Link-State Routing with Hop-by-Hop Forwarding Can Achieve Optimal Traffic Engineering

Dahai Xu*
AT&T Labs - Research
dahaixu@research.att.com

Mung Chiang
Dept. of EE, Princeton University
chiangm@princeton.edu

Jennifer Rexford
Dept. of CS, Princeton University
jrex@cs.princeton.edu

***Abstract*—Link-state routing with hop-by-hop forwarding is widely used in the Internet today. The current versions of these protocols, like OSPF, split traffic evenly over shortest paths based on link weights. However, optimizing the link weights for OSPF to the offered traffic is an NP-hard problem, and even the best setting of the weights can deviate significantly from an optimal distribution of the traffic. In this paper, we propose a new link-state routing protocol, PEFT, that splits traffic over multiple paths with an exponential penalty on longer paths. Unlike its predecessor, DEFT [1], our new protocol provably achieves *optimal* traffic engineering while retaining the *simplicity* of hop-by-hop forwarding. A gain of 15% in capacity utilization over OSPF is demonstrated using the Abilene topology and traffic traces. The new protocol also leads to significant reduction in the time needed to compute the best link weights. Both the protocol and the computational methods are developed in a new conceptual framework, called Network Entropy Maximization, which is used to identify the traffic distributions that are not only optimal but also realizable by link-state routing.**

Keywords: Interior gateway protocol, traffic engineering, routing, OSPF, optimization, network entropy maximization.

I. Introduction

A link-state routing protocol has three components. First is *weight computation*: the network-management system computes a set of link weights through a periodic and centralized optimization. Second is *traffic splitting*: each router uses the link weights to decide traffic splitting ratios for every destination among its outgoing links. Third is *packet forwarding*: each router independently decides which outgoing link to forward a packet based only on its destination prefix, in order to realize the desired traffic splitting. The popularity of link-state protocols can be attributed to their ease of management; in particular, each router's decision on traffic splitting is conducted autonomously without further assistance from the network-management system, and each packet's forwarding decision is made in a hop-by-hop fashion without memory or end-to-end tunneling.

Such simplicity seems to carry a cost on optimality. In a procedure known as Traffic Engineering (TE), network operators minimize a convex cost function of the link loads, by tuning the link weights to be used by the routers. With Open Shortest Path First (OSPF), the major variant of link-state protocol in use today, computing the right link weights is NP-hard and even the best setting of the weights can deviate significantly from optimal TE [2]. Contrary to some popular belief, the optimal TE method in [3] is *not* distributed link-state routing, and the following question remains open: can a link-state protocol with hop-by-hop forwarding achieve optimal TE? This paper shows that the answer is in fact positive, by developing a new link-state protocol, Penalizing Exponential Flow-spliTting (PEFT), proving that it achieves optimal TE, and demonstrating that link weight computation for PEFT is highly efficient in theory and in practice.

*The work was done when Xu was in Dept. of EE, Princeton University.

TABLE I
Comparison of various TE schemes (new contributions in *italics*).

	Commodity Routing	**Link-State Routing**	
		OSPF	**PEFT**
Traffic Splitting	Arbitrary	Even among shortest paths	Exponential
Scalability	Low	High	High
Optimal TE	Yes	No	*Yes*
Complexity Class	Convex Optimization	NP Hard	*Convex Optimization*

In PEFT, packet forwarding is just the same as OSPF: destination-based and hop-by-hop. The key difference is in traffic splitting. OSPF splits traffic evenly among the shortest paths, and PEFT splits traffic along all paths but penalizes longer paths (i.e., paths with higher sums of link weights) exponentially. While this is a difference in how link weights are *used* in the routers, it also enables a change in how link weights are *computed* by the operator. It turns out that using link weights in the PEFT way achieves optimal traffic engineering. Using the Abilene topology and traffic traces, we observe a 15% increase in the efficiency of capacity utilization by PEFT over OSPF. Furthermore, exponential penalty in traffic splitting is the *only* penalty that can lead to this optimality result. The corresponding best link weights for PEFT can be efficiently computed: as efficiently as solving a linearly constrained concave maximization and much faster than the existing weight computation heuristics for OSPF.

Clearly, if the complexity of managing a routing protocol were not a concern, other approaches could be used to achieve optimal TE. One possibility is multi-commodity-flow type of routing, where an optimal traffic distribution is realized by dividing an arbitrary fraction of traffic over many paths. This can be supported by the forwarding mechanism in Multi-Protocol Label Switching (MPLS) [4]. However, optimality then comes with a cost for establishing many end-to-end *tunnels* to forward packets. Second, other studies explored more flexible ways to split traffic over shortest paths [3], [5], but these solutions do not enable routers to *independently* compute the flow-splitting ratios from link weights. Instead, a central management system must compute and configure the traffic-splitting ratios, and update them when the topology changes, sacrificing the main benefit of running a distributed link-state routing protocol. Clearly, there is a tension between optimal but complex routing or forwarding methods and the simple but to-date suboptimal link-state routing with hop-by-hop forwarding. Recent works [1], [6] attempted to attain optimality and simplicity simultaneously but neither proved optimality for TE nor developed sufficiently fast methods for computing link weights. A summary is provided in Table I.

There are several new ideas in this paper that enable a proof of optimality and a much faster computational beyond, for example, the theory and algorithm in DEFT [1]. One of these ideas is to develop both the traffic splitting and the weight computation methods from the conceptual framework of Network Entropy Maximization (NEM). As a proof technique, we will construct an optimization called NEM that is solved neither by the operator nor by the routers, but by us, the protocol developers. The optimality condition of NEM reveals the structure of hop-by-hop forwarding and is later used to guide both the router's traffic splitting and the operator's weight computation. In short, it turns out that a certain notion of entropy can identify those optimal traffic distributions that can be realized by link-state protocols.

The rest of the paper is organized as follows. Background on optimal traffic engineering is introduced in Sec. II. The theory of Network Entropy Maximization in Sec. III leads to the routing protocol PEFT in Sec. IV and the associated link weight computation algorithm in Sec. V. Extensive numerical experiments are then summarized in Sec. VI. Differences from our previous results [1] are summarized in Sec. VII, before we conclude with further observations and extensions in Sec. VIII. The key notation used in this paper is shown in Table II.

II. Background on Optimal TE

A. *Definitions of Optimality*

Consider a wireline network as a directed graph $G = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ is the set of nodes (where $N = |\mathbb{V}|$), $\mathbb{E}$ is the set of links (where $E = |\mathbb{E}|$), and link (u, v) has capacity $c_{u,v}$. The offered traffic is represented by a traffic matrix $D(s, t)$ for source-destination pairs indexed by (s, t).

