Semi-Oblivious Traffic Engineering: The Road Not Taken

Praveen Kumar *Cornell*
Yang Yuan *Cornell*
Chris Yu *CMU*
Nate Foster *Cornell*
Robert Kleinberg *Cornell*
Petr Lapukhov *Facebook*
Chiun Lin Lim *Facebook*
Robert Soulé *Università della Svizzera italiana*

Abstract

Networks are expected to provide reliable performance under a wide range of operating conditions, but existing traffic engineering (TE) solutions optimize for performance or robustness, but not both. A key factor that impacts the quality of a TE system is the set of paths used to carry traffic. Some systems rely on shortest paths, which leads to excessive congestion in topologies with bottleneck links, while others use paths that minimize congestion, which are brittle and prone to failure. This paper presents a system that uses a set of paths computed using Räcke's *oblivious routing* algorithm, as well as a centralized controller to dynamically adapt sending rates. Although oblivious routing and centralized TE have been studied previously in isolation, their combination is novel and powerful. We built a software framework to model TE solutions and conducted extensive experiments across a large number of topologies and scenarios, including the production backbone of a large content provider and an ISP. Our results show that semi-oblivious routing provides near-optimal performance and is far more robust than state-of-the-art systems.

1 Introduction

Two roads diverged in a wood, and I –
I took the one less traveled by,
And that has made all the difference.

—Robert Frost

Networks are expected to provide good performance even in the presence of unexpected traffic shifts and outright failures. But while there is extensive literature on how to best route traffic through a network while optimizing for objectives such as minimizing congestion [3, 9, 13, 14, 15, 22, 24, 26, 47], current traffic engineering (TE) solutions can perform poorly when operating conditions diverge from the expected [32, 42].

The tension between performance and reliability is not merely a hypothetical concern. Leading technology companies such as Google [18,24] and Microsoft [22,32] have identified these properties as critical issues for their private networks. For example, a central goal of Google's B4 system is to drive link utilization to 100%, but doing this means that packet loss is "inevitable" when failures occur [24]. Meanwhile a different study of availability at Google identified "no more than a few minutes of downtime per month" as a goal, where downtime is defined as packet loss above 0.1%-2% [18].

Stepping back, one can see that there are two fundamental choices in the design of any TE system: *(i)* which forwarding paths to use to carry traffic from sources to destinations, and *(ii)* which *sending rates* to use to balance incoming traffic flows among those paths. Any TE solution can be viewed in terms of these choices, but there are also practical considerations that limit the kinds of systems that can be deployed. For example, setting up and tearing down end-to-end forwarding paths is a relatively slow operation, especially in wide-area networks, which imposes a fundamental lower bound on how quickly the network can react to dynamic changes by modifying the set of forwarding paths [25]. On the other hand, modifying the sending rates for an existing set of forwarding paths is a relatively inexpensive operation that can be implemented almost instantaneously on modern switches [32]. Another important consideration is the size of the forwarding tables required to implement a TE solution, as there are limits to how many paths can be installed on each switch [7, 22, 42].

These considerations suggest that a key factor for achieving reliable performance is to select a small set of diverse forwarding paths that are able to route a range of demands under a variety of failure scenarios. Unfortunately, existing TE solutions fail to meet this challenge. For example, using k-shortest paths works well in simple settings but leads to excessive congestion in topologies with shortcut links, which become bottlenecks. Using k-edge-disjoint paths between pairs of nodes does not fare much better, since paths between different node pairs still contend for bandwidth on bottleneck links [24,32]. Using

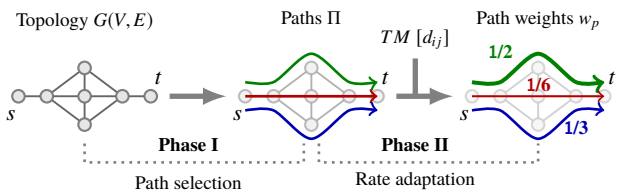


Figure 1: Semi-oblivious TE system model

a constraint solver to compute forwarding paths that optimize for given scenarios and objectives effectively avoids bottlenecks, but it can also "overfit" to specific scenarios, yielding a brittle and error-prone solution. In addition, it is difficult to impose a budget on the number of paths used by the solver, and common heuristics for pruning the set of paths degrade performance [42].

Our approach. We present SMORE, a new TE system based on two key ingredients. First, it uses a set of forwarding paths computed using *oblivious routing*[1] [38,39], rather than shortest, edge-disjoint, or optimal paths, as in current approaches [22, 24, 32]. The paths computed by oblivious routing enjoy three important properties: they are low-stretch, diverse, and naturally balance load. Second, it dynamically adapts sending rates [9,22,26,28,47] on those paths to fit current demands and react to failures. While these ideas have been explored in isolation previously [3, 7, 20], their combination turns out to be surprisingly powerful, and allows SMORE to achieve near-optimal performance. Our work is the first practical implementation and comprehensive evaluation of this combined approach, called *semi-oblivious routing*. Through extensive experiments with data from the production networks of a large content provider (anonymized as BigNet) and a major ISP, as well as large-scale simulations, we demonstrate that SMORE achieves performance that is near-optimal, competitive with state-of-the-art solutions [22, 24], and better than the worst-case scenarios predicted in the literature. SMORE also achieves a level of robustness that improves on solutions explicitly designed to be fault tolerant [32, 42].

Contributions. Our contributions are as follows:

1. We identify a general model for TE systems, and survey various approaches to wide-area TE (§2).
2. We present SMORE's design and discuss key properties that affect performance and robustness (§3).
3. We demonstrate the deployability of SMORE on a production network, and develop a framework for modeling and evaluating TE systems (§4).
4. We conduct an extensive evaluation comparing SMORE with other systems in a variety of scenarios (§5-§6).

Overall, SMORE is a promising approach to TE that is based on solid theoretical foundations and offers attractive performance and robustness.

[1]We use the term *oblivious routing* to refer to Räcke's algorithm, and not other demand-oblivious approaches.

Algorithm	*Load balanced*		*Diverse*	*Low-stretch*
	Capacity aware	*Globally optimized*		
SPF / ECMP	×	×	×	✓
CSPF	✓	×	×	✓
k-shortest paths (KSP)	×	×	?	✓
Edge-disjoint KSP	×	×	✓	✓
MCF	✓	✓	×	×
Robust MCF [42]	✓	✓	✓	×
VLB [45]	×	×	✓	×
B4 [24]	✓	✓	?	?
SMORE / Oblivious [39]	✓	✓	✓	✓

? : difficult to generalize without considering topology and/or demands.

Table 1: Properties of paths selected by different algorithms.

2 System Model and Related Work

This section develops a general model that captures the essential behavior of TE systems and briefly surveys related work in the area.

Abstractly, a TE system can be characterized in terms of two fundamental choices: which forwarding paths to use to carry traffic, and how to spread traffic over those paths. This is captured in the two phases of the model shown in Fig. 1: *(i) path selection* and *(ii) rate adaption*. The first phase maps the network topology to a set of forwarding paths connecting each pair of nodes. Typically this phase is executed only infrequently—e.g., when the topology changes—since updating end-to-end forwarding paths is a relatively slow operation. In fact, in a wide-area network it can take as long as several minutes to update end-to-end paths due to the time required to update switch TCAMs on multiple geo-distributed devices. In the second phase, the system takes information about current demands and failures, and generates a weighted set of paths that describe how incoming flows should be mapped onto those paths. Because updating path weights is a relatively fast operation, this phase can be executed continuously as conditions evolve. For example, the system might update weights to rebalance load when demands change, or set the weight of some paths to zero when a link breaks. The main challenge studied in this paper is how to design a TE system that selects a small set of paths in the first phase that is able to flexibly handle many different scenarios in the second phase.

2.1 Path Properties

The central thesis of this paper is that path selection has a large impact on the performance and robustness of TE systems. Even for systems that incorporate a dynamic rate adaption phase to optimize for specific performance

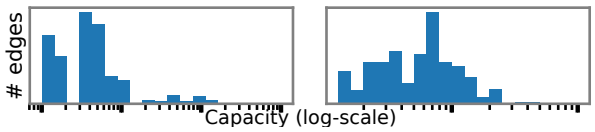


Figure 2: Link capacities in two production WANs.

objectives, the set of paths selected is crucial as it defines the state space for optimization. Desirable path properties include:

A. low stretch for minimizing latency,
B. high diversity for ensuring robustness, and
C. good load balancing for achieving performance.

