Algorithmica (2005) 43: 113–131
DOI: 10.1007/s00453-005-1161-2

Algorithmica

Better Alternatives to OSPF Routing[1]

Jessica H. Fong,[2] Anna C. Gilbert,[3] Sampath Kannan,[4] and Martin J. Strauss[3]

Abstract. The current standard for intra-domain network routing, Open Shortest Path First (OSPF), suffers from a number of problems—the tunable parameters (the *weights*) are hard to optimize, the chosen paths are not robust under changes in traffic or network state, and some network links are over-used at the expense of others.

We present prototypical scenarios that illustrate these problems. Then we propose several variants of a protocol to eliminate or alleviate them and demonstrate the improvements in performance under those scenarios. We also prove that these protocols never perform significantly worse than OSPF and show that for at least a limited class of network topologies, it is possible to find efficiently the optimal weight settings.

Some of the problems with OSPF are well known; indeed, there are several routing protocols that perform better than OSPF in routing quality (i.e., in terms of congestion, delay, etc.). OSPF's popularity persists in part because of its efficiency with respect to several resource bounds. In contrast, many competing protocols that provide routing superior to OSPF are computationally prohibitive. Motivated by this consideration, we designed our protocols not only to achieve better routing quality than OSPF, but also to use resources in amount comparable with OSPF with respect to offline broadcast communication, size of and time to compute routing tables, packet delivery latency, and packet header structure and size.

Key Words. Shortest path routing, Intra-domain routing, Network optimization.

1. Introduction.

The Open Shortest Path First (OSPF) protocol, developed by the Internet Engineering Task Force for IP networks, is the most common interior gateway protocol (IGP) today. The current version, OSPF-2, is described in [7]. Routers in an autonomous system (AS), a group of routers that chooses its routing protocol independently of other ASs, share one IGP. For our purposes, an AS consists of routers connected by directed, weighted links. A routing protocol specifies how packets flow through a network to satisfy source and destination constraints. Briefly, in OSPF, routers remember the same "link state" of the network, i.e. active links and reachable neighbors, and periodically flood other routers to advertise changes. A router computes lowest-weight paths to each destination and distributes flow to that destination across some subset of

[1] This work was done while the first author was visiting AT&T Labs (supported by the AT&T Labs Fellowship Program), the second and fourth authors were at AT&T Labs, and while the third author was at AT&T Labs, on leave from the University of Pennsylvania.
[2] Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08544, USA. jfong@cs.princeton.edu.
[3] Department of Mathematics, University of Michigan, 530 E. Church Street, Ann Arbor, MI 48109, USA. {annacg,martinjs}@umich.edu.
[4] Computer and Information Sciences, University of Pennsylvania, Philadelphia, PA 19104, USA. kannan@central.cis.upenn.edu.

Received December 2003; revised July 2004. Communicated by L. Zhang.
Online publication June 21, 2005.

neighbors (“next hops”) lying on these paths. That subset may include all or only some of the neighbors. A number of OSPF implementations choose only one neighbor. Regardless of the size of this subset, all flow is split evenly.

Routing algorithms have several theoretical goals, most notably to optimize the quality of routing. One goal is to minimize the route length traversed by each packet. As we describe formally in what follows, the path length traversed by a packet is the weighted length of the path it traverses, where the weights may be assigned to model physically significant phenomena. Another goal is to minimize congestion in the network. Intuitively, congestion occurs when traffic streams are mapped onto available paths inefficiently, causing over-utilization of some paths while leaving others under-utilized. Thus we define a total cost function to model congestion and we seek to minimize this cost function. We refer to these qualities of the routing as static qualities and develop a set of static routing criteria.

The above measures of quality assume that the network is static, that demands do not change, and that nodes and links do not enter or leave the network. One errant backhoe can wreak havoc on a highly optimized network. Similarly, taking a router offline to update software or to prevent the spread of a virus can generate tremendous congestion in a network. Routing protocols must be able to function in a dynamic network environment and, as such, we define one dynamic routing quality.

A protocol is also evaluated on its use of computational resources, including the time required to generate routing tables and the space necessary to store these tables. For example, the OSPF protocol comprises several algorithms and data structures—a protocol by which routers broadcast link states,[5] a common collection of edge weights, an algorithm by which each router locally computes a *routing table* (that tells the router to which of several neighbors a packet should be forwarded, based on the packet’s destination and other information), and a standard structure for packet headers. OSPF remains popular despite its disadvantages in part because these algorithms run quickly and the routing table size is limited. To separate the quality of a routing protocol from how that protocol achieves that quality, we call these qualities algorithmic criteria. With the exception of [1], [4] and [5], few papers on improvements to OSPF provide rigorous analysis of the performance trade-offs or algorithmic qualities of a protocol. Although OSPF is the standard in intra-domain routing, little work (until [4]) has been done on optimal weight setting, for example.

In this paper we propose several alternative intra-domain routing protocols and analyze them from a theoretical perspective. Our protocols use the same general structure and mechanisms as OSPF (limited broadcast, routing table lookup, standard IP headers, etc.), and use either the same amount of resources as OSPF or just slightly worse (we give rigorous bounds). This allows us to compare the quality of our routings with those of OSPF; also, we can appeal to the popularity and success of OSPF when considering the practicality of implementing our algorithm. We evaluate them under static, dynamic, and algorithmic criteria.

A more detailed treatment of the overall goals, objective functions, and algorithms

[5] Other functions are performed in a decentralized way—an advantage of OSPF over routing protocols with more centralized control.

used by routing protocols is given in the next section. However, we briefly describe the results in this paper.

- We present a class of routing protocols as alternatives to OSPF. Our protocols split flow to destination t along various next hops n with probabilities that depend on the weights of the lightest paths through n to t, taking into account two values, τ' and k. The value τ' is the currently remaining time-to-live of the packets in the flow and k is a global parameter which, informally, specifies the degree to which we use non-shortest paths to t.
- Using prototypical examples, we demonstrate scenarios where our protocols achieve significantly better values for the objective function, a proxy for congestion.
- We show that for any possible input topology and demand matrix, our protocols can be tuned to be no worse than OSPF in various performance measures.
- We point out some of the drawbacks of OSPF such as the way the chosen routes and their cost can change dramatically with small changes in the tunable parameters or link connectivity state. We show that our protocols do not suffer from these problems.
- We show that for a class of inputs including single-sink demands, it is possible to set the weights optimally for any of our protocols. Interestingly, this class of inputs includes inputs on which, for OSPF, the corresponding problem (and even a relaxed approximate version) is NP-hard.
- We show that our protocols use an amount of computational resources comparable with that used by OSPF.

