MicroTE: Fine Grained Traffic Engineering for Data Centers

Theophilus Benson[†], Ashok Anand[†], Aditya Akella[†] and Ming Zhang [⋆]
[†] University of Wisconsin-Madison; [⋆] Microsoft Research

ABSTRACT

The effects of data center traffic characteristics on data center traffic engineering is not well understood. In particular, it is unclear how existing traffic engineering techniques perform under various traffic patterns, namely how do the computed routes differ from the optimal routes. Our study reveals that existing traffic engineering techniques perform 15% to 20% worse than the optimal solution. We find that these techniques suffer mainly due to their inability to utilize global knowledge about flow characteristics and make coordinated decision for scheduling flows.

To this end, we have developed MicroTE, a system that adapts to traffic variations by leveraging the short term and partial predictability of the traffic matrix. We implement MicroTE within the OpenFlow framework and with minor modification to the end hosts. In our evaluations, we show that our system performs close to the optimal solution and imposes minimal overhead on the network making it appropriate for current and future data centers.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Network**]: Modeling and prediction

General Terms

Design, Performance, Reliability

Keywords

Data center network, traffic engineering

1. INTRODUCTION

Commercial corporations, private Enterprises, and universities heavily employ data centers to run a variety of applications and cloud-based services. These services range from Internet-facing “sensitive” applications, such as, Web services, instant messaging, financial applications and gaming, to computationally intensive applications, such as, indexing Web content, data analysis, and scientific computing.

The performance of these applications depends on efficient functioning of the data center’s network infrastructure. For example, a congested data center network, where internal traffic is routinely subjected to losses and poor throughput, could lead search queries to take longer to complete, IM message to get delayed, gaming experience to deteriorate, and POP mail services and Web transactions to hang. Dissatisfied end-users and subscribers could choose alternate providers, resulting in a significant loss in revenues.

In this paper, we identify the need for, and requirements underlying, a data center traffic engineering mechanism. We design and implement MicroTE, a fine-grained traffic engineering scheme that works atop a variety of underlying data center network topologies. Using real data center traces and synthetic workloads, we establish the effectiveness of MicroTE in accommodating various data center traffic patterns.

Data center traffic engineering is in a primitive state today. Most operators tweak wide-area traffic engineering and single-path routing mechanisms to perform traffic engineering in their data centers. This is only natural given that these mechanisms come bundled with current switches and they are well-understood. However, this naive approach is effective only if the traffic of data center networks and wide area networks share basic similarities. Other recent proposals for new data center interconnects adopt Equal-cost Multi-path-based techniques; e.g., both Fat-Tree [1] and VL2 [9] rely on ECMP to “spread” traffic across multiple candidate paths. However, these proposals have not been evaluated under a range of realistic traffic patterns and it is unclear whether simplistic ECMP is sufficient for such situations.

First, we seek to understand the extent of the performance gap between existing traffic engineering mechanisms and recent ECMP-based proposals. Next, we plan to analyze these techniques to determine the inefficiencies in their designs that lead to sub-optimal performance. In order to answer these questions, we conduct simulations using traces collected from two data centers, a large cloud computing data

ACM CoNEXT 2011, December 6–9 2011, Tokyo, Japan.

center and a university's private data center. We find that existing mechanisms achieve only 80-85% of the performance of an optimal routing mechanism that uses perfect knowledge about instantaneous demands and generates routes that minimize congestion. Thus, there is significant room for improvement. We find that existing techniques perform sub-optimally due either to (1) the failure to utilize multipath diversity, or (2) the failure to adapt to changes in traffic load, or (3) the failure to use a global view of traffic to make routing decisions.

While centralized routing platforms [14] can be leveraged to address the third shortcoming above, it is less clear how to design techniques that can optimally accommodate dynamic variations in data center network traffic to address the first two shortcomings. One option is to use recent proposals for traffic engineering in ISPs [4, 21, 15, 20] that aim to compute routes offering robust performance under a range of traffic matrices. However, these techniques function at the time-scales of hours. In contrast, measurement studies [13, 5] of data center traffic have shown that data center traffic is bursty in nature, and unpredictable at such long time-scales (especially at 100s or longer timescales), making these ISP-oriented techniques inapplicable.

Through our analysis of traces from the two data centers, we find that, despite the traffic being bursty, a significant fraction of data center traffic is predictable on *short* timescales of 1-2s. For example, upon examining the cloud data center, we find that in 80% of the 1s time intervals we measured, at least 30% of data center-wide traffic was contributed by ToR pairs (i.e., pairs of Top-of-Rack switches) whose traffic volume was roughly the same in the prior 1s interval. We also found that for nearly 70% of the ToR pairs the traffic remains stable for 1.5-2.5s on average. We found very similar properties in the private data center.

Motivated by this, we develop a technique, MicroTE, that leverages the existence of short-term predictable traffic to mitigate the impact of congestion due to the unpredictable traffic. MicroTE must operate at the granularity of seconds making it a fine-grained technique. Intuitively, MicroTE works as follows: it relies on a central controller to track which ToR pairs have predictable traffic at a fine granularity, and routes them optimally first. The remaining unpredictable traffic is then routed along weighted equal-cost multipath routes, where the weights reflect the available capacity after the predictable traffic has been routed.

Two challenges must be addressed in employing MicroTE: ensuring minimal modifications to data centers networks, and achieving scalability. To address the former challenge, we introduce careful end-host modifications coupled with an upgrade to the firmware in existing data center switches to use OpenFlow. To address the latter challenge, we introduce a variety of heuristics to scale statistics gathering of fine grained monitoring data and to minimize network-wide route re-computation when traffic changes.

We conduct large scale experiments using real data center traces to show the effectiveness of MicroTE in practice. The key observations from our evaluation are as follows: (1) MicroTE offers performance within 1–15% of optimal for real traffic traces. The sub-optimality is due to the fact that MicroTE operates at fine time-scales, but at a coarse spatial granularity. (2) Modifying MicroTE to operate at a fine spatial granularity, e.g., to monitor and route predictable server-to-server traffic enables MicroTE to perform 1–5% away from optimal. (3) When traffic predictability is very high, MicroTE performs closer to optimal routing. When traffic predictability is very low, MicroTE seamlessly shifts to ECMP. For intermediate situations, MicroTE offers performance between ECMP and optimal routing. (4) MicroTE imposes low overhead in terms of control messages required both to monitor traffic and to modify routing entries in OpenFlow switches.

Thus, the main contributions of our paper are threefold: (1) an evaluation and analysis of existing and recently-proposed traffic engineering techniques under real data center traces; (2) an empirical study of predictability of traffic in current data centers; and, (3) a new, fine-grained, adaptive, load-sensitive traffic engineering approach that virtually eliminates losses and reduces congestion inside data centers.

The remainder of this paper is structured as follows: we discuss background material on current TE approaches and on DC measurement studies in section 2. In section 3, we present a study of the performance of existing and proposed data center traffic engineering approaches. In section 4, we use our observations to list design guidelines for data center traffic engineering. We apply these guidelines in designing MicroTE in section 5. We present details of our implementation in section 6 and evaluate it in section 7. We present other related work in section 8 and conclude in section 9.

