.
[10] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg, "Fast accurate computation of large-scale IP traffic matrices from link loads," in *Proc. ACM SIGMETRICS*, New York, NY, USA, 2003, pp. 206–217.
[11] D. Awduche, "MPLS and traffic engineering in IP networks," *IEEE Commun. Mag.*, vol. 37, no. 12, pp. 42–47, Dec. 1999.
[12] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," in *Proc. 20th Annu. IEEE INFOCOM*, 2001, vol. 3, pp. 1300–1309.
[13] D. Applegate and E. Cohen, "Making routing robust to changing traffic demands: Algorithms and evaluation," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1193–1206, Dec. 2006.
[14] M. Kodialam, T. V. Lakshman, J. Orlin, and S. Sengupta, "Oblivious routing of highly variable traffic in service overlays and IP backbones," *IEEE/ACM Trans. Netw.*, vol. 17, no. 2, pp. 459–472, Apr. 2009.
[15] T. Stern, "A class of decentralized routing algorithms using relaxation," *IEEE Trans. Commun.*, vol. COM-25, no. 10, pp. 1092–1102, Oct. 1977.
[16] C. E. Agnew, "On quadratic adaptive routing algorithms," *Commun. ACM*, vol. 19, no. 1, pp. 18–22, Jan. 1976.
[17] D. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1439–1451, Aug. 2006.
[18] F. Paganini and E. Mallada, "A unified approach to congestion control and node-based multipath routing," *IEEE/ACM Trans. Netw.*, vol. 17, no. 5, pp. 1413–1426, Oct. 2009.
[19] Y. Xi and E. Yeh, "Node-based optimal power control, routing, and congestion control in wireless networks," *IEEE Trans. Inf. Theory*, vol. 54, no. 9, pp. 4081–4106, Sep. 2008.
[20] P. Mahey, A. Ouorou, L. J. LeBlanc, and J. Chifflet, "A new proximal decomposition algorithm for routing in telecommunication networks," *Networks*, vol. 31, no. 4, pp. 227–238, 1998.
[21] D. Bertsekas and R. Gallager, *Data Networks*. Upper Saddle River, NJ, USA: Prentice-Hall, 1992.
[22] M. Wang, C. W. Tan, W. Xu, and A. Tang, "Cost of not splitting in routing: characterization and estimation," *IEEE/ACM Trans. Netw.*, vol. 19, no. 6, pp. 1849–1859, Dec. 2011.
[23] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge Univ. Press, 2004.
[24] J. Lygeros, K. Johansson, S. Simic, J. Zhang, and S. Sastry, "Dynamical properties of hybrid automata," *IEEE Trans. Autom. Control*, vol. 48, no. 1, pp. 2–17, Jan. 2003.
[25] J. Lygeros, K. Johansson, S. Simic, J. Zhang, and S. Sastry, "Continuity and invariance in hybrid automata," in *Proc. 40th IEEE Decision Control*, 2001, vol. 1, pp. 340–345.
[26] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," *Comput. Commun. Rev.* vol. 40, no. 4, pp. 63–74, Aug. 2010.
[27] Cisco Systems, Inc., San Jose, CA, USA, "OSPF link-state advertisement (LSA) throttling," 2012 [Online]. Available: http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/fsolsath.html
[28] CVX Reaserch, Inc., "CVX: Matlab software for disciplined convex programming, version 2.0 beta," Sep. 2012 [Online]. Available: http://cvxr.com/cvx
[29] J. Lepropre, S. Balon, and G. Leduc, "Totem: A toolbox for traffic engineering methods," presented at the Poster and Demo Session of IEEE INFOCOM Apr. 2006.
[30] Stanford University, Stanford, CA, USA, "NetFGPA reference router walkthrough," Aug. 2013 [Online]. Available: https://github.com/NetFPGA/netfpga/wiki/ReferenceRouterWalkthrough

Nithin Michael (S'06) received the B.S. degree (*summa cum laude*) in electrical and computer engineering from Drexel University, Philadelphia, PA, USA, in 2008, and the M.S. and Ph.D. degrees in electrical engineering with a minor in applied mathematics from Cornell University, Ithaca, NY, in 2012 and 2013, respectively.

His research is focused on the optimization of engineering networks.

Dr. Michael was the recipient of first Honors in ECE and the Highest Academic Achievement Award for graduating first in his class from the College of Engineering, Drexel University, in 2008. At Cornell, he was a Jacobs Fellow in 2008 and 2010.



Ao Tang (S'01–M'07–SM'11) received the B.E. in electronics engineering from Tsinghua University, Beijing, China, in 1999, and the M.S. and Ph.D. degrees in electrical engineering with a minor in applied and computational mathematics from the California Institute of Technology, Pasadena, CA, USA, in 2002 and 2006, respectively.

He is currently an Associate Professor with the School of Electrical and Computer Engineering, Cornell University, Ithaca, NY, USA, where he works on control and optimization of communication networks.

Dr. Tang received the following recent awards, the Cornell Engineering School Michael Tien '72 Excellence in Teaching Award in 2011, the Young Investigator Award from the Air Force Office of Scientific Research (AFOSR) in 2012, and the Presidential Early Career Award for Scientists and Engineers (PECASE) from the White House in 2012.