Lying Your Way to Better Traffic Engineering

Marco Chiesa
Universitè catholique de Louvain
marco.chiesa@uclouvain.be

Gábor Rétvári
MTA-BME Information Systems Research Group
retvari@tmit.bme.hu

Michael Schapira
Hebrew University of Jerusalem
schapiram@cs.huji.ac.il

ABSTRACT

To optimize the flow of traffic in IP networks, operators do traffic engineering (TE), i.e., tune routing-protocol parameters in response to traffic demands. TE in IP networks typically involves configuring static link weights and splitting traffic between the resulting shortest-paths via the Equal-Cost-MultiPath (ECMP) mechanism. Unfortunately, ECMP is a notoriously cumbersome and indirect means for optimizing traffic flow, often leading to poor network performance. Also, obtaining accurate knowledge of traffic demands as the input to TE is elusive, and traffic conditions can be highly variable, further complicating TE. We leverage recently proposed schemes for increasing ECMP's expressiveness via carefully disseminated bogus information ("lies") to design COYOTE, a readily deployable TE scheme for robust and efficient network utilization. COYOTE leverages new algorithmic ideas to configure (static) traffic splitting ratios that are optimized with respect to all (even adversarially chosen) traffic scenarios within the operator's "uncertainty bounds". Our experimental analyses show that COYOTE significantly outperforms today's prevalent TE schemes in a manner that is robust to traffic uncertainty and variation. We discuss experiments with a prototype implementation of COYOTE.

1. INTRODUCTION

To adapt the routing of traffic to the demands network operators do traffic engineering (TE), i.e., tune routing-protocol parameters so as to influence how traffic flows in their networks [1–3]. Today's prevalent scheme for TE within an organizational IP network is based on configuring static link-weights into shortest-path protocols such as OSPF [4] and splitting traffic between the resulting shortest-paths via ECMP [5]. Traditional TE with ECMP significantly constrains both route-computation and traffic splitting between multiple paths in two crucial ways: (1) traffic from a source to a destination in the network can only flow along the shortest paths between them (for the given configuration of link weights), and (2) traffic splitting between multiple paths (if multiple shortest paths exist) can only be done in very specific manners (see Section 2.1 for an illustration).

ECMP's lack of expressiveness makes traffic engineering with ECMP a notoriously hard task that often results in poor performance. Indeed, not only does ECMP's inflexibility imply that traffic flow might be arbitrarily far from the global optimum [6], but even choosing "good" link weights for TE with ECMP is infeasible in general [7]. Beyond ECMP's deficiencies, today's dominant TE schemes also suffer from other predicaments, e.g., obtaining an accurate view of traffic demands so as to optimize TE is elusive, as most networks lack the appropriate measurement infrastructure. Also, traffic can be highly variable and routing configurations that are good with respect to one traffic scenario can be bad with respect to another. We thus seek a TE scheme that is backwards compatible with legacy routing infrastructure (i.e., OSPF and ECMP), yet *robustly* achieves high performance even under uncertain or variable traffic conditions.

Introducing COYOTE: optimized, OSPF/ECMP-compatible TE. We leverage recently introduced approaches for enriching ECMP's expressiveness without changing router hardware/software to design COYOTE (COmpatible Yet Optimized TE). Recent studies show that by injecting "lies" into OSPF-ECMP (specifically, information about fake links and nodes), OSPF and ECMP can support much richer traffic flow configurations [8, 9]. We exploit these developments to explore how OSPF-ECMP routing can be extended to achieve consistently high performance even under great uncertainty about the traffic conditions and high variability of traffic. To accomplish this, COYOTE relies on new algorithmic ideas to configure (static) traffic splitting ratios at routers/switches that are optimized with respect to *all* (even *adversarially* chosen) traffic scenarios within operator-specified "uncertainty bounds".

We view COYOTE as an important additional step in the recent exploration of how SDN-like functionality can be accomplished without changing today's networking infrastructure (see [8,9]). Our experimentation with COYOTE on real network topologies shows that COYOTE indeed consistently and robustly achieves good performance even with very lim-

CoNEXT '16, December 12 - 15, 2016, Irvine, CA, USA

 ISBN 978-1-4503-4292-6/16/12...$15.00
DOI: http://dx.doi.org/10.1145/2999572.2999585

ited (in fact, sometimes even no) knowledge about the traffic demands and, in particular, exhibits significantly better performance than (optimized) traditional TE with ECMP. Our experiments with a prototype implementation of COYOTE also demonstrate its performance benefits. We briefly discuss below the algorithmic challenges facing the design of COYOTE and how these are tackled.

New algorithmic framework: destination-based oblivious routing. A rich body of literature in algorithmic theory investigates "(traffic-demands-)oblivious routing" [10–12], i.e., how to compute provably good routing configurations with limited (possibly even no) knowledge of the traffic demands. Past studies [11, 13] show that, even though lacking accurate information about the traffic demands, demands-oblivious routing algorithms yield remarkably close-to-optimal performance on real-world networks. Unfortunately, the above-mentioned algorithms involve forwarding packets based on both source and destination and are so inherently incompatible with destination-based routing via OSPF-ECMP. In addition, realizing these schemes in practice entails either excessive use of (e.g., MPLS) tunneling/tagging in traditional IP networks [11, 14], or the ubiquitous deployment of per-flow routing software-defined networking infrastructure [15].

