To Change Traffic Engineering, Change the Traffic Matrix

Paper 155

ABSTRACT

Traditional traffic engineering adapts the routing of traffic *within* the network to maximize performance. We propose a new approach to traffic engineering that also adaptively changes where traffic *enters* and *leaves* the network. To change the traffic matrix, we leverage recent innovations in virtual-machine migration (in data-center networks) and link migration through router grafting (in ISP backbone networks). We present an optimization framework for *traffic engineering with migration* and develop algorithms that determine which traffic end-points should migrate, and where. Our experiments with Internet2 traffic and topology data show that migration allows the network to carry at least 25% more traffic (at the same level of performance) over optimizing routing alone. Our theoretical results provide provable *worst-case* guarantees for maximizing network throughput.

1. INTRODUCTION

The rapid growth of online services, from video streaming to 3D games and virtual worlds, is placing tremendous demands on the underlying networks. ISP backbone networks carry more traffic than ever, and emerging data-center networks support tens of thousands of servers with widely varying traffic demands.

To address these challenges, network operators do *traffic engineering* (TE). Traditionally, traffic engineering involves tuning routing-protocol parameters to control how traffic is routed across the network, to optimize performance and use network resources effectively. Thus, today's traffic engineering adapts routing *within* the network for a given *traffic matrix*, *i.e.*, the volume of traffic between *fixed* traffic ingress and egress points.

Recent innovations challenge this traditional approach to traffic engineering. In data centers, live virtual-machine migration allows network operators to move virtual machines from one physical location to another; in ISP networks, network operators can dynamically rehome their ends of links to external networks via *router grafting* [14]. In both environments, the network operator can go beyond adapting the routing protocol to control where traffic enters and exits the network. Thus, the network operator now has the power to, in effect, *change the traffic matrix*.

In this paper, we introduce *traffic engineering with migration*, and present an optimization framework that addresses the following questions:

- How much can traffic migration improve over traditional traffic-engineering techniques?
- Which traffic end-points should migrate, and where should they migrate to?
- Can a "good" placement of traffic end-points be computed efficiently?

While finding the optimal solution in this new setting is computationally intractable, we show that a relatively simple heuristic offers significant performance improvements in practice. Our experiments with Internet2 traffic and topology data show that migration would enable the network to carry 25% more traffic at the same level of performance. These results apply to two different measures of performance—minimizing congestion (as a sum over all the links) and maximizing overall throughput (so as to deliver as much traffic as possible).

We show, however, that this heuristic can miss opportunities for even better performance. This motivates the design of computationally-efficient algorithms with provably "good" worst-case performance, relative to an idealized optimal solution. We explore how well the optimal solution for traffic engineering with migration can be *approximated*. We develop approximation algorithms and prove that our algorithms achieve good approximation guarantees in environments of interest (over 87% of the optimum in some cases). Our algorithms leverage an interesting connection we establish between traffic engineering with migration and classical constraint satisfaction problems in complexity theory. Understanding the average-case performance of these algorithms (on realistic traffic and topology data) is a natural next step, and part of our ongoing work.

Organization. After a brief review of traditional traffic engineering in Section 2, we introduce traffic engineering with migration in Section 3. Section 4 presents a simple heuristic for traffic engineering with migration and an experimental evaluation of this heuristic. We present algorithms with provably good worst-case performance in Section 5. Section 6 presents our initial results on extensions to our algorithms that cluster traffic end-points that have the same set of potential homing locations. We wrap up with a presentation of related work in Section 7, and conclusion in Section 8.

2. TRAFFIC ENGINEERING TODAY

In traditional traffic engineering, the network is represented by a graph $G = (V, E)$, where the vertex set V

represents routers or switches, and the edge set E represents the links. Every edge $e \in E$ has capacity $c_e > 0$. We are also given a *traffic matrix* $D = \{d_{ij}\}_{i,j \in V}$, where entry $d_{ij} \geq 0$ is the amount of traffic that vertex i wishes to send vertex j. The goal is to distribute flow across the paths from i to j to optimize an objective function. We now present two common objective functions:

Minimizing total link usage (TLU): TLU minimization reflects a common goal in ISP networks [8]. Each link e has a "cost" that reflects its level of congestion, where lightly-loaded links are "cheap" and links become exponentially more "expensive" as the link becomes heavily loaded. The *cost function* ϕ_e specifies the cost as a function of f_e (the total flow traversing the edge) and c_e (the edge capacity). Every ϕ_e is a piecewise linear, strictly increasing and convex function (see Section 4 and [8] for concrete examples). The goal is to distribute the *entire* demand between every pair of vertices in a manner that minimizes the sum of all link costs (i.e., $\Sigma_{e \in E}\ \phi(f_e, c_e)$). (Observe that the flow along an edge can exceed the edge's capacity.)

Maximizing the sum of throughputs (SoT): SoT maximization captures the objective of maximizing the network throughput, that is, the overall amount of received traffic. This is reasonable when a company (or other organization) wishes to fully utilize network bandwidth (*e.g.*, if traffic is not delay sensitive). When maximizing SoT the goal is compute a traffic flow that (1) does not exceed edge capacities; (2) does not exceed the offered demands; and (3) maximizes the total traffic volume $\Sigma_{i \in V} \Sigma_{j \in V}\ f_{ji}$, where f_{ji} is the amount of traffic received at vertex i that originates in vertex j.

Both objectives can be cast into the classical multicommodity flow framework; TLU minimization can be formulated as *minimum-cost multicommodity flow*, whereas SoT maximization is the extensively studied *maximum multicommodity flow*. Thus, these two objective functions are realizable using existing algorithms for computing multicommodity flows. Realizing these objectives in practice can be done via MPLS and a management system that solves the optimization problem and installs the resulting paths. Network operators often take the indirect approach of tuning Interior Gateway Protocol (IGP) weights to closely approximate the optimal distribution of the traffic [8].

3. TE WITH MIGRATION

In this section, we present mechanisms for changing the underlying traffic matrix, in both ISP and data-center networks. Then, we formulate a new model for traffic engineering that incorporates migration.

3.1 Traffic Migration Techniques

Traditional traffic engineering assumes that the locations of traffic sources and sinks cannot change over time. The ability to migrate users—to adaptively change where users connect to the network—gives rise to new possibilities in traffic engineering. Dynamically relocating traffic end-points can redirect traffic to decrease the traffic traversing a congested bottleneck, or capitalize on unused bandwidth. We now describe mechanisms for migrating traffic in ISP and data-center networks.

3.1.1 ISP Networks: Migration with Router Grafting

An ISP network connects to neighboring networks (customers, peers, or providers) at its perimeter. To establish a link to another network, the ISP selects one of its routers to connect to the adjacent network. Traditionally, the link remains fixed unless there is significant reason for change. This is because changing to a different internal router in real time can be extremely disruptive to the Border Gateway Protocol (BGP) session with the neighboring network, requiring significant coordination such as scheduling a maintenance window. During the transition period, data packets may be lost or delivered out of order, and routers throughout the Internet receive additional BGP update messages.