2. Routing Protocols

2.1. *Background*. A routing protocol specifies how packets flow through the network, to satisfy some demand. Its input is as follows: We are given a capacitated directed graph $G = (V, E, c)$, describing the physical topology of a network, and a demand matrix D. In G, V is the set of nodes in the network, E is the set of links between the nodes, and c is a capacity function mapping arcs to integers. The entry $D(s, t)$ specifies the rate at which packets with source s are injected, bound for destination t. We assume that the demand matrix represents steady-state demands.

Several numerical quantities are associated with each arc, namely, capacity, load, utilization, and local cost. The *capacity*, given as input, is a physical property of the link. The *load* represents actual traffic over a link and is dependent on the routing protocol used and the traffic demand. Each of capacity and load is specified in bytes per second. Their dimensionless ratio is the *utilization* of the arc. The *local cost* at each edge is a metric of optimality. It is a function of edge utilization and, possibly, varies from edge to edge—for example, the local cost could be the utilization itself, the square of the utilization, or $b_a u_a^2$, where u_a is the utilization of arc a and b_a is an arbitrary constant, capturing the importance of arc a. (Note that this differs from the assigned *weight* at each edge.) The local costs are combined into a total cost of the routing, by summing the local costs or taking the maximum of the local costs.

We assume that packets resemble IP packets and have a time-to-live field which specifies the remaining number of edges a packet can traverse (i.e., its hop count) before it is dropped. Each router decrements the time-to-live field upon forwarding it to a neighbor. If the time-to-live expires, the router simply drops the packet.

The protocols we consider have the following three-part structure.

- Weights are assigned to arcs and changed infrequently (e.g., in response to long-term trends in network traffic and topology).
- Next, the routers periodically broadcast to each other the network's *link state* (which links are currently up) or when a link goes down. From the link state and the known weights, each router R computes and stores a local *routing table* that specifies, for each destination router t, to which neighbor(s) of R packets bound for t should be forwarded.
- Finally, when a packet arrives at R bound for t, R consults its routing table and the packet's time-to-live field and sends the packet to one of the active neighbors or drops the packet.

All protocols that we consider assume that positive weights have been assigned to the links of the network in some fashion. To avoid confusion we use the terms *hop count* and *depth* and speak of *shallow* and *deep* paths when we talk about the unweighted graph. We use the terms *weight* and (less frequently) *length* to refer to the weighted graph, thus the *lightest* or *shortest* paths refer to paths of least total weight.

2.2. *Routing Protocol Criteria.* We can evaluate a routing protocol using a variety of metrics. We divide these metrics into three categories: the static quality of the routing, the dynamic quality of the routing, and finally the algorithmic quality of the protocol. To assess the static routing quality, we assume that the demand is static and evaluate individual performance metrics such as congestion in the network. For the dynamic routing quality, we evaluate how well a particular static solution holds up under changing or uncertain conditions. The algorithmic quality of the routing evaluates the computational resources used by the protocol to effect the routing and how that performance compares with the optimal routing.

Static Routing Quality. Assuming a given static demand, we can evaluate the latency, or the expected transmission time, that packets experience under the protocol. The latency depends on two factors: *queuing delay*, or the amount of time a packet spends waiting in various queues, and *transmission delay*, the time for a packet to travel across its chosen links. We can also determine what fraction of packets must be retransmitted because they were dropped or lost. Unfortunately, both queuing delay and packet loss are hard to determine analytically or empirically so we frequently use congestion as a proxy for these measures. Here, congestion is measured in several ways. Some common measures are, first, $\sum_a b_a(u(a))^2$, where $u(a)$ represents the utilization on arc a and b_a is an arbitrary constant; and, second, $\max_a u(a)$. Finally, we can calculate the weight of the paths taken by a packet and compare that weight with that of the shortest path (the optimal weight being the shortest path).

Dynamic Routing Quality. All of the above criteria are static in the sense that we assume that we have an unchanging traffic pattern and network topology. We take the first few steps towards a more realistic dynamic view by considering the following situations for specific inputs: (1) What happens to a protocol's performance when a node or link goes down? (2) What happens when the protocol is run with weights that have been optimized for demand matrix D and the demand matrix changes to a new one, D'? We introduce the *robustness* of a protocol: assuming weights that have been optimized for one particular topology and demand matrix, how much does the routing produced by a protocol change when there is a small change in the demand matrix or in the topology?

Algorithmic Quality. The protocol must perform the above tasks under greatly different computational limitations. The protocol must optimze the edge weights. Often this is a difficult task, one that may take days as long as they are feasible. We can ask how easy it is to produce a "good" settings of the weights that make the protocol perform well. We can also evaluate how close the performance is with optimal settings to the optimal solution for the general routing problem (i.e., multi-commodity flow). Next, the computation of the routing tables, performed by the routers, occurs periodically (on the order of an hour) and must be relatively efficient. Furthermore, space in routing cards is expensive, so routing table sizes should be kept small. Finally, when a packet arrives, there is just enough time for a router to look up a list of allowable next hops from a pre-computed table.

2.3. *Existing Protocols*

OSPF. The input for OSPF is the standard input that we described above. OSPF works as follows: The routers share edge weights. Periodically the routers agree on the link state of the network via broadcast. Each router locally performs Dijkstra's shortest path algorithm; for each destination t, the router v determines which of its neighbors lie along the lightest path to t (or which set of neighbors lie along equally light paths). This next-hop by destination information is stored in routing tables. Upon receiving a packet bound for destination t, the router v forwards the packet to the neighbor along the lightest path to t. In case there is more than one neighbor with this property, v splits the traffic bound for t evenly among these next hops. This last property of OSPF is crucial—by this mechanism, OSPF can spread traffic along many edges to avoid congestion—and several heuristic weight-setting procedures try to exploit it.

As specified, OSPF guarantees that packets reach their destinations with optimum latency. It does not show how to set weights or suggest an objective function to optimize. Fortz and Thorup have shown how to set up link costs and approximate a solution close to optimal for minimizing maximum link utilization; nevertheless, approximating the optimum weight setting within a constant factor, given the network topology, capacities, and demand, is an NP-hard problem [4]. Cisco recommends that the weights be set inversely proportional to link capacities [3], but this ignores an optimization criterion. Also, the weights should account for demand and adapt for changes in demand.