Unfortunately, current TE systems fail to guarantee at least one of these properties, as shown in Table 1. For example, approaches based on k-shortest paths (KSP) fail to provide good load balancing properties in many topologies due of two fundamental reasons. First, KSP is not capacity-aware. Note that wide-area topologies evolve over time to meet growing demands [18], leading to heterogeneous link capacities as shown in Fig. 2. As KSP does not consider capacities, it often over-utilizes low-capacity links that lie on many shortest paths. Using inverse of capacity as link weight is a common technique to handle this, but it can lead to increased latency due to capacity heterogeneity. Second, because KSP computes paths between each pair of nodes independently, it does not consider whether any given link is already heavily used by other pairs of nodes. Hence, lacking this notion of globally optimized path selection, even if one shifts to using seemingly more diverse edge-disjoint k-shortest paths, the union of paths for all node pairs may still over-utilize bottleneck links.

In general, to achieve low stretch and good load balancing properties, a path selection algorithm must be *capacity aware* and *globally optimized*. To illustrate, consider the topology in Fig. 3 where unit flows arrive in the order f_1, f_2, and f_3. In Fig. 3a, we use the shortest paths to route, as in KSP, and thus link (G, E) becomes congested. In Fig. 3b, we greedily assign the shortest path with sufficient capacity to each flow in order of arrival, as in CSPF, which leads to a locally optimal but globally suboptimal set of paths since some paths have high latency. Finally, in Fig. 3c we depict the globally optimal set of paths. The challenge is to compute a set of paths that closely approximates the performance of these optimal paths while remaining feasible to implement in practice.

2.2 Related Work

The textbook approach to TE merges the two phases in our model and frames it as a combinatorial optimization problem: given a capacitated network and a set of demands for flow between nodes, find an assignment of flows to paths that optimizes for some criterion, such as

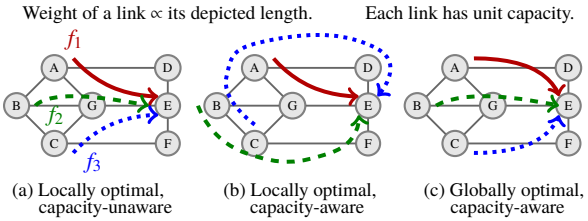


Figure 3: Local vs. globally optimal path selection

minimizing the *maximum link utilization* (MLU). This is known as the *multi-commodity flow* (MCF) problem in the literature, and has been extensively studied. If flows are restricted to use a single path between node pairs, then the problem is NP-complete. But if fractional flows are allowed, then optimal solutions can be found in strongly polynomial time using linear programming (LP) [44].

Another approach, which has been widely used in practice, is to tune link weights in distributed routing protocols, such as OSPF and ECMP, so they compute a good set of forwarding paths [14, 15], and not perform any rate adaptation. This approach is simple to implement as it harnesses the capabilities of widely-deployed conventional protocols, but optimizing link weights for ECMP is NP-hard. Moreover, it often performs poorly when failures occur, or during periods of re-convergence after link weights are modified to reflect new demands. COYOTE [8] aims to improve performance of such distributed approaches by carefully manipulating the view of each switch via fake protocol messages.

Several recent centralized TE systems explicitly decouple the phases of path selection and rate adaption. SWAN [22] distributes flow across a subset of k-shortest paths, using an LP formulation that reserves a small amount of "scratch capacity" for configuration updates. The system proposed by Suchara et al. [42] (henceforth referred as "R-MCF") performs a robust optimization that incorporates failures into the LP formulation to compute a diverse set of paths offline. It then uses a simple local scheme to dynamically adapt sending rates at each source. Recent work by Chang et al. [6] also used robust optimization to validate designs that provide performance and robustness guarantees that are better than worst-case bounds. FFC [32] recommends (p, q) link-switch disjoint paths and spreads traffic based on an LP to ensure resilience to up to k arbitrary failures. B4 [24] selects paths greedily based on demands. It uses BwE [28] and heuristic optimizations to divide flows onto paths to improve utilization while ensuring fairness.

Another line of work has explored the space of oblivious approaches that provide strong guarantees in the presence of arbitrary demands [2, 3, 7]. Valiant Load Balancing (VLB) routes traffic via randomly selected intermediate nodes. Originally proposed as a way to balance load

in parallel computers [45], VLB has recently been applied in a number of other settings including WANs [48]. However, the use of intermediate nodes increases path length, which can dramatically increase latency—e.g., consider routing traffic from New York to Seattle via Paris.

Oblivious routing, which generalizes VLB, computes a probability distribution on low-stretch paths and forwards traffic according to that distribution no matter what demands occur when deployed—in other words, it is *oblivious* to the demands. Remarkably, there exist oblivious routing schemes whose congestion ratio is never worse than $O(\log n)$ factor of optimal. One such scheme, proposed in a breakthrough paper by Räcke [38], constructs a set of tree-structured overlays and then uses these overlays to construct random forwarding paths. While the $O(\log n)$ congestion ratio for oblivious routing is surprisingly strong for a worst-case guarantee, it still requires overprovisioning capacities by a significant amount. Applegate and Cohen [3] developed an LP formulation of optimal oblivious routing. They showed that in contrast to the $O(\log n)$ overprovisioning suggested by Räcke's result, in most cases, it is sufficient to overprovision the capacity of each edge by a factor of 2 or less. While better than the worst-case bounds, it is still not competitive with the state-of-the-art. This lead us to explore augmenting oblivious routing for path selection with dynamic rate adaption in order to achieve better performance.

3 SMORE Design

Our design for SMORE follows the two-phase system model introduced in the preceding section: we use oblivious routing (§3.1) to select forwarding paths, and we use a constraint optimizer (§3.2) to continuously adapt the sending rates on those paths. This approach ensures that the paths used in SMORE enjoy the properties discussed in §2 by construction—i.e., they are low-stretch, diverse, and load balanced.

Performing a robust, multi-objective optimization to compute paths based on anticipated demands is challenging in practice in the presence of resource constraints [6, 32]. Moreover, if the actual conditions differ from what was predicted—e.g., due to failures, or in an ISP where customers may behave in ways that are difficult to anticipate—performance will suffer in general [3]. In contrast, because the paths in oblivious routing are computed without knowledge of the demands, they avoid overfitting to any specific scenario, which makes the system naturally robust. Finally, SMORE comes pre-equipped with a simple mechanism for imposing a budget on the total number of paths used, which allows it to degrade gracefully in the presence of resource constraints, unlike many other approaches.

3.1 Path selection

The core of SMORE's oblivious path selection is based on a structure we call a *routing tree* that implicitly defines a unique path for every node pair in the network $G(V, E)$. A routing tree comprises: *(i)* a logical tree $T(V_t, E_t)$ whose leaves correspond to nodes of G, i.e., there is a one-to-one mapping $m'_V : V \to V_t$, and *(ii)* a mapping $m_E : E_t \to E^*$ that assigns to each edge e_T of T a corresponding path in G, such that edges sharing a common endpoint in T are mapped to paths sharing a common endpoint in G. One can obtain a path $\text{Path}_T(u, v)$ from u to v in G by finding the corresponding leaves $m'_V(u)$ and $m'_V(v)$ of T, identifying the edges of the unique path in T that joins these two leaves, and concatenating the corresponding physical paths based on m_E in G. Generalizing this idea, a *randomized routing tree* (RRT) is a probability distribution over routing trees. The corresponding oblivious routing scheme computes a u–v path by first sampling a routing tree T, then selecting the path $\text{Path}_T(u, v)$. One way to think of oblivious routing is as a hierarchical generalization of VLB, where the network is recursively partitioned into progressively smaller subsets, and one routes from u to v by finding the smallest subset in the hierarchy that contains them both, and constructing a path through a sequence of random intermediate destinations within this subset. Räcke's breakthrough discovery [39] was an efficient, iterative algorithm for constructing RRTs.

We illustrate how the set of paths selected by SMORE have the required properties, in contrast to other well known path selection algorithms, such as ECMP, KSP, edge-disjoint k-shortest paths (EDKSP), VLB and MCF using a representative WAN topology (Hibernia Atlantic).[2] Fig. 4 shows the paths selected by various algorithms for all node pairs and uses a color-coding to indicate load (i.e., the sum of weights of paths using each link). The inset images show the latencies of paths selected by each algorithm for different node pairs.