A new technology, called *router grafting* [14], enables an ISP to move its end of the link without disrupting user performance and without coordination with the neighboring network. Router grafting rehomes the layer-three link (through signaling in the programmable transport network), migrates the local end-point of the TCP connection to the neighbor's router, and transparently transfers the routing-protocol state to a different internal router. Router grafting has no impact on the neighboring network—the neighbor is not aware that grafting has happened, and sees no change in where traffic enters or leaves its own routers. (This is sharp contrast to traditional interdomain traffic engineering, where an ISP changes its BGP policies to shift traffic from one edge link to another—triggering both BGP update messages and changes in where traffic enters or leaves neighboring networks.) Router grafting makes it possible to migrate a link within a few seconds without disruption, allowing network operators to change the ingress and egress points for traffic in real time.

The overhead of router grafting is relatively low. Grafting involves the export of state from one router, the transference of state, and the import of state at another router. Changing the network topology requires some routers to repeat the route-selection process, leading to a temporary increase in CPU load. In addition, some routers may change their routing decisions, leading to a temporary increase in BGP update messages. These overheads are short-lived, and do not disrupt the flow of data traffic. As such, network operators can afford to make periodic adjustments to where they terminate the links to neighboring networks.

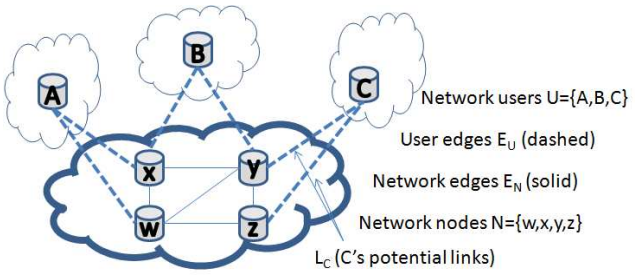


Figure 1: Network model for traffic engineering with migration.

3.1.2 Data Centers: Virtual Machine Migration

Data centers can contain tens of thousands of servers [9], and are becoming bigger every year. The networks connecting server clusters are also growing larger, and must satisfy the demands of applications like MapReduce that impose large and highly-variable traffic demands [10]. As such, traffic engineering in data-center networks is increasingly important. Commonly, the physical servers can host multiple virtual machines (VMs). *Live migration* allows a running VM to move from one physical server to another; this capability is available in each of the major server virtualization platforms (e.g., KVM [15], Xen [25], and VMWare [23]).

Migration of VMs can be done in time ranging from seconds to minutes, depending on how much state there is, and on whether there is disk to be transferred. In environments where VM migration is too expensive, data-center operators can "fire up" a new VM instance offering the same service, while gradually "draining" the load off the old virtual machine. Either way, the effect is the same—the service connects to a new location in the data-center network.

Just as with router grafting, VM migration also has a cost. In VM migration, the amount of state exported, transferred, and imported can potentially be large. Fortunately, these transfers can take place "in the background," with only the final transfer (of the last changes to the VM before it moves) happening in real time. So, while data-center operators must be judicious in migrating VMs, periodically moving some VMs to new locations does not introduce much overhead.

3.2 Migration-Aware Traffic Engineering

We now extend the traffic-engineering model in Section 2 to incorporate migration. Table 1 summarizes the notation.

Distinguishing users from network nodes: In our model for traffic engineering with migration, the network (see Figure 1) is represented by a graph $G = (V, E)$, where the vertex set V is the union of two disjoint subsets, U and N. U is the set of *network users*, that is, originators and receivers of traffic, and N is the

Notation	Description
G	Network graph $G = (V, E)$
V	Network vertex, union of U and N
E	Network edge, union of E_U and E_N
U	Set of network users
N	Set of network nodes
E_U	Subset of edges that connect user $u \in U$ to network nodes in N, $E_U \subseteq U \times N$
E_N	Subset of edges that connect network node $n \in N$ to network nodes in N, $E_N \subseteq N \times N$
c_e	capacity of edge $e \in E$
L_u	Potential links, $L_u \subseteq E_U$
D	Demand matrix, $D = \{d_{ij}\}_{i,j \in U}$
d_{ij}	Amount of traffic that user i wishes to send user j
ϕ_e	cost function used in TLU minimization, function of f_e/c_e
f_e	Total flow traversing the edge e

Table 1: Summary of notation used in model of traffic engineering with migration.

the set of *network nodes*, that is, the routers/switches in the network. The term "users" here refers to users of the network and not to end-users. In an ISP network, the set of users U represents routers in neighboring networks ("adjacent routers") and the set of network nodes N represents the routers in the ISP's internal network ("internal routers"). In data center networks, the set of users U represents the VMs and the set of network nodes N represents the network switches.

User edges are potential links: To capture the ability to migrate, we introduce the notion of *potential links* that represent the locations where the user can *possibly* connect to the network. The edge set E is the union of two disjoint subsets, E_U and E_N, where $E_U \subseteq U \times N$ is the subset of edges connecting users to network nodes, and $E_N \subseteq N \times N$ is the subset of edges connecting network nodes to other network nodes. Each edge $e \in E_N$ has capacity $c_e \geq 0$, which measures the amount of flow that can traverse edge e. We impose no capacity constraints on the edges in E_U (that is, these edges have infinite capacity). We call the set of all edges $L_u \subseteq E_U$ that connect user $u \in U$ to network nodes in N "the set of u's *potential links*" (that is, $\forall u \in U$, $L_u = \{e = (u, v) |\ e \in E_U\}$).

In ISP networks, the set of potential links L_u for each adjacent router (user) u represents the points at which u can connect to the ISP network. This can, in practice, depend on the underlying transport network that can, for example, limit a user to connecting only to network nodes in nearby geographical regions. In addition, the set of potential links can reflect latency considerations, *e.g.*, it is beneficial to home frequently-communicating

users near each other. In data center networks, the set of potential links can depend on a number of factors such as the network topology and application requirements (*e.g.,* VMs that need to be in the same broadcast domain, have access to a common storage area network, reside in different portions of the data center so as to increase tolerance to faults, *etc.*).

Demand matrix is user-to-user: Our model distinguishes network users from network nodes, and our demand matrix captures this distinction; we are now given a demand matrix $D = \{d_{ij}\}_{i,j \in U}$, where each entry d_{ij} specifies the amount of traffic *user* i wishes to send *user* j.

Each user must use a single potential link: The high-level goal is, for every pair of users i and j such that $d_{ij} > 0$, to distribute flow from i to j between the routes from i to j in G, subject to the constraint that every user can only connect to the network via a *single* link. That is, for every user $u \in U$, traffic flowing from that user to the other users, and vice versa, can only traverse a single edge in L_u; traffic along all other edges in L_u must be 0. When optimizing the flow of traffic through the network we can again consider the TLU minimization and SoT maximization objectives.

