2022

Design, Analysis, and Optimization of Traffic Engineering for Software Defined Networks

Mohammed Ibrahim Salman
Wright State University

DESIGN, ANALYSIS, AND OPTIMIZATION OF TRAFFIC ENGINEERING FOR SOFTWARE DEFINED NETWORKS

A Dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

by

MOHAMMED IBRAHIM SALMAN
M.S., University of Anbar, Iraq, 2012
B.S., University of Anbar, Iraq, 2006

2022
Wright State University

Wright State University
GRADUATE SCHOOL

July 28, 2021

I HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER MY SUPERVISION BY Mohammed Ibrahim Salman ENTITLED Design, Analysis, and Optimization of Traffic Engineering for Software Defined Networks BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Doctor of Philosophy.

Bin Wang, Ph.D.
Dissertation Director

Yong Pei, Ph.D.
Director, Computer Science and Engineering Ph.D. Program

Barry Milligan, Ph.D.
Vice Provost for Academic Affairs
Dean of the Graduate School

Committee on
Final Examination

Advisor, Bin Wang, Ph.D.

Yong Pei, Ph.D.

Krishnaprasad Thirunarayan, Ph.D.

Phu Phung, Ph.D.

ABSTRACT

Salman, Mohammed Ibrahim. Ph.D., Department of Computer Science and Engineering, Wright State University, 2022. *Design, Analysis, and Optimization of Traffic Engineering for Software Defined Networks.*

Network traffic has been growing exponentially due to the rapid development of applications and communications technologies. Conventional routing protocols, such as Open-Shortest Path First (OSPF), do not provide optimal routing and result in weak network resources. Optimal traffic engineering (TE) is not applicable in practice due to operational constraints such as limited memory on the forwarding devices and routes oscillation. Recently, a new way of centralized management of networks enabled by Software-Defined Networking (SDN) made it easy to apply most traffic engineering ideas in practice.

Toward creating an applicable traffic engineering system, we created a TE simulator for experimenting with TE and evaluating TE systems efficiently as this tool employs parallel processing to achieve high efficiency. The purpose of the simulator is two aspects: (1) We use it to understand traffic engineering, (2) we use it to formulate a new traffic engineering algorithm that is near-optimal and applicable in practice. We study the design of some important aspects of any TE system. In particular, the consequences of achieving optimal TE by solving the multi-commodity flow problem (MCF) and the consequences of choosing single-path routing over multi-path routing. With the help of the TE simulator, we compare many TE systems constructed by combining different paths selection techniques with two objective functions for rate adaptations: load balancing (LB) and average delay (AD). The results confirm that paths selected based on the theoretical approach known as Oblivious Routing combined with AD objective function can significantly increase the performance in terms of throughput, congestion, and delay.

However, the new proposed system comes with a cost. The AD function has a higher complexity than the LB function. We show that this problem can be tackled by training deep learning models. We trained two models with two different neural network architectures:

Multilayer Perceptron (MLP) and Long-Short Term Memory (LSTM), to get a responsive traffic engineering system. The input training data is based on synthetic data obtained from the simulator. The output of the two models is the split ratios that the SDN controller uses to instruct the switching devices about how to forward traffic in the network. The result confirms that both models are effective and can be used to forward traffic in an optimal or near-optimal way. The LSTM model has shown a slightly better result than MLP due to its ability to predict a longer output sequence.

Contents

List of Figures

List of Tables

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

– Robert Frost

"Like a dandelion up through the pavement, I persist."

– Wentworth Miller

1 Introduction

1.1 Overview

Network traffic has been growing exponentially due to the rapid development of communication technologies. In some applications, e.g., multimedia applications, packets retransmission is not an option. To achieve performance in computer networks, we need to study and do traffic engineering (TE). TE is about steering traffic to achieve a goal set in advance, e.g., load balancing traffic, minimizing the delay, etc. Computer networks and traffic patterns are dynamic. Networks may change over time as new network elements, and links are added/removed to/from the network. Some of these changes occur due to, e.g., link/router failure. Apart from that, traffic demands are highly dynamic. Due to this dynamicity, TE systems need to be adaptive to these changes to achieve a good performance.

This work proposes a centralized TE that suits SDN networks with the best possible performance and fast response to changes in the network. We aim to design a TE system by considering the operational constraints when applying it in the real world, e.g., routes oscillations and the limited memory on forwarding elements.

With the development of software-defined networking (SDN), a new type of traffic engineering system has been the focus of some recent research (e.g., [2, 3, 4, 5, 6], and many others). Big technology companies such as Microsoft [4] and Google [2] have shifted to this type of technology a few years back. The basis of all of these approaches is a centralized TE model. The output of a TE system includes a routing scheme that produces a set of candidate paths and the splitting ratios of traffic among these paths. TE techniques are now applicable under SDN because SDN provides two functionalities: (i) global visibility of the network (ii) direct control over network elements.

Finding answers to some of the questions related to this centralized TE model is an important step to develop TE for SDN. These questions center on TE objective functions,

path cardinality, path selection, and multi-path/single-path routing.

1.2 Thesis Statement

Building a traffic engineering system that is near-optimal and responsive is a challenging task. Paths selected using the Räcke's traffic engineering system, rate adaptation based on the average delay (AD) objective function, and deep learning models can be leveraged to build a responsive and near-optimal traffic engineering system.

1.3 Problem Statement

This work aims to build a traffic engineering algorithm with two goals in mind: 1) The traffic engineering algorithm should give a performance that is close to optimal. 2) The traffic engineering algorithm should give the resultant routing scheme as fast as possible. The routing problem can be formulated as a linear program (LP) or a mixed-integer linear program (MILP) to get the optimal routing scheme. This problem is called the multi-commodity flow problem (MCF). However, the time required to solve the LP or MILP can be significant. Figure 1.1 shows the required time to find the optimal solution using Gurobi optimization software [7] for two objective functions: average delay (AD) and load balancing (LB). The time increases as we start adding more nodes and links to the network. This raises a scalability issue which forces researchers to look for another solution (e.g., heuristic). Furthermore, the optimal solution we get by solving the MCF problem cannot be applied directly to a real-world network due to some operational constraints, e.g., limited memory, routing overhead. Therefore, the question we are trying to solve in this dissertation: Is there a traffic engineering algorithm that can give a solution that is instantaneous and close to optimal and scales better than the optimal solution we get from

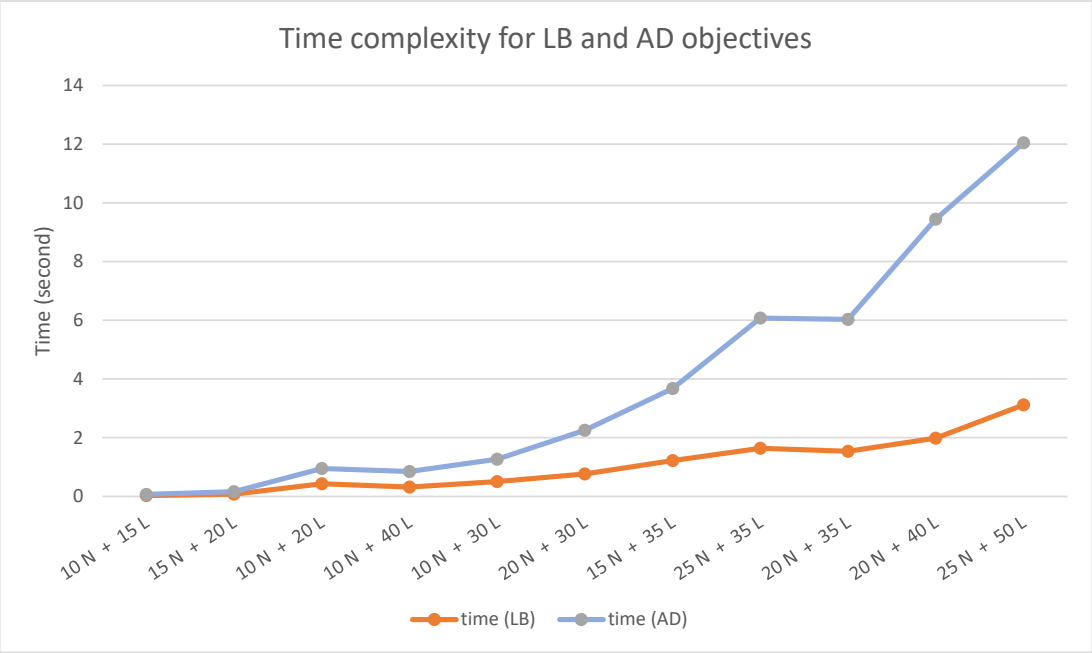


Figure 1.1: A graph showing the increased time complexity when solving the linear program for increasingly large networks for LB and AD objectives (N: number of nodes, L: number of links). Hardware specification: Intel(R) Core(TM) i7-685K CPU @ 3.60GHz

solving LP or MILP?

1.4 Rresearch Objectives

In this work, we try to answer these two questions: (1) Is there a TE system that outperforms state-of-the-art? (2) How to make that TE system update its routing scheme fast enough in response to changing demands or changing topology?

As it turns out, optimal routing requires solving LP or MILP. Reducing the time required to find the optimal routing scheme is necessary because the optimal routing scheme is a routing scheme that must be re-calculated whenever a change happens in the network. That change might be a change in traffic matrix, which is the usual case, or a change in network topology due to the addition/removal of a link or node or a link/node failure. This work will also study other questions that, if we solve them, we might find the answer to the original question, which is reducing the time required to get to the optimal or close-

to-optimal routing scheme. These questions can be categorized into three categories: 1) questions related to path cardinality and path selection algorithms. 2) questions related to the splitting ratios and objective functions. 3) questions related to the choice between single-path/multipath routing. For the first category, these questions are: How many paths are required to obtain a solution within a few percent of the optimal solution, and whether that number is fixed for any size of the network? Is there a way to select a subset of these paths rather than select all the available paths for each source-destination pair? For the second category, how should traffic flow be split? What is the time complexity of the two well-known traffic engineering objective functions (AD and LB), and whether these two objectives are similar or close to each other? For the third category, how does single-path/multi-path routing affect the network's performance? Is there a difference in time complexity when solving a single-path case over solving it with a multi-path case?

1.5 Contributions

Critical contributions in this dissertation are as follows:

1. We present a routing scheme (RACKE+AD) that outperforms current state-of-the-art techniques.

2. We introduce a new efficient TE simulator that can test many routing schemes simultaneously. The new simulator is optimized for testing different route selection algorithm and objective function combinations and can be easily extended to test future TE systems.

3. We demonstrate that a TE system with static routes and adaptive traffic splitting offers many benefits, including throughput, latency and resource utilization.

4. We show the effectiveness of multi-path routing over single-path routing and their impact on network performance.

5. We propose two DL models, DNN (MLP) and DNN (LSTM), for the routing problem. The models learn traffic split ratios obtained from the optimal solutions as a result of solving the routing problems using LP. Furthermore, we test the trained models in a TE simulator and report the gap in throughput between the optimal LP solution and the solution we get from the trained model.

6. We compare the performance of the two proposed DL models. The result confirms that LSTM neural network performs better than MLP.

1.6 Dissertation Structure

Chapter 2 contains a review of previous TE techniques. We discuss optimal routing and why it is not applicable in practice. We categorize TE systems and focus on the drawbacks of these systems.

Chapter 3 contains the design and analysis of our new traffic engineering simulator. We use this simulator for all our experiments throughout the dissertation. Furthermore, the simulator is used to prepare training data for the proposed deep learning models.

Chapter 4 contains the proposal of a new TE algorithm, RACKE+AD. The proposed algorithm is compared with state-of-the-art traffic engineering algorithms, and the results are presented.

Chapter 5 the proposed algorithm in chapter 4, though close to optimal, comes with a drawback that makes the system unresponsive. In this chapter, we propose two deep learning models to make the proposed system responsive.

Chapter 6 contains the conclusions and a discussion of possible future directions.

2 Review of Literature

2.1 Introduction

This chapter presents a review of some proposed TE solutions. We start by discussing the optimal routing, and why, though optimal, it cannot be applied in practice. Next, we discuss the conventional TE systems and the drawbacks of using these systems in section 2.3. In section 2.4 we highlight the importance of careful path selection and the bottleneck caused by the shortest path routing. Next, we discuss the oblivious routing model and why it cannot be used in practice. Some of the centralized TE approaches are discussed in section 2.6. Finally, we discuss another approach for optimal routing (hop-by-hop forwarding) and how it facilitates the utilization of machine learning techniques for the routing problem (section 2.8).

2.2 Optimal Routing

The textbook's approach [8, 9] for the TE problem is to formulate and solve it as a linear program (LP), the problem referred to as a multi-commodity flow problem (MCF). Usually, the used objective function is to minimize the maximum congested link in the network - Maximum Link Utilization (MLU), also known in the literature as load balance (LB) objective function. The approximated average delay (AD) objective function is also used but not as widely as the LB objective. This approach raises two problems from both sides the optimization side and the production network side. MCF approach considers using all the simple paths (decision variables) provided as input to the optimization problem, making the problem intractable for large networks. On the production side, there is a limit on the number of routes used in the routing table. Forwarding devices such as routers and switches have a limited TCAM memory. Even when all routes are used in production networks,

routes oscillation [3, 10] resulted by solving the MCF problem can be very high, which raises a significant overhead on routers and switches. Thus, using fewer paths is always preferable to keep the routing table as small as possible and reduce the routing overhead. However, as the case in almost all literature, routing schemes calculated with MCF are used as a benchmark to measure the performance of the proposed solution.

2.3 Conventional TE Systems

The conventional approach involves tuning links weights to find a good routing scheme that can increase throughput or minimize congestion in the network [10, 11], for example, open shortest path first (OSPF) and intermediate-system intermediate-system (IS-IS). It turns out that OSPF may not reach optimal routing because it uses Equal Cost Multi-Path (ECMP), which splits traffic evenly among the available shortest paths without rate adaptation. Another limitation of ECMP is the limited number of used paths because only paths with "equal-cost" are used. Furthermore, optimizing links weights is an NP-hard problem [12].

2.4 Shortest Path Routing

Many studies [4, 13, 14, 15] suggest that to achieve reliable performance, a set of shortest paths should be used in a TE system. Unfortunately, choosing the shortest paths may exacerbate congestion for topologies with high heterogeneity of links capacity. For example, consider Figure 2.1, routers A, B, and C needs to send traffic to router D. If we consider using constrained shortest path first (CSPF) routing, we assume the flows have arrived in the following order (B, A, C), the flow received by C may take the path [C, E, D] and the flow received by A will be forced to take the path [A, B, M, D]. With this CSPF approach, there is no choice left for B because it has to take a path with at least one link being shared

with the other two flows. This has inspired a recent research [3] for thinking of a way for selecting paths that are *capacity-aware* and *diverse*.

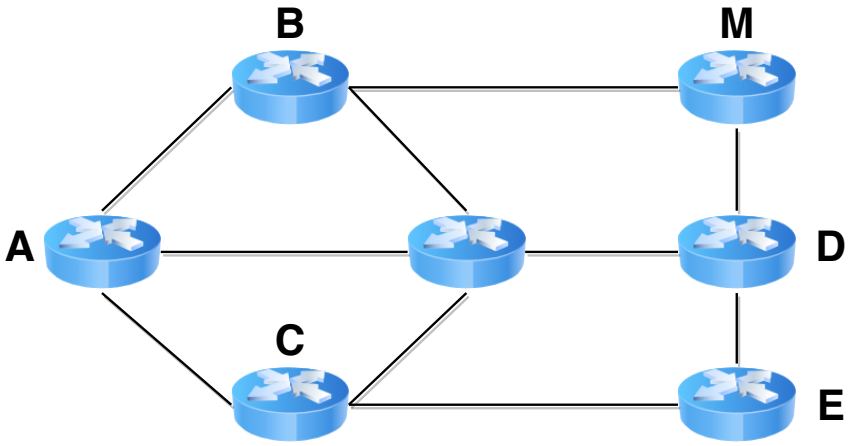


Figure 2.1: A topology of 7 routers

2.5 Oblivious Routing and VLB

Valiant load balancing (VLB) [16] works by routing packets through random intermediate nodes, and these nodes, in turn, forward these packets to the destination. VLB has been shown to work well on mesh-like topologies and has been applied in wide area networks [17]. However, wide area network topologies are not mesh or mesh-like topologies. In addition, VLB may lead to unnecessary utilization of network resources because it may lead to very long paths due to randomness.

Oblivious routing [18, 19, 20] which is a generalization of VLB to non-mesh topologies has also been proposed to find a routing scheme that performs well under all possible demands. The generalization is made by providing a hierarchy of intermediate nodes instead of a single intermediate node. The algorithm iteratively computes a set of randomized routing trees. Each tree is computed using an approximation algorithm [21] that gives paths with lengths close to the shortest paths. In each iteration, the weight of each link is

adjusted based on its cumulative uses in the past few routing trees to ensure no single link is over-utilized. The system is called oblivious because it is oblivious to the demands, and it guarantees a congestion ratio that is never worse than $O(\log n)$ factor of optimal (where n is the number of nodes in the graph). Despite this guaranteed congestion ratio, this approach cannot outperform systems like SWAN [4] due to the fact it considers all possible traffic demands. However, this approach has inspired some researchers [3], including this work, for a "careful path selection".

2.6 Centralized TE systems

Recently, because of the emerging software-defined networking (SDN), a centralized traffic engineering approach came to the picture that decouples the two phases of TE. SWAN [4] distributes traffic over a set of *k-shortest paths* using an LP that reserves a small amount of "scratch capacity" on links to apply updates in a congestion-free manner. SOL [5], similar to VLB [16], uses a greedy approach to select paths randomly in the hope that this random selection might help in load balancing traffic across the network. This approach may lead to unnecessarily long paths and, as a result, increases latency. SOL also uses *k-shortest paths* in a hope to decrease latency.

SMORE [3] is inspired by Räcke's oblivious routing model [20] for selecting paths carefully that would increase the performance and robustness of a TE system. Oblivious routing meets the criteria for a good path selection algorithm. Paths that are selected this way have the feature of *low-stretch*, which is important to decrease latency, and *capacity-aware*, which is important for load balancing. The authors of SMORE suggest that the TE system should be modeled by choosing a static set of paths but with a dynamic adaptation rate. In practice, deleting or adding paths is a slow and expensive operation because this can cause routing overhead. On the other hand, updating splitting ratios is a relatively cheap and fast operation.

2.7 Link-State Hop-by-Hop Routing

In 2011, Dahai *et. al.* proposed a new solution to the optimal routing called PEFT [22, 23] by using a link-state protocol with hop-by-hop forwarding. Their approach was the first that can give an optimal solution using only link-state with hop-by-hop forwarding. They introduced a new way of computing and using link weights in a network. Link weights are *computed* using the gradient descent algorithm and *used* to identify splitting ratios, different than OSPF.

In 2015, Nithin et al. introduced a new approach called HALO [24] that made some improvements to PEFT. Nithin has specified some cases that PEFT couldn't converge to optimal. In contrast to PEFT, which requires full knowledge of the TM, HALO is adaptive and can infer weights based on link flow rates only.

In all of these proposed solutions and another similar work [15], the big achievement is simplicity and optimality with much lower complexity than what we would get by solving the optimal MCF with LP. Solving MCF with link-based formulation has the complexity $O(N^2E)$, where N is the number of nodes and E is the number of links. On the contrary, HALO and PEFT can achieve optimal TE by calculating per-edge weights, and that has $O(NE)$ complexity.

The question remains whether these systems can cause routing overhead and whether they are scalable or not?

2.8 Machine Learning Solutions

2.8.1 Deep Reinforcement Learning

Recently, there have been breakthroughs in the field of AI by combining Deep Learning with Reinforcement Learning (RL), Deep Reinforcement Learning (DRL). One prominent example in [25] where the authors created an agent that would perform different and chal-

lenging tasks of learning to play 49 games on the Atari 2600 platform. Another example in [26] where the agent learns to play the game of Go that has a gigantic search space. In DRL, an agent is responsible for observing which set of actions lead to better performance. The agent sends actions as input and receives observations and rewards as output. The reward serves as feedback to the learning algorithm. From the TE perspective, an input might be a set of selected routes as actions to forward traffic, and an output might be congestion status as observations and the maximum utilized link as a metric (or a reward). Valadarsky *et. al.* [27] has leveraged ideas from machine learning (ML) to solve the routing problem. They showed that applying deep reinforcement learning can give promising results regarding the routing problem. Their idea is based on hop-by-hop routing inspired by PEFT [22, 23] and HALO [24] and utilizes the Trust Region Policy Optimization TRPO [28] algorithm. However, their results showed an optimality gap of about 20% when they set TM sparsity level to 30%. Furthermore, the authors did not investigate the scalability issue which is very critical when designing a routing protocol. They tested their algorithm on 11 nodes which is an easy target. Wide area networks may have more than 25 nodes. Some notable examples of RL/DRL for the routing problem can be found in [29, 30, 31, 32]. There are two main drawbacks with deep reinforcement learning or reinforcement learning when it comes to traffic engineering: (1) it learns to route traffic based on its own experience. Learning from historical decisions, however, does not assure the optimal or near-optimal decisions. According to a study in [33], proof of convergence is not fully and convincingly addressed yet. To ensure an optimal solution, the model may have to rely on optimally labeled data. (2) Another issue with RL is the speed of convergence. According to the same study in [33], further investigation is required to provide the bounded delay of the routing decision made by the RL-based routing algorithm.

2.8.2 Supervised Learning

The work of [34] proposed a sequence-to-sequence deep learning model to predict the forwarding path between each SD pair from historical forwarding experiences. They further used an attention mechanism [35] and beam search [36] to ensure the connectivity and a proper ordering of nodes. However, as mentioned earlier, relying on historical forwarding experiences does not guarantee an optimal solution. In [37], in an approach similar to our approach, they proposed a DL model trained on optimal decisions attained by solving MILP optimization problems to predict the optimal single path between every two nodes. Our approach is different from their approach in that we use multi-path routing instead of a single path, and the TE system we use to learn the model is also different from their system.