A. *SMORE's paths have low stretch.* The central ingredient in Räcke's construction of RRTs is a reduction from oblivious routing to the problem of computing *low-stretch routing trees*, defined as follows. The input is an undirected graph G whose edges are assigned positive *lengths* $\ell(e)$ and *weights* $w(e)$. The length of a path P, $\ell(P)$, is defined to be the sum of its edge lengths, and the average stretch of a routing tree T is defined to be the ratio of weighted sums

$$\text{stretch}(T) = \frac{\sum_{e=(u,v)} w(e)\ell(\text{Path}_T(u, v))}{\sum_{e=(u,v)} w(e)\ell(e)},$$

where both sums range over all edges of G. The problem is to select T so as to minimize this quantity, which can be interpreted as the (weighted) average amount by which we

[2] From the Internet Topology Zoo (ITZ) dataset [23]

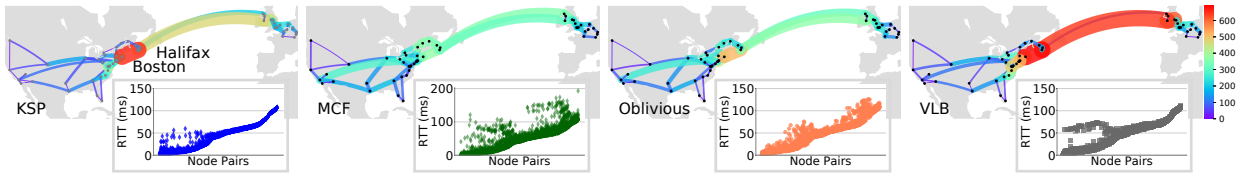


Figure 4: Figure showing sum of path weights for each link, for different path-selection algorithms. Inset shows length of paths selected by each algorithm for different node pairs. x-axis has node pairs sorted by their geographical distances.

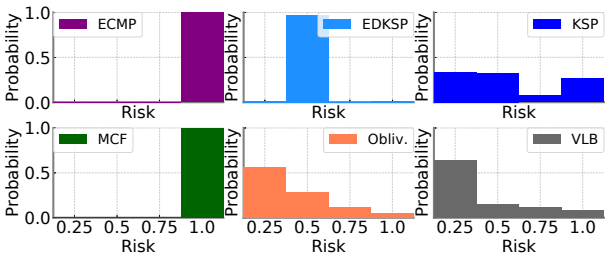


Figure 5: Distribution of *risk* (r_{euv}) for Hibernia Atlantic.

inflate the length of an edge e when we join its endpoints using the path determined by T.

For every instance of the low-stretch routing tree problem, there exists a solution with average stretch $O(\log n)$, and a randomized algorithm, due to Fakcharoenphol, Rao, and Talwar (FRT) [10], efficiently finds such a tree. The algorithm works by computing all-pairs shortest paths in G to define a metric space, then hierarchically decomposing this metric space into clusters of geometrically decreasing diameter, each with a distinguished vertex called the *cluster center*. The topology of the routing tree is defined to be the Hasse diagram of this hierarchical decomposition ordered by inclusion, and the paths associated to its edges are shortest paths between the corresponding cluster centers. Thus, to route from a source u to a destination v, one constructs a path from u by bubbling up through the hierarchy, taking shortest paths to centers of increasingly large clusters, until one reaches the center of a cluster containing both u and v; let's call this center the least common ancestor ($LCA_{u,v}$); then one reverses this process to route from that cluster center to v. If both u and v belong to a cluster C, then the length of the path thus constructed is bounded by a constant times the diameter of C. This explains why paths tend to avoid the lengthy detours that can plague VLB, especially when the source and destination are near one another as shown in insets in Fig. 4. In practice, oblivious routing is often competitive with shortest-path based approaches in terms of latency. Also note that while MCF optimizes for congestion, it may pick long detours to avoid bottlenecks.

B. *SMORE uses diverse paths for robustness.* VLB achieves robustness by routing through random intermediaries, which avoids treating any particular link as critical. Oblivious routing generalizes VLB by allowing for a hierarchy of random intermediate destinations rather than just one. A u–v path is constructed by concatenating paths through a sequence of intermediate destinations representing their ancestors in the sampled routing tree T, up to and including $LCA_{u,v}$. A well-chosen RRT will have the property that the detour through $LCA_{u,v}$ rarely consumes much more capacity than directly taking a shortest path. This allows routing with RRTs to attain aggregate utilization that is nearly as efficient as shortest-path routing, worst-case load balancing that matches or improves VLB, as well as good robustness properties.

One can quantify robustness by generalizing the concept of a SRLG[3] and grouping u–v paths, $\Pi(u,v)$, by the edges they share, such that an edge failure can break all the paths in the shared risk group. We define *risk*, r_{euv}, of an edge e with respect to a node pair (u,v) as the fraction of $\Pi(u,v)$ paths using e. If e is not used by (u,v), then r_{euv} is undefined. For highest resilience, $\Pi(u,v)$ consists of pairwise edge-disjoint paths, and for any edge e, $r_{euv} \leq \frac{1}{|\Pi(u,v)|}$. A fragile $\Pi(u,v)$ has paths sharing some common edge e', and $r_{e'uv} = 1$. Thus, low risk implies high resilience to faults, and low impact on congestion when reacting to failures as more paths are available to share the load. A robust set of paths will have less high risk edges. Fig. 5 shows the distribution of risk when using up to 4 paths per node pair. Ideally, EDKSP should have the entire mass at 0.25. But, a closer look at the topology reveals that for most node pairs, only two edge-disjoint paths exist, implying risk of 0.5 for all edges in those paths, as illustrated in Fig. 6. KSP always finds 4 (u,v) paths which differ slightly and these differing edges have low risk, but the significant number of overlapping edges in these paths have high risk. Interestingly, the set of paths computed using MCF also tend to be brittle. On this topology, both oblivious routing and VLB compute diverse sets of paths, and thus are robust to failures.

C. *SMORE's paths are optimized for load-balancing.* SMORE's path selection algorithm is a capacity-aware iterative algorithm that constructs a sequence of instances of the low-stretch routing tree problem, with the same graph topology but varying edge lengths, and solves each in-

[3]Shared Risk Link Groups (SRLGs) usually refer to links sharing a common physical resource. If one link fails because of the shared resource, other links in the group may fail too.

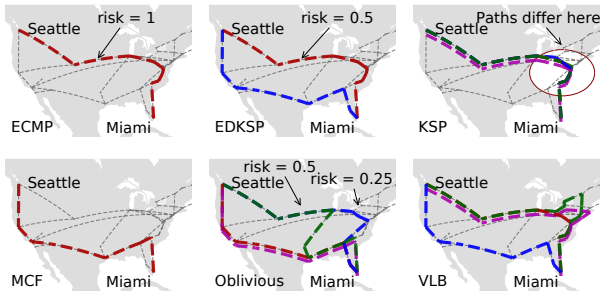


Figure 6: Paths from Seattle to Miami in the Hibernia Atlantic topology used by various TE schemes.

stance using FRT. In a given iteration, the root is selected randomly and the length of each edge is multiplied by an exponential function of the "cumulative usage", relative to capacity, of that edge in previously computed routing trees[4]. The tree computed in each iteration is thus penalized for re-using edges that have been heavily utilized in previous trees, and consequently, the ensemble of all routing trees in the sequence (with suitable probabilities) balances load among edges in a way that ensures $O(\log n)$ congestion ratio regardless of the traffic matrix (TM).

To illustrate this, consider the nodes Boston and Halifax in Fig. 4. There is a direct "shortcut" link connecting these nodes, as well as a slightly longer path to the north. Shortest path based algorithms overload the shorter link and ignore the detour, while MCF balances load equally on the two paths. Oblivious routing distributes load unequally, preferring the direct link but reducing its load by using the detour for a fraction of traffic.

We note another interesting observation based on the Seattle-Miami paths depicted in Fig. 6. Paths selected by KSP are identical for most part, and the only variation occurs at nodes in close proximity within the US northeast. In our experience, this phenomenon occurs often, and failures of such shared edges can adversely affect performance. In contrast, oblivious routing and VLB select a more diverse set of paths which are edge-disjoint with higher probability. Another intuitive way to look at this is that most path selection algorithms compute paths greedily for individual node pairs while oblivious routing globally optimizes paths considering all pairs simultaneously, like MCF. For instance, even though EDKSP computes diverse paths for individual pairs, when paths for all pairs are considered together, the shortcut links can become overloaded, as they may be used by many pairs.

[4]This is an instance of the *multiplicative weights update method* [4], a general iterative method for solving programs such as packing and covering LPs. The method has a tunable parameter ϵ which governs the trade-off between the approximation accuracy and the number of iterations required. Our implementation uses $\epsilon = 0.1$.

	Variable	*Definition*
Input	G(V, E)	Input graph
	Π	The base set of paths allowed in G
	$\mathbf{D}$	Predicted traffic matrix
Auxiliary	$\Pi(s,t)$	The set of all s to t paths in Π
	$d_{(s,t)}$	Demand from s to t specified by $\mathbf{D}$
	$\mathsf{cap}(e)$	Capacity of link e
	U_e	Expected utilization of link e
	Z	Expected maximum link utilization
	$ep(P)$	End-points of path P
Output	w_P	Weight of path P. ($w_P \in [0,1]$)

$$
\begin{aligned}
&\textit{minimize} \quad Z \\
&s.t.: \quad \forall s,t \in V: \sum_{P \in \Pi(s,t)} w_P = 1 \\
&\qquad \forall e \in E: \quad U_e \leq Z \\
&\qquad\qquad U_e = \sum_{P \in \Pi : e \in P} \frac{w_P \cdot d_{ep(P)}}{\mathsf{cap}(e)} \\
&\qquad \forall P \in \Pi: \; w_P \geq 0
\end{aligned}
$$

Table 2: Smore LP formulation for rate adaptation.