The load $f_{u,v}$ on each link (u, v) depends on how the network decides to route the traffic. An objective function enables quantitative comparisons between different routing solutions in terms of the load on the links. Traffic engineering usually considers a link-cost function $\Phi(f_{u,v}, c_{u,v})$ that is a increasing function of $f_{u,v}$.

For example, $\Phi(f_{u,v}, c_{u,v})$ can be the link utilization $f_{u,v}/c_{u,v}$, and the objective of traffic engineering can be to minimize $\max_{(u,v)\in\mathbb{E}} \Phi(f_{u,v}, c_{u,v})$.

As another example, let $\Phi(f_{u,v}, c_{u,v})$ be a piecewise-linear approximation of the M/M/1 delay formula [7], e.g.,

TABLE II
Summary of Key Notation

Notation	Meaning
$D(s,t)$	Traffic demand from source s to destination t
$c_{u,v}$	Capacity of link (u,v)
$f_{u,v}$	Flow on link (u,v)
$\widetilde{c}_{u,v}$	Necessary capacity of link (u,v)
$f^t_{u,v}$	Flow on link (u,v) destined to node t
f^t_u	Total incoming flow (destined to t) at u
$w_{u,v}$	Weight assigned to link (u,v)
w_{min}	Lower bound of all link weights
d^t_u	The shortest distance from node u to node t. $d^t_t = 0$
$h^t_{u,v}$	Gap of shortest distance, $h^t_{u,v} \triangleq d^t_v + w_{u,v} - d^t_u$
$\Gamma(h^t_{u,v})$	Traffic splitting function

$$\Phi(f_{u,v}, c_{u,v}) = \begin{cases} f_{u,v} & f_{u,v}/c_{u,v} \le 1/3 \\ 3f_{u,v} - 2/3\,c_{u,v} & 1/3 \le f_{u,v}/c_{u,v} \le 2/3 \\ 10f_{u,v} - 16/3\,c_{u,v} & 2/3 \le f_{u,v}/c_{u,v} \le 9/10 \\ 70f_{u,v} - 178/3\,c_{u,v} & 9/10 \le f_{u,v}/c_{u,v} \le 1 \\ 500f_{u,v} - 1468/3\,c_{u,v} & 1 \le f_{u,v}/c_{u,v} \le 11/10 \\ 5000f_{u,v} - 16318/3\,c_{u,v} & 11/10 \le f_{u,v}/c_{u,v}, \end{cases} \tag{1}$$

and the objective is to minimize $\sum_{(u,v)} \Phi(f_{u,v}, c_{u,v})$.

More generally, we use "$\Phi(\{f_{u,v}, c_{u,v}\})$" to represent any increasing and convex objective function. The optimality of traffic engineering is with respect to this objective function.

At this point we can already observe that there is a "gap" between the objective of TE and the mechanism of link-state routing. Optimality is defined directly in terms of the traffic flows, whereas link-state protocols represent the paths indirectly in terms of link weights. Bridging this gap is one of the challenges that have prevented researchers from achieving optimal traffic engineering using link-state routing thus far.

B. *Optimal TE Via Multi-Commodity Flow*

Consider the following convex optimization problem: minimizing the TE cost function over flow conservation and link capacity constraints:

COMMODITY:

$$\min \quad \Phi(\{f_{u,v}, c_{u,v}\}) \tag{2a}$$

$$\text{s.t.} \quad \sum_{v:(s,v)\in\mathbb{E}} f^t_{s,v} - \sum_{u:(u,s)\in\mathbb{E}} f^t_{u,s} = D(s,t), \forall s \neq t \tag{2b}$$

$$f_{u,v} \triangleq \sum_{t\in\mathbb{V}} f^t_{u,v} \le c_{u,v}, \forall (u,v) \tag{2c}$$

$$\text{vars.} \quad f^t_{u,v}, f_{u,v} \ge 0. \tag{2d}$$

The above multi-commodity problem can be readily solved in polynomial-time, where the flow destined to a single destination is treated as a commodity, and $f^t_{u,v}$ is amount of flow on link (u, v) destined to node t [1].

The resulting solution, however, may not be realizable through link-state routing and hop-by-hop forwarding. Indeed,

[1] To prevent bandwidth waste, we can eliminate flow loop in the optimal routing with a $O(E \log N)$-time algorithm for each commodity [8]. The loop-free property is important in designing link-state routing [3] as demonstrated later in Sec. V.

for a network with N nodes and E links, the multi-commodity-flow solution may require up to $O(N^2E)$ tunnels (i.e., explicit routing) [9], making it difficult to scale. In contrast, link-state routing is much simpler, requiring only $O(E)$ parameters (i.e., one per link).

Furthermore, while it is true that, from the solution of the COMMODITY problem, a set of link weights can be computed such that all the commodity flow will be forwarded along the shortest paths [3], [5], the flow-splitting ratios among these shortest paths are *not* related to the *link weights*, forcing the operator to specify up to $O(NE)$ *additional* parameters (one parameter on each link for each destination) as the flow-splitting ratios for all the routers.

The rest of this paper shows that optimal traffic engineering can, in fact, be achieved using only E link weights.

III. Theoretical Foundations: NEM

In this section, we present the theory of realizing optimal TE with link-state protocols. We first compute the minimal load that each link must carry to achieve optimal traffic distribution, then examine all the traffic splitting choices subject to necessary (minimal) link capacities. The traffic splitting configurations that is realizable with hop-by-hop forwarding can be picked out by exploiting a property it has: maximizing a weighted sum of the entropies of traffic splitting vectors. In addition, the corresponding link weights can be found by solving the new optimization problem using gradient projection. It is important to realize that the proposed NEM framework developed in this section is used to *design* the protocol—the NEM problem itself is *not* solved by the operator or routers. It is constructed as a proof technique and an intermediate step towards the results in the next two sections.

A. Necessary Capacity

Given the traffic matrix and the objective function, the solution to the COMMODITY problem (2) provides the optimal distribution of traffic. We represent the resulting flow on each link (u, v) as the *necessary capacity* $\widetilde{c}_{u,v} \triangleq f_{u,v}$ (or $\widetilde{\boldsymbol{c}}$ as a vector). The necessary capacity is a minimal [2] set of link capacities to realize optimal traffic engineering.

There could be numerous ways of traffic splitting that realize optimal TE. If we replace link capacity $c_{u,v}$ in COMMODITY (2) with the necessary capacity $\widetilde{c}_{u,v}$, we are free to impose another objective function to pick out a particular optimal solution to the original problem. A key challenge here is to design a new objective function, purely for the purpose of protocol development, such that the resulting routing of flow can be realized *distributively with link-state routing protocols*.