In [38, 39, 40] the authors proposed a decentralized deep learning system where each node has several DL models equal to the number of destinations in the network. This decentralized DL model works in a hop-by-hop fashion by predicting the next hop the packets should be forwarded to. All the DL models along the path collaborate to form the path of the packet. The input to these neural networks is the traffic patterns and the output is a 1-hot binary vector, where the 1 indicates the next hop the router should forward traffic to. The drawback in this approach is that each node has to train as many models as the number of destinations. They tested their approach on a small topology, a 4x4 grid; however, only 12 actual nodes were used as edge routers. The other 4 intermediate nodes in the middle serve as forwarding nodes. In addition, they collected their data based on the OSPF routing protocol. OSPF routing protocol requires calculating the shortest path periodically, increasing CPU utilization, and disturbing network services. Furthermore, it has been shown in the literature that utilizing the shortest path may not help minimize the congestion due to many flows competing for the same highly utilized link [3, 41]. Moreover, the work in [38, 39, 40] requires propagating the traffic patterns to all the models periodically, resulting in increased overhead.

In [42] the authors proposed a DL-based distributed routing system that can guarantee

connectivity with the help of the link reversal theory. They have shown that even a small error in the DL model can result in routing loops/blackholes. However, similar to [38, 39, 40], their approach is distributed hop-by-hop routing that has difficulty in achieving optimality. Furthermore, they use the shortest path routing, which may degrade performance because many flows may compete for the same bottle-necked link. They also use the load balancing objective function, which has some performance issue under stressed traffic conditions [41, 43]. In a similar approach in [44], they trained a separate model for each source and destination pair in the network. Furthermore, they use supervised learning with data generated from heuristic solutions, which does not guarantee optimality.

2.9 Multipath/Singlepath studies

Many studies highlighted the importance of multipath in networking (e.g., [45], [46]) while other studies suggested that single-path routing gives the same performance of a multipath routing for a large network under the condition that all nodes are sending and receiving, i.e., competing for the same resources (multi-commodity flow problem) [47] [48]. This lack of certainty gave us the motivation to build a TE simulator and test the gain of multipath routing in different metrics.

2.10 Network Simulators

There are many packet-level simulators for computer networks (e.g., NS-3 [49], OPNET [50] and many others). All of these simulators model the network at a low level of abstraction. Therefore, they are not suitable for answering some of the traffic engineering questions. Moreover, flow-level simulators are much faster than packet-level simulators. The two frameworks that are suitable for simulating TE are YATES [51] and REPETITA [52]. However, one cannot answer our traffic engineering questions with the aid of these

software frameworks. We need a framework that is easy to use and provides flexibility for configuring the network parameters and solving many instances simultaneously (i.e., software supporting the parallel execution of multi-processing).

3 Traffic Engineering Simulator Framework

3.1 Introduction

We built a simulator to model and test different TE scenarios, focusing on efficiency, simplicity, and extendibility. Although many network simulators have been proposed previously [49, 50, 51, 52], they are generally not optimized for modeling TE approaches and/or do not provide ease of use or extendibility.

This chapter presents the design and implementation of our new TE simulator. The TE simulator aims at reducing the cost of experimenting and evaluating different TE scenarios. It offers a set of tools for modeling topologies, traffic matrices, and routing schemes.

The proposed simulator was built in Python and can test many TE models simultaneously while recording statistics in the background. We use Gurobi optimization [7] to solve the linear programming problems, by integrating it with Python. We show the capabilities of this TE simulator by presenting the results of different TE experiments. Furthermore, we compare the efficiency of our simulator with YATES [51].

The TE simulator is designed in such a way that guarantees a fair comparison between different scenarios. It also has a straightforward interface where the user can change different TE scenarios all in one place, a design that made it very simple to perform what-if analysis. Performing these analyses in a simulator (such as ns-3 [49] and OPNET [50]) or an emulator (such as Mininet [53]) requires writing a large code to simulate traffic engineering behavior in WANs. Furthermore, these tools model the traffic at a low level (packet-level), whereas the TE task we need is to model it with a higher level (flow-level). Thus, similar to YATES [51] and REPETITA [52], our TE simulator is a flow-level simulator.

Simulator inputs (e.g., topology, demands, path selection algorithms, objective functions, etc.) are specified in a Python script or configuration file. The simulation produces visualized throughput graphs for each TE system, Fig. 3.1. The graphs are updated pe-

riodically as throughput data becomes available. Three time-series metrics for each TE system are recorded in the background during simulation: overall throughput, congestion per link, and latency per path. Topology and traffic matrices are provided as input files, where the user provides the location to these files in the configuration file. If the locations are unavailable, random topology and traffic matrices will be generated according to provided parameters, including the number of nodes N, the number of links L, and the traffic distribution matrix.

The framework, data and the implementation of some TE algorithms are all available online[1].

[1] https://github.com/MohammedSalman/TE-SIMULATOR

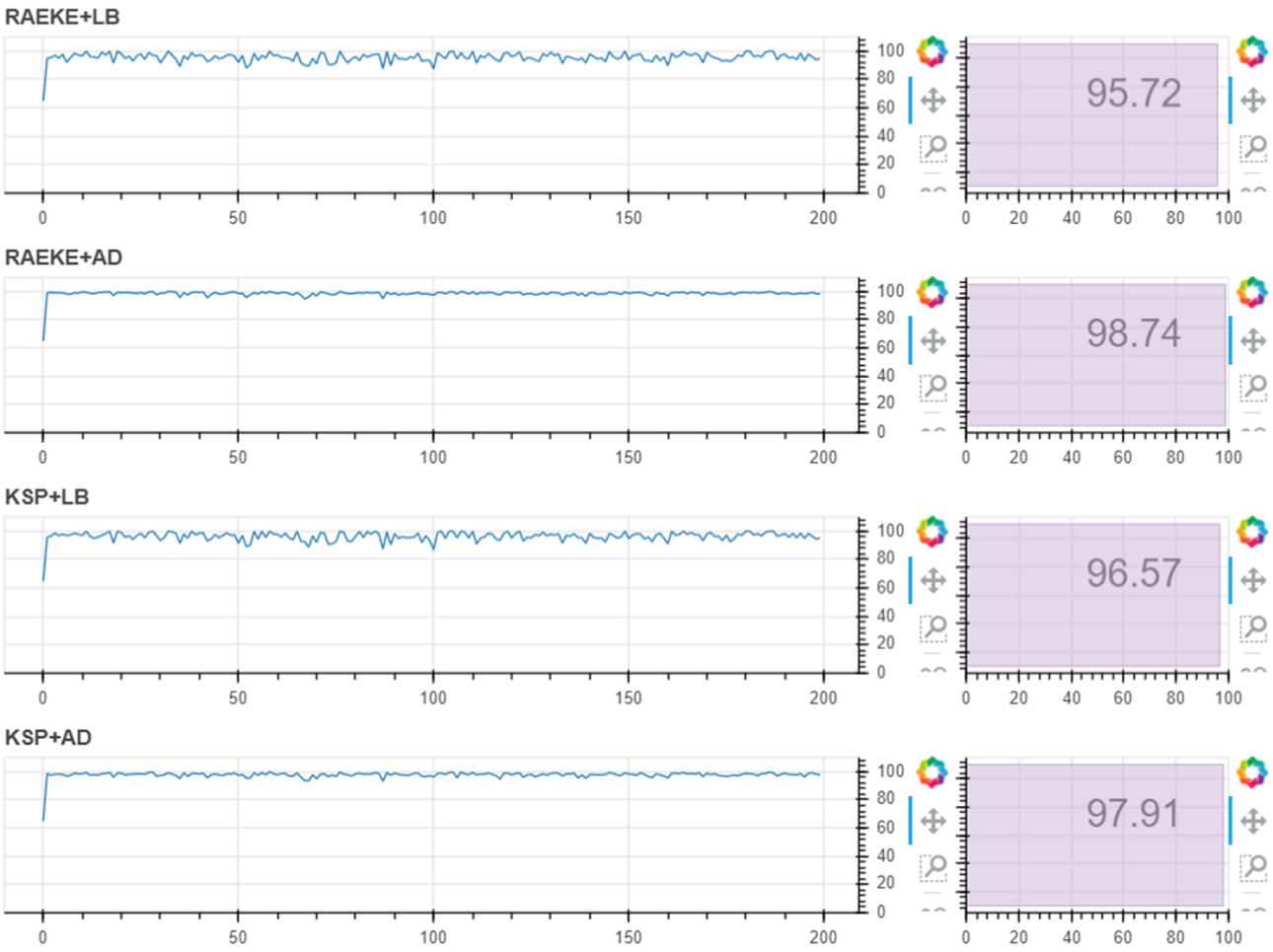


Figure 3.1: A visualized output of four traffic engineering algorithms. The four white windows on the left represent the throughput for each traffic engineering algorithm. The four pink boxes on the right are the moving average of the throughput.

3.2 Motivations

Finding an optimal solution to the TE problem often requires solving a mathematical model (e.g., LP or MILP). Solving a mathematical model, however, can be time-consuming. Furthermore, to keep the system optimal, the LP or the MILP models must be solved frequently whenever the input traffic matrix changes. Thus, regardless of the optimal solutions they give, the system may become unresponsive. Solving many LP and MILP instances in a simulator can take time because the process requires finding a solution per each TE system and per each traffic matrix. To ease the process of benchmarking several TE systems, we aim at building an efficient and flexible TE simulator. To gain efficiency, we design the TE simulator that takes advantage of parallel processing as described in section 3.4. This parallelization is also useful when we generate data to train a deep learning model, as described in Chapter 5.

3.3 TE Framework Overview

Our goal in this work is to develop a software tool to allow the user to understand better the effects of different sets of parameters on TE optimization objectives while avoiding the nuances of low-level optimization formulations, making the testing and tuning of TE more accessible to researchers and students.

Figure 3.2 summarizes the steps of the TE simulator from a different perspective with four main steps. We can think of the simulator as software that formulates many LP or MILP based on the input configuration parameters. The framework takes the topology information and other parameters as an input configuration file (e.g., number of nodes, number of links, traffic matrix models, optimization requirements, and so on). The framework will find different sets of combinations of these input parameters, formulate a proper optimization problem, and treat each one (with a different set of input parameters) as an

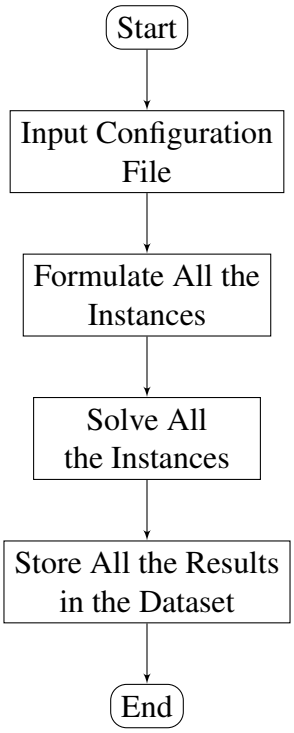


Figure 3.2: Summarization of the developed framework in four main steps.

instance that is ready to be solved to obtain an optimal solution (with the help of an LP solver). The optimization requirement (i.e., TE objective) is also provided as input to the framework. The output from the framework is a dataset that has all the information for each instance of the problem solved by the LP solver. The dataset attained is ready for analysis and visualization. The size of the generated data depends on the number of problem instances based on the input configuration file. The framework is open for extension. Therefore, additional features can be added to generate needed information for TE to be included in the dataset for analysis and visualization.

In this TE framework, the user can define parameters as inputs to establish a network. These parameters include the number of nodes n and the number of links l[2]. The number of nodes and number of links can be disregarded if the user chooses to use a

[2] Anywhere in this dissertation when referring to *number of links* we mean *number of link pairs*. If any two nodes are connected, they are connected in both directions.

topology file instead. For example, Listing 3.1 specifies a set of network parameters in Python. This TE framework creates several topologies by combining N and L and distributing links between nodes using the Erdős–Rényi model. If the resulting topology is disconnected, the framework will continue to create new ones until the topology is connected or stops after attempting a specified number of times and failing to produce a valid network topology. The Erdős–Rényi model can find a connected topology immediately if $l >> n$. Other configuration information that needs to be fed to the system is shown in Listing 3.1. The information includes capacity type, capacity set, weight set, demand traffic matrix model(s), the objective function(s), number of candidate paths, and routing strategies. The example below (Listing 3.1) is an input configuration file that creates 6400 instances of TE problem, each configured differently due to different links distribution and/or different parameter settings. The number 6400 is calculated from $|N| \times |L| \times Nu\_of\_TMs\_Per\_TOPO \times Nu\_of\_TOPOs\_Per\_N\_L \times |TM\_TYPES| \times |OBJECTIVES| \times |CANDIDATE\_PATHS| \times |ROUTING\_STRATEGIES|$. The parameter $Nu\_of\_TOPOs\_Per\_N\_L$ is the number of topologies for each pair of N and L, and the parameter $Nu\_of\_TMs\_Per\_TOPO$ is the number of traffic matrices per topology.

```
N = {30, 40}
L = {80, 100}
Nu_of_TOPOs_Per_N_L = 10
Nu_of_TMs_Per_N_L = 10
capacity_type = {'EDGE_BETWEENNESS'}
capacity_set = {30, 35, 40}
weight_setting = {'INV_CAP'}
tm_types = {'BIMODAL', 'GRAVITY'}
Network_Load = [0.4]
objectives = {'LB', 'AD'}
candidate_paths = {3, 7}
routing_strategies = {'MULTIPATH', 'SINGLEPATH'}
```

Listing 3.1: An input configuration file.

3.4 Design and Implementation

Efficiency is the main concern when designing this software framework for the following reasons. First, one unsolved instance of network design problem formulated as LP/MILP may have more than one million decision variables. Therefore, the solution space can be vast and impossible to solve in real-time. These large number of decision variables may result from many paths between any pair of nodes in the network. The number of paths grows exponentially as more nodes or links are added to the network. Second, the MILP problem is more complex than the LP problem due to the extra binary variables introduced to the problem. We tackle these problems from three aspects. First, we need to reduce the number of decision variables in the problem formulation. Second, we need to use a fast LP/MILP solver. To the best of our knowledge, the fastest LP/MILP solver is Gurobi [7]. Compared to the default solver used in PuLP [54], an optimization modeler in Python,

Gurobi is a lot faster. Third, we need to come with a design so that these formulations can be solved in parallel.

When designing a network, using all the available paths between SD pairs is unnecessary because the network uses shorter paths. Thus, longer paths can be excluded without significantly affecting optimality, as explained in Section 3.3. As a result, the solution space can be shrunk using fewer decision variables when using only a few shortest paths per SD pair.

The architecture of our design consists of three main steps (Figure 3.3). These are *settings*, *routing scheme formulation*, and *simulator*. The setting component consists of three main inputs to the TE algorithm: topology, traffic matrices, and routing configurations. The topology contains information about nodes and how these nodes are connected and links' information such as bandwidths and weights. The traffic matrices object is a set of traffic matrices where each traffic matrix incorporates data about sources, destinations, and demand volumes. The routing configuration is a set of parameters that specify how the routing is done, for example, the number of paths used and whether to forward traffic on a single-path or multi-path. The last concept in the architecture is the *simulator*. In this step, two main operations took place: the simulation and metrics calculation. The simulation is the actual allocation of traffic performed based on the routing scheme of the given TE algorithm. The metrics calculation is where the calculation of throughput, latency, and links utilization, is done.

Figure 3.4 shows the design diagram that is like a tree structure but with leaves join at the end, which makes it easier to exploit execution in parallel because each object created is independent of other objects in the same module. In this simulator, we try to exploit the execution in parallel whenever two or more independent operations exist. As depicted in the design diagram, the main process has three forks operations with their companion joins operations. The first fork is for each traffic matrix (TM). The second fork is when both options of single-path and multi-path routing are provided. The third fork is for each

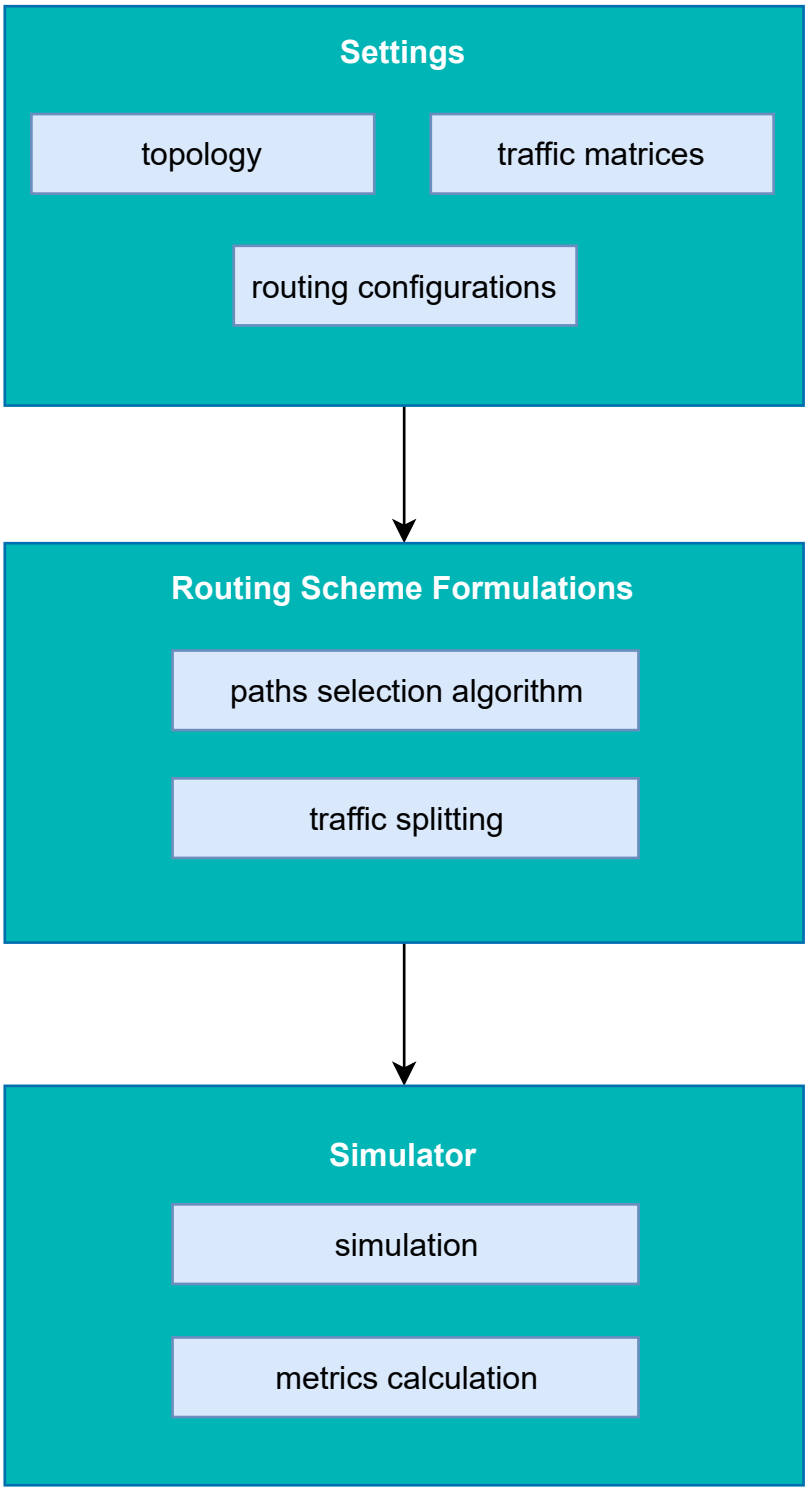
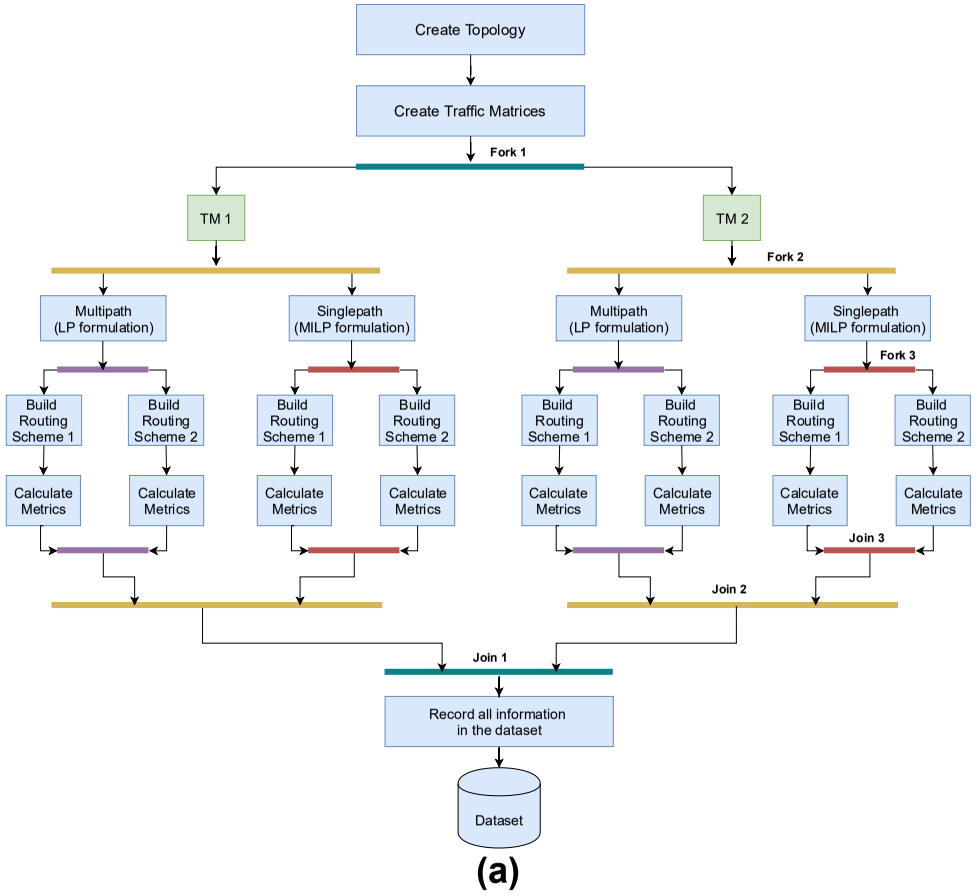


Figure 3.3: TE simulator architecture showing the three main concepts.

resulted routing scheme from the previous configuration, followed by traffic allocation and metrics calculation. These child processes are not executed all at once; instead, they are executed in chunks according to the number of CPU cores available in the system to achieve better computational efficiency.




```
N = {25}
L = {37}
Nu_of_TOPOs_Per_N_L = 1
Nu_of_TMs_Per_N_L = 2
capacity_type = {'EDGE_BETWEENNESS'}
capacity_set = {30, 35, 40}
weight_setting = {'INV_CAP'}
tm_types = {'GRAVITY'}
Network_Load = [0.5]
objectives = {'LB'}
candidate_paths = {3}
routing_strategies = {'MULTIPATH', 'SINGLEPATH'}
```

(b)

Figure 3.4: (a) Design diagram showing the four main modules. (b) The corresponding configuration file.

