Sustainable Networking: A Study on Power Optimization in DCNs Using Traffic-Aware Methods, Along with SD-DCNs Emulation Strategies

Thesis for the Degree of Doctor of Philosophy (PhD)

by Mohammed Al-shamarti

Supervisor:

Dr. Gergely Kovásznai

UNIVERSITY OF DEBRECEN

Doctoral Council of Natural Sciences and Information Technology

Doctoral School of Informatics

Debrecen, 2024

Hereby I declare that I prepared this thesis within the Doctoral Council of Natural Sciences and Information Technology, Doctoral School of Informatics, University of Debrecen in order to obtain a PhD Degree in Informatics at Debrecen University.

The results published in the dissertation are not reported in any other PhD theses.

Debrecen, 202. . . . . .

signature of the candidate

Hereby I confirm that Mohammed Al-shamarti candidate conducted his studies with my supervision within the Applied Information Technology and its Theoretical Background Program of the Doctoral School of Informatics between 2020 and 2024. The independent studies and research work of the candidate significantly contributed to the results published in the thesis.
I also declare that the results published in the thesis are not reported in any other theses.
I support the acceptance of the thesis.

Debrecen, 202. . . . . .

signature of the supervisor

Sustainable Networking: A Study on Power Optimization in DCNs Using Traffic-Aware Methods, Along with SD-DCNs Emulation Strategies

Dissertation submitted in partial fulfillment of the requirements
for the doctoral (PhD) degree in informatics

Written by Mohammed Al-shamarti, certified computer scientist.

Prepared in the framework of the Doctoral School of Inforamtics of University of Debrecen
(Applied Information Technology and its Theoretical Background programme)

Dissertation advisor: Dr. Gergely Kovásznai

The official opponents of the dissertation:

Dr. ....
Dr. ....

The evaluation committee:

chairperson: Dr. ....
members: Dr. ....
Dr. ....
Dr. ....
Dr. ....

The date of the dissertation defense: ..... 20...

List of abbreviations

DCNs . . . . . Data Center Networks

SDN . . . . . Software-Defined Networking

SD-DCNs . . . . Software-Defined Data Center Networks

CAGR . . . . . Compound Annual Growth Rate

CO2 . . . . . Carbon Dioxide

ML . . . . . Machine Learning

DL . . . . . Deep Learning

QoS . . . . . Quality of Service

SMT . . . . . Satisfiability Modulo Theories

GPUs . . . . . Graphics Processing Units

NNs . . . . . Neural Networks

SM-FPLF . . . . SMart-Fill Prefer Path First

EQ-PoPS . . . . Enhanced QoS-Aware Power-Optimized Path Selection

RIP . . . . . Routing Information Protocol

OSPF . . . . . Open Shortest Path First

CLI . . . . . command-line interface

SNMP . . . . . Simple Network Management Protocol

OF . . . . . OpenFlow

SVMs . . . . . Support Vector Machines

LP . . . . . Linear Programming

ILP . . . . . Integer linear programming

MILP . . . . . Mixed Integer Linear Programming

POD . . . . . Point of Delivery.

HVAC . . . . . Heating, Ventilation, and Air Conditioning

TCP . . . . . Transmission Control Protocol

UDP . . . . . User Datagram Protocol

ET . . . . . Elastic-Trees

LO . . . . . Linear Optimizer

GO . . . . . Greedy Optimizer

FNSS . . . . . Network Simulation Setup

API . . . . . Application Programming Interface

D-ITG . . . . . Traffic Manager Distribution Internet Traffic Generator

FPLF . . . . . Fill Preferred Link First

LU . . . . . Link-Utility

LC . . . . . Link-Cost

FSP . . . . . Fill-Shortest Path

LR . . . . . Linear Regression

VoIP . . . . . Voice over IP

ICMP . . . . . Internet Control Message Protocol

RF . . . . . Random Forests

GB . . . . . Gradient Boosting

PCA . . . . . Principal Component Analysis

FTP . . . . . File Transfer Protocol

ReLU . . . . . Rectifier Linear Unit

SGD . . . . . Stochastic Gradient Descent

Adam . . . . . Adaptive Moment Estimation

Contents

1

Introduction

THIS chapter provides a summary of the driving factors behind this study, along with a brief description of optimization methods that used throughout the thesis's chapters. It also outlines the research questions and highlights the contributions made to existing knowledge. Finally, the chapter offers an overview of the subsequent chapters within the thesis.

1.1 Motivation

Nowadays, energy efficiency is a hot topic in scientific research, and it is a topic of great importance in our society that relies heavily on technical information. Energy consumption in both wired and wireless networks poses a significant challenge for various reasons, including the need for optimizing energy use and reducing costs, especially in the context of Data Center Networks (DCNs) [1].

DCNs have gained increasing prominence in our daily lives due to the rapid growth of modern information technologies, such as the Internet of Things, Big Data, Cloud Computing, and Mobile Sensing Networks [2, 3]. Therefore, DCNs are designed with a focus on high reliability and stability, achieved through the implementation of *redundant links* and *sufficient capacity*.