B. Network Entropy Maximization

Denote $P_{s,t}$ as the set of paths from s to t, and $x^i_{s,t}$ as the probability (fraction) of forwarding a packet of demand $D(s, t)$ to the i-th path ($P^i_{s,t}$). Obviously, $\sum_i x^i_{s,t} = 1$. To be realized with hop-by-hop forwarding, the values of $x^i_{s,t}$ should satisfy (3) below where $w_{u,v}$ is the weight assigned to link (u, v), and $g(\cdot)$ is a known function for all the routers.

$$\frac{x^i_{s,t}}{x^j_{s,t}} = \frac{g\left(\sum_{(u,v)\in P^i_{s,t}} w_{u,v}\right)}{g\left(\sum_{(u,v)\in P^j_{s,t}} w_{u,v}\right)}. \tag{3}$$

We find that the set of values of $x^i_{s,t}$ satisfying (3) maximizes a "network entropy" defined as follows. Consider the entropy function $z(x^i_{s,t}) = -x^i_{s,t} \log x^i_{s,t}$ for source-destination pair (s, t). The weighted sum, $\sum_{s,t}\left(D(s,t)\sum_{P^i_{s,t}} z(x^i_{s,t})\right)$, is defined as the network entropy. [3]

Now we define the Network Entropy Maximization (**NEM**) problem under the necessary capacity constraints as follows:

NEM:

$$\max \quad \sum_{s,t}\left(D(s,t)\sum_{P^i_{s,t}} z(x^i_{s,t})\right) \tag{4a}$$

$$\text{s.t.} \quad \sum_{s,t,i:(u,v)\in P^i_{s,t}} D(s,t)x^i_{s,t} \leq \widetilde{c}_{u,v}, \forall (u,v) \tag{4b}$$

$$\sum_i x^i_{s,t} = 1, \forall s,t \tag{4c}$$

$$\text{vars.} \quad x^i_{s,t} \geq 0. \tag{4d}$$

From the optimal solution of the COMMODITY problem, we know the feasibility set of NEM is non-empty. For a concave maximization over a non-empty, compact constraint set, there exist globally optimal solutions to NEM.

C. Solve NEM by Dual Decomposition

We will connect the characterization of optimal solutions to NEM with hop-by-hop forwarding and exponential penalty. Towards that end, and to provide a foundation for link weight computation in Sec. V, we first investigate the Lagrange dual problem of NEM and a dual-gradient-based solution.

Denote dual variables for constraints (4b) as $\lambda_{u,v}$ for link (u, v) (or $\boldsymbol{\lambda}$ as a vector). The maximization of the Lagrangian over $\boldsymbol{x}$ can be solved as a TRAFFIC-DISTRIBUTION problem (5):

TRAFFIC-DISTRIBUTION:

$$\max \quad \sum_{(u,v)\in\mathbb{E}} \lambda_{u,v}\widetilde{c}_{u,v} + \sum_{s,t}\left(D(s,t)\sum_{P^i_{s,t}} z(x^i_{s,t})\right) - \sum_{(u,v)\in\mathbb{E}} \lambda_{u,v}\left(\sum_{s,t,i:(u,v)\in P^i_{s,t}} D(s,t)x^i_{s,t}\right) \tag{5a}$$

$$\text{s.t.} \quad \sum_i x^i_{s,t} = 1. \tag{5b}$$

Then, the dual problem can be solved by using the gradient projection algorithm as follows for iterations indexed by q,

[2]But may not be the minimum capacity. $\widetilde{\boldsymbol{c}}$ is minimal if $\nexists \widetilde{\boldsymbol{c}'} : \widetilde{\boldsymbol{c}'} \neq \widetilde{\boldsymbol{c}} \wedge \widetilde{\boldsymbol{c}'} \preceq \widetilde{\boldsymbol{c}}$ whereas $\widetilde{\boldsymbol{c}}$ is the minimum if $\forall \widetilde{\boldsymbol{c}'} : \widetilde{\boldsymbol{c}} \preceq \widetilde{\boldsymbol{c}'}$.

[3]The physical interpretation of entropy for IP routing and the uniqueness of choosing the entropy function to pick out the right flow distributions are presented in [10].

$$\begin{aligned}&\lambda_{u,v}(q+1)\\&=\left[\lambda_{u,v}(q)-\alpha(q)\left(\widetilde{c}_{u,v}-\textstyle\sum_{s,t,i:(u,v)\in P^i_{s,t}} D(s,t)x^i_{s,t}(q)\right)\right]^+\\&=\left[\lambda_{u,v}(q)-\alpha(q)\left(\widetilde{c}_{u,v}-f_{u,v}(q)\right)\right]^+,\quad\forall(u,v)\in\mathbb{E}.\end{aligned}\tag{6}$$

where $\alpha(q)>0$ is the step size, $x^i_{s,t}(q)$ are solutions of the TRAFFIC-DISTRIBUTION problem (5) for a given $\boldsymbol{\lambda}(q)$, and $f_{u,v}(q)$ is the total flow on link (u,v).

After the above dual decomposition, the following result can be proved with standard convergence analysis for gradient algorithms [11]:

Lemma 1: By solving the TRAFFIC-DISTRIBUTION problem for the NEM problem and the dual variable update (6), $\boldsymbol{\lambda}(q)$ converge to the optimal dual solutions $\boldsymbol{\lambda}^*$ and the corresponding primal variables $\boldsymbol{x}^*$ are the globally optimal primal solutions of (4).

D. Solve TRAFFIC-DISTRIBUTION Problem

Note that, the TRAFFIC-DISTRIBUTION problem is also separable, i.e., the traffic splitting for each demand across its paths is independent of the others since they are not coupled together with link capacity constraint (4b). So we can solve a subproblem (7) below for each demand $D(s,t)$ separately:

DEMAND-DISTRIBUTION for $D(s,t)$:

$$\max\quad D(s,t)\sum_{P^i_{s,t}} z(x^i_{s,t})\tag{7a}$$

$$-\sum_{(u,v)\in\mathbb{E}}\lambda_{u,v}\left(\sum_{i:(u,v)\in P^i_{s,t}} D(s,t)x^i_{s,t}\right)$$

$$\text{s.t.}\quad\sum_i x^i_{s,t}=1.\tag{7b}$$

We write the Lagrangian associated with the DEMAND-DISTRIBUTION subproblem in (8).

$$\begin{aligned}&L^r(\boldsymbol{x}_{s,t},\mu_{s,t})\\=\;&\left(D(s,t)\textstyle\sum_{P^i_{s,t}} z(x^i_{s,t})\right)-\mu_{s,t}(\sum_i x^i_{s,t}-1)\\&-\textstyle\sum_{(u,v)\in\mathbb{E}}\lambda_{u,v}(\sum_{i:(u,v)\in P^i_{s,t}} D(s,t)x^i_{s,t})\end{aligned}\tag{8}$$

where $\mu_{s,t}$ is the Lagrangian variable associated with (7b).