Our design of COYOTE relies on a novel algorithmic framework for demands-oblivious IP routing. We initiate the study of optimizing oblivious routing under the restriction that forwarding is destination-based. In light of the recent progress on enhancing OSPF-ECMP's expressiveness, we view the algorithmic investigation of destination-based oblivious routing as an important and timely research agenda. We take the first steps in this direction. Our first result establishes that, in contrast to unconstrained oblivious routing, computing the optimal destination-based oblivious routing configuration is computationally intractable. We show how, via the decomposition of this problem into sub-problems that are easier to address with todays mathematical toolkit, and by leveraging prior research, good routing configurations can be generated. We regard our algorithmic results along these lines as a first, yet promising, step en route to better TE in IP networks, and leave the reader with many interesting open questions in Section 6.

2. COYOTE: OVERVIEW AND DESIGN

We next motivate COYOTE through a simple example, present the algorithmic challenges facing COYOTE's design, and explain how these are tackled.

2.1 Motivating Example

Consider the toy example in Fig. 1a. Two network users, s_1 and s_2, wish to send traffic to target t. Suppose that each user is expected to send between 0 and 2 units of flow and each link is of capacity 1. Suppose also that the network operator is oblivious to the actual traffic demands or, alternatively, that traffic is variable and user demands might drastically change over time. The operator aims to provide robustly good network performance, and thus has an ambitious goal: configuring OSPF-ECMP routing parameters

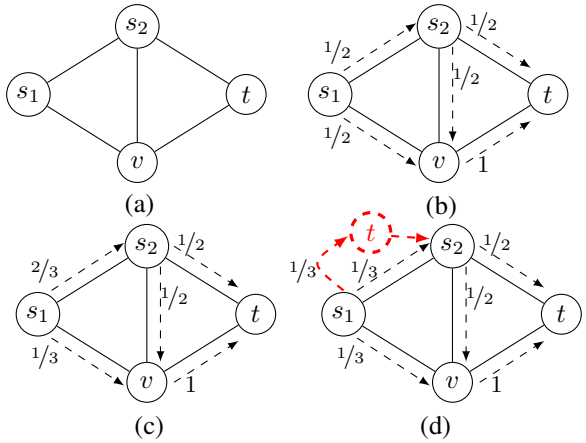


Figure 1: A sample network: (a) topology with unit capacity links; (b) per-destination ECMP routing (oblivious performance ratio $3/2$); (c) COYOTE (oblivious performance ratio $4/3$); and (d) COYOTE implementation with a fake node inserted at s_1 for realizing the required splitting ratio.

so as to minimize link over-subscription across *all* possible combinations of traffic demands within the above-specified uncertainty bounds.

Consider first the traditional practice of splitting traffic equally amongst the next-hops on shortest-paths to the destination (i.e., traditional TE with ECMP, see Fig. 1b), where the shortest path DAG towards t is depicted by dashed arrows labelled with the resulting flow splitting ratios. Observe that if the actual traffic demands are 2 and 0 for s_1 and s_2, respectively, routing as in Fig. 1b would result in link (over-)utilization that is $3/2$ higher than that of the optimal routing of these specific demands (which can send all traffic without exceeding any link capacity). Specifically, routing as in Fig. 1b would result in $3/2$ units of traffic traversing link (v, t), whereas the total flow could be optimally routed without at all exceeding the link capacities by equally splitting it between paths $(s_1\ s_2\ t)$ and $(s_1\ v\ t)$. One can actually show that this is, in fact, the *best* guarantee achievable for this network via traditional TE with ECMP, i.e., for *any* choice of link weights, equal splitting of traffic between shortest paths would result in link utilization that is $3/2$ higher than optimal for *some* possible traffic scenario. Can we do better?

We show that this is indeed possible if more flexible traffic splitting than that of traditional TE with ECMP is possible. One can prove that for *any* traffic demands of the users, per-destination routing as in Fig. 1c results in a maximum link utilization at most $4/3$ times that of the optimal routing [1]. We explain later how COYOTE realizes such uneven per-destination load balancing without any modification to legacy OSPF-ECMP.

[1]In fact, even the routing configuration in Fig. 1c is not optimal in this respect. Indeed, COYOTE's optimization techniques, discussed in Section 2.3, yield configurations with better guarantees (see [16]).

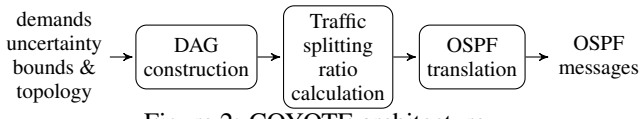


Figure 2: COYOTE architecture.

2.2 Challenges

Realizing hop-by-hop destination-based traffic engineering boils down to computing, for each destination, (1) a Directed Acyclic Graph (DAG) rooted in that destination (so as to guarantee loop-free packet forwarding) along which traffic is to be forwarded, and (2) the splitting of traffic within each DAG (c.f., Fig. 1c). Thus, the optimization problem that underlies COYOTE is the following: compute per-destination DAGs and traffic splitting ratios so as to minimize the worst-case link utilization (also referred to as congestion in TE literature [1, 2]) across *all* possible traffic scenarios (within the operator-specified uncertainty bounds). Our first (and negative) result shows that this is, in fact, intractable. The proof is omitted due to space constraints and is available in [16].

THEOREM 1. *Given a capacitated network graph $G = (V, E)$ and a set $\mathcal{D}$ of possible traffic demands between nodes, computing the optimal combination of DAGs and traffic splitting ratios with respect to minimizing the worst-case link utilization is NP-hard.*

Hence, efficiently computing the *optimal* selection of DAGs *and* in-DAG traffic splitting ratios is beyond reach. We next describe how COYOTE's design addresses this challenge. COYOTE's flow-computation decomposes the task of computing destination-based oblivious routing configurations into two sub-problems, and tackles each independently. First, COYOTE applies a simple heuristic to compute destination-oriented DAGs. Then, COYOTE optimizes in-DAG traffic splitting ratios through a combination of optimization techniques, including iterative geometric programming. We show in Section 3 that COYOTE's routing algorithm empirically exhibits good network performance.

2.3 COYOTE Design

Figure 2 presents an overview of the COYOTE architecture. COYOTE gets as input the (capacitated) network topology and the so-called "uncertainty bounds", i.e., for every two nodes (routers) in the network, i and j, a real-valued interval $[d_{ij}^{min}, d_{ij}^{max}]$, capturing the operator's uncertainty about the traffic demand from i to j or, alternatively, the potential variability of traffic. COYOTE then uses this information first to compute a forwarding DAG rooted in each destination node, and then to optimize traffic splitting ratios within each DAG. Lastly, the outcome of this computation is converted into OSPF configuration by injecting "lies" into routers. We next elaborate on each of these components.

Computing DAGs. Theorem 1 implies that computing DAGs so as to support optimal routing (through the appropriate in-DAG traffic splitting) is intractable. In COYOTE, DAGs rooted in different destinations are not coupled in any way, allowing network operators to specify any set of DAGs. We show in Section 3, however, that COYOTE significantly outperforms TE with ECMP even when the underlying DAGs are selected with the following simple heuristic: (1) compute, for each destination, the shortest-path DAG rooted in that destination when the link weights are the inverse capacities, and then (2) augment each DAG with additional links by orienting each link that does not appear in the shortest-path DAG towards the incident node that is closer to the destination, breaking ties lexicographically (suppose that the nodes are numbered). Revisiting our running example in Fig.1a, observe that while the shortest-path DAG rooted at t does *not* contain link (s_2, v) if all links have the same weight, the augmented forwarding DAGs will also utilize this link (in some direction).

Computing traffic splitting ratios for each DAG. The second fundamental building block of COYOTE is an algorithm that receives as input a set of per-destination DAGs and optimizes traffic splitting within these DAGs, with the objective of minimizing the worst-case congestion (link utilization) over a given set of possible traffic demands. Whether this problem can be solved optimally in a computationally-efficient manner remains an open question (see Section 6. This seems impossible within the familiar mathematical toolset of TE, namely, integer and linear programming. We found that a different approach is, however, feasible: casting the optimization problem as a geometric program (in fact, a mixed linear-geometric program [17]).

Stating COYOTE's traffic splitting optimization as a geometric program is not straightforward and involves careful application of various techniques (convex programming, monomial approximations, LP duality). We provide an intuitive exposition of some of these ideas below using the running example in Fig. 1 (more details in in [16]).

Again, s_1 and s_2 send traffic to t, let the DAG for t be as in Fig. 1c, and suppose that the capacity on links (s_1, s_2), (s_1, v), and (s_2, v) is infinite (that is, arbitrarily large) and on (s_2, t) and (v, t) is 1. We are given a set of possible traffic demands $\{d_{s_1}, d_{s_2}\}$ for the two users and our goal is to find the traffic splitting ratios ϕ so that the worst-case link utilization across all demands is minimized. A simplified mathematical program for this problem would take the following form (see explanations below):

$$\min \alpha \tag{1}$$

$$\frac{d_{s_1}\phi(s_1,s_2)\phi(s_2,t) + d_{s_2}\phi(s_2,t)}{\text{capacity}(s_2,t)} \leq \alpha \quad \forall d_{s_1}, d_{s_2} \tag{2}$$

$$\frac{d_{s_1}(1-\phi(s_1,s_2)\phi(s_2,t)) + d_{s_2}\,(1-\phi(s_2,t))}{\text{capacity}(v,t)} \leq \alpha \quad \forall d_{s_1}, d_{s_2} \tag{3}$$

The objective is to minimize α, which represents worst-case link utilization, i.e., the load (flow divided by capacity) on the most utilized link across all the admissible traffic demands. Each variable $\phi(x, y)$ denotes the fraction of the incoming flow at vertex x that is routed on link (x, y). Constraints (2) and (3) force α to be at least the value of the link utilization of links (s_2, t) and (s_1, s_2), respectively. For

the sake of simplicity, we do not show the link utilization constraints for the remaining links. Now, consider constraint (2) for link (s_2, t). Observe that from user s_1 the fraction of traffic sent through (s_2, t) equals the fraction of s_1's traffic through (s_1, s_2) (i.e., $\phi(s_1, s_2)$) times the fraction sent through (s_2, t) by s_2 (i.e., $\phi(s_2, t)$). The fraction of s_2's traffic through (s_2, t) is simply $\phi(s_2, t)$. Accordingly the total flow on (s_2, t) equals $d_{s_1} \cdot \phi(s_1, s_2) \cdot \phi(s_2, t) + d_{s_2} \cdot \phi(s_2, t)$. Hence, the link utilization of (s_2, t) is this expression divided by the capacity of (s_2, t), and the corresponding constraint (2) requires that this utilization be at most α for *all* demands d_{s_1}, d_{s_2}. Constraint (3) states the same for link (v, t), where the fraction of traffic sent by s_1 (s_2) to t through (v, t) is equal to 1 minus the fraction of flow sent from s_1 (s_2) to t through (s_2, t).

Two difficulties with these constraints immediately arise: one is that it is *universally quantified* over an entire set of traffic demands, possibly of infinite cardinality, and the other is that it involves a *product of unknowns*, namely, $\phi(s_1, s_2) \cdot \phi(s_2, t)$, and such products do not fit into the framework of standard linear and integer programming. For a discrete demand set we can handle the first problem by stating (2) and (3) for each individual demand. Otherwise (if the set of demands is of infinite size) the elegant dualization technique from [11] can be used. To handle the second issue, however, we need a small trick from geometric programming [17]. Let $d_{s_1} = 1$ and $d_{s_2} = 1$ and consider constraint (2):

$$\phi(s_2, t) + \phi(s_1, s_2) \cdot \phi(s_2, t) \leq \alpha \ .$$

Now, substitute for new variables, $\widetilde{\phi}(s_1, s_2) = \log \phi(s_1, s_2)$ and $\widetilde{\phi}(s_2, t) = \log \phi(s_2, t)$, and take the logarithm of both sides:

$$\log \left(e^{\widetilde{\phi}(s_2,t)} + e^{\widetilde{\phi}(s_1,s_2)+\widetilde{\phi}(s_2,t)} \right) \leq \log \alpha \ .$$

This constraint is now a logarithm of a sum of exponentials of linear functions and so is convex, opening the door to using standard convex programming. Our implementation uses a convex program based on the above ideas and others delicate techniques to compute the traffic splitting ratios. The reader is referred to our technical report for a detailed explanation [16].

Translation to OSPF-ECMP configuration. As explained above, using OSPF and ECMP for TE constrains the flow of traffic in two significant ways: (1) traffic only flows on shortest-paths (induced from operator specified link weights), and (2) traffic is split equally between multiple next-hops on shortest-paths to a destination. Recent studies show how OSPF-ECMP's expressiveness can be significantly enhanced by effectively deceiving routers. Specifically, Fibbing [8, 9] shows how *any* set of per-destination forwarding DAGs can be realized by introducing fake nodes and virtual links into an underlying link-state routing protocol, thus overcoming the first limitation of ECMP. [18] shows how ECMP's equal load balancing can be extended to much more nuanced traffic splitting by setting up virtual links alongside existing physical ones, thus relaxing the second of these limitations.

We revisit our running example to show how COYOTE exploits these techniques. Consider Fig. 1d. Inserting a fake advertisement at s_1 into the OSPF link-state database can "deceive" s_1 into believing that, besides its available shortest paths via s_2 and v , destination t is also available via a third, "virtual" forwarding path. The forwarding adjacency in the fake advertisement is mapped to s_2, so that out of s_1's three next-hops to t node s_2 will appear *twice* while v only appears once. Consequently, the traffic is *effectively* split between s_2 and v in a ratio 2/3 to 1/3. Beyond changing how traffic is split *within* a given shortest-path DAG, as illustrated in Fig. 1d, fake nodes/links can be injected into OSPF so to as change the forwarding DAGs themselves at the per-IP-destination-prefix granularity, as shown in [9]. COYOTE leverages the techniques in [9] and in [18] to carefully craft "lies" so as to generate the desired per-destination forwarding DAGs and approximate the optimal traffic splitting ratios with ECMP. Section 3 that highly optimized TE is achievable even with the introduction of few virtual nodes and links.

3. EVALUATION

We experimentally evaluate COYOTE in order to quantify its performance benefits and its robustness to traffic uncertainty and variation. Importantly, our focus is solely on destination-based TE schemes (i.e., TE schemes that can be realized via today's IP routing). We show below that COYOTE provides significantly better performance than ECMP even when *completely* oblivious to the traffic demands. Also, COYOTE's increased path diversity does not come at the cost of long paths: the paths computed by COYOTE are on average only a factor of 1.1 longer than ECMP's. We also discuss experiments with a prototype implementation of COYOTE.

While the reader might think that COYOTE's performance benefits over traditional TE with ECMP are merely a byproduct of its greater flexibility in selecting DAGs and in traffic splitting, our results show that this intuition is, in fact, false. Specifically, we show that, similarly to unconstrained (i.e., source and destination based) oblivious routing [11], even the *optimal* routing with respect to *estimated* traffic demands, which can unevenly split traffic, fares much worse than COYOTE if the *actual* traffic demands are not very "close" to the estimated demands. Hence, COYOTE's good performance should be attributed not only to its expressiveness but also, in large part, to its built-in algorithms for optimizing performance in the presence of uncertainty, as discussed in Section 2.3.

3.1 Simulation Framework

We use the set of 16 backbone Internet topologies from the Internet Topology Zoo (ITZ) archive [19] to assess the performance of COYOTE and ECMP. When available, we use the link capacities provided by ITZ. Otherwise, we set the link capacities to be inversely-proportional to the ITZ-provided ECMP weights (in accordance with the Cisco-recommended default OSPF link configuration [20]). When neither ECMP link weights nor capacities are available we use

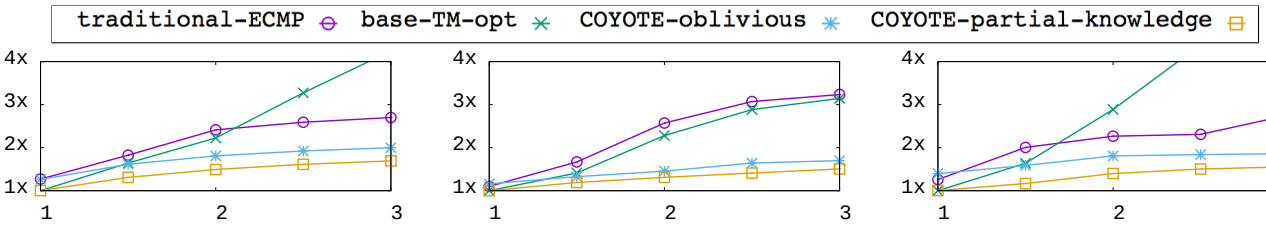


Figure 3: Geant, gravity model.

Figure 4: Digex, gravity model.

Figure 5: AS 1755, bimodal model

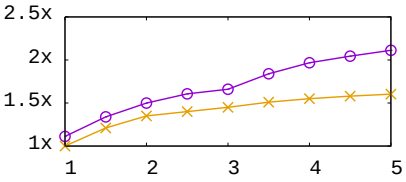


Figure 6: Abilene, optimized-ECMP.

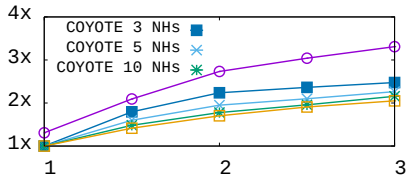


Figure 7: Approximation.

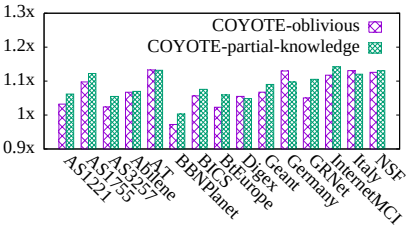


Figure 8: Average stretch.

unit capacities and link weights. We evaluate COYOTE against ECMP using two simple DAG-construction heuristics: (1) constructing augmented shortest-path DAGs as explained in Section 2 with the link weights provided in the ITZ archive, and (2) optimizing the link weights using the local greedy algorithm of Altin et al [12]. In both cases, we augment the shortest path DAGs as explained in Section 2.

To compute COYOTE's in-DAG traffic splitting ratios (see Section 2), we use AMPL [21] as the problem formulation language and MOSEK [22], a non-linear convex optimization solver. The running time with our current single-threaded proof-of-concept implementation ranges from few minutes (for small networks) to few days (for large networks).

While we are currently working on improving the running times, we would like to point out that the computation of the in-DAG traffic splitting ratios needs only be performed once or on a daily/weekly-base, as routing in COYOTE is not dynamically adjusted, and that routing configurations for failure scenarios (e.g., every single link/node failure) can be precomputed.

We measure performance in terms of the worst-case link utilization (referred to as "congestion" in TE literature [1, 2]), i.e., the performance of (multicommodity) flow of traffic f is $\max_l \frac{f(l)}{c_l}$, where $f(l)$ is the flow traversing link l and c_l is link l's capacity.

3.2 Network Performance

We compare COYOTE to ECMP for both DAG-construction heuristics described above and for two types of base traffic demands: (1) *gravity* [23], where the amount of flow sent from router i to router j is proportional to the product of i's total outgoing capacity and j's total outgoing capacity, and (2) *bimodal* [24], where a small fraction of all pairs of routers exchange large quantities of traffic, and the other pairs send small flows.

We first present our results with respect to the ITZ [19] link weights and an ideal version of COYOTE capable of arbitrarily fine-grained traffic splitting. We then show that a close approximation of the optimal splitting ratios can be obtained with the introduction of a limited number of additional virtual links. Fig. 3 and Fig. 4 describe our results for two networks (Geant and Digex, respectively), the gravity model, and augmented shortest path DAGs based on the ITZ link weights. The x-axis represents the "uncertainty margin": let $d_{i,j}$ be the amount of flow from router i to router j in the base traffic demands (namely, gravity), a margin of uncertainty of x means that the actual flow from i to j can be any value between $\frac{d_{i,j}}{x}$ and $x \cdot d_{i,j}$. We increase the uncertainty margin in increments of 0.5 from 1 (no uncertainty whatsoever) to 3 (fairly high uncertainty). The y-axis specifies how far the computed solution is from the *demands-aware* optimum within the same DAGs.

We plot four lines, corresponding to the performance of four different protocols: (1) traditional TE with ECMP, (2) the optimal demands-aware routing for the base gravity model (with no uncertainty), which can be obtained with linear programming techniques [25], (3) COYOTE (oblivious) with traffic splitting optimized with respect to *all* possible traffic demands (i.e., assuming *nothing* about the demands), (4) COYOTE (partial-knowledge) optimized with respect to the traffic demands within the uncertainty margin. Observe that both variants of COYOTE provide significantly better performance than TE with ECMP and, more surprisingly, both COYOTE and (sometimes) ECMP outperform the optimal base routing, whose performance quickly degrades even with little demands uncertainty. Our results thus show that COYOTE's built-in robustness to traffic uncertainty, in the form of optimization under specified uncertainty margins, indeed leads to superior performance in the face of inaccurate knowledge about the traffic demands or, alternatively, variable traffic conditions.

We observe the same trends when the base traffic demands are sampled from the bimodal model, as shown in Fig. 5.

We now discuss our results with respect to our second DAG-construction heuristic, which is based on the heuristic of Altin et al. [12] for oblivious ECMP routing configura-

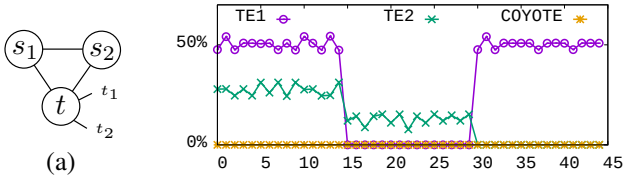
tion. Specifically, [12] presents a heuristic that starts from the link weights provided by the ITZ dataset, and applies a local-search heuristic that greedily changes one link weight if this change improves the worst-case ECMP link utilization across all the admissible traffic demands. We use the output of this procedure as the ECMP configuration and augment it with additional links to obtain COYOTE's per destination DAGs, as explained in Section 2. Fig. 6 presents a comparison of COYOTE and ECMP using the bimodal model as the base traffic demands. We use the above heuristic to compute, for each uncertainty margin in the range $1-5$, increasing in 0.5 increments, the (traditional) ECMP configuration and COYOTE DAGs with respect to the bimodal-based traffic demands. We then compare the worst-case link utilization of the two, again, normalized by the *demands-aware* optimum within the same (augmented) DAGs. We note that ECMP is, on average, almost 80% times further away from the optimum than COYOTE.

Approximating the optimal traffic splitting. We evaluated above COYOTE under the assumption that arbitrarily fine-grained traffic splitting is achievable, yet in practice, the resolution of traffic splitting is derived from the number of virtual links introduced. Clearly, an excessive number of virtual links should be avoided for at least two reasons: (a) each virtual next-hop is installed into the finite-sized Forwarding Information Base (FIB), and (b) injecting additional information into OSPF comes at the cost of additional computational overhead. Our results, illustrated in Fig. 7 for AS 1755 network's topology (all other topologies exhibit the same trend), show that even with just 3 additional virtual links per router interface, COYOTE achieves a 50% improvement over traditional TE with ECMP. We observe that with 10 virtual links the computed routing configuration closely approximates the ideal solution.

Average path lengths. COYOTE augments the shortest path DAG with additional links so as to better utilize the network. Consequently, traffic can potentially traverse longer paths. We show, however, that COYOTE's increased path redundancy does not come at the expense of long paths. Specifically, the average stretch (increase in length) of the paths in COYOTE is typically bounded within a 10% factor with respect to the OSPF/ECMP paths. Fig. 8 plots the average stretch across all pairs for a margin of 2.5. Similar results are obtained for all different margins between 1 to 5. Observe that the DAGs computed by COYOTE rely on shortest-path computation with respect to the link weights, whereas the stretch is measured in terms of the number of hops. Thus, it is possible for the stretch to be less than 1, as is the case, e.g., for BBNPlanet.

4. PROTOTYPE IMPLEMENTATION

We implemented and experimented with a prototype of the COYOTE architecture, as described in Section 2. Our prototype extends the Fibbing controller code, written in Python and provided by Vissicchio et al. [9], and uses the code of Nemeth et al. from [18] for approximating the splitting ratios. We plan to make our code public in the near future.



(a) (b)

Figure 9: Mininet topology (a) and packet drop rate (b).

We next illustrate the benefits of COYOTE over traditional TE, as reflected by an evaluation of our prototype via the mininet [26] network emulator.

Consider the example in Fig 9a: a target node t advertises two IP prefixes t_1 and t_2 and two sources, s_1 and s_2, generate traffic destined for these IP prefixes. As in traditional TE with ECMP, the network operator must use the same forwarding DAG for each destination, this forces either s_1 or s_2 to route all of its traffic only on the direct path to the destination. Thus, three forwarding DAGs are possible: (1) both s_1 and s_2 route all traffic on their direct paths to t (TE1), (2) s_1 equally splits its traffic between t and s_2, and s_2 forwards all traffic on its direct link to t (TE2), and (3) same as the previous option, but s_1 and s_2 swap roles (TE3).

We evaluate these three TE configurations in mininet with links of bandwidth 1Mbps. We measure the cumulative packet drop rate towards two IP destinations, t_1 and t_2, for three 15-seconds-long traffic scenarios, where traffic is UDP generated with `iperf3` and units are in Mbps: $(s_1-t_1, s_2-t_2) = (0,2)$, $(s_1-t_1, s_2-t_2) = (1,1)$, $(s_1-t_1, s_2-t_2) = (2,0)$.

Fig 9b plots the results of this experiment for each of the TE schemes, described above (excluding TE3, which is symmetric to TE2). The x-axis is time (in seconds) and the y-axis is the measured packet loss rate, i.e., the ratio of traffic received to traffic sent (observe that sent traffic is 30 megabits in all scenarios). During the first 15 seconds the experiment emulates the first traffic scenario described above, in the next 15 seconds the second traffic scenario is emulated, and in the last 15 seconds the third scenario is emulated.

Observe that each of the TE schemes (TE1-3) achievable via traditional TE with ECMP leads to a significant packet-drop rate (25%-50%) in at least one of traffic scenarios. COYOTE, in contrast, leverages its superior expressiveness to generate different DAGs for each IP prefix destination, as follows: traffic to for destination t_1 is evenly split at node s_1 and traffic to destination t_2 is evenly split at s_2. This is accomplished by injecting a lie to s_2 so as to attracts half of its traffic to t_2 to the (s_2, s_1) link. Consequently, as seen in Fig 9b, the rate of dropped packets is significantly reduced.

5. RELATED WORK

TE with ECMP. TE with ECMP is today's prevalent approach to TE (see surveys in [1, 2]). Consequently, this has been the subject of extensive research and, in particular, selecting good link weights for ECMP TE has received much attention [6, 7, 12, 27–30]. To handle uncertainty about traf-

fic demands and variation in traffic, past studies also examined the optimization of ECMP configuration with respect to multiple expected traffic demands [6, 30, 31], or even with no knowledge of the traffic demands [11]. Unfortunately, while careful and delicate optimizations of ECMP configuration can be close-to-optimal in some networks [30], this approach is fundamentally plagued by the intrinsic limitations of ECMP, specifically, routing only on shortest paths and equally splitting traffic at each hop, and can hence easily result in poor network performance. Worse yet, this scheme suffers from inherent computational intractability, as shown in [7, 27].

Lying for more expressive OSPF-ECMP routing. The first technique to approximate unequal splitting through ECMP via the introduction of virtual links was introduced by Nemeth et al. in [18] (see also [32]). [18], however, was still limited to shortest-path routing and, consequently, coarse-grained traffic flow manipulation. Recently, Fibbing [8, 9] showed how any set of destination-based forwarding DAGs can be generated through the injection of fake nodes and links into the underlying link-state protocol (e.g., OSPF).

Adaptive TE schemes. One approach to overcoming ECMP's limitations is dynamically adapting the routing of traffic in response to changes in traffic conditions as in, e.g., [27]. Adaptive schemes, however, typically require frequently gathering fairly accurate information about traffic demands, potentially require new routing or measurement infrastructure, and can be prone to routing instability [33], slow convergence, packet reordering, and excess control plane burden [3] (especially in the presence of failures). COYOTE, in contrast, reflects the exact opposite approach: optimizing the *static* configuration of traffic flow so as to simultaneously achieve good network performance with respect to *all*, even adversarially chosen, traffic demands within specified "uncertainty bounds".

Demands-oblivious routing. A rich body of literature on algorithmic theory investigates so-called "(demand-)oblivious routing" [10–12]. Breakthrough algorithmic results by Räcke established that the static (non-adaptive) routing can be optimized so as to be within an $O(\log n)$ factor from the optimum (demands-aware) routing with respect to *any* combination of traffic demands [10]. Applegate and Cohen [11] showed that when applied to actual (ISP) networks, such demand-oblivious routing algorithms yield remarkably close-to-optimal performance. Kulfi [13] uses semi-oblivious routing to improve TE in wide-area networks. Unfortunately, all the above demand-oblivious algorithms involve forwarding packets based on both the source and destination, these immediately hit a serious deployability barrier in traditional IP networks (e.g., due to extensive tunneling [28]). COYOTE, in contrast, is restricted to OSPF-based destination-based routing, and so tackles inherently different (and new) algorithmic challenges and techniques, as discussed in Sect. 2.

6. CONCLUSION

We presented COYOTE, a new OSPF-ECMP-based TE scheme that efficiently utilizes the network even with little/no knowledge of the traffic demands. We showed that COYOTE significantly outperforms today's prevalent TE schemes while requiring no changes whatsoever to routers. We view COYOTE as an important additional step in the recent exploration [8, 9] of how SDN functionality can be accomplished without changing today's networking infrastructure. We next discuss two important directions for future research.

Further exploring destination-based oblivious routing. To efficiently utilize the network in an OSPF-ECMP-compatible manner, COYOTE leveraged new algorithmic insights about destination-based oblivious routing. We believe that further progress on optimizing such routing configurations is key to improving upon COYOTE. We next mention two interesting research questions in this direction: (1) We showed in Section 2 that computing the optimal oblivious IP routing configuration is NP-hard. Can the optimal configuration be provably *well-approximated*? (2) COYOTE first computes a forwarding DAG rooted in each destination node, and then computes traffic splitting ratios within each DAG. The latter computation involves nontrivial optimizations, e.g., via iterative geometric programming, yet, it remains unclear whether traffic splitting within a given set of DAGs is, in fact, efficiently and optimally solvable.

Evaluating COYOTE under real-world network conditions. Our experimental evaluation of COYOTE combined simulations on 16 backbone Internet topologies from the Internet Topology Zoo (ITZ) archive [19] with small-scale experiments on the mininet network emulator [26]. An important direction for future research is experimenting with COYOTE on empirically-derived traffic traces, e.g., data collected from the Internet2 Network [34], and running larger-scale experiments with our prototype implementation of COYOTE on mininet and on SDN-capable network testbeds such as [35].

Acknowledgements

We thank the anonymous reviewers of the CoNEXT PC and Walter Willinger for their valuable comments. We thank Francesco Malandrino for useful discussions about the geometric programming approach, and Olivier Tilmans and Stefano Vissicchio for guiding us through the Fibbing code. This research is (in part) supported by European Union's Horizon 2020 research and innovation programme under the ENDEAVOUR project (grant agreement 644960). The 1st and 3rd authors are supported by the Israeli Center for Research Excellence in Algorithms. The 2nd author is with the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics.

7. REFERENCES

[1] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. *Communications Magazine, IEEE*, 40(10):118–124, 2002.

[2] Ning Wang, Kin Ho, G. Pavlou, and M. Howarth. An overview of routing optimization for internet traffic

engineering. *Communications Surveys Tutorials, IEEE*, 10(1):36–56, 2008.

[3] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. DevoFlow: Scaling Flow Management for High-performance Networks. In *SIGCOMM 2011*.

[4] John Moy. OSPF version 2. RFC 2328, 1998.

[5] C. Hopps. Analysis of an ECMP Algorithm. RFC 2992, 2000. www.ietf.org/rfc/rfc2992.txt.

[6] Bernard Fortz and Mikkel Thorup. Increasing Internet Capacity Using Local Search. *Computational Optimization and Applications*, 29(1):13–48, 2004.

[7] Marco Chiesa, Guy Kindler, and Michael Schapira. Traffic Engineering with Equal-Cost-MultiPath: An Algorithmic Perspective. In *INFOCOM'14*, pages 1590–1598, 2014.

[8] Stefano Vissicchio, Laurent Vanbever, and Jennifer Rexford. Sweet little lies: Fake topologies for flexible routing. In *HotNets-XIII*, pages 1–7, 2014.

[9] Stefano Vissicchio, Olivier Tilmans, Laurent Vanbever, and Jennifer Rexford. Central control over distributed routing. In *SIGCOMM 2015*.

[10] Harald Räcke. Optimal Hierarchical Decompositions for Congestion Minimization in Networks. In *STOC '08*, pages 255–264, 2008.

[11] D. Applegate and E. Cohen. Making Routing Robust to Changing Traffic Demands: Algorithms and Evaluation. *Networking, IEEE/ACM Transactions on*, 14(6):1193–1206, 2006.

[12] Ayşegül Altin, B. Fortz, and Hakan Ümit. Oblivious OSPF Routing with Weight Optimization Under Polyhedral Demand Uncertainty. *Netw.*, 60(2):132–139, 2012.

[13] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert D. Kleinberg, and Robert Soulé. Kulfi: Robust traffic engineering using semi-oblivious routing. *CoRR*, abs/1603.01203, 2016.

[14] David Applegate and Mikkel Thorup. Load optimal MPLS routing with N+M labels. In *Proceedings IEEE INFOCOM 2003*.

[15] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[16] Marco Chiesa, Gábor Rétvári, and Michael Schapira. Lying Your Way to Better Traffic Engineering. *CoRR*, abs/1610.02728, 2016.

[17] S. Boyd, S. Kim, L. Vandenberghe, and A. Hassibi. A Tutorial on Geometric Programming. *Optimization and Engineering*, 8(1):67–127, 2007.

[18] K. Németh, A. Körösi, and G. Rétvári. Optimal OSPF traffic engineering using legacy Equal Cost Multipath load balancing. In *IFIP Networking Conference 2013*.

[19] Internet Topology Zoo. www.topology-zoo.org, 2010.

[20] Configuring OSPF Cisco. http://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/7039-1.html, 2005.

[21] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury-Thomson, 2003.

[22] Mosek ApS. www.mosek.com, 2015.

[23] M. Roughan, A. Greenberg, C. Kalmanek, M. Rumsewicz, J. Yates, and Y. Zhang. Experience in Measuring Backbone Traffic Variability: Models, Metrics, Measurements and Meaning. In *Proc. Workshop on Internet Measurment*, IMW '02, 2002.

[24] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic Matrix Estimation: Existing Techniques and New Directions. *SIGCOMM Comput. Commun. Rev.*, 32(4):161–174, August 2002.

[25] D. G. Cantor and M. Gerla. Optimal routing in a packet-switched computer network. *IEEE Trans. Comp.*, 23(10):1062–1069, 1974.

[26] Brandon Heller, Nikhil Handigol, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container based emulation. In *CoNEXT*, 2012.

[27] Bernard Fortz and Mikkel Thorup. Internet Traffic Engineering by Optimizing OSPF Weights. In *INFOCOM'00*, pages 519–528, 2000.

[28] Yufei Wang, Zheng Wang, and Leah Zhang. Internet traffic engineering without full mesh overlaying. In *INFOCOM'01*, volume 1, pages 565–571, 2001.

[29] Ashwin Sridharan, R. Guerin, and C. Diot. Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks. *Networking, IEEE/ACM Transactions on*, 13(2):234–247, 2005.

[30] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE Journal of Selected Areas in Communications*, 20(4):756–767, 2002.

[31] M. Ericsson, M.G.C. Resende, and P.M. Pardalos. A Genetic Algorithm for the Weight Setting Problem in OSPF Routing. *Journal of Combinatorial Optimization*, 6(3):299–333, 2002.

[32] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. WCMP: weighted cost multipathing for improved fairness in data centers. In *EuroSys 2014*.

[33] D. P. Bertsekas. Dynamic Behavior of Shortest Path Routing Algorithms for Communication Networks. *IEEE Trans. on Automatic Control*, 27:60–74, 1982.

[34] Internet2 NOC. http://noc.net.internet2.edu/, 2016.

[35] OCEAN. http://ocean.cs.illinois.edu/, 2016.