3.2 Rate adaptation

These observations on properties of paths selected by oblivious routing motivate using a static set of paths while dynamically adjusting the distribution of traffic over those paths as the demand varies and/or network elements fail and recover. This combination of a static set of paths and time-varying adaptation of flow rates on those paths has been called *semi-oblivious routing* [20]. From a worst-case standpoint, this approach is not significantly better than oblivious routing. Hajiaghayi et al. [20] proved that any semi-oblivious routing scheme that uses polynomially many forwarding paths must suffer a congestion ratio of $\Omega(\log n / \log(\log(n)))$ in the worst case. However, the proof of the lower bound involves constructing highly unnatural TMs and topologies such as recursive series-parallel graphs and grids satisfying specific properties. In contrast, WAN topologies grow in a planned manner, and capacities are augmented based on forecasted demands. Hence, *real-world topologies and TMs are implicitly correlated*. This raises the question of whether it is possible for semi-oblivious routing schemes to approach or match the performance of optimal MCF in practice.

In Smore, we select the static set of paths, Π, using Räcke's algorithm to obtain a distribution over routing trees, taking the union of the path sets defined by each routing tree in the support of this distribution, pruning this distribution to the paths with the highest weights to respect path budget constraints, and then re-normalizing. To distribute flow over paths, we solve a variation of MCF using a linear program (LP). This is similar to the

usage of LP in SWAN and FFC, for instance, but with different objective function and constraint set. In SMORE the LP formulation (Table 2) is used to minimize MLU by balancing traffic over the allowed base set of paths. The output variables w_P express the relative weight of paths for each source-destination pair. The constraints ensure that the weights sum to 1 (i.e., all flows are assigned some path) and that capacity constraints are respected.

4 Implementation

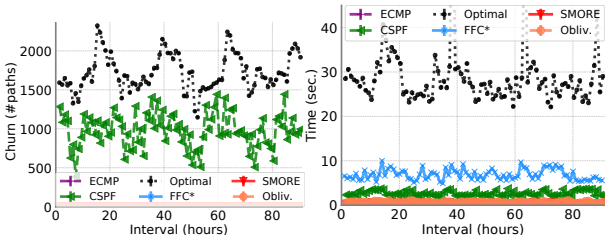
We discuss two implementations to understand and evaluate TE systems. First, we describe a real-world deployment in a production WAN. We highlight a number of practical issues that arose in that deployment (§4.1). Second, we present an implementation using YATES [29], a general framework for rapid prototyping and evaluation of TE approaches (§4.2). We discuss how we calibrated YATES's simulator against a hardware testbed and the production WAN.

4.1 IDN Deployment

To better understand the practical challenges associated with bringing SMORE to production, we deployed a TE system, which dynamically load-balances traffic over a static set of paths, on BigNet's inter-datacenter network (IDN), which is similar to Google's B4 [24] and Facebook's EBB [11].

Architecture. IDN consists of four identical planes (topologies), each of which can be programmed independently. The backbone routers at each datacenter site are connected to an aggregation layer similar to Fat-Cat [41], which distributes outgoing traffic across the planes equally using ECMP. IDN employs a hybrid control model with distributed LSP agents as well as a centralized controller. It supports two traffic classes (high and low priority) which can be managed using different TE algorithms. This architecture facilitates experimenting with different TE algorithms on a subset of planes while the other planes provide a safe fallback.

Controller. The IDN controller allows the routes for each plane to be updated every 15 seconds. The inputs to the controller are obtained from a *state snapshotter* service that captures the live state of the network including: *(i)* configured components for IDN from a central repository for network information [43], *(ii)* live link state information from Open/R [12], *(iii)* any operational overrides (link, router, or plane drains), and *(iv)* real-time TM estimated from sFlow [37] samples exported by routers. When it receives a snapshot, the controller first computes a new set of routes and splitting ratios and then sends instructions to reconfigure the routers as needed.



(a) Path churn per TM **(b)** Re-solver time per TM

Figure 7: Overhead of OPTIMAL TE on BigNet's LBN.

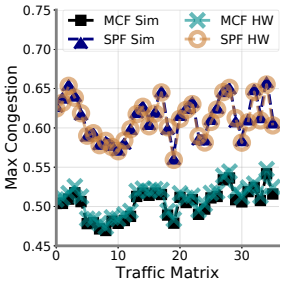
Path budget. The IDN controller maintains an *MPLS LSP mesh*—i.e., a set of LSPs connecting every pair of end points. For operational simplicity and to (indirectly) bound the number of forwarding entries that must be installed on routers, we limit the number of LSPs per node pair to a fixed budget, typically 4.

Traffic splitting. To allow splitting traffic over different LSPs, IDN supports programming up to 64 *next-hop groups* on the ingress router for each pair of nodes. Multiple next-hop groups can map to the same path, and thus we can split traffic among LSPs at granularities of up to $\frac{1}{64}$. Since packets are mapped to next-hop groups (and paths) based on hashing header fields, packets belonging to the same flow take the same path and avoid any issues related to packet reordering in multipath transmission.

Failures. In the event of a failure, traffic is routed along pre-programmed backup paths until the IDN controller computes a new routing scheme. In addition to data-plane faults, failures can also arise due to control-plane errors, such as router misconfiguration or control-plane having an inconsistent view of the network. We proactively test resilience using a *fault-injector* that can introduce both kinds of failures in a controlled manner.

State churn and update time. To quantify the operational overheads of different TE approaches, we measure path churn and the time required to compute an updated routing scheme. Churn is undesirable for several reasons: it increases CPU and memory load on routers and adds significant complexity to the management infrastructure. Fig. 7a shows the number of paths that would be changed every hour when running MCF and CSPF[5]. Likewise, routing schemes that are expensive to compute impose a burden on the controller. Fortunately, the LP that SMORE uses for rate adaptation is less complex than the LP used to solve MCF—the time needed to solve each problem instance is two orders of magnitude less than MCF (order of 100ms vs. 10s), as shown in Fig. 7b, using Gurobi [19] for optimization on a 16-core machine with 2.6 GHz CPU. Furthermore, because SMORE only updates path weights, which takes just a few milliseconds, it is more responsive than MCF, which requires updating whole paths, which can take tens of seconds [24, 32].

[5]We implement a centralized version with 80% link capacities.



PCC (SPF Sim, SPF HW): 0.996

PCC (MCF Sim, MCF HW): 0.997

	SPF	MCF	Production
Ground truth	Testbed	Testbed	Deployment
RMSE	0.006	0.006	0.06

RMSE: Root Mean Squared Error

PCC: Pearson Correlation Coefficient

Figure 8: Calibration: simulator vs. SDN testbed and LBN.

4.2 YATES Framework

We evaluate the performance of a wide range of TE approaches under a variety of workloads and operational scenarios using YATES [29], which consists of a TE simulator and a SDN prototype. Although numerous simulators and emulators have been proposed over the years [5,21,30,34,36,46], YATES is designed specifically to evaluate TE algorithms. With YATES, TE algorithms are implemented as modules against a general interface. Table 3 contains a partial list of TE algorithms that we have implemented and made available under an open-source license[6]. For clarity, we report results from only a subset of these.

Simulator. YATES's simulator can model diverse operational conditions and record detailed statistics. It requires three inputs: *(i)* topology, *(ii)* a timeseries of *actual* TMs to simulate network load, and *(iii)* corresponding *predicted* TMs. For each predicted TM, it computes the routing scheme based on the algorithm (OPTIMAL uses actual TMs) and then simulates the flow of traffic where each source generates traffic based on the actual TM. We choose the *fluid model* [27] to simulate traffic owing to its scalability without sacrificing accuracy of macroscopic behavior. In case the actual TM is unsatisfiable using the routing scheme, YATES still admits the entire demand at each source. However, it assigns each flow its max-min fair share at each oversubscribed link and drops any traffic exceeding the flow's fair share for that link.

SDN prototype. The prototype, which consists of a SDN controller and an end-host agent, allows us to evaluate different TE algorithms using an approach similar to centralized MPLS-TE. The controller manages the forwarding rules installed on OpenFlow-enabled switches, while the end-host agent, implemented as a Linux kernel module, splits flows and assigns them to paths at the source. Although SDN allows us to easily implement the backend, it is not a requirement. It can be easily replaced with other control mechanisms, such as PCEP [31].

