# Link-State Routing Can Achieve Optimal Traffic Engineering

Dahai Xu Mung Chiang Jennifer Rexford
Princeton University

Under anonymous submission to SIGCOMM 2007. Please do not disclose it widely.

## ABSTRACT

This paper settles a long-standing question with a positive answer: optimal traffic engineering can be realized using just link-state routing protocols. In conventional link-state protocols like OSPF and IS-IS, routers compute shortest paths based on link weights, splitting traffic evenly when multiple shortest paths exist. Unfortunately, in these particular cases of link-state protocols, computing the optimal link weights for a given traffic matrix is an NP-hard problem. Even the best setting of the weights can lead to traffic loads that deviate significantly from the ideal distribution of the traffic. In this work, we show that splitting traffic over multiple paths as an exponential function of path length can achieve optimal traffic engineering. We also present a polynomial-time algorithm for computing the optimal link weights, which in practice is also much more efficient than the local-search heuristics used today. In addition, we show how to apply the algorithm to robust traffic engineering, where one set of link weights must perform well for multiple traffic matrices or failure scenarios. These results are based on our new Network Entropy Maximization framework for routing-protocol design, which parallels the role of Network Utility Maximization for congestion control.

## 1. INTRODUCTION

Most large IP networks run link-state routing protocols, like OSPF (Open Shortest Path First) and IS-IS (Intermediate System-Intermediate System), that select paths based on link weights. Routers use these protocols to exchange link weights and construct a complete view of the topology inside an Autonomous System. Then, each router computes path lengths (as the sums of the link weights along the paths), and forwards incoming packets to an outgoing link according to certain rules. For example, OSPF and IS-IS forwards packets along the shortest paths to the destination. When links or nodes fail, the routers automatically compute new paths based on the current topology and link weights. To alleviate congestion in the network, operators simply adjust one or two link weights to distribute the traffic over different paths, in a process known as *traffic engineering* (TE). In this paper, we show that new variants of link-state protocols that split traffic over multiple paths, based only on the link weights, can achieve optimal traffic engineering with a polynomial-time algorithm for setting the link weights

### 1.1 Limitations of Traffic Engineering Today

In practice, the link weights are *configured*, by network operators or automated management systems, to satisfy traffic-engineering goals by minimizing an objective function that captures the congestion in the network. Two common objectives are minimizing the maximum link utilization and minimizing the sum of a convex cost function of the link utilizations [1]. These objective functions define the meaning of optimality of traffic engineering. Typically, the link weights are computed by applying offline, centralized optimization techniques, which may evaluate the effects of changes to the link weights on the distribution of traffic in the network. However, in OSPF, even the best setting of the link weights leads to traffic loads that *deviate significantly from an optimal distribution of the traffic*, where the routers have complete freedom in how they forward traffic. In addition, optimizing the link weights is *an NP-hard problem*, forcing operators to resort to heuristics like local search [2, 3]. Selecting the link weights is even more difficult for robust traffic engineering, where a single setting of the link weights must perform well for multiple traffic matrices or failure scenarios [4, 5].

Computing the link weights and achieving optimal performance are difficult for the same underlying reason: today's link-state protocols force all traffic to traverse *only the shortest paths*, with *even splitting of traffic* when multiple shortest paths exist. Although easy to implement in the routers, even splitting over shortest paths can make it impossible to express the optimal distribution of traffic through a setting of the link weights. In contrast, an optimal traffic distribution can be realized by dividing an *arbitrary* fraction of traffic over many paths through the network, as is possible with MultiProtocol Label Switching (MPLS) [6]; however, optimality in MPLS routing comes with a cost for establishing many tunnels to forward data packets over these

| | Commodity Routing | Link-State Routing | |
|---|---|---|---|
| | | **Shortest-Path Routing** | **Exponential Splitting** |
| Traffic Splitting | Arbitrary | Even | Exponential |
| Scalability | Low | High | High |
| Optimal TE | Yes | No | *Yes* |
| Computational Complexity | Polynomial | NP Complete | *Polynomial* |
| Computation Techniques | Convex Optimization | Heuristic | *Convex Optimization + Combinatorial Algorithm* |

**Table 1: Comparison of various traffic-engineering schemes (new contributions in *italics*)**

paths. Other studies explore more flexible ways to split traffic over shortest paths [7, 8], but these solutions do not enable routers to *independently* compute the flow-splitting ratios from link weights. Clearly, there is a tension between the optimal but complex routing and the simple but suboptimal link-state routing. This paper shows that routing can be both simple and optimal.

Achieving optimal traffic engineering with link-state routing requires changes to both

- how paths are determined from link weights (i.e., the online algorithm running on the routers) and
- how the link weights are computed (i.e., the offline algorithm running in the management system).

On the first problem, it is important to realize that link-state protocols do not have to use only the shortest paths. Two recent studies proposed alternative ways for routers to split traffic over multiple paths based solely on the link weights. First, Fong et al. [9] suggested that routers divide traffic over all paths in *inverse proportion* to the path length, or as an *exponential function* of path length. In both approaches, data packets may traverse loops, since traffic does not always make forward progress toward the destination. Second, Xu et al. [10] proposed using only the "downward" paths that move packets closer to the destination, splitting traffic over the downward paths as an exponential function of path length. Both papers argue that more flexible flow-splitting rules would lead to simpler optimization problems for setting the link weights, as well as better distributions of traffic[1].

## 1.2 Optimal Routing Using Only Link Weights

The work in [9] and [10] still leaves two important questions unanswered: *(i) can any of these link-state routing protocols realize optimal traffic engineering?* and *(ii) if so, can the optimal link weights be found in polynomial time?* In this paper, we show that the answers to both questions are "**yes**" (Theorem 1 in Sec. 3.2). In fact, we prove that dividing traffic over all paths as an exponential function of their length (as suggested in [9]) can, in fact, achieve an optimal distribution of the traffic, with an appropriate setting of the link weights. We present a novel, polynomial-time algorithm for computing these link weights. The algorithm runs relatively slow in practice, however, since it must solve a large system of linear equations. Observing that optimal routing rarely (if ever) creates flow loops, we limit the routers to "downward paths" (as suggested in [10]) and present a very efficient polynomial-time algorithm for optimizing the link weights. Our experiments show that the running time can be 2000 times faster than local-search methods for OSPF routing. We also show how to use the algorithm for robust traffic engineering for a family of traffic matrices and failure scenarios. Table 1 summarizes the state of the art, with our new results highlighted in italics.

Demonstrating that optimal traffic engineering is possible using only link weights requires several innovations in the analysis and design of network routing. In particular, it turns out that splitting traffic based on an *exponential* function is by no means an arbitrary choice. Many other seemingly natural splitting rules, such as inversely proportion to the path length, do *not* enable optimal traffic engineering. The exponential function falls naturally out of formulating the traffic-engineering problem in terms of Network Entropy Maximization (NEM). Although the general principle of entropy maximization has been used to solve other challenging networking problems [11–14], this is the first work connecting entropy with IP routing. In addition, as we summarize later in Table 6, our NEM framework for routing has interesting parallels with recent work relating TCP congestion control to Network Utility Maximization (NUM) [15–18].

The rest of the paper is organized as follows. Background on optimal traffic engineering is introduced in Sec. 2. The algorithms for realizing optimal traffic engineering with link-state protocols are presented in Sec. 3. Extensive numerical experiments of using the algorithms are shown in Sec. 4. In Sec. 5, the theoretical foundation for the algorithms is presented. In Sec. 6, the algo-

[1]They also addressed several important concerns on implementing non-shortest-path routing in practice, such as realizing non-even-splitting with hashing techniques, bounding the worst-case delay, and ensuring similar performance if link weights have to be integral.

rithms are extended to robust traffic engineering considering multiple traffic matrices or link failures. The interesting and general framework of Network Entropy Maximization is further discussed in Sec. 7. The paper is concluded in Sec. 8. The physical interpretation of using entropy maximization is summarized in Appendix A, and the implementation of non-shortest path routing covered in Appendix B.

## 2. BACKGROUND ON OPTIMAL TE

In this section, we formally define optimal traffic engineering and summarize the well-known fact that multi-commodity-flow routing can achieve the optimal distribution of the traffic. The key notation used in this paper is shown in Table 2.