3.5 Configuring Links Capacities and Links Weights

Currently, the software tool implements the **'edge_betweenness'** [55] method of configuring link capacity. This model takes the capacities (CAPACITY_SET) as input and distributes them based on metrics that capture the importance of links and nodes. In [55], three models were suggested for this task, and these are edge betweenness centrality, degree centrality gravity, and communicability centrality gravity. The one that we use is edge betweenness centrality. Edge betweenness centrality is an important metric that measures the "criticality" of a link, or a node [56]. For example (see Figure 3.5), the link (2, 3) is configured the highest capacity, which is 40 because this link is the only link that connects node 2 to all other nodes in the network. Likewise, one can see why the capacities are assigned for the other nodes, as shown in the figure. However, if the topology is provided to the simulator as an external file, link capacities will be taken from that file.

As suggested by Cisco, links' weights can be set proportionally to the inverse of link capacity by setting the parameter (WEIGHT_SETTING) to 'INV_CAP'. For example (see Figure 3.5), these weights are used to calculate the shortest paths for any pair of nodes in the network. Link weights can also be set 'FIXED'; in that case, shortest paths will be calculated based on hop counts.

3.6 Path-based vs. Link-based Formulation

We believe that expressing the network optimization problems using path-based (a.k.a Link-Path) formulation is more beneficial than casting them using link-based (a.k.a Node-Link) formulation (even though we study the link load in some parts of our study) for the following reasons: First, our main purpose of this work is to study the behavior of the network in terms of paths that are selected to attain performance optimality or near-optimality. Second, it is normally easier for application developers and network operators

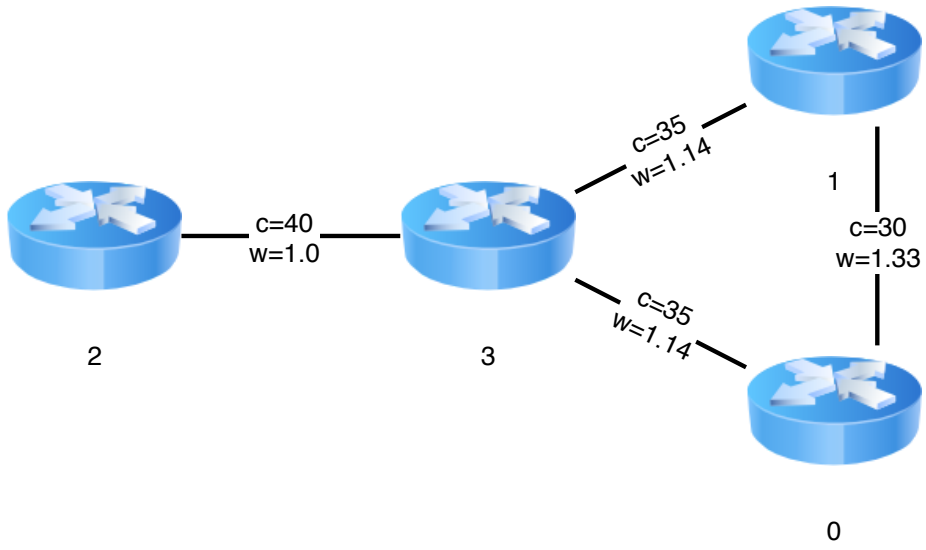


Figure 3.5: A four-node topology shows how the weights are configured proportionally to the inverse of the link capacity.

to think about paths instead of links. Finally, it is easy to model the routing table in an SDN switch based on traffic carried on paths that traverse that switch. Furthermore, key policy requirements for a spectrum of applications are expressed in terms of paths. For example, traffic engineering and service chaining [5]. All the TE objectives can be expressed using the path-based formulation. Bi *et. al.* [57] mentioned in their study that path cardinality constraint (i.e., the number of paths allowed) greatly affects the attainable performance of a given routing scheme and the actual implementation complexity. For all the reasons above, we use path-based over link-based formulation. However, we do not use the full set of available paths between each ingress-egress node, except when compared with the optimal routing. In fact, to reach optimality or near-optimality, we only need no more than 5 paths, as we will show later. The use of a full set of available paths will introduce two efficiency problems (since the number of paths grows exponentially as the size of the network increases). First, calculating all of these paths may take a long time. Second, using all the available paths will increase the number of variables in the LP formulation, increasing the size and complexity of the optimization problem.

3.7 Multi-path vs. Single-path

Although the LP and MILP problems in networking design optimization can be mitigated by omitting unnecessary long paths, the problems are still hard to solve if the traffic flows are unsplittable and only one single path is permitted for each traffic demand. The single-path constraint introduces discreteness (discrete constraints) into the optimization problem. Additional binary variables are introduced to the problem to decide whether the path per traffic demand should be used or not. Solving the single-path problem for small topologies should not require a significant amount of time, especially with the high-performance computation power that we have today. Still, when we start increasing the number of nodes and links, the problem becomes intractable. The simulator provides both of these routing options.

3.8 ISP Topologies vs. Arbitrary Topologies

Topologies can be provided as an external file or be generated randomly. It is important to consider arbitrary topologies rather than being limited to existing well-known ISP topologies. One reason is that we may want to study the characteristics of TE no matter what the topology is. The use of available ISP topologies may restrict us from fully understanding our TE techniques.

3.9 Traffic Matrix Models

In all our experiments, each node sends to and receives from all other nodes in the network. Three models are implemented to generate traffic matrices (i.e., Nucci Heuristic [58], Gravity [59], and Bimodal [60]). These models can be chosen using the parameter TM_TYPES. The generation of the Nucci Heuristic is based on the work proposed by Nucci et al. [58],

which comprises three steps to generate the traffic matrix. One common parameter (Max_u) needs to be set to represent the target maximum link utilization used for scaling traffic volumes for all three models. The bimodal model [60] assumes that only a few ingress-egress pairs have substantial flows and then assign demands for these pairs uniformly. The gravity model [59] is based on a capacity-based heuristic (also used in [10], [18]) which assumes that each demand for each node is proportional to the combined capacity of connecting links.

The TE framework will then translate all of the inputs above into low-level constraints using Gurobi, the mathematical programming solver [7]. Finally, the framework stores the results in a dataset. Table 3.1 shows the information the framework may produce.

Table 3.1: Description of the information produced by the framework

Parameter	Type	Description
n	discrete	Number of nodes used
l	discrete	Number of links used
Avg_nodal_degree	discrete	Average nodal degree
Network_load	discrete	Network loads (traffic scales) used in the traffic matrix
TM_TYPE	discrete	Traffic matrix generator models (‘lognormal’, ‘gravity’, ‘bimodal’)
Obj_type	discrete	Objective function used
Obj_val	continuous	Optimal or near-optimal value of the objective function
k	discrete	Number of candidate paths used
Routing_strategy	discrete	Multipath and/or Single-path
Residual_cap	continuous	Percentage of available capacity after solving the problem instance
Links_utilization_and_residual	dictionary data structure	Nested data structure for storing information about all the links in each instance
Flow_agg_per_path	dictionary data structure	Nested data structure for storing information about all routes in each instance

3.10 Supported Traffic Engineering Models

We consider a multi-commodity network flow problem (MCF) and a directed communication network $G = (N, L)$ where N is the set of nodes and L is the set of links in graph G. Each link l has a capacity $c \geq 0$ for each $l \in L$. A set of demands is denoted as D, with traffic volume h_d, $d \in D$. We consider the case when all nodes in the network send and receive traffic to/from all nodes in the network and only when the system is feasible to accommodate the demands. All the TE models will be expressed using the path-based (a.k.a. link-path) formulation [8].

A TE model takes the network topology and traffic demands as input and finds how much traffic must flow on path p as output for each source-destination (SD) pair. In case of multi-path routing, the goal is to find the optimal set of flow variables $x = (x_{dp} : d \in D, p \in P_d)$ where P_d is the set of paths for demand d. In a single path case, the goal is to find one single path for each SD pair that must be used to achieve specific optimality. The goal of TE is to traffic engineer paths to ensure good performance in terms of an optimization objective (e.g., minimizing the delay) and efficient use of network resources (e.g., load balancing). In our framework, the implemented objective functions include load balancing (LB) and average delay (AD).

In the following, we briefly introduce these objective functions and problem formulations.

3.10.1 Objective Functions

Load Balance

The load balance (LB) objective is also known as minimizing MLU, Wozencraft objective [9], or minimizing congestion, where LB minimizes the load on the most congested link. Thus, the LB problem can be expressed as [61]

$$\min \quad F(x) = r \tag{3.1}$$

$$\text{s.t.} \quad \sum_{p \in P_d} x_{dp} = h_d, \qquad d \in D \tag{3.1a}$$

$$\sum_{d \in D} \sum_{p \in P_d} \delta_{dpl} x_{dp} \leq c_l r, \qquad l \in L \tag{3.1b}$$

where: x_{dp} is the flow on path p for demand d; h_d is the volume for demand d; c_l is the capacity for link l; P_d is the number of candidate paths for demand d; δ_{dpl} = 0, 1 is a link-path indicator, with δ_{dpl} = 1 if path p for demand d uses link l, and 0 otherwise.

Two constraints are applied. The demand constraint (3.1a) ensures that all demands are satisfied over some paths. The capacity constraint (3.1b) ensures that load does not exceed the link capacity where $r \leq 1$, after solving (3.1). The linear program formulation above is the final form of the problem, whereas the original problem is non-linear. The reader is referred to Chapter 4 of [61] for details on how the problem can be converted to the current form.

Average Delay

For this objective function, delay for any network link can be modeled as $y/(c - y)$, as shown in (Figure 3.6, solid line). Similar to the LB objective, the original AD problem is non-linear and cannot be formulated directly as a linear program. Thus, the delay function is a piecewise linear approximation (3.2) (Figure 3.6, dotted line)

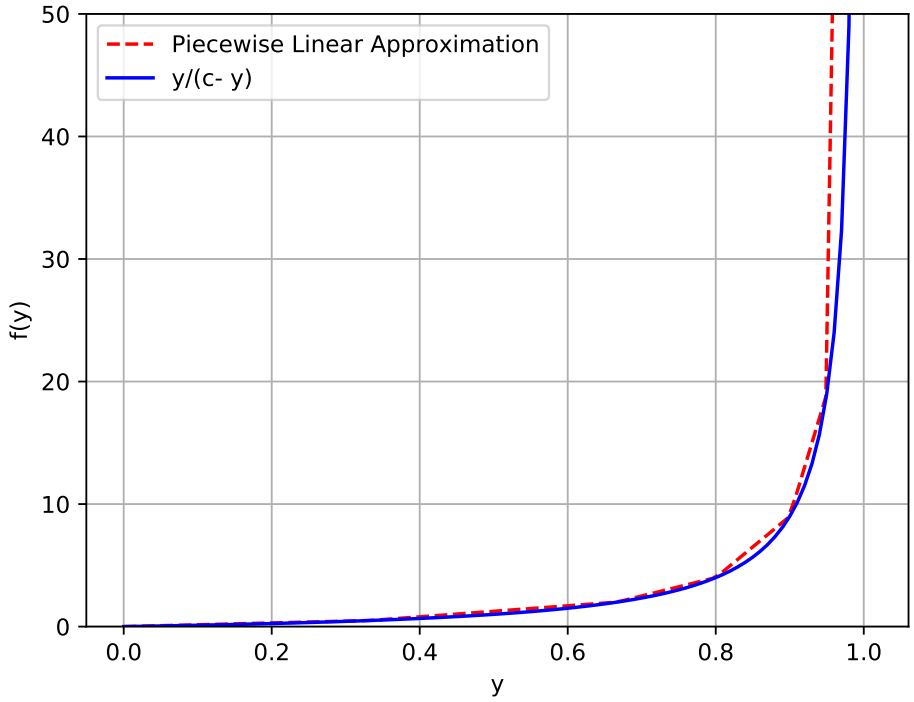


Figure 3.6: Piecewise linear approximation of the delay function.

$$g(z) = \begin{cases} (3/2)z & \text{for } 1 \le z < 1/3 \\ (9/2)z - 1 & \text{for } 1/3 \le z < 2/3 \\ 15z - 8 & \text{for } 2/3 \le z < 4/5 \\ 50z - 36 & \text{for } 4/5 \le z < 9/10 \\ 200z - 171 & \text{for } 9/10 \le z < 19/20 \\ 4000z - 3781 & \text{for } z \ge 19/20 \end{cases} \tag{3.2}$$

The linear program for this AD problem is

$$\text{min} \qquad F = \sum_{l=1}^{L} \frac{r_l}{c_l} \tag{3.3}$$

$$\text{s.t.} \qquad \sum_{p=1}^{P_d} x_{dp} = h_d, \quad d = 1, 2, ..., D \tag{3.3a}$$

$$\sum_{d=1}^{D} \sum_{p=1}^{P_d} \delta_{dpl} x_{dp} = y_l, \quad l = 1, 2, ..., L \tag{3.3b}$$

$$r_l \geq \frac{3}{2} y_l, \quad l = 1, 2, ..., L \tag{3.3c}$$

$$r_l \geq \frac{9}{2} y_l - c_l, \quad l = 1, 2, ..., L \tag{3.3d}$$

$$r_l \geq 15 y_l - 8 c_l, \quad l = 1, 2, ..., L \tag{3.3e}$$

$$r_l \geq 50 y_l - 36 c_l, \quad l = 1, 2, ..., L \tag{3.3f}$$

$$r_l \geq 200 y_l - 171 c_l, \quad l = 1, 2, ..., L \tag{3.3g}$$

$$r_l \geq 4000 y_l - 3781 c_l, \quad l = 1, 2, ..., L \tag{3.3h}$$

$$x_{dp} \geq 0, \quad p = 1, 2, ..., P_k, d = 1, 2, ..., D \tag{3.3i}$$

$$y_l \geq 0, \quad l = 1, 2, ..., L \tag{3.3j}$$

which is considerably more accurate [61] than the Fortz et al. [11] approximation.

3.10.2 Path Selection Algorithms

Räcke's oblivious routing model

Räcke's oblivious routing model iteratively computes a distribution over randomized routing trees using an approximation algorithm. Link weights are adjusted for each iteration based on how much the link has been utilized in previous routing tree sets. A routing tree has leaves corresponding to nodes in the original topology. Thus, a path can be obtained be-

tween nodes u and v in the original graph by finding corresponding leaves for u and v in the routing tree. Paths for Räcke's oblivious routing model are computed without considering demands; thus, they do not overfit to a specific scenario [3].

The Räcke's oblivious routing algorithm computes a full and fixed routing scheme that does not change over time but guarantees a congestion ratio as explained in Section 2.5. If a path budget is specified for this model in the simulator configurations, the simulator will only take paths with the highest weights and according to the number of paths specified, the rest of the precomputed paths will be eliminated.

K-shortest paths (KSP)

The KSP algorithm is based on Yen's algorithm, the most commonly used algorithm for TE. KSP is a generalization of the shortest path routing problem. The algorithm returns loopless k shortest paths ordered from shortest to longest.

Selecting All Paths

For the sake of comparison with the optimal solutions that rely on all available paths per SD pair, the simulator provides the option of using all available paths in the system. However, not all paths might be used as this depends on the objective function and current traffic patterns.

3.11 Running Time Analysis

This section compares the running time of the TE simulator with the YATES traffic engineering simulator. We use two topologies, ATT North America and Abilene. The information of these two topologies is given in Chapter 5. For each run, we use 200 traffic matrices. We use the same settings for both runs for a fair comparison: traffic matrices, number of paths, objective function. The result (Fig. 3.7) confirms that our TE simulator is

4-6 times faster than the YATES simulator. We attribute this performance to the use of parallel processing in the TE simulator. Another reason is that we consider the TE simulator as a lightweight version of YATES with the same functions; for example, the TE simulator does not have an SDN backend similar to the backend in YATES.

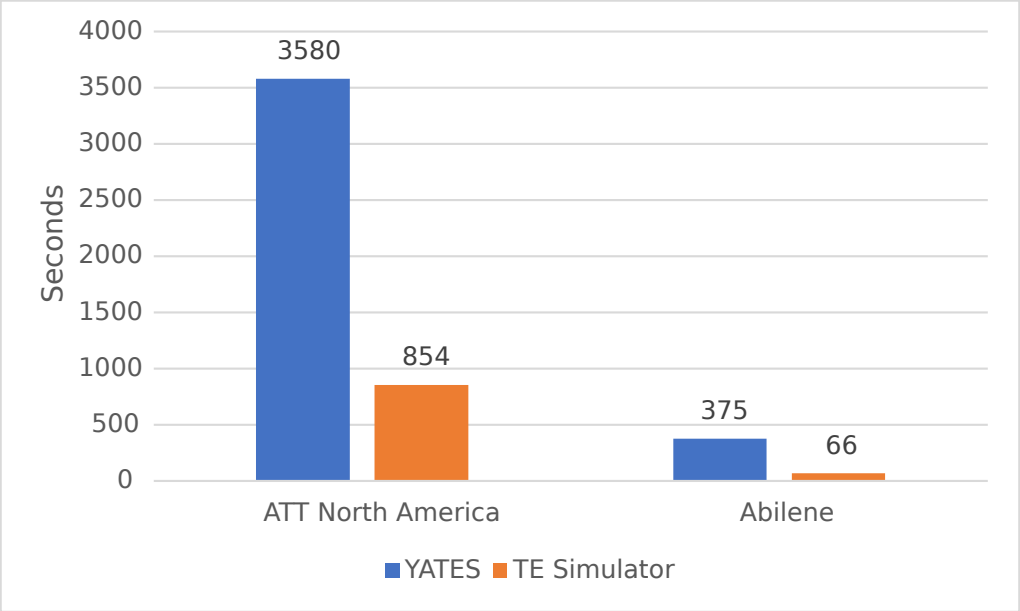


Figure 3.7: Running time comparison between YATES and TE simulator.

3.12 Experimenting with Traffic Engineering

We present two case studies that focus on path budget and routing strategies (single-path/ multipath routing). Section 3.13 is on path budget while section 3.14 is about routing strategies.

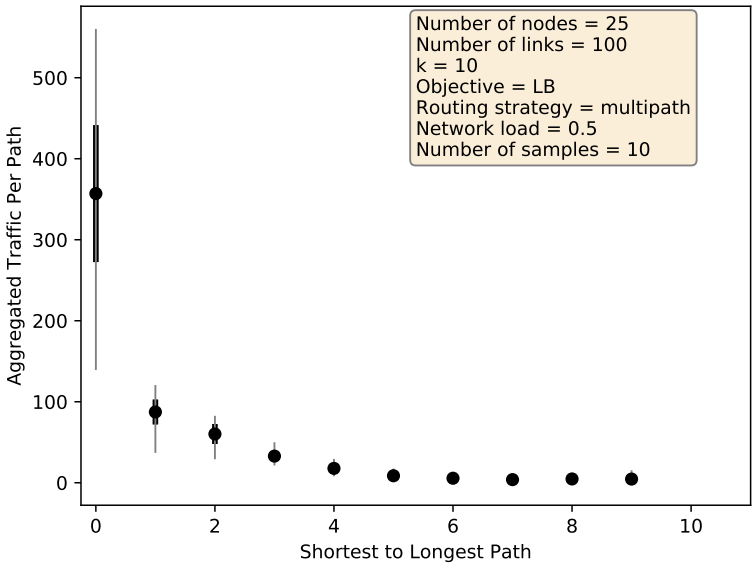
3.13 Path Budget

This section gives one example to illustrate a notion about path budget for networks of the same size. Then we generalize the example for networks of different sizes to depict a

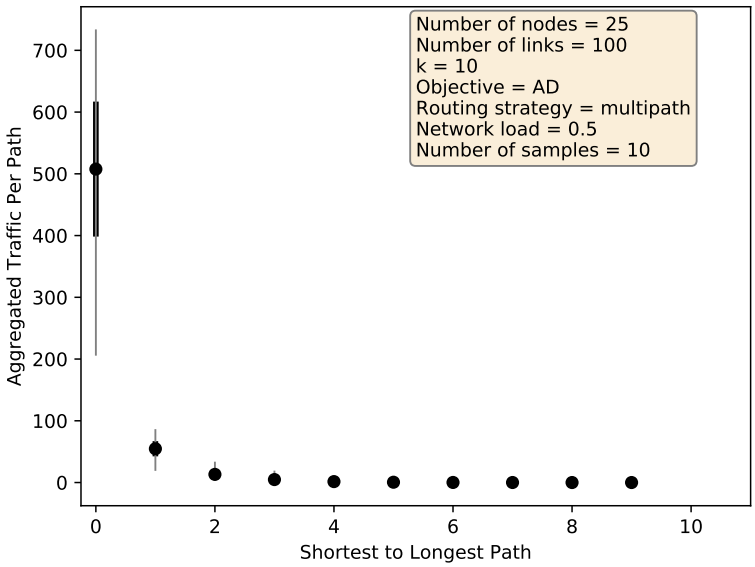
relationship between the nodal degree of the network and the path budget.