2. BACKGROUND

In this section, we present a brief overview of current traffic engineering techniques and the issues they face. Then we discuss some recent measurement studies in data centers, and their implications on traffic engineering.

2.1 Current TE Approaches

Most of the current data center topologies use **Spanning Tree** or **ECMP** for routing. In Spanning Tree, all traffic traverses a single tree, leaving many links unused. This approach avoids routing loops, but at the cost of efficiency. At high traffic load, the Spanning Tree algorithm can suffer high congestion and packet losses. **ECMP** mitigates some of these issues by utilizing the underlying path diversity. ECMP randomly splits flows across the available equal cost paths. ECMP works well when short flows dominate the network. However if there are "elephant" flows, then the random placement of flows can lead to persistent congestion on some links, while other links remain under utilized.

The problem with existing data center topologies is their inability to provide enough capacity between the servers they interconnect. This restriction exists because current data centers are built from high-cost hardware and data center operators oversubscribe the higher tiers of the tree to keep the total cost low. Thus, data centers are prone to congestion under heavy traffic workloads. Recent approaches (**Fat-Tree** [1, 16, 2], **VL2** [9]) have proposed better interconnects that can, in theory, support arbitrary traffic matrices under the hose model. Next, we review these approaches and their routing methodologies.

Fat-Tree is built on top of commodity switches using a Fat-Tree interconnection topology, which increases the aggregate bisection bandwidth while keeping the cost of networking infrastructure low. In this approach, a centralized scheduler places the long lived flows on distinct paths to reduce the likelihood of congestion. The design assumes that most bytes are carried by long-lived flows, and thus careful placement of long lived flows should be enough to avoid congestion. However, if flow size distribution is more uniform and consists mainly of large flows, their approach could lead to congestion and losses in network. For placing other flows, Fat-Tree utilizes flow classification, which like ECMP load balances flows across available paths. Since the flow size is not known apriori, every 5 seconds, Fat-Tree examines traffic from each flow and shifts large flows unto alternate links. With most of the flows lasting less than 1 second [13, 5], it is not clear if flow classification at this granularity is good enough. Moreover, their flow classification is performed locally on the paths leading up the tree towards the core, so it is likely for multiple flows to choose the same downward path from the core, leading to congestion downstream.

VL2 calls for replacement of specialized switches with commodity switches which are arranged in a Valiant load balancing architecture. This provides similar benefits as Fat-Tree: support for full bi-section bandwidth and cheaper networking infrastructure. VL2 provides similar guarantees as oblivious routing by utilizing ECMP style routing over the Valiant load balancing architecture. Although VL2 provides bounds over the worst case, it does this at the cost of average case performance. Further, VL2 forces all paths in the DC to be of equal length by inflating shorter paths. Ideally, Valiant load balancing should be performed on the level of packets to achieve the theoretical guarantees. However, to prevent reordering, Vl2 load balances at the granularity of flows, leaving it vulnerable to the same issues as ECMP (discussed earlier).

2.2 DataCenter Traffic Characteristics

Not much is known about the traffic characteristics in data center. A few recent studies [13, 6, 5, 9, 12] have shed some light. Here we review some of their findings and their implications on traffic engineering.

In [6, 5], Benson et al. evaluated several data centers and discovered that losses occur primarily at the edge of the data centers. In examining traffic at the edge of a data center, they found that the arrival pattern for the traffic can be characterized as a log-normal arrival process having ON periods, OFF periods, and inter arrival times drawn from 3 different log normal processes. Thus, they found that the traffic follows a heavy tailed distribution and is bursty. The implication of those observations on traffic engineering is that we cannot directly adopt traffic engineering techniques designed for ISPs, which function at coarse time scales of several hours and assume relatively smoother traffic patterns.

Both Kandula et al [13] and Benson et al [5] found that most data center flows (80%) last less than 10 sec and in general 100 new flows arrive every millisecond. They found that there are few long running flows (less than 0.1% last longer than 200s) contributing less than 20% of the total bytes, while more than half the bytes are in flows that last no longer than 25s.

The implications on traffic engineering is that, any traffic engineering technique (1) must scale to handle a large number of flows and (2) must accommodate both short and long lived flows without making assumptions on the size.

3. COMPARATIVE STUDY

In this section, we evaluate the effectiveness of various traffic engineering techniques and data center network architectures at accommodating data center traffic patterns. We perform an extensive study using simulations with traces from a data center running MapReduce style applications (We call this data center CLD). This data center is comprised of 1500 servers and 75 ToR switches. We collected several days worth of network events from the 1500 servers at 1s granularity. From these network events, we extracted, using a mapping of servers to ToR switches, the ToR-2-ToR traffic matrix. In our simulations of the various topologies and traffic engineering techniques, we feed as input into our simulator a sequence of per second ToR-2-ToR traffic matrix representing traffic over a 2 hour period. We note that this data center is identical to that studied by previous work [13].

We also repeated the analysis using similar traces collected at 1s time granularity from a university's private data center. Although this data center has a 3-tier hierarchy with 2 core routers, it differs in terms of size and hosted applications. While the previous data center hosts a variety of MapReduce style applications, the university data center (which we call UNV) hosts a variety of 2-Tier web-services and consists of about 500 physical servers and 2000 virtual servers. Our observations for UNV are qualitatively similar to CLD and we omit them for brevity.

Canonical tree topology: We first examine a canonical 2-tier tree topology with two cores, similar to that used in the cloud data center from which the data was gathered. On this topology, we first examine the performance of single path static routing, and then we examine the performance of ECMP (where flows are spread across 4 equal cost paths).

In Figure 1, we present the cumulative distribution of the maximum link utilization (MLU) every second for the tree

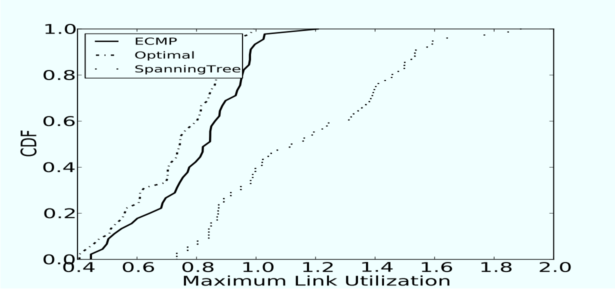


Figure 1: Distribution of the MLU for optimal, ECMP and Spanning Tree on a canonical tree topology. MLU greater than 1 indicates loss.

topology over the 2 hour period, when employing Spanning Tree, ECMP and optimal routing. Optimal routing is computed assuming perfect knowledge of the traffic matrix every second, where we formulate a linear program with the objective of minimizing the MLU. In general, we evaluate different TE techniques using MLU because it allows us to better understand the general health of the network. Highly utilized links are unable to accommodate traffic burst and will in general lead to buffering and increased latency.

We observe that ECMP and Spanning Tree perform worse than the optimal algorithm. In certain cases with MLU values greater than 1 indicating periods of severe congestion, we noticed packet loss for both ECMP and Spanning Tree. As expected, ECMP achieves lower MLU than Spanning Tree, because ECMP leverages multiple network paths and thus a larger network bandwidth. We find that there is a of 15-20% gap between optimal routing and ECMP under heavy loads. This gap exists because ECMP does not employ a global view when it schedules flows. We examine the impact of the lack of global view in the context of the Fat-Tree topology next.

