Autonomous Traffic Engineering with Self-Configuring Link Weights

Srikanth Sundaresan, Cristian Lumezanu, Nick Feamster
Georgia Tech

Pierre François
Université Catholique de Louvain

ABSTRACT

Network operators use traffic engineering to control the flow of traffic across their networks. Existing TE methods establish static topologies offline, either by setting link weights or by configuring paths *a priori*. These methods require manual configuration and may not be robust in the face of failures. Some methods also require knowledge about traffic demands and may not be able to handle traffic fluctuations. Even when changes in demand are expected, operators must manually tune network configurations to prepare for them.

Because adjusting configurations is difficult to get right, we start from an extreme design point, asking instead whether it is possible to perform traffic engineering online without having to perform any *a priori* configuration. Our traffic engineering technique, SculpTE, adapts to changing traffic demands by automatically configuring link weights in a stable manner. SculpTE balances load across the network by continually adjusting link weights to expose lightly-loaded paths. We evaluate SculpTE using a simple analytical model and simulations on realistic ISP network topologies. Our results show that SculpTE achieves excellent load balancing, responsiveness, and stability compared to state-of-the-art TE schemes, without requiring network operators to perform any offline configuration.

1. Introduction

Traffic fluctuations, caused by both predictable changes in user behavior and unexpected events such as flash crowds, routing changes, and network failures, can degrade user performance, impede reliability, and waste network resources. To manage resource allocation and balance traffic demand across network links, operators apply traffic engineering (TE) to the paths on which traffic flows in the network. Networking practitioners and researchers have long tried to develop traffic engineering techniques that automatically respond to network failures, planned maintenance, and shifts in traffic without requiring reconfiguration. In this paper, we present the first stable traffic engineering approach that is essentially configuration-free. Although we have evaluated our approach in the context of intradomain traffic engineering for transit networks, the general approach should apply much more broadly in practice.

One of the biggest challenges that traffic engineering approaches face is configuring the network topology[1] (*i.e.*, setting up paths or link weights) that establishes the paths between source and destination. Existing approaches do so through static, *a priori* configuration of paths using tunnels or link weights, and change how traffic is forwarded over those paths. Approaches such as link-weight tuning [13–16] solve an offline optimization to compute link weights. Oblivious routing [3, 46] designs a routing scheme to handle both expected and unexpected traffic demands. "Online" protocols such as TeXCP [24] and MATE [11] establish tunnels *a priori* and shift traffic between these tunnels in response to traffic fluctuations.

These approaches are promising, but they have drawbacks. First, setting up topologies incurs management and configuration overhead, and can be difficult in practice. Even when the traffic demands are known in advance, adjusting the paths to account for a known traffic surge is cumbersome and disruptive [39]. Second, the resulting configurations may not be robust in the presence of link or node failures. Third, some of these approaches rely on knowing or estimating the traffic demands, which is difficult to do accurately in practice.

Because network configuration for traffic engineering is cumbersome and sometimes hard to get right, particularly under stress or failure conditions, we ask instead whether a network could achieve traffic engineering goals without any *a priori* configuration. The answer lies in an approach that automatically searches for lightly loaded paths and adjusts link weights such that load is balanced in a smooth and stable manner. We find that this approach can yield TE methods that achieve load balance, responsiveness, and stability comparable to existing online TE, but (1) do not require *a priori* configuration; and (2) operate even in simple destination-based, hop-by-hop forwarding scenarios.

In this paper, we present SculpTE, an intradomain traffic engineering technique that continually adjusts the network link weights to balance traffic load across links. The link weights in the network topology are derived by periodically sampling the loads on those links; thus, the cost of the link is directly related to the amount of traffic on each link and continually changes as traffic demands change. We have designed these link-weight updates to ensure that

[1]Throughout the paper, unless otherwise stated, we use the term "topology" to refer to the network-layer paths between nodes that map to the same underlying physical topology. Each individual topology has its own set of link weights.

the network is both responsive and stable; SculpTE avoids known problems with load-based link-weight tuning [6]. SculpTE is autonomous, does not require *a priori* topology configuration, and works well over a wide range of settings, even when nothing is known about traffic demands or the network that it is deployed on.

To preserve stability, SculpTE must divert traffic away from only the most loaded link (or set of links). A natural approach to achieve this is to increase the weight of the link by its *key metric*. The key metric [19] of a link for an ingress-egress pair is the additional weight that must be added to the link to remove it from the unique shortest path between that ingress-egress pair. By increasing the weight of the most loaded link by its key metric for one or more paths, SculpTE shifts traffic away from that link for at least one of the shortest paths that traverse it.

Continually adapting the topology by adjusting the weight of the most loaded link presents several challenges related to stability and responsiveness. First, when there are multiple loaded links, SculpTE may shift traffic continuously amongst them, leading to oscillations. To avoid this situation, SculpTE adjusts paths in a way that moves traffic to a path or set of paths that do not traverse another highly loaded link. Second, because SculpTE responds to instantaneous changes in traffic, sudden link weight updates, although minimal, may still trigger instability. SculpTE mitigates this by using multiple topologies; each router maintains k independent topologies and spreads flows randomly across these topologies. Link weights are adjusted on only one topology at a time.

This paper makes the following contributions. First, we present the design of SculpTE, a novel TE technique that is stable, responsive, yet essentially configuration free (Section 3). SculpTE balances load by continually searching for lightly loaded paths and configuring link weights so that traffic is shifted to those paths in a stable manner. SculpTE's design explicitly minimizes the possibility of instability and oscillation, without affecting responsiveness. Second, we evaluate SculpTE on realistic ISP topologies, both under normal operation and in stress conditions, and compare it to state-of-the art offline and online TE approaches (Section 4). We show that (1) SculpTE consistently achieves close to optimal load balance, for a variety of ASes (within 10% of optimal utilization); it outperforms offline TE such as link-weight optimization and InvCap, and achieves performance close to published results of TeXCP, without requiring *a priori* configuration of link weights or paths; (2) SculpTE converges quickly (less than 30 iterations) to acceptable load balance; (3) SculpTE maintains close to optimal performance and responsiveness under a variety of "stress" scenarios, such as link failures or traffic surges; (4) SculpTE is robust to initial parameter settings, such as the number of topologies and the frequency with which links are updated. We also describe how SculpTE can be combined with approaches such as route caching [27] and flowlet routing [43] to limit the amount of packet reordering that dynamically reconfiguring the topology might introduce (Section 5).

2. Related Work

Existing traffic engineering methods are either online or offline. Online TE approaches preset multiple virtual paths and adaptively distribute traffic across them based on feedback from the network. In contrast, offline TE implicitly establishes the topology by configuring link weights or routing based on existing traffic demands. Unlike SculpTE, both online and offline TE route traffic over fixed topologies that must be established in advance.

Online TE. The ARPANET routing protocol was an online, adaptive algorithm [31] that updated link weights based on measured link delays. Due to the unpredictable relationship between loads and queuing delay, especially at high loads [6, 45], this approach was unstable. A revised protocol used average link delays to dampen feedback and reduce oscillations [26, 32]. SculpTE also adapts link weights online but does so based on measured link utilization to continually reflect the current state of the network.

Gallager proposed one of the earliest online algorithms to achieve optimal-delay routing [20]. His distributed scheme guarantees minimum delay and loop-free routing given static network and traffic demands.

Current online TE proposals preset multiple paths between the same pair of ingress-egress nodes, using RSVP-TE [4, 8] and adapt routing across these paths. MATE [11] uses instantaneous network state to come up with a routing scheme that minimizes delay and reduces congestion. TeXCP [24] uses a feedback system based on XCP [25] to divert traffic away from congested links. He *et al.* [23] split traffic across multiple paths using distributed feedback and optimizing multiple utilization functions.

To achieve robustness, some online TE approaches use multiple topology routing (MTR) to gain access to more paths [41]. This technique naturally improves load balancing and responsiveness to faults and unexpected surges, as shown by Mitzenmacher [35]. Kvalbein *et al.* [29] store multiple topologies such that there is always one topology that avoids a certain failed or congested link. Homeostasis [28] diverts traffic from congested links by choosing alternate paths that do not increase delay. As in these approaches, SculpTE splits traffic over multiple topologies to preserve stability.

Although online TE methods achieve good load balance and robustness to network faults, they still configure paths *offline* (*e.g.*, by setting up tunnels), which is difficult to get right. Suboptimal paths can leave the network unprotected from traffic surges or failures. At the same time, computing optimal configurations to account for traffic diversity can be difficult, and continually adjusting configurations to account for traffic shifts is cumbersome, even when the traffic shifts are planned or known in advance [39]. In SculpTE, we use a different approach: instead of computing a set of (possibly suboptimal) paths, we adapt paths online to the variations in traffic. This preserves the responsiveness of online TE, but eliminates the complexity of presetting paths.

Offline TE. Recent TE methods, such as IGP Weight optimizer (IGP-WO) [13, 14], configure the set of link weights that minimize the maximum network utilization for a fixed traffic demand matrix. Because traffic patterns vary over time, it is important to find paths that offer good performance even when demands change. Fortz *et al.* compute link weights for a set of demand matrices that capture expected traffic changes or link failures [15, 16]. Other solutions set up routing schemes in advance that optimize for expected demands but provide performance guarantees for rare, abnormal demand surges [3, 47].

Offline TE approaches must make assumptions about traffic demands; thus, these approaches may be vulnerable to unexpected traffic surges or link failures. Like most offline TE methods, SculpTE uses link weights to implicitly establish paths between ingress-egress nodes. However, SculpTE does not configure weights based on estimated demands; instead, it makes no assumptions about the traffic and continually adjusts the weights based on current network conditions, reacting quickly to unexpected traffic surges or link failures.

3. Design

In this section, we present the design of SculpTE. We begin by introducing terminology and identifying the goals that a good TE algorithm should achieve. We then describe SculpTE and show how it discovers good paths to ease congestion and limit oscillations. We also discuss how using multiple topologies mitigates instability.

Terminology. A *topology* is a network in the form of a graph (V, E), with V being the set of nodes and E the set of weighted edges (links). The function $weight(l) \rightarrow \mathbb{Z}^+$ defines weights on each link l in E. *Updating* a topology means updating a subset of link weights, which are used to determine paths between ingress-egress pairs. An *SD* pair represents two nodes S and D that exchange traffic. We denote the path between S and D as *SD path.* A shortest path between an SD pair refers to the lowest-cost path between source S and destination D.

Assumptions. First, routers running SculpTE use *Equal Cost Multi-Path* (ECMP), which divides traffic flows across multiple paths with the same cost. Second, routers have the ability to maintain *multiple topologies*, which refers to multiple Forwarding Information Bases (FIBs) that are independent of each other. Third, IGP convergence time is fast. Finally, SculpTE applies link-weight updates in a synchronized manner across the network. In Section 5, we explain how the final two assumptions can be relaxed or realized in practice.

Design goals. Traffic engineering adjusts the paths that traffic takes to satisfy pre-defined performance objectives. Every TE algorithm should satisfy the following goals:

- **Load balance** A common performance objective is to minimize the maximum utilization in the network, by spreading load across many paths. This improves network reliability, by avoiding congested links, and performance, by creating paths with more available bandwidth and hence lower delay.
- **Responsiveness** TE must quickly respond to changes in demands or link failures, while still offering acceptable performance.
- **Stability** When adapting paths after link failures or changes in demand, TE must ensure that paths do not change so rapidly, or traffic is shifted in such a way that it induces oscillation in traffic or route flapping.
- **Minimum configuration** Manually establishing and adjusting configurations is difficult to do correctly in practice. To limit the impact of configuration errors, TE must be autonomous without or with little *a priori* configuration.
- **Low setup and management overhead** The overhead from setting up and adjusting paths must not affect performance, responsiveness, and stability.

In Section 4, we evaluate how well SculpTE achieves each of these goals in various scenarios.

3.1 SculpTE

SculpTE continually balances load across the network by adjusting link weights online to gradually move traffic away from the most loaded link onto alternate, lightly loaded paths. At each iteration, SculpTE identifies the most loaded link, l, and updates its weight such that l is no longer on a unique shortest path between an SD pair. This update creates at least two equal-cost shortest paths between S and D, out of which only one traverses l. Using ECMP routing, SculpTE splits the traffic from S to D equally across these paths, alleviating the load on l.

Continually shifting traffic away from highly loaded links can introduce oscillations. To preserve stability, SculpTE must move traffic in small, steady steps. SculpTE has two approaches that together ensure near-optimal, stable load balancing. First, it finds good alternate paths and avoids other congested links by using the *key metric* of the most loaded link. Second, it distributes traffic across *multiple, independent topologies* to dampen feedback and reduce oscillations. Next, we describe these approaches in detail.

3.2 Discovering Alternate Paths

To ease congestion and balance load while preserving stability, SculpTE diverts some traffic away from only the most loaded link in the network. A natural way to achieve this is to increase the weight of the link by its *key metric* [19].

3.2.1 *The key metric*

The key metric of a link for a pair of endpoints is the additional weight that must be added to the link to remove it from the unique shortest path between the endpoints. We define the *alternate shortest path* for a link l and a pair of endpoints S and D whose traffic traverses l as the shortest path between S and D that does not traverse l. Intuitively, the key metric for l with respect to the SD pair is the difference between the cost of the shortest path and the cost of the alternate shortest path. Adding the key metric to the weight of l makes the cost of the shortest path equal to the cost of the alternate shortest path. ECMP routing enables traffic to be distributed evenly across the equal-cost paths.

3.2.2 *Choosing a key metric*

In practice, many pairs of endpoints send traffic through the most loaded link l. Each pair has its own key metric with respect to l. Next, we discuss how to choose the key metric to apply to l.

The choice of key metric presents a tradeoff between stability and aggressiveness in load balancing. Choosing the minimum key metric results in traffic moving away from only *one* path (assuming all paths have unique key metrics with respect to l). Conversely, applying the maximum key metric affects traffic on all shortest paths traversing l. This is because applying the maximum key metric to l creates new shortest paths, that no longer pass through l, for all pairs with lower key metrics. As a result, all traffic through l (except that of the pair with maximum key metric) move to the new shortest paths.

SculpTE must move just enough traffic to alleviate the load on l, yet preserve stability. To do so, SculpTE applies the minimum key metric of all paths traversing l. By increasing the weight of l with the smallest key metric, only traffic between one pair of endpoints (the pair that yields the smallest key metric) is shifted from the most loaded link. Traffic between all other pairs is unaffected. In Figure 1, we show how SculpTE chooses the key metric on a simple topology. Figure 5 shows the pseudocode for the key metric computation algorithm.

The key metric for the most loaded link l and an SD pair may be 0 or may not exist. The key metric is 0 when l is already on one of multiple equal cost paths from S to D. In this case, SculpTE increases the weight of l by 1, which completely removes from l the traffic from S to D. The key metric does not exist when all paths between S and D traverse l. In this case, no traffic between S and D can be shifted away from l. However, it may be possible to shift traffic between another pair of endpoints whose shortest path traverses l. We discuss the case when no SD pair has alternate paths that bypass l in Section 3.2.3.

3.2.3 *Choosing a key metric for improved stability*

Choosing and applying the key metric as described above does not always prevent oscillations. Consider the following scenario. After applying the minimum key metric to the most loaded link l, part of l's traffic moves to another loaded link, l', which now becomes the new most loaded link. At the next iteration, when we apply the key metric to l', some traffic may move back to l, making it the most loaded link once more. This oscillation may continue *ad infinitum*.

To avoid such a situation, SculpTE maintains a set L of highly loaded links and attempts to minimize the traffic that it shifts from l to paths traversing these links. To compute the key metric for l, SculpTE considers only alternate shortest paths that do not contain links from L. However, these alternate shortest paths may not be shortest anymore after applying the key metric to l. A lower cost path that traverses a link in L other than l and is not considered when computing the key metric may exist. We depict an example scenario in Figure 2(a)-(c).

To reduce the traffic shifted from l to other links in L for some SD pair, we apply the key metric with respect to SD to each link in L that would otherwise end up being on the unique shortest path between S and D, which ensures that all alternate SD paths that traverse a link in L have the same cost as an alternate shortest path that does not traverse any link in L (see Figure 2). Although this approach does offload some traffic to other links in L, the amount of traffic is smaller because it creates more equal-cost paths. If another link in L becomes the new most loaded link, SculpTE adds 1 to its weight at the next iteration, since it is already on one of many equal-cost paths. This adjustment prevents persistent oscillations. We present the pseudocode of the alternate path computation algorithm in Figure 4. Figure 3 shows the weight update algorithm that SculpTE performs at each iteration.

Selecting the initial links for L is challenging. Choosing the most utilized links works well in most cases but may lead to instability when link capacities are very skewed. Let the residual capacity of a link l be $(1 - u_l)c_l$, where u_l is the utilization and c_l the capacity of link l. Consider the case where a low capacity link that is not in L has a lower residual capacity than a higher utilization, higher capacity link that belongs to L. Shifting traffic to the link with lower residual capacity will likely cause oscillations, as its utilization will grow faster than the link with more capacity. Selecting the links with lowest residual capacity for L does not solve the problem because it may ignore many high utilization links that have high capacity.

We define the *weighted residual capacity* of a link l as $(u_m - u_l)c_l$, where u_m is the maximum network utilization for that iteration. The L set contains links with the

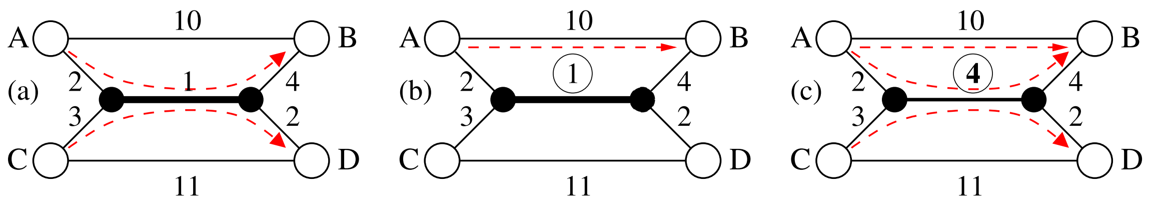


Figure 1: Running SculpTE on a simple topology involves the following steps: (a) identify the most loaded link l (thick line); AB and BC shortest paths traverse l; (b) compute the key metric for all pairs of nodes whose shortest path goes through l: $k(A, B, l) = 3$, $k(C, D, l) = 5$; the AB pair yields the minimum key metric (we show the alternate shortest path for AB); (c) add the value of the minimum key metric to the weight of l; now there are two shortest paths between A and B; half of the traffic between A and B will move away from l, lowering its utilization. Note that the traffic between C and D does not change; it still traverses l.

```
UPDATE-TOPOLOGY $(E, m)$
      ▷ $E$ is the set of edges in the topology
      ▷ $m$ is the number of links in $L$
 1    $k$, $SD$ pair, $cost$, $L$,$m$ = GET-ALT-PATH $(E)$
 2    if $L$ is $\emptyset$
        then ▷ All $m$ links in $L$ disconnect $E$
               ▷ No alternate paths for any path traversing $l$ in $L$
 3             return
 4    if congested link $l$ already on ECMP shortest path
 5      then
 6             $weight(l)$++ ▷ Removes $l$ from ECMP shortest path
 7             return
 8    Sort $L$ in increasing order of weighted residual capacity
 9    $E' \leftarrow E - L$ ▷ Remove $L$ from $E$ for computing low utilization paths
10    for link $l$ in $L$
          do
11             Add $l$ back to $E'$ to check if $l$ in new shortest path
12             Get shortest path $p'$ for $SD$ pair in $E'$
13             $cost'$ is the cost of $p'$
14             $k' = cost' - cost$
15             $weight(l)$ += $k'$ ▷ Ensure $l$ is also on ECMP path
16    return ▷ Key metric successfully applied to $L$
```

Figure 3: Algorithm to update link weights

```
GET-ALT-PATH $(E, m)$
     ▷ Attempt to find the key metric $k$ link with least weighted
     residual capacity in $L$ with a path that avoids all $m$ links in $L$
1    $L$ is the set of $m$ links with weighted residual capacity,
     within 10% of max load, sorted low to high, i
     and upper bound by 10% of total number of links
2    for $i \leftarrow 0$ to $m$
         do
3            $l \leftarrow L[i]$ ▷ iteration tries to find key metric for $l$
             ▷ For current $l$, find only paths that do
             *not* pass through links with higher utilization
4            while $size(L) > i$
                 do
5                    $k$,$SD$ pair, $cost$ = GET-KEY-METRIC $(E, L, l)$
6                    if $k$ exists
                       then ▷ Alternate paths that bypass $l$ exist
7                             return $k$, $SD$ pair , $cost$, $L$
                       else ▷ $L$ is cut-set of graph
8                             Pop link from $L$ with max norm. residual capacity
     ▷ $L$ is empty; there is no alternate path
9    return $\emptyset$
```

Figure 4: Algorithm to compute alternate shortest paths

```
GET-KEY-METRIC (E,L,l)
 1   $E' \leftarrow E - L$ ▷ Remove $L$ from $E$ for computing paths
 2   Get all $SD$ pairs $P$ with shortest paths through $l$
 3   for each $SD$ pair $\in P$
         do
 4           Get path $p$ for $SD$ pair in $E$
 5           Get path $p'$ for $SD$ pair in $E'$
 6           $cost_p$ is cost of $p$
 7           $cost_{p'}$ is cost of $p'$
 8   Get $SD$ pair with $min(cost_{p'} - cost_p)$
 9   $k \leftarrow min(cost_{p'} - cost_p)$
10   return $k$, $SD$ pair , $cost_{p'}$
```

Figure 5: Algorithm to compute the key metric

lowest weighted residual capacity during that iteration, which avoids the problems with both the approaches listed above. Note that the most utilized link has zero weighted residual capacity. In our evaluation (Section 4), we find that limiting L to links whose weighted residual capacity is less than 10% of the load of the most utilized link gives excellent performance and minimizes oscillations. We also bound the size of L to 10% of the total number of links to ensure that an adequate number of alternate paths are available. We also show that the choice of these settings are arbitrary, and do not affect the performance of SculpTE.

At times, it may be impossible to find alternate paths that do not traverse links in L (*i.e.*, when L is a cut-set for the topology). At this point, SculpTE attempts to find an alternate shortest path that avoids the most loaded link, but traverses another link in L. To do so, SculpTE reduces the size of L by iteratively removing the link with the highest weighted residual capacity until L is no longer a cut-set for the topology (lines 4 to 8 in Figure 4). If the size of L is one (*i.e.*, it contains only the most loaded link), and L is still a cut-set, SculpTE cannot decrease the maximum utilization, but it may be possible to shift traffic away from other congested links. SculpTE tries each link in L in increasing order of their weighted residual capacity until it finds a link from which it can deflect traffic (loop starting line 2, Figure 4). SculpTE also ensures that it only deflects traffic from that particular link if it can find paths with links with higher weighted residual capacity than itself. (line 4, Figure 4). Although this step does not minimize the maximum utilization in the network, it improves efficiency by balancing load more evenly among other highly loaded links.

3.3 Using Multiple Topologies

SculpTE reacts to instantaneous changes in traffic by adapting paths based on current link utilization. However, when too much traffic moves to the new paths, the potential for instability increases. When the most loaded link,

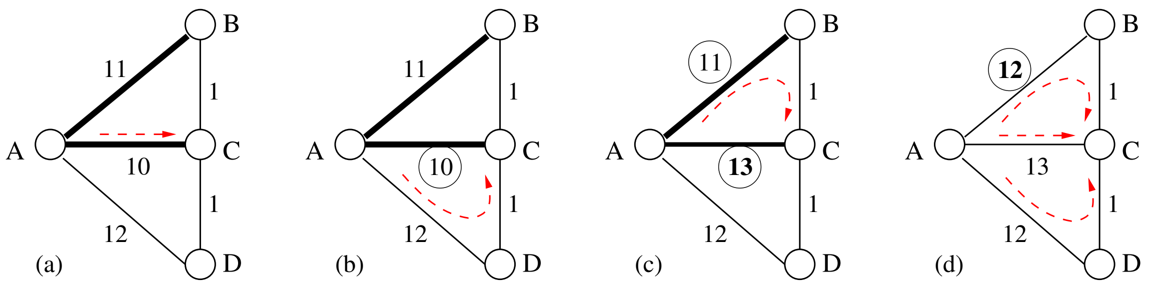


Figure 2: Deflecting traffic away from a congested link. (a) AC **is the most loaded link,** AC **and** AB **are in the set** L **of highly utilized links; (b)** ADC **is the shortest alternate path between** A **and** C **that avoids all links in** L**; (c) when we apply the key metric of** AC **with respect to** SD **pair** (A, C)**;** $k(A, C, AC)(= 3)$ **to** AC**, the path** ABC **becomes shortest; (d) because we want to avoid moving traffic to links in** L**, we also apply** $k(A, C, AB)(= 1)$ **to link** AB**; this creates three equal-cost paths from** A **to** C**.**

l is updated with the key metric with respect to SD, a new alternate shortest path appears and half of the traffic from S to D moves to the new path. If the amount of traffic exchanged by S and D is large, significant oscillations may occur, as another link would now be overloaded. Similarly, when multiple pairs of endpoints have the same key metric, the amount of traffic deflected from the most loaded link is proportionally larger.

To mitigate the effect of destabilizing feedback from link utilization, SculpTE sets up k multiple, independent topologies. Each flow is randomly assigned to one topology. At each iteration, SculpTE updates a single topology. This approach enables SculpTE to (1) prevent reaction to instantaneous traffic patterns, because each topology is updated every k iterations, and (2) reduce the amount of feedback, because flows are divided among k topologies. Prior work by Mitzenmacher [35] shows that having choice helps in achieving good load balancing: multiple topologies increase choice in network paths as different topologies can open up different sets of paths.

4. Evaluation

In this section, we evaluate SculpTE using simulations on real ISP topologies. We show that SculpTE converges quickly to its best performance, which is close to optimal load balancing (Section 4.2), under normal as well as "stress" conditions, such as link failures and sudden surges (Section 4.3). We also study how varying SculpTE's parameters (*i.e.*, number of topologies, size of set L, update frequency) affect its performance (Section 4.4). We show that SculpTE performs comparably with state-of-the-art online and offline traffic engineering schemes, such as TeXCP [24] and MATE [11] (Section 4.5). Finally, we present a simple analysis on the stability of SculpTE.

4.1 Experiment Setup

Simulator. We wrote a flow-level simulator in Python to conduct our experiments[2]. Flow arrivals follow a Poisson distribution as in previous work [24, 28], and their sizes are derived from a truncated Pareto distribution [28]. The minimum flow size is 2 MB and the maximum is 8 GB. The flow-size distribution does not affect the results, however; these settings attempt to capture the wide variety of flows that exist in networks. We assign flows to topologies at random; once a flow is assigned to a path, is not re-routed when link weights change.

AS Topologies. We evaluate SculpTE on real and inferred AS topologies: the Abilene network [2] and commercial ISP networks obtained from Rocketfuel [44]. We present results for six most representative topologies, summarized in Table 1. To speed up the experiments without affecting results, we prune off the "hanging trees" from each AS topology[3]. Applegate *et al.* performed a similar optimization when evaluating various offline routing schemes [3]. For Rocketfuel topologies, we assign capacities of either 2.5 Gbps or 10 Gbps to links, based on the node connectivity, as in prior work [24]. For Abilene, each link has 10 Gbps.

Traffic demands. We generate traffic demands using the gravity [42] and the bimodal [7] models (also used in [3, 24, 33]). To compute optimum routing for each demand matrix, we use a multi-commodity formulation [34] with average instantaneous demands as input and solve it using `glpk`, the GNU Linear Programming Kit [1].

Metric. We use *the deviation from optimal routing* to compare the performance of SculpTE to other TE schemes. We compute the deviation from optimum of a TE method as the ratio between the average maximum utilization obtained with the method and the average maximum utilization with optimal routing. We compute all utilizations using instantaneous demands.

4.2 Performance

We investigate how SculpTE balances load and achieves convergence under normal conditions. We begin by comparing SculpTE to standard offline TE approaches and show that it balances load more evenly. We then show that SculpTE converges quickly (*i.e.*, it achieves close to optimal results in only a few iterations). Unless otherwise

[2]The simulator is available for download at http://gtnoise.net/projects/9-future-routing-for-internet-and-data-centers/25-sculpte.

[3]For AS 3257 we use the full topology because of problems in generating IGP-WO link weights for the reduced topology using the Totem toolset.

ISP	AS ID	No. of nodes	No. of links	Reduced no. of nodes	Reduced no. of links
Metromedia	6461	19	34	15	30
Tinet	3257	41	87	40	76
Level3	1	42	55	23	36
Level3	6395	25	54	22	51
AOL	1668	26	64	21	32
Abilene	—	11	14	—	—

Table 1: List of topologies used in simulations. The reduced topologies indicates that the "hanging trees" have been removed.

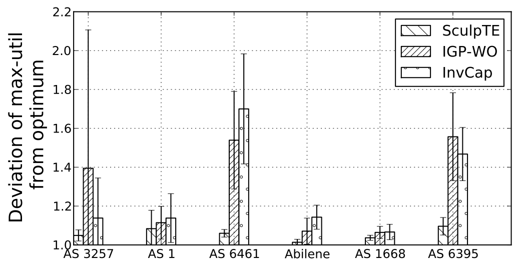


Figure 6: Comparison of various schemes relative to optimal.

noted, the update frequency is ten seconds, and the number of topologies is three. We made the initial link weights (for SculpTE) inversely proportional to link capacity.

Comparison to offline schemes. We compare SculpTE to standard offline TE algorithms such as IGP-WO [14, 15], proposed by Fortz *et al.*, and InvCap [10]. IGP-WO configures link weights that minimize the maximum utilization given the network graph and an expected traffic matrix. InvCap sets weights inversely proportional to the link capacity. We generate 20 different traffic matrices, using the gravity and bimodal models, such that average maximum utilization with optimal routing is between 0.1 and 0.9. This setup captures a wide variety of traffic demands in low- and high-utilization scenarios. We compute the IGP-WO weights for each traffic matrix using the Totem toolset [30]. Figure 6 shows the average deviation from the optimum of SculpTE, IGP-WO, and InvCap. The average maximum utilization for SculpTE is much closer to optimum than for IGP-WO and InvCap. The variation in performance obtained across different ASes with the offline schemes is high, in contrast to SculpTE, which performs uniformly well. It is interesting to note that even optimizing for known traffic matrices does not guarantee good performance with IGP-WO. Sudden variations in traffic make it hard for offline schemes to perform reliably.

Convergence time. Convergence represents the time taken to achieve the best performance (*i.e.*, lowest deviation from optimum routing) and is an important metric for any online traffic engineering scheme. SculpTE's initial link weights are the same as for InvCap. We measure the number of iterations SculpTE takes to converge to its steady-state value after the transient period, when it starts applying link-weight updates. Figure 7 shows how the deviation from optimum varies as the number of iterations

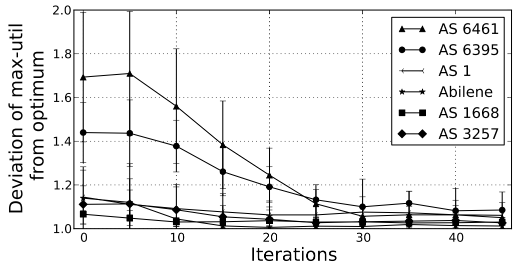


Figure 7: Convergence time for SculpTE.

Topology	Scheme	Normal Demand	Surge Demand	Subsided Demand
Abilene	SculpTE	1.01	1.02	1.01
	IGP-WO$_{multi}$	1.20	1.23	1.21
AS 6461	SculpTE	1.06	1.06	1.06
	IGP-WO$_{multi}$	1.64	1.67	1.65
AS 6395	SculpTE	1.10	1.07	1.17
	IGP-WO$_{multi}$	1.52	1.68	1.51
AS 1	SculpTE	1.09	1.07	1.14
	IGP-WO$_{multi}$	1.18	1.19	1.18

Table 2: Performance of SculpTE compared to IGP-WO$_{multi}$ under sudden surges in load. Normal demand denotes the normal traffic, Surge Demand denotes that certain source-destination pairs see a spike in demand, and Subsided Demands denotes that the spike subsides, but to a level higher than Normal Demand.

increases. The average maximum utilization converges within about 10% of optimal within 30 iterations, even when the initial weights are suboptimal (AS 6461 and AS 6395). If the initial weights perform better, convergence occurs even faster (within 10% after 10 iterations).

4.3 Performance Under Stress Conditions

We evaluate how SculpTE performs under unexpected conditions such as traffic surges and link failures.

Traffic surges. We compare the performance of SculpTE and IGP-WO$_{multi}$ under sudden load surges. IGP-WO$_{multi}$ [15] is a variation of IGP-WO that seeks to protect the network against unexpected changes in demands by optimizing link weights over multiple traffic matrices. To introduce a traffic spike, we choose ten random pairs of nodes in the initial traffic matrix and inflate the demand between them by a factor of five. We then obtain a third demand matrix, the subsided demand, by inflating the original demand between the same ten pairs of nodes by a factor of 1.5. These three matrices are the input to IGP-WO$_{multi}$.

We run SculpTE and IGP-WO$_{multi}$ with 20 base traffic matrices on four ASes. Table 2 shows the results. SculpTE maintains its close-to-optimum performance under a variety of traffic demands. On the other hand, the results for IGP-WO$_{multi}$ show that optimizing link weights to handle multiple traffic demands makes the network stable to these surges, but with a significant penalty in performance compared to IGP-WO (see Fig. 6).

Link failures. We compare the performance of SculpTE and IGP-WO$_{fault}$ when link failures occur. IGP-

Topology	Scheme	Number of link failures			
		0	1	2	3
Abilene	SculpTE	1.01	1.01	1.01	1.03
	IGP-WO$_{fault}$	1.15	1.16	1.24	1.16
AS 6461	SculpTE	1.04	1.05	1.11	1.10
	IGP-WO$_{fault}$	1.41	1.43	1.66	1.66
AS 1	SculpTE	1.06	1.11	1.07	1.04
	IGP-WO$_{multi}$	1.10	1.16	1.38	1.20

Table 3: Performance of SculpTE compared to IGP-WO$_{fault}$ under link failures.

WO$_{fault}$ [16] is a variation of IGP-WO that generates link weights such that the network is protected against critical link failures. For each AS topology, we remove three links sequentially, ensuring that we leave no "hanging" nodes. Table 3 presents the deviation from optimum after each link failure. We see that SculpTE is not affected by link failures: it maintains its performance after failures. IGP-WO$_{fault}$ performs reasonably well, too. As with the case of optimizing for traffic surges, this comes at a cost of worse performance compared to IGP-WO, which optimizes for a single demand matrix and network topology (Figure 6). Unlike SculpTE, IGP-WO$_{fault}$'s performance is sub-optimal when there are no failures.

Both cases illustrate a classic dilemma associated with offline TE: optimizing for the rare occurrences (traffic surges, link failures) produces sub-optimal performance under normal circumstances. On the other hand, optimizing for the normal case runs the risk of significant performance hit when the rare event does occur. By adapting to current network state, SculpTE avoids such issues.

4.4 Parameter Tuning

We study how varying the number of topologies, the update frequency, and the size of set L affects the performance of SculpTE. We show that SculpTE is robust to these settings and allows a wide margin for error.

Number of topologies. Multiple topologies improve the stability of SculpTE. Figure 8 shows the deviation from optimum as we vary the number of topologies. Initially, the performance improves with more topologies. Because less traffic is shifted at each iteration, the feedback from link utilization is smaller, which helps the network stabilize without losing its responsiveness. As the number of topologies increases, performance degrades slightly, because it takes longer for SculpTE to reflect the network state (only one topology is updated at one iteration), thereby affecting responsiveness. That SculpTE requires only a few topologies to achieve excellent performance is significant: the memory footprint needed to store multiple FIBs is not very high.

Update Frequency. SculpTE updates link weights at fixed time intervals. Because the network state continually changes, it is important that updates be frequent. On the other hand, updating links too often may lead to IGP convergence problems. Figure 9 shows the effect of update frequency on performance. As expected, updating

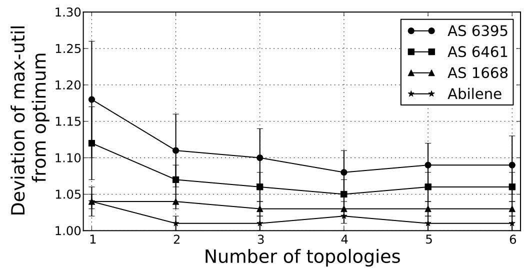


Figure 8: Effect of number of topologies.

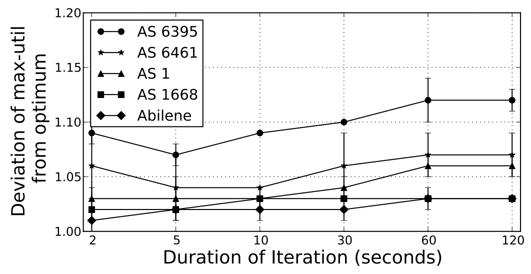


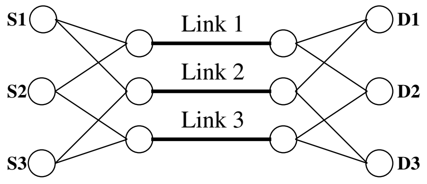
Figure 9: Effect of frequency of updates.

less frequently degrades performance because SculpTE is slower to capture the current state. As with the choice of the number of topologies, performance is remarkably robust to the update frequency. Even with infrequent updates (every 120 seconds), the deviation from optimum is less than 15%.

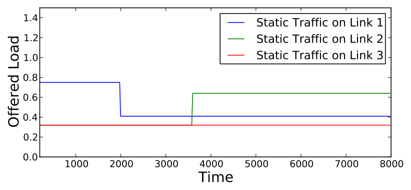
The L set. To reduce oscillations when shifting traffic from the most loaded link, SculpTE tries to avoid links with normalized residual capacity lower than 10% of the maximum utilization, as mentioned in section 3.2.3. We also upper-bound the size of the L set to 10% of the total number of links. SculpTE is quite robust to these settings: varying both parameters between 0.05 and 0.30 has a negligible effect on performance. We note that high values for both these parameters would have the effect of reducing choice of alternate paths, but SculpTE adapts the L set in its search for alternate paths.

4.5 Comparison with online TE

We compare the performance of SculpTE with TeXCP [24] and MATE [11]. Because MATE is proprietary and TeXCP was not available upon request, we compare against their published results. We use the same network (shown in Figure 10(a)) and static and dynamic Poisson traffic sources (shown in Figure 10(b)) as used in the original experiment (as confirmed by TeXCP's authors). The static traffic cannot be re-routed, while the dynamic traffic can. The experiment tests the protocols' responsiveness because the static load drops at time 2000 and surges at 3600. Figures 11(a), 11(b) show the published results of MATE and TeXCP respectively for this experi-



(a) The topology used for the experiment



(b) Static traffic that cannot be re-routed

Figure 10: Experiment setup for MATE and TeXCP comparison

ment, and 11(c) show the results for SculpTE using three topologies and an update frequency of five seconds. We see that SculpTE is more stable than MATE and comparable to TeXCP. The small oscillation seen in SculpTE's performance is because it continuously strives to reduce the load of the most loaded link. However, we see that the oscillation is tightly bound; on the whole, SculpTE responds quickly, but without inducing instability.

4.6 Stability

We aim to understand how SculpTE works and why it is stable. We evaluate SculpTE on a simple network where pathological oscillations are expected. We show the nature of oscillations that SculpTE might induce and how these oscillations can be mitigated. We also perform a simple theoretical analysis to understand the limitations of SculpTE. Although both the model and the analysis make some simplifying assumptions, they provide important insights into how SculpTE works.

We introduce an oscillation bound, ϕ, to quantify the stability of SculpTE. This bound is the difference between the utilizations of bottleneck links, normalized by the maximum utilization. To compute the oscillation bound, we find the average normalized difference between the utilization of the bottleneck links as a result of weight updates. A high oscillation bound reflects instability and occurs when a large amount of traffic is continually shifted from one bottleneck link to another.

4.6.1 *Simulation*

We want to compute the oscillation bound in an extreme scenario where the potential for oscillation is high. Therefore, we consider a network where all traffic fluctuates between two bottleneck links, as in Figure 12. "Link 1" and "Link 2" are the two bottleneck links, with finite capacity, and initial weights of 10. All other links have infinite capacity. Each of the n sources S_i sends traffic to a single destination D. Each S_i has two paths to D, through Link 1 and Link 2. We initialize link weights such that all the sources select Link 1 initially. We also set the weights of links S_iA and S_iB such that key metric for Link 1 with respect to each S_iD path is unique.

We simulate SculpTE while varying both the number of independent topologies and the number of sources. The oscillation bound represents the average normalized difference in load between Link 1 and Link 2. Figure 13 shows that the oscillation bound decreases when adding more topologies and sources. A few shortest paths traversing a link and two or three topologies is sufficient to limit oscillation. This is important because SculpTE relies, for its effectiveness, on having a choice between multiple SD pairs when it searches for paths avoiding bottleneck links. Having many such paths means that the contribution of each path to the load of the link is proportionately lower, therefore diverting its traffic away from the link minimizes the chances of oscillation.

4.6.2 *Simplified analysis*

We now derive the oscillation bound for the two bottleneck link case.Let $\mathcal{G}$ be a network with T topologies. Let n paths go through l, the most loaded link. For each path p_i passing through l, $\mathcal{D}_{p_i}$ is the demand associated with the endpoints of p_i. The total traffic through l, $F_l = \sum_{p=1}^{n} \mathcal{D}_{p_i}$. To simplify our analysis, we assume that all demands are approximately equal: $\forall p_i, p_j, \mathcal{D}_{p_i} \approx \mathcal{D}_{p_j} \approx d$. Therefore,

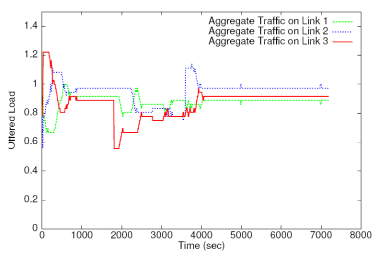
$$F_l \approx n \times d \tag{1}$$

Traffic between any pair of end-nodes is divided randomly among T topologies. Therefore, each topology gets $\frac{d}{T}$ of the total demand between any pair.

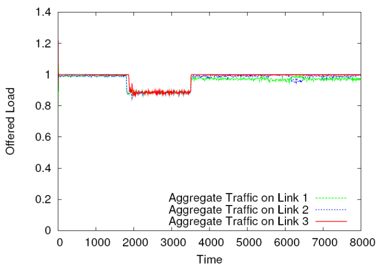
At each iteration, SculpTE applies the key metric such that one SD pair has half its traffic in a single topology diverted away from l. This amount, ϕ_l, is given by: $\phi_l = \frac{d}{2T} = \frac{F_l}{2nT}$ If we consider SculpTE to be a feedback system that minimizes the maximum utilization, the oscillation bound is the maximum feedback to the system at every iteration t:

$$\phi_l(t) = \begin{cases} \frac{F_l}{2nT} & \text{if } l \text{ is the most congested link} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

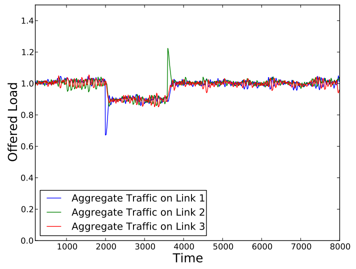
The feedback at time t, $\phi_l(t)$, is inversely proportional to the number of topologies and the number of shortest paths that traverse l. For reasonably sized networks, where there are many pairs of endpoints that exchange traffic, the feedback is not destabilizing. In a network that has a reasonable number of topologies and paths through the bottleneck link, traffic is shifted away in a stable manner. The assumptions made about equal flows and unique key metrics are important for this particular result, but it is easy to see that unless the flow sizes are extremely skewed,



(a) MATE Performance (from [11])



(b) TeXCP Performance (from [24])



(c) SculpTE Performance

Figure 11: Comparison to MATE and TeXCP using published results.

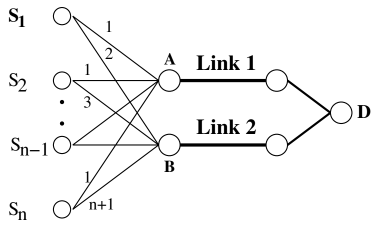


Figure 12: The simple topology used for understanding SculpTE. The link weights for the source links (the numbers on top of each link) are set so that the key metric for each path that passes through the bottleneck links initially are unique.

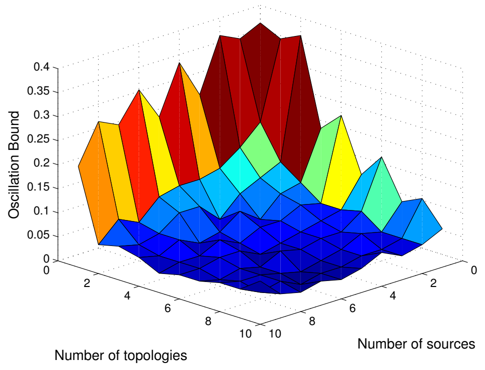


Figure 13: The oscillation bound decreases as the number of topologies and the number of sources increase. Having a few shortest paths through the most loaded link and a few initial topologies is enough to provide tight bounds on stability.

or there are very few SD paths through the bottleneck link (both unlikely in reasonable sized networks), by increasing the number of topologies, the feedback per iteration can be made small enough not to destabilize the system, without neutralizing the potency of SculpTE. This is verified by our simulation results on realistic networks, where we see that with only 3 topologies, SculpTE performs exceptionally well. If certain SD pairs are known to have large flows between them, then it is possible to exclude such paths from SculpTE by methods such as route caching, which we discuss in the next section.

5. Implementation Considerations

In this section, we discuss three concerns related to the implementation of SculpTE: (1) how SculpTE can update topologies in practice, both as part of distributed routing protocols and in the context of centralized approaches that attempt to separate the data and control planes (*e.g.*, RCP [9, 12], 4D [21]); (2) how adjusting the topology can introduce various problems like packet reordering and trigger IGP convergence and how existing technology trends can help mitigate those effects; and (3) how SculpTE might be applied in other network settings, such as data center networks and access networks.

5.1 Realizing SculpTE in Routing Protocols

Because SculpTE's mechanism entails only simple link weight updates, it can be realized in a variety of ways. This flexibility means that SculpTE could be implemented in both existing networks that use standard intradomain routing protocols (*e.g.*, OSPF [36], IS-IS [38]), or in a setting with more centralized control (*e.g.*, 4D [21], OpenFlow [22]). We describe each of these deployment scenarios below. A SculpTE deployment does assume that whatever entity is responsible for computing shortest paths in the topology has ready access to the link weights, as they are updated; we describe how link weight updates might be propagated in each case.

Implementation with existing intradomain routing protocols. SculpTE requires routers to maintain multiple topologies over a single, shared physical infrastructure. Existing routers already provide this function, through multi-topology routing (sometimes called "Multiple Router Configuration", or MRC [40, 41]). Essentially, this function allows routers to run multiple instances of the same routing protocol in parallel on the same network, each with slightly different configurations. SculpTE can apply routers' MRC function to maintain multiple independent topologies and periodically run the procedure from Figure 3 to update them. Routers use an additional bit or set of bits in the IP header, typically in the ToS field, to determine which topology to use to forward a particular packet. SculpTE could use that same function to balance traffic across multiple topologies.

4D-Style implementation. SculpTE could also be deployed in networks that use a logically centralized route controller to install forwarding table entries in the network's routers. As in an implementation with distributed control, the routers must be able to store multiple forwarding table instances, although they need not be able to run multiple versions of the control protocols in parallel, since the centralized controller could compute the outcome of shortest paths for each of SculpTE's topologies.

SculpTE can be implemented either with a Routing Control Platform [9] as the centralized controller, or with OpenFlow switches [37] and one or more NOX controllers [22]. In this setting, the controller could periodically collect information about link loads from the switches, compute shortest paths for the respective topology, and push the resulting paths into the switches in the form of flow-table entries. SculpTE's multiple topologies could be implemented by dividing "flow space": switches can store multiple forwarding table entries to the same destination that match on different header fields (e.g., VLAN identifier).

5.2 Convergence, Reordering, and Loops

A potential concern with SculpTE's is that updating link weights continually triggers IGP convergence. As shown in Figure 9, though, SculpTE may not trigger reconvergence that often in practice: Figure 9, SculpTE performs well, even when iteration intervals are on the order of minutes (perhaps longer than many flows). Additionally, recent advances in IGP convergence have made it possible to trigger IGP convergence across a large transit network in under 200 milliseconds, even for a distributed IGP [18]; other protocol enhancements show how to effectively perform FIB updates to minimize and even eliminate forwarding loops during convergence [17].

Nevertheless, flows that are significantly longer than the update interval could experience re-routing, potentially causing packet reordering. To account for this, SculpTE could either use *route caching*, which allows the routers' forwarding plane to make the same forwarding decisions on existing flows [27]; or *flowlet switching* [43], which uses a small routing table to ensure that packets from the same TCP burst are forwarded along the same path through the network

5.3 Other Deployment Scenarios

Threshold triggered updates. In networks where it is not feasible or required to update link weights continually, SculpTE could be triggered with thresholds so that it operates only when some network event such as a link failure or traffic surge occurs. In these cases, SculpTE can quickly configure the network to adapt to the new conditions and then go back to "sleep".

Data centers. Data center traffic is qualitatively different from ISP and backbone traffic: it is characterized by short-lived bursts [5], as well as long-term surges that are caused MapReduce-style applications. Although prolonged surges are usually known well in advance, setting up alternate paths to account for these surges (*e.g.*, using the MPLS auto-bandwidth feature) is challenging [39]. To handle short bursts, SculpTE can be deployed with iteration durations as long as router synchronization allows it. SculpTE handles pro, as shown in results in Section 4.3.

Access networks. The bottleneck links in some ISPs might be the Provider to Provider-Edge (P-PE) links, due to over-provisioned backbones. In this case, balancing traffic across two or more upstream or downstream links might be more important than balancing traffic in the core. A simple modification to SculpTE provides a solution: We can define *families of links* with only nodes having ownership of a family being able to update links in that family. In such a scenario, a node would compute the L set from its upstream links (the family it owns), but use the entire set of links to compute alternate paths and the key metric, and adjust the weights of the links in such a way that traffic is balanced across them. It can do the same for its downstream links. This is very similar to the two-link problem studied in Section 4.6.1, where we showed that SculpTE achieves very good load balance. This approach is completely distributed: each edge node can update its PE link weights independently and inform the rest of the network of its updates.

6. Conclusion

We have presented the design and evaluation of SculpTE, an online TE mechanism for intradomain traffic engineering that uses self-configuring topologies to achieve traffic load balance in a destination-based, hop-by-hop forwarding network. We believe that this is the first proposal that is essentially configuration-free. In comparison to existing approaches, SculpTE offers the following benefits: (1) it operates without *a priori* configuration thereby reducing management overhead; (2) it performs well under stress conditions, such as traffic surges and link failures; (3) it is stable, and oscillations that do result are well understood, so they can be quantified and mitigated. In our future work, we aim to realize self-configuring topologies in a deployed test network. The simplicity of SculpTE's approach may make it applicable in other settings. For example, congestion, traffic surges, and equipment failures are common in data centers, so applying self-configuring topologies to the design of data-center traffic engineering techniques may be promising.

REFERENCES

[1] GNU Linear Programming Kit. `http://www.gnu.org/software/glpk/`.

[2] Abilene observatory. `http://abilene.internet2.edu/observatory/`.

[3] D. Applegate and E. Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: understanding

fundamental tradeoffs. In *Proc. ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003.

[4] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. *RSVP-TE: Extensions to RSVP for LSP Tunnels*. Internet Engineering Task Force, Dec. 2001. RFC 3209.

[5] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99, 2010.

[6] D. Bertsekas. Dynamic behavior of shortest path routing algorithms for communication networks. *Automatic Control, IEEE Transactions on*, 27(1):60–74, 1982.

[7] S. Bhattacharya, C. Diot, J. Jetcheva, and N. Taft. In *Geographical and temporal characteristics of inter-POP flows: view from a single POP*, volume 13, pages 5–22, 2002.

[8] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. *Resource ReSerVation Protocol (RSVP)*. Internet Engineering Task Force, Sept. 1997. RFC 2205.

[9] M. Caesar, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *Proc. 2nd USENIX NSDI*, Boston, MA, May 2005.

[10] Cisco. Configuring OSPF. `http://www.cisco.com/en/US/docs/ios/12_0/np1/configuration/guide/1cospf.html`.

[11] A. Elwalid, C. Jin, S. Low, and I. Widjaja. Mate: Mpls adaptive traffic engineering. In *in Proc. IEEE INFOCOM 2001*, pages 1300–1309, 2001.

[12] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and K. van der Merwe. The case for separating routing from routers. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Portland, OR, Sept. 2004.

[13] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. *IEEE Communications Magazine*, Oct. 2002.

[14] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proc. IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000.

[15] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE JSAC*, 20(4):756–767, May 2002.

[16] B. Fortz and M. Thorup. Robust optimization of ospf/is-is weights. In *In Proc. International Network Optimization Conference*, pages 225–230, 2003.

[17] P. Francois and O. Bonaventure. Avoiding transient loops during the convergence of link-state routing protocols. *IEEE/ACM Transactions on Networking (TON)*, 15(6):1280–1292, 2007.

[18] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure. Achieving sub-second igp convergence in large ip networks. *ACM Computer Communications Review*, 35:35–44, 2005.

[19] P. Franiçois, M. Shand, and O. Bonaventure. Disruption-free topology reconfiguration in OSPF networks. In *IEEE Infocom*, 2007.

[20] R. Gallager. A minimum delay routing algorithm using distributed computation. *Communications, IEEE Transactions on*, pages 73–85, 1977.

[21] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. *ACM Computer Communications Review*, 35(5):41–54, 2005.

[22] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, July 2008.

[23] J. He, M. Suchara, J. Rexford, and M. Chiang. Rethinking internet traffic management: From multiple decompositions to a practical protocol. In *Proc. CoNEXT*, dec 2007.

[24] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: Responsive yet stable traffic engineering. In *Proc. ACM SIGCOMM*, Philadelphia, PA, Aug. 2005.

[25] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[26] A. Khanna and J. Zinky. The Revised ARPANET Routing Metric. In *Proc. ACM SIGCOMM*, pages 45–56, Austin, TX, Sept. 1989.

[27] C. Kim, M. Caesar, A. Gerber, and J. Rexford. Revisiting route caching: The world should be flat. In *Proc. of PAM*. Springer, 2009.

[28] A. Kvalbein, C. Dovrolis, and C. Muthu. Multipath load-adaptive routing: Putting the emphasis on robustness and simplicity. In *IEEE International Conference on Network Protocols*, 2009.

[29] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne. Fast IP Network Recovery using Multiple Routing Configurations. In *Proc. IEEE INFOCOM*, pages 23–26, Barcelona, Spain, Mar. 2006.

[30] G. Leduc, H. Abrahamsson, S. Balon, S. Bessler, M. D'Arienzo, O. Delcourt, J. Domingo-Pascual, S. Cerav-Erbas, I. Gojmerac, X. Masip, A. Pescaph, B. Quoitin, S. Romano, E. Salvatori, F. Skiv, H. Tran, S. Uhlig, and H. mit. An open source traffic engineering toolbox. *Computer Communications*, 29(5):593–610, March 2006.

[31] J. McQuillan, G. Falk, and I. Richer. A review of the development and performance of the ARPANET routing algorithm. *Communications, IEEE Transactions on*, 26(12):1802–1811, December 1978.

[32] J. McQuillan, I. Richer, and E. Rosen. The new routing algorithm for the ARPANET. *Communications, IEEE Transactions on*, 28(5):711–719, May 1980.

[33] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: Existing techniques and new directions. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[34] D. Mitra and K. Ramakrishnan. A case study of multiservice, multipriority traffic engineeringdesign for data networks. In *GLOBECOM*, 1999.

[35] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.

[36] J. Moy. *OSPF Version 2*, Mar. 1994. RFC 1583.

[37] OpenFlow Switch Consortium. `http://www.openflowswitch.org/`, 2008.

[38] D. Oran. *OSI IS-IS intra-domain routing protocol*. Internet Engineering Task Force, Feb. 1990. RFC 1142.

[39] A. Premji. Using MPLS auto-bandwidth in MPLS networks. `http://www.juniper.net/solutions/literature/app_note/350080.pdf`, 2005. Application Note.

[40] T. Przygienda, Z. Sagl, S. N., and N. Sheth. *M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)*. Internet Engineering Task Force, Feb. 2008. RFC 5120.

[41] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault. *Multi-Topology Routing in OSPF*. Internet Engineering Task Force, June 2007. RFC 4915.

[42] 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 *in ACM SIGCOMM Internet Measurement Workshop*, pages 91–92. ACM Press, 2002.

[43] S. Sinha, S. Kandula, and D. Katabi. Harnessing TCPs Burstiness using Flowlet Switching. In *3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, San Diego, CA, November 2004.

[44] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[45] H. Wang and M. Ito. Dynamics of load-sensitive adaptive routing. In *IEEE International Conference on Communications*, pages 213–217, 2005.

[46] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg. COPE: traffic engineering in dynamic networks. In *SIGCOMM*, 2006.

[47] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg. COPE: Traffic engineering in dynamic networks. *SIGCOMM Comput. Commun. Rev.*, 36(4):99–110, 2006.