3.13.1 Path Budget in Networks of Fixed Size

The number of paths in a network grows exponentially as the size of the network increases. Thus, it is not advisable to select all of the paths in the network as this will make it very hard to solve the LP/MILP formulations in a reasonable amount of time. Heorhiadi *et. al.* [5] and Ayyub *et. al.* [6] suggest using one of two strategies to deal with the increase of paths depending on the application, random paths for load balancing applications and shortest paths for latency-sensitive applications. Some researchers ([22], [62]) were able to achieve the optimal or near-optimal solution by giving priority to the shortest paths and directing less traffic on non-shortest paths with an exponential penalty on longer paths. Their work gave us the intuition to test the relationship between paths and their traffic ratios. We show the effectiveness of this strategy through extensive experiments and visualization of the datasets produced. Figure 3.8 shows the aggregated traffic flow across ordered paths (ordered according to their costs from the lowest to the highest). In this experiment, we fix the number of nodes as (25) and the number of links as (100) and vary only the distribution of links among nodes (traffic matrix will differ as a result). We repeat the experiment 100 times ($Nu\_of\_TOPOs\_Per\_N\_L = 100$) for each objective and then produce all the sub-figures (3.8(a), 3.8(b)). The X-axis represents the chosen number of candidate paths (ordered from the shortest path to the longest path), while Y-axis represents the aggregated traffic flow per path. The highest point in each line is the maximum aggregated traffic volume, while the lowest point is the minimum aggregated traffic volume. The bold dots represent the average of the aggregated traffic volumes for each path, while the bold lines represent the standard deviation. In this experiment, we consider the two objectives for TE (i.e., LB, AD). Notice how the need for the next path exponentially decreases.



(a) LB



(b) AD

Figure 3.8: Aggregated traffic flow for 100 networks across ordered paths (from the shortest to the longest).

```
N = {25}
L = {100}
Nu_of_TOPOs_Per_N_L = 100
Nu_of_TMs_Per_N_L = 10
capacity_type = {'EDGE_BETWEENNESS'}
capacity_set = {30, 35, 40}
weight_setting = {'INV_CAP'}
tm_types = {'GRAVITY'}
Network_Load = [0.5]
objectives = {'LB', 'AD'}
candidate_paths = {10}
routing_strategies = {'MULTIPATH'}
```

Listing 3.2: Input configuration file used to generate results in Fig. 3.8.

3.13.2 Path Budget in Networks of Different Sizes

The previous experiment reflects the needed number of paths to stay close to the optimal. However, we would like to know whether this range of path budget is always the same for networks of different sizes. To find out, we conduct another set of experiments with different network sizes so that the topologies used reflect the relationship between the nodal degree and the required number of paths. Thus, we present the optimality gap (Figure 3.9) that visualizes the percentage gap between two problem instances with the same settings except varying the number of candidate paths. This visualization as given in Figure 3.9 (where k = {3, 7}, the objective function is LB, $Nu\_of\_TMs\_Per\_TOPO = 100$) shows the average gap in optimality (3 paths against 7 paths) over 100 experiments for each instance. Each value represents the average gap between the objective value of the maximum k ($k = 7$) and the objective value of the minimum k ($k = 3$). The X-axis represents the number of nodes N while the Y-axis represents the number of links L (which can be

thought of as the nodal degree). The gap is calculated as $((opt1 - opt2)/opt1) \times 100$, where $opt1$ is the optimal value when $k = 3$ and $opt2$ is the optimal value when $k = 7$. The figure made it very clear that it is not recommended to use 3 routes as candidate paths when the number of nodes is small and the number of links is large for the LB objective, i.e., when the nodal degree is large. Choosing the number of candidate paths (k) depends on the size of the network, nodal degree, and the objective function. Fortunately, for all the Internet Service Provider (ISP) networks that we know today, the largest nodal degree is no more than 6.0 [47]. It is worth mentioning that calculating all the instances shown in Figure 3.9 takes a lot of time. This is because of the large number of instances that have to be solved (one for each k), and all are repeated by the $Nu\_of\_TMs\_Per\_TOPO$ times. We have used one computing node with a 40-core CPU in an HPC system to produce Figure 3.9 in about 8 hours.

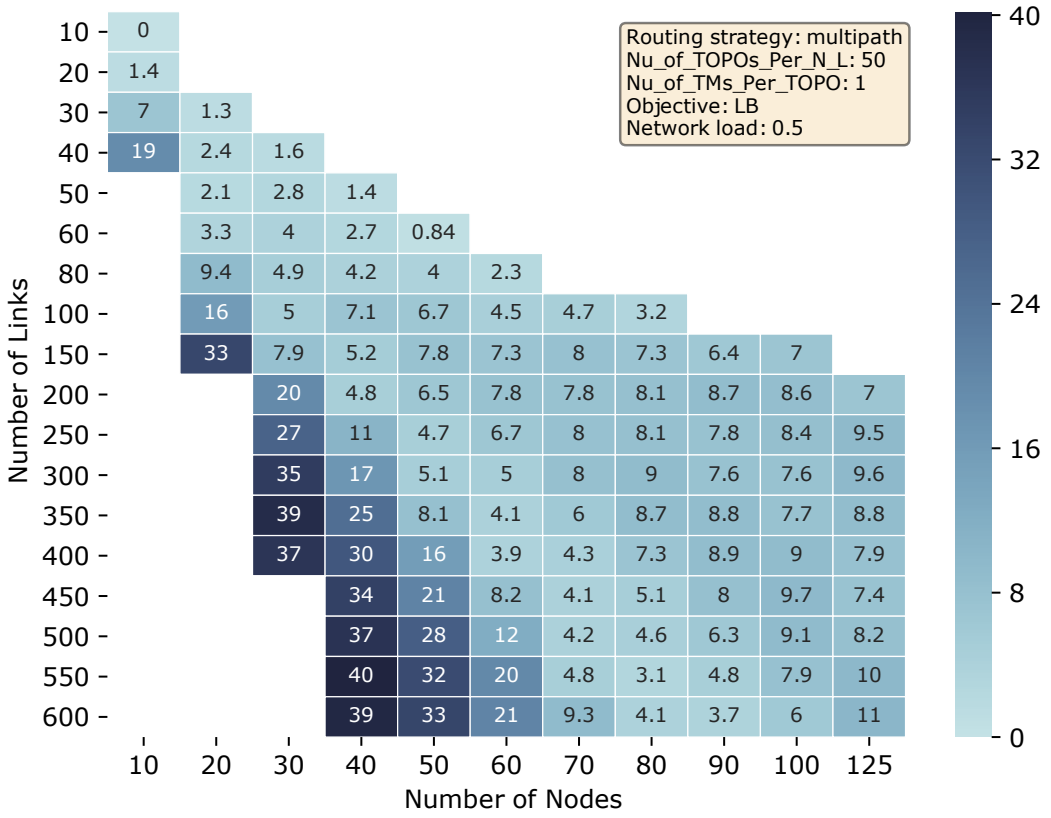


Figure 3.9: Average gap in LB objective for 100 instances with $k = 3$ and another time with $k = 7$ where k is the path budget.

```
N = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100,
    125}
L = {10, 20, 30, 40, 50, 60, 80, 100, 150,
    200, 250, 300, 350, 400, 450, 500, 550,
    600}
Nu_of_TOPOs_Per_N_L = 100
Nu_of_TMs_Per_N_L = 10
capacity_type = {'EDGE_BETWEENNESS'}
capacity_set = {30, 35, 40}
weight_setting = {'INV_CAP'}
tm_types = {'GRAVITY'}
Network_Load = [0.5]
objectives = {'LB', 'AD'}
candidate_paths = {3, 7}
routing_strategies = {'MULTIPATH'}
```

Listing 3.3: Input configuration file used to generate results in Fig. 3.9

3.14 The Effect of Multi-path and Single-path on Network Utilization

Because our framework can also record link information, we study the effect of multi-path/single-path routing on link utilization. We present one example in one network and then look at performance in networks with a different number of links. We notice that there is a difference in utilization between multi-path and single-path approaches. Thus, we calculate the percentage of available capacity in the network as $R/C \times 100$ where R is the summation of available residual capacities of all links, and C is the summation of capacities of all links.

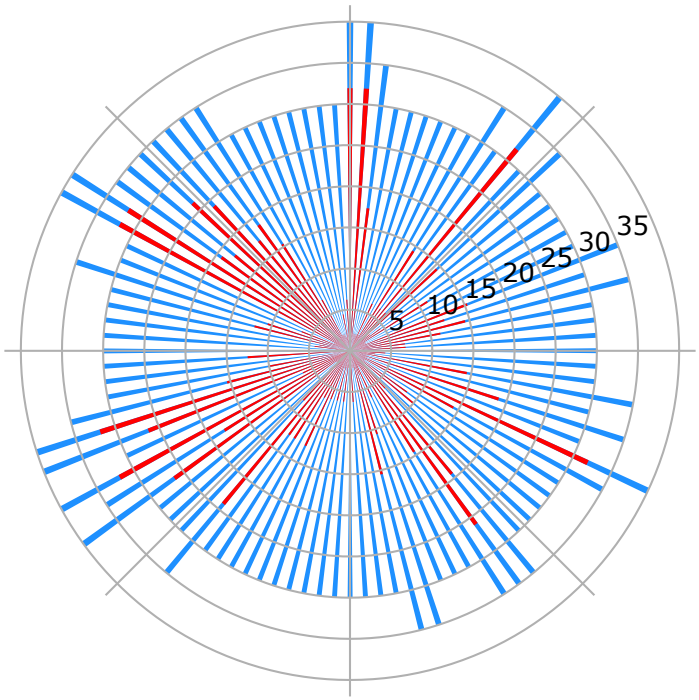
3.14.1 Result in One Network

Studying the benefit of multi-path routing from only one perspective, e.g., comparing the value of the objective function is not sufficient. For example, the LB objective, which minimizes the maximum congested link in the network, does not reflect the status of all links in the network. Instead, it reflects the maximum congested link and only that link. Multi-path routing still has the advantage over single-path routing when it comes to utilization.

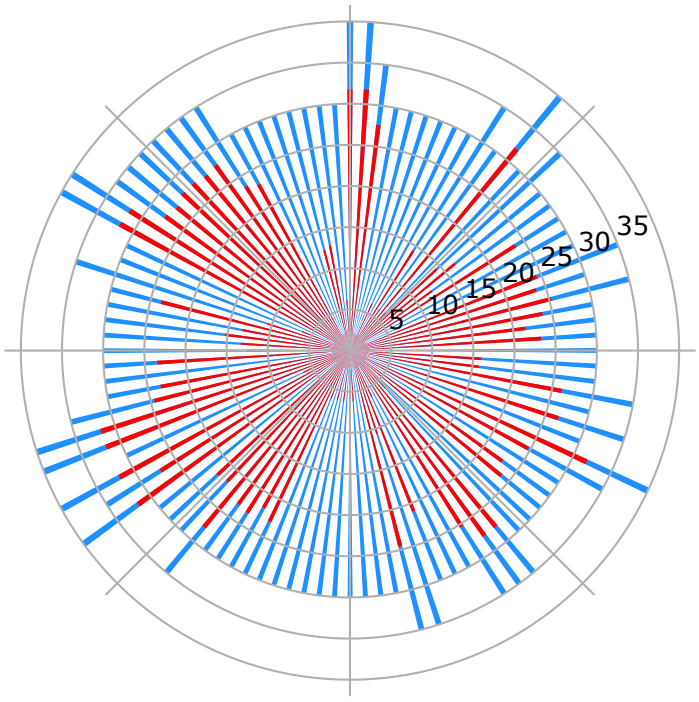
```
N = {25}
L = {100}
Nu_of_TOPOs_Per_N_L = 1
Nu_of_TMs_Per_N_L = 1
capacity_type = {'EDGE_BETWEENNESS'}
capacity_set = {30, 35, 40}
weight_setting = {'INV_CAP'}
tm_types = {'GRAVITY'}
Network_Load = [0.5]
objectives = {'LB'}
candidate_paths = {4}
routing_strategies = {'SINGLEPATH', 'MULTIPATH'}
```

Listing 3.4: Input configuration file used to generate results in Fig. 3.10.

In this example, we show how multipath/single-path routing has an impact on the residual capacity as opposed to what was claimed in [47]. It turns out, in most cases, multi-path routing utilizes links better than single-path routing. Figure 3.10 shows links utilization for the multi-path case (Figure 3.10(a)) and for single-path case (Figure 3.10(b)) for the same network with the same configuration. In this particular example, we see that single-path routing uses more links (with higher utilization) than multi-path routing uses,



(a) Multi-path case



(b) Single-path case

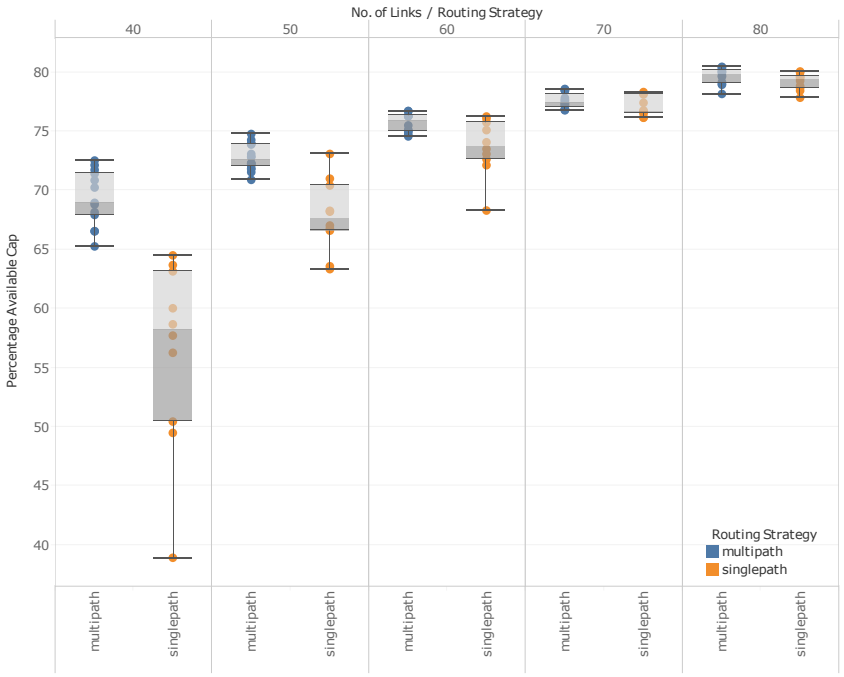
Figure 3.10: A visualization that shows link load (in red) and link capacities (in blue). The single-path case utilizes the network resources 16.25% more than the multi-path case.

even though the value of the objective function is the same for both cases. For the multi-path routing case, the residual capacity is 67.16%, while it is 50.91% for the single-path routing case, a difference of about 16.25%.

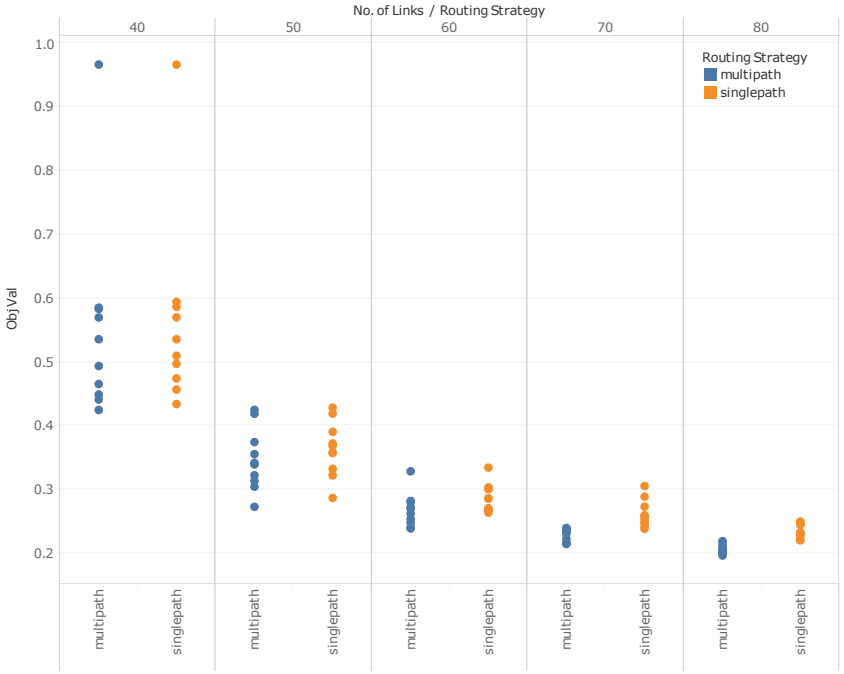
3.14.2 Result in Many Networks

We would like to see if the observations of the last section apply to any networks. In this example, we fix the number of nodes and vary only the number of links. We see the effect of increasing the number of links on the objective function and the residual capacity (Figure 3.11).

We have found a relationship between the type of routing strategy (single-path/multipath) and the number of links regarding residual capacity. In some experiments, we notice that single-path routing over-utilizes the network resources by a significant amount. Figure 3.11(a) shows an over-utilization gap over 30% for all topologies when the number of links is 40. This gap reduces when the number of links increases. Thus, multi-path routing is recommended when the number of links is not large. It is worth noticing that the objective value does not change that much when using multi-path and single-path routing, as shown in Figure 3.11(b).



(a) Available capacity % (residual network)



(b) Value of LB objective

Figure 3.11: (a) The gap in the residual network resources between single-path and multi-path approaches. (b) shows that the LB objective stays the same for the same set of networks used in (a).

```
N = {25}
L = {40, 50, 60, 70, 80}
Nu_of_TOPOs_Per_N_L = 1
Nu_of_TMs_Per_N_L = 100
capacity_type = {'EDGE_BETWEENNESS'}
capacity_set = {30, 35, 40}
weight_setting = {'INV_CAP'}
tm_types = {'GRAVITY'}
Network_Load = [0.5]
objectives = {'LB'}
candidate_paths = {4}
routing_strategies = {'SINGLEPATH', 'MULTIPATH'}
```

Listing 3.5: Input configuration file used to generate results in Fig. 3.11.

3.15 Summary

In this chapter, we have presented the design and implementation of a new TE simulator. The simulator differs from previous network simulators in three main aspects. First, it was specifically designed to help answer questions related to traffic engineering. Second, it provided ease-of-use and ease-of-extendability. Third, it utilizes parallel processing to achieve much faster results than the available TE simulators. We have provided two case studies to demonstrate the use of the TE simulator and to answer the questions of some TE problems we care about in this work. We have reported new results from two case studies that center on path cardinality and multi-path/single-path routing. For the first case, it turns out that the number of needed paths depends on the size of the network. For the second case, we observe that in some cases, multi-path routing is preferable to single-path routing for a network with a small number of links to utilize the network resources efficiently.

However, there is a need to incorporate other significant TE metrics, such as the stability of the TE system. This is because we may need to strike a proper balance between optimality and network stability. Moreover, we also need to include other TE systems that do not require solving an LP/MILP problem (e.g., oblivious routing, Equal-Cost-Multi-Path (ECMP) [63], Constrained-Shortest-Path-First (CSPF) [64], and so on).

4 A Novel TE system

Paths selection algorithms and rate adaptation objective functions are usually studied separately. In contrast, this work evaluates some traffic engineering (TE) systems for software defined networking obtained by combining path selection techniques with average delay and load balancing, the two most popular TE objective functions. Based on TE simulation results, the best TE system suitable for software defined networks is a system where the paths are calculated using an oblivious routing model and its adaptation rate calculated using an average delay objective function. Thus, we propose the RACKE+AD system combining path sets computed using Räcke's oblivious routing and a traffic splitting objective function using average delay. This model outperforms current state-of-the-art models, maximizes throughput, achieves better network resource utilization, and minimizes delay. The proposed system outperformed SMORE and SWAN by 4.2% and 9.6% respectively, achieving 27% better utilization and delivering 34% more traffic with 50% less latency compared with both systems on a GÉANT network.

4.1 Background

Centralized traffic engineering (TE) has gained much attention following new software defined networking (SDN) developments. Large technology companies such as Microsoft [4] and Google [2] have shifted to this technology over the last few years.

Some previous studies have deviated from the standard SDN centralization feature to improve scalability and fast adaptation to changing traffic conditions, e.g. Contra [65], HULA [66], MP-HULA [67], and DASH [68] balance load traffic entirely in the data plane to reduce controller overhead. These solutions provide scalable systems with short response time, but degrade performance, with resulting distributed solutions far from optimal [69].

Performance can also be affected by the traffic splitting objective function. Some TE

systems balance load over some paths by minimizing maximum link utilization (MLU) [3, 4]. However, minimizing MLU does balance load and enhance performance for low traffic and degrades performance significantly during peak hours since it requires additional constraints to satisfy all the demands [43]. Other TE systems use meta-heuristic [70] or heuristic [71] solutions that can provide fast routing convergence, but the solutions are sub-optimal since they may be only local optima. Prior to SDN, several studies considered different objectives [72, 73]. To our knowledge, performance impacts from these objectives and path selection strategies have not been properly considered for SDN. Any TE system has two key ingredients: which set of paths is used for forwarding traffic, and how to split traffic over these selected paths. To the best of our knowledge, no previous study has focused on boosting performance by optimizing combinations of these key ingredients, in contrast, previous work has focused on either path selection algorithms or traffic-splitting objective functions, but not both.

Many studies suggest that a set of shortest paths should be used in TE systems to achieve reliable performance [4, 13, 14]. Unfortunately, choosing shortest paths may exacerbate congestion for topologies with high link capacity heterogeneity. Oblivious routing[1] strategies offer network demand independent routing schemes, i.e., the routing scheme that is oblivious to the demands [18, 19, 20, 74]. Although oblivious routing schemes can be devised with guaranteed congestion ratio, the resulting routing scheme is static and unable to adapt to changing traffic conditions. Several studies have shown that route allocations calculated using an oblivious routing model achieve comparable quality to adaptive solutions [3, 75]. Selected paths from this approach are capacity-aware and diverse, which improves not only system performance, but also robustness.