In general, we find that ECMP is a better fit for data center networks, however, it is not perfect as it still results in a substantial amount of loss. ECMP can not eliminate loss because it attempts to create even utilization across all links by balancing the number of flows across multiple paths; however, ECMP does not take into account the instantaneous load of each flow which is central to controlling network-wide load and losses. Consider two source-destination pairs whose traffic is highly bursty, but the average load due to either pair is low. Nothing stops ECMP from assigning the two sets of flows to a common set of network links. Since ECMP does not re-assign based on observed load, it cannot help overcome losses due to temporary over-subscription on the path, which may happen when both sets of flows experience bursty transmission at similar times.

These analysis illuminate an interesting point, namely that although traffic engineering techniques must exploit multiple-path routing in existing data center topologies in order to obtain better performance, simply striping traffic across multiple paths is insufficient.

Fat-Tree: Next, we examine a recent proposal, the Fat-Tree interconnect, that supports high bisection bandwidth [1, 16]. In [1], the authors leverage a fixed number of shortest path routes between each source-destination pair, and use a *local* heuristic to balance load across the shortest paths in order to meet reduce congestion and ultimately loss. In particular, at regular intervals (say, every second), each switch in the lower level of the topology measures the utilization of its output ports (averaged over some period of time, say 10s) and reassigns a minimal number of flows if the utilizations of the output ports are mis-matched. Ideally, the Fat-Tree topology should be able to ensure *zero losses* on all links. In studying Fat-Tree we find that the local heuristic prevents this goal from being achieved. As observed in [1], there is a 23% performance gap between Fat-Tree and optimal due to conflicts between locally optimal decisions and globally optimal decisions. We omit the detailed results for brevity.

VL2: Although we have not evaluated VL2's architecture [9], we note the authors of VL2 perform a similar evaluation to ours. In evaluating VL2, Maltz et al. observed a performance gap of up to 20% with respect to the optimal. They attributed this performance gap to drawbacks in the underlying ECMP technique on which VL2 is based, namely the inability of ECMP to track and adapt to instantaneous demand.

Hedera: We conclude by evaluating Hedera [2] an approach which utilizes ECMP for short lived flows but utilizes a centralized approach to route large flows with over 100MB of data. In evaluating Hedera with our traces, we found that it performed comparable to ECMP as most of the contending traffic belonged in flows with less than 100MB of traffic.

To summarize, using real traces from a cloud data center, we have found that existing techniques fail to control losses in the presence of bursty traffic in data centers for one of the following reasons: (1) Not using multipath routing (2) Not taking instantaneous load into account and (3) Not making decisions on the basis of a global view of the network.

4. DESIGN REQUIREMENTS FOR TE

In summary, our evaluations show a performance gap of 15-20% between optimal routing and current routing practices. Even on recently proposed data center topologies such as Fat-Tree, the gap is 23%. The primary reason behind this gap is lack of global knowledge.

An ideal TE mechanism should configure routes dynamically by taking a global view of the future traffic matrix (TM) and computing optimal routes for the future traffic matrix at hand. One way to approximate this is to predict the future TM by extrapolating it from a series of historical TMs generated from the global view. However, this approach can only be used if the traffic matrix shows some predictability.

Recent measurement studies [13, 6] have examined the predictability of traffic in data centers; these studies show that data center traffic is bursty and that the traffic matrix lacks predictability at times scales of 150 seconds or longer. Furthermore, Benson et al. [6, 5], show that the arrival processes for data center traffic follows an ON-OFF pattern whose parameters can be explained by heavy tailed distributions, which provides further evidence that traffic is bursty and unpredictable on long time-scales.

4.1 Predictability

Given these results, initially it appears as though there is no predictability for us to leverage and hence our exercise in building an effective traffic engineering technique may be moot. In what follows, we present preliminary results to show that a significant amount of traffic is predictable at short time-scales.[1] We also examine the time-scales of predictability across different ToR pairs. To perform this study we examine traffic from the two data centers, CLD and UNV.

4.1.1 Prevalence and Time-scales of Predictability

We examine the change in traffic exchanged between each pair of ToR switch and find that across different time periods approximately 35% or 0.35 of the *total* traffic exchanged remains predictable. This is shown more precisely in Figure 2 (a), where we present the distribution of the fraction of total traffic demand contributed by those ToR pairs which had no significant change in traffic over a 1 second time period; we use $> 20\%$ change as a thresold for significant change. From Figure 2 (a), we observe evidence of different stability patterns across the different data centers. More specifically, in the CLD data center we observe that for 80% of the 1s intervals, more than 27% of the total traffic demand in the 1s interval remains predictable (i.e., does not change significantly) for at least 1 second into the future. For the UNV data center, we observe that in over 65% of time intervals 100% of the traffic remains stable within the next 1s time period. These results provides us with proof that a reasonable amount of short-term stability exists in the traffic demands of data centers.

Next, we attempt to more accurately determine the duration of the observed stability. To do this, we examine the run-length of the sequence of seconds where change in traffic for each ToR pair remains insignificant, i.e., less than 20% compared to the demand at the beginning of the sequence. In Figure 2 (b), we present the distribution of the mean run-lengths for each ToR-pair from both data centers studied. From this figure, we observe that in the CLD data center 60% of the pairs remain predictable for between 1.6 and 2.5 seconds on average. We observe a wider range of run lengths in the case of UNV: in nearly 35% of the ToR-pairs, the traffic is predictable for 1.5s-5s on average. Taken together, these observations prove, that for the predictable traffic, we should be able to use routes based on historical TM for the last 1 second to effectively route them. However, the traffic engineering technique must also be able to deal with ToR-pairs with much less or no predictability traffic.

4.1.2 Factors Causing Predictability

We observed that despite different purposes and sizes, predictability exists in both the data centers studied. We explain the observed predictability by revisiting the nature of the applications running in these data centers.

As stated earlier, the CLD data center hosts MapReduce style applications. In MapReduce, a master and several client machines work together in a two-step process to accomplish a task. In the first step the master maps tasks to various clients by sending them a set of commands. In the second step, the clients transfer the result of processing chunks of data back to the master. The first step often requires little network traffic, while the second step requires large network-wide bandwidth to transfer Megabytes/Gigabytes of data. In this particular data center, the MapReduce engine tries to eliminate communication between different ToR by co-locating clients and the master within the same rack. However, this is not always possible for large jobs, where the MapReduce engine is forced to map clients to other racks, leading to traffic between different ToR pairs. The stability we observe, is due to the reduce step when clients in a different rack from the master return results to the master. The amount and time-scale of predictable traffic is dependent on the number and spread of external clients. The unpredictable traffic is due to a mixture of different types of control traffic, ranging from control messages between the MapReduce engine and various clients, to data block replication between different clients, which happens asynchronously.