### 2.1 Definitions of Optimality

Traffic engineering involves optimizing routing based on the offered traffic. We model the 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 $D(s, t)$ that captures the rate of traffic entering the network at node $s$ and leaving at node $t$. The traffic matrix can be computed based on traffic measurements or may represent explicit subscriptions or reservations from users.

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. As a building block for the objective function, most traffic-engineering studies consider a link-cost function $\Phi(f_{u,v}, c_{u,v})$ that is a strictly increasing function of $f_{u,v}$, to penalize solutions that place a high load on one or more links. For example, $\Phi(f_{u,v}, c_{u,v})$ could be the link utilization $f_{u,v}/c_{u,v}$, leading to an objective of minimizing $\max_{(u,v)}\{\Phi(f_{u,v}, c_{u,v})\}$ (a convex minimization), which favors routing solutions that minimize the maximum link utilization in the network.

Minimizing the maximum link utilization can lead to routing solutions that reduce the load on the most congested link at the expense of using circuitous paths that place a heavy load on many of the remaining links. Instead, most traffic-engineering studies choose a link-cost function that prevents congestion by placing an increasingly penalty on links as the load approaches capacity [19], such as:

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

| 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 $(u,v)$ |
| $\tilde{c}_{u,v}$ | Necessary capacity of link $(u,v)$ |
| $f^t_{u,v}$ | Flow on link $(u,v)$ destined to node $t$ |
| $\overrightarrow{f^t_u}$ | 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 allocation function |

**Table 2: Summary of Key Notation**

leading to an objective function that considers the total cost $\sum_{(u,v)\in\mathbb{E}} \Phi(f_{u,v}, c_{u,v})$ [1, 7]. Often, a piece-wise linear function is used for efficient computation of an optimal solution.

More generally, we use "$\Phi(\{f_{u,v}\})$" to represent the objective function, and define "optimal traffic engineering" as a distribution of traffic $\{f_{u,v}\}$ that minimizes a given, convex network objective function $\Phi$.

Even at this point, we can observe that there is a "gap" between the objective of optimal traffic engineering and the mechanism of link-state routing. Optimal traffic engineering is defined directly in terms of the traffic flows on the network, 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.

### 2.2 Optimal TE Via Multi-Commodity Flow

Optimal traffic engineering is possible in a routing system that can distribute an *arbitrary* proportion of traffic over multiple paths between each pair of nodes. The solution can be computed by solving the a multi-commodity-flow problem, 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 $t$. The formulation for a given traffic matrix is as follows:

*COMMODITY:*

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

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

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

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

Constraint (2b) ensures the conservation of traffic at intermediate node $y$, and constraint (2c) ensures the total flow on a link is bounded by the capacity.

For any convex network objective function, the optimization problem (2) is a convex optimization that can

be solved in polynomial time:

**Fact** 1. Optimal traffic engineering for a given traffic matrix with a convex network objective function can be realized with multi-commodity-flow routing within polynomial time.

Unfortunately, for a network with $N$ nodes and $E$ links, the routing solution may require up to $O(N^2E)$ tunnels (i.e., explicit routing) [20], making it difficult to scale. In contrast, link-state routing is much simpler, requiring only $O(E)$ parameters (i.e., one per link). The next section argues that optimal traffic engineering can, in fact, be achieved using only link weights.

## 3. OPTIMAL TE WITH LINK WEIGHTS

In this section, we formally define link-state routing protocols, where routers forward traffic based only on link weights and a *traffic-allocation function* [10]. We state our main result, proven later in Sec. 5, that an exponential flow-splitting rule can achieve optimal traffic engineering. We then present a polynomial-time algorithm that the network-management system can run to optimize the link weights for a given traffic matrix. Then, we present a much more efficient algorithm that can be used to optimize the link weights, if the network runs a link-state routing protocol with a simple variant of the exponential splitting rule, where routers direct traffic only via "downward" paths.

### 3.1 Link-State Routing Protocols

In link-state routing, each node $u$ makes an *independent* decision on how to forward traffic destined to node $t$ among its outgoing links using *only* the link weights. Each link $(u, w)$ has a single, configurable weight $w_{u,v}$. Based on a complete view of the topology and link weights, node $u$ computes its shortest distance $d_u^t$ to node $t$; $d_v^t + w_{u,v}$ represents the distance from $u$ to $t$ when routed through neighboring node $v$. To indicate if link $(u, v)$ lies on a shortest path from $u$ to $t$, we define *shortest distance gap*, $h_{u,v}^t \triangleq d_v^t + w_{u,v} - d_u^t$, which is always greater than or equal to 0. Then, $(u, v)$ lies on a shortest path to $t$ if and only if $h_{u,v}^t = 0$.

Link-state routing protocols vary in which paths they use, and in what proportions. As such, we define a traffic-allocation function $\Gamma(h_{u,v}^t)$ that indicates the relative amount of traffic destined to $t$ that node $u$ will forward via outgoing link $(u, v)$. To translate the traffic-allocation function to the actual flow on each link, we define $\overrightarrow{f_u^t}$ as the total incoming flow (destined to $t$) at node $u$. Then, the total outgoing flow of traffic (destined to $t$) traversing link $(u, v)$, $f_{u,v}^t$, can be computed as follows:

$$f_{u,v}^t = \overrightarrow{f_u^t} \frac{\Gamma(h_{u,v}^t)}{\sum_{(u,j)\in\mathbb{E}} \Gamma(h_{u,j}^t)}. \quad (3)$$

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.

The traffic-allocation function allows us to readily describe different link-state routing protocols. For example, in shortest-path routing (e.g., OSPF), each node splits flow evenly across all the outgoing links along shortest paths, leading to the following traffic-allocation function:

$$\Gamma_O(h_{u,v}^t) = \begin{cases} 1 & \text{if } h_{u,v}^t = 0, \\ 0 & \text{if } h_{u,v}^t > 0. \end{cases} \quad (4)$$

In contrast, the work in Fong et al. [9] proposes two new link-state routing protocols where nodes forward traffic on all paths as a function of their length. First, forwarding traffic in inverse proportion to the path length corresponds to the following traffic-allocation function:

$$\Gamma_I(h_{u,v}^t) = \frac{1}{(d_v^t + h_{u,v}^t)^\beta}, \quad (5)$$

where $\beta$ is a constant. Second, forwarding traffic on paths with an exponential penalty on longer paths corresponds to the following traffic-allocation function:

$$\Gamma_X(h_{u,v}^t) = e^{-h_{u,v}^t}. \quad (6)$$

Since these two traffic-allocation functions make use of all paths, the traffic does not necessarily make forward progress toward the destination at each hop; i.e., traffic may move "backwards" to a node that is further away from the destination. To prevent flow loops, Xu et al. [10] proposes a traffic-allocation function that only directs traffic on "downward" paths, as follows:

$$\Gamma_D(h_{u,v}^t) = \begin{cases} e^{-h_{u,v}^t} & \text{if } d_u^t > d_v^t, \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

They refer to the resulting link-state routing protocol as Distributed Exponentially-weighted Flow Splitting (DEFT).

### 3.2 Algorithm for Optimizing Link Weights

The key question now becomes: can optimal traffic engineering be realized using only link weights for some simple traffic-allocation function? Our main result, with a proof in Sec. 5, is as follows:

**Theorem** 1. *Optimal traffic engineering for a given traffic matrix can be realized with link weights, computed within polynomial time, by using traffic-allocation function $\Gamma_X(\cdot)$ in link-state routing protocols.*

In this subsection, we present a polynomial-time algorithm for computing the link weights for the exponential traffic-allocation function $\Gamma_X(\cdot)$. The algorithm consists of two main parts:

**Computing the optimal traffic distribution for a given traffic matrix:** Given the traffic matrix and the objective function, the solution to the COMMODITY problem (2) from Sec. 2.2 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 the minimal (maybe not the minimum)[2] set of link capacities to realize optimal traffic engineering. Next, we consider a new traffic-engineering problem on the same topology, where the link capacities are $\widetilde{c}_{u,v}$. Note that any feasible routing solution, subject to the necessary capacities, also achieves optimal traffic engineering for the original network in terms of the $\Phi(\{f_{u,v}\})$ objective.