The capacity aware concept not only applies to path selection only, but also to sending rates. For example, the Kleinrock delay objective function [76] minimizes congestion by increasing highly utilized link costs, thus, avoiding highly congested links. The widely used

[1] The terms “Oblivious routing” and “Räcke’s oblivious routing” are used interchangeably.

load balancing (LB) objective function [3, 4, 5, 47, 61] minimizes utilization (relative load) for all links, and can also be considered a capacity-aware objective function. The main goal for demand aware objectives is to mitigate proportional increases for all demands [72] by minimizing MLU. However, all source destination (SD) pair demands do not increase at the same rate, and it is not trivial to predict future demands. Thus, sending rates should not only be capacity aware, but also demand aware.

Therefore, and motivated by SMORE [3] and AD objective functions [47, 48, 61] we propose RACKE+AD, a centralized, adaptive, semi-oblivious, demand aware, near optimal TE system with static routes allocated using Räcke's oblivious routing model [19, 20, 74] and dynamic rate adaptation by approximating the average delay (AD) objective function. RACKE+AD outperformed SWAN [4] and SMORE [3] for throughput, congestion, and latency evaluated on GÉANT and ATT topologies.

4.2 Related Works

The classic approach for TE problems is to solve them as a linear program (LP) [9, 61], referred to as a multi-commodity flow problem, where the objective function usually minimizes MLU. The approximation of AD objective function is not as widely as used. However, this classical approach does not consider decoupling TE system phases because all available paths are provided as inputs. Choosing all available paths has two limitations: more paths means more decision variables in the LP, and forwarding devices, such routers and switches, have limited TCAM memory, hence fewer number paths is always preferable to keep the routing table as small as possible.

The conventional approach adjusts link weights to find a good routing scheme that can increase throughput or minimize congestion in the network [10, 11]. However, OSPF can never reach optimal because it uses the equal cost multi-path approach that splits traffic evenly among available shortest paths without rate adaptation. Furthermore, optimizing

link weights is an NP-hard problem.

Potentially centralized TE approaches recently became viable due to software-defined networking (SDN) developments, that clearly decouple the two TE phases. SWAN [4] distributes traffic over a set of k-shortest paths using an LP that reserves a small amount of scratch capacity on links to apply updates in a congestion-free manner. SOL [5] uses a greedy approach to randomly select paths with the promise that this random selection will help load balancing traffic across the network. This latter approach is somewhat similar to valiant load balancing [16] but can lead to unnecessarily long paths and consequently increased latency.

Oblivious routing [18, 19, 20] has also been proposed to find a routing scheme that performs well under all possible demands. The Räcke oblivious routing model [19] guarantees a congestion rate that is never worse than $O(\log n)$ of optimal, where n is the number of nodes in the graph. However, despite the guaranteed congestion ratio, this approach cannot outperform systems like SWAN since it considers all possible traffic demands. On the other hand, the oblivious routing approach has inspired several studies (including the current study) to investigate a careful path selection approach. SMORE [3] was inspired by Räcke's oblivious routing model to carefully select paths that increase TE system performance and robustness. Paths selected this way have low stretch, which is important to decrease latency, and are capacity aware, which is important for load balancing. The proposed approach in this work suggests that careful route selection is not sufficient performance enhancement to reach the expected maximum performance. However, a different objective function from the commonly employed LB could further enhance performance. Hence we were inspired to compare LB and AD objective function performance, and subsequently propose the RACKE+AD TE system using oblivious routing for path selection with AD to achieve better link delay and network performance.

4.3 System Model

All TE systems comprise two phases: identifying a set of paths to be used to forward traffic (path selection), and identifying splitting ratios to distribute traffic over these paths (rate adaptation). Generally, routes selected in the path selection phase are static, i.e., selected once and only recalculated when the network topology changes. Path selection is usually offline because updating end-to-end paths may take hundreds of seconds for wide area networks. In contrast, the rate adaptation phase must update path weights regularly due to frequent demand changes. However, the time required to update path weights is considerably less than the time required to update paths in the network. Among many techniques of paths selection algorithms and rate adaptation objective functions, the aim of this research is to find the best combination of these phases to enhance network performance.

Path and Rate Adaptation Properties: Intuitively, independently chosen paths may not provide better performance than dependently chosen paths. However, SMORE showed that path selection has considerable effect on performance [3]. Selected paths should be low stretch to minimize latency and naturally load balanced to provide better performance. Low stretch motivated us to compare SMORE performance and latency against k-shortest paths (KSP) approaches. SMORE is naturally load balanced since route computation in Räcke's oblivious routing model is not independent and incorporates some randomness, i.e., the obtained route set may not be the same if we were to run the model again. Thus, we expect different performance for each run. On the other hand, KSP selected paths are not capacity aware, whereas Räcke's model selected paths are capacity-aware due to the natural load balancing. Performance can be further boosted if we use the same concept for splitting traffic over the selected paths, and we expect best performance may be achieved using phases, path selection, and rate adaptation.

4.4 Simulation Setup

4.4.1 Evaluating Routing Scheme Quality

We evaluate TE systems based on congestion, throughput, and delay. Congestion reflects how a TE system utilizes network resources, and we mostly care about congestion when traffic demand exceeds link capacity. Thus, avoiding congestion can be considered as preserving as much residual capacity as possible, which is important for unexpected traffic surges that could cause bottlenecks. Congestion has negative impact on delay due to queuing. We measure path delay by summing queuing delay for each link along that path, $l/(c-l)$, where l is the absolute link load and c is the link capacity. Throughput is the proportion of total demand that is successfully delivered to the destinations.

4.4.2 Simulation Settings

Path selection algorithms. We use three approaches for path selection (i) paths selected using Räcke's oblivious routing model, (ii) paths selected using KSP algorithm, and (iii) select all available simple paths. We refer to these RACKE, KSP, and OPTIMAL, respectively.

Rate adaptation objective functions. We use two objective functions for rate adaptation: AD and LB. We refer to a routing scheme with paths selected using KSP and rate adaptation using LB objective function as KSP+LB. Similarly, models where the routing scheme selects all available paths and rate adaptation uses AD is referred to as OPTIMAL (AD), etc. The RACKE+LB routing scheme parallels that used in SMORE [3], and KSP+LB is an approximation to the SWAN scheme [4]. Table 4.1 shows the TE systems used in our experiment.

Path budget. Similar to SMORE and SWAN, and to ensure a fair comparison, we use 4 paths to evaluate any routing scheme. If the Räcke's oblivious routing model produces a

Table 4.1: Implemented TE algorithms

TE System	Description
KSP+LB	*k-Shortest* Paths (KSP) for paths, LB for weights
KSP+AD	*k-Shortest* Paths (KSP) for paths, AD for weights
RACKE+LB	Räcke's oblivious routing for paths, LB for weights
RACKE+AD	Räcke's oblivious routing for paths, AD for weights
OPTIMAL(LB)[a]	All paths, LB for weights
OPTIMAL(AD)[b]	All paths, AD for weights

[a] The best load balance is achieved with this system.
[b] The best average delay is achieved with this system.

routing scheme with SD pairs that has more than 4 paths, we use the 4 highest weight paths, similar to SMORE.

Traffic matrix generation. We use the gravity model to generate the traffic matrix (TM) [3, 18]. The gravity model approximates real-world TMs for a production network [77]. TMs are deduced based on incoming/outgoing flow for each forwarding device. Since that information is not available, we use a capacity based heuristic rather than incoming/outgoing flow information [18].

Topologies. We evaluate many TE systems for ATT and GÉANT[2] production topologies. The GÉANT network (European academic network) contains 38 nodes and 104 directed links with heterogeneous capacities. Fig. 4.1 shows the link capacity distribution for this network. Different TE systems may behave differently depending on link capacity distributions. Shortest-path TE systems may introduce a bottleneck in heterogeneous link capacities as many SD pairs compete for the same resources.

[2] dataset available at: http://www.topology-zoo.org/dataset.html

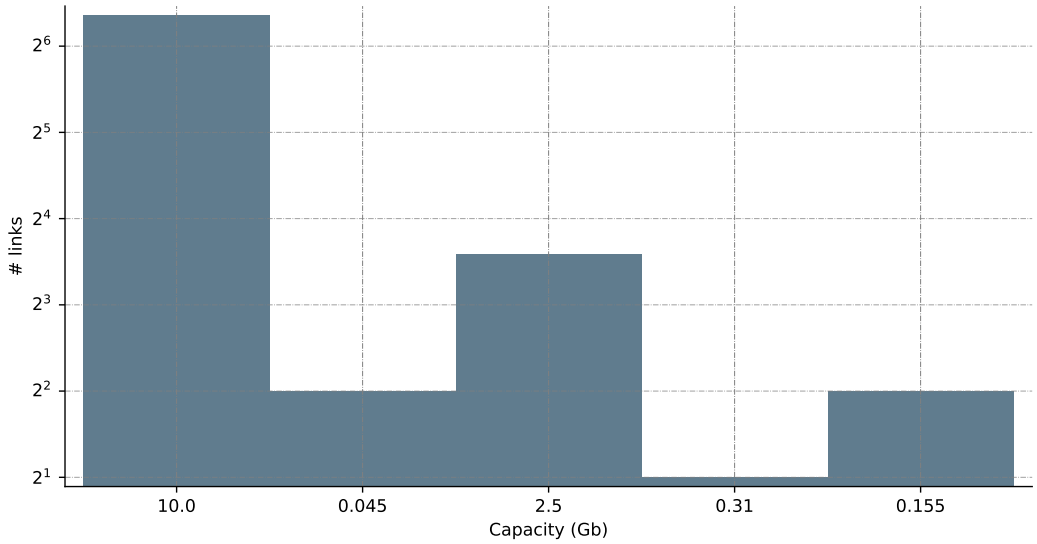


Figure 4.1: Capacity distribution for GÉANT network (log scaled).

4.5 Results

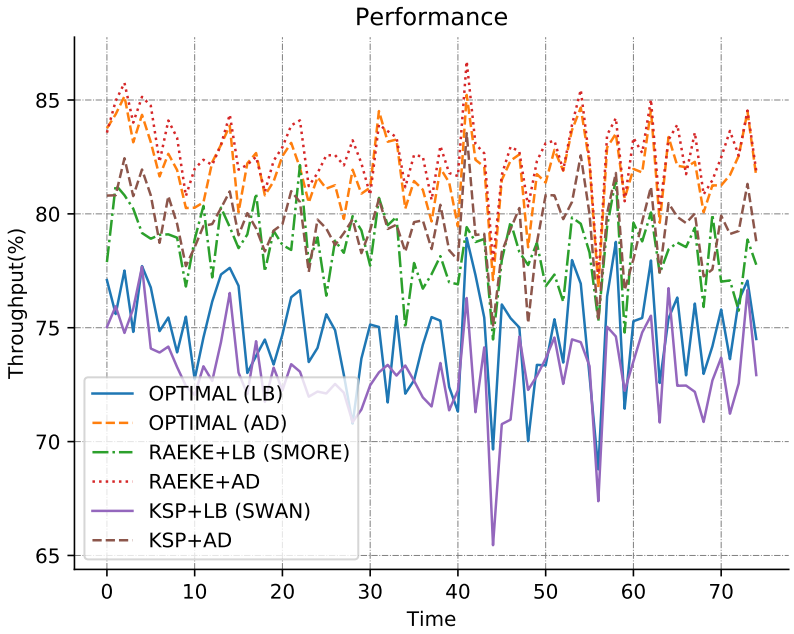
We evaluated multiple routing schemes using criteria focused on:

- how each TE system performs regarding throughput and congestion, and
- SMORE and KSP TE system impacts on latency.

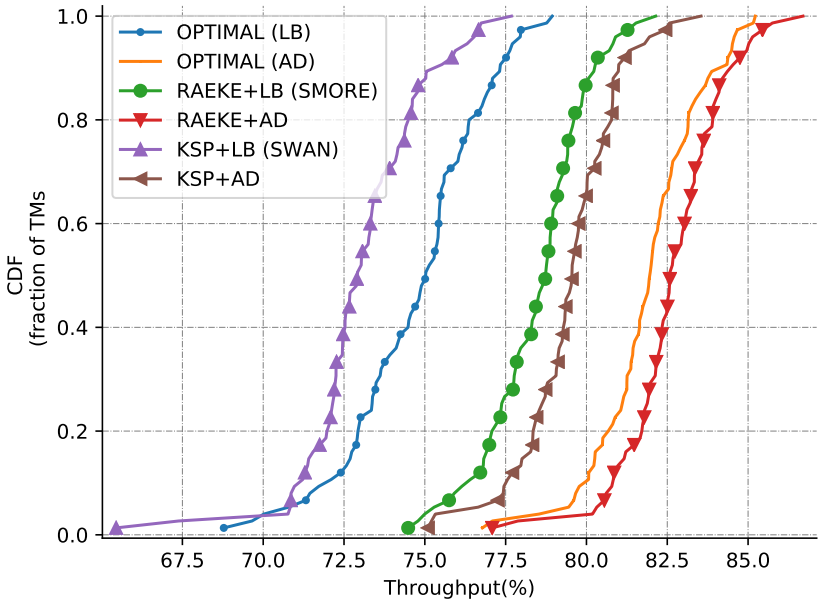
4.5.1 Throughput

Performance for many TE systems were evaluated on GÉANT and ATT networks with path budget = 4 for a fair comparison with SMORE. Figures 4.2(a) and 4.2(b) show throughput and corresponding throughput distribution for GÉANT network, respectively. Rate adaptation using AD objective function significantly increases throughput, achieving 4.2% and 9.6% improvement over SMORE and KSP+LB, respectively, which confirms path selection effectiveness using Räcke's oblivious routing algorithm.

Similar to GÉANT topology, a higher throughput was achieved for ATT topology



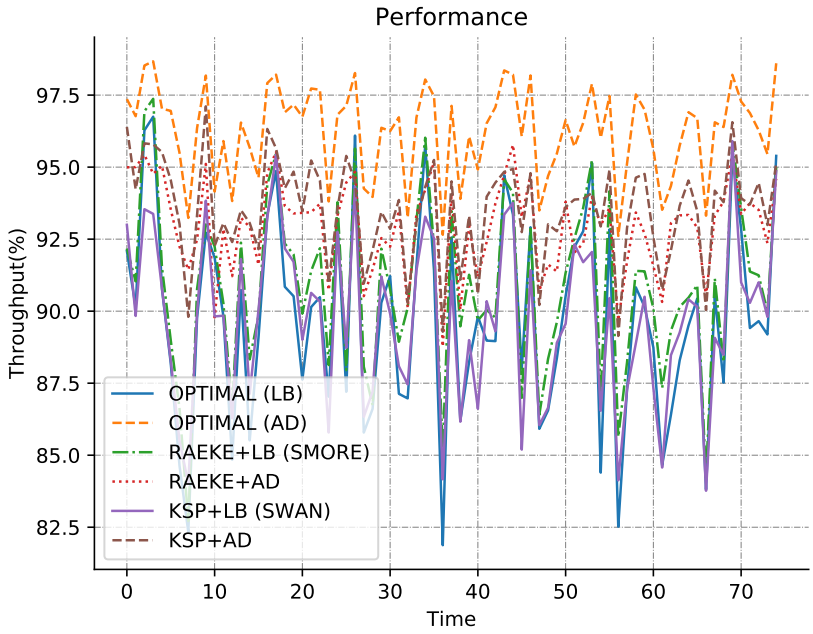
(a) Throughput



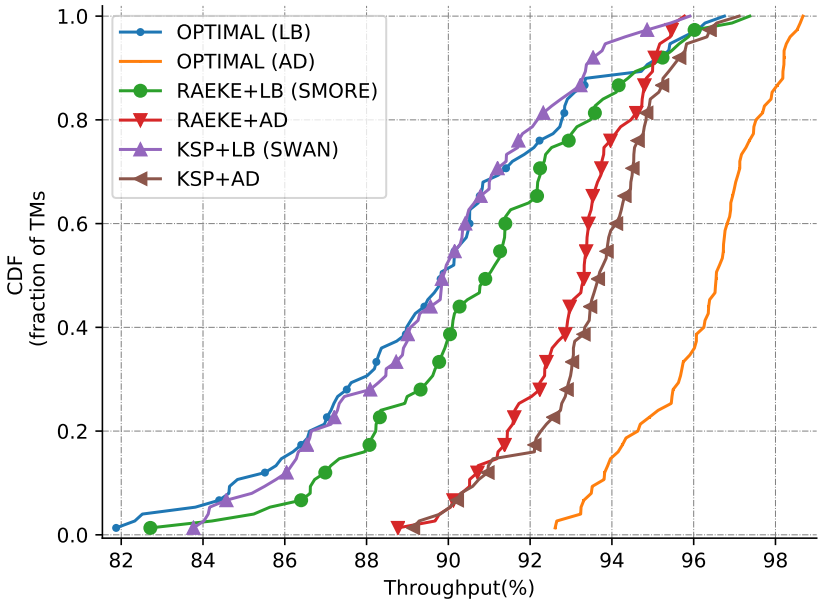
(b) Throughput distribution

Figure 4.2: Throughput on GÉANT topology

using the AD adaptation rate objective function. KSP had slightly better throughput than



(a) Throughput



(b) Throughput distribution

Figure 4.3: Throughput on ATT topology

Räcke's oblivious routing path selection algorithm (Figs. 4.3(a) and 4.3(b)).

Räcke's oblivious routing model with LB adaptation rate performed 1.14% better than

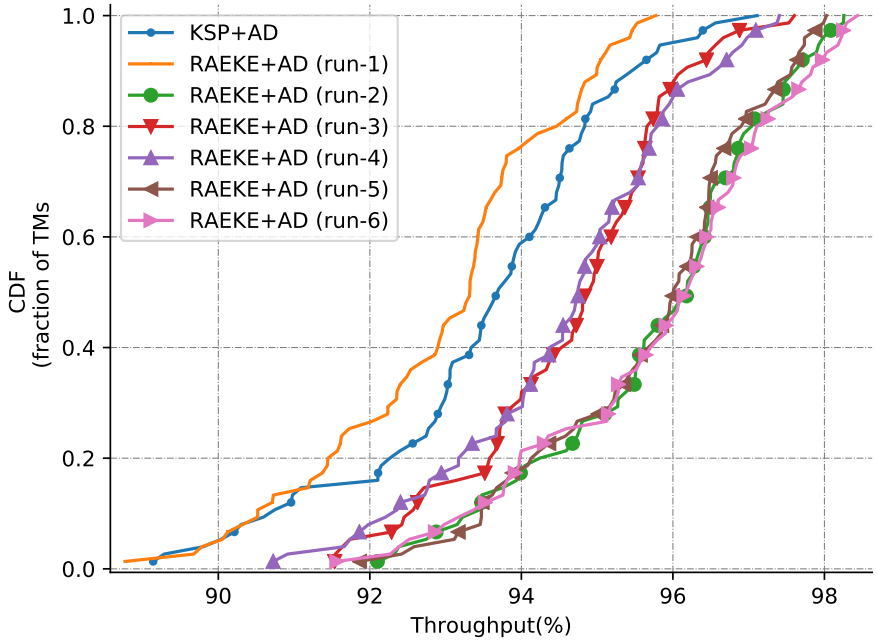


Figure 4.4: Throughput distribution for ATT topology for 6 Räcke's schemes and 1 KSP scheme.

KSP on average. This may confirm that AD favors shortest paths when all links have the same capacity. However, there is no guarantee that SMORE will always outperform (or underperform) KSP under the same conditions due to oblivious routing scheme randomness. Figure 4.4 shows throughput distributions for KSP+AD with a different Räcke's oblivious routing TE systems obtained by repeatedly calculating the oblivious routing scheme. Output from KSP+AD remained constant since KSP+AD is deterministic. Räcke's oblivious routing scheme outperformed KSP for 5 runs and underperformed for 1 run. Thus, there is a worst case scenario where KSP may perform better than SMORE. The best run had 2.29% higher throughput than KSP+AD. Therefore, a network operator may choose to run Räcke's scheme several times and choose the best outcome.

4.5.2 Congestion

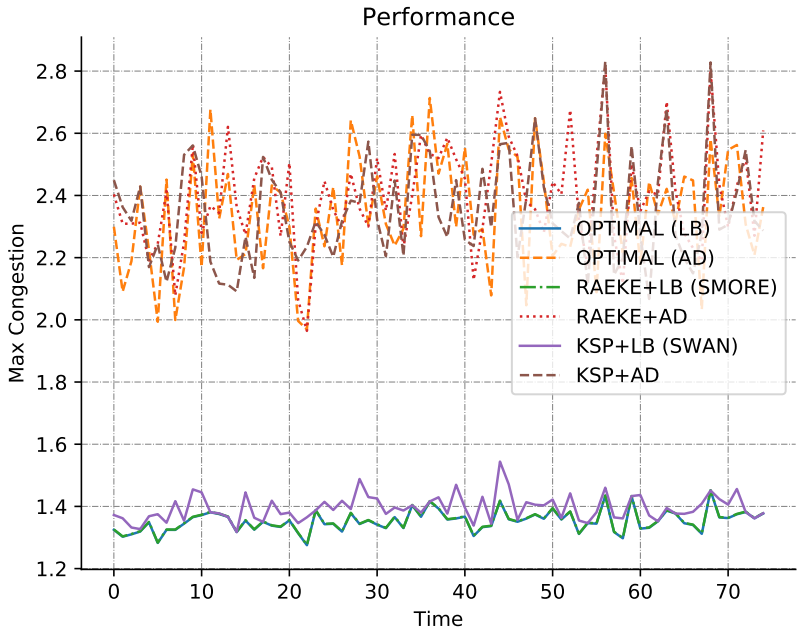
Figures 4.5(a) and 4.5(b) show network congestion for GÉANT topology using AD and LB. The AD objective function scheduled link loading differently from LB. Figure 4.5(a) shows the maximum congested link over time. All TE systems scheduled link loads that exceeded specific link capacities since we deliberately fed the system with high volume demands to investigate TE system performance well under stressed conditions. AD (Fig. 4.5(a)) seems to have higher MLU whereas Fig. 4.5(b)) shows that the AD objective utilizes link loads much better than LB. TE systems with LB caused a bottleneck for more than 40% of links whereas TE systems with AD objective caused a bottleneck for 13% of links. This low congestion ratio for AD is the main reason for the higher throughput (Fig. 4.2).