3.3 Practical Considerations

Naturally, our formal framework does not capture all the constraints that could arise in practice. We now present several such constraints and discuss how these can be incorporated into our model. We revisit some of these in later sections and leave the others as interesting directions for future research.

Cost of migration. Our framework does not model the cost of migration (in terms of processing, offline time, and more) yet this is expected to be a consideration in practice (we present some indication of the impact of this cost, based on experiments with Internet2 data, in Section 4.3). We can incorporate that cost into our model as follows. The input will include, in addition to the other components, an edge $\overline{e_u} \in L_u$, for every user $u \in U$, that represents the link that user u is *currently* using to connect to the network, and also costs associated with changing each user u's current connection edge to other edges in L_u.

Router/switch limitations. Other practical considerations are the physical limitations of the individual vertices in the network, including the number of links that each vertex can support, and also the capacity of the node (in terms of processing, memory, bandwidth, *etc.*). This can be incorporated into our model through additional constraints (*e.g.*, limits on the number of incoming links per node, node-capacity functions dependent on incoming traffic amount, *etc.*).

Multi-homed users. We did not model the case that users are multi-homed, that is, that users connect to the network at more than one location. This alters our constraint that a single potential link must be chosen per user. To incorporate this into our model we can introduce a variable for each user u that specifies how many links in L_u that user is allowed to send/receive traffic along. It also adds the complexity that changing the ingress point may alter the egress point (*i.e.,* "hot-potato routing" [22]), thus changing the traffic matrix beyond the change introduced with migration. The design and evaluation of heuristics/algorithms for this more general environment is left for future work.

User dependencies. In data center networks, there can be dependencies between users, *e.g.*, VMs that must be placed at proximate locations (so as to be in the same broadcast domain, or to have access to a common storage area network), VMs that must reside in different portions of the datacenter (so as to increase tolerance to faults), *etc.*. Our definition of the potential links (the L_u's) does not capture all such dependencies (*e.g.*, two VMs that need to be close to each other, without caring exactly where they both are placed). Understanding how to address such dependencies in traffic engineering is important. To model this, it is possible to use ideas and formulations from work on optimizing the placement of VMs (*e.g.,* [17]).

4. MIGRATION IMPROVES TE

Multicommodity flow provides an optimal solution for traditional traffic engineering with a fixed traffic matrix. Ideally, we would also be able to find an optimal solution to traffic engineering with migration. Unfortunately, this is intractable.

THEOREM 4.1. *Computing the TLU minimizing or the SoT maximizing outcome in traffic engineering with migration is NP-hard even when* $|L_u| \leq 2$ *for all* $u \in U$.

However, even a relatively simple heuristic, that we term the "*max-link heuristic*", achieves good performance on the Internet2 traffic and topology. We now present the max-link heuristic, that utilizes multicommodity flow, along with the performance evaluation using Internet2 data. Then, we show that the max-link heuristic has poor worst-case performance, motivating the design of algorithms with provably "good" worst-case guarantees in Section 5.

4.1 The Max-Link Heuristic

The max-link heuristic first computes the multicommodity flow in the input network that contains *all* potential links. Then, the heuristic uses this "fully fractional" flow (where users' traffic can be split between all of their potential links) to choose a *single* potential link for each user, thus constructing a feasible ("integral")

solution. To do this, the max-link heuristic discards, for every user $u \in U$, all potential links in L_u but the single potential link which carries the most traffic in the multicommodity flow solution (breaking ties arbitrarily).

The max-link heuristic consists of these three steps:

- **Step I: Compute multicommodity flow** f in the input network G (that contains *all* potential links for each user) for the given demand matrix D. That is, compute the minimum-cost multicommodity flow for TLU minimization, or maximum multicommodity flow for SoT maximization, without restricting users to sending and receiving traffic along a single potential link. The multicommodity flow solution f tells us how much traffic every user u sends and receives along each of the potential links in L_u. We let $t(l_u)$ denote the sum of traffic that user u sends and receives along the potential link $l_u \in L_u$.
- **Step II: Use the most utilized potential link.** Choose, for every user $u \in U$, a potential link for which $t(l_u)$ is maximized. (Migrate users' potential links if necessary.)
- **Step III: Compute the multicommodity flow in the resulting network**, that is, in the network obtained through the removal from G of all potential links but those chosen above. The max-link heuristic outputs (1) the choice of a single potential link for each user and (2) the optimal routing of traffic subject to these migration decisions.

4.2 Experimental Results on Internet2

We now present our experimental evaluation of the max-link heuristic. The goal of this evaluation is to demonstrate the benefits of using migration in traffic engineering, even with a simple heuristic. We show that our max-link heuristic does indeed lead to significant improvement in network performance.

We based all of our experiments on data collected from Internet2 [12], which consists of $N = 9$ core routers and $U = 133$ external routers. We used previously collected NetFlow data which provides summaries of the sampled flows (at the rate of 1/100 packets) for each router. Every NetFlow entry contains the incoming interface, which we used to represent an external source user. We used the routing tables for each of the routers to determine the egress router for each flow, along with the specific interface on the egress router that the flow exits the network on, which we used to represent the external destination user. This enabled us to generate an external-user-to-external-user traffic matrix.

4.2.1 TLU Minimization

Our results for TLU minimization appear in Figure 2, which shows results for the original (optimally engineered) network (the "original" line), and for traffic engineering with migration with 2 and 3 potential links per user.

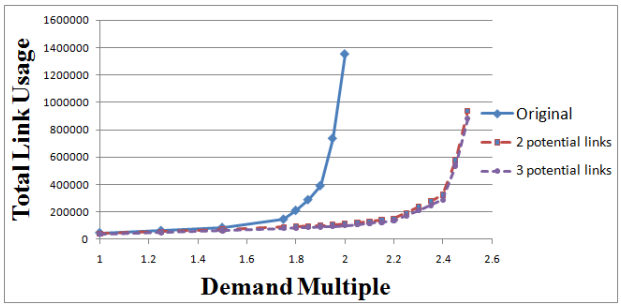


Figure 2: Results for TLU minimization with the max-link heuristic.

When computing the multicommodity flow (in Step I), we used a demand matrix that is a multiple of 1.75 times the measured traffic matrix, representing a point where the current network could still operate without experiencing the exponential rise in costs. Our choice of the sets of potential links (the L_u's) in the experiments was based on geographical distance, with the users' locations inferred from which router they are connected to in the original topology. We do not present results for 4 potential links per-user, as in this case (in our small topology) almost every two users end up connected to a common network node, and thus traffic between these users does not traverse the network at all. (To elaborate, consider the extreme case in which all users have potential links to all routers. Here, a multicommodity flow solution will give no guidance on which links to use since no traffic will even traverse the network.)

To obtain the graph in Figure 2, we varied the traffic demand by scaling all entries by a multiplicative factor, plotted on the x-axis, and optimized for the TLU for each. TLU minimization captures the goal of avoiding congestion, and involves an exponentially increasing cost for utilizing a link (see Section 3). We used the cost function from [8], shown below:

$$\phi_e(f_e, c_e) = \begin{cases} f_e & 0 \le \frac{f_e}{c_e} < \frac{1}{3} \\ 3f_e - \frac{2}{3}c_e & \frac{1}{3} \le \frac{f_e}{c_e} < \frac{2}{3} \\ 10f_e - \frac{16}{3}c_e & \frac{2}{3} \le \frac{f_e}{c_e} < \frac{9}{10} \\ 70f_e - \frac{178}{3}c_e & \frac{9}{10} \le \frac{f_e}{c_e} < 1 \\ 500f_e - \frac{1468}{3}c_e & 1 \le \frac{f_e}{c_e} < \frac{11}{10} \\ 5000f_e - \frac{16318}{3}c_e & \frac{11}{10} \le \frac{f_e}{c_e} < \infty \end{cases}$$

Due to the exponentially increasing cost, the network operator will wish to be at a point in the curve that comes before the exponential rise, that is, before the "knee" in the curve. Observe that this "knee" shifted to the right by roughly 25%, and so, with migration, the network can handle 25% more traffic with the same level of congestion. Note also that with 3 potential links per

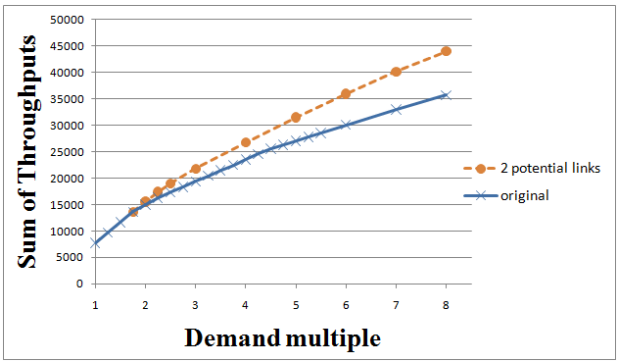


Figure 3: Results for SoT maximization with the max-link heuristic.

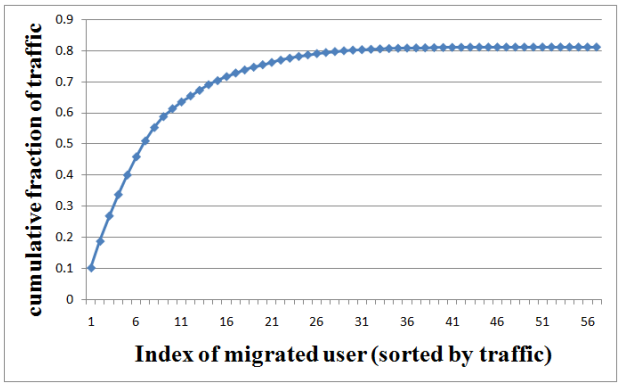


Figure 4: Traffic sent and received by migrated users as a fraction of the total traffic.

user we only achieve slightly better results than with 2 potential links per user. This is due both to how we selected the sets of potential links (based on the geographic locations), and to the exponentially rising cost of using a link. Specifically, traffic from a user in one geographic region to a user in a remote geographic location will often have to traverse a certain link regardless of the specific ingress and egress points. That link will therefore be congested in both the 2 potential link case and 3 potential link case, and will then dominate the TLU because of the exponentially rising cost function.

4.2.2 SoT Maximization

Similarly to the TLU minimization experiment, in Figure 3 we show our results for SoT maximization for the original network, as well as for the 2 potential links case. We omit the 3 potential links case as the results are roughly equivalent to the 2 potential links case. When computing the multicommodity flow (in Step I) we used a demand matrix that is a multiple of 8 times the measured traffic matrix, representing a point where the current network is congested.

When maximizing SoT, the goal is to maximize the amount of traffic that gets through the network, and so higher points in the graph in Figure 3 are more desirable. Under a certain load, the network can handle all offered traffic, as illustrated by the linear relation for low values on the x-axis. As expected, the results are similar to the results for TLU minimization in that we are able to send more traffic through the network with migration.

4.3 Effects of Migration Cost

To obtain the results for the max-link heuristic in our experiment with Internet2 data shown in Figure 2, we migrated 57 of the 133 users. Our formulation did not incorporate the cost of migration, yet this is expected to be a consideration in practice. To decide which users to migrate, we can weigh the cost of migrating a user against the gain from migrating that user; when the impact of migrating a user is low (*e.g.*, when that user generates and consumes negligible amounts of traffic), migration might be undesirable.

To investigate this, we plotted in Figure 4 the users whose links were migrated (sorted by the amount of traffic they generate/consume) on the x-axis, and the cumulative fraction of the total traffic on the y-axis. From this we learn that 75% of the traffic comes from the first 10 users, and 95% from the first 20 users. Hence, migrating 20 users is sufficient to achieve a significant improvement in network performance.

While 57 (or even 20) users may seem to be a significant fraction of total users, this does not imply that a constantly high rate of migration will be necessary in practice. The starting point in our experiments was a network that has not capitalized on migration yet; presumably, once users are homed to good locations, much fewer migrations would be necessary thereafter.

4.4 Worst-Case Performance of Max-Link

We now show that, while our experimental evaluation establishes that max-link heuristic leads to significant improvement in network performance, this heuristic suffers from poor worst-case performance. This motivates the design of algorithms with provable worst-case guarantees in Section 5.

We focus on the SoT maximization objective. Consider the network in Figure 5. User A aims to send 1 unit of traffic to user B across the network. A has two potential links, l_1 and l_2, and B has two potential links, l_3 and l_4. The capacities of the network edges are as in the figure. Observe that, in the case that we allow all potential links to be utilized, the maximum (SoT maximizing) multicommodity flow solution gets A's entire demand across the network to B. To achieve this, A sends $\frac{1}{2}+\epsilon$ along l_1 and $\frac{1}{2}-\epsilon$ along l_2. B then receives $\frac{1}{2}-\epsilon$ traffic along l_3 and $\frac{1}{2}+\epsilon$ traffic along l_4. Now, observe that the execution of the max-link heuristic results in user A using only the link l_1 and user B using only the link l_4. Thus, A can only send 2ϵ units of traffic across the network. Contrast this with the optimal

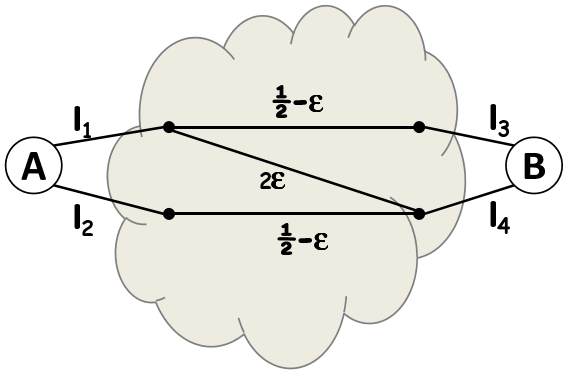


Figure 5: Worst-case performance of the max-link heuristic.

solution in which A uses l_1 and B uses l_3 (or where A uses l_2 and B uses l_4) to see that the ratio of the output of the max-link heuristic (in which the SoT is 2ϵ) and the optimal solution (in which the SoT is $\frac{1}{2} - \epsilon$) is unbounded (as ϵ can be arbitrarily small).

Intuitively, the max-link heuristic has poor worst-case performance because the choice of potential link for each user is independent from the choices for other users, and is based solely it makes on the traffic along the user's own potential links. The algorithms we present in the next section achieve much better worse-case performance by making "global" decisions.

5. APPROXIMATION ALGORITHMS

In this section, we present several algorithms that provably *approximate* the optimal SoT-maximizing outcome; we leave the design of TLU-minimizing algorithms for future research—we do point out that our hardness results for SoT maximization have implications for TLU minimization as well.

We first present a fast and simple combinatorial algorithm for traffic engineering with migration that has provable approximation guarantees. We then establish an interesting connection between our framework and classical problems in complexity theory, and show that this can be leveraged to design algorithms with better approximation ratios (*e.g.*, a randomized algorithm that achieves at least 87% of the optimum when each user has at most 2 potential links).

The choice of which of these algorithms to use in practice must reflect the tradeoffs between them in terms of worst-case performance, average-case performance and computational costs. These can depend on the specific context (network size, demand matrix, *etc.*), and are left for future research.

5.1 Overview of Our Algorithms

Ideally, we would be able to achieve constant-factor approximations to the optimum in a computationally-efficient manner regardless of the network topology, the number of users, the number of potential links, *etc.* Unfortunately, we show that, in general, even finding a constant-factor approximation to the optimum is intractable; as the maximal number of potential links per user grows, the approximation guarantees must become worse.

THEOREM 5.1. *Approximating the SoT maximizing outcome in traffic engineering with migration within a constant factor is NP-hard.*

We present algorithms whose approximation ratios depend on $k = max_{u \in U}|L_u|$.

5.1.1 Introducing the Link-2-Link Graph

We now present an illustration of some key ideas in our algorithms. Consider the users A and B, who communicate across the network in Figure 6(a). Both A and B have two potential links (see figure). Our algorithms first compute the maximum (SoT maximizing) multicommodity flow f in the network that contains all potential links (l_1, l_2, l_3, and l_4 in the figure), for the given demand matrix. The multicommodity flow solution is then used to construct an artifact we term the "*link-2-link graph*", illustrated in Figure 6(b) for the specific network in Figure 6(a).

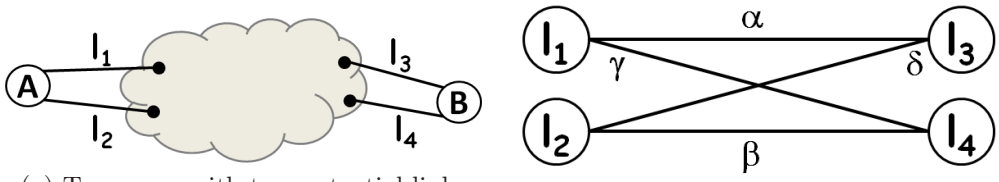
The vertices in the link-2-link graph represent users' potential links (see figure), and each edge weight is the total flow in f between the potential links that are its endpoints, *e.g.*, α in the figure is the sum of the total flow in f that leaves user A along l_1 and reaches user B along l_2 and the total flow in f that leaves user B along l_2 and reaches user A along l_1. Observe that the link-2-link graph does not contain information about how traffic is routed in f *within* the network, but does capture correlations between choices of potential links for different users (absent in the max-link heuristic).

Once the link-2-link graph is constructed, our algorithms use it to guide the choice of which potential link to choose for each user. We present several methods for doing this (combinatorial, using linear programming, and using semidefinite programming) and analyze the worst-case performance of these methods.

5.1.2 Algorithmic Framework

Our algorithms consist of the following four steps.

- **Step I: Compute the maximum multicommodity flow** f in the input network G (that contains *all* potential links for each user) for the given traffic matrix D. Clearly, the value of this maximum multicommodity flow OPT_f is an upper bound on the optimum when we restrict every user to only send/receive traffic along a single potential link. Set $f(l_u, l_v)$ to be the total amount of flow that leaves user u through the potential link $l_u \in$



(a) Two users with two potential links

(b) Link-2-link graph

Figure 6: Examples illustrating how to provably approximate the optimal solution

L_u and reaches user v through the potential link $l_v \in L_v$.

- **Step II: Construct the "link-2-link graph"** $\widehat{G} = (\widehat{V}, \widehat{E})$ as follows. Create, for each user $u \in U$, a set of k vertices $\widehat{V}_u$ that represents the set of u's k potential links L_u, and that we shall thus henceforth identify with L_u. Set $\widehat{V} = \bigcup_{u \in U} \widehat{V}_u$ $(= \bigcup_{u \in U} L_u)$. Create an edge $e \in \widehat{E}$ between every vertex $l_u \in L_u$ and every vertex $l_v \in L_v$ for $u \neq v \in U$, and set e's *weight* to be $w_e = f(l_u, l_v) + f(l_v, l_u)$.

  Observe that every $|U|$ vertices $v_1, \ldots, v_{|U|} \in \widehat{V}$, such that no two are in the same L_u set, represent a possible solution (in which users send/receive traffic only along these potential links), and shall hence be called a "*solution*". Observe also that for every such $v_1, \ldots, v_{|U|} \in \widehat{V}$, the total sum of weights of the edges between them is a lower bound on maximum SoT in the network in which (only) these potential links are selected.

- **Step III: Compute a solution T^* in $\widehat{G}$**, that is, $|U|$ vertices $v_1, \ldots, v_{|U|} \in \widehat{V}$, such that no two are in the same L_u set.

- **Step IV: Output the maximum multicommodity flow in the resulting network**, that is, in the network obtained through the removal from G of all potential links but those in the chosen solution T^*. Our algorithms output (1) the choice of a single potential link for each user and (2) the optimal routing of traffic subject to these migration decisions.

We now move on to presenting algorithms for SoT maximization in traffic engineering with migration. The algorithms we develop all have the above structure and differ only in how the solution T^* in Step III is chosen.

5.2 $\frac{1}{k^2}$-Approximation Algorithm

We present a simple deterministic algorithm that achieves a $\frac{1}{k^2}$ approximation ratio. The algorithm computes the multicommodity flow for the input network $G = (V, E)$ and then constructs the graph $\widehat{G} = (\widehat{V}, \widehat{E})$, as in Step I and Step II above. We now explain how the algorithm selects a solution T^* in Step III (which it then uses to output the SoT maximizing flow as in Step IV above). Intuitively, this algorithm constructs k possible solutions, and then selects the best of these to be T^*.

We introduce the following notation: $\forall X \subseteq \widehat{V}, W(X)$, called X's *weight*, is the total sum of the weights of edges in $\widehat{E}$ whose endpoints both lie in X. Observe that $W(\widehat{V}) = OPT_f$. Given the link-2-link graph $\widehat{G}$, the algorithm constructs k candidate solutions $T_1, \ldots, T_k$, and selects the solution for which $W(T_i)$ is maximized, as follows.

- Set $T_i := 0$ for every $i \in [k]$.

- Go over the users in U in some arbitrary order $u = 1, \ldots, |U|$, and, at each step, place the vertices in $\widehat{V}_u$ within the sets $T_1, \ldots, T_k$, subject to the constraint that no two vertices be placed in the same T_i set, so as to maximize the expression $\Sigma_i W(T_i)$

  (Consider, for example, the link-2-link graph in Figure 6(b), and the case that we start with user A. A's potential links, l_1 and l_2 will be placed in T_1 and T_2, respectively (this choice is arbitrary). Then, B's potential links, l_3 and l_4 will be placed in T_1 and T_2, respectively, if $\alpha + \beta > \gamma + \delta$, and in T_2 and T_1, respectively, otherwise.)

- Set $T^* := argmax_i W(T_i)$.

THEOREM 5.2. *The algorithm is computationally efficient and has approximation ratio* $\frac{1}{k^2}$.

PROOF. To see that the algorithm is computationally efficient observe that, as k is constant, going over all the possibilities of placing the vertices in $\widehat{V}_u$ in the T_i sets can be done efficiently.

We are left with showing that the approximation ratio is indeed $\frac{1}{k^2}$. Consider the following simple randomized process for constructing $T_1, \ldots, T_k$. Go over all users and, for each user u, place the vertices in $\widehat{V}_u$ in the sets $T_1, \ldots, T_k$ uniformly at random, subject to the constraint that no two vertices be placed in the same T_i sets. Observe that for the set generated using this process it holds that $E(\Sigma_i W(T_i)) \geq \frac{1}{k} W(\widehat{V})$.

This randomized process can be derandomized using the method of conditional probabilities so that the resulting deterministic process construct the sets $T_1, \ldots, T_k$

precisely as in our algorithm. Hence, we have that our algorithm generates sets $T_1, \ldots, T_k$ such that $\Sigma_i W(T_i)) \geq \frac{1}{k} W(\widehat{V})$. Observe that when choosing the best solution T^* it must therefore hold that $W(T^*) \geq \frac{1}{k^2} W(\widehat{V})$. The theorem now follows from the observation that $W(\widehat{V}) = OPT_f$. □

5.3 MAX-2-AND and Traffic Engineering

5.3.1 Leveraging Solutions to MAX-2-AND

MAX-2-AND is a special case of the maximum constraint satisfiability problem (MAX-CSP), which is a classical problem in complexity theory that has been the subject of extensive research. We now establish an interesting connection between traffic engineering with migration and MAX-2-AND. Specifically, we show that Step II and Step III in Section 5.1 can be cast into the language of MAX-2-AND. This enables us to leverage existing algorithms for MAX-2-AND to design algorithms for traffic engineering with migration with significantly better approximation ratios than $\frac{1}{k^2}$.

In MAX-2-AND, the input consists of (1) n variables $x_1, \ldots, x_n$, where each variable x_i takes values in a set S_i (we consider the general case that not all variables are necessarily boolean); (2) m constraints $c_1, \ldots, c_m$, where each constraint c_i is of the form "$x_i = \alpha$ *AND* $x_j = \beta$"; and (3) a weight $w_{c_i} > 0$ for every constraint c_i. The objective is to find an assignment to the variables that maximizes the total sum of the weights of the satisfied constraints.

We make the following observation. Given a link-2-link graph $\widehat{G} = (\widehat{V}, \widehat{E})$ as in Step II in Section 5.1 we can create an instance of MAX-2-AND as follows. We create, for each user $u \in U$, a variable x_u and set S_u to be $\widehat{V}_u$. We also create, for every edge $e = (l_u, l_v) \in \widehat{E}$, a constraint c_e that is only satisfied if $x_u = l_u$ and $x_v = l_v$, and has weight $w_{c_e} = w_e$. Observe that this is an *approximation preserving* reduction, and thus finding a solution T^* in Step III in Section 5.1 is equivalent to finding an assignment to the variables in the constructed instance of MAX-2-AND.

Hence, we can replace Step II and Step III in Section 5.1 with equivalent two steps where we construct an instance of MAX-2-AND and then find an assignment to the variables in the MAX-2-AND instance. This implies that obtaining a good approximation in our context (when taking this approach) boils down to finding a good approximation to MAX-2-AND. We can thus leverage the extensive work on designing algorithms for MAX-CSP to design algorithms for traffic engineering with migration.

5.3.2 Implications for TE with Migration

We now show how known algorithm for MAX-CSP can be used to to approximate traffic engineering with migration beyond the $\frac{1}{k^2}$ ratio presented in Section 5.2. Using these algorithms we can obtain an approximation ratio of $\frac{1}{k}$ for every $k > 0$, and an approximation ratio of nearly 90% for the interesting special case $k = 2$.

- **Improved combinatorial algorithm.** Datar *et al.* [6] present a combinatorial algorithm for MAX-CSP. Using this algorithm, we can improve over the approximation ratio of the combinatorial algorithm we presented in Section 5.2 and achieve an approximation ratio of $\frac{4}{9}$ for the case that $k = 2$, and of $\frac{1}{2k-2}$ for $k \geq 3$.

- **Linear programming algorithm with approximation ratio $\frac{1}{k}$.** Serna *et al.* [21] present an LP-based algorithm for MAX-CSP (see also [6]). This algorithm can be used to provide a $\frac{1}{k}$ approximation algorithm for SoT maximization in traffic engineering with migration.

- **Semidefinite programming algorithm for the case $k = 2$ with approximation ratio ∼90%.** We can also leverage semidefinite programming algorithms for MAX-2-CSP to design algorithms for SoT maximization for the special case that $k = 2$. Specifically, we can use the algorithm of Lewin *et al.* [16] to approximate traffic engineering with migration to within a ratio of at least 0.874.

We can further exploit the connection between traffic engineering with migration and MAX-2-AND to show that the last algorithm is essentially "tight" for the 2 potential links case. This follows from the inapproximability result for MAX-2-AND in [5].

THEOREM 5.3. *When every user has at most* 2 *potential links, approximating the SoT maximizing outcome within a factor of* 0.87435 *is NP-hard if the Unique Game Conjecture is true.*

The choice of which of the above algorithms to use in practice must reflect the tradeoffs between them in terms of worst-case performance, average-case performance and computational costs. These can depend on the specific context (network size, demand matrix, *etc.*), and are left for future research.

6. DISCUSSION: CLUSTERING USERS

When evaluating the max-link heuristic, we also investigated the benefits of considering the users in groups, and not individually. Grouping users together is motivated by the fact that, in practice, often large numbers of users can connect to the network at the exact same locations (*e.g.,* via a common access network). In addition, clustering users reduces the network size, thus making computation more efficient. Our experimental results, however, did not match our expectations; these show that clustering users can indeed be beneficial, yet

that this is dependent on careful choices of the cluster sizes and contents. We now present this approach, along with its performance evaluation. We conjecture that clustering will be more effective in larger networks.

6.1 The Cluster User Heuristic

Under the cluster user heuristic, users are considered in groups and not individually. The intuition behind this heuristic is the following. Consider the scenario that the set of users is divided into groups (or clusters) of users, such that every cluster contains a large number of users who all can connect to the network at the exact same locations (network nodes), and such that each user's demands constitute a small fraction of the demands of the cluster as a whole. We observe that, in this case, each cluster can be regarded as a single user with the ability to split traffic among its potential links almost as in the optimal multicommodity flow solution. This follows from the fact each user in the cluster sends/receives a negligible amount of the total traffic, and so users in the cluster can be mapped to outgoing links so as to closely mimic the multicommodity flow solution.

To illustrate this point, consider the example in Figure 7(a). There are six users (labeled A-F) which are grouped into two clusters (cluster 1 and cluster 2). In the cluster user approach, users can (but do not necessarily) belong to the same cluster if their sets of potential links connect to the network at the exact same network nodes.

We create a new network where each user cluster is replaced by a single user (with a set of potential links that connect to the network at the exact same network nodes). We then solve multicommodity flow for this network (allowing traffic to be split between multiple potential links) to get the fraction of traffic flowing over each potential link (shown in Figure 7(a)).

We now use the multicommodity flow solution to map users to potential links, as shown in Figure 7(b). In our example, 0.79 units of cluster 1's traffic should be via w and 0.21 units via x. Hence, we map A (0.60) and B (0.20) to w, and C (0.20) to x. Observe that this is the "best fit" as splitting the traffic exactly as in the multicommodity flow is impossible.

We note that, in general, finding the best fit can easily be shown to be NP-hard, even in the case that every user has 2 potential links, yet good approximations are achievable. In our experiments, we were able to use a brute-force approach to determine the best fit; for each cluster, we go over all possibilities for mapping users to potential links; we pick the mapping that minimizes the sum of gaps between the traffic sent along the potential links in the mapping and in the multicommodity flow solution. In Figure 7(a), for example, a possible mapping is to map user A to node w, and users B and C to node x, which leads to a "penalty" of 0.38 (as $0.38 = |0.79 - 0.60| + |0.21 - (0.2 + 0.2)|$); mapping users A and B to w and C to x is the optimal solution with penalty 0.02.

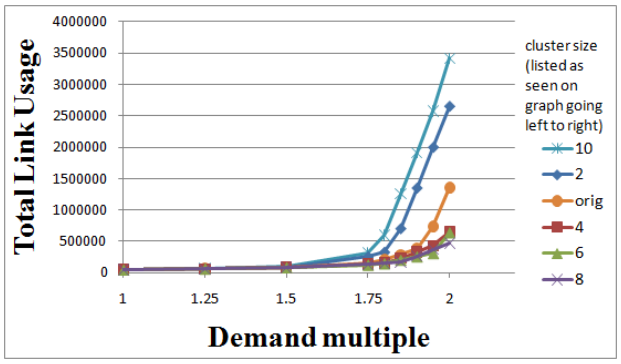


Figure 8: TLU minimization with cluster user heuristics (for 2 potential links per-user).

6.2 Evaluation with Internet2 Traffic

While the intuition behind the cluster user heuristic seems solid, our results are not conclusive due to the variables involved: the cluster size, and the cluster contents. Using the same setup as in Section 4, we experimented with several cluster sizes for the 2 potential links per user case (Figure 8) (the 3 potential links per user case does not provide any additional insights). Our results, show that traffic engineering with migration via cluster user has benefits, yet the cluster composition must be carefully chosen.

6.3 How to Select the Right Clusters?

Selecting the "right" cluster of users (and cluster size) is not trivial for the following reasons: (1) Traffic sent from a user in a cluster to another user in the same cluster is not considered (as the entire cluster is treated as a single user). Interestingly, the intuition that the bigger the cluster size is the more unconsidered traffic we have is not necessarily true; sometimes increasing the cluster size changed the division points and resulted in highly communicating users falling into different clusters, thus actually decreasing the unconsidered traffic. (2) Best-fitting users to potential links to match the computed multicommodity flow can lead to lost traffic if the fitting is not perfect (*e.g.*, in the multicommodity flow solution 0.79 units of traffic flow over the link to node w in Figure 7, yet the closest we can come is 0.80 units). In Figure 9 we show the effects of these two factors for the 2 potential links per user case. Observe that while the best results with the cluster user heuristic were achieved for a cluster size of 8 (see Figure 8), this is not obvious from Figure 9.

Understanding how to select user clusters (in a computationally-efficient manner) is a direction for future work. We suspect that the cluster user approach is more suitable for larger networks with more users, where we

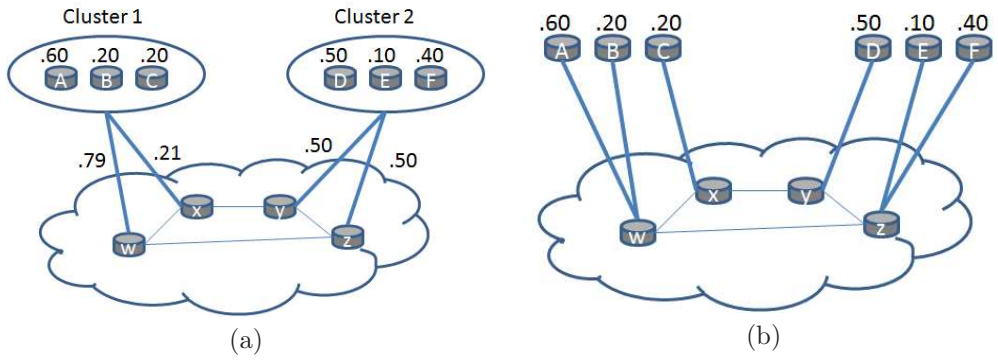


Figure 7: Illustration of the cluster user approach.

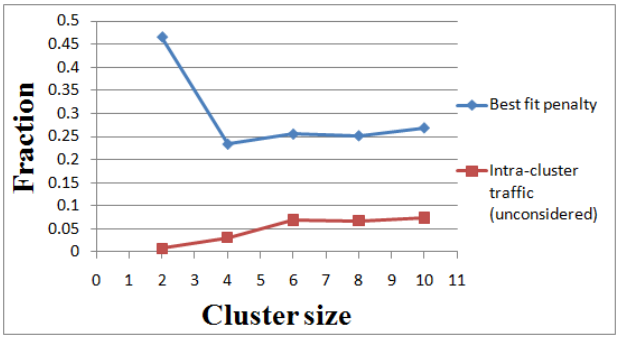


Figure 9: Implications of cluster size and contents with cluster user heuristics (for 2 potential links per-user).

have more freedom in choosing the composition of the clusters. In such networks, we can place users who communicate with each other in separate clusters so as to avoid not considering that traffic. We can also form big clusters with even distribution of traffic, thus making it easier not to lose traffic in the course of best-fitting users to potential links.

7. RELATED WORK

Due to its importance for network performance, there have been much research on traffic engineering.

7.1 ISP Networks

There has been much work on schemes for traffic engineering in ISP networks [24][13][7][4][27]. This work interprets traffic engineering as the adaptation of the routing of traffic within the network so as to optimize performance. We, in contrast, *also* explore how to adapt traffic's ingress and egress points.

Changes to the traffic matrix can also result from actions of the users themselves (*e.g.,* using overlay routing to route around congested areas, as in Detour [20] and RON [3]). However, such "selfish" overlay routing can, as pointed out in [19], significantly reduce the effectiveness of traffic engineering (as it lies outside the control of the ISP network operator). Interestingly, as these overlays shift the traffic, migration could be used to better handle the traffic within the ISPs network.

Migration in the ISP context has received but little attention. Mechanisms for the re-home of customers have been introduced [1] [14], but the implications for traffic engineering has not been studied.

7.2 Data center Networks

Traffic engineering in data center networks typically involves mechanisms that enable the network to carry the observed traffic patterns. Traffic patterns in data centers are more dynamic and so setting up static routes does not have the same effectiveness as in ISP networks. Instead, decisions are made on a finer granularity. VL2 [10] does this through the use of Valiant load balancing and modifications to the end-host operating systems to enable source routing. Hedera [2] adaptively schedules the paths flows take in a multi-stage switching fabric to efficiently utilize aggregate network resources. Both do this within the context of a network which relies on oversubscription of links.

Related is also the work on placing VMs in data centers. This work explores placement from the perspective of the server and application performance, and thus focuses on proximity in the network (*e.g.,* it is better to co-locate highly-communicating VMs; jobs for data processing applications such as MapReduce must be placed close to the data, otherwise the network suffers significant burden [26]). Network proximity in this work is incorporated into the overall placement cost. In [17], for instance, this "proximity cost" is simply the number of network switches between the VMs. Our focus is on the implications of VM placement for the network.

Existing techniques for placing VMs can lead to a situation where VMs interfere with one another (*e.g.,* with competing memory access patterns [18]). VM migration can be used to address this (*e.g.,* migrating VMs to improve network paths, as in SecondNet [11]).

8. CONCLUSIONS AND FUTURE WORK

The introduction of migration mechanisms challenges the traditional approach to traffic engineering. We pro-

posed a new approach to traffic engineering where instead of only optimizing network performance for fixed (predicted) traffic patterns, we also influence where traffic enters and exits the network. We showed that even a simple heuristic leads to significant improvements in network performance and presented algorithms with provably good worst-case guarantees.

We view our work as a first step in this direction. Incorporating more practical aspects of traffic engineering into our model (see Section 3.3), and designing algorithms to handle these, are promising directions for future research. Understanding the average-case performance and computational overhead of our algorithms, and further exploring the cluster user approach, are also left for future work.

9. REFERENCES

[1] M. Agrawal, S. Bailey, A. Greenberg, J. Pastor, P. Sebos, S. Seshan, J. van der Merwe, and J. Yates. RouterFarm: Towards a dynamic, manageable network edge. In *SIGCOMM Workshop on Internet Network Management*, September 2006.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proc. USENIX Networked Systems Design and Implementation*, 2010.

[3] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. ACM SOSP*, October 2001.

[4] D. Applegate, L. Breslau, and E. Cohen. Coping with network failures: Routing strategies for optimal demand oblivious restoration. In *Proc. ACM SIGCMETRICS*, June 2004.

[5] P. Austrin. Towards sharp inapproximability for any 2-CSP. In *FOCS*, pages 307–317, 2007.

[6] M. Datar, T. Feder, A. Gionis, R. Motwani, and R. Panigrahy. A combinatorial algorithm for MAX CSP. *Inf. Process. Lett.*, 85(6):307–315, 2003.

[7] A. Elwalid, C. Jin, S. Low, and I. Widjaja. MATE: MPLS adaptive traffic engineering. In *Proc. IEEE INFOCOM*, 2001.

[8] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proc. IEEE INFOCOM*, 2000.

[9] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39, December 2008.

[10] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, and P. Pat. VL2: A scalable and flexible data center network. In *Proc. SIGCOMM*, 2009.

[11] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: A data center network virtualization architecture with bandwidth guarantees. In *Proc. CoNEXT*, Dec. 2010.

[12] Internet2. `http://www.internet2.org`.

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

[14] E. Keller, J. Rexford, and J. van der Merwe. Seamless BGP session migration with router grafting. In *Proc. Networked Systems Design and Implementation*, 2010.

[15] KVM-The Linux Kernel-Based Virtual Machine. `http://linux-kvm.com/`.

[16] M. Lewin, D. Livnat, and U. Zwick. Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems. In *IPCO*, pages 67–82, 2002.

[17] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Proc. IEEE INFOCOM*, 2010.

[18] T. Moscibroda and O. Mutlu. Memory performance attacks: Denial of memory service in multi-core systems. In *Proc. USENIX Security Symposium*, 2007.

[19] L. Qiu, Y. R. Yang, Y. Zhang, and S. Shenker. Selfish routing in Internet-like environments. In *Proc. SIGCOMM*, 2003.

[20] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: A case for informed Internet routing and transport. *IEEE Micro*, January 1999.

[21] M. J. Serna, L. Trevisan, and F. Xhafa. The approximability of non-boolean satisfiability problems and restricted integer programming. *Theor. Comput. Sci.*, 332(1-3):123–139, 2005.

[22] R. Teixeira, T. Griffin, A. Shaikh, and G. Voelker. Network sensitivity to hot-potato disruptions. In *Proc. SIGCOMM*, 2003.

[23] VMWare. `http://www.vmware.com/`.

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

[25] Xen Hypervisor. `http://www.xen.org/`.

[26] Y. Yu, P. K. Gunda, and M. Isard. Distributed aggregation for data-parallel computing: interfaces and implementations. In *Proc. ACM Symposium on Operating Systems Principle*, 2009.

[27] C. Zhang, Z. Ge, J. Kurose, Y. Liu, and D. Towsley. Optimal routing with multiple traffic matrices: Tradeoff between average case and worst case performance. In *Proc. International Conference on Network Protocols*, Nov. 2005.