OSPF-OMP. The Optimized Multi-Path version of OSPF [9] allows the network to adapt to changing link utilization by dynamically assigning weights based on current link utilization. Thus OSPF-OMP can potentially alleviate congestion, compared with ordinary OSPF. Utilization-sensitive routing, however, requires greater link-state information exchange and, more importantly, can destabilize the network. To our knowledge, there have been no rigorous comparisons between OSPF-OMP and OSPF with respect to destabilization or congestion. We do not compare our protocol with OSPF-OMP for this reason, and because the main advantage of OSPF-OMP is its ability to adapt to changing utilization while we are addressing the more basic problem of optimizing static routing for steady-state demand.

Theoretical Protocols. Several theoretical protocols address the issue of network congestion. Aiello et al. present a deterministic, adaptive, and distributed routing protocol that avoids "traffic jams" in that each router needs to buffer only a fixed number of packets [1]. Unlike the above-mentioned algorithms and ours, this protocol focuses on minimizing differences in buffer sizes at the nodes rather than on shortest paths, under an adversarial queuing model. It also requires more communication between the routers.

Wang and Crowcroft [10] also present a deterministic routing protocol which is a modification of shortest-path routing algorithms. Their protocol, called shortest path first with emergency exits (SPF-EE), sends packets along shortest paths and, only in the face of congestion, does it resort to emergency exits or alternate paths that temporarily bypass the congested area. This protocol does not split flows to alleviate congestion, it deviates from a shortest path first protocol only if there is congestion in the network. Furthermore, it uses only one alternate path at a time rather than sending packets along different alternate paths.

Bak and Cobb [2] present a randomized routing protocol which is similar in spirit to the original randomized routing of Valiant and Brebner [8]. In this scheme the source s picks a random node e and routes from s to e and then from e to the destination d along shortest paths. A parameter k is used to limit the maximum distance e is from s. In this protocol no allowance is made for the different link capacities on different edges.

We do not discuss these algorithms in greater detail, as the algorithms differ widely from ours in their models, goals, and underlying techniques.

3. Our Protocols.

We present a class of routing protocols that compromise, in a tunable way, between minimizing weighted hop-count and minimizing total network cost. Rather than sending all the packets in a greedy fashion along the shortest paths, our protocols split flow from source s to destination t among several paths. We propose that some use of non-shortest paths (of which loops are special cases) can be done in a controlled way and can go a long way toward alleviating congestion. The fraction of flow sent to various neighbors n_i depends on the weights of paths from s through the n_i to t and on the current time-to-live, τ', of packets in the flow. Several variants are possible; we can use the time-to-live to determine the proportion of the flow to send along the paths or we can use the time-to-live to determine along which paths the flow is routed. We define three protocols below.

Class A Protocols. At vertex s, for flow to destination t, let w_n denote the weight of the lightest path from s to t via neighbor n. Split flow among neighbors $n_1, \ldots, n_\delta$ in proportion $g(w_{n_1}) : \cdots : g(w_{n_\delta})$, where $g(w) = w^{-\alpha}$ and α is a positive constant. This protocol favors light paths over heavy ones. While these protocols are perhaps too simple, they do, however, typify our protocols in a clear way for comparison with OSPF and (for $\alpha = 1$) they are easy to analyze by hand.

Class B Protocols. Class A protocols do not account for packet expiration. To remedy this, we incorporate the current time-to-live of a packet into our protocol. Let τ' denote the current time-to-live of a packet and let β and k be tunable parameters (which may be global or specific to a packet). Split flow among neighbors $n_1, \dots, n_\delta$ in proportion $g(w_{n_1}) : \cdots : g(w_{n_\delta})$, where $g(w) = e^{-k(\beta-\tau')^2 w}$. The weight of the paths favored by this protocol depends on the time-to-live of the packet; as the packet is about to expire, the favorableness of the light paths increases.

Class C Protocols. Class B protocols do not ensure that all packets are delivered (subject to adequate buffer sizes). We adjust our protocol slightly and use the time-to-live value not to split the flow, but rather to prune the set of paths over which to split. Let $w_{t,n,\tau'}$ denote the weight of the lightest path of depth at most τ' to t via n. Split flow among neighbors $n_1, \dots, n_\delta$ in proportion $g(w_{t,n_1,\tau'}) : \cdots : g(w_{t,n_\delta,\tau'})$, where $g(w) = e^{-kw}$, and k is a tunable global parameter.

4. Static Routing Criteria.

The static routing performance of a routing protocol is the first natural set of criteria with which to evaluate the protocol. How well does the protocol get packets to their destinations? Rather than struggle to analyze the queueing delay and packet loss of our proposed protocols, we use congestion as a stand-in for these measures. We show two families of inputs to OSPF in which OSPF performs significantly worse than our proposals. Next, we show that the weight of the paths taken by a packet under our protocols is not much worse than the optimal weight (i.e., the weight of the shortest path).

4.1. *Congestion.*

The two main differences between OSPF and our proposed randomized protocols are (1) OSPF considers exactly the shortest paths, whereas we potentially route over all paths; and (2) OSPF divides flow evenly among the candidate neighbors, whereas we potentially divide flow with more varied ratios. Below, we show various situations where our proposed algorithm yields a lower cost (i.e., smaller objective function value) than OSPF. We use various *gadgets*, or inputs to OSPF. Unless otherwise specified, we optimize under the follow situations:

- The objective function is the maximum utilization over edges. In other words, the problem is to minimize Φ where $\Phi = \max_{e \in E} \ (load(e)/capacity(e))$.
- The splitting function $g(w) = 1/w$ is from our Class A protocol with $\alpha = 1$.

We choose Φ and g for clarity of exposition; our examples demonstrate the same concepts for the other cost functions and splitting classes, using weights optimized appropriately.

4.1.1. *Ladder.*

This family of networks shows the problem with splitting traffic equally among next hops that lead to shortest paths. The *ladder* is a family of gadgets where the nth member of the family has two directed paths of length $n-1$, and links along those paths have infinite capacity. There is a link from each ith vertex on the first path to the ith vertex of the second path, of unit capacity. Figure 1(a) shows the sixth member of the ladder family. Under an optimal weight setting for OSPF (where all source-sink paths have the same weight), OSPF sends half of the traffic downwards and half to the right

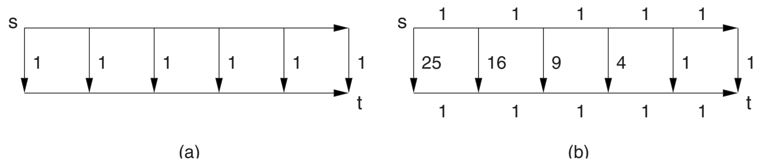

Fig. 1. Gadget LADDER (a), with our optimum weights (b).

at each node on the first path. This causes the edge from the source to the first vertex on the second path to have utilization $\frac{1}{2}$ which is also the maximum utilization.