According to Karush-Kuhn-Tucker (KKT) conditions [12], at the optimal solution of the DEMAND-DISTRIBUTION subproblem, we have

$$z'(x^{i^*}_{s,t})-\textstyle\sum_{(u,v)\in P^i_{s,t}}\lambda_{u,v}-\frac{\mu^*_{s,t}}{D(s,t)}=0.\tag{9}$$

For the entropy function, $z(x)=-x\log x$, $z'(x)=-1-\log x$, we have

$$x^{i^*}_{s,t}=e^{-(\sum_{(u,v)\in P^i_{s,t}}\lambda_{u,v}+\frac{\mu^*_{s,t}}{D(s,t)}+1)}.\tag{10}$$

where $x^{i^*}_{s,t},\mu^*_{s,t}$ are the values of the $x^i_{s,t},\mu_{s,t}$ respectively at the optimal solution.

Then for two paths i, j from s to t, we have

$$\frac{x^{i^*}_{s,t}}{x^{j^*}_{s,t}}=\frac{e^{-(\sum_{(u,v)\in P^i_{s,t}}\lambda_{u,v})}}{e^{-(\sum_{(u,v)\in P^j_{s,t}}\lambda_{u,v})}}.\tag{11}$$

If we use $\lambda_{u,v}$ as the weight $w_{u,v}$ for link (u,v), the probability of using path $P^i_{s,t}$ is inversely proportional to the exponential value of its path length. It is important to observe at this point that, since (11) has no factor of $\mu^*_{s,t}$, an intermediate router can ignore the source of the packet in forwarding. Equally importantly, from (6), in iteration q, the procedure of link price (weight) updating does not need the values of $x^i_{s,t}(q)$. Instead, it just needs $f_{u,v}(q)$, the aggregated bandwidth usage. We will show how to calculate it efficiently in Sec. V-B.

Now, combining the optimality results in Sec. II-B and Lemma 1 with the distributed nature of (11), we have

Theorem 1: Optimal traffic engineering for a given traffic matrix can be realized with link weights using exponential flow splitting (11).

IV. A New Link-State Routing Protocol: PEFT

In this section, we translate the theoretical results in Sec. III into a new link-state routing protocol run by routers. Each router makes an *independent* decision on how to forward traffic to a destination (i.e., flow-splitting ratios) among its outgoing links using *only* the link weights. We first present PEFT from (11), and summarize the notation of traffic-splitting function [1] for calculating flow-splitting ratios. Then for a PEFT flow, we show an efficient way to calculate its traffic-splitting function, which can be approximated to further simplify the computation of traffic splitting in practice.

A. PEFT

Based on (11), we propose a new link-state routing protocol, called Penalizing Exponential Flow-spliTting (**PEFT**). The fraction of the traffic (from u to t) distributed across the i-th path (or probability of forwarding a packet), $x^i_{u,t}$, is inversely proportional to the exponential value of its path length $p^i_{u,t}$ (sum of $w_{u,v}$ of the links along the path), as shown in (12).

$$\text{PEFT:}\qquad x^i_{u,t}=\frac{e^{-p^i_{u,t}}}{\sum_j e^{-p^j_{u,t}}}.\tag{12}$$

Theorem 1 in Sec. III shows PEFT can achieve optimal TE. A PEFT flow can be realized with hop-by-hop forwarding. For the sample network in Fig. 1, for the two paths from s to t, $s\rightarrow u\rightarrow a\rightarrow t$ and $s\rightarrow u\rightarrow b\rightarrow t$, and two paths from u to t, the flows on them for PEFT (12) satisfy (13).

$$f_{s\rightarrow u\rightarrow a\rightarrow t}:f_{s\rightarrow u\rightarrow b\rightarrow t}=f_{u\rightarrow a\rightarrow t}:f_{u\rightarrow b\rightarrow t}\tag{13}$$

Therefore, router u can treat the packets from different sources (e.g. s or u) equally by forwarding them among the outgoing links with precalculated splitting ratios. Further discussions can be found in [10].

As a link-state routing protocol, we need to define the traffic splitting function for PEFT as follows.

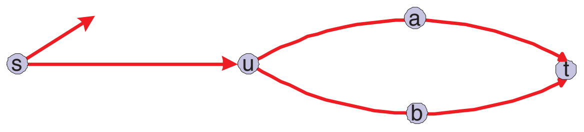


Fig. 1. Realize a PEFT flow using hop-by-hop forwarding

B. Review: Traffic Splitting Function

The notation of traffic-splitting (allocation) function was introduced in [1] to succinctly describe link-state routing protocols. In a directed graph, each unidirectional link (u, v) has a single, configurable weight $w_{u,v}$. Based on a complete view of the topology and link weights, a router can compute the shortest distance d_u^t from any node u to node t; $d_v^t + w_{u,v}$ represents the distance from u to t when routed through neighboring node v. *Shortest distance gap*, $h_{u,v}^t$, is defined as $d_v^t + w_{u,v} - d_u^t$, which is always greater than or equal to 0. Then, (u, v) lies on a shortest path to t if and only if $h_{u,v}^t = 0$. Traffic-splitting function $(\Gamma(h_{u,v}^t))$ indicates the relative amount of traffic destined to t that node u will forward via outgoing link (u, v) [4]. Let f_u^t denote the total incoming flow (destined to t) at node u (including the bypassing flow and self-originated flow). The total outgoing flow of traffic (destined to t) traversing link (u, v), $f_{u,v}^t$, can be computed as follows:

$$f_{u,v}^t = f_u^t \frac{\Gamma(h_{u,v}^t)}{\sum_{(u,j)\in\mathbb{E}} \Gamma(h_{u,j}^t)}. \tag{14}$$

Consistent with hop-by-hop forwarding, u splits the traffic over the outgoing links without regard to the source node or the incoming link where the traffic arrived. Implementation of PEFT on both data-plane and control-plane can be readily accomplished using today's technology, as discussed in [1].

C. Exact Traffic Splitting Function for PEFT

The traffic splitting function for PEFT can be calculated in polynomial time. From the definition of PEFT (12), more traffic should be sent along an outgoing link used by more paths and the paths should be treated differently based on their path lengths. To compute the traffic splitting on each outgoing link, we first define a positive real number Υ_u^t as the "equivalent number" of shortest paths from node u to destination t, and let $\Upsilon_t^t \triangleq 1$.