The LB objective always distributes traffic perfectly across the available routes, in the sense that all paths are used and all nodes send and receive traffic with quite similar link utilization (relative load) for all links. Thus, all links might be over-utilized under high demands when the system is not feasible. On the other hand, AD deals more with delay and throughput, but generates worse MLU than from LB. However, MLU is not a true network metric as it only considers congestion for a single link rather than the whole network. Thus, congestion distribution seems like a more reasonable metric, and we only measured MLU to make that point since it is heavily used in the literature.

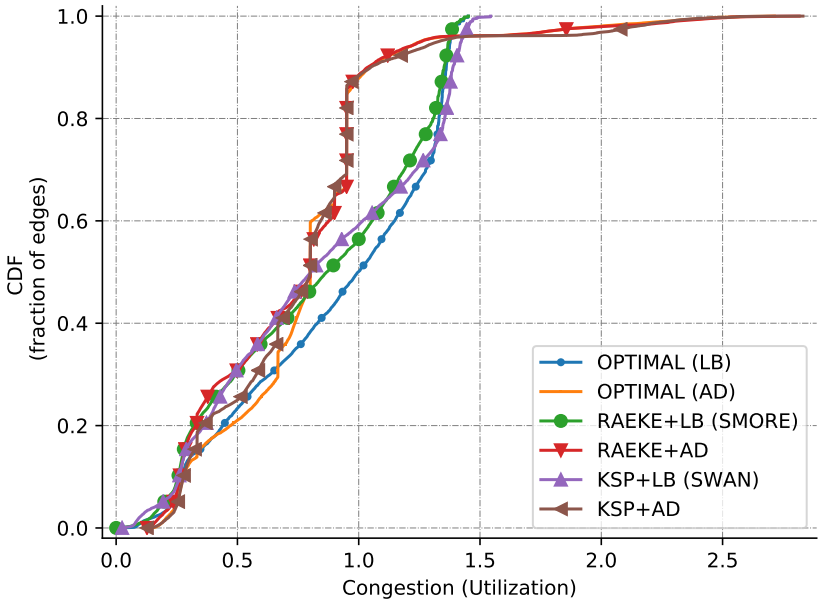
Thus, two factors contributed to better throughput and less congestion: routes selected using Räcke's oblivious routing algorithm, and using the AD objective. Similar results were obtained for ATT topology (Figs. 4.6(a) and 4.6(b)).

4.5.3 Latency

Figure 4.7 shows link delay distribution with respect to traffic delivered within that delay for GÉANT and ATT topologies. Latency for each path was computed by summing



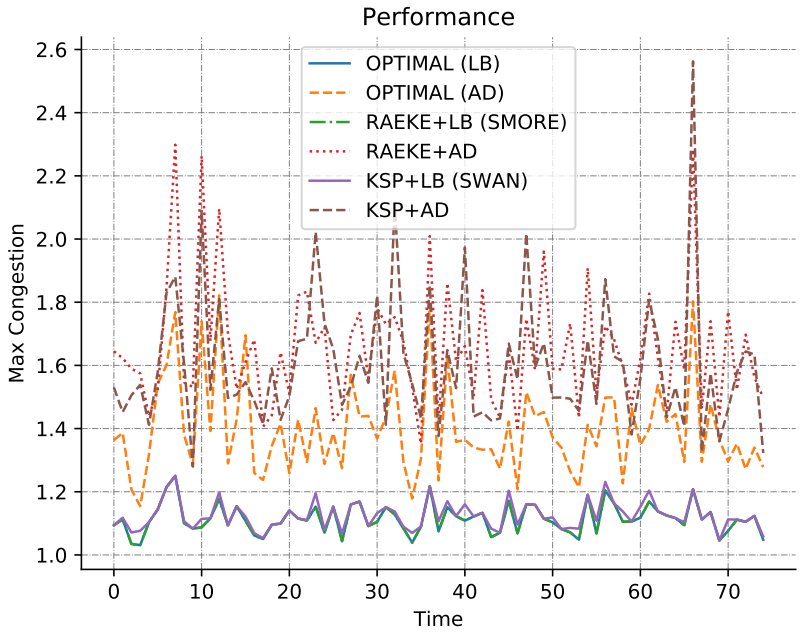
(a) Max link congestion, GÉANT topology



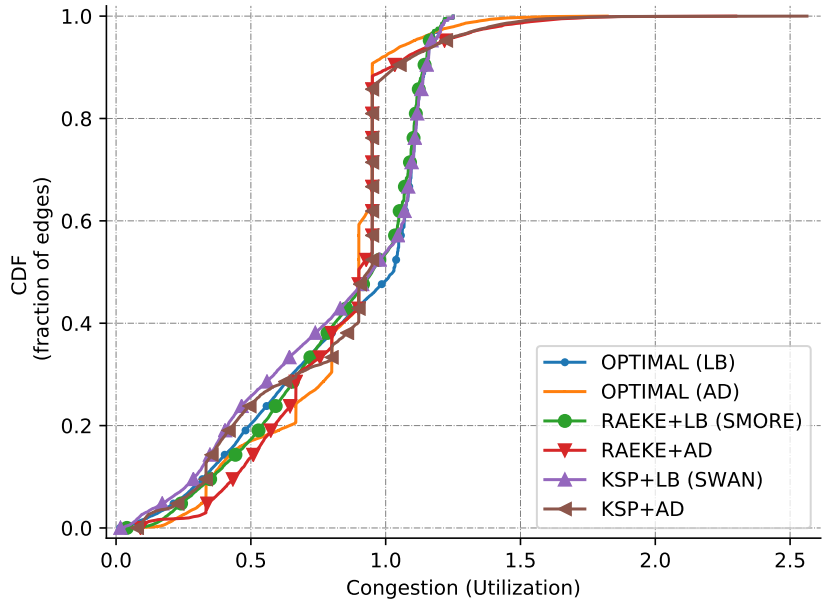
(b) CDF of link congestions, GÉANT topology

Figure 4.5: Max link congestion and links' congestion distribution on GÉANT topology

the link delays to obtain the path delay. Including AD selection outperforms LB, achieving significantly lower latency. Figure 4.7(a) shows that LB and AD TE systems different



(a) Max link congestion, ATT topology



(b) CDF of link congestions, ATT topology

Figure 4.6: Max link congestion and links' congestion distribution on ATT topology

considerably for GÉANT topology. TE systems with AD objective initially deliver approximately 34% traffic more than those with LB objective, which also has latency 50% lower

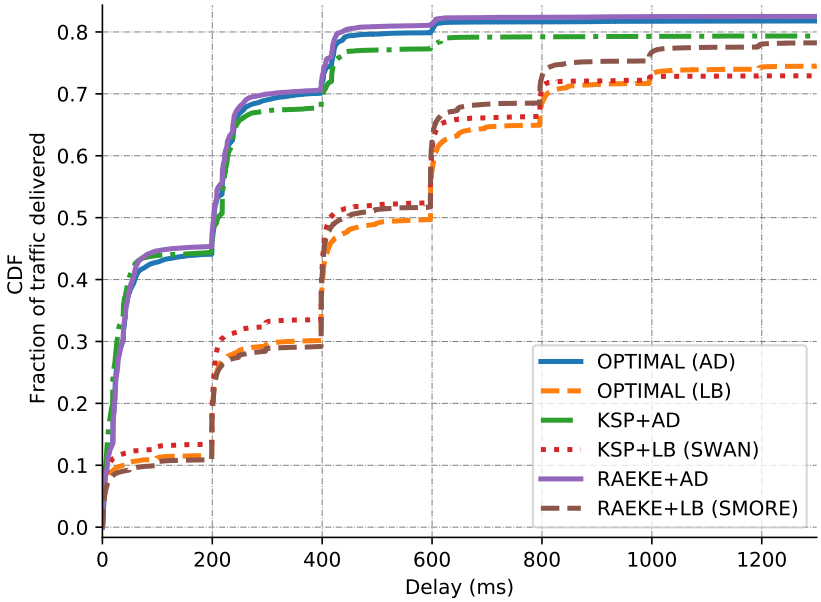
latency than TE systems with AD. RACKE+AD routing delivered slightly more traffic than OPTIMAL(AD) since OPTIMAL(AD) goal is to reduce total delay rather than throughput. Figure 4.7(b) shows that routing schemes with AD also delivered more traffic than those with LB for ATT topology. However, the gap between the two groups is somewhat smaller than for GÉANT topologies (Fig. 8(a)) because ATT network links are heterogeneous, hence smaller performance differences between individual links.

4.6 Discussion

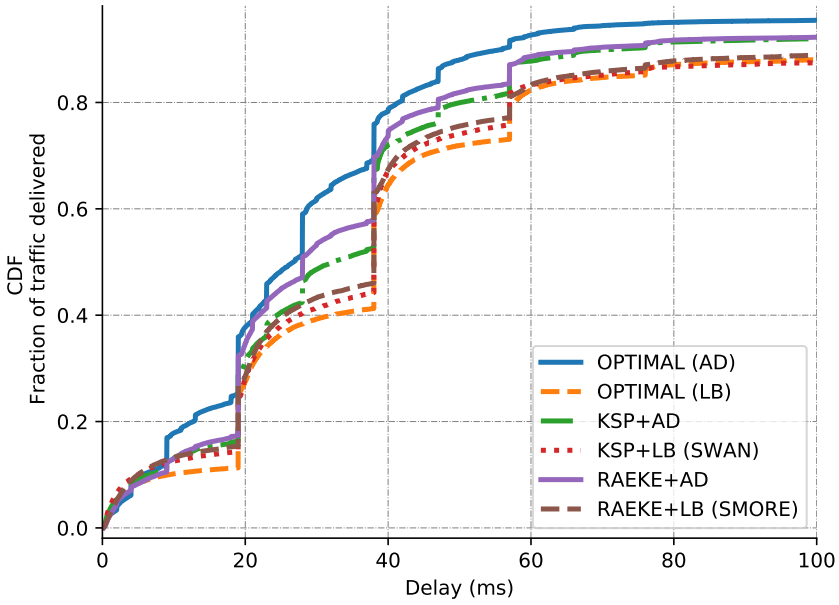
This section discusses the reason behind the high gap in performance and delay between LB and AD objective functions and one potential limitation for this work. The LB objective function tends to make the relative load the same for all links when all SD pairs are sending and receiving traffic. This can enhance performance to some extent but causes bottlenecks between some SD pairs under stressed conditions and unpredicted demands, with consequential congestion loss. On the other hand, the AD objective function increases the cost for highly utilized links to avoid utilizing them if other less heavily utilized links are available. Thus, AD is more demand aware than LB and hence offers better contribution to performance. However, solving LP for LB is much faster than for AD, particularly for larger networks due to the increased number of constraints and decision variables.

4.7 Summary

Although a few TE systems have been optimized previously using different path selection algorithms, few studies have investigates performance enhancement by testing many objective functions for splitting traffic. These phases have only been studied in isolation previously, with no prior studies testing all possible combinations to find a routing scheme



(a) GÉANT topology.



(b) ATT topology.

Figure 4.7: Latency distribution

with the best available performance.

This work proposed RACKE+AD TE system and validated its performance advan-

tages by testing many possible combinations. RACKE+AD selects routes using Räcke's oblivious routing model and the average delay objective function. Although the intuitive AD goal is to minimize network delay, it also provides surprisingly better throughput than minimizing MLU (commonly known as load balancing).

Simulations confirmed the proposed RACKE+AD system outperformed state-of-the-art routing TE systems in terms of throughput, congestion, and delay. We discussed a caveat when running Räcke's oblivious routing model, where k-shortest paths may give better performance due to randomness in oblivious routing, and also discussed the importance of excluding the maximum congestion metric when evaluating TE systems, particularly system that split traffic not based on the LB objective function.

5 Responsive Traffic Engineering System

The routing problem for traffic engineering can be solved using different techniques. For example, the problem can be formulated as a linear program (LP) or a mixed-integer linear program (MILP) that requires solving a complex optimization problem. Thus, this approach typically cannot be used for solving a large problem in real time. Alternatively, heuristic algorithms may be devised that, though fast, do not guarantee an optimal decision. This work proposes a novel design of a system that employs a deep learning model trained on optimal decisions to solve the routing problem. The model learns to adapt to traffic dynamics by updating the traffic split ratios to distribute traffic to a few paths between a source and a destination instead of frequently computing a single path for a source and destination pair. This solves the problem of network disturbance and traffic disruption. Specifically, we train two deep learning models: DNN (MLP), which is fully connected layers of neurons, and DNN (LSTM) that consists of a few layers of LSTM neural network and a dense layer. The two models are evaluated in a TE simulator. The system offers two important features of a good traffic engineering system: producing close to optimal traffic engineering results and responding to traffic dynamics in real time. We perform simulations on two topologies, the ATT North America topology, and a 4x4 grid topology. The results show that our proposed system can learn from optimal decisions to attain a responsive traffic engineering system. The paramount difference between previous work and our work is that the routing problem is usually treated as selecting paths to send packets from sources to destinations in previous work. In contrast, our work treats the problem as splitting traffic among a few paths that are preselected for each SD pair.

5.1 Background

Recently, there have been many proposals that exploit Machine Learning (ML)/Deep Learning (DL) techniques to solve the routing problem in Traffic Engineering (TE) to minimize the network congestion or delay [38, 37, 44, 29, 34, 39, 40, 78, 30, 31, 32]. Regardless of the different optimization objectives, most past research treats the TE problem as finding a dynamic single path between any source-destination (SD) pair. It has been shown that multi-path routing, i.e., traffic splitting across different paths, is more advantageous than single-path routing, as we will show in Section 5.2. A dynamic optimal single path in a production network can cause additional overhead to network elements, i.e., switching or routing devices, due to the need for installing new paths whenever there is a change in the network traffic patterns. It has been shown that, under real network settings, traffic splittings between paths are relatively fast operation compared to installing new routes in the system that may take several minutes to update on many geo-distributed networking devices [3]. Besides, installing and deleting routes may introduce network disruption and stability problems. Thus, it is more advisable to choose static multi-path routing over dynamic single-path routing.

Considerably, the most prevalent solutions to the TE problem is to model it using LP or MILP. The use of such mathematical modeling can give an optimal solution, but requires significant amount of computation, and therefore is not responsive to real time requirements [37]. As a result, researchers have designed heuristic solutions to solve for the same objective function, a task that is non-trivial in many cases and does not guarantee an optimal or near-optimal solution [70, 71].

ML/DL offers promising solutions to the TE problem. Researchers often use a Supervised Learning (SL) model that was trained on sub-optimal or local optimal data, e.g., training data obtained from standard routing protocols [38] such as the Open-Shortest Path First (OSPF) protocol. However, an ML model trained on such data cannot produce optimal or quasi-optimal routing decisions. This is the reason why researchers tend to use

Reinforcement Learning (RL). However, reinforcement learning has two major problems that have not been addressed yet, proof of optimality and speed of convergence [33].

In this work, we strive for a TE system with the following desirable characteristics:

- **Multipath TE** The TE system makes use of more than one path between each SD pair to take full advantage of network resources.
- **Optimality** The TE solution is optimal or quasi-optimal.
- **Responsiveness** The TE system must quickly adapt to changing traffic patterns while producing the optimal or quasi-optimal solutions.
- **Stability** The TE system must ensure stability by using a few precomputed paths or dynamic paths but with minimal changes to these paths in response to changing traffic patterns.

Deep Neural Networks (DNNs) can generalize from previously seen examples and process a new input instantaneously after the training process is finished. Unlike traditional Machine Learning (ML) techniques, DNN can perform feature engineering with the underlying learning system instead of selecting features manually. In addition, it is challenging to perform feature engineering for the routing problem as all the input features we use are of the same type, a quantity of demand that needs to be forwarded from a source to a destination.

To this end, we propose DNN models to learn traffic forwarding based on traffic splitting instead of learning optimal paths. We train the DNN to learn traffic routing based on traffic engineering decisions from a TE system known as RACKE+AD described in [41] that has been shown superiority over other TE systems in terms of throughput, delay, and congestion. The RACKE+AD TE system forwards traffic based on routes computed using the Räcke's oblivious routing algorithm [19, 20, 74] and split traffic across these routes

based on the average delay objective function described in Section 5.2. Routes selected using the Räcke's model mitigate congestion that occurs when the TE system always selects the shortest path without being aware of network link capacity.

The routing problem is not limited to communication networks but can arise in many types of networks, such as transportation networks [**?**], and complex networks [**?**, **?**].

The contributions of this work are as follows:

- We propose two DL models, DNN (MLP) and DNN (LSTM), for the routing problem. The models learn traffic split ratios obtained from the optimal solutions as a result of solving the routing problems using LP. Furthermore, we test the trained models in a TE simulator and report the gap in throughput between the optimal LP solution and the solution we get from the trained models.

- We compare the performance of the two proposed DL models. The result confirms that the LSTM neural network performs better than MLP.

- We show the effectiveness of multi-path routing over single-path routing and their impact on network performance.

The rest of this chapter is organized as follows. In Section 5.2, we describe the motivation behind our proposal with some early experimental results that shows the performance of different TE systems. We describe our system in section 5.3. In Section 5.4, we report and discuss our findings.

5.2 Motivation

To the best of our knowledge, most of the previously proposed ML/DL solutions for the routing problem involve a single path planning [29, 34, 37, 38, 39, 40, 44, 78] and ignore the effect of multipath routing on network performance, i.e., the lack of modeling a TE system that splits traffic across many available paths between the source and destination

nodes. In this section, we perform a series of simulations to show the benefit of splitting traffic across a few static paths between each SD pair over forwarding traffic on a single but dynamic optimal path between each SD pair.

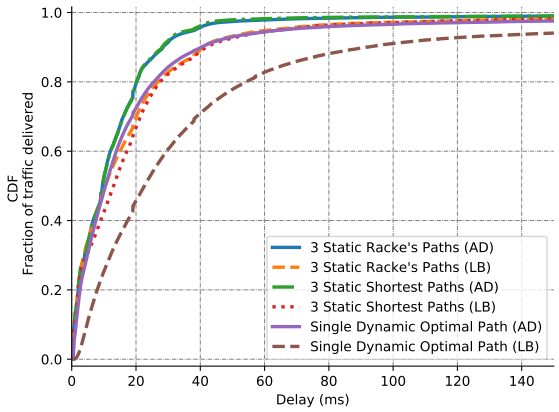
Figure 5.1: Abilene topology, regenerated from [1].

Simulation Setup. We compare six TE systems, four of which use three static paths between each SD pair. These four multi-path systems are the combinations of two path selection strategies and two traffic splitting objective functions. The two routing strategies are the k-shortest paths and paths extracted from the Racke's oblivious routing algorithm. The two traffic splitting objective functions are Load Balance (LB), also known in the literature as the minimization of the maximum link utilization (MLU), and Average Delay (AD) objective function which is a piecewise linear approximation of the delay function depicted in Fig. 3.6. The other two TE systems are the optimal systems that use a single dynamic path with the same two traffic objective functions mentioned earlier. We conduct the experiment using the Abilene topology[1] as depicted in Fig. 5.1. The traffic matrices are

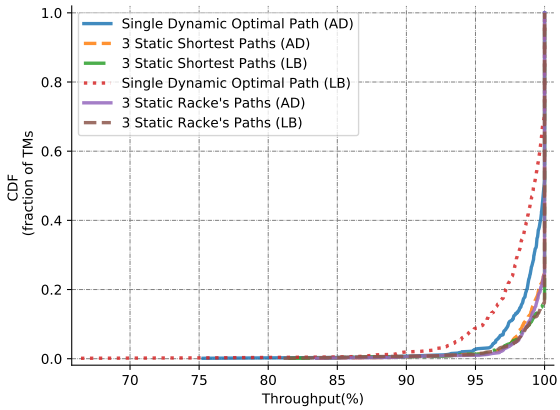
[1] dataset available at: http://www.topology-zoo.org/dataset.html

generated using the gravity model [3, 18].

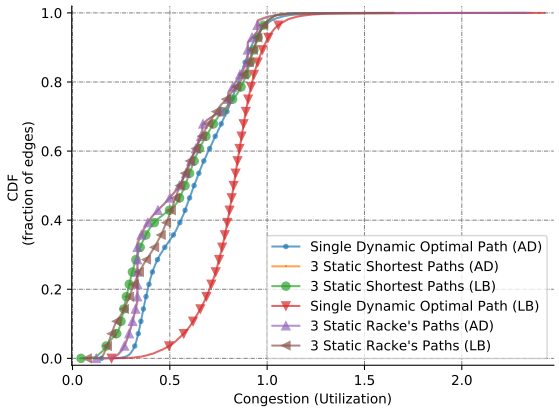
Fig. 5.2 shows the cumulative distribution function of the six TE systems for 3 performance metrics: delay, throughput, and link congestion. The results confirm that TE systems with dynamic optimal single paths do not perform better than multi-path systems. For example, in Fig 5.2(c), using 3 static paths over one single path enhanced the utilization of about 40% when the LB objective function was used. Thus, we propose a deep learning model that learns traffic splitting instead of learning the best single path to route traffic. Furthermore, we propose modeling a TE system with paths chosen based on the Racke's oblivious routing model and traffic splitting based on the AD objective function. This traffic engineering approach achieves comparable performance in delay, throughput, and link utilization with respect to the optimal traffic engineering [41]. The AD objective function has shown better performance than MLU [41] due to the degraded performance of MLU under heavy traffic loads, which may limit the total throughput of the network [43]. The aforementioned TE system is referred to in the literature as *RACKE+AD* [41].