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: $\quad \boldsymbol{w} \leftarrow$ Link_Weight_Update($\boldsymbol{f}$)
6: $\quad \boldsymbol{f} \leftarrow$ 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: $\quad w_{u,v} \leftarrow w_{u,v} - \alpha\,(\widetilde{c}_{u,v} - f_{u,v})$
3: **end for**
4: Return new link weights $\boldsymbol{w}$

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

**Computing the link weights that would achieve this distribution of the traffic:** The second step uses the traffic distribution found in the first step as input, and need not consider the objective function any further. Starting with an initial setting of the link weights, the algorithm (see Algorithm 1) repeatedly updates the link weights until the flow 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 $\alpha$ is a positive step size, which can be constant or dynamically adjusted; we find that setting $\alpha$ to the reciprocal of the maximum necessary link capacity ($\frac{1}{\max \widetilde{c}_{u,v}}$) performs well in practice, as discussed later in Sec. 4.

In terms of computational complexity, we know from Sec. 2.2 that COMMODITY can be solved in polynomial time. The complexity of Algorithm 2 is $O(E)$. The remaining questions are (i) whether Algorithm 1 terminates after a polynomial number of iterations and (ii) whether *Traffic_Distribution* can be solved in polynomial time. In Sec. 5, we prove that the answer to both questions is "yes" for the traffic-allocation function $\Gamma_X(\cdot)$. We present the polynomial-time *Traffic_Distribution* procedure for any traffic-allocation function (including the function $\Gamma_X(\cdot)$ of interest) in the next subsection.

### 3.3 Traffic Distribution with $\Gamma_X(\cdot)$ Function

To compute the distribution of traffic, we must first compute the shortest paths between each pair of nodes; then, the lengths of the shortest paths can be used to compute all values of $\Gamma_X(h^t_{u,v})$, as shown in the first step of Algorithm 3. Computing the resulting distribution of traffic is complicated by the fact that $\Gamma_X(\cdot)$ may direct traffic "backwards" to a node that is further away from the destination; that is, some traffic may "come back" to a node. To capture these effects, recall that $\overrightarrow{f^t_u}$ 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:

$$\overrightarrow{f^t_s} - \sum_{x:(x,s)\in\mathbb{E}} \overrightarrow{f^t_x} \left( \frac{\Gamma_X(h^t_{x,s})}{\sum_{(x,j)\in\mathbb{E}} \Gamma_X(h^t_{x,j})} \right) = D(s,t). \quad (8)$$

That is, the traffic $D(s, t)$ entering the network at node $s$ matches the total incoming flow $\overrightarrow{f^t_s}$ 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 $\overrightarrow{f^t_x}$ over their links based on the traffic-allocation function.

1: For link weights $\boldsymbol{w}$, construct all-pairs shortest paths (e.g. with Floyd-Warshall algorithm) and compute $\Gamma_X(h^t_{u,v})$
2: Compute $\overrightarrow{f^t_u}$ by solving linear equations (8)
3: $f^t_{u,v} \leftarrow \overrightarrow{f^t_u} \frac{\Gamma_X(h^t_{u,v})}{\sum_{(u,j)\in\mathbb{E}} \Gamma_X(h^t_{u,j})}$
4: $f_{u,v} \leftarrow \sum_{t\in\mathbb{V}} f^t_{u,v}$
5: Return $\boldsymbol{f}$/*set of $f_{u,v}$, total flow on each link*/

**Algorithm 3:** Traffic_Distribution($\boldsymbol{w}$) with $\Gamma_X(\cdot)$

Algorithm 3 computes the traffic distribution by solv-

[2] $\widetilde{\boldsymbol{c}}$ 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 $\forall \widetilde{\boldsymbol{c'}} : \widetilde{\boldsymbol{c}} \preceq \widetilde{\boldsymbol{c'}}$.

ing the system of linear equations (8) and computing the resulting flow on each link $(u, v)$. Though polynomial-time solvable, the $N^2$ linear equations require $O(N^4 \log N)$ time [21] to solve, due to the need for matrix inversion. This can lead to a long running time for Algorithm 3 and, in turn, for each iteration of Algorithm 1.

### 3.4 Traffic Distribution with $\Gamma_D(\cdot)$ Function

The key to further reducing the computational complexity of the *Traffic_Distribution* procedure is to observe that the optimal traffic distribution should not have a flow loop. Thus, in the last iteration in Algorithm 1, the loopback flow should be negligible. In fact, preventing loopback flow in intermediate iterations could lead to faster computation of the optimal traffic distribution. This leads us to consider the traffic-allocation function $\Gamma_D(\cdot)$ that, while using an exponential splitting rule similar to $\Gamma_X(\cdot)$, forwards traffic only on "downward" paths. Therefore, the traffic distribution for each intermediate iteration can be computed using a combinatorial algorithm, which is significantly faster than solving linear equations (8).

1: For link weights $\boldsymbol{w}$, construct all-pairs shortest paths and compute $\Gamma_D(h^t_{u,v})$
2: **for each** destination $t$ **do**
3: **for each** source $s \neq t$ in the decreasing order of its shortest distance to $t$ **do**
4: $\overrightarrow{f^t_s} \leftarrow D(s,t) + \sum_{x:(x,s)\in\mathbb{E}} f^t_{x,s}$
5: $f^t_{s,v} \leftarrow \overrightarrow{f^t_s} \frac{\Gamma_D(h^t_{s,v})}{\sum_{(s,j)\in\mathbb{E}} \Gamma_D(h^t_{s,j})}$
6: **end for**
7: **end for**
8: $f_{u,v} \leftarrow \sum_{t\in\mathbb{V}} f^t_{u,v}$
9: Return $\boldsymbol{f}$/*set of $f_{u,v}$*/

**Algorithm 4:** Traffic_Distribution($\boldsymbol{w}$) with $\Gamma_D(\cdot)$

As in Sec. 3.3, we first compute the shortest paths between all pairs of nodes, as well as the values of $\Gamma_D(h^t_{u,v})$, as shown in the first step of Algorithm 4. Then, we consider each destination $t$ independently, since the flow to each destination can be computed without regard to the other destinations. The computation starts at the node with the *longest* distance to $t$, since this node would never carry any traffic to $t$ that originates at other nodes. Proceeding through the nodes $s$ in decreasing order of their distance to $t$, 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-allocation function.

In Algorithm 4, computing the all-pairs shortest paths with the Floyd-Warshall algorithm has time complexity $O(N^3)$ [22]. For each destination, sorting the remaining nodes based on distance requires $O(N \log N)$ 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 4 in each iteration of Algorithm 1 is $O(N^3 + N(\log N + E)) = O(N^3)$. The precise 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 4.

## 4. PERFORMANCE EVALUATION

In this section, we present the numerical results of various schemes under many practical scenarios. We consider two network objective functions ($\Phi(\{f_{u,v}\})$): maximum link utilization, and total link cost (1). The optimal values of both objectives are computed by solving linear program (2) with CPLEX 9.1 [23] via AMPL [24], and serve as the performance benchmarks.

To determine link weights under OSPF, we use the state-of-the-art local-search method in [2], which performs the best among several typical approaches, including UnitOSPF, RandomOSPF, InvCapOSPF, L2OSPF, etc. (refer to [2] for detail). We adopt TOTEM 1.1 [25], the open-source software project with IGP weight optimization, which follows the same approach as [2], and has similar quality of the results. We use the same parameter setting for local search as in [1, 2] where link weight is restricted as an integer from 1 to 20, initial link weights are chosen randomly, and the best result is collected after 5000 iterations.

To determine link weights under DEFT, we run Algorithm 1 with up to 5000 iterations of computing traffic distribution and updating link weights. Step-size $\alpha$ is set as the reciprocal of the maximum necessary link capacity. In this section, we use the term DEFT to denote the traffic engineering with Algorithm 1 (including two sub-Algorithms 2 and 4). However, we do *not* reproduce the performance of DEFT using the optimization model developed in [10], which can obtain near-optimal traffic distribution but with a much lower speed than the algorithms developed in this paper.

### 4.1 Topologies and Traffic Matrices

We run the simulation on a real backbone network and several synthetic networks. The properties of the networks used are summarized in Table 3. First is the Abilene network (Fig. 1) [26], which has 11 nodes and 28 directional links with 10Gbps capacity. In a transit network, like Abilene, routers often have multiple egress points for directing traffic toward an external destination [27]. In practice, they choose the closest egress point in terms of IGP link weights as the *early-exit* or *hot-potato* routing. We can convert a demand matrix in a backbone network into a pure intra-domain traffic matrix by creating a virtual node to represent each unique

egress point set, and connecting each egress point to the virtual node with an unidirectional link of unlimited capacity [28]. We use the Netflow data on Nov. 15th, 2005, which has 29 distinct egress sets. The traffic demands are extracted from the sampled Netflow data with a sampling rate of 1/100. $D(s,t)$ represent the average traffic demand in an hour from source $s$ to a virtual destination $t$ (egress point set). In addition, 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 traffic engineering.

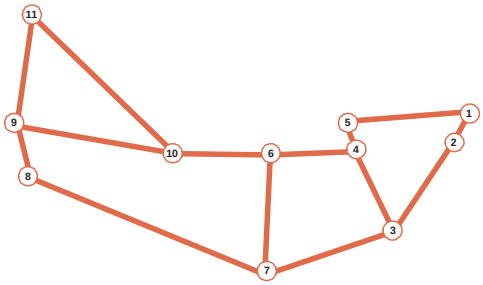



**Figure 1: Abilene Network**

For fair comparison with the state-of-the-art OSPF local search algorithms, 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 an link between two nodes is a constant parameter and all link capacities are 1000 units. In these cases, for a same network, traffic demands are uniformly increased to simulate different congestion levels.

| Net. ID | Topology | Node # | Link # |
|---|---|---|---|
| abilene | Backbone | 11 | 28 |
| hier50a | 2-level | 50 | 148 |
| hier50b | 2-level | 50 | 212 |
| rand50 | Random | 50 | 228 |
| rand50a | Random | 50 | 245 |
| rand100 | Random | 100 | 403 |

**Table 3: Networks for Performance Evaluation**

## 4.2 Minimize Maximum Link Utilization

Given that the link capacities uniformly decrease or traffic demands uniformly increase in different test cases for the same network, 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 "capacity utilization", $\eta$, which is defined as the following ratio: the percentage of the traffic demand satisfied when the MLU reaches 100% under a routing scheme over that in optimal traffic engineering. The improvement on $\eta$ is referred to as the "Internet capacity increase" [2].

For any test case of a network, if MLU of optimal traffic engineering, OSPF, and DEFT are $\xi$, $\xi_O$ and $\xi_D$ respectively, then $\eta_O = \frac{\xi}{\xi_O}$, and $\eta_D = \frac{\xi}{\xi_D}$. Then DEFT can increase Internet capacity over OSPF by $\eta_D - \eta_O$.

Table 4 shows the maximum link utilizations of optimal traffic engineering, DEFT, and Local Search OSPF for the test case with the lightest loading of each network. Fig. 2 illustrates the capacity utilization of the three schemes. They show that DEFT is very close to optimal traffic engineering in minimizing MLU, and increases Internet capacity over OSPF by **14.7%** for Abilene network and **23.8%** for hier50b network respectively.

| Net. ID | Optimal TE | DEFT | OSPF |
|---|---|---|---|
| abilene | 33.9% | 33.9% | 39.8% |
| hier50a | 56.4% | 56.5% | 58.6% |
| hier50b | 44.7% | 45.0% | 59.2% |
| rand50 | 60.6% | 60.6% | 60.6% |
| rand50a | 60.8% | 60.8% | 64.7% |
| rand100 | 55.0% | 55.0% | 71.5% |

**Table 4: Maximum link utilization of optimal traffic engineering, DEFT and Local Search OSPF for light-loading networks**

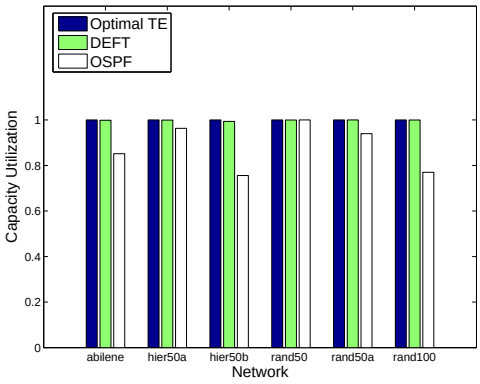



**Figure 2: Capacity Utilization of Optimal traffic engineering, DEFT and Local Search OSPF**

## 4.3 Minimize Total Link Cost

As mentioned in [2, 10], maximum link utilization is not a metric as accurate as total link cost since it cannot reveal the number of congested links. 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.

The results for different topologies with seven different traffic matrices are shown in Fig. 4.3, 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, DEFT 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 DEFT 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. 4.5).

### 4.4 Convergence Behavior

Fig. 4 shows the optimality gap in terms of total cost achieved by DEFT of different step-sizes within the first 5000 iterations for Abilene network with the least link capacities, which provides convergence behavior typically observed. The legends show the ratio of the step-size over the default setting. It demonstrates that the algorithms developed in Sec. 3 for DEFT protocol converges very fast even with default setting, and reduces the gap to 5% after 100 iterations and 1% after 3000 iterations. In addition, increasing step-size a little will speed up the convergency, but too large a step-size (e.g., 1.9 in the above example) would cause oscillation. More elaborate or dynamically adjustable step-size-setting is left as future work.

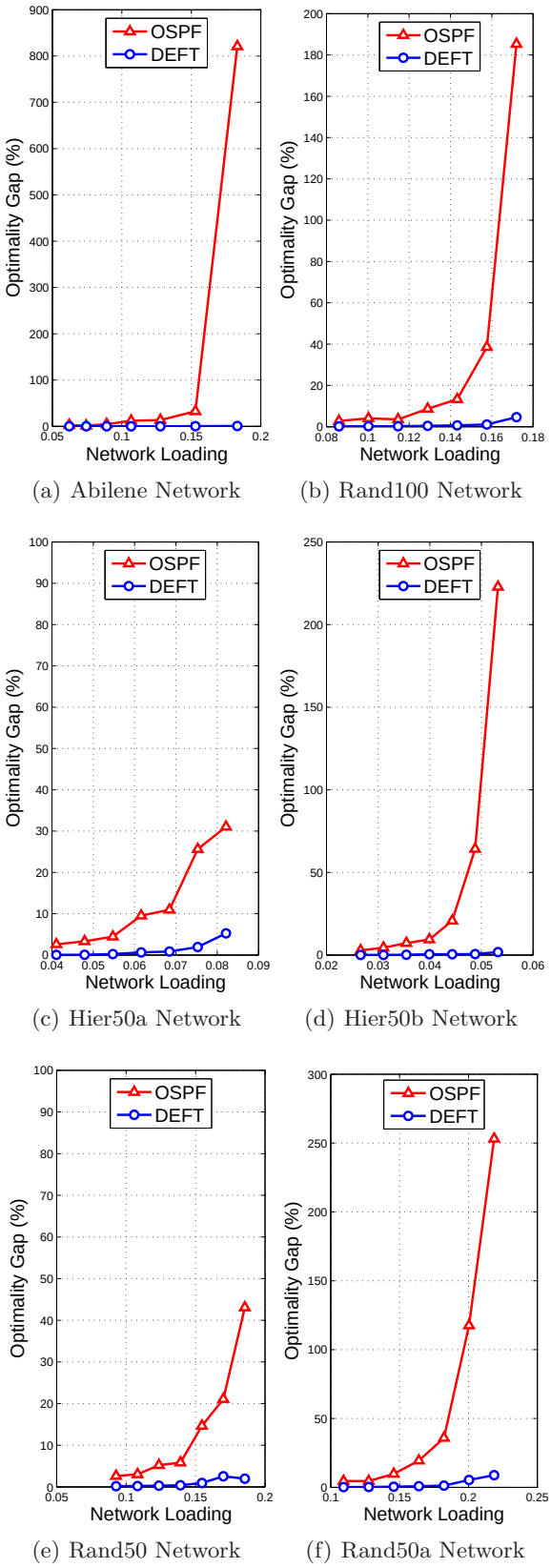
### 4.5 Running Time Requirement

The tests for DEFT 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 traffic matrix since a solution with acceptable optimality can be reached very fast for light traffic matrices. Therefore, we show their average running times per iteration for qualitative reference.

Table 5 shows the running time for different networks. We observe that our algorithm is very fast, which only requires at most 2 minutes even for the largest network tested, while the OSPF local search algorithms needs tens of hours. Roughly speaking, the algorithm developed in this paper to find link weights for DEFT routing is **2000** times faster than local search algorithms for OSPF routing.

## 5. THEORETICAL FOUNDATIONS

In this section, we provide theoretical support for the algorithm (in Sec. 3) of realizing optimal traffic engi-




(a) Abilene Network (b) Rand100 Network

(c) Hier50a Network (d) Hier50b Network

(e) Rand50 Network (f) Rand50a Network

**Figure 3: Comparison of DEFT and Local Search OSPF in terms of optimality gap on minimizing total link cost**

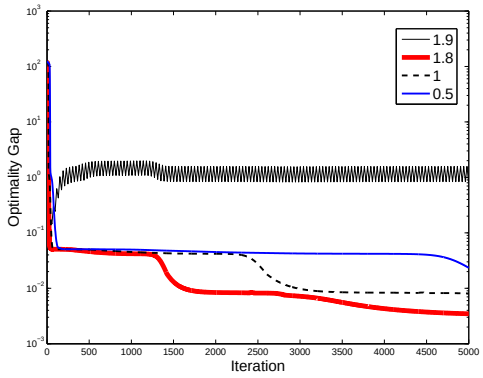



**Figure 4: Evolution of optimality gap of DEFT with different step sizes**

| Net. ID | Time per Iteration (second) | |
|---|---|---|
| | DEFT | OSPF |
| abilene | 0.002 | 6.0∼13.9 |
| hier50a | 0.004 | 6.0∼13.9 |
| hier50b | 0.005 | 6.4∼17.4 |
| rand50 | 0.005 | 3.2∼9.0 |
| rand50a | 0.005 | 6.1∼14.1 |
| rand100 | 0.031 | 39.5∼105.1 |

**Table 5: Average running time per iteration required by DEFT and local search OSPF to attain the performance in Fig. 4.3**

neering with link-state protocols. Some of the technical difficulties to be solved in this section is briefly summarized below.

- We develop the technique of using "necessary capacities" to define a new optimization problem that only requires that link load not exceed the necessary capacity. We know this problem has a solution, and now have an extra degree of freedom to pick an objective function for the new optimization problem. Based on intuition described in Appendix A, we pick an objective function based on maximizing network entropy, thus formulating the new Network Entropy Maximization (NEM) problem.

- NEM can be decomposed using standard optimization techniques, leading to two independent subproblems, corresponding directly to the *Traffic_Distribution* and *Link_Weight_Update* procedures in Sec. 3. We also show that the decomposition of the Traffic_Distribution problem can be solved independently for each $(s,t)$ pair (called a "traffic demand") in the traffic matrix.

- Using the properties of the entropy function, the solution to the optimization problem for each demand produces a traffic allocation that corresponds directly to $\Gamma_X(\cdot)$. Using other objective functions leads to traffic-splitting ratios that are *not* independent of the source of the traffic and, as such, cannot be expressed in terms of link weights.

For simplicity of exposition, our initial description assumes that we can enumerate all of the simple (i.e., loop-free) paths in the network. This is clearly inefficient, since there may be an exponential number of such paths. Fortunately, we can show that path enumeration is not necessary.

## 5.1 Network Entropy Maximization

As mentioned in Sec. 2.2, optimal traffic engineering can be realized with multi-commodity-flow routing. i.e., setting up one or multiple flow pipes (tunnel) for every pair of node and reserving an appropriate amount of bandwidth based on the result of a centralized optimization. In fact, there are numerous ways of multi-commodity-flow routing (different bandwidth reservations on the tunnels) to realize optimal traffic engineering. In the COMMODITY problem (2), if we replace link capacity $c_{u,v}$ with the necessary capacity $\widetilde{c}_{u,v}$, we are free to impose another objective function to favor a particular optimal solution to the original problem. A key challenge here is to design a new objective function such that the resulting routing of flow can be easily realized *distributively with link-state routing protocols*.

To start with, consider the approach of describing all the possible ways of flow routing and choosing the best one in terms of the new objective function. Accordingly, we first need to enumerate all the simple (i.e., loop-free) paths for every node pair [19]. Note that, the number of simple paths could be as high as the exponential function of total number of nodes in the network. Such simple-path enumeration is only for analysis purpose, and in the final solution, path enumeration is not required. Denote $P_{s,t}$ as the set of simple 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, we have $\sum_i x^i_{s,t} = 1$.

We now develop the framework of Network Entropy Maximization. More discussions on the physical meaning of entropy and the motivation of maximizing entropy for routing are presented in Appendix A.

Denote $z(x^i_{s,t})$ as the entropy function of $x^i_{s,t}$, as is common in previous studies of network entropy, which is defined as $-x^i_{s,t}\log x^i_{s,t}$. Using entropy (weighted by $D(s,t)$) as the new objective, we define Network Entropy Maximization (**NEM**) problem (9) with the necessary capacities as follows:

*NEM:*

$$\max \quad \sum_{s,t} D(s,t) \left( \sum_{P^i_{s,t}} z(x^i_{s,t}) \right) \tag{9a}$$

$$\text{s.t.} \quad \sum_{s,t,i:(u,v)\in P^i_{s,t}} D(s,t) x^i_{s,t} \le \widetilde{c}_{u,v}, \tag{9b}$$

$$\sum_i x^i_{s,t} = 1, \tag{9c}$$

$$\text{vars.} \quad 1 \ge x^i_{s,t} \ge 0. \tag{9d}$$

The NEM problem must have a feasible solution from the optimal solution of the COMMODITY problem [20]. Then the NEM problem must have a globally optimal solution, according to Lemma 1.

**Lemma** 1. A global optimal solution for the NEM problem exists and can be found within polynomial time.

**Proof**: Since the NEM problem always has a feasible solution and entropy function, $z(x^i_{s,t})$, is strictly concave, the NEM problem is a convex optimization with a compact region. Thus NEM problem must have a global optimal solution and polynomial-time solvable [29]. ■

## 5.2 Solve NEM by Dual Decomposition

The NEM problem (9) can be solved with dual decomposition. Although link-weight computation is a centralized optimization procedure, a properly designed decomposition of NEM is still highly valuable: it exploits the separability structures of NEM to show how link weights can be discovered from the optimal solutions of NEM.

Denote dual variables for constraints (9b) as $\lambda_{u,v}$ for link $(u,v)$ (or $\boldsymbol{\lambda}$ as a vector). We first write the Lagrangian $L(\boldsymbol{x},\boldsymbol{\lambda})$ associated with the NEM problem

$$\begin{array}{l} L(\boldsymbol{x},\boldsymbol{\lambda}) \\ = \sum_{s,t} D(s,t) \left( \sum_{P^i_{s,t}} z(x^i_{s,t}) \right) \\ \quad - \sum_{(u,v)\in E} \lambda_{u,v} (\sum_{s,t,i:(u,v)\in P^i_{s,t}} D(s,t) x^i_{s,t} - \widetilde{c}_{u,v}). \end{array} \tag{10}$$

The Lagrange dual function is

$$\begin{array}{rll} Q(\boldsymbol{\lambda}) = & \max & L(\boldsymbol{x},\boldsymbol{\lambda}), \\ & \mathbf{1} \succeq \boldsymbol{x} \succeq \mathbf{0} & \\ & ||\boldsymbol{x}_{s,t}|| = \mathbf{1}. & \end{array} \tag{11}$$

where $\mathbf{0}$ and $\mathbf{1}$ are the vectors whose elements are all zeros and ones respectively and $\boldsymbol{x}_{s,t}$ is the vector of $x^i_{s,t}$.

The dual problem is formulated as

$$\begin{array}{ll} \min & Q(\boldsymbol{\lambda}) \\ \text{s.t.} & \boldsymbol{\lambda} \succeq \mathbf{0}. \end{array} \tag{12}$$

To solve the dual problem, we first consider problem (11). The maximization of the Lagrangian over $\boldsymbol{x}$ can be solved as a TRAFFIC-DISTRIBUTION problem (13):

*TRAFFIC-DISTRIBUTION:*

$$\begin{aligned} \max \quad & \sum_{(u,v)\in E} \lambda_{u,v} \widetilde{c}_{u,v} + \sum_{s,t} D(s,t) \left( \sum_{P^i_{s,t}} z(x^i_{s,t}) \right) \\ & - \sum_{(u,v)\in E} \lambda_{u,v} \left( \sum_{s,t,i:(u,v)\in P^i_{s,t}} D(s,t) x^i_{s,t} \right) \end{aligned} \tag{13a}$$

$$\text{s.t.} \quad \sum_i x^i_{s,t} = 1, \tag{13b}$$

$$\text{vars.} \quad 1 \ge x^i_{s,t} \ge 0. \tag{13c}$$

Then, dual problem (11) can be solved by using the sub-gradient algorithm as

$$\begin{array}{l} \lambda_{u,v}(q+1) \\ = \left[ \lambda_{u,v}(q) - \alpha(q) \left( \widetilde{c}_{u,v} - \sum_{s,t,i:(u,v)\in P^i_{s,t}} D(s,t) x^i_{s,t}(q) \right) \right]^+, \\ = \left[ \lambda_{u,v}(q) - \alpha(q) \left( \widetilde{c}_{u,v} - f_{u,v}(q) \right) \right]^+, \\ \quad \forall (u,v) \in \mathbb{E}. \end{array} \tag{14}$$

where for iteration $q$, $\alpha(q)$ is the step size, $x^i_{s,t}(q)$ are solutions of the TRAFFIC-DISTRIBUTION problem (13) for a given $\boldsymbol{\lambda}(q)$, and $f_{u,v}(q)$ is the total flow on link $(u,v)$, which removes the *dependence* on the concrete values of $D(s,t)$ and $x^i_{s,t}$.

After the above dual decomposition, the following result can be proved using standard techniques in gradient algorithm's convergence analysis [30]:

**Lemma** 2. *By solving the TRAFFIC-DISTRIBUTION problem for the NEM problem, dual variables $\boldsymbol{\lambda}(q)$ converge to the optimal dual solutions $\boldsymbol{\lambda}^*$ and the corresponding primal variables $\boldsymbol{x}^*$ are the globally optimal primal solutions of (9).*

## 5.3 Solve TRAFFIC-DISTRIBUTION Problem

Note that, the TRAFFIC-DISTRIBUTION problem is also separable, i.e., the traffic distribution for each demand across its paths is independent of the others since they are not coupled together with link capacity constraint (like (9b)). So we can solve a subproblem (15) below for each demand $D(s,t)$ separately:

*DEMAND-DISTRIBUTION:*

$$\begin{aligned} \max \quad & D(s,t) \left( \sum_{P^i_{s,t}} z(x^i_{s,t}) \right) \\ & - \sum_{(u,v)\in E} \lambda_{u,v} \left( \sum_{i:(u,v)\in P^i_{s,t}} D(s,t) x^i_{s,t} \right) \end{aligned} \tag{15a}$$

$$\text{s.t.} \quad \sum_i x^i_{s,t} = 1, \tag{15b}$$

$$\text{vars.} \quad 1 \ge x^i_{s,t} \ge 0. \tag{15c}$$

We write the Lagrangian associated with the DEMAND-DISTRIBUTION subproblem in (16).

$$\begin{aligned} & L^r(\boldsymbol{x}_{s,t}, \mu_{s,t}) \\ = \; & D(s,t)\left(\sum_{P^i_{s,t}} z(x^i_{s,t})\right) - \mu_{s,t}(x^i_{s,t} - 1) \\ & - \sum_{(u,v)\in E} \lambda_{u,v}(\sum_{i:(u,v)\in P^i_{s,t}} D(s,t)x^i_{s,t}) \end{aligned} \tag{16}$$

where $\mu_{s,t}$ is the Lagrangian variable associated with constraint (15b).

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

$$z'(x^{i^*}_{s,t}) - \sum_{(u,v)\in P^i_{s,t}} \lambda_{u,v} - \frac{\mu^*_{s,t}}{D(s,t)} = 0. \tag{17}$$

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

$$x^{i^*}_{s,t} = e^{-(\sum_{(u,v)\in P^i_{s,t}} \lambda_{u,v} + \frac{\mu^*_{s,t}}{D(s,t)} + 1)}. \tag{18}$$

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. Combined with (15b), we can get unique values for $\mu^*_{s,t}$ and $x^{i^*}_{s,t}$.

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

$$\frac{x^{i^*}_{s,t}}{x^{j^*}_{s,t}} = \frac{e^{-(\sum_{(u,v)\in P^i_{s,t}} \lambda_{u,v})}}{e^{-(\sum_{(u,v)\in P^j_{s,t}} \lambda_{u,v})}}. \tag{19}$$

If we use $\lambda_{u,v}$ as the weight for link $(u,v)$, i.e. $w_{u,v}$, the probability of using path $P^i_{s,t}$ is inversely proportional to the exponential value of its path length. And it is easy to verify the optimal solution to DEMAND-DISTRIBUTION is equivalent to the traffic distribution by employing traffic allocation function $\Gamma_X(\cdot)$ (6) introduced in Sec. 3.1.

It is important to observe at this point that, since (19) has no factor of $\mu^*_{s,t}$, an intermediate router can ignore the source of the packet in forwarding. Thus the router can treat all the packets as being initiated from itself and distribute them across outgoing links according to $\Gamma_X(\cdot)$ (6). Equally importantly, from (14), in iteration $q$, the procedure of link price (weight) updating does not need the concrete values of $x^i_{s,t}(q)$. Instead, it just needs $f_{u,v}(q)$, the aggregated bandwidth usage, which can be quickly computed from $\Gamma_X(\cdot)$ for a given demand traffic by solving linear equations (8) shown in Sec. 3.3.

Now, the key Theorem 1 introduced in Sec 3.2 becomes an obvious conclusion from Fact 1, Lemmas 1 and 2, and (8).

Furthermore, recall that the linear equations (8) have $N^2$ equations and $N^2$ variables, which demand very efficient algorithm to be solved for large networks. From the basic observation that there exists no flow loop in the optimal traffic distribution, we can see that, in the last iteration in Algorithm 1, the loopback flow must be negligible. In fact, there could be numerous sets of dual variables $\lambda_{u,v}$ (or link weights $w_{u,v}$) to realize the optimal traffic engineering. If the link weights are large enough, the loopback flow is negligible even in intermediate iterations due to Proposition 1. Therefore, we can adopt $\Gamma_D(\cdot)$ function to approximate traffic distribution, which can be computed with a much faster combinatorial algorithm (Algorithm 4). In practice (Sec. 4), the approximate solution with $\Gamma_D(\cdot)$ also can achieve near optimal traffic engineering. Therefore, to use $\Gamma_D(\cdot)$, we need to specify a large $w_{min}$, e.g. 10, as in our numerical experiments, and use a set of link weights with large values as the starting point for solving the NEM problem.

**Proposition** 1. If the smallest link weight $w_{min}$ is large enough, the loop flow is negligible in an optimal solution of the TRAFFIC-DISTRIBUTION problem

**Proof**: For link $(u,v)$, if the shortest distance to $t$ of $u$, $d^t_u \le d^t_v$, then $h^t_{u,v} = d^t_v + w_{u,v} - d^t_u \ge w_{u,v}$ and $\Gamma_X(h^t_{u,v}) \ge e^{-w_{u,v}}$, and the flow destined to $t$ on $(u,v)$ is close to 0 if $w_{u,v}$ is large enough, e.g., $e^{-10} \approx 0.005\%$. Since most flow of optimal solution always makes forward progress towards the destination, i.e., from the router with larger $d^t_u$ to the router with smaller $d^t_u$, the loopback flow can be neglected.■

## 6. ROBUST TRAFFIC ENGINEERING

In robust traffic engineering, one set of link weights must perform well for a family of traffic matrices or failure scenarios. Compared to the case with single traffic matrix, selecting the link weights in robust traffic engineering is more difficult or even impossible [4] under OSPF. In this section, we first generalize multiple traffic matrices and failures with a unifying notation of *scenario*, and then extend the result for a single, given traffic matrix in Sec. 5 to robust traffic engineering with multiple scenarios.

We index the scenarios by $\theta$, and denote $\mathbb{E}^\theta$ and $D^\theta(s,t)$ as the available edge set and demands for scenario $\theta$ respectively. Obviously, $\theta$ covers the cases with multiple traffic matrices (where $D^\theta(s,t)$ are different among scenarios) and link failures (where $\mathbb{E}^\theta \subset \mathbb{E}$). Note that a node failure can be modeled by a scenario where all of its incident links have failed.

Define $\Phi^\theta$ as $\sum_{u,v\in\mathbb{E}^\theta} \Phi(f^\theta_{u,v}, c_{u,v})$, the sum of link costs for scenario $\theta$, where $f^\theta_{u,v}$ is the total flow on link $(u,v)$ for scenario $\theta$. There are many possible types of convex compositions of $\Phi^\theta$. The simplest type is $\sum_\theta \Phi^\theta$, e.g., for multiple traffic matrices. Let $\theta = 0$ (with $\Phi^0$) denote a normal scenario, $\theta = e$ (with $\Phi^e$) denote the scenario with a failure on link $e$. Then we can get different optimization problems using different types of convex composition as follows.

| | **Congestion Control (TCP)** | **Traffic Engineering (IP)** |
|---|---|---|
| Traffic type | Elastic | Inelastic |
| Flow distribution | Fixed | Variable |
| Participants | End user and router | Operator and router |
| Timescale | Seconds | Hours |
| Mathematical framework | Network Utility Maximization | *Network Entropy Maximization* |
| Reverse engineering | Tahoe, Reno, Vegas, etc. | *Even splitting over shortest paths in OSPF* |
| Forward engineering | FAST TCP, etc. | *DEFT* |
| Role of Lagrange multipliers | Feedback prices | *Penalty weights* |
| Important implications | New, stabilized TCP protocols | *Efficient algorithm to compute link weights achieving optimal TE* |

**Table 6: Comparison of Congestion Control in TCP and IP layer (Our contributions in *italics*)**

- $\min \Phi^0$: Traffic engineering for non-failure scenario.
- $\min \Phi^0 + \mu \sum_{e\in\mathbb{E}} \Phi^e$: Weighted traffic engineering considering single link failure, where $\mu$ is a parameter weighing the seriousness of a single failure [4].
- $\min \max_{\theta\in\mathbb{E}} \Phi^\theta$: The worst performance of a single failure.

Denote $\Phi(\{f^\theta_{u,v}\})$ as a convex function coming from any compositions of the functions $\Phi(\theta)$ upon multiple scenarios. Then we have a ROBUST-COMMODITY problem (20) below to find optimal robust traffic engineering with the minimum aggregated $\Phi$ cost:

*ROBUST-COMMODITY:*

$$\min \quad \Phi(\{f^\theta_{u,v}\}) \tag{20a}$$

$$\text{s.t.} \quad \sum_{z:(y,z)\in\mathbb{E}^\theta} f^{t,\theta}_{y,z} - \sum_{x:(x,y)\in\mathbb{E}^\theta} f^{t,\theta}_{x,y} = D^\theta(y,t), \forall y \neq t \tag{20b}$$

$$f^\theta_{u,v} = \sum_{t\in\mathbb{V}} f^{t,\theta}_{u,v}, \tag{20c}$$

$$\text{vars.} \quad f^{t,\theta}_{u,v}, f^\theta_{u,v} \geq 0. \tag{20d}$$

From the optimal solution to the ROBUST-COMMODITY problem (20), we redefine the necessary capacity, $\widetilde{c}_{u,v} = \sum_\theta f^\theta_{u,v}$. Similarly, we can extend the NEM problem (9) to a ROBUST-NEM problem (21) below:

*ROBUST-NEM:*

$$\min \quad \sum_\theta \sum_{s,t} D^\theta(s,t) \left( \sum_{P^{i,\theta}_{s,t}} z(x^{i,\theta}_{s,t}) \right) \tag{21a}$$

$$\text{s.t.} \quad \sum_\theta \sum_{s,t,i:(u,v)\in P^{i,\theta}_{s,t}} D^\theta(s,t) x^{i,\theta}_{s,t} \leq \widetilde{c}_{u,v}, \tag{21b}$$

$$\sum_i x^{i,\theta}_{s,t} = 1, \tag{21c}$$

$$\text{vars.} \quad 1 \geq x^{i,\theta}_{s,t} \geq 0. \tag{21d}$$

We can extend the approaches in Sec. 5. In particular, the ROBUST-NEM problem must have a global optimal solution and the Lagrangian dual variables, $\boldsymbol{\lambda}$, can be used as link weights. Then we have Theorem 2 blow. The proof is omitted due to space limitation.

**Theorem** 2. *Optimal robust traffic engineering for a convex composition of various scenarios (including multiple traffic matrices and failures) can be realized with link weights, computed within polynomial time, by using traffic allocation function $\Gamma_X(\cdot)$ in link-state routing protocols.*

## 7. NEM: A FRAMEWORK FOR LINK-STATE ROUTING

Before concluding the paper, we would like to highlight the important differences between Network Entropy Maximization (NEM) and Network Utility Maximization (NUM), as well as the interesting parallels between the two.

As explained in Section 5, NEM is developed in this paper as a unifying mathematical model that both recovers existing link-state routing protocols (e.g., OSPF) as a special case of solution, and enables the discovery and development of new ones (e.g., DEFT). More discussions on the intriguing intuitions behind NEM can be found in Appendix A.

On the other hand, TCP congestion control protocols have been studied extensively since 1998 as solutions to another family of optimization models called NUM. The notion of network utility was first advocated in the seminal paper [31] in 1995 for bandwidth allocation among elastic demands on *source rates*. The NUM problem (22) was first introduced for TCP congestion control (e.g., [15–18]). Consider a communication network with $L$ logical links, each with a fixed capacity of $c_l$ bps, and $S$ sources (i.e., end users), each transmitting at a source rate of $x_s$ bps. Each source $s$ emits one flow, using a fixed set $L(s)$ of links in its path, and has an increasing (and often concave) function $U_s(x_s)$

called utility function. Each link $l$ is shared by a set $S(l)$ of sources. NUM, in its basic version, is the following problem of maximizing the network utility $\sum_s U_s(x_s)$, over the source rates $\mathbf{x}$, subject to linear flow constraints $\sum_{s\in S(l)} x_s \le c_l$ for all links $l$ (Note that routing is fixed in NUM formulation):

$$\begin{array}{ll} \text{maximize} & \sum_s U_s(x_s) \\ \text{subject to} & \sum_{s\in S(l)} x_s \le c_l,\ \forall l, \\ \text{variables} & \mathbf{x} \succeq 0. \end{array} \tag{22}$$

There is a useful economics interpretation of the dual-based distributed algorithm for NUM, in which the Lagrange dual variables can be interpreted as shadow prices for resource allocation, and each end user and the network maximize their net utilities and net revenue, respectively. Many reverse-engineering of existing TCP variants and forward-engineering of new congestion control protocols have been developed with the NUM model as a starting point.

The NEM problem proposed in this paper is *not* a special case of NUM, since entropy is not an increasing function, and the design freedom in NEM is routing rather than rate control. Instead, there is a useful and interesting *parallelism* between the framework of NEM proposed this paper, for link-state routing protocols in Layer 3, and that of NUM matured over the last decade, for end-to-end congestion control protocols in Layer 4. The comparison between the two frameworks is shown at Table 6, where results achieved in this paper are highlighted in italics.

## 8. CONCLUDING REMARKS

Link-state routing protocols such as OPSF are simple, while commodity-flow-based routing protocols such as commodity-flow-based routing is optimal. Many years of evidence suggest that optimal traffic engineering *cannot* be realized by the simple link-state protocols that let each router independently forward packets based just on a set of link weights, no matter how smart the link-weight computation is.

This paper proves that optimal traffic engineering, in fact, *can* be achieved by link-state routing, and the right link weights *can* be computed efficiently (polynomial-time in theory and very fast in practice), as long as flow splitting on non-shortest paths is allowed but properly penalized. In terms of algorithmic methods, this paper proposes highly efficient algorithms to compute link weights that lead to optimal traffic engineering through link-state routing, and develops a new proof logic, several proof techniques, and the framework of Network Entropy Maximization. In terms of practical implications, the paper shows how to compute link weights that gives much lower total link cost, or maximum link utilization, than OSPF local search, within a much shorter amount of computation time.

# Appendix

## A Entropy Maximization and Most Likely Flow Configuration

There are several intriguing relationships between the framework of Network Entropy Maximization for link-state routing and statistical physics. We speculate some of the thought-provoking connections in this appendix.

In classical statistical mechanics, many microscopic behaviors aggregate into macroscopic states, and an isolated thermodynamic system will eventually reach an equilibrium macroscopic state that is the most likely one. Interestingly, entropy maximization for traffic engineering can be motivated by an argument of most likely flow configuration, as shown below.

Consider a network with only just one source-destination $(s,t)$ and $P$ un-capacitated paths between them. If there are $T$ packets to be transmitted from $s$ to $t$, let $T_i \ge 0$ be the number of packets on path $i$, with $\sum_i T_i = T$. Each set of such $\{T_i\}$, which can be represented as a vector, is referred to as a *macroscopic state*. In contrast, each collection of routing decisions for individual packets represents a *microscopic state*. There are a total of $P^T$ possible microscopic states. The number of microscopic states consistent with a given macroscopic state can be viewed as a measure of likelihood of that macroscopic state.

The number of microscopic states corresponding to the macroscopic state $\{T_i\}$ is $K = \frac{T!}{\prod_i T_i!}$. We want to search for the macroscopic state with the largest number of $K$, i.e., $\max K$, or, equivalently, $\max \log K = \max \log \frac{T!}{\prod_i T_i!}$. For large system asymptote, $T$ and $T_i$ are large numbers, hence using Stirling's approximation: $n! \approx n^n\, e^{-n}$, we have $\log K \approx \log(e^{-T}T^T) - \sum_i \log(e^{-T_i}T_i^{T_i}) = -T\sum_i \frac{T_i}{T}\log\frac{T_i}{T}$.

This shows that the system equilibrium is the flow configuration that maximizes the entropy, $-\sum_i Tx_i \log x_i$, where $x_i = \frac{T_i}{T}$ is the fraction of flow on path $i$.

An interesting discovery made through the developments of this paper is that putting entropy in the objective function of problem (9) can lead to a flow configuration that both (i) optimizes the $\Phi$ cost function and (ii) is realizable by link weights. This is an important step in showing that optimal traffic engineering is realizable through link-state routing. And there are even clues suggesting that entropy is the *unique* objective function for problem (9) satisfying both (i) and (ii). Similarly, exponential flow splitting function $\Gamma_X(\cdot)$ seems to be the unique function to realize optimal traffic engineering. Proving these uniqueness properties is an interesting next step of research.

## B Implementing Non-Shortest Routing with Shortest Routing

As long as link weights can be arbitrarily set, there

is no significant difference, in terms of implementation, between non-shortest routing and shortest routing since the former can be realized by the latter. Note that, any flow routing, subject to the necessary capacities, is also the optimal solution of (23) [7] below since its optimal objective value is only related with the necessary capacities, i.e., $\sum_{(u,v)\in\mathbb{E}} \widetilde{c}_{u,v}$.

$$\min \quad \sum_{t\in\mathbb{V},(u,v)\in\mathbb{E}} f^t_{u,v} \tag{23a}$$

$$\text{s.t.} \quad \sum_{z:(y,z)\in\mathbb{E}} f^t_{y,z} - \sum_{x:(x,y)\in\mathbb{E}} f^t_{x,y} = D(y,t), \forall y \neq t \tag{23b}$$

$$f_{u,v} \triangleq \sum_{t\in\mathbb{V}} f^t_{u,v} \leq \widetilde{c}_{u,v}, \tag{23c}$$

$$\text{vars.} \quad f^t_{u,v}, f_{u,v} \geq 0. \tag{23d}$$

The dual problem for (23) is (24) below [7]:

$$\max \quad \sum_{t\in\mathbb{V},y\in\mathbb{V}:y!=t} D(y,t)\, d^t_y - \sum_{(u,v)\in\mathbb{E}} \widetilde{c}_{u,v}\, \widehat{w}_{u,v} \tag{24a}$$

$$\text{s.t.} \quad d^t_u - d^t_v \leq \widehat{w}_{u,v} \tag{24b}$$

$$d^t_t = 0 \tag{24c}$$

$$\text{vars.} \quad d^t_u \geq 0, \widehat{w}_{u,v} \geq 0. \tag{24d}$$

From [7, 8], solving (24) will give us a set of link weights $\widehat{\boldsymbol{w}}$ such that the traffic destined to $t$ will only be sent along shortest paths to $t$. Thus to realize non-shortest path routing (like DEFT) with shortest routing, we can let routers broadcast two sets of link weights: $\widehat{\boldsymbol{w}}$ and $\boldsymbol{w}$. $\widehat{\boldsymbol{w}}$ is used in shortest paths calculation for the decision on forwarding or not while $\boldsymbol{w}$ is for traffic allocation function (like $\Gamma_D(\cdot)$) to determine how much to forward.

# 9. REFERENCES


[1] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *INFOCOM'00, Tel Aviv, Israel*, 2000, pp. 519–528.

[2] ——, "Increasing Internet capacity using local search," *Computational Optimization and Applications*, vol. 29, no. 1, pp. 13–48, 2004.

[3] D. H. Lorenz, A. Orda, D. Raz, and Y. Shavitt, "How good can IP routing be?" DIMACS, Tech. Rep. 2001-17, May 2001.

[4] A. Sridharan and R. Guérin, "Making IGP routing robust to link failures." *Networking*, pp. 634–646, 2005.

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

[6] D. Awduche, "MPLS and traffic engineering in IP networks," *IEEE Communication Magazine*, vol. 37, no. 12, pp. 42–47, Dec. 1999.

[7] 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.

[8] Z. Wang, Y. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying," in *INFOCOM'01, Anchorage, Alaska*, Apr. 2001.

[9] 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.

[10] D. Xu, M. Chiang, and J. Rexford, "DEFT: Distributed exponentially-weighted flow splitting," in *IEEE INFOCOM'07, Anchorage, Alaska, May. 2007, http://www.princeton.edu/~dahaixu/pub/deft/deft.pdf.*

[11] W. R. Blunden, *Introduction to traffic science.* Printerhall, London, 1967.

[12] J. A. Tomlin and S. G. Tomlin, "Traffic distribution and entropy," *Nature*, vol. 220, pp. 974–976, 1968.

[13] J. A. Tomlin, "A new paradigm for ranking pages on the world wide web," in *WWW '03: Proceedings of the 12th international conference on World Wide Web.* New York, NY, USA: ACM Press, 2003, pp. 350–355.

[14] A. K. Agrawal, D. Mohan, and R. S. Singh, "Traffic planning in a constrained network using entropy maximisation approach," *Journal of the Institution of Engineers, India. Civil Engineering Division*, vol. 85, pp. 236–240, 2005.

[15] F. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, Mar. 1998.

[16] H. Yäiche, R. R. Mazumdar, and C. Rosenberg, "A game theoretic framework for bandwidth allocation and pricing in broadband networks," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 667–678, 2000.

[17] S. H. Low, "A duality model of TCP and queue management algorithms," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 525–536, 2003.

[18] R. Srikant, *The Mathematics of Internet Congestion Control (Systems and Control: Foundations and Applications).* Springer Verlag, 2004.

[19] D. Bertsekas and R. Gallager, *Data networks.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1987.

[20] 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.

[21] A. Tveit, "On the complexity of matrix inversion," *Mathematical Note, IDI, NTNU, Trondheim, Norway*, Nov. 2003.

[22] T. Corman, C. Leiserson, and R. Rivest, *Introduction to Algorithms.* The MIT Press, Cambridge, 1990.

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

[24] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming.* Danvers, MA, USA: Boyd & Fraser Publishing Co., 1993.

[25] TOTEM, http://totem.info.ucl.ac.be.

[26] Abilene Backbone Network, http://abilene.internet2.edu/.

[27] R. Teixeira, T. G. Griffin, M. G. C. Resende, and J. Rexford, "TIE breaking: Tunable interdomain egress selection," *IEEE/ACM Transactions on Networking*, Oct. 2007.

[28] M. G. Resende and P. M. Pardalos, Eds., *Handbook of Optimization in Telecommunications.* New York: Spinger Science + Business Media, February 2006.

[29] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, 2004.

[30] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Athena Scientific, 1999. [Online]. Available: http://www.athenasc.com/nonlinbook.html

[31] S. Shenker, "Fundamental design issues for the future internet," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 13, no. 7, pp. 1176–1188, September Sep. 1995.