In a PEFT flow, we have

$$\begin{aligned}
\Upsilon_u^t &\triangleq \textstyle\sum_i e^{-(p_{u,t}^i - d_u^t)} \\
&= \sum_{(u,v)\in\mathbb{E}} \left(\sum_{j:(u,v)\in P_{u,t}^j} e^{-(p_{u,t}^j - w_{u,v} - d_v^t + d_v^t + w_{u,v} - d_u^t)} \right) \\
&= \sum_{(u,v)\in\mathbb{E}} \left(e^{-(d_v^t + w_{u,v} - d_u^t)} \sum_{j:(u,v)\in P_{u,t}^j} e^{-(p_{u,t}^j - w_{u,v} - d_v^t)} \right) \\
&= \sum_{(u,v)\in\mathbb{E}} \left(e^{-h_{u,v}^t} \Upsilon_v^t \right)
\end{aligned} \tag{15}$$

The recursive relationship represented in (15) can be used in the following way: $e^{-h_{u,v}^t}\Upsilon_v^t$ is an "equivalent number" of shortest paths from u to t for those paths bypassing link (u, v) and the router should distribute the traffic from u on link (u, v) in proportion to $e^{-h_{u,v}^t}\Upsilon_v^t$. Then we have an exact traffic splitting function [5] for PEFT at link (u, v):

$$\Gamma_{PX}(h_{u,v}^t) = \Upsilon_v^t e^{-h_{u,v}^t} \tag{16}$$

To enable hop-by-hop forwarding, each router needs to independently calculate $\Gamma_{PX}(h_{u,v}^t)$ for all node pairs. Then each router first computes the all-pairs shortest paths, using, e.g., the Floyd-Warshall algorithm with time complexity $O(N^3)$ [13], and calculates the values of $e^{-h_{u,v}^t}$. Then for each destination t, to compute the values of Υ_u^t, each router needs to solve N linear equations (15), which requires O(N^3) time [13]. Thus the total complexity is O(N^4).

D. Traffic Splitting Function for Downward PEFT

To prevent loops in link-state routing, packets are usually forwarded along a "downward path" where the next hop is closer to destination. This inspires the following *Downward PEFT*, whose traffic splitting function is $\Gamma_{PD}(h_{u,v}^t)$ [6]:

$$\Gamma_{PD}(h_{u,v}^t) = \begin{cases} \Upsilon_v^t e^{-h_{u,v}^t} & \text{if } d_u^t > d_v^t, \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

$\Gamma_{PD}(h_{u,v}^t)$ can approximate $\Gamma_{PX}(h_{u,v}^t)$ and further simplify the computation of Υ_u^t and traffic splitting as discussed below and utilized in Sec. V-C.

We consider each destination t independently. After temporarily removing link (u, v) where $d_u^t \le d_v^t$ since there is no flow on it, we get an acyclic network and do topological sorting on the remaining network. Proceeding through the nodes u in *increasing* topological order (starting with destination t), we compute the value of Υ_u^t using (15). For each destination, topology sorting requires $O(N + E)$ time, and summarizing the Υ_u^t across the outgoing links requires $O(N + E)$ time. Thus, the total time complexity to calculate Υ_u^t is $O(N^3 + N(N + E)) = O(N^3)$.

In general, downward PEFT does not provably achieve optimal TE, although it comes extremely close to optimal TE in practice, with the associated link weight computation even faster than that for exact PEFT. In the case where the

[4]For example, the traffic-splitting function for even-splitting across shortest paths (e.g., OSPF) is

$$\Gamma_O(h_{u,v}^t) = \begin{cases} 1 & \text{if } h_{u,v}^t = 0, \\ 0 & \text{if } h_{u,v}^t > 0. \end{cases}$$

[5]P in the subscript emphasizes that the calculation of traffic splitting considers the paths towards destination, and X means the exactness.

[6]D in the subscript emphasizes "downward".

lower bound of all link weights, w_{min}, is large enough, the downward PEFT is same as exact PEFT [7].

V. Link Weight Computation for PEFT

The last section described traffic splitting under PEFT. A new way to use link weights also means the network operator needs a new way to compute, centrally and off-line, the optimal link weights. It turns out that the NP-hard problem of link weight computation in OSPF can be turned into a convex optimization when link weights are used by PEFT. To do that, we will convert the iterative method of solving the NEM problem in Sec. III into a simple and efficient algorithm. We first present an algorithm that iteratively chooses a tentative set of link weights and evaluates the corresponding traffic distribution by simulating the exact PEFT traffic splitting run by the routers. From Theorem 1, the algorithm is guaranteed to converge to a set of link weights, which realizes optimal TE with PEFT. To further speed up the calculation, the traffic distribution with exact PEFT for each iteration can be *approximated* with downward PEFT. The simulation in Sec. VI show that such an approximation is very close to optimal and provides substantial speedup in practice.

A. Algorithm Framework for Optimizing Link Weights

The iterative algorithm consists of two main parts:

1) Computing the optimal traffic distribution (necessary capacities) for a given traffic matrix by solving the COMMODITY problem (2).
2) Computing the link weights that would achieve the optimal traffic distribution.

Starting with an initial setting of link weights, the algorithm (see Algorithm 1) repeatedly updates the link weights until the load on each link is the same as the necessary capacity. Each setting of the link weights corresponds to a particular way of splitting the traffic over a set of paths. The *Traffic_Distribution* procedure computes the resulting link loads $f_{u,v}$, based on the traffic matrix. Then, the *Link_Weight_Update* procedure (see Algorithm 2) increases the weight of each link (u,v) linearly if the traffic exceeds the necessary capacity, or decreases it otherwise. The parameter α is a positive step size, which can be constant or dynamically adjusted; we find that setting α to the reciprocal of the maximum necessary link capacity ($\frac{1}{\max \widetilde{c}_{u,v}}$) performs well in practice. Algorithm 1 is guaranteed to converge to the global optimal solution as stated in Lemma 1.

In terms of computational complexity, we know that COMMODITY can be solved efficiently. The complexity of Algorithm 2 is $O(E)$. The remaining question is how to solve the subproblem Traffic_Distribution($\boldsymbol{w}$) efficiently.