Thus, the motivation for achieving energy-efficient usage of DCNs is closely linked to the increasing digitization of daily life. The demand for digital services is growing at a remarkable pace. Since 2010, the number of internet users worldwide has doubled, and global internet traffic has expanded twentyfold. For example, according to information from Statista[i] [4], in 2020, data generation and replication reached an unprecedented peak, totaling 64.2 zettabytes, exceeding earlier predictions. This surge can be attributed primarily to the

[i] `https://www.statista.com/statistics/871513/worldwide-data-created`

heightened demand resulting from the COVID-19 pandemic, with more individuals working and studying from home and engaging in home entertainment activities more frequently. Looking ahead, it is projected that global data production will exceed 180 zettabytes in the five years leading up to 2025, as shown in Figure 1.1.

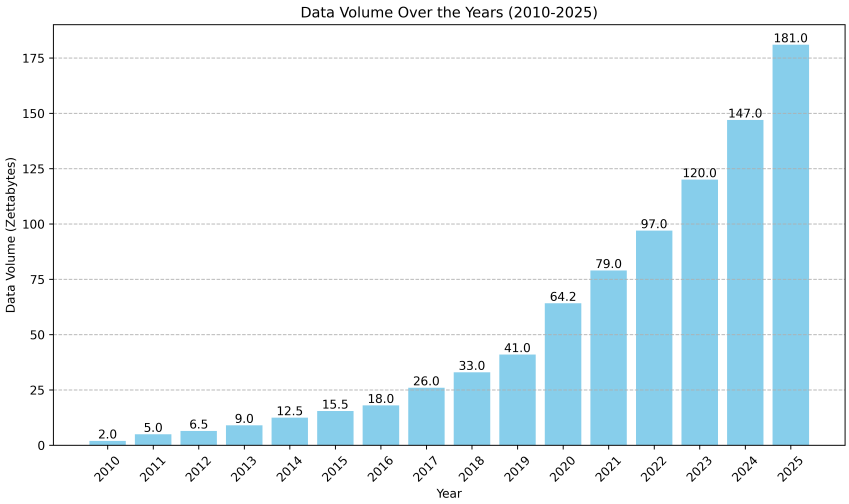


Figure 1.1: Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025 (in zettabytes) [4].

The same report indicated that only a small fraction of the newly generated data is preserved for the long term. Specifically, just about two percent (2%) of the data produced and used in 2020 was retained into 2021. Reflecting the substantial growth in data volume, the total storage capacity in data centers is expected to surge, with a projected Compound Annual Growth Rate (CAGR) of 19.2 percent between 2020 and 2025. By the end of 2020, the total installed storage capacity had already reached a significant 6.7 zettabytes. However, the proliferation of data centers and transmission networks that support digitalization has resulted in increased energy consumption. This has two significant impacts (1) higher costs for both of service providers and end-users[ii] [5], as well as (2) environmental concerns related to Carbon Dioxide (CO2) emissions[iii] [6, 7]. To address the challenges of increasing

[ii] Currently, Europe's energy grid is facing an unparalleled crisis. Since early 2021, wholesale costs of electricity and gas have increased by as much as 15 times, which has had devastating consequences on both individuals and companies.

[iii] Currently, large-scale alternatives to fossil fuels that are secure, affordable, and low-carbon are lacking globally. The connection between access to energy and greenhouse gas

energy demand and emissions, it is crucial to prioritize improving energy usage efficiency. This is particularly important in limiting the growth of energy demand from data centers and data transmission networks. Estimates indicate that in 2005, data centers consumed 153 TWh, and by 2010, this consumption had increased to 273 TWh of global electricity use, accounting for approximately $1 - 1.5\%$ of global electricity demand [8]. Furthermore, according to a research note from the Borderstep Institute, it is projected that the energy consumption of data centers worldwide may increase by a factor of 40 through 2030 [9].

On the other hand, a notable issue arises as network devices typically operate at *full capacity* 24 hours a day, resulting in substantial energy consumption. Unfortunately, these network devices are *underutilized* most of the time, leading to extremely low network energy efficiency [10]. Consequently, this problem has captured the attention of many researchers who are working to develop techniques that can save energy while still maintaining network performance, such as in [6].

Therefore, researchers, in collaboration with strong government and industry efforts on energy efficiency and renewable energy procurement, play a crucial role in mitigating and curbing energy demand and emissions growth over the next decade.

In the following sections, we will provide a brief description of the methods utilized in the various chapters of the thesis, including Linear Programming, Heuristic Algorithms, and Machine Learning. These optimizations were carried out within a real-time emulation environment in the context of Software-Defined Networking (SDN), which represents the new paradigm in network management.