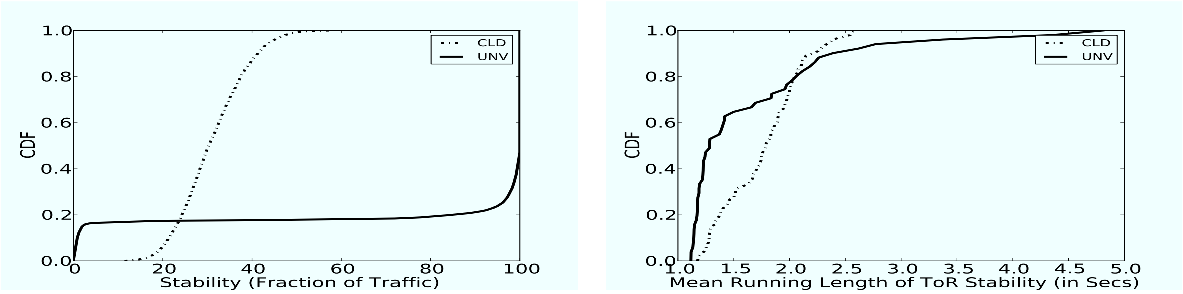
The UNV data center hosts a variety of 3-Tier web applications. The front-end receives requests from users and load balances along a set of business logic tier servers. These business logic servers in turn maintain persistent connections with the backend servers: the business logic pulls and pushes data to the backend. Unlike in CLD where care was taken to restrict most of the traffic to within a ToR, in UNV the applications are engineered in such a way that most of the traffic is exchanged between different ToR. The servers connected to any given ToR hosts application servers for one particular tier. For example, a particular ToR would host all HTTP web-servers for the front-end tier. To fulfill user requests, application servers from different tiers and thus diffe-rent ToR switches must communicate. Predictability arises due to the tiered nature of the applications; front-end only talks with business logic and the business logic only talks with the backend. However, this predictability is short-lived due to small size of messages exchanged and dependence on request pattern.

4.2 Design Requirements

Pulling all of our observations from our study of TE mechanisms in data centers and from our study on the predictability of data center traffic, we have established a set of three design principles that a TE mechanism designed for data centers must adhere to in order to effectively alleviate loss and reduce the maximum link utilization:

(1) Multipath Routing: An ideal approach must take

[1] Prior works implicitly rely on such predictability to augment data centers with "flyways" [12, 19].



(a) (b)

Figure 2: (a) Distribution of the fraction of total traffic demand and (b) Distribution of the mean run-length for top 100 ToR pairs.

advantage of the path diversity available in the data center topologies. Failure to do this will limit the available network capacity and increase the likelihood of congestion.

(2) Coordinated Scheduling using a global view of Traffic: An ideal approach must coordinate the scheduling of traffic across the available network paths using global view of the network. Failure to do this will lead to global suboptimal scheduling of flows - the local optimal paths may not satisfy the prevalent traffic demand leading to congestion.

(3) Exploiting short-term predictability for adaptation: An ideal approach must take into account short term predictability where applicable but must also adapt quickly to variations in the underlying traffic patterns, while generating routes that closely approximate the performance of the optimal routing TE. In worst case, such an algorithm should perform no worse than existing approaches. If the approach performs poorly, then it provides operators with no incentive to adopt it.

Next, we discuss our TE proposal keeping these design goals in mind.

5. MicroTE: ARCHITECTURE

In this section, we present the architecture for a framework, called MicroTE, that satisfies the design requirements laid out in Section 4. MicroTE is a TE mechanism with the ability to adapt to variations in the underlying network traffic patterns at the *microscopic* (per-second) level. MicroTE works by logically isolating predictable traffic from non-predictable traffic and taking appropriate routing decisions for both during every 1s interval. Specifically, MicroTE routes predictable traffic optimally within the network first, by computing routes meeting some global objective (e.g., minimizing the MLU. Then, MicroTE uses weighted ECMP to stripe unpredictable traffic (at the flow level) across the left over capacity along candidate paths for the traffic. We note that when traffic is not predictable, even prior proposals for optimal oblivious routing [3] degenerate into a form of weighted load-balancing (especially on highly regular topologies such as those in data centers). In contrast with these oblivious routing schemes, however, our ECMP-based heuristic is far simpler to implement, and as we will show, it is quite effective in practice.

MicroTE is designed to gracefully deal with varying levels of predictability: when a large portion of traffic is predictable, MicroTE routes all of it optimally. When the opposite is true, MicroTE seamlessly shifts to using ECMP. This intrinsic flexibility allows MicroTE to handle any type of data center.

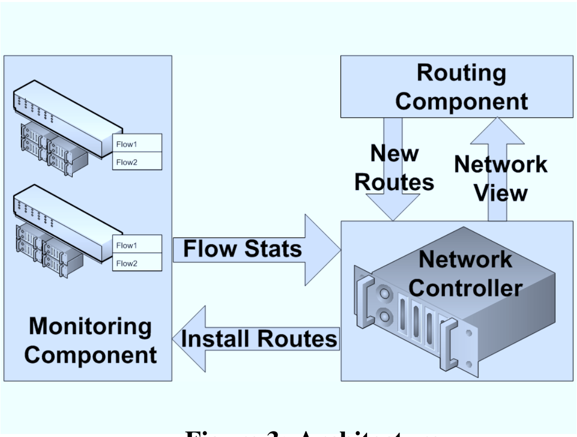


Figure 3: Architecture

In Figure 3, we show the key components of our framework and how they interact. MicroTE consists of three components; the monitoring component, a kernel module that monitors traffic patterns and determines predictability between racks; the routing component which calculates network routes based on the aggregated information provided by the network controller; and the network controller which aggregates the traffic demands from the servers and installs routes unto the network switches. Next, we discuss the design issues in each of these components in detail below.

5.1 Monitoring Component

The monitoring component monitors traffic demands, or flow statistics, between the rack that it is attached to and the other racks in the network. In terms of obtaining the flow statistics, there are currently two design alternatives: (1) the network controller periodically poll switches; many switches allow such probes and the OpenFlow API provides mechanisms to poll statistics for specific flows. The switches respond to the network controller with byte counts for flow entries for inter-ToR traffic, where the counts are aggregated over the poll seconds between successive polls. (2) the servers in a rack can perform measurements of their traffic sending patterns to other parts of the data center, and inform the controller of the demands, either at regular intervals or in a triggered fashion.

We chose to instrument the monitoring component within the server; the monitoring component is comprised of the set of servers attached to the edge of the data center. Using the servers over the network devices directly provides us with 3 advantages, namely: (1) it allows MicroTE to proactively respond to changes in demand, (2) it allows MicroTE to scale to a large network, and (3) it reduces the processing overhead imposed by MicroTE on the network devices. The server-based system offers these advantages in the following ways: (1) it allows triggered updates of traffic demands to the controller (when traffic demands change significantly), while a purely switch based approach, at least in the current implementation of OpenFlow, only supports polling by the controller, which is far less flexible; (2) it prevents the network controller from creating a significant amount of control traffic on the network by constantly polling all switches on nearly a per-second granularity; and (3) it shifts the bottleneck of constantly generating flow statistics from the switches to the endhosts.