Simulator calibration. For simulation results to be credible, it is critical that they accurately correspond to results from real deployments. We validate the accuracy of YATES's simulator with: *(i)* a small-scale hardware testbed, and *(ii)* BigNet's large backbone network (LBN) (§5). We use the SDN backend to emulate Internet2's Abilene backbone network [1] on a testbed of 12 switches and replay traffic based on NetFlow traces collected from the actual Abilene network. As shown in Fig. 8, the simulation results closely match observed results in the hardware testbed. We also implement a centralized approximation of the distributed TE algorithm used in production in LBN. The network and demands are highly dynamic, and the production TE scheme reacts to such changes at a very fine time scale. As a result, we are able to only approximate its behavior. Still, the values reported by YATES closely match those seen in production.

[6] http://github.com/cornell-netlab/yates

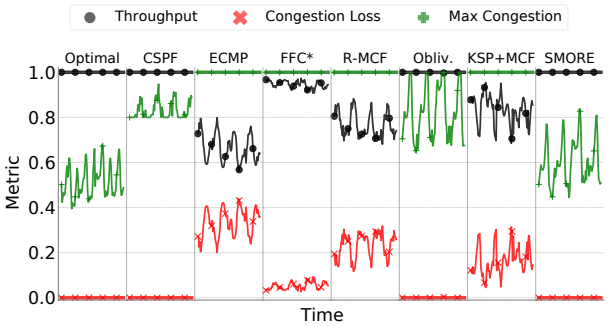
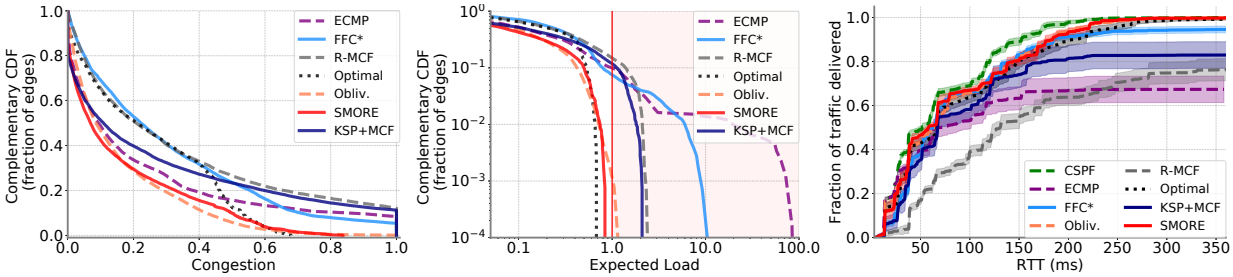


Figure 9: Expected performance on LBN over half a week.

5 BigNet WAN

We evaluate SMORE in a production setting using multiple criteria. For the setting, we use data from BigNet's large backbone network (LBN). LBN is one of the largest global deployments and carries a mix of traffic ranging from real-time video streaming and messaging to massive data synchronization globally. For criteria, we focus on four key questions: How close is SMORE to optimal in terms of performance (§5.1)? What is the impact on latency for not choosing strictly shortest paths (§5.2)? How is performance impacted under failures (§5.3) and other operational constraints (§5.4)? §6 explores whether these results generalize to other settings, using large-scale simulations over a diverse set of network scenarios.

Overview of BigNet's WAN. The network models a common content-provider design, with connections between several large datacenters across Asia, Europe, and the US as well as connectivity to numerous Points-of-Presence (PoPs) around the globe. The topology has hundreds of routers and thousands of high-speed interconnecting links, varying vastly in capacity and latency. This heterogeneity largely stems from the way the net-



(a) CCDF of link congestions. $P(X \geq x) = y$ **(b)** CCDF of expected link loads. **(c)** CDF of latency.

Figure 10: BigNet LBN: predicted distribution of expected load, congestion and latency.

work evolved over many years. The topology exhibits clustered structure, with clusters following geographic constraints imposed by continents and links between clusters running over transoceanic paths, similar to the topology that we used for illustration in §3.

Traffic on LBN exhibits multiple strong diurnal patterns, modulated by the activities of billions of users in different time zones. Being a global system, different parts of the network experience peak loads at different times. Overall traffic patterns over the WAN can be split into two major categories: *(i)* traffic between datacenters, and *(ii)* traffic from datacenters to PoPs. Inter-datacenter traffic typically consists of various replication workflows, such as those related to cache consistency or bulk traffic for Hadoop and database replication. A significant amount of this traffic is delay tolerant, and could be routed over non-shortest paths between the datacenters. However, the traffic from datacenters to PoPs is latency sensitive, as it represents content routed to BigNet users.

Methodology. We collect production data from LBN consisting of accurate network topology, link capacities, link latencies, aggregate site-to-site TMs, and paths used by traffic in production. Using this data, we perform high-fidelity simulations with YATES and present results based on the statistics reported. Traffic with different latency requirements are routed separately in production to avoid excessive path stretch for latency-sensitive traffic. For simplicity, we choose to route both types of traffic using the same TE scheme. Traffic on BigNet's WAN is growing at a rapid pace, and so the network also evolves with it. We present results based on the network state and demands for a month in late 2016.

5.1 Performance

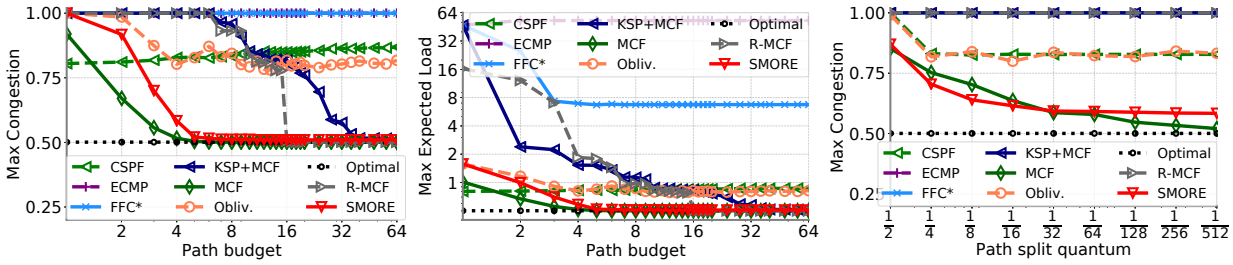
For each hourly snapshot of the network under regular operating conditions (i.e. without any failures), we measure various performance statistics (with a path budget $k = 4$, same as reported in B4 [24]) using different TE approaches. Fig. 9 shows the *(i)* throughput normalized to total demand, *(ii)* maximum congestion (fractional link utilization), and *(iii)* normalized traffic dropped due to congestion over a period of half a week. CSPF and OPTIMAL[7] dynamically compute paths with sufficient capacity for each flow, and thus avoid congestion. Remarkably, only oblivious routing and SMORE are able to achieve 100% throughput, while other centralized TE approaches aren't able to do so and introduce bottlenecks.

As expected, OPTIMAL (which uses MCF to minimize MLU) achieves the lowest maximum congestion, which varies between 0.40 and 0.67 following a diurnal pattern. We find that oblivious routing performance remains within a factor of 2 as had been previously studied [3], while SMORE is closest to optimal with maximum congestion within 16% of OPTIMAL, on average and within 41% in the worst case.

Clearly, SMORE's path selection plays a crucial role in it being so competitive. To gain further insight, we examine the distribution of congestion and expected link utilizations, i.e., how much traffic each link would have carried if packets weren't dropped due to capacity constraints. Figs. 10a and 10b show the corresponding complementary CDFs. We observe that ECMP[8], FFC*[9], R-MCF and KSP+MCF (an approximation of SWAN's path selection and rate adaptation)[10] scheduled ~10% of links to carry traffic exceeding their capacity—ECMP even oversubscribed a link 80×! We find that these bottleneck links usually appear in the shortest paths between many pairs of nodes. In contrast, Räcke's algorithm iteratively samples paths while avoiding overloading any link, and SMORE load balances over these paths to reduce congestion even further. On scaling up the demands, we do expect to see congestion loss with all the approaches, including SMORE. From Fig. 10a, we also note that SMORE maintains a lower congestion consistently for all links and has a 95th-percentile congestion of 0.57—same as OPTIMAL.

Even though FFC*, KSP+MCF and R-MCF dynamically load balance traffic over a set of paths, they perform suboptimally. This could be because the paths selected under the budget constraint did not provide enough flexi-

[7] OPTIMAL does not have any budget or other operational constraints.
[8] Using RTTs as link weights for computing shortest paths.
[9] Our implementation configures FFC to handle single link failures by combining edge-disjoint k-shortest paths with fault-tolerance LP.
[10] We implement a version that uses k-shortest paths as tunnels and uses MCF to assign path weights.



(a) Max congestion vs. path budget **(b)** Max expected load vs. path budget **(c)** Performance with path split quantization

Figure 11: Performance with different operational parameters on BigNet's LBN network.

bility to eliminate bottlenecks for any traffic splitting ratio, and this was further exacerbated as the inadmissible fraction of demands also contributed to congestion before being dropped. To validate this, we measure performance with increasing budget in Figs. 11a and 11b. KSP+MCF and R-MCF, indeed, become near-optimal when the sets of paths become diverse enough. However, FFC* doesn't improve beyond a point as the number of disjoint paths is very limited. Even though the "working set" [22] of paths for KSP+MCF is small, it needs 8× as many total number of paths as SMORE to achieve similar performance.