1.2 The Way to Network Softwarization

In recent years, several studies have focused on managing and reducing energy consumption in DCNs. Various techniques have been proposed to reduce energy consumption in DCNs, with many of these techniques primarily targeting servers. Firstly, implementing intelligent cooling system management within the computational environment, as demonstrated in [11]. Then, Dynamic Voltage and Frequency Scaling (DVFS) techniques are utilized to manage power usage, as shown in [12]. More recently, studies have employed server

emissions is the aspect of energy that receives the most attention

consolidation strategies to minimize the number of physical machines in DCNs, considering the capacity of each machine to match the service demand, which is another well-known technique [13], [14]. However, fewer studies have focused on the routing layer, i.e., the network layer. This may be attributed to the limitations imposed by the physical substrates of legacy networks, which inherit drawbacks from the TCP/IP protocol stack.

The reason behind this is that the network forwarding devices of legacy networks are functional through the integration of two principal components: the *control plane* and the *data plane*. Consequently, legacy switches discover the network topology using distributed protocols to make decisions regarding the forwarding of incoming packets. These decisions are made based on protocols like Open Shortest Path First (OSPF) [15] and Routing Information Protocol (RIP) [16], which reside within the control plane.

On the other hand, the data plane makes forwarding decisions based on the information provided by the control plane layer. This decentralized operation requires network operators/administrators to manually configure a large number of switches using the command-line interface (CLI) to ensure the network functions correctly, which can be *time-consuming* and costly in terms of *efforts*.

Furthermore, sophisticated routing-aware techniques rely on prior knowledge of the network's state. In legacy networks, protocols such as Simple Network Management Protocol (SNMP) and tools such as IPFIX [17], and NetFlow [18], are used for analyzing the network's status. However, these legacy networks often involve diverse forwarding equipment from different suppliers, resulting in variations in capabilities and configurations. As a result, collecting accurate statistics and monitoring data can become *challenging*. For instance, the SNMP protocol may encounter discrepancies in the supported SNMP agents across different network devices, leading to inconsistencies in the collected statistics. Similarly, in the case of the reporting system, diverse network equipment can hinder the consistent collection of monitoring data.

Based on the aforementioned points, we have summarized the limitations of legacy networks as follows: (1) limited support for new ideas and innovations, (2) lack of management flexibility, with the *control plane* and *data plane* tightly coupled, necessitating changes to both when altering the control plane, (3) using distributed routing protocols such as OSPF and RIP requires a large number of multicast messages to construct the routing table of the switches, (4) an increased energy footprint.

The challenges posed by these inconveniences prompted network operators to seek alternatives to the inflexible proprietary solutions, with a desire for more *programmable* and open options that could facilitate the development of agile networks, departing from the traditional rigid framework. A research team at Stanford University introduced the OpenFlow protocol (OF) in 2008-2010 [19] as a revolutionary technology, which found application within the university campus network. OpenFlow serves as an interface allowing remote control of the forwarding tables within network components like switches and routers. Figure 1.2 presents the OpenFlow architecture, wherein one or more controllers manage network elements. An OpenFlow-enabled switch comprises two main elements: (1) a secure channel, which serves as a connection interface established over an encrypted channel, often using Secure Sockets Layer (SSL) and/or Transport Layer Security (TLS), between the controller and the switch, and (2) a flow table, which can be single or multiple and contains numerous *flow entries*, or forwarding rules, necessary for processing incoming packets. The OpenFlow has driven the development of new network paradigm named

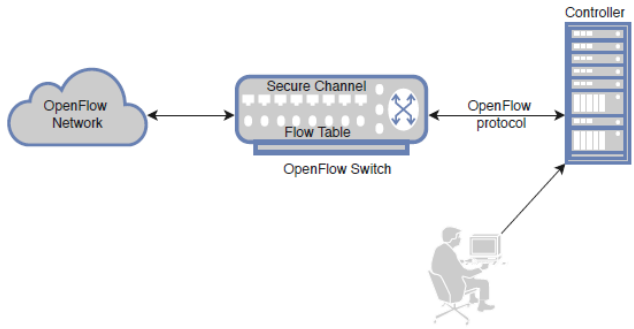


Figure 1.2: The basic OpenFlow switch architecture.

Software-Defined Networking (SDN) as a next generation. SDN emerges as a solution by separating the *control plane* from the *data plane*. This separation facilitates the introduction of new ideas, such as new routing schemes, traffic orchestration, and real-time monitoring, and offers unprecedented advantages in terms of addressing various network problems, including management, security, and energy efficiency.

The SDN paradigm makes network management more flexible and software-defined by placing the application layer on top of the controller. This arrangement allows for controlling the multitude of OpenFlow switches through secure OpenFlow channels, unifying the network style and enabling management from a single point, even when dealing with switches from different *suppliers* that

support the OpenFlow protocol. The OpenFlow switches are responsible for forwarding traffic, following the instructions received from the controller.

Therefore, one of the goals of the current thesis is to examine the flexibility and dependability of the SDN paradigm in facilitating the optimization of DCNs. This will be achieved by implementing the proposed models on top of the SDN controller to manage network state and install rules that aim to minimize power usage in DCNs.

1.3 Optimization Approaches

This section presents a brief introduction to the optimization methods used in the current thesis to optimize power usage in DCNs. More details will be provided in the next chapters.