[7]For link (u,v), if the shortest distance to t of u, $d_u^t \leq d_v^t$, then $h_{u,v}^t = d_v^t + w_{u,v} - d_u^t \geq w_{u,v}$ and $\Gamma_{PX}(h_{u,v}^t) \leq \Upsilon_v^t e^{-w_{u,v}}$, and the flow destined to t on (u,v) is close to 0 if $w_{u,v}$ is large enough, e.g., $e^{-10} \approx 0.005\%$. Therefore, most flow in PEFT always makes forward progress towards the destination, i.e., from router u with larger d_u^t to router v with smaller d_v^t.

```
1: Compute necessary capacities $\widetilde{\boldsymbol{c}}$ by solving (2)
2: $\boldsymbol{w} \leftarrow$ Any set of link weights
3: $\boldsymbol{f} \leftarrow$ Traffic_Distribution($\boldsymbol{w}$)
4: while $\boldsymbol{f} \neq \widetilde{\boldsymbol{c}}$ do
5:     $\boldsymbol{w} \leftarrow$ Link_Weight_Update($\boldsymbol{f}$)
6:     $\boldsymbol{f} \leftarrow$ Traffic_Distribution($\boldsymbol{w}$)
7: end while
8: Return $\boldsymbol{w}$ /*final link weights*/
```

Algorithm 1: Optimize Over Link Weights

```
1: for each link $(u,v)$ do
2:     $w_{u,v} \leftarrow w_{u,v} - \alpha\,(\widetilde{c}_{u,v} - f_{u,v})$
3: end for
4: Return new link weights $\boldsymbol{w}$
```

Algorithm 2: Link-Weight_Update($\boldsymbol{f}$)

B. Compute Traffic Distribution with Exact PEFT

To compute the traffic distribution for PEFT, we should first compute the shortest paths between each pair of nodes and all the values $\Gamma_{PX}(h_{u,v}^t)$ as in Sec. IV-C. Computing the resulting distribution of traffic is complicated by the fact that $\Gamma_{PX}(\cdot)$ may direct traffic "backwards" to a node that is further away from the destination. To capture these effects, recall that f_u^t is the total incoming flow at node u (including traffic originating at u as well as any traffic arriving from other nodes) that is destined to node t. In particular, the traffic $D(s,t)$ that enters the network at node s and leaves at node t satisfies the following linear equation:

$$f_s^t - \sum_{x:(x,s)\in\mathbb{E}} f_x^t \left(\frac{\Gamma_{PX}(h_{x,s}^t)}{\sum_{(x,j)\in\mathbb{E}} \Gamma_{PX}(h_{x,j}^t)} \right) = D(s,t). \quad (18)$$

That is, the traffic $D(s,t)$ entering the network at node s matches the total incoming flow f_s^t at node s (destined to node t), excluding the traffic entering s from other nodes. The transit flow is captured as a sum over all incoming links from neighboring nodes x, which split their incoming traffic f_x^t over their links based on the traffic-splitting function.

The N linear equations (18) for each t typically require O(N^3) time [13] to solve. Thus the total complexity is O(N^4).

C. Approximate Traffic Distribution with Downward PEFT

To further reduce the computational overhead, we realize that the optimal traffic distribution should be loop free. Thus, in the last iteration in Algorithm 1, the flow loop should be negligible. In addition, the accurate solution for each intermediate iteration is not necessary in practice, we can approximate Exact PEFT ($\Gamma_{PX}(\cdot)$) with Downward PEFT ($\Gamma_{PD}(\cdot)$) to forward traffic only on "downward" paths, the traffic distribution for each intermediate iteration can be computed using a combinatorial algorithm, which is significantly faster than solving linear equations (18).

As in Sec. V-B, we first compute the shortest paths between all pairs of nodes, as well as the values of $\Gamma_{PD}(h_{u,v}^t)$, as shown in the first step of Algorithm 3. The following procedure is

```
1: For link weights $\boldsymbol{w}$, construct all-pairs shortest paths and
   compute $\Gamma_{PD}(h^t_{u,v})$
2: for each destination $t$ do
3:    Temporarily remove link $(u,v)$ where $d^t_u \leq d^t_v$
4:    Do topological sorting on the remaining network
5:    for each source $s \neq t$ in the decreasing topological
      order do
6:       $f^t_s \leftarrow D(s,t) + \sum_{x:(x,s)\in\mathbb{E}} f^t_{x,s}$
7:       $f^t_{s,v} \leftarrow f^t_s \frac{\Gamma_{PD}(h^t_{s,v})}{\sum_{(s,j)\in\mathbb{E}} \Gamma_{PD}(h^t_{s,j})}$
8:    end for
9: end for
10: $f_{u,v} \leftarrow \sum_{t\in\mathbb{V}} f^t_{u,v}$
11: Return $\boldsymbol{f}$ /*set of $f_{u,v}$*/
```

Algorithm 3: Traffic_Distribution($\boldsymbol{w}$) with $\Gamma_{PD}(\cdot)$

very similar to but subtly different from that for calculating $\Gamma_{PD}(h^t_{u,v})$. We consider each destination t independently, since the flow to each destination can be computed without regard to the other destinations. After temporarily removing link (u,v) where $d^t_u \leq d^t_v$ since there is no flow on it, we get an acyclic network and do topological sorting on the remaining network. The computation starts at the node without incoming link in the acyclic network, since this node would never carry any traffic to t that originates at other nodes. Proceeding through the nodes s in *decreasing* topological order, we compute the total incoming flow at node s (destined to t) as the sum of the flow originating at s (i.e., $D(s,t)$) and the flow arriving from neighboring nodes x ($f^t_{x,s}$). Then, we use the total incoming flow at s to compute the flow of traffic toward t on each of its outgoing links (s,v), using the traffic-splitting function $\Gamma_{PD}(\cdot)$.

In Algorithm 3, computing the all-pairs shortest paths with the Floyd-Warshall algorithm has time complexity $O(N^3)$ [13]. For each destination, topology sorting requires $O(N+E)$ time, and summarizing the incoming flow and splitting across the outgoing links requires $O(N+E)$ time. Thus, the total time complexity to run Algorithm 3 in each iteration of Algorithm 1 is $O(N^3 + N(N+E)) = O(N^3)$.

Finally, the total running time for Algorithm 1 depends on the time required to solve (2) and the total number of iterations required for Algorithms 2 and 3. Although the original NEM problem involves an exponential number of variables, the complexity of Algorithm 1 is still comparable to solving a convex optimization with polynomial number of variables (like the COMMODITY problem (2)) using gradient projection, since we do not need to solve NEM directly.

VI. Performance Evaluation

How well can the new routing protocol perform and how fast can the new link weight computation be? PEFT has been proven to achieve optimal TE in Sec. III, with a complexity of link weight computation similar to that of solving convex optimization (with a polynomial number of variables). In this section, we numerically demonstrate that its approximate version, Downward PEFT, can make convergence very fast in practice while coming extremely close to TE optimality.

A. Simulation Environment

We consider two network objective functions ($\Phi(\{f_{u,v}, c_{u,v}\})$): maximum link utilization, and total link cost (1) (as used in operator's TE formulation). For benchmarking, the optimal values of both objectives are computed by solving linear program (2) with CPLEX 9.1 via AMPL, and serve as the performance benchmarks.

To compare with OSPF, we use the state-of-the-art local-search method in [2]. We adopt TOTEM 1.1 [14], which follows the same approach as [2], and has similar quality of the results [8]. We use the same parameter setting for local search as in [2], [7] where the link weights are restricted as integers from 1 to 20 since a larger weight range would slow down the searching [7], initial link weights are chosen randomly, and the best result is collected after 5000 iterations.