(a) CDF of latency



(b) CDF of throughput



(c) CDF of link congestion

Figure 5.2: Three measured metrics for the Abilene topology for different routing strategy settings.

5.3 System Description

This section describes our proposed model that learns near-optimal traffic split ratios that will be used to route traffic flows for better delay, throughput, and resource utilization. Fig. 5.3 shows the three main steps of the proposed system. Please note that the first two steps, training data preparation and model training, happen offline, then the resulting model can be applied online through the third step, the running phase.

5.3.1 Training Data Preparation

Training data are generated, in an offline mode, by solving tens of thousands of problem instances in parallel using a Linear Program (LP). One LP instance takes the topology information and TM as input. It produces the corresponding routing strategy, i.e., split ratios over a fixed set of routes as output. We use the Gurobi optimization software [7] to solve these problem formulations in parallel. To output a valid routing prediction, data should not be scaled or normalized on a per column basis. Instead, the whole dataset should be normalized or scaled altogether as there are dependencies between individual columns in the dataset. Alternatively, data can be generated on a scaled topology, i.e., link capacities are scaled in the range [0, 1]. This is necessary to match the output of activation functions (for example, ReLU and Sigmoid).

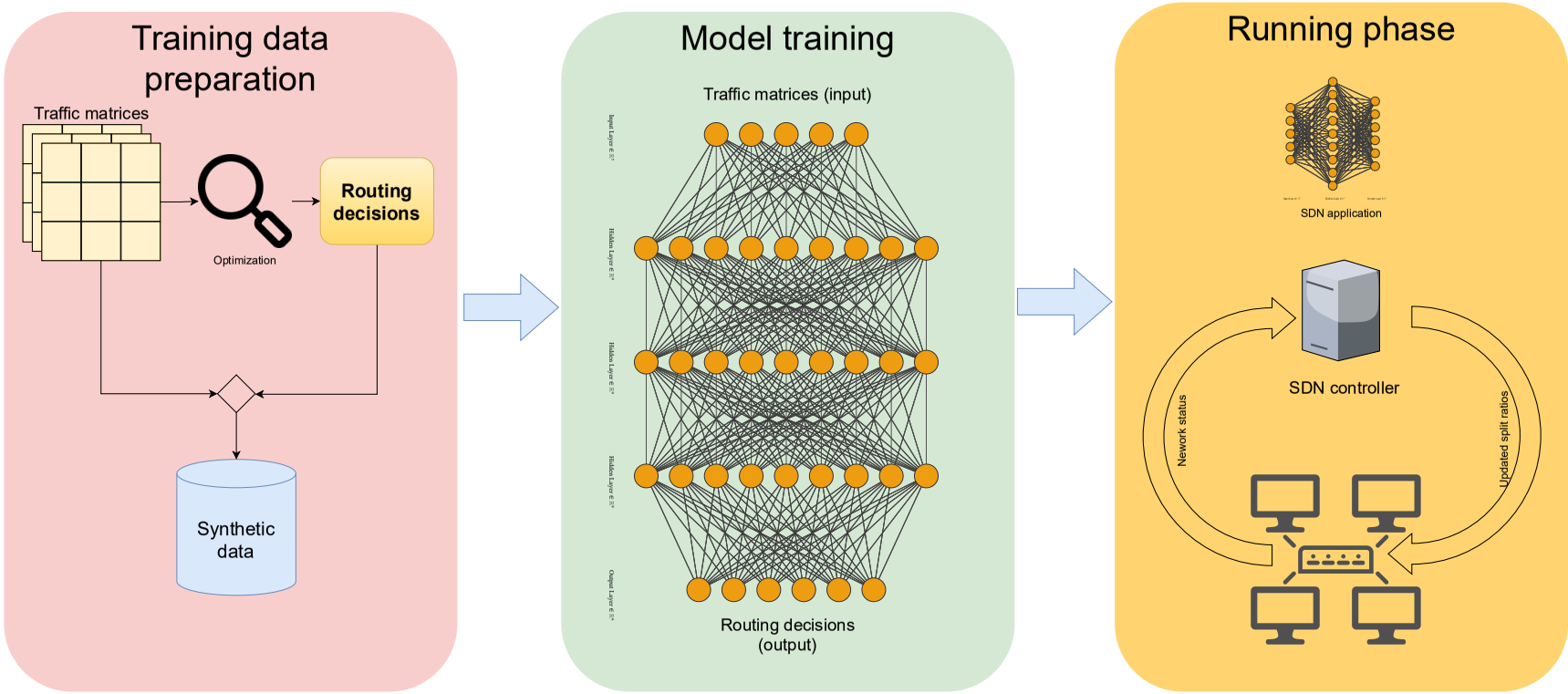


Figure 5.3: A pipeline showing the steps of the proposed traffic engineering system that leverages Deep Learning.

5.3.2 Model Training

We train two deep learning models, DNN (MLP) and DNN (LSTM), that, once trained, can find the routing decision instantaneously. They learn mapping traffic matrices to their corresponding near-optimal traffic split ratios, with routes selected statically using the Racke's oblivious routing model [20, 74]. To ensure unbiased evaluation, each traffic matrix (TM) is generated independently from a distribution of a sparsified gravity model. The split ratios are calculated based on the piecewise linear approximation (PLA) [47, 41] of the average delay objective function [79]. One of the motivations for considering demands only as input features is that they directly relate to network status. When a new flow arrives, some links will be affected. Likewise, when an existing flow terminates, some links' loads will be reduced. The traffic matrices can be easily collected due to SDN technology. The neural network models are trained using a dataset generated as described in Section 5.3.1 by solving many optimization problems using LP. LP is an optimal method to solve the routing problem but may not be efficient, especially for a large network. Thus, in this work, we try to utilize the output we get from LP to train a neural network model to do the job instead. In our case, the routing strategy is to obtain only split ratios of traffic per each SD pair over a limited number of static routes. These routes are calculated based on the Racke's oblivious routing model that has shown superiority [3, 41] over the shortest path-based techniques [4]. Similar to the first step in Section 5.3.1, this step is also done in an offline mode.

5.3.3 Running Phase

When the training is completed, the SDN controller can use the model to provide switches with the new split ratios periodically. The frequency at which the routing decision is updated is up to the network operator. The real-time traffic matrix must be collected by the SDN controller and fed to the model to decide on the new ratios. The evaluation results have shown that this model is accurate enough to achieve near-optimal performance. Using

Table 5.1: Network topologies used in the evaluation.

Topology	Nodes	Directed Links
4 x 4 grid	16	48
ATT North America	25	114

this approach, the SDN controller does not need to install new routes on switching devices; instead, the same routes are used. Traffic is adapted to changing traffic patterns by changing the split ratios only. In a production network, updating split ratios is a relatively cheap operation compared to the operation of updating routes on network elements [3].

The quantity of the split ratios might be quite large for large networks. A network of N nodes with 4 selected routes per each SD pair (assuming 4 routes per SD pair are available) will lead to a total of $N * (N - 1) * 4$ split ratios. It has been shown that using only a few routes per SD pair can give a performance that is only a few percent away from the optimal [3, 41, 5]. Because the running phase relies on the trained model, this phase is applied online. Thus, the SDN controller can query the model and get the routing configuration in real-time. This process of querying the trained model is much faster compared to solving an optimization problem.

5.4 Performance Evaluation

This section evaluates the performance of the two proposed deep learning models using two topologies, a 4 x 4 grid topology (Fig. 5.4) and ATT North America topology (Fig. 5.5). The information of these two topologies is given in Table 5.1. We trained two models, DNN (LSTM) and DNN (MLP), for each topology.

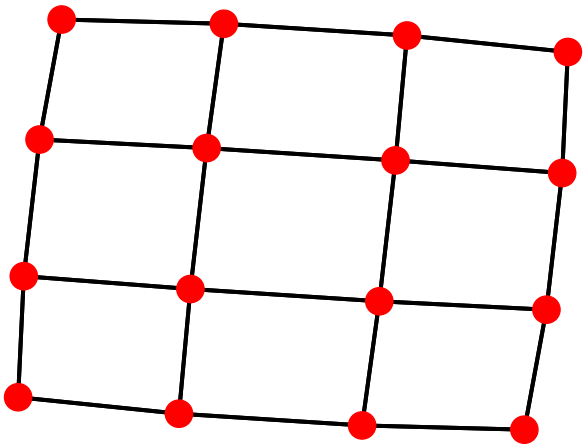


Figure 5.4: A 4 by 4 grid topology used in evaluation.

5.4.1 Evaluation Setup

We conducted model training on a remote computer with an Intel Xeon 2.20Ghz CPU, 16GB RAM, and Tesla V100-SXM2-16GB GPU. Due to the large size of the output (routing decision) that needs to be predicted by the neural network, we restricted our evaluation on two backbone topologies, a 4 x 4 grid topology and the ATT North America topology. The length of the input vector represents the number of demands between all SD pairs. Thus the input vector length is 240 and 600 for the 4 x 4 grid topology and the ATT North America topology, respectively. The length of the output vector that needs to be predicted depends on the input vector, i.e., the number of demands and the number of paths used between each SD pair. Thus, with only 3 paths used between each SD pair, the length of the output vector is 720 and 1800 for the two topologies, respectively. However, although 3 paths are allocated for each SD pair, some paths are not used at all as this depends on the objective function being used. We use grid search to decide on the number of neurons and number of layers in the DNN and other hyperparameters. We tried different configurations regarding the learning rate, drop rate, and the optimization algorithm. The rest of the used hyperparameters are depicted in Table 5.2. We checked the performance of the model for each configuration by testing it on a validation set, and selected the model with the least Mean Absolute Error (MAE). The MAE was also used to compare the performance of the

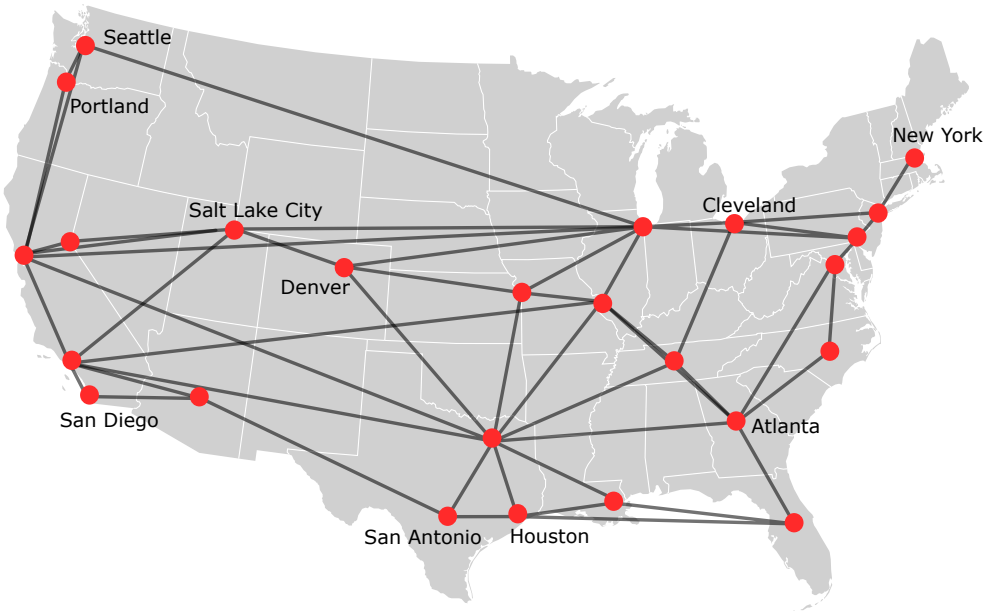


Figure 5.5: The ATT North America topology, regenerated from [1].

two mentioned DNN architectures. For DNN (MLP), the best performance is achieved with two layers. The first layer has a number of neurons 25% larger than the length of the input vector, and the second layer has a number of neurons equal to the length of the output vector. For the DNN (LSTM), 2 layers of LSTM cells with one last dense layer were used.

5.4.2 Evaluation

In this section, the performance of our two proposed deep learning models is evaluated. We evaluated the performance in two stages. First, we show how the models are learning traffic splitting with decreasing model training error over time (Fig. 5.6(a) and 5.7(a)). Second, we perform model inference (Fig. 5.6(b) and 5.7(b)) by integrating the best model obtained in stage 1 into a TE simulator from [41]. The response time for each approach is reported in Table 5.3. Using the TE simulator, we show how close the throughput from the DL model to the throughput obtained from solving the LP in two cases, RACKE+AD and optimal.

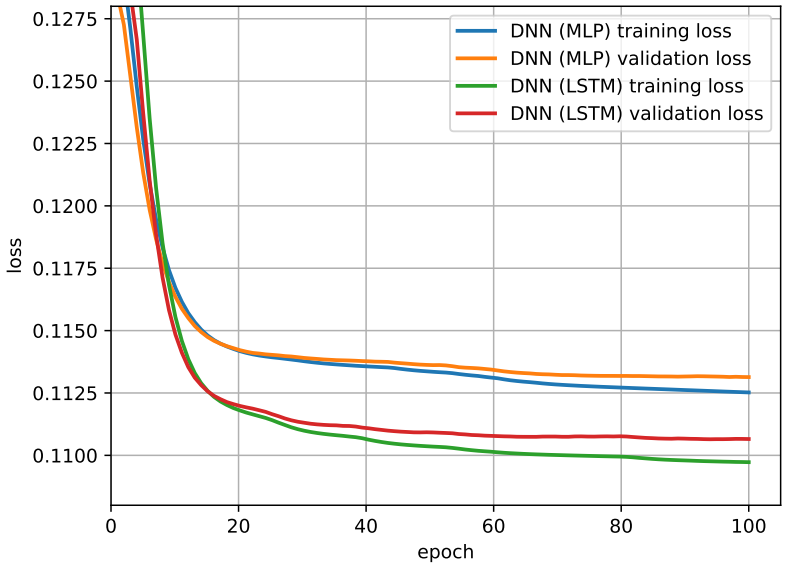
Table 5.2: Hyperparameters used for training DNN (LSTM) and DNN (MLP) for two topologies.

	4 x 4 grid		ATT North America	
	DNN (MLP)	DNN (LSTM)	DNN (MLP)	DNN (LSTM)
Learning rate α	0.01, **0.001**	0.01, **0.001**	0.01, **0.001**	0.01, **0.001**
Dropout rate	0, **0.1**, 0.2	0, 0.1, **0.2**	0, **0.1**, 0.2	0, 0.1, **0.2**
Optimizer	**Adam**, RMSProp	**Adam**, RMSProp	**Adam**, RMSProp	**Adam**, RMSProp
Layers information:				
1 hidden layer	[255]	[65]	[450]	[80]
	[240]	[85]	[600]	[100]
	[300]	[95]	[750]	[120]
	[360]	[115]	[900]	[140]
2 hidden layers	[240, 720]	[65, 95]	[600, 1800]	[80, 120]
	[300, 720]	**[85, 85]**	**[750, 1800]**	[90, 110]
	[360, 720]	[95, 65]	[900, 1800]	**[100, 100]**
				[110, 90]

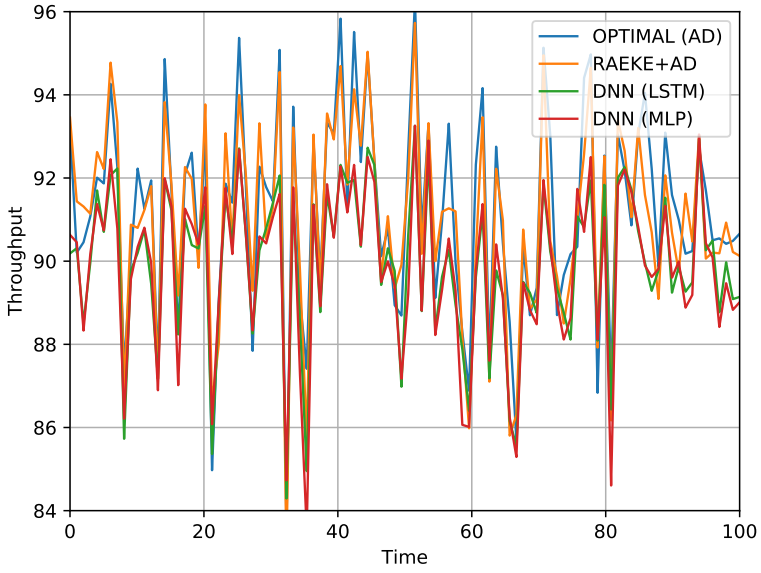
We define the optimal solution as using all the paths between all the source-destination pairs in the network with the AD objective function described in Section 5.2. We refer to this optimal solution as OPTIMAL (AD).

Model training

As can be seen in Fig. 5.6(a) and 5.7(a), both models are able to learn the splitting ratios for both topologies. The model that uses LSTM results in a slightly better prediction performance. This could be attributed to the fact that LSTM has memory and a much more complex architecture than MLP. However, the results show slight overfitting to the training data, which can be mitigated using the deep learning regularization techniques. Both models took less than one hour to finish training, making them feasible to retrain in case of topology change.

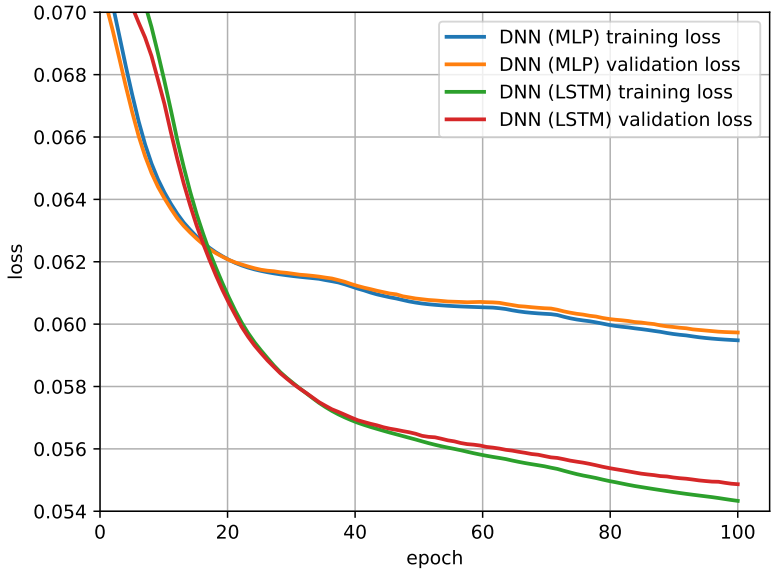


(a) The mean absolute error in DNN (MLP) model and DNN (LSTM) model for 4 x 4 grid topology.

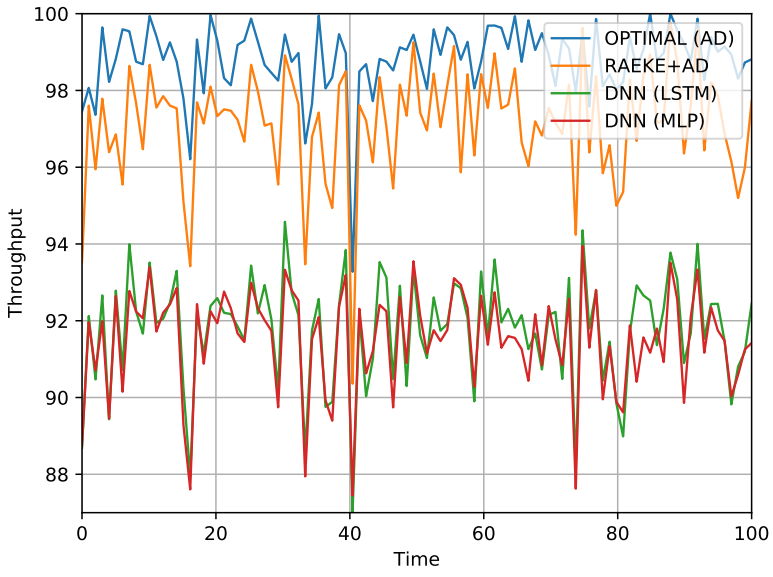


(b) Throughput for 4 x 4 grid topology.

Figure 5.6: (a) Training errors for models; (b) network throughput achieved using the two models for 4x4 grid topology.



(a) The mean absolute error in DNN (MLP) model and DNN (LSTM) model for ATT North America topology.



(b) Throughput for ATT North America topology.

Figure 5.7: (a) Training errors for models; (b) network throughput achieved using the two models for ATT North America topology.

Table 5.3: Total response time for 100 traffic matrices (in seconds). $\Delta 1$ is the percentage difference between the current model and RACKE+AD. $\Delta 2$ is the percentage difference between the current model and the optimal solution.

Algorithm	4x4 grid	($\Delta 1$)	($\Delta 2$)	ATT N. A.	($\Delta 1$)	($\Delta 2$)
DNN (MLP)	**20.85**	(-83.48%)	(-94.27%)	**25.48**	(-81.39%)	(-99.07%)
DNN (LSTM)	**24.09**	(-80.92%)	(-93.38%)	**30.134**	(-77.99%)	(-98.91%)
RACKE+AD	**126.26**	(0.0%)	(-65.32%)	**136.94**	(0.0%)	(-95.04%)
OPTIMAL (AD)	**364.10**	(+188.37%)	(0.0%)	**2765.52**	(+1919.51%)	(0.0%)

Model inference

We would like to see how the learned models from the first stage perform in the simulator. We integrated the two proposed DL models into the TE simulator to evaluate how the prediction performs against the original near-optimal solution obtained by the RACKE+AD TE system and the optimal solution. As shown in Fig. 5.6(b) and Fig. 5.7(b), network performance is evaluated for the two DL models, RACKE+AD TE, and the optimal solution, using the throughput metric. Because of the consequent overlapping between lines in Fig. 5.6(b), the results in Fig. 5.6(b) and 5.7(b) are aggregated in Table 5.4. The throughput on average, as can be seen in Table 5.4, is about 1% and 6% away from the RACKE+AD throughput for the 4 x 4 grid and the ATT North America topologies, respectively. The gap in the throughput between the learned models and the RACKE+AD system on the 4 x 4 grid topology is much smaller than the gap on the ATT North America topology. This is due to the much smaller size of the predicted splitting ratios vector of the 4 x 4 grid compared to the output size in the ATT North America topology which is 720 to 1800. In the simulator, although it is not the case in practice, we assumed that the RACKE+AD TE system could find the optimal solution instantaneously without accounting for the time required to find the solution each time the TM is updated. Thus, we expect the learned DL models to perform much better in a real production network than the solution obtained by solving LP, RACKE+AD and OPTIMAL (AD). We compare our results mainly with

Table 5.4: Overall average of network throughput.