1.3.1 Integer linear programming

Integer Linear Programming (ILP) is a mathematical problem used to optimize (MAX/MIN) the objective function (multiple or single objective function) subject to linear constraints, where some or all of the decision variables are required to be integers. If all variables are integers, the problem is termed a pure ILP. On the other hand, when both continuous and integer variables coexist, it is called Mixed Integer Linear Programming (MILP). Following the general formulation of the matrix notation, as shown in [20], the ILP programming is written as:

$$\begin{aligned} &\text{Minimize} && \mathbf{cx} \\ &\text{Subject to} && \mathbf{Bx} \geq \mathbf{b} \\ &&& \mathbf{x} \geq \mathbf{0} \text{ and integer} \end{aligned}$$

In the above formulation, $\mathbf{x}$ is the vector of integer variables, and $\mathbf{c}$ is the coefficient vector of $\mathbf{x}$. The objective function is subject to the constraints, which are represented here by $\mathbf{B}$ and $\mathbf{b}$; they are the coefficient matrix and the right-hand side of these constraints, respectively.

The elements of $\mathbf{B}$, $\mathbf{b}$, and $\mathbf{c}$ are constants. In many cases, the constraints are equations and of the type ($\leq$).

It is worth noting that for an ILP problem with v variables and m constraints, $\mathbf{x}$ is a $v \times 1$ vector, $\mathbf{c}$ is a $1 \times v$ vector, $\mathbf{B}$ is an $m \times v$ matrix, and $\mathbf{b}$ is an $m \times 1$ vector.

Note that any variable in $\mathbf{x}$ can assume any nonnegative integer, as long as all constraints are satisfied. In some special cases, all integer variables are restricted to be Boolean, i.e., either 0 or 1. Such an ILP problem is named a binary ILP (BILP) or binary MILP (BMILP). In this case, the constraint set ($\mathbf{x} \geq 0$, and integer) is replaced by $\mathbf{x} \in \{0, 1\}$.

There are numerous solution methods available for solving ILP or MILP problems. However, none of these methods are considered reliable in terms of computational efficiency. Moreover, the majority of these methods can be categorized into cutting-plane techniques, enumeration techniques, or, in some cases, a combination of both [21]. Commercial solver tools, such as CPLEX[iv] and Gurobi[v], are commonly employed to find solutions.

Note: This thesis employed ILP to optimize both DCNs workloads toward more efficient power usage. Then compared the results with heuristic approaches in terms of the quality of the solution and time cost.

1.3.2 Heuristic Methods

Heuristic algorithms constitute a class of problem-solving approaches utilized to discover effective solutions for intricate problems. They prove particularly valuable in situations where no precise algorithm exists to provide an exact solution or where the computational expense of using such an algorithm is prohibitive [22].

In optimizing DCNs, heuristic methods play a crucial role by offering practical and efficient approaches to address the complex challenges involved in managing and enhancing network performance in *real-time*, as discussed in this thesis. In contrast to time-consuming formal methods optimization, heuristic methods are employed to optimize various functions in DCNs. For instance, they are utilized to determine optimal paths for data traffic in a DCN, place

[iv] https://github.com/aimms/documentation/blob/master/platform/solvers/cplex.rst#cplex

[v] https://www.solver.com/gurobi-solver-engine

virtual machines strategically, implement energy-efficient routing, and manage failure recovery. Consequently, heuristic methods present practical and computationally efficient solutions to the challenges of managing large, dynamic, and complex networks [10].

A variety of common heuristics find application in heuristic algorithms, including Bin Packing[vi], hill climbing, simulated annealing, genetic algorithms, and particle swarm optimization.

1.3.3 Machine Learning

Machine Learning (ML) stands out as one of the most promising methods for solving a diverse range of scientific problems, including Natural Language Processing, Face and Speech Recognition, and time series pattern extraction [23].

The advantage of utilizing ML algorithms lies in their inspectability, allowing us to comprehend what they have learned as a feature that can be challenging for certain algorithms. As illustrated in Figure 1.3, the learning algorithm does not only finds a solution but also permits inspection to assess the quality of that solution. When needed, it can iterate for further problem investigation and to enhance the resulting solution.

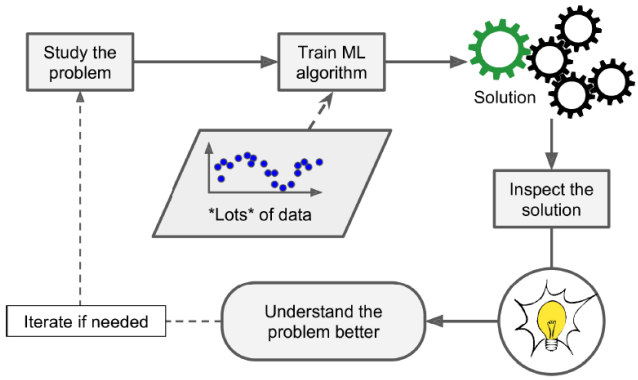


Figure 1.3: Learning and Inspection to improve the resulting solution.

[vi] Bin Packing could be considered as a deterministic algorithm if carefully consider all possibilities to find the best arrangement of variables (flows in our case), which it impossible in networking where the conditions can change rapidly. Therefore, algorithms sacrifice optimality for efficiency and speed.