Any of our protocols describe weights that can be set so that an equal fraction of the traffic is sent on each of the links from the first path to the second, resulting in a utilization of $1/n$ for the nth ladder. The weight setting in Figure 1(b) shows this for the sixth ladder with a Class A protocol. At the ith vertex from the sink (on the top path), we send flow down and right in a $1/(i^2+i) : 1/(i+1)$ or $1 : i$ ratio.

4.1.2. *Tree with Leaves Tied Together.* Another family of gadgets that demonstrates the same problems with OSPF is the family of complete binary trees with all the leaves connected by infinite capacity links to a sink (see Figure 2). (The dth member of the family is a depth d tree.) At each node, the capacity of the link to the left child is one-third the capacity of the edge from the parent, and the capacity of the link to the right child is two-thirds of the capacity of the edge from the parent.

In OSPF, at any node v the cost of routing a flow of f through the left subtree of v is the same as the cost of routing a flow of $2f$ at the right subtree of v since all capacities in the right subtree are scaled up by a factor of 2 over the "corresponding" capacities in the left subtree. Furthermore, the cost of routing f through the link to the left child is the same as the cost of routing $2f$ through the link to the right child. Thus, it does not matter whether OSPF splits the incoming flow equally or routes all of it through the right subtree. Assume without loss of optimality that all the flow follows the rightmost path. Then in a tree of depth d, the dth edge on this path has utilization $1/(\frac{2}{3})^d = \frac{3}{2}^d$.

In our protocols we can achieve utilization 1 as follows: Assign weight 1 to the last tree edge on the rightmost path from s. Now backtrack along the path. When at node v on the path, assume inductively that weights have already been assigned to the links in

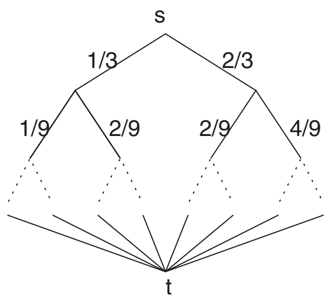

Fig. 2. Gadget TREE with edge capacities.

the right subtree of v. For each link in the left subtree of v assign a weight that is twice the weight of the "corresponding" edge in the right subtree. In particular, the edge from v to its left child is assigned a weight that is twice the weight of the edge to the right child. It can be shown that this weight setting will send one-third of the incoming flow at each node into the left subtree and two-thirds of the flow into the right subtree resulting in a utilization of 1 at all links of the tree.

4.2. *Path Weight*. In this section we consider the effect that our protocols have on the path weight of paths taken by packets. We show that, by construction of our algorithm, the path weight of packets routed by our algorithm is comparable with that of OSPF. That is, the additional or extra path weight is the expectation of the weight of a path taken by our packets minus the weight of the lightest path. Often OSPF weights are chosen to encode some natural significance, such as inverse capacity or expected link traversal time, since OSPF's main goal is to minimize weighted path length. In this case, comparing our path weights with those in OSPF is a reasonable measurement of worst-case performance. We first consider Class B protocols and then Class C protocols. We give thorough proofs only for Class C protocols; the proofs for the Class B protocols are similar.

THEOREM 1. *In a Class B protocol with splitting function e^{-kw}, on a network with maximum degree δ, the packets that reach their destination experience expected additional path weight $O(\ln(\delta)/k)$ per hop*.

PROOF. Similar to the proof of Theorem 3. □

THEOREM 2. *In a Class B protocol with β greater than the initial time-to-live τ, on a network with maximum degree δ, the expected total extra path weight experienced by packets that reach their destination is at most $2\ln(\delta)/k$*.

PROOF. Recall that the splitting function is $e^{k(\beta-\tau')^2 w}$. Using Theorem 1, the expected total extra latency is

$$\begin{aligned}\sum_{\tau'=0}^{\tau} \frac{\ln(\delta)}{k(\beta-\tau')^2} &\leq \frac{\ln(\delta)}{k} \sum_{\tau'=0}^{\tau} \frac{1}{(1+\tau-\tau')^2} \\ &\leq \frac{\ln(\delta)}{k} \sum_{n=1}^{\infty} \frac{1}{n^2} \\ &\leq \frac{2\ln(\delta)}{k}.\end{aligned}$$ □

Thus, in a Class B protocol, the path weight of our algorithm does not exceed by much the path weight of OSPF. We now consider whether packets are likely to arrive at their destination before their time-to-live expires.

In some networks, the edge weights can all be set to 1 or close to 1. In this case, extra path weight coincides with extra hop count, i.e., the number of routers a packet visits.

The above shows we can guarantee that packets do not experience much additional hop count. More generally, one can formally define a notion of distortion for a weight-setting that measures the similarity between path weight and hop count. By bounding excess path weight and distortion, we can limit the hop count.

We turn now to Class C protocols. We assume the hypothesis (and, thus, the conclusion) of the following:

FACT 1. *In a protocol of Class C all packets arrive at their destination in a number of hops at most their initial time-to-live, provided the initial time-to-live is at least as large as the shortest path and the packet is not dropped because of a full queue.*

We now consider the weight of paths taken by packets routed by a protocol in Class C. It will be convenient to consider the expected additive excess weight experienced by our packets; i.e., the expectation of the weight of a path taken by our packets minus the weight of the lightest path.

THEOREM 3. *In a network of maximum degree δ, using a Class C protocol with parameter k, the expected excess weight taken by packets with initial time-to-live τ is at most $\tau \log \delta / k$.*

PROOF. Let s be an arbitrary source and let t be an arbitrary destination. We first construct a tree containing all paths of length at most τ from s to t. The nodes of the tree are labeled with the vertices of the network. The root of this tree is s and all the leaves are labeled t. Each node of the tree represents a prefix of a path of length at most τ from s to t. An internal node v has a unique child labeled u iff the prefix represented by v can be extended with u to make a valid prefix of a path of length at most τ.