5.2 Latency

Fig. 10c shows the distribution of latency as a fraction of total demand that is delivered within a given latency[11]. To compute latency experienced by traffic along a path, we simply sum the measured RTTs for each hop along the path. TE approaches which route over shortest paths while respecting capacity constraints, like CSPF, have the least latency. Oblivious routing doesn't ensure that the shortest paths are necessarily selected. However, as we showed in §3.1, the paths computed have low stretch. SMORE uses the same set of low-stretch paths. Intuitively, longer paths can increase congestion as the same set of packets contribute to congestion at more links. As SMORE optimizes for congestion, it also indirectly favors shorter paths. We find that SMORE is competitive with other shortest-path based approaches. Even if we ignore dropped traffic and normalize y−values in Fig. 10c with throughput, the median latency for SMORE (58.3ms) is similar to KSP+MCF (62.7ms). We find that for any node pair, SMORE finds a path with latency within 1.09× the shortest path, on average. Furthermore, if we include factors such as buffering at routers, which depends on congestion, we expect to see better latency for SMORE as it has better congestion guarantees.

5.3 Robustness

There is a trade-off between performance and robustness, and often TE systems that optimize for performance

[11]Assuming dropped traffic has infinite latency, curves reach a maximum y-value equal to throughput.

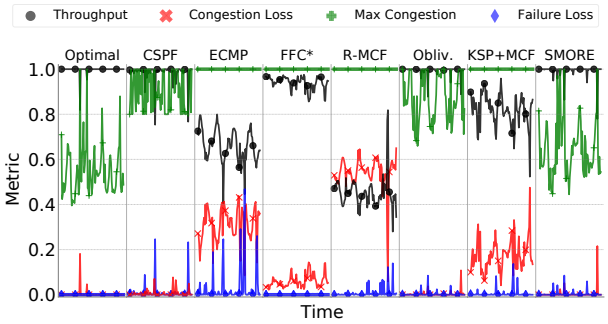


Figure 12: Expected robustness on LBN over half a week.

tend to overfit and become brittle. In Fig. 12, we evaluate the robustness of TE approaches to network failures. Here, we fail a unique link every hour and note the impact on performance. We implement a simple recovery mechanism which re-normalizes path weights to shift traffic from failed paths on to unaffected paths, if available. The recovery method is fast as it does not need setting up new paths, and it decreases loss due to failure at the cost of congestion. Usually, this increases throughput, but there are exceptions as illustrated in §B.1. We see this with R-MCF in Fig. 12. OPTIMAL knows failures in advance and reacts by setting up globally optimal paths instantaneously. FFC* is always able to find backup paths as it uses disjoint paths, and thus avoids loss due to failure. However, these paths are suboptimal for achieving the best throughput as congestion causes packet drops.

SMORE continues to deliver ∼100% throughput. Although maximum congestion increases because of recovery, SMORE remains within 18% of OPTIMAL, on average and within 71% in the worst case. SMORE's high resilience can be attributed to the fact that the paths it uses are diverse and have low risk, as we saw earlier in §3.1. This ensures that, in most cases, SMORE has sufficient options to re-route traffic and load balance efficiently without overloading any link.

5.4 Operational constraints

Various operational constraints need to be accounted for while deploying a TE system. We describe one such constraint—*path-split quantization*. So far, our evalu-

ation has assumed that traffic could be split in arbitrary proportions. This is usually not the case, and path weights are quantized. Most routers support splitting by allowing to specify a certain number, typically up to 64, of next-hop groups [24]. This means that path weights should be multiples of the path-split quantum, $\frac{1}{64}$. Fig. 11c shows the impact of quantization on performance (at path budget of 4). We approximate traffic split ratio generated by different TE schemes to be multiples of the path-split quantum using a greedy approach. SMORE degrades gracefully when quantization becomes restrictive and performs well for practical settings when path-split quantum $\geq \frac{1}{64}$.

6 Large-Scale Simulations

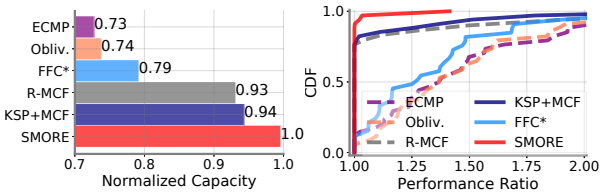
The evaluation on LBN in the preceding section showed that SMORE achieves near-optimal performance and robustness for the topology and workload in a large production network. We also obtained similar results for experiments conducted using data from a major ISP (omitted from paper due to space constraints). In this section, we show that these performance (§6.1) and robustness (§6.2) results generalize to a wider range of scenarios.

Methodology. We evaluate 17 TE algorithms over 262 ISP and inter-DC WAN topologies using YATES. We model a diverse set of operational conditions by varying demands, failures, TM prediction, and path budget. We present a subset of our experimental data that illustrates our main results over the scenarios described next.

Topologies. We select 28 topologies, shown in §C, from ITZ and other real-world networks, to overlap with ones used to evaluate TE approaches in the literature [24, 26].

TM Generation. We use YATES to generate TMs based on the gravity model [40], which assigns a weight w_i to each host i and assumes that $i \rightarrow j$ demand $\propto w_i \cdot w_j$. We sample w_i from a heavy-tailed Pareto distribution obtained by fitting real-world TMs. We model diurnal and weekly variations by randomly perturbing the Fourier coefficients of the observed time-series and temporal variations at a finer scale by using the Metropolis-Hastings algorithm to sample from a Markov chain on the space of TMs, whose stationary distribution is the gravity model with Pareto-distributed weights described above. The algorithm updates w_i at each time step to model gradual variation over time. The following experiments scale TMs such that the minimum possible MLU for the first TM is 0.4, which matches the average MLU observed in traditional overprovisioned WANs [22, 24].

TM Prediction. YATES offers a suite of algorithms to predict future TMs. These include standard ML methods such as linear regression, lasso/ridge regression, logistic regression, random forest prediction etc., as well as algebraic methods like FFT fit and polynomial fit. For each pair of hosts, we perform an independent time series prediction where, at each time step t, the demand is predicted using demands observed in $[t-k, t)$. We optimize the size of the sliding window (k) separately using cross-validation. Regression and random forest models are trained using the previous k time steps as k features. FFT fit finds a function with a bounded number of non-zero Fourier coefficients, while polynomial fit finds a bounded-degree polynomial function that minimizes the absolute difference between predicted and actual TMs over the past k time steps. This best-fit function is evaluated in the current time step to yield the predicted demand. Finally, YATES exposes an error parameter to assess the sensitivity of TE algorithms to inaccuracy in prediction.



(a) Network capacity **(b)** Performance ratio

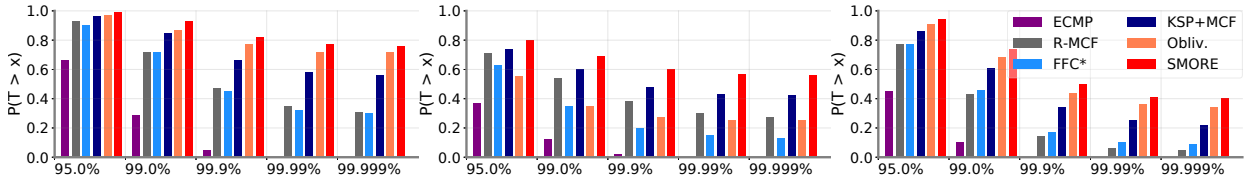
Figure 13: Aggregate performance at base demand.

6.1 Performance

We start by evaluating basic properties of TE approaches including the effective capacity of the network and the performance ratio with respect to OPTIMAL.

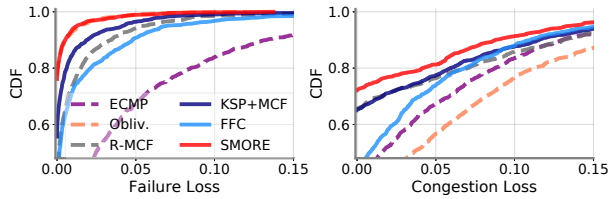
Network capacity. During normal operation, the MLU of a network is usually below 1, meaning there is spare capacity in the network. Given a routing scheme and a TM, we can define *network capacity* as the factor by which the TM can be scaled up before it experiences congestion. This spare capacity could be used to handle unexpected surges in traffic, or to schedule background traffic. As OPTIMAL minimizes MLU, it has the highest possible network capacity. Fig. 13a shows network capacity for different TE approaches, normalized with respect to OPTIMAL. As expected, oblivious routing, which can use up more bandwidth on more links to route the same TM, is unable to admit a significant fraction of TMs that OPTIMAL can handle. We find that *SMORE has the highest network capacity and is near-optimal* owing to efficient load balancing over a diverse set of paths.