ML models are broadly categorized into three major types: (1) Supervised Learning, (2) Unsupervised Learning, and (3) and Reinforcement Learning. The current thesis employs Supervised Techniques to tackle a significant problem in SD-DCNs. In Supervised Techniques, the training data fed to the learning process includes the desired solutions (labels). These algorithms have proven effective in tasks such as classifying objects into different categories and predicting numerical values. Specifically, classification involves sorting objects into distinct categories, while predicting numerical values typically involves regression. Noteworthy algorithms in this category include Linear/Logistic Regression, Support Vector Machines (SVMs), and Neural Networks[vii] (NNs).

The thesis utilized supervised techniques to develop a new classifier model capable of real-time classification of a wide range of traffic. This enhancement boosts the SDN controller's ability to prioritize Quality of Service (QoS) for sensitive traffic, minimize time delays, and prevent packet drops. Additionally, the learning algorithms can be used to analyze traffic patterns and predict future traffic demand. This, in turn, assists SDN controllers in making real-time adjustments and reconfiguring the network devices within their control domain to optimize power usage and reduce consumption.

Moreover, ML-based approaches can be employed to analyze the behavioral patterns of network devices, detect the efficiency of each individual device, and optimize network configuration for more efficient power usage, as highlighted in our survey [10].

In conclusion, learning-based approaches may hold the potential to significantly reduce power consumption in SD-DCNs.. Therefore, another goal of the current thesis is to develop and train machine learning algorithms aimed at improving Quality of Service (QoS) and energy efficiency, thereby reducing the carbon footprint of SD-DCNs.

1.4 Research Questions

The thesis aims to answer specific research questions regarding the possibility of reducing power consumption in DCNs, as well as the reliability and adaptability of the SDN paradigm in improving the efficiency of power usage in DCNs. These research questions can be summarized as follows:

[vii] Some neural network architectures, such as autoencoders and restricted Boltzmann machines, can be unsupervised.

1. What are the limitations of existing techniques in terms of power saving, suitability for real-time applications, and the comprehensiveness of power optimization experiments?
2. How can mathematical optimization approaches be employed to improve power usage in DCNs?
3. Why is the combination of Software-Defined Networking and Data Center Networks (SD-DCNs) considered a cornerstone for optimizing DCNs resources?
4. How to adopt a traffic-aware approach to reduce the power consumption in SD-DCNs?
5. How to predict the class of service of each flow in SD-DCNs, as a way to improve the quality of flow, while considering power usage optimization?

1.5 Contributions

The contributions of this thesis to addressing above research questions include:

1. The thesis provides a survey of state-of-the-art methods for reducing energy consumption via: (1) enhanced scheduling and, (2) enhanced aggregation of traffic flows using SD-DCN, focusing on the advantages and disadvantages of these approaches [10].
2. The thesis provides a formal model for using a minimum topology active subset, i.e., ports, links, and switches, to accommodate the current traffic demand and turning off the remaining idle devices for DCNs [24].
3. The thesis highlights the strength of the SD-DCNs combination by consolidating the network state into a central point, the SDN controller, to achieve more efficient resource optimization.
4. The thesis proposes an adaptive routing algorithm for efficient power consumption in SD-DCNs. The proposed approach is based on a new link utility-based heuristic algorithm called Fill Prefered Link First (FPLF) that strives for a balance between energy consumption and performance [25].
5. The thesis demonstrates how to classify the network traffic flows so that the QoS of each flow-class can be guaranteed efficiently [26].

1.6 Roadmap of the Thesis

This thesis is organized as follows: *Chapter 2* provides background information and a literature review, which have been conducted based on research issues related to the introduced research questions. The main aim of this chapter is to outline general optimization methods that have been used in the literature and to identify the current gaps. *Chapter 3* provides a detailed study of how to apply formal approaches to optimize power consumption in DCNs, where an ILP model is proposed and solved using leading optimization tools. Additionally, it introduces the Portfolio Solver for DCN Optimization called NEO-DCN. *Chapter 4* presents details of the emulation environment along with the required tools used to emulate the proposed frameworks in the next chapters. *Chapter 5* provides a heuristic approach used to adopt a traffic-aware strategy to reduce power consumption without deteriorating the performance of real-time applications. In *Chapter 6*, machine learning is used to classify a wide range of DCNs traffic flow.

In both Chapters 5 and 6, the models are proposed, implemented, and an experimental study is conducted to evaluate the performance of the new approaches. *Chapter 7* contains the research conclusions and also provides an overview of the strengths and weaknesses of the thesis. It also discusses potential future directions for further exploration to achieve advancements in this research area.

2

Background and Literature

Contents

2.1 Introduction

The purpose of this chapter is to shortly explore the essential components of data center network structures, various topology types, and the power profiling of network switches. In addition to this, we have undertaken a survey of contemporary methods designed to mitigate energy consumption. The primary focus revolves around two key approaches: (1) enhanced scheduling approaches, and (2) improved aggregation approaches for traffic flows in DCN. The ensuing discussion will delve into the respective advantages and disadvantages associated with each of these approaches.