Now we transform the weights in the manner described below for ease of analysis. The transformation relies on the following easy fact about our protocol. The fraction of traffic (bound for destination t) that is forwarded from v to u depends only on the amount by which the lightest path from v to t through u exceeds the overall lightest path. Thus, it is only necessary that the weights faithfully reflect these excesses. So we transform the weights as follows.

Working top-down, at each node v in the tree, for each of the children $u_1, u_2, \ldots, u_p$ ($p \leq \delta$) of v we compute $w_1, w_2, \ldots, w_p$ where w_i is the weight of the shortest path from v to t through the child u_i. Let $w* = \min_i(w_i)$. We set the weight of the tree edge from v to u_i to be $w_i - w*$. Note that all tree weights are non-negative and that at least one of the edges from any node to its children will get a weight 0. By reordering, without loss of generality we can assume that the weight from each node to its leftmost child is 0.

The expected weight traversed in the first step is

$$E = \frac{\sum_{i=1}^{\delta} w_i e^{-kw_i}}{\sum_{i=1}^{\delta} e^{-kw_i}} = \frac{1}{k} \cdot \frac{\sum_{i=1}^{\delta} \hat{w}_i e^{-\hat{w}_i}}{\sum_{i=1}^{\delta} e^{-\hat{w}_i}}, \tag{1}$$

where $\hat{w}_i = kw_i$. Since w_1 is fixed to be 0, we regard the above expectation as a function only of the other w_i's and we wish to bound this from above, subject to non-negative weights. In other words, the feasible region is the positive orthant. We would like to show that the maximum value occurs in the interior of the positive orthant. To see this, note that if one of the w_i's ($i \geq 2$) is 0 then it contributes 0 to the numerator while contributing 1 to the denominator. Raising this w_i increases the numerator and decreases the denominator thereby raising the expected value. Similarly if one of the w_i's approaches ∞, then it contributes nothing to the numerator ($\lim_{w \to \infty} we^{-w} = 0$) and nothing to the denominator. Assume that this fraction E is of the form p/q, for p and q positive. Then we would increase this fraction if for example we set that $w_i \geq p/q$ but still finite, since $(p + w_i e^{-w_i})/(q + e^{-w_i}) > p/q$ for $w_i > p/q$.

Since this function E is differentiable everywhere in the region of interest, it suffices to find the points where the derivative is 0. (We will show that there is a unique maximum.)

For each $j > 1$, we have

$$\begin{aligned}\frac{\partial E}{\partial \hat{w}_j} &= \frac{(1-\hat{w}_j)e^{-\hat{w}_j}\sum_{i=1}^{\delta} e^{-\hat{w}_i} + e^{-\hat{w}_j}\sum_{i=1}^{\delta}\hat{w}_i e^{-\hat{w}_i}}{k\left(\sum_{i=1}^{\delta} e^{-\hat{w}_i}\right)^2} \\ &= \frac{e^{-\hat{w}_j}}{k\sum_{i=1}^{\delta} e^{-\hat{w}_i}}\left[1 - \hat{w}_j + \frac{\sum_{i=1}^{\delta}\hat{w}_i e^{-\hat{w}_i}}{\sum_{i=1}^{\delta} e^{-\hat{w}_i}}\right],\end{aligned}$$

and we want this to be 0 for each $j > 1$. This can only happen if the second factor is 0, i.e., equal for all j, whence $\hat{w}_2 = \cdots = \hat{w}_\delta$. In that case the expected weight is

$$\begin{aligned}E = \frac{\sum_{i=1}^{\delta}\hat{w}_i e^{-\hat{w}_i}}{k\sum_{i=1}^{\delta} e^{-\hat{w}_i}} &= \frac{1}{k}\frac{\sum_{i=2}^{\delta}\hat{w}_i e^{-\hat{w}_i}}{(1+\sum_{i=2}^{\delta} e^{-\hat{w}_i})} \\ &= \frac{(\delta-1)\hat{w}e^{-\hat{w}}}{k(1+(\delta-1)e^{-\hat{w}})} \\ &= \frac{1}{k}\cdot\frac{\hat{w}e^{-\hat{w}}}{\Delta + e^{-\hat{w}}},\end{aligned}$$

where $\hat{w} = \hat{w}_2 = \cdots = \hat{w}_\delta$, $\hat{w} > 0$, and $\Delta = 1/(\delta - 1)$. We have

$$\begin{aligned}\frac{dE}{d\hat{w}} &= \frac{1}{k}\frac{(1-\hat{w})e^{-\hat{w}}(\Delta + e^{-\hat{w}}) - \hat{w}e^{-\hat{w}}(-e^{-\hat{w}})}{(\Delta + e^{-\hat{w}})^2} \\ &= \frac{e^{-\hat{w}}}{k(\Delta + e^{-\hat{w}})^2}[(1-\hat{w})(\Delta + e^{-\hat{w}}) + \hat{w}e^{-\hat{w}}] \\ &= \frac{e^{-\hat{w}}}{k(\Delta + e^{-\hat{w}})^2}[\Delta - \Delta\hat{w} + e^{-\hat{w}}],\end{aligned}$$

and we want $\Delta - \Delta\hat{w} + e^{-\hat{w}} = 0$, i.e., $\hat{w} = 1 + e^{-\hat{w}}/\Delta$. It is clear that, for all Δ, the graphs of $\hat{w}$ and $1 + e^{-\hat{w}}/\Delta$ intersect exactly once. For $\delta - 1 = 1/\Delta \geq 8 \geq e^2$, put

$\hat{w} = \ln \Delta$, and we get

$$\begin{aligned}\Delta - \Delta\hat{w} + e^{-\hat{w}} &= \Delta(1 + \ln(\Delta)) + \Delta \\ &= \Delta(2 - \ln(1/\Delta)) \\ &\leq 0,\end{aligned}$$

so the maximum of E occurs at $\hat{w} \leq \ln(1/\Delta)$. At the maximum, $\Delta - \Delta\hat{w} + e^{-\hat{w}} = 0$, so $\Delta + e^{-\hat{w}} = \Delta\hat{w}$, and

$$\frac{\hat{w}e^{-\hat{w}}}{\Delta + e^{-\hat{w}}} = \frac{e^{-\hat{w}}}{\Delta} = \hat{w} - 1 \leq -\ln(\Delta) - 1 = \ln(\delta - 1) - 1,$$

so, for $\delta \geq 9 \geq e^2 + 1$, we have

$$E = \frac{1}{k} \cdot \frac{\hat{w}e^{-\hat{w}}}{\Delta + e^{-\hat{w}}} \leq \frac{1}{k}\left[\ln(\delta - 1) - 1\right] \leq \frac{\ln(\delta)}{k}.$$

(The truth is closer to $(1/k)\,[\ln(\delta - 1) - \ln\ln(\delta - 1) - 1]$, which is a lower bound for $\delta \geq 1 + e$.)

On the other hand, Table 1 gives a numerically computed bound on the expected excess weight per hop for small values of δ and $k = 1$. These values are smaller than $\ln(\delta)$. In general, letting E_k denote the value of E for the given value of k, $E_k \leq E_1/k$ by (2), so $E_k \leq \ln(\delta)/k$.

Thus the expected excess weight in traversing one layer of the tree is at most $\ln(\delta)/k$. By induction, the expected excess weight in traversing any of the height $t-1$ trees below $n_1, \ldots, n_\delta$ is at most $((t - 1)\ln(\delta))/k$, and, by linearity of expectation, the expected excess weight in traversing the entire tree of height t is at most $(t\ln(\delta))/k$ (with no hidden constant factors).

Finally, we note that a δ-ary tree in which, out of each node, one arc has weight 0 and the others have weight w (as optimized above) is an example network making the bound tight. □

In particular, one can use our algorithm on weights designed for OSPF. If we make the parameter k large, then our algorithm essentially simulates OSPF with those weights.

Table 1. Bounds on expected excess weight per hop, E, for the case $k = 1$, by degree, δ.

δ	max E	$\ln(\delta)$
2	0.27846	0.69314
3	0.46306	1.09861
4	0.60355	1.38629
5	0.71782	1.60944
6	0.81455	1.79176
7	0.89864	1.94591
8	0.97314	2.07944

As we make k smaller, our algorithm behaves less and less like OSPF in the sense that more packets take paths that are not optimal in weight (and potentially less congested than the weight-optimal paths). Still, for any k, we can guarantee that packets take paths of weight close to optimal, depending on the parameter k, the degree δ of the network, and the depth τ of the deepest path.

In our algorithm the initial time-to-live value τ of packets takes on an important role not present in OSPF. In both our algorithm and OSPF, a packet may travel for τ hops before being dropped, and the algorithm encourages the packet to take fewer (weighted) hops. In our algorithm the bias toward shorter paths is tunable (by setting k), and, in some cases, packets may take nearly the allowable τ hops. Furthermore, we cannot bound the total excess path weight taken by packets, only the excess weight per hop. Thus controlling the time-to-live bounds not only the expected hop count of packets, but also the expected weighted hop count of packets, which, intuitively, is a good idea if the weights have some natural meaning (like inverse capacity).

5. Dynamic Routing Criteria.

The previous section addresses the static quality of the routing protocol but a network is almost never static, demands change over time and the network itself changes (nodes become unavailable, links fail, etc.). An optimal weight setting for one demand matrix may be far from optimal for a different matrix. The same may be true for changes in network topology as well. In this section, we show two inputs to OSPF in which OSPF performs significantly worse than our proposed protocols under changing demands and network topology. These two examples are, by no means, an exhaustive characterization of the dynamic properties of our protocols; rather, they are simple illustrations of the fragility of OSPF.

5.1. *Changing Demands*. The gadget TRIANGLE in Figure 3 provides a favorable comparison of the performance of our protocol to OSPF in the face of changing demands. Initially the demand is 1 from a to c and 1 from b to c. For this demand OSPF can achieve utilization 1 by setting the ac and bc weights to 1 and the ab and ba weights to some positive value. Note that this is the optimal utilization since 2 units of flow have to flow across a cut of capacity 2 leading to c.

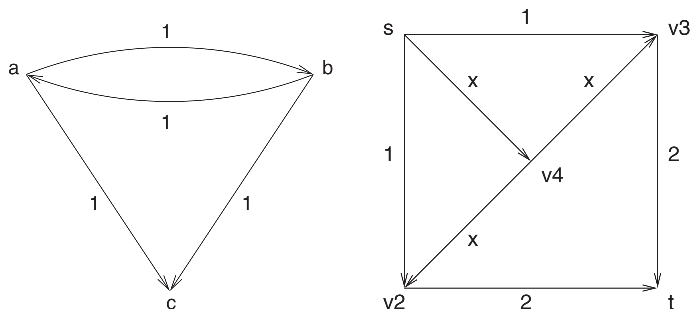

Fig. 3. Gadgets TRIANGLE (left) and SQUARE (right).

We can also achieve utilization 1, for example, by setting the weights of ac and bc to be 1 and the weights of ab and ba to be a small constant $\varepsilon > 0$. It is clear by symmetry that there is 1 unit of flow on the edges ac and bc. The edges ab and ba each carry flow which is upper bounded by the geometric series $\frac{1}{2} + \frac{1}{8} + \cdots$ which is $\frac{2}{3}$.

Now suppose the demand changes to being just 1 unit of flow from b to c. OSPF will still have utilization 1 since it cannot split this flow based on the previous weight setting. We, on the other hand, achieve a utilization arbitrarily close to $\frac{2}{3}$ by choosing ε arbitrarily small. The most utilized edge is bc where the flow in the limit is given by the geometric series above.

5.2. *Modified Network*. One characteristic of real networks is that machines and links enter and leave the network as servers or connections fail. We show that our protocols are more resilient to changes in the network, using the case where a machine becomes inaccessible. We model this by setting all incoming and outgoing edges of the "removed" vertex to have infinite weight. In the network SQUARE of Figure 3, our protocols and OSPF have the same cost, but if v_2 or v_3 were to leave the network, our protocol responds better than OSPF. The diagonal links in the network have capacity x for some $0 \le x \le 1$, the links from s to v_2 and v_3 have capacity 1, and the remaining edges have capacity 2. We compare the cost of a unit demand from s to t with and without node v_2.

The impact of the network change for OSPF depends on x, for some values of x make a three-way split from s advantageous. For $x \ge \frac{2}{3}$, OSPF uses the edge (s, v_4) with cost $1/(3x)$ before node v_2 is inaccessible and cost $1/(2x)$ after. In contrast, we can set our weights to give optimum cost for the network at both times, with cost $1/(2 + x)$ and $1/(1 + x)$, respectively. Our advantage over OSPF increases by a factor between $[1, \frac{16}{15}]$ for $x \in [\frac{2}{3}, 1]$ after node v_2 is unreachable.

6. Algorithmic Routing Criteria.

In addition to assessing how well a routing protocol performs, it is equally important to address *how* a protocol performs that job, how much computation is required to find the optimal weight setting (if such a setting can be found), how easy it is to implement the protocol, and how much storage and computational power the protocol uses. In this section we show that our protocols are competitive with OSPF with respect to these algorithmic criteria.

6.1. *Optimal Flow and Weight Setting*. We will show that we can find weights to achieve an optimal flow for single-destination demands. By contrast, this is hard for OSPF [5]. First, we formally define and solve the NETWORK FLOW problem as a multi-commodity flow problem, following [5].

DEFINITION 4 (NETWORK FLOW). For each arc a, there is a function, Φ_a, that is a convex, continuous, increasing function of the utilization u_a on arc a.

Input:

- a directed network with capacity labels on arcs, and
- a demand matrix.

Task:

- find a multi-commodity flow that minimizes $\sum_a \Phi_a(u_a)$ or $\max_a \Phi_a(u_a)$.

Consider also the class of dual problems—maximizing some norm of the demand while keeping $\sum_a \Phi_a(u_a)$ below a given budget.

The function Φ_a and the method to combine the Φ_a's (the sum or the maximum) are parameters to the definition. Often all the Φ_a are the same. If they differ, they may be multiples $c_a\Phi$ of a common Φ, where the coefficients c_a can be chosen to give more weight to some links over others. For example, 10% over-congestion on a major backbone link may be worse than 10% over-congestion on a dialup link.

LEMMA 5. *There is a polynomial-time algorithm to solve NETWORK FLOW with either sum or maximum combination method and with either of the following costs*:

- *The Φ_a's are piecewise linear and consist of at most a polynomial number of pieces.*
- *$\Phi_a(x)$ is a quadratic in x (provided $\Phi_a(x)$ is increasing for $x \geq 0$).*

PROOF. The sum of piecewise linear Φ_a's can be handled by linear programming (see [5]), and the maximum of piecewise linear Φ_a's is similar. The quadratic cases are similarly handled by convex quadratic programming [6]. $\square$

Now, we show that one can efficiently find an optimal flow from a given demand and topology using our protocol, i.e., efficiently set our weights to produce optimal flow. Note that an optimal single-destination flow is acyclic.

THEOREM 6. *Given an acyclic flow F for a single-destination demand, and a finite value for the parameter k, there is a polynomial-time algorithm to find a set W of arc weights such that, with weights W, our Class C protocol realizes flow F.*

PROOF. Note that, in the case of a single-destination demand, the multi-commodity flow degenerates into a single commodity. Also, we can put infinite weights on arcs not carrying flow, so, without loss of generality, we assume there are no such arcs.

Assume the nodes, $\{1, \ldots, n\}$, are labeled so that $i < j$ iff i precedes j in topological sort order. Consider the nodes in reverse order; at node i, we will set the weights on arcs leaving i. The sink, n, has no outgoing arcs, so we start with $n - 1$, which has only parallel arcs to n. Choose weights $w_1, w_2, \ldots$ so that the ratio $e^{-kw_1} : e^{-kw_2} : \cdots$ matches the ratio specified by the flow, F. In general, when considering node i, all the weights on all the paths from i to n have been set except the weights on arcs leaving i. Set these weights so that the weights $w_1, w_2, \ldots$ of lightest paths from i to n via each of its out-neighbors make the ratio $e^{-kw_1} : e^{-kw_2} : \cdots$ equal to what is specified by F. $\square$

We emphasize that, for these types of networks and demands, not only can we find the optimal weight settings for our protocol, but we can also achieve *the* optimal routing (which is optimal over *all* protocols).

By contrast,

THEOREM 7. *Given a network and a demand with one destination and n sources*, *for a typical cost function*, *it is NP-hard to find weights under which OSPF achieves cost within the factor* $(1 + 1/(2n))$ *of the cost of an optimal routing*.

There is a class of networks and demands for which we can find the optimal weights for our protocol, *and thereby achieve the optimal flow*, *but for which it is NP-hard to find the weights under which OSPF achieves cost within a constant factor of optimal*.

PROOF. The proof of the first statement closely follows [5]. The algorithm in the second statement closely follows Theorem 6 and the NP-hardness of the second statement is given in [5]. □

The NP-hardness proof in [5] relies on the fact that OSPF splits flow evenly, if at all; this is used to enforce that a variable of a 3SAT instance is either true or false but not both. On the other hand, for any finite value of the parameter k, our algorithm can split flow in any desired ratio. Thus the ability of our protocol to split flow unevenly improves not only its congestion, as might be expected, but also the *ease of setting its weights*, compared with OSPF.

6.2. *Efficiency of Proposed Protocols*. In this subsection we show that Class C protocols are relatively easy to implement and that not much more storage or computational power is required of a router than OSPF currently requires. Routers currently check the time-to-live field of each packet and discard those that have expired and they periodically perform a lightest path computation to populate their routing tables.

Our Class C protocol requires computation of lightest paths of bounded depth. We now consider the complexity of this problem. We can assume that each router knows the weights on all of the edges, by using a broadcast mechanism similar to what OSPF currently uses.

THEOREM 8. *There is an* $O(\tau|E|)$ *algorithm*, A, *such that*, *given a graph* $G = (V, E)$ *in which each arc has a non-negative weight*, *given a source vertex* s, *and given a maximum path depth* τ, A *finds a tree of the lightest paths of depth at most* τ' *from* s *to each other vertex*, t, *for* $\tau' \leq \tau$.

PROOF. Use a table T, indexed by path depth and vertex. Each entry $T(\tau', v)$ contains the weight of the shortest path from s to v at depth τ' and the vertex previous to v on that path.

```
// initialize at $\tau' = 1$
for each $v \in V$
    $T(1, v).prev := s$
    if $(s, v) \in E$
       then $T(1, v).wt :=$ weight$(s, v)$
       else $T(1, v).wt := \infty$
```

```
// find the shortest path at each incremental depth
for $\tau' = 2$ to $\tau$
    for each $v \in V$
        find $u$ such that $T(\tau' - 1, u).wt$ + weight$(u, v)$ is minimum over $(u, v) \in E$
        if $T(\tau' - 1, v).wt > T(\tau' - 1, u).wt$ + weight$(u, v)$
          then $T(\tau', v).wt := T(\tau' - 1, u).wt$ + weight$(u, v)$
               $T(\tau', v).prev := u$
          else $T(\tau', v) := T(\tau' - 1, v)$
```

To regenerate the path from s to v at depth τ', trace back the $prev$ fields starting at $T(\tau', v)$.

Storing the adjacency list of incoming vertices at each v allows the above algorithm to run in $O(\tau|E|)$ time. This should be compared with the runtime of $O(|V| \log |V| + |E|)$ currently required by OSPF to run Dijkstra's shortest path algorithm using a Fibonacci heap. □

Next, we consider the complexity of using the shallow-light information to route packets. The output produced by the above algorithm has size $\Theta(\tau|V|)$. To run our Class C protocol, a router r needs to know, for each out-neighbor n and for the arbitrary time-to-live τ' of a packet to be routed, the lightest path though n, to destination t, of depth at most τ'. If the router itself stores all this information, it needs space $|V|\tau|\Gamma(r)|$ space, where τ is the maximum possible time-to-live and $|\Gamma(r)|$ is the out-degree of r. By comparison, OSPF theoretically requires space $|V| \cdot |\Gamma(r)|$ at each router, since OSPF can potentially split flow among all $|\Gamma(r)|$ neighbors.

In practice, in OSPF a router assumes a fixed limit on the number of outgoing edges considered, say six, ignoring additional paths. This hard limit makes optimization difficult, so it is assumed that there are never more than six ties and the router does not need to limit the number of paths.

For us, the factor τ in the space complexity can plausibly be about $\log_\delta |V|$ in a well-connected network. Furthermore, we can reduce this factor to, say, $\log \tau$, by only storing for some α and all $i \leq \log_\alpha(\tau)$, values for $\tau' = \lceil \alpha^i \rceil$. The router, with value τ', rounds up to the next power of α (say, τ''), and uses the weight of the lightest path of depth τ'' instead of the lightest path of depth τ'. This may cause packets to take a factor of α additional hops and thereby blow up the excess path weight by a factor of α. Such approximations are available to the basic Class C protocol, making it competitive with OSPF.

Note that our Class B protocols may require that packets store the parameters k and/or β. We anticipate that k will be a global parameter that the packets need not store. The parameter β may depend only on the source and destination of the packet, and so, possibly, can be deduced from that information (which is already stored in the packet).

Thus, in terms of computation of routing tables from weights, performance of routing from routing tables, and packet headers, our protocols require little more resources than what is currently used by OSPF.

7. Conclusions. This paper presents some preliminary evidence that the classes of new protocols we have described are interesting and worthy of further investigation.

Class A protocols split flow in a "polynomial" fashion and are proposed primarily for illustrative purposes. **Class B** protocols split the flow in an exponential fashion. **Class C** protocols also split flow in an exponential fashion but restrict the set of paths to those that the packet can traverse before it expires. These protocols use a global tunable parameter k which controls the bias towards shortest paths, and, as such, is a natural extension to OSPF.

These protocols are competitive alternatives to OSPF with respect to static, dynamic, and algorithmic routing criteria. They generate significantly less congestion than OSPF (for specific inputs) and the weight of the paths the packets traverse under these protocols is comparable with that of OSPF. In other words, without significant additional path weight, our protocols yield significantly less network congestion than OSPF. We also provide several dynamic examples in which OSPF performs poorly compared with our protocols. We show that for a class of inputs including single-sink demands, we can find optimal weight settings for our protocols. For this class of inputs, the corresponding problem for OSPF is NP-hard. Finally, we demonstrate that the computational resources required of our protocols is not drastically different from OSPF.

While these protocols (**Class C** in particular) suggest attractive alternatives to OSPF, much additional theoretical and empirical work needs to be done in order to understand these protocols fully and make a strong case for their adoption in networks. Some important theoretical directions are listed below:

- How hard is it to optimize weights for these protocols for arbitrary network topologies and demand matrices?
- How does this optimal setting compare with the multi-commodity flow solution?
- Under what conditions can this optimization be done in a distributed manner?
- Currently we have specific gadget families on which our protocols cause less congestion than OSPF. We also know that there are network topologies and demands on which our protocols achieve the best congestion that can be achieved by any protocol. Can we show analytical separation bounds between the congestion achieved by us and the congestion achieved by OSPF for a large and interesting class of inputs?
- At present we have a limited set of examples illustrating the behavior of our protocol when demands and topology change. This is an area that requires additional study.
- There are deterministic and stochastic models of time-varying traffic patterns. Can we analyze the behavior of our protocols for congestion and latency under traffic drawn from these models? Part of the problem will be to account for the cost of changing link weights and optimize the points in time at which weights are changed.
- What kinds of data structures and algorithms should be used by routers to implement the bare bones versions of our protocols? What happens when we add the desideratum that semantically related streams of packets should not be split?

There are also many empirical questions. How do our protocols coexist with OSPF? If some routers use our protocols and some routers use OSPF, what happens to the overall network performance? To measure performance, we need to perform simulations of these protocols on realistic traffic matrices and topologies, followed by actual tests on real networks.

References

[1] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosen. Adaptive packet routing for bursty adversarial traffic. In *Proceedings of the* 30*th ACM Symposium on Theory of Computing* (*STOC*), pages 359–368, New York, 1998.

[2] S. Bak and J. A. Cobb. Randomized distance-vector routing protocol. In *Proceedings of the ACM Symposium on Applied Computing* (*SAC*), pages 78–84, San Antonio, TX, 1999.

[3] Cisco. Configuring OSPF. `http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113ed_cr/np1_c/1cospf.htm`, 1997.

[4] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proceedings of the* 19*th INFOCOM*, volume 2, pages 519–528, Los Alamitos, CA, 2000.

[5] B. Fortz and M. Thorup. Increasing internet capacity using local search. *Computational Optimization and Applications*, **29**(1) (2004), 13–48.

[6] S. Kapoor and P. Vaidya. Fast algorithms for convex quadratic programming and multicommodity flows. In *Proceedings of the* 18*th ACM Symposium on Theory of Computing* (*STOC*), pages 147–159, New York, 1986.

[7] J. Moy. OSPF version 2. `http://search.ietf.org/rfc/rfc2328.txt`, April 1998. IETF RFC 2328.

[8] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the* 13*th ACM Symposium on Theory of Computing* (*STOC*), pages 263–277, Milwaukee, WI, 1981.

[9] C. Villamizar. OSPF Optimized Multipath. `http://fictitious.org/ospf-omp/ospf-omp.html`.

[10] Z. Wang and J. Crowcroft. Shortest Path First with Emergency Exits. *Computer Communication Review*, **20**(4) (1990), 166–176.