Performance ratio. Another way to compare TE approaches is to measure how far they are from OPTIMAL, with respect to minimizing MLU for a given set of TMs. Here, we follow the metric defined by Applegate and Cohen [3], but use throughput after accounting for congestion loss, if any, instead of using the demand TM to compute congestion and performance ratio. Fig. 13b compares the distribution of performance ratio over various topologies and TMs. We find that SMORE and KSP+MCF



(a) base demand, single link failures **(b)** 2× demand, single link failures **(c)** base demand, double link failures

Figure 14: Robustness: Probability of achieving a throughput SLA (x) under different conditions.



(a) Base demand: failure loss **(b)** 2× demand: congestion loss

Figure 15: Robustness: CDF of throughput loss

remain optimal in 75-80% of the cases but *SMORE has closest to optimal performance ratio* (§C.1).

6.2 Robustness

Similar to §5.3, we systematically inject failures in the network to study the trade-off between performance and robustness. Fig. 15 shows the distribution of loss in throughput over all possible single link failures. With base demands, failures are the main cause of loss. As demand scales up, loss due to congestion becomes significant. *SMORE performs better in both cases by being more fault tolerant as well as spreading traffic evenly to avoid congestion*. Although TE approaches like FFC* and R-MCF are designed to be fault tolerant, they do not achieve the best throughput. This is because FFC* relies on disjoint paths to be available to reroute traffic; the number of such paths is limited in real-world topologies. For instance, GÉANT's topology has nodes with degree 1. These nodes have a single edge-disjoint path to any other node; the failure of any edge along this path leads to loss of traffic. Using (p,q) link-switch disjoint paths also doesn't improve the robustness much. Moreover, FFC* is not congestion-optimal by design and incurs high loss due to congestion as demands increase. R-MCF relies on using a large number of paths for fault tolerance and applying a budget deteriorates its performance [42].

Typically, SLAs refer to availability of a network in terms of "nines". This can also be translated in terms of throughput and given a failure characteristic [17, 18, 33], a network operator could be interested in questions such as "what is the probability that throughput is greater than 99.9%?" Fig. 14 compares TE approaches on how likely are they to achieve different levels of SLA under various operational conditions. In addition to the scenario where single link failures can happen with uniform probability under regular load (Fig. 14a), we perform two more experiments where we study robustness under increased load (Fig. 14b), and concurrent failures (Fig. 14c). We find that SMORE consistently outperforms other TE approaches. Oblivious TE is robust under both single and concurrent link failures at base demands, but its resilience deteriorates for increased load. SMORE benefits from the robust set of paths selected by oblivious routing, and also load balances efficiently even during increased load. Thus, *SMORE is highly robust and achieves SLAs with highest probability under diverse operational conditions.*

7 Conclusion

In TE, there is a fundamental trade-off between performance and robustness. Most systems are designed to optimize for one or the other, but few manage to achieve both. This challenge is further exacerbated by operational restrictions such as the number of paths, overhead due to churn, quantized splitting ratio imposed by hardware, etc.

This paper presents SMORE, a new approach that navigates these trade-offs by combining careful path selection with dynamic weight adaptation. As shown through a detailed evaluation on a production backbone network, SMORE achieves near-optimal performance in terms of congestion and load balancing metrics, is competitive with shortest-path based approaches in terms of latency, and is also robust, allowing traffic to be re-routed around failures without introducing bottlenecks while respecting operational constraints. Our large-scale evaluation shows that these performance and robustness guarantees hold across a broader class of networks. More generally, our experiences designing and implementing SMORE suggests lessons that are broadly applicable to TE systems including the importance of capacity-aware and globally optimized selection of low-stretch and diverse paths, as well as the consideration of operational constraints when building a practical TE system.

Acknowledgments. We would like to thank the anonymous NSDI reviewers and our shepherd Srikanth Kandula for their valuable feedback. This work was partially supported by NSF grant CCF-1637532 and ONR grant N00014-15-1-2177. We are grateful to Omar Baldonado and Sandeep Hebbani for their continued support, and we thank Bruce Maggs, Ratul Mahajan, Nick McKeown, Jennifer Rexford, Michael Schapira, Amin Vahdat and Minlan Yu for helpful discussions.

References

[1] Historical Abilene Data. http://noc.net.internet2.edu/i2network/live-network-status/historical-abilene-data.html.

[2] Altin, A., Fortz, B., and Ümit, H. Oblivious OSPF Routing with Weight Optimization under Polyhedral Demand Uncertainty. *Networks 60*, 2 (2012), 132–139.

[3] Applegate, D., and Cohen, E. Making Intra-domain Routing Robust to Changing and Uncertain Traffic Demands: Understanding Fundamental Tradeoffs. In *ACM SIGCOMM* (2003).

[4] Arora, S., Hazan, E., and Kale, S. The Multiplicative Weights Update Method: a Meta-Algorithm and Applications. *Transactions on Computers 8*, 1 (2012), 121–164.

[5] Chang, X. Network Simulations with OPNET. In *31st Conference on Winter Simulation* (1999), ACM.

[6] Chang, Y., Rao, S., and Tawarmalani, M. Robust Validation of Network Designs under Uncertain Demands and Failures. In *USENIX NSDI* (2017).

[7] Chiesa, M., Kindler, G., and Schapira, M. Traffic Engineering with Equal-Cost-MultiPath: An Algorithmic Perspective. *IEEE/ACM Transactions on Networking* (2016).

[8] Chiesa, M., Rétvári, G., and Schapira, M. Lying Your Way to Better Traffic Engineering. In *ACM CoNEXT* (2016).

[9] Elwalid, A., Jin, C., Low, S., and Widjaja, I. MATE: MPLS Adaptive Traffic Engineering. In *IEEE INFOCOM* (2001).

[10] Fakcharoenphol, J., Rao, S., and Talwar, K. A Tight Bound on Approximating Arbitrary Metrics by Tree Metrics. In *35th STOC* (2003), pp. 448–455.

[11] Building Express Backbone: Facebook's new long-haul network. http://code.facebook.com/posts/1782709872057497/building-express-backbone-facebook-s-new-long-haul-network.

[12] Introducing Open/R - a new modular routing platform. http://code.facebook.com/posts/1142111519143652/introducing-open-r-a-new-modular-routing-platform.

[13] Fischer, S., Kammenhuber, N., and Feldmann, A. REPLEX: Dynamic Traffic Engineering Based on Wardrop Routing Policies. In *ACM CoNEXT* (2006).

[14] Fortz, B., Rexford, J., and Thorup, M. Traffic Engineering with Traditional IP Routing Protocols. *IEEE Communications Magazine 40*, 10 (Oct. 2002).

[15] Fortz, B., and Thorup, M. Internet Traffic Engineering by Optimizing OSPF Weights. In *IEEE INFOCOM* (2000), vol. 2.

[16] Garg, N., and Könemann, J. Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems. *SICOMP 37*, 2 (May 2007), 630–652.

[17] Gill, P., Jain, N., and Nagappan, N. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. *ACM SIGCOMM CCR 41*, 4 (2011).

[18] Govindan, R., Minei, I., Kallahalla, M., Koley, B., and Vahdat, A. Evolve or Die: High-Availability Design Principles Drawn from Google's Network Infrastructure. In *ACM SIGCOMM* (2016).

[19] Gurobi Optimizer. http://www.gurobi.com.

[20] Hajiaghayi, M., Kleinberg, R., and Leighton, T. Semi-oblivious Routing: Lower Bounds. In *SODA* (2007), pp. 929–938.

[21] Henderson, T. R., Lacage, M., Riley, G. F., Dowell, C., and Kopena, J. Network Simulations with the ns-3 Simulator. *SIGCOMM demonstration 14* (2008).

[22] Hong, C.-Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M., and Wattenhofer, R. Achieving High Utilization with Software-Driven WAN. In *ACM SIGCOMM* (2013).

[23] The Internet Topology Zoo. http://www.topology-zoo.org.

[24] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., and Vahdat, A. B4: Experience with a Globally Deployed Software Defined WAN. In *ACM SIGCOMM* (2013).

[25] Jin, X., Liu, H., Gandhi, R., Kandula, S., Mahajan, R., Rexford, J., Wattenhofer, R., and Zhang, M. Dionysus: Dynamic Scheduling of Network Updates. In *ACM SIGCOMM* (2014).

[26] Kandula, S., Katabi, D., Davie, B., and Charny, A. Walking the Tightrope: Responsive Yet Stable Traffic Engineering. In *ACM SIGCOMM* (2005).

[27] Kelly, F., and Williams, R. Fluid Model for a Network Operating under a Fair Bandwidth-Sharing Policy. *The Annals of Applied Probability 14*, 3 (2004).

[28] Kumar, A., Jain, S., Naik, U., Raghuraman, A., Kasinadhuni, N., Zermeno, E. C., Gunn, C. S., Ai, J., Carlin, B., Amarandei-Stavila, M., et al. BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing. In *ACM SIGCOMM* (2015).

[29] Kumar, P., Yu, C., Yuan, Y., Foster, N., Kleinberg, R., and Soulé, R. YATES: Rapid Prototyping for Traffic Engineering Systems. In *ACM SOSR* (2018).

[30] Lantz, B., Heller, B., and McKeown, N. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *ACM HotNets* (2010).

[31] Le Roux, J., and Vasseur, J. Path Computation Element (PCE) Communication Protocol (PCEP). https://tools.ietf.org/html/rfc5440, 2009.

[32] Liu, H. H., Kandula, S., Mahajan, R., Zhang, M., and Gelernter, D. Traffic Engineering with Forward Fault Correction. In *ACM SIGCOMM* (2014).

[33] Markopoulou, A., Iannaccone, G., Bhattacharyya, S., Chuah, C.-N., Ganjali, Y., and Diot, C. Characterization of Failures in an Operational IP Backbone Network. *IEEE/ACM Transactions on Networking 16*, 4 (2008), 749–762.

[34] McCanne, S., and Floyd, S. NS network simulator, 1995.

[35] Mitra, D., and Ramakrishnan, K. A Case Study of Multiservice, Multipriority Traffic Engineering Design for Data Networks. In *IEEE GLOBECOM* (1999), vol. 1.

[36] Cisco NetSim Network Simulator. http://www.boson.com/netsim-cisco-network-simulator.

[37] Panchen, S., Phaal, P., and McKee, N. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. https://tools.ietf.org/html/rfc3176, 2001.

[38] Räcke, H. Minimizing Congestion in General Networks. In *43rd FOCS* (2002), pp. 43–52.

[39] Räcke, H. Optimal Hierarchical Decompositions for Congestion Minimization in Networks. In *40th STOC* (2008), pp. 255–264.

[40] Roughan, M., Greenberg, A., Kalmanek, C., Rumsewicz, M., Yates, J., and Zhang, Y. Experience in Measuring Backbone Traffic Variability: Models, Metrics, Measurements and Meaning. In *IMC* (2002), ACM, pp. 91–92.

[41] Roy, A., Zeng, H., Bagga, J., Porter, G., and Snoeren, A. C. Inside the Social Network's (Datacenter) Network. In *ACM SIGCOMM* (2015).

[42] Suchara, M., Xu, D., Doverspike, R., Johnson, D., and Rexford, J. Network Architecture for Joint Failure Recovery and Traffic Engineering. In *SIGMETRICS* (2011), pp. 97–108.

[43] Sung, Y.-W. E., Tie, X., Wong, S. H., and Zeng, H. Robotron: Top-down Network Management at Facebook Scale. In *ACM SIGCOMM* (2016).

[44] Tardos, E. A Strongly Polynomial Algorithm to Solve Combinatorial Linear Programs. *Operations Research 34*, 2 (1986), 250–256.

[45] Valiant, L. A Scheme for Fast Parallel Communication. *SICOMP 11*, 2 (1982), 350–361.

[46] Varga, A., et al. The OMNeT++ Discrete Event Simulation System. In *European Simulation Multiconference* (2001).

[47] Wang, H., Xie, H., Qiu, L., Yang, Y. R., Zhang, Y., and Greenberg, A. COPE: Traffic Engineering in Dynamic Networks. In *ACM SIGCOMM* (2006).

[48] Zhang-Shen, R., and McKeown, N. Designing a Fault-Tolerant Network Using Valiant Load-Balancing. In *IEEE INFOCOM* (Apr. 2008).

A YATES

TE Algorithm	*#LOC*
Shortest Path Routing (SPF)	17
Equal-Cost, Multi-Path (ECMP)	166
Constrained Shortest Path First (CSPF)	287
k-Shortest Paths (KSP)	119
Edge-disjoint k-Shortest Paths (EDKSP)	212
Räcke's oblivious routing [39]	645
Applegate-Cohen's oblivious routing [3]	456
Valiant Load Balancing (VLB) [45]	226
Multi-Commodity Flow (MCF) solved as LP [35]	308
MCF solved with Multiplicative Weights (MW) [16]	194
ECMP for paths, MCF for weights (similar to [13])	474
KSP for paths, MCF for weights (SWAN) [22]	427
EDKSP for paths, MCF for weights	520
Disjoint paths, FFC LP for weights (FFC) [32]	493
Räcke for paths, MCF for weights (SMORE)	953
VLB for paths, MCF for weights	534
MCF with failures for paths, MCF for weights (R-MCF) [42]	390
Omniscient MCF (OPTIMAL)	308

Table 3: List of TE algorithms implemented in YATES.

B Illustrations

B.1 Throughput decrease on recovery

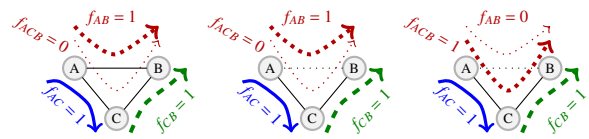
One way to recover from failures is to shift traffic on to unaffected paths by re-normalizing the paths weights while excluding failed paths. This re-normalization based recovery method is fast as it does not need setting up new paths, and it decreases loss due to failure at the cost of congestion. Usually, this increases throughput, but there are exceptions as illustrated in Fig. 16. Initially, there are two $A \rightarrow B$ paths, but only the direct path (A-B) is being used and we achieve a total throughput of 3. Failure of AB link causes all $A \rightarrow B$ traffic to be dropped and this reduces the total throughput to 2. Recovery shifts $A \rightarrow B$ traffic on to path A-C-B. This causes bottleneck at AC and CB, and the fair share for each flow becomes 0.5. This results in the total throughput further reducing to 1.5.

Another fast way to implement recovery while using the unaffected subset of paths is to use a restricted version of MCF that optimizes for congestion to calculate new weights. But similar to the previous case, throughput can decrease in cases when the minimum possible MLU is greater than 1.

C Large scale simulations

Topology	*# nodes(n)*	*# edges (m)*	*Topology*	*# nodes(n)*	*# edges (m)*
Abilene	12	15	Airtel	16	26
ATT	25	56	BTNorthAmerica	36	76
CESNET	44	51	CRLNetwork	33	38
CWIX	36	41	Darkstrand	28	31
Digex	31	35	GÉANT	40	61
GRnet	37	42	Google (B4)	12	19
Highwinds	18	31	IBM	18	21
IIJ	37	65	Integra	27	36
Intellifiber	73	95	InternetMCI	19	33
Janet Backbone	32	45	NTT	32	63
PacketExchange	21	27	Palmetto	45	64
Quest	20	31	Sprint	11	18
Tinet	53	89	UUNET	49	84
Xeex	24	34	Xspedius	34	48

Table 4: Topologies used in evaluation.



a) Base, $tput = 3$ b) AB fail, $tput = 2$ c) Recovery $tput = 1.5$

Figure 16: Normalizing path weights to handle failures may introduce bottlenecks and decrease throughput.

C.1 Performance

Performance ratio. Following Applegate and Cohen, we define *performance ratio* of a routing scheme r to measure how far is it from optimal with respect to minimizing congestion for a given TM and topology [3]. We modify it slightly to use actual throughput after accounting for congestion loss, if any, instead of the demand TM to compute congestion. For a TM $\mathbf{D}$, suppose $\mathbf{D}^r$ is the throughput matrix, and d^r_{uv} is the $u \rightarrow v$ throughput using r. The maximum congestion for TM $\mathbf{D}$ with scheme r is:

$$\max_{e \in E} \frac{\sum_{u,v} d^r_{uv}(e)}{\texttt{cap}(e)}$$

where $d^r_{uv}(e)$ denotes the amount of $u{-}v$ traffic routed through link e, and $\texttt{cap}(e)$ is the capacity of e. Let OPT denote the optimal scheme that minimizes maximum link congestion. Thus, the *performance ratio* of r with respect to $\mathbf{D}$ is:

$$\text{PERF}(r, \{\mathbf{D}\}) = \frac{\max_{e \in E} \sum_{u,v} d^r_{uv}(e)/\texttt{cap}(e)}{\max_{e \in E} \sum_{u,v} d^{OPT}_{uv}(e)/\texttt{cap}(e)}$$

In the absence of any loss due to failure, the *performance ratio* for any routing scheme is at least 1. This definition can be extended for a set of TMs D as:

$$\text{PERF}(r, D) = \max_{\mathbf{D} \in D} \text{PERF}(r, \{\mathbf{D}\})$$

It is the worst performance ratio achieved for any $\mathbf{D} \in D$. We find that SMORE and SWAN remain optimal in 75-80% of the cases and overall, *SMORE has closest to optimal performance ratio.*