It is worth noting that a substantial portion of the content in this chapter draws upon the insights presented in the work of [10].

2.2 Structure of a Data Center

Data centers aim to provide reliable and scalable computing infrastructure for massive internet services [27]. Data centers promise several benefits such as: (1) flexibility without sacrificing forwarding performance; (2) high efficiency which is achieved by optimizing routing; (3) ease of deployment/administration; and finally, (4) cost reduction [6].

The data center's structure encompasses various components. Notably, the physical infrastructure includes the building itself, power infrastructure (comprising high-capacity electrical systems and backup generators), and cooling systems, such as HVAC [i]. Additionally, the networking infrastructure involves critical elements like switches, routers, load balancers, and server racks and cabinets that house server hardware and storage systems.

Several studies emphasize optimizing power usage and cooling systems for physical devices like servers, as mentioned in section 1.1. However, this thesis uniquely focuses on switches, aiming to enhance their efficiency in *proportion* to traffic load. Recent research highlights the substantial energy consumption of DCNs. By 2020, the estimated energy consumption in the US surpassed 139 billion kWh, with interconnection devices (switches and links) accounting for *10% to 20%* of the total energy consumption [1]. The power consumption of

[i] HVAC (Heating, Ventilation, and Air Conditioning) systems are employed to manage the heat generated by servers and other equipment.

switches and links in DCNs depends on the design of the device, as well as the network traffic and workloads transmitted, as demonstrated in the following section.

2.3 Power Profiling of Switches

The topology of a DCN can be either homogeneous or heterogeneous. In a homogeneous topology, all switches in the network are of the same type, which can provide advantages in terms of *simplicity* and ease of *management*, because all switches can be configured and managed in a similar way. On the other hand, in a heterogeneous topology, switches from multiple vendors or switches with different capabilities or configurations are used. This can provide greater flexibility and functionality, but it can also make the network more complex to manage and to troubleshoot. Therefore, DCNs are often designed to be homogeneous [28].

The power consumption of DCN switches can be measured *dynamically* or *statically*. Dynamic measurement measures the power consumption of active links and the power depends on the speed of the link. Different devices have different power profiles. The power profile of a device characterizes the power it consumes as a function of the bit rate of the switch port under consideration. To provide benchmark values for the power involved, we cite the authors of [29] who state that the power consumed per port, when the switch port is operating at the speeds 10 Mbps, 100 Mbps, and 1 Gbps, is 63 mW/port, 260 mW/port, and 913 mW/port, respectively. The switch considered was the commercial Pronto switch, which has OpenFlow capabilities. In contrast, the NEC ProgrammableFlow Networking Suite "PF5240" OpenFlow switch consumes significantly more power, consuming approximately 0.2 W/port, when it operates at 10 Mbps. In conclusion, the design of the switch plays a large role in its power consumption. In this case the Pronto switch consumes 31.5% of the power of the NEC switch when operating at the same data rate. Given that the network manager may not have a choice in the equipment at their disposal, the network manager's focus should be on the manner in which the switch is used, to reduce power consumption. The present contribution looks to optimize *link* usage to reduce power consumption.

The static measurement approach involves assessing the power consumed by the components that are responsible for keeping the system operational, maintaining

connectivity, and performing background tasks. This includes considering the power consumed by components such as chassis, fans, and switching fabric.

2.4 Network Energy Problem

The predominant observation in the literature is that the DCN often operates in a low-load state, as extensively noted [6]. The problem is that energy consumption of DCN resources, such as links, is wasteful when the utilization level is *low*. This is because engaging certain equipment in DCN activities, e.g., routing, does not justify the additional cost of the associated energy consumption [27].

This thesis sets out to address this issue by proposing an approach that ensures OpenFlow switches operate in *proportion* to the traffic demand of the DCN. The proposed strategy involves a *gradual increase* in the number of active links as traffic demand rises, ensuring optimal resource utilization. Conversely, during periods of low traffic demand, the approach adopts a contrasting strategy to minimize energy consumption. Additionally, the thesis takes into account other crucial QoS metrics to comprehensively address the network energy problem.

2.5 Data Center Topologies

A DCN topology intricately organizes the channels that facilitate connections among nodes, servers, and computing devices, encompassing both wired and wireless network connections. This organizational framework delineates the communication methods among devices within the DCN, playing a pivotal role in addressing critical considerations such as load balancing, link failure, and power consumption. The more commonly employed types of DCNs' topologies are: (1) Two-Tier [30], (2) Three-Tier [31], (3) DCell [32], (4) BCube [33], and (5) Fat-Tree topology [34]. In this thesis, for various reasons, all experiments were conducted on the Fat-tree topology. The fat-tree architecture was developed to address several key issues in DCN. It aims to eliminate single points of failure inherent in hierarchical architectures, reduce congestion and oversubscription ratios, and achieve approximately non-blocking paths with full bandwidth utilization.