Topology	Algorithm	Mean
4 x 4 grid	DNN (MLP)	89.836
	DNN (LSTM)	89.992
	RACKE+AD	90.990
	OPTIMAL (AD)	91.177
ATT North America	DNN (MLP)	91.506
	DNN (LSTM)	91.771
	RACKE+AD	97.136
	OPTIMAL (AD)	98.807

RACKE+AD because our two DL models are trained on data obtained from RACKE+AD. However, the result for the optimal model is also provided as a reference. RACKE+AD was 0.187% and 1.671% away from the optimal solution in 4 x 4 grid and ATT North America topologies, respectively. It should be noted that although we provided the optimal solution in terms of performance (Figs. 5.6(b) and 5.7(b)) and time taken to find the solution (Table 5.3), in practice, it is difficult to apply the optimal solution due to three reasons. First, the optimal solution uses all the available routes in the system which introduces routes oscillation problem [3, 41]. Second, switching devices typically have limited TCAM memory and it is difficult to store all available paths in that limited memory. Third, due to the large number of variables and constraints in the LP of the optimal solution, the optimal solution is difficult to be realized in real-time as can be seen in Table 5.3.

Responsiveness

Table 5.3 indicates that the two DL models require a remarkably less time in processing the input than the LP solutions, RACKE+AD and OPTIMAL (AD). The numbers in Table 5.3 (highlighted in a bold font) represent the total time in seconds an approach took to process the inputs of 100 traffic matrices.

For 4 x 4 the grid topology, the two DL models process the input 80.92% – 83.48% faster than RACKE+AD while it is 93.38% – 94.27% faster than OPTIMAL (AD). On the other hand, for the ATT North America topology, the two DL models process the input -77.99% – 81.39% faster than RACKE+AD and 98.91% – 99.07% faster than OPTIMAL (AD). As there is a trade-off between performance and responsiveness, we expect our learned approach to give better than or comparable performance of that of LP as network demands are dynamic in nature and frequent update is required.

5.5 Summary

In this chapter, we have shown two main contributions. First, using the TE simulator presented in Chapter 3, we showed the prominent effect of multi-path systems on performance over single path systems. We also showed the effect of the average delay objective function. Second, due to the high complexity of the average delay objective function, we proposed two deep learning models that learn traffic splitting based on that function. The two models are multi-layers feed-forward artificial neural networks and two layers of LSTM with one dense layer that serves as the output. The results have shown that these aforementioned models can learn traffic splitting from the dataset generated from the TE simulator. The benefit of training such deep learning models is to get a responsive traffic engineering system due to the instantaneous model inference a neural network can achieve. However, as more nodes are added to the network, the output vector can be large, and as a result, this affects the accuracy of the prediction; in a small network such as the 4 x 4 grid network, the prediction of the DL models matched with the output of the LP with negligible error. For the larger network, the ATT North America topology, the result has shown a gap in the prediction between the LP and the DL approaches that resulted in roughly 5% degraded performance in terms of throughput. In the real-world scenario, however, we expect the proposed DL models to outperform the LP due to the high complexity the average delay

function has that makes the system unresponsive.

6 Conclusions and Future Directions

In this dissertation, we presented a new, responsive, near-optimal traffic engineering system. The new TE system has many enablers that we discuss here briefly.

6.1 A New TE Simulator

In Chapter 3, we have presented the design and implementation of a new TE simulator. The simulator provides ease-of-use and ease-of-extendability. It mainly differs from the previous TE simulator in its efficiency realized by the utilization of parallel processing. We also showed how to use this simulator by changing the input parameters. Our main use of the simulator was to show how a solution based on a specific configuration deviates from the optimal solution. Specifically, with the aid of the simulator, we were able to realize the following:

- Using only a few paths per SD pair is enough to achieve near-optimal performance.
- A new TE system, RACKE+AD, that, for each SD pair, selects static paths that guarantee a congestion level no worse than n log(n), where n is the number of nodes in the network, and, for each flow between SD pair, splits traffic using the AD objective function.
- The AD function has much better performance than the LB function in three important metrics: throughput, delay, and lesser utilization of network resources.
- Multipath routing is superior over single path routing and can also significantly help achieve better performance in terms of the same three metrics mentioned above.

6.2 A New TE System

In Chapter 4, we presented a new TE system, RACKE+AD. The new system realized by constructing new TE techniques resulted from combining different path selection algorithms, specifically, oblivious paths, shortest paths, and simply selecting all paths, with two objective functions, specifically, AD and LB. Combining an item from the first set of different path selection algorithms and an item from the set of rate adaptation functions resulted in ten different TE algorithms. All the ten algorithms have been compared, and the result has shown that RACKE+AD has superiority over all the rest nine TE algorithms.

6.3 Responsive TE System

In Chapter 5, we have presented the design and evaluation of a new DL-based traffic engineering system for a software-defined network. The new system achieves two important characteristics of a good TE system, closeness to optimality and fast responsiveness. The fast responsiveness is naturally achieved due to the instantaneous inference of ML models. The closeness to optimality is achieved by training DL models on optimal solutions calculated beforehand using linear programming mathematical modeling. The proposed system also achieves an important feature which is the stability of network operation. The stability is achieved by changing traffic split ratios as needed rather than installing/deleting new/old routes in the network in response to the change in traffic patterns. The main difference between our work and previous work is how we characterize the input and the output of the deep learning model. We have shown that relying on traffic demand patterns only, as features, can give a good performance due to the direct relationship between demands and routing of demands over multiple paths. The output of the DL models is characterized by the traffic split ratios, as opposed to paths between SD pairs in previous work. The proposed system has shown an optimality gap with the ATT North America topology larger

than the optimality gap with the 4 x 4 grid topology due to the much larger output for the ATT North American topology. The optimality gap is relatively high only for large networks due to the increased size in the output vector that needs to be predicted. However, during the simulations, we assumed that the linear program could make an instantaneous decision. While it actually takes several seconds to find a solution just for a single TM in a production network. On the other hand, the AD objective function comes with higher complexity than the LB objective function.

As a future directions, we aim at reducing the limitation of this work in two ways (i) reformulating a new LP to reduce the complexity (ii) building many machine learning models to learn routing strategy to mitigate the gap in optimality.

6.4 Future Directions

6.4.1 Formulating New LP

As can be seen by now, the AD objective function is a good choice when designing a TE system. It performs much better than the LB objective function in terms of throughput, congestion, and delay. It is a good fit to manage network resources efficiently while providing a high quality of service to the end-user. Thus, we try to find another LP formulation with fewer decision variables and fewer constraints. The work in [80] does not apply directly to our case because it considers the unsplittable flow (single-path case), but it is a good place to start. For example, any link being traversed by given traffic that belongs to a specific SD pair cannot be on a line segment with an upper breakpoint smaller than the demand flow. This can be used to reduce the number of constraints used in the LP.

Another idea is to approximate the function with a fewer number of line segments. The approximation we used [8, 47] approximating the nonlinear function with 6 line segments. We would also try approximating with fewer segments. However, this may not give the

same performance that we expect.

6.4.2 Building Many Machine Learning Models

To reduce the optimality gap and enhance the scalability of the proposed TE system, we would like to train many DL models where each model is in charge of predicting a small part of the large output. We expect better results if we break the output into k parts and specify one ML model for each part. For example, if we partition the destinations into five groups, five ML models are trained, where each model learns the routing strategy for one group. All the models in the network use the same input traffic pattern but produce different outputs, where each output is only a subset of the main output. We expect this to work due to the ability of deep neural networks to find the proper set of features that are directly affecting the output; each model should extract different features than other models and based on the subset of the main output sequence.

Bibliography

[1] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *Selected Areas in Communications, IEEE Journal on*, 29(9):1765 – 1775, october 2011.

[2] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, August 2013.

[3] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, NSDI'18, page 157–170, USA, 2018. USENIX Association.

[4] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 15–26, New York, NY, USA, 2013. Association for Computing Machinery.

[5] Victor Heorhiadi, Michael K. Reiter, and Vyas Sekar. Simplifying software-defined network optimization using sol. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, page 223–237, USA, 2016. USENIX Association.

[6] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. In *Proceedings of the*

ACM SIGCOMM 2013 Conference on SIGCOMM, SIGCOMM '13, page 27–38, New York, NY, USA, 2013. Association for Computing Machinery.

[7] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020.

[8] Deepankar Medhi and Karthikeyan Ramasamy. 4 - network flow modeling. In Deepankar Medhi and Karthikeyan Ramasamy, editors, *Network Routing*, The Morgan Kaufmann Series in Networking, pages 108 – 140. Morgan Kaufmann, San Francisco, 2007.

[9] R. G. Gallager, A. Segall, and J. M. Wozencraft. Data network reliability. Massachusetts Inst. of Tech. Report, July 1976.

[10] Srikanth Sundaresan, Cristian Lumezanu, Nick Feamster, and Pierre Francois. Autonomous traffic engineering with self-configuring topologies. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 417–418, New York, NY, USA, 2010. Association for Computing Machinery.

[11] B. Fortz and M. Thorup. Internet traffic engineering by optimizing ospf weights. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 2, pages 519–528 vol.2, March 2000.

[12] Bernard Fortz and Mikkel Thorup. Increasing internet capacity using local search. *Computational Optimization and Applications*, 29:13–48, 2004.

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

[14] Zhanyou Ye, Shi Hong Marcus Wu, and Themistoklis Prodromakis. *Computing Shortest Paths in 2D and 3D Memristive Networks*, pages 537–552. Springer International Publishing, Cham, 2014.

[15] K. Xu, M. Shen, H. Liu, J. Liu, F. Li, and T. Li. Achieving optimal traffic engineering using a generalized routing framework. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):51–65, 2016.

[16] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, 1982.

[17] R. Zhang-Shen and N. McKeown. Designing a fault-tolerant network using valiant load-balancing. In *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, pages 2360–2368, 2008.

[18] D. Applegate and E. Cohen. Making routing robust to changing traffic demands: Algorithms and evaluation. *IEEE/ACM Transactions on Networking*, 14(6):1193–1206, Dec 2006.

[19] H. Racke. Minimizing congestion in general networks. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 43–52, 2002.

[20] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, page 255–264, New York, NY, USA, 2008. Association for Computing Machinery.

[21] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, page 448–455, New York, NY, USA, 2003. Association for Computing Machinery.

[22] Dahai Xu, Mung Chiang, and Jennifer Rexford. Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering. *IEEE/ACM Trans. Netw.*, 19(6):1717–1730, December 2011.

[23] D. Xu, M. Chiang, and J. Rexford. Corrections to "link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering" [dec 11 1717-1730]. *IEEE/ACM Transactions on Networking*, 23(5):1702–1703, 2015.

[24] N. Michael and A. Tang. Halo: Hop-by-hop adaptive link-state optimal routing. *IEEE/ACM Transactions on Networking*, 23(6):1862–1875, Dec 2015.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.

[26] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[27] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. Learning to route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, page 185–191, New York, NY, USA, 2017. Association for Computing Machinery.

[28] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.

[29] Junjie Zhang, Minghao Ye, Zehua Guo, Chen-Yu Yen, and H. Jonathan Chao. Cfr-rl: Traffic engineering with reinforcement learning in sdn. *IEEE Journal on Selected Areas in Communications*, 38(10):2249–2259, 2020.

[30] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai, Mike J. Hibbett, Giovani Estrada, Khaldun Ma'ruf, Florin Coras, Vina Ermagan, Hugo Latapie, Chris Cassar, John Evans, Fabio Maino, Jean Walrand, and Albert Cabellos. Knowledge-defined networking. *SIGCOMM Comput. Commun. Rev.*, 47(3):2–10, September 2017.

[31] Jose Suarez-Varela, Albert Mestres, Junlin Yu, Li Kuang, Haoyu Feng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Feature engineering for deep reinforcement learning based routing. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6, 2019.

[32] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. Experience-driven networking: A deep reinforcement learning based approach. *IEEE INFOCOM'2018*, 2018.

[33] Zoubir Mammeri. Reinforcement learning based routing in networks: Review and classification of approaches. *IEEE Access*, 7:55916–55950, 2019.

[34] Yuan Zuo, Yulei Wu, Geyong Min, and Laizhong Cui. Learning-based network path planning for traffic engineering. *Future Generation Computer Systems*, 92:59–67, 2019.

[35] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[36] Graham Neubig. Neural machine translation and sequence-to-sequence models: A tutorial. *CoRR*, abs/1703.01619, 2017.

[37] Joao Reis, Miguel Rocha, Truong Khoa Phan, David Griffin, Franck Le, and Miguel Rio. Deep neural networks for network routing. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[38] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani. State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. *IEEE Communications Surveys Tutorials*, 19(4):2432–2455, 2017.

[39] N. Kato, Z. M. Fadlullah, B. Mao, F. Tang, O. Akashi, T. Inoue, and K. Mizutani. The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective. *IEEE Wireless Communications*, 24(3):146–153, 2017.

[40] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani. Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning. *IEEE Transactions on Computers*, 66(11):1946–1960, 2017.

[41] Mohammed I. Salman and Bin Wang. Boosting performance for software defined networks from traffic engineering perspective. *Computer Communications*, 167:55 – 62, 2021.

[42] Shihan Xiao, Haiyan Mao, Bo Wu, Wenjie Liu, and Fenglin Li. Neural packet routing. In *Proceedings of the Workshop on Network Meets AI & ML*, NetAI '20, page 28–34, New York, NY, USA, 2020. Association for Computing Machinery.

[43] C. Zhang, S. Zhang, Y. Wang, W. Li, B. Jin, R. K. P. Mok, Q. Li, and H. Xu. Scalable traffic engineering for higher throughput in heavily-loaded software defined networks.

In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7, 2020.

[44] Li Yanjun, Li Xiaobo, and Yoshie Osamu. Traffic engineering framework with machine learning based meta-layer in software-defined networks. In *2014 4th IEEE International Conference on Network Infrastructure and Digital Content*, pages 121–125, 2014.

[45] S. K. Singh, T. Das, and A. Jukan. A survey on internet multipath routing and provisioning. *IEEE Communications Surveys Tutorials*, 17(4):2157–2175, Fourthquarter 2015.

[46] M. Li, A. Lukyanenko, Z. Ou, A. Ylä-Jääski, S. Tarkoma, M. Coudron, and S. Secci. Multipath transmission for the internet: A survey. *IEEE Communications Surveys Tutorials*, 18(4):2887–2925, Fourthquarter 2016.

[47] X. Liu, S. Mohanraj, M. Pióro, and D. Medhi. Multipath routing from a traffic engineering perspective: How beneficial is it? In *2014 IEEE 22nd International Conference on Network Protocols*, pages 143–154, Oct 2014.

[48] B. Fortz, L. Gouveia, and M. Joyce-Moniz. On the convex piecewise linear unsplittable multicommodity flow problem. In *2016 12th International Conference on the Design of Reliable Communication Networks (DRCN)*, pages 9–13, March 2016.

[49] Thomas R. Henderson, Mathieu Lacage, and George F. Riley. Network simulations with the ns-3 simulator. In *In Sigcomm (Demo*, 2008.

[50] Opnet projects. http:opnetprojects.com/.

[51] Praveen Kumar, Chris Yu, Yang Yuan, Nate Foster, Robert Kleinberg, and Robert Soulé. Yates: Rapid prototyping for traffic engineering systems. In *Proceedings of*

the Symposium on SDN Research, SOSR '18, New York, NY, USA, 2018. Association for Computing Machinery.

[52] Steven Gay, Pierre Schaus, and Stefano Vissicchio. REPETITA: repeatable experiments for performance evaluation of traffic-engineering algorithms. *CoRR*, abs/1710.08665, 2017.

[53] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, New York, NY, USA, 2010. Association for Computing Machinery.

[54] Stuart Mitchell, Stuart Mitchell Consulting, and Iain Dunning. Pulp: A linear programming toolkit for python, 2011.

[55] Lorenzo Saino, Cosmin Cocora, and George Pavlou. A toolchain for simplifying network simulation setup. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, SimuTools '13, page 82–91, Brussels, BEL, 2013. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[56] A. Tizghadam and A. Leon-Garcia. Betweenness centrality and resistance distance in communication networks. *IEEE Network*, 24(6):10–16, November 2010.

[57] Y. Bi, C. W. Tan, and A. Tang. Network utility maximization with path cardinality constraints. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[58] Antonio Nucci, Ashwin Sridharan, and Nina Taft. The problem of synthetically generating ip traffic matrices: Initial recommendations. *SIGCOMM Comput. Commun. Rev.*, 35(3):19–32, July 2005.

[59] Matthew Roughan, Albert Greenberg, Charles Kalmanek, Michael Rumsewicz, Jennifer Yates, and Yin Zhang. Experience in measuring backbone traffic variability: Models, metrics, measurements and meaning. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, IMW '02, page 91–92, New York, NY, USA, 2002. Association for Computing Machinery.

[60] Supratik Bhattacharyya, Christophe Diot, Nina Taft, and Jorjeta Jetcheva. Geographical and temporal characteristics of inter-pop flows: View from a single pop. *European Transactions on Telecommunications*, 13(1):5–22, 2002.

[61] Deep Medhi and Karthik Ramasamy. Chapter 4 - network flow models. In Deep Medhi and Karthik Ramasamy, editors, *Network Routing (Second Edition)*, The Morgan Kaufmann Series in Networking, pages 114 – 157. Morgan Kaufmann, Boston, second edition edition, 2018.

[62] D. Xu, M. Chiang, and J. Rexford. Deft: Distributed exponentially-weighted flow splitting. In *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, pages 71–79, May 2007.

[63] C. Hopps. Rfc2992: Analysis of an equal-cost multi-path algorithm, 2000.

[64] Mark Ziegelmann. *Constrained Shortest Paths and Related Problems - Constrained Network Optimization*. VDM Verlag, Saarbrücken, DEU, 2007.

[65] Kuo-Feng Hsu, Ryan Beckett, Ang Chen, Jennifer Rexford, and David Walker. Contra: A programmable system for performance-aware routing. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 701–721, Santa Clara, CA, February 2020. USENIX Association.

[66] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. Hula: Scalable load balancing using programmable data planes. In *Proceedings*

of the Symposium on SDN Research, SOSR '16, New York, NY, USA, 2016. Association for Computing Machinery.

[67] Cristian Hernandez Benet, Andreas J. Kassler, Theophilus Benson, and Gergely Pongracz. Mp-hula: Multipath transport aware load balancing using programmable data planes. In *Proceedings of the 2018 Morning Workshop on In-Network Computing*, NetCompute '18, page 7–13, New York, NY, USA, 2018. Association for Computing Machinery.

[68] Kuo-Feng Hsu, Praveen Tammana, Ryan Beckett, Ang Chen, Jennifer Rexford, and David Walker. Adaptive weighted traffic splitting in programmable data planes. In *Proceedings of the Symposium on SDN Research*, SOSR '20, page 103–109, New York, NY, USA, 2020. Association for Computing Machinery.

[69] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 503–514, New York, NY, USA, 2014. Association for Computing Machinery.

[70] M. M. Tajiki, B. Akbari, M. Shojafar, S. H. Ghasemi, M. L. Barazandeh, N. Mokari, L. Chiaraviglio, and M. Zink. Cect: computationally efficient congestion-avoidance and traffic engineering in software-defined cloud data centers. *Cluster Computing*, 21(4):1881–1897, Dec 2018.

[71] W. Quan, N. Cheng, M. Qin, H. Zhang, H. A. Chan, and X. Shen. Adaptive transmission control for software defined vehicular networks. *IEEE Wireless Communications Letters*, 8(3):653–656, 2019.

[72] Eric Gourdin and Olivier Klopfenstein. Comparison of different qos-oriented objectives for multicommodity flow routing optimization. In *Proceedings of the International Conference on Telecommunications (ICT*, 2006.

[73] Simon Balon, Fabian Skivée, and Guy Leduc. How well do traffic engineering objective functions meet te requirements? In Fernando Boavida, Thomas Plagemann, Burkhard Stiller, Cedric Westphal, and Edmundo Monteiro, editors, *NETWORKING 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, pages 75–86, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[74] Philipp Czerner and Harald Räcke. Compact oblivious routing in weighted graphs, 2020.

[75] M. Chiesa, G. Rétvári, and M. Schapira. Oblivious routing in ip networks. *IEEE/ACM Transactions on Networking*, 26(3):1292–1305, 2018.

[76] Leonard Kleinrock. *Communication Nets; Stochastic Message Flow and Delay*. Dover Publications, Inc., USA, 1972.

[77] Matthew Roughan. Simplifying the synthesis of internet traffic matrices. *SIGCOMM Comput. Commun. Rev.*, 35(5):93–96, October 2005.

[78] Bomin Mao, Fengxiao Tang, Zubair Md. Fadlullah, and Nei Kato. An intelligent route computation approach based on real-time deep learning strategy for software defined communication systems. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2019.

[79] L. Fratta, M. Gerla, and L. Kleinrock. The flow deviation method: An approach to store-and-forward communication network design. *Networks*, 3(2):97–133, 1973.

[80] Bernard Fortz, Luís Gouveia, and Martim Joyce-Moniz. Models for the piecewise linear unsplittable multicommodity flow problems. *European Journal of Operational Research*, 261(1):30 – 42, 2017.