Each of the servers in the monitoring components tracks the network traffic being sent/received over its interfaces as well as with whom these bytes were exchanged; however, only one server per rack is responsible for aggregating, processing, and summarizing the network statistics for the entire rack. This server, called the designated server, is also charged with sending the summarized traffic matrix to the network controller. To fulfill its role, the designated server must be able to perform the following tasks: (1) collect data from other servers in the rack, (2) aggregate the server to server data into Rack to Rack data, (3) determine predictable ToR pairs and (4) communicate this information with the network controller. Next, we discuss how each of these is realized.

5.1.1 Bootstrapping a new server

In order to report flow statistics to the controller, a new server must first register its address and location with the controller. We leverage the OpenFlow control messages to do this. In OpenFlow, whenever a new server attaches to a ToR switch, the switch sends a message to the NOX controller. Upon receiving this message, the controller records the location and address of the new server. With this control message, the controller is able to: (1) determine which ToR switch the new server belongs to; (2) delegate the first server that connects to a ToR switch as the designated monitor for the rack; (3) inform the new server of the location of the network controller; and (4) inform subsequent servers of the designated monitor for the rack.

In our current implementation, the network controller and monitoring components are preconfigured to communicate on a specific port.

5.1.2 Summarizing traffic data

The designated server in each rack receives traffic statistics from other servers in the rack on host-to-host traffic information. To reduce network overhead, this information should be compressed into ToR-to-ToR switch traffic information before being sent to the controller. To perform this compression, a server-to-ToR switch mapping must be available to each designated server. In our current implementation, MicroTE disseminates this mapping to the designated server when the controller registers the designated server's address (section 5.1.1). The controller then proceeds to provide updates to the mapping as changes occur. This allows each designated server to compress the host-to-host traffic matrix into a ToR-to-ToR switch traffic matrix.

5.1.3 Determining traffic predictability

Determining traffic predictability is one of the most crucial tasks performed by the designated servers. To determine predictable traffic, the server examines the ToR-to-ToR traffic matrix received over time and calculates the mean on each individual entry within the matrix. The traffic between a ToR switch pair is considered predictable if the mean and the instantaneous traffic being sent are within δ of each other. MicroTE uses an empirically determined default value of 20%, due to space constraints we exclude results of our evaluation.

In applying the mean, we need to choose the granularity at which measurements are collected and the mean window size. In Section 4, we found traffic between ToR switch pairs is predictable on a 2-second time scale. MicroTE must collect measurements and compute running average at a time scale smaller than 2 seconds. In our current design, a kernel patch is applied to each server in the DC to (1) collect the server's TM every 0.1 seconds (prior work [17] has shown that the CPU overhead incurred in analyzing network state for each connection at this rate is less than 4%) and (2) send a set of the TMs to the designated server every second, and (3) act as the designated server when informed by the network controller.

5.1.4 Updating traffic statistics

The designated servers need to continually send the summarized traffic data to the network controller. In a naive implementation, a designated server would send the entire traffic matrix to the network controller every time it is updated. However, this unnecessarily increases the amount of control messages transmitted over the network and processed by the controller. In our current implementation, the designated servers send updates to the controller only if the set of predictable ToR-to-ToR entries changes or if the traffic volume of any predictable ToR-to-ToR changes by more than δ.

5.2 Network Controller

The network controller is charged with the task of aggregating the information from the monitors. Upon annotating the network view with traffic volumes between ToR pairs and traffic predictability, this view is passed up to the routing component for the creation of network routes.

In addition to receiving information from the monitors, the network controller also relays routes received from the routing component to the switches; the controller installs the appropriate flow entries in the forwarding tables of each switch in the network.

To successfully fullfill its role, the network controller must be able to accomplish a few tasks, namely; (1) aggregate data from the monitors into a global view and (2) install flow forwarding entries into the flow tables on network switches. Next, we discuss how each of these is realized.

5.2.1 Aggregating monitoring data

MicroTE requires two pieces of global information about the network: the ToR-to-ToR traffic matrix and the topology. The controller can easily reconstruct the ToR-to-ToR traffic matrix from the data submitted by the monitoring components.

To create the network topology, the controller relies on a class of OpenFlow control messages exchanged between the switches and controller. This class of control messages are exchanged whenever a link or a switch becomes active or inactive. With these messages, the controller can determine the status of the links and switches as well as the network topology.

5.2.2 Recomputing flow entries

The routing component needs to periodically recompute the network paths as it receives an influx of traffic matrix updates from monitoring components. In fact, the network paths should be recomputed upon any changes in the traffic matrix, e.g., the set of predictable ToR pairs changes or the traffic volume of any predictable ToR pairs changes by more than δ. To guard against frequent re-computations triggered by a burst of traffic matrix updates, we choose a minimum re-computation interval of 1 second after the averages are received. i

5.3 Routing Component

This component is charged with computing network paths using the global view provided by the network controller. In Figure 4, we present a flowchart of the actions performed by the routing component. The routing component tracks the predictable and unpredictable ToR-to-ToR entries in the TM based on the available historical measurements of the entries. The routing component computes network paths by first routing predictable ToR-to-ToR traffic entries in the TM and then relegating the unpredictable ToR-to-ToR traffic entries in the TM to a weighted form of ECMP; the weight on each paths reflects the amount of bandwidth already consumed by the predictable traffic. By doing this, the routing component avoids the risk of performing worse than the existing ECMP based approaches, thus satisfying the third design requirement.

By taking into account the global view of traffic for generating routes that utilize all network paths, our first and second design requirements are fulfilled. Next, we explore a set of routing algorithms that can be applied within the routing components.

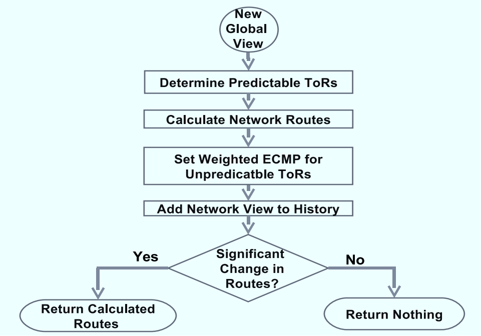

New Global View
Determine Predictable ToRs
Calculate Network Routes
Set Weighted ECMP for Unpredicatble ToRs
Add Network View to History
Significant Change in Routes?
Yes
No
Return Calculated Routes
Return Nothing

Figure 4: Flow chart of the logic within the routing component.

5.3.1 LP formulation

We represent a network as a graph $G = (V, E)$, where each vertex is either a server rack or a switch and each edge is a physical link. Let $f_{u,v,k}$ be the fraction of traffic between the pair (u, v) assigned to the k^{th} equal hop length path ($1 \leq k \leq K$), $P_{u,v,k}$ be the set of links on the k^{th} path between (u, v), and $T_{u,v}$ be the traffic volume between (u, v).

We have two constraints in the LP formulation. First, all the traffic between each (u, v) is routed along the K equal hop length paths between (u, v).

$$\sum_{1 \leq k \leq K} (f_{u,v,k}) \quad = \quad 1$$

Second, the total traffic volume traversing an edge e does not exceed its capacity Cap_e multiplied by the link utilization limit we are targeting, $util$.

$$\sum_{\forall u,v,k,e \in P_{u,v,k}} T_{u,v} * f_{u,v,k} \leq util \times Cap_e$$

The objective is to minimize $util$, the maximum link utilization, under the two constraints above.

5.3.2 Bin-packing heuristic

Now, we present our bin-packing heuristic. We begin by sorting the predictable ToR pairs in decreasing order according to their traffic volume. For each ToR pair, we compute the minimum cost path where the cost of each link is the reciprocal of its available capacity. We only consider K equal hop length paths between a ToR pair in finding the minimum cost path. After assigning the traffic of a ToR pair to a path, we update the cost each link along the path by deducting the ToR pair's traffic volume from the residual capacity of the corresponding links. In this way, a highly utilized link is unlikely to be assigned more traffic and the maximum link load will be minimized.

Next, we analyze the time complexity of the heuristic. Let N and E be the number of nodes and edges in the topology.

Computing a minimum cost path requires $O(E + NlogN)$ time. Suppose there are P predictable ToR pairs, the overall complexity of the heuristic is $O(PNlogN + PE + PlogP)$. The third term is for sorting the predictable entries in traffic matrix.

Although we have considered only two routing algorithms, MicroTE allows the plug-in of other algorithms to attain a dedicate balance between speed and optimality.

6. IMPLEMENTATION

We implement MicroTE using the OpenFlow framework. In OpenFlow, a logically-centralized "NOX" controller [18] written in software can add and delete forwarding entries at fine-grained time-scales. This enables low-level programmatic control over the routing and forwarding of the entire network. By sending appropriate commands to the switches, the controller is able to gather global network view and determine how flows traverse the network (by installing appropriate flow entries). We implement the network controller and routing component as C++ modules in the "NOX" framework and implement the monitoring component in C as a kernel module in Linux.

7. EVALUATION

In this section, we conduct a thorough trace-driven analysis of the effectiveness of MicroTE. In particular, we answer the following questions:

Performance: How well does MicroTE perform when compared to optimal routing or to ECMP-based approaches?

Fidelity: How well does MicroTE perform under different levels of predictability?

Speed, Scale and Accuracy: How well does MicroTE scale to large data centers? What overhead does it impose? How fast does it compute routes and install state?

We use the traces describes in the earlier sections in answering these questions.

7.1 Performing Under Realistic Workloads

We examine MicroTE's performance by replaying the data center traces on the canonical tree topology used for Figure 1. As before, in these simulations, ECMP is allowed to split traffic over 4 equal cost paths. In Figure 5, we present our results as CDF graphs of the MLU at various 1s time intervals for running the MicroTE, ECMP, and Optimal algorithms on the UNV(5 (a)) and CLD(5 (b)) data centers.

From Figure 5, we observe that MicroTE outperforms ECMP in all 1s intervals, but it also performs poorly (1% to 15% for UNIV) relative to Optimal.

Upon investigating MicroTE, we found that the sub-optimal performance is due to the coarse *spatial* granularity at which MicroTE currently functions, namely, routing predictable traffic at the granularity of the ToR pairs. As we observed in Section 4, the predictability is largely due to the behavior of individual applications. Thus, MicroTE can, in theory, achieve better performance if it were to operate at an even finer spatial granularity of application-level flows, or, even simpler, server-to-server traffic.

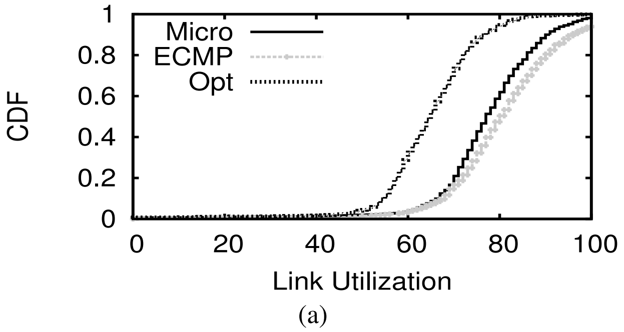
Next, we evaluate a modified version of our algorithm which operates at a finer spatial granularity by routing predictable traffic at the level of server-to-server. To emulate this situation using our Tor-to-Tor traffic matrices, we assume that: (1) traffic between a pair of servers in different racks is proportional to the "popularity" of the servers (we assign popularity to servers at random from (0,1)) and (2) if a ToR pair's traffic is predictable then the traffic between all the constituent pairs of servers is also predictable at the same time scales. In Figure 6, we present a graph of the MLU for the various algorithms operating at the finer spatial granularity. We observe that the gap between MicroTE and Optimal is significantly reduced to between 1% and 5%. The performance gains occurs because operating at such a fine spatial granularity allows MicroTE to better balance predictable traffic across existing links. Furthermore, we validate these performance gains by examining the goodput across all flows in the data center. We find that, in comparison to ECMP, MicroTE results in a 65% reduction in bytes lost overall and reduces the number of affected communicating pairs by 60% under heavy load situations (considering those instances when maximum utilization across all links using ECMP is $\geq$ 90%). Although both MicroTE and ECMP result in packet loss, MicroTE is able to reduce the number of affected flows by taking global view and thus result in a significant reduction in the amount of packets lost.

Employing the finer spatial granularity, while resulting in better performance, does lead to greater amount of control traffic and higher computational overhead. However, we believe that the optimizations we have proposed to summarize traffic statistics (Section 6.1) and the bin-packing heuristic (Section 6.3) help in eliminating these overheads. In Section 7.3, we show that these overheads are indeed small in practice.

7.2 Performing Under Different Levels of Predictability

Next, we examine the effectiveness of our techniques under varying levels of predictability. Figure 7, shows the performance of MicroTE and ECMP under two levels of predictability derived from the UNV traffic traces. The graph on the left plots all time intervals (on the x-axis) where predictable traffic contributed between 0 and 75% of all bytes and the one on the right plots all time intervals (on the x-axis) where predictable traffic contributed between more than 75% of all bytes. In general, we observe that across the different levels of predictability, MicroTE outperforms ECMP relative to the optimal algorithm.

When most traffic is unpredictable (left graph), we note that MicroTE becomes ECMP. In fact, in the former case, MicroTE also results in poorer performance than ECMP in a small fraction of intervals, whereas this does not happen in the latter case.



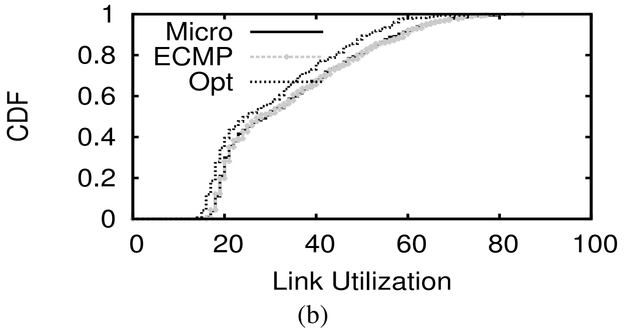


Figure 5: Comparison of MicroTE, Optimal and ECMP for (a) UNV and (b) CLD. For CLD, Micro and ECMP curves overlap.

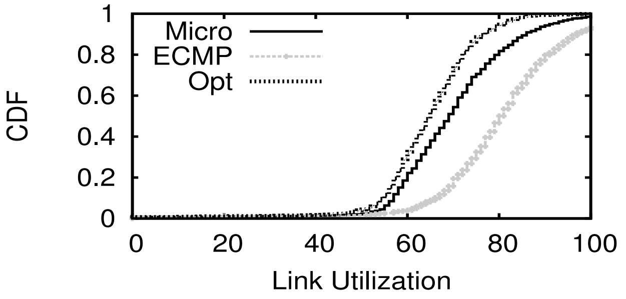


Figure 6: Comparison of MicroTE, Optimal and ECMP for UNV; ideal case with application level comparison.

In time instances when the traffic is more predictable (right graph), we also observed an increase in the overall traffic. Correspondingly, the average MLU for the various algorithms also rises. In Figure 7 (b), we observe that with higher traffic the naive ECMP algorithm starts to drop a significant amount of traffic with several peaks over 100 (and with a few peaks over 110). MicroTE, like Optimal routing, is able to utilize knowledge about the predictable traffic and route it optimally to avoid most of the losses that ECMP incurs.

On the whole, we find that MicroTE is able to seamlessly shift between performing like ECMP when predictability is low and performing like Optimal when predictability is high.

7.3 Scaling to Large Data centers

Next, we examine the ability of MicroTE to scale to large data centers. To do this, we examine several aspects of the architecture. We examine the size of the control messages exchanged between the network controller and the monitoring component as well as the time it takes for the routing component to computing network paths using the two routing algorithms described earlier. We evaluate MicroTE under data centers of varying sizes; the largest data center we consider has 10K hosts (a similar sized DC network as used in the evaluation of prior works [13, 6, 9, 12, 2]).

Prior work [13] has shown that within a 10s time period, a server in a data center communicates with 1-10% of servers outside the rack, and 0.5% of server pair communications are across the racks. As stated earlier, our CLD traces are identical as those used by prior work [13], and for our UNV traces we observe much more sparsity. We use the observations from the denser CLD DC as a model for studying the scalability of MicroTE.

7.3.1 Control Messages

We start our evaluation by examining the burden placed on the network by the control message between the ToR switches, the monitoring servers and the controller. A low network footprint is crucial for a TE mechanism because a high network footprint will force the TE to contend for network bandwidth and ultimately introduce new congestion points within the network. MicroTE is composed of 4 types of control messages: at server boot-up time the switch informs the controller of the servers, the controller elects and inform the servers of the designated monitoring server, the designated monitoring servers tracks network demand and informs the controller of this demand, and finally the controller configures network paths into the switches.

Of the four control messages, only two messages are constantly exchanged over the network, namely the message transmitting the network demand to the switches and the messages installing the network paths into the switches.

We first examine the footprint of the control messages carrying the traffic demand, which are transmitted every time unit of granularity on which MicroTE operates (i.e. a second). We assume that each rack in the data center has 25 servers. Each of the control messages carrying the network demand from a designated server to the controller is based on a compressed format requiring 8B of data per pair of communicating ToR switch (2B for source switch identifier, 2B for destination switch identifier, and 4B for traffic volume). When MicroTE operates in the finer grain server-to-server mode, the worst case overheard can become much higher. However, we notice that due to the sparsity of the communication matrix explained by Kandula [13], the actual control messages are 4 MB or less than 0.5% of a server's uplink capacity.

Next, we look at the footprint of the flow installation messages. The format of the messages between the controller and the switches is based on the OpenFlow protocol and uses 100B per flow entry. MicroTE defines a flow as a pair of communicating servers. In a network with 10K servers,

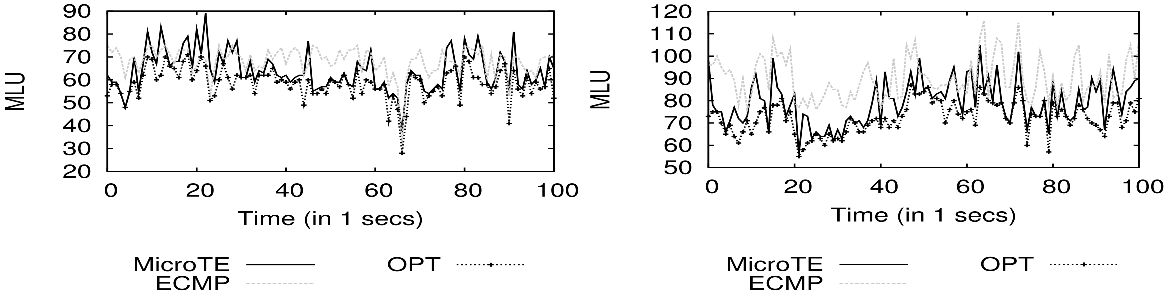


Figure 7: (a) Low and Medium predictability (between 0% and 75% of the matrix is predictable), and (b) High predictability (greater than 75% of the matrix is predictable). MLU greater than 1 indicates loss.

ToRs#	LP				Bin-Packing Heuristic			
	5%	**15%**	**25%**	**50%**	**5%**	**15%**	**25%**	**50%**
50	0.004	0.006	0.01	0.04	0.0007	0.002	0.004	0.007
100	0.03	0.17	0.22	0.6	0.007	0.015	0.026	0.055
200	0.52	1.55	3.05	6.85	0.025	0.08	0.12	0.25
400	4.47	28.07	-	-	0.1	0.3	0.5	0.98

Table 1: Runtime (in seconds) of Algorithm for different DC sizes

given the sparsity of the actual traffic matrix, we observed that the combined overhead would be 50 MB, which is less than 5% of a server's uplink capacity and as such the impact of these messages on the network will be minimal.

7.3.2 Reactiveness: Path Installation Speed

Now we compare the time required by the network controller to install routes in a network under varying number of average flows per server. To evaluate flow installation time, we setup a testbed with 2 HP Procurve 5400 switches running the OpenFlow firmware and a controller running an Intel Core 2 Quad CPU running at 2.66GHz, with 3072KB cache and 8GB of RAM. Each of the Procurve switches was directly connected to the controller with a 1Gig link.

We observe that in the average case, MicroTE is be able to install routes in a network where each server has on average 10 flows in less than 40 milliseconds. Similarly in the worst case we observe that MicroTE is able to install routes quickly, more specifically routes can be installed in under 150 milliseconds for a network wherein each server has on average 80 flows. Given our evaluations of flow installation time we believe that MicroTE should be able to install new routes in sufficient time to take advantage of the predictability within the underlying traffic.

7.3.3 Reactiveness: Routing Component Speed

Now we examine the time it takes to run the routing component on networks of varying sizes. To evaluate the run time performance of the various routing algorithms, we run the algorithms on a machine with an Intel Core 2 Quad CPU running at 2.66GHz, with 3072KB cache and 8GB of RAM.

Table 1 shows the runtime of just using the LP approach for data centers of varying size and communicating pairs. For small data centers with 50 ToRs, or those with 100 ToRs but with few communicating pairs being predictable (up to 15%), the LP algorithm can be used for MicroTE. However, in a network with 100 ToR, an increase in number of communicating pairs with predictable performance causes a rapid degradation in the performance of LP — it takes 0.6 seconds to compute routes for the case where 50% of ToR pairs send predictable traffic. Thus, the LP is not a good choice in this regime, especially when the requirement is compute routes to operate at a granularity of 1 second. For larger data centers, the performance is worse across the board: for a 200 ToR data center, it takes over 0.5 seconds to compute routes when only 5% of the pairs are involved in sending predictable traffic.

Table 1 also shows the runtime of the bin-packing approach. In contrast to the LP algorithm, bin-packing scales much better across all situations. For example, for the data center with 200 ToRs, it takes less than a quarter of a second to compute routes for the case with up to 50% of communicating pairs having predictable traffic. We note for 75% of the traces bin-packing performs as well as the LP; however, due to lack of predictability and the coarse grained nature of the bin-packing heuristic it performs as good as or worse than ECMP for 25% of our traces.

We next look at the runtimes of these algorithms for the case of using server-to-server traffic matrix. The total number of all-possible communicating pairs for server-to-server traffic matrix is quite large, however as indicated by earlier study [13], the server-to-server traffic matrix is relatively sparse in practice. We find that for 0.5% of communicating server pairs, the bin-packing algorithm takes close to 0.5 seconds for a data center with 3000 hosts. For very large scale data centers, further scalability can be achieved by parallelizing both of these algorithms.

Our results show that for mid to large-sized data centers, our bin-packing algorithm is appropriate because of its speed, while for small data centers, or mid-sized data centers (100 ToRs) with few predictable communicating pairs, LP is appropriate due to its greater accuracy.

To summarize, we find that MicroTE imposes a low overhead on the network (§ 7.3.1) and is able to calculate(§ 7.3.3) and install network paths (§ 7.3.2) in less than 1 second. We showed that MicroTE is able to support networks of varying sizes. Furthermore, the server-to-server version of MicroTE can scale in practice and provide better performance benefits compared to using ToR-to-ToR matrix, however the latter provides scalability even in the worst case communication patterns. In addition, we believe that work currently being done on the NOX controller and the OpenFlow firmware will only lead to improvements of our framework by reducing both the time it takes to install routes and the size of the control traffic required to install these routes.

8. OTHER RELATED WORK

Traffic Engineering. Traditional TE techniques, e.g., those applied to WAN traffic in ISP networks, work with predicted traffic matrices and operate on coarse time-scales of several hours [4, 21, 15, 7]. These are inapplicable to data centers because data centers traffic are predictable at much finer time scales. Other more responsive TE proposals for ISP, such as TEXCP [11], rely on local load balancing decisions which our work shows to be suboptimal in the context of DCs.

New Architectures. Contrary to our approach of reducing congestion by reconfiguring the routing within the network, others [9, 1, 10, 2] have argued for a forklift upgrade of the DC's network. They argue for the replacement of existing networks with a new substrate that can support a larger bisection bandwidth. We argue that such techniques will face slow adoption because a forklift upgrade will require significant capital investment thus delaying deployment.

Augmenting Data Center Networks. Complementary to our approach, is the use of techniques such as flyways([12, 19, 8]) that argue for adding extra links or devices as a means of tackling hot spots. These extra links provide additional capacities, where and whenever needed. Like us, they also find the predictability of traffic demands at short time scales, allowing flyways to keep up with the changing demand. Their solution also relies on a centralized controller for adapting flyways in dynamic manner. In contrast, our approach argues for fine grained TE with existing links while leveraging short term predictability of traffic demands. Our approach is more general and applicable to any DC topology, including DC topologies with flyways.

9. CONCLUSION

In this paper, we studied the effectiveness of various traffic engineering techniques and found that many of these techniques are inadequate for today's data centers for at least one of three reasons: (1) lack of multipath routing, (2) lack of load-sensitivity and (3) lack of a global view in making traffic engineering decisions.

We develop the requirements for an ideal TE mechanism for data centers based on empirical insights drawn from traffic patterns within a cloud data center and a private campus data center. In particular, we observe that data center traffic exhibits short-term predictability that last on order of a few seconds. Thus, we argue that a ideal TE mechanism: (1) must utilize multipath routing, (2) must coordinate scheduling of traffic, and (3) must adaptability to short term predictability.

To this end, we developed MicroTE, a new fine-grained DC traffic engineering approach, that satisfies our design goals. MicroTE uses a central controller to aggregate and create a global view of network conditions and traffic demands. Furthermore, MicroTE uses OpenFlow to coordinate scheduling of traffic within the network. We describe various options for routing and polling algorithms that allow MicroTE to estimate and adapt to short term predictability as well as to perform multipath routing. Extensive simulations and experiments show that MicroTE offers close to the optimal performance in practice when traffic is predictable, and degenerates to ECMP when traffic is not predictable. We note that ECMP is the best one can hope to achieve when traffic is unpredictable. Our results show that MicroTE outperforms recent proposals across a variety of settings.

10. ACKNOWLEDGMENTS

We would like to thank Srikanth Kandula and Dale Carder for providing the data center traces. We would also like to thank Chen-Nee Chuah (our shepherd) and the anonymous reviewers for their insightful feedback. This work is supported in part by a Cisco gift and the following NSF grants; CNS-0746531, CNS-1017545, and CNS-1050170. Theophilus Benson is supported by an IBM PhD Fellowship. Ashok Anand is supported by a Google Phd Fellowship.

11. REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. SIGCOMM '08, New York, NY, USA, 2008. ACM.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI '10*.

[3] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Racke. Optimal oblivious routing in polynomial time. STOC '03.

[4] B. Fortz and M. Thorup. Internet Traffic Engineering by Optimizing OSPF Weights. In *Infocom*, 2000.

[5] T. Benson, A. Akella, and D. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *Proceedings of IMC*, 2010.

[6] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding Data Center Traffic Characteristics. In *Proceedings of Sigcomm Workshop: Research on Enterprise Networks*, 2009.

[7] A. Elwalid, C. Jin, S. Low, and I. Widjaja. Mate: multipath adaptive traffic engineering. *Comput. Netw.*, 40:695–709, December 2002.

[8] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. SIGCOMM '10, NY, USA, 2010.

[9] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *SIGCOMM*, 2009.

[10] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. Dcell: a scalable and fault-tolerant network structure for data centers. SIGCOMM '08.

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

[12] S. Kandula, J. Padhye, and P. Bahl. Flyways to de-congest data center networks. In *Proc. ACM Hotnets-VIII*, New York City, NY. USA., Oct. 2009.

[13] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The Nature of Data Center Traffic: Measurements and Analysis. In *IMC*, 2009.

[14] N. Mckeown, S. Shenker, T. Anderson, L. Peterson, J. Turner, H. Balakrishnan, and J. Rexford. Openflow: Enabling innovation in campus networks.

[15] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: existing techniques and new directions. SIGCOMM '02.

[16] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM*, 2009.

[17] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. NSDI'11.

[18] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker. Applying nox to the datacenter. In *Proc. of (HotNets-VIII)*, 2009.

[19] G. Wang, D. G. Andersen, M. Kaminsky, M. Kozuch, T. S. E. Ng, K. Papagiannaki, and M. Ryan. c-Through: Part-time optics in data centers. In *Proc. ACM SIGCOMM*, New Delhi, India, Aug. 2010.

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

[21] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. Traffic Engineering with Estimated Traffic Matrices. Miami, FL, Oct. 2003.