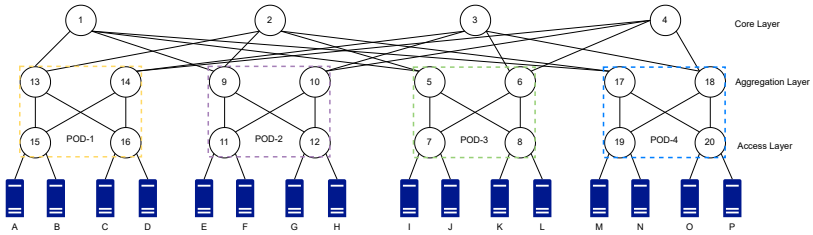


Figure 2.1: 4-ary fat-tree DCN topology.

An important feature of this architecture is the use of the same model of energy-efficient and *uniformly* configured switches across all layers. This homogeneous topology not only enhances performance but also contributes to cost-effectiveness in setting up a Fat-tree data center [34].

2.5.1 Fat-Tree Topology Architecture

The architecture of the fat-tree topology operates with network switches across multiple layers, including aggregate and core layers, as illustrated in Figure 2.1. This fat-tree topology is presented as a 4-ary structure, constructed with $k = 4$ Point of Delivery (PODs). Each POD consists of two layers of $k/2$ switches.

The core layer is composed of $(k/2)^2$ k-port core switches, with each switch establishing connections to all k PODs. In contrast, switches in the access layer are linked to $k/2$ servers, and the remaining ports are connected to switches in the aggregation layer.

2.6 Routing Optimization Techniques

The authors of [35] analyzed the traffic of a wide range of DCN network datasets belonging to different layers of DCN topologies and the results showed that the link utilization was low and varied from one layer to another. The low-utilization links motivated researchers to propose new approaches that were more energy-aware than commonly used routing algorithms (e.g., Equal Cost Multiple Path (ECMP)). To address this problem, two types of methods have been proposed: (1) flow aggregation techniques, and (2) flow scheduling techniques.

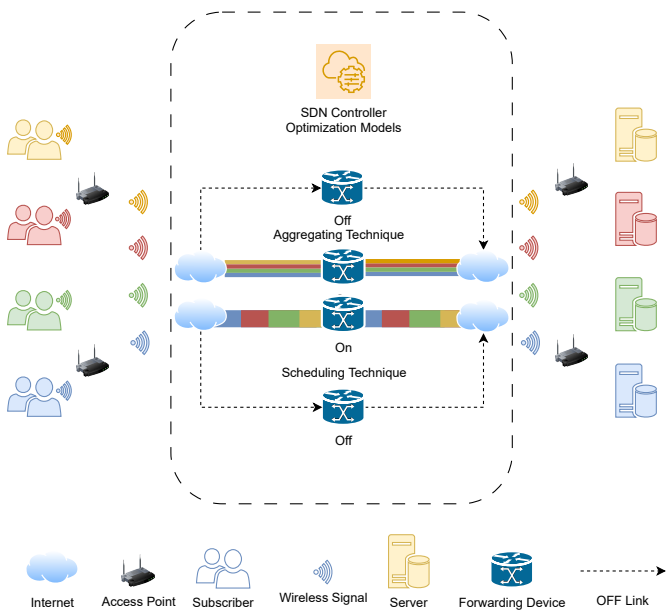


Figure 2.2: illustrates Flow scheduling/aggregation techniques.

2.6.1 Flow Aggregation Techniques

Flow aggregation techniques consolidate data flows into a smaller set of links and switches that are sufficient to support existing data traffic demands, subject to a tolerance to a certain level of delay, packets loss, etc. To achieve minimum power consumption for a specific traffic matrix, switches and ports that are being used unnecessarily are put into sleep or shutdown mode. Figure 2.2 shows how four flows share one link fairly based on the Transmission Control Protocol (TCP) sharing scheme. The disadvantage of these techniques is that using only a subset of the switches and links, a sub-topology, may result in performance degradation, which is typically characterized by a QoS measure. This QoS measure may indicate the significance of increases in delay time (i.e., due to computational complexity of the output solutions), or the extent to which links with higher utilization become overloaded and more susceptible to unplanned failures [36]. Balancing between the level of energy consumption and routing techniques that meet a desired QoS is of great importance. Next, we introduce and discuss the concept of flow aggregation.

2.6.1.1 Methods

The first analysis on Elastic-Trees (ET) was published in 2010 [27] by researchers from Deutsche Telekom and Stanford University. They considered three optimization techniques: Linear Optimizers (LO), Greedy Optimizers (GO), and ET. All of these optimizers work to consolidate traffic into a small subset of links that can handle the traffic volume. The results showed that LO are the worst due to their high computational complexity and time cost when the number of switches is high, while ET outperformed GO and improved link switch utilization. The authors evaluated ET using both simulations and experiments on a real network. They found that compared to traditional network architectures it could save significant amounts of energy. However, the study did not consider the correlation between flows.