To determine link weights under PEFT, we run Algorithm 1 with up to 5000 iterations of computing traffic distribution and updating link weights. Abusing terminology a little, in this section we use the term PEFT to denote the traffic engineering with Algorithm 1 (including two sub-Algorithms 2 and 3).

We run the simulation on a real backbone network and several synthetic networks. First is the Abilene network, which has 11 nodes and 28 directional links with 10Gbps capacity. The traffic demands are extracted from the sampled Netflow data on Nov. 15th, 2005. To simulate networks with different congestion levels, we create different test cases by uniformly decreasing the link capacity until the maximal link utilization reaches 100% with optimal TE.

We also test the algorithms on the same topologies and traffic matrices as those in [2]. The 2-level hierarchical networks were generated using GT-ITM, which consists of two kinds of links: local access links with 200-unit capacity and long distance link with 1000-unit capacity. In the random topologies, the probability of having a link between two nodes is a constant parameter and all link capacities are 1000 units. In these test cases, for each network, traffic demands are uniformly increased to simulate different congestion levels.

B. Minimize Maximum Link Utilization

Since we create different levels of congestion for the same network by uniformly decreasing link capacities or uniformly increasing traffic demands, we just need to compute the Maximum Link Utilization (MLU) for one test case in each network because MLU is proportional to the ratio of total demand over total capacity. In addition to MLU, we are particularly interested in the metric "efficiency of capacity utilization", η, which is defined as the following ratio: the percentage of the traffic demand satisfied when the MLU reaches 100% under a traffic engineering scheme over that in optimal traffic engineering. The improvement in η is referred to as the "Internet capacity increase" in [2].

[8]Proprietary enhancements can bring in factors of improvement, but as we will see, PEFT's advantage of computational speed is orders-of-magnitude.

For any test case of a network, if MLU of optimal TE, OSPF, and PEFT are ξ, ξ_O and ξ_D respectively, then $\eta_O = \frac{\xi}{\xi_O}$, and $\eta_D = \frac{\xi}{\xi_D}$. Thus PEFT can increase Internet capacity over OSPF by $\eta_D - \eta_O$. Fig. 2 illustrates the efficiency of capacity utilization of the three schemes. They show that PEFT is very close to optimal traffic engineering in minimizing MLU, and increases Internet capacity over OSPF by **15%** for Abilene network and **24%** for hier50b network, respectively.

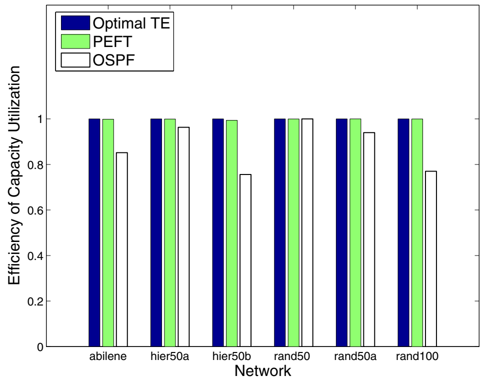


Fig. 2. Efficiency of capacity utilization of optimal traffic engineering, PEFT and local Search OSPF

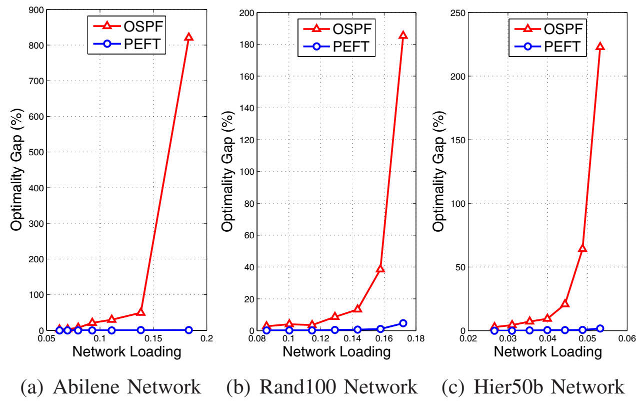
C. Minimize Total Link Cost

We also employ the cost function (1) as in [2]. Comparison is on the optimality gap, in terms of the total link cost, compared against the value achieved by optimal traffic engineering. Typical results for different topologies with various traffic matrices are shown in Fig. 3, where the network loading is the ratio of total demand over total capacity. From the results, we observe that the gap between OSPF and optimal traffic engineering can be very significant (up to 821%) for the most congested case of Abilene network. In contrast, PEFT can achieve almost the same performance as the optimal traffic engineering in terms of total link cost. Note that, within those figures, the maximum optimality gap of PEFT is only up to 8.8% in Fig. 3(b), which can be further reduced to 1.5% with a larger step-size and more iterations (which is feasible as the algorithm runs very fast to be shown in Sec. VI-D).

D. Running Time Requirement

The tests for PEFT and local search OSPF were performed under the time-sharing servers of Redhat Enterprise Linux 4 with Intel Pentium IV processors at 2.8∼3.2 Ghz. Note that the running time for local search OSPF is sensitive to the traffic matrix since a near-optimal solution can be reached very fast for light traffic matrices. Therefore, we show the range of their average running times per iteration for qualitative reference.

Fig. 4 shows the optimality gap (in a log scale) achieved by local search OSPF and PEFT, within the first 500 iterations for a typical scenario (Fig. 3(c)). It demonstrates that Algorithm 1 for PEFT converges much faster than local search for OSPF. Table III shows the average running time per iteration for different networks. We observe that our algorithm is very fast,



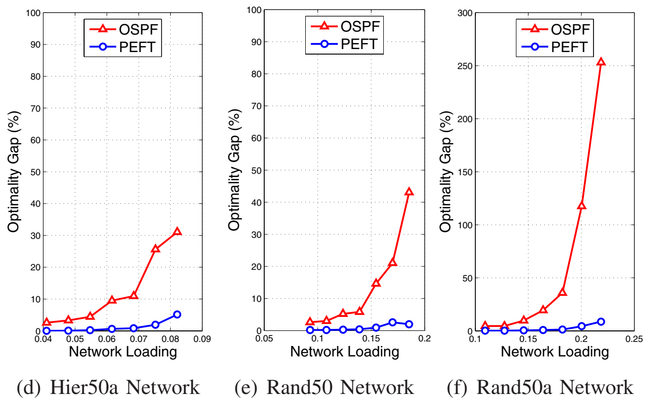


Fig. 3. Comparison of PEFT and Local Search OSPF in terms of optimality gap on minimizing total link cost

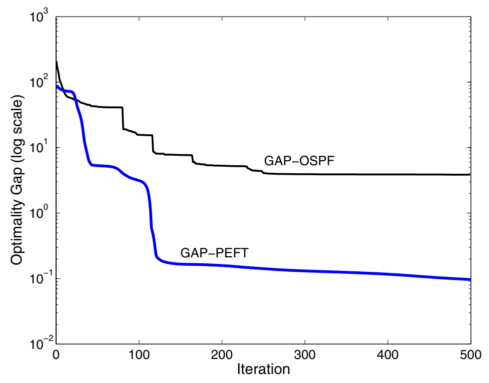


Fig. 4. Comparison of the drop in optimality gap between Local Search OSPF and PEFT in a 2-level topology with 50 nodes and 212 links

requiring at most 2 minutes even for the largest network (with 100 nodes) tested, while the OSPF local search needs tens of hours. On average, the algorithm developed in this paper to find link weights for PEFT routing is 2000 times faster than local search algorithms for OSPF routing.

VII. Difference between PEFT and DEFT

Link-state routing protocols can be categorized as link-based and path-based in terms of flow splitting. Their difference is illustrated in Fig. 5, with a network that only has traffic demand

TABLE III
Average running time per iteration required by PEFT and local search OSPF to attain the performance in Fig. 3

Net. ID	Topology	Node #	Link #	Time per Iteration (second)	
				PEFT	OSPF
abilene	Backbone	11	28	0.002	6.0∼13.9
hier50a	2-level	50	148	0.006	6.0∼13.9
hier50b	2-level	50	212	0.007	6.4∼17.4
rand50	Random	50	228	0.007	3.2∼9.0
rand50a	Random	50	245	0.007	6.1∼14.1
rand100	Random	100	403	0.042	39.5∼105.1

from s to t. Assume the weights of the links are shown in Fig. 5(a). Obviously, the shortest distance from s to t is 2 units and both nodes t and u are on the shortest paths from s to t. In a link-based splitting scheme (e.g. OSPF, Fong [6] and DEFT [1]), node s evenly splits traffic across its *two* outgoing links (s,t) and (s,u) as shown in Fig. 5(b). Whereas in a path-based splitting scheme, e.g. PEFT, there are *three* equal-length paths from (s,t) and s evenly splits traffic across them as shown in Fig. 5(c). Note that, the path-based model does not imply explicit routing to set up tunnels for all the possible paths. Instead, each node just needs to compute and stores the aggregated flow-splitting ratio across its outgoing links, like 66% on link (s,u) for the sample network in Fig 5(c). Therefore, path-based splitting schemes can still be realized with hop-by-hop forwarding.

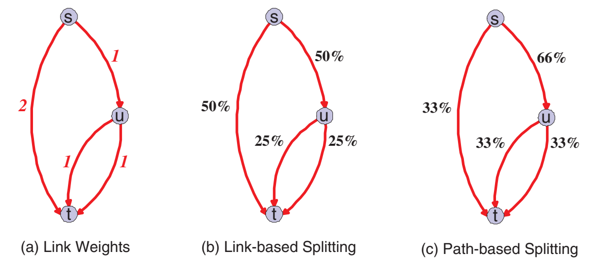


Fig. 5. Difference in traffic splittings for link-based and path-based link-state routing protocol

This paper is substantially different from our previous work on [1], with the following key differences:

1) DEFT is a link-based flow splitting while PEFT is a path-based flow splitting.
2) The core algorithms for setting link weights are completely different. [1] introduces a non-convex non-smooth optimization for DEFT and a two-stage iterative solution method, while the theory for PEFT is Network Entropy Maximization. The two-stage method for DEFT is *much slower* than the algorithms developed for PEFT in this paper.
3) [1] *numerically* shows DEFT can realize *near optimal* TE in terms of a particular objective (total link cost), while this paper *proves* that PEFT can realize *optimal* TE with any convex objective function.

VIII. Concluding Remarks

Commodity-flow-based routing protocols are optimal for any convex objective in Internet TE but introduce much configuration complexity. Link-state routing is simple but does not seem to achieve optimal TE based on prior works. This paper proves that optimal traffic engineering, in fact, *can* be achieved by link-state routing with hop-by-hop forwarding, and the right link weights *can* be computed efficiently, as long as flow splitting on non-shortest paths is allowed but properly penalized. In [10], we also show uniqueness of the exponential penalty in achieving optimal TE, and discuss interpretations of NEM from the viewpoints of statistical physics and combinatorics.

We also highlight that optimization is used in three different ways in this paper [10]. First, it is used when developing algorithms to solve the link weight computation problem for PEFT. In a more interesting way, the level of difficulty of optimizing link weights for OSPF is used as a hint that perhaps we need to revisit the standard assumption on how link weights should be used, in the approach of "Design For Optimizability". In yet another way, optimization in the form of NEM is introduced as a conceptual framework to develop link-state routing protocols with hop-by-hop forwarding.

Acknowledgment

This research is in part supported by DARPA W911NF-07-1-0057, ONR YIP N00014-07-1-0864, AFOSR FA9550-06-1-0297, NSF CNS-0519880 and CNS 0720570. We appreciate the helpful discussions with D. Applegate, B. Fortz, J. He, J. Huang, D. Johnson, H. Karloff, Y. Li, J. Liu, M. Prytz, A. Tang, M. Thorup, J. Yu, and J. Zhang.

References

[1] D. Xu, M. Chiang, and J. Rexford, "DEFT: Distributed exponentially-weighted flow splitting," in *INFOCOM'07, Anchorage, AK*, May 2007.
[2] B. Fortz and M. Thorup, "Increasing Internet capacity using local search," *Computational Optimization and Applications*, vol. 29, no. 1, pp. 13–48, 2004.
[3] Z. Wang, Y. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying," in *INFOCOM'01, Anchorage, AK*, 2001.
[4] D. Awduche, "MPLS and traffic engineering in IP networks," *IEEE Communication Magazine*, vol. 37, no. 12, pp. 42–47, Dec. 1999.
[5] A. Sridharan, R. Guérin, and C. Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 2, pp. 234–247, 2005.
[6] J. H. Fong, A. C. Gilbert, S. Kannan, and M. J. Strauss, "Better alternatives to OSPF routing," *Algorithmica*, vol. 43, no. 1-2, pp. 113–131, 2005.
[7] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *INFOCOM'00, Tel Aviv, Israel*, 2000, pp. 519–528.
[8] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *Journal of Computer and System Sciences*, vol. 26, no. 3, pp. 362–391, 1983.
[9] D. Mitra and K. G. Ramakrishnan, "A case study of multiservice multipriority traffic engineering design for data networks," in *GLOBECOM'99, Rio de Janeiro, Brazil*, Dec. 1999, pp. 1077–1083.
[10] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," in *techreport, http://www.research.att.com/∼dahaixu/pub/nem/peft.pdf*, Jul. 2007.
[11] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Athena Scientific, 1999.
[12] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
[13] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. The MIT Press, Cambridge, 1990.
[14] TOTEM, http://totem.info.ucl.ac.be.