To address the high correlation between flows, the CARPO (CoRrelation-aware Power Optimization) algorithm in [37] aiming to reduce energy consumption in a DCN, dynamically consolidated traffic flows on a small set of links and switches, and switched off idle network components. CARPO uses correlation analysis among flows to consolidate traffic flows with low correlation while keeping the QoS at an acceptable level. A heuristic algorithm is used to find a consolidation and rate configuration solution with acceptable runtime overheads. CARPO introduced parameters to represent flow correlation, and the results showed that this extension of the ET, led to a power saving of 46%.

The study in [38] suggested a platform composed of both software and hardware components, because there was no experimental environment available to test the optimization model. The software part was composed of monitoring (to check the state of the network), an optimizer (to calculate the subset of the topology), power controller (to change the state of the devices on/off), and routing algorithm (to calculate the paths). The hardware parts were composed of a traffic generator and power measuring device implemented using the NetFPGA platform. Experiments investigated the effect of changing the frequency of the clock rate and the traffic consolidation on the power consumption in the DCN according to traffic demand. Unfortunately, the authors did not provide sufficient details about the algorithms or the API interfaces that gave access to the components.

The authors reported that the traffic patterns of GEANT networks are generally regular and predictable in [39]. To save power, they divided the traffic into intervals, and implemented link sleeping, which temporarily suspends certain

links when the minimum or maximum link utility thresholds were met. They also rerouted the flows among the nodes of the GEANT network topology to maximize flow through the active links. Additionally, the study considered the network's performance under high-traffic conditions and balanced the traffic appropriately.

The study described in [40] investigated the effect of using different values of over-subscription factors on reducing power consumption in a DCN, which was using traffic aggregation strategies. Three algorithms, First-Fit, Best-Fit, and Worst-Fit, were evaluated using Python and the Network Simulation Setup (FNSS). Different network workloads (20%, 50%, and 80%) and oversubscription factors ($1:1$, $1:5$, and $1:20$) were tested. Devices that were not enrolled in the current routing state of the network were disconnected. The authors reported that up to a 70.02% power saving could be achieved when the over-subscription factor was $1:20$ and the topology size was $k = 12$. These results conform with intuition because of the increase in the bandwidth and the number of links and switches in the topology.

Motivated by the need for an adaptive routing framework for SD-DCN that can optimize power consumption while maintaining good network performance, our contribution in [25] proposed an adaptive routing algorithm that considers factors such as link utilization and switch power consumption to determine the most energy-efficient path for network traffic in real-time. The algorithm used an ILP model and a heuristic algorithm. The authors argue that the ILP model is costly and the time for finding a solution increases dramatically when 100 flows are injected simultaneously into the DCN. Conversely, the algorithm is integrated into an SDN controller and uses the OpenFlow protocol to communicate with switches. The proposed algorithms outperform existing routing algorithms such as ECMP in terms of energy consumption and network performance, as measured by the number of dropped packets. The research group built on this work and contributed new findings for the ILP model in [24]. Evaluations were conducted using LinGo [41], which is one of the computationally expensive commercial solvers. The authors of [25] re-implemented the model using solvers such as GUROBI, CP-SAT, and so on. Experiments were implemented in the authors' proposed tool NEO-DCN. This comparison revealed that GUROBI outperformed the other solvers by one or two orders of magnitude for different traffic patterns.

OpenFlow switches are increasingly being used in data centers due to their potential to reduce energy consumption. However, the existing OpenFlow protocol does not include any mechanisms to optimize energy consumption. In [42], a power-aware extension to the OpenFlow protocol that does not

compromise network performance was proposed. The extension defined new control messages in the OpenFlow standard such as OFPT-ORT-MOD and designed an OpenFlow Switch Controller (OSC) that can turn on and off switches and disable-enable ports of NetFPGA-based OpenFlow switches [38]. The authors conducted experiments to evaluate the energy savings achieved by the proposed extension, using a prototype implementation of the proposed extension on an OpenFlow switch. The experiments were designed to evaluate the energy savings achieved by the proposed extension. However, the authors did not provide sufficient details about the methods and the experiments conducted. Further research is needed to evaluate the effectiveness of these extensions. Table 2.1 summarizes the flow aggregation methods according to the adopted evaluation criteria and the main objectives.

Table 2.1: Flow aggregation methods for energy efficiency.

Study	Experiments				Objective	QoS	SDN
	Environment	Traffic	Topology	Controller			
Elastic Tree [27].	testbed	R	Fat-Tree	NOX	Link	-	√
CARPO [37].	testbed	R	Fat-Tree	-	Link	√	-
NetFPGA OpenFlow-based Platform [38].	testbed	R & A	Fat-Tree	NOX	Switch & Link	-	√
MTSDPFPR [39].	Mininet	R	GEANT	Floodlight	Link	√	√
Aggregation Strategies [40].	FNSS & Python	A	Fat-Tree	-	Link	-	√
OSC [38].	testbed	A	-	NOX	switch & Link	-	√
FPLF [25].	Mininet	A	Fat-Tree	POX	Link	√	√

2.6.2 Flow Scheduling Techniques

Since the SDN controller maintains a global view of the underlay DCN infrastructure, it can calculate deadlines for flows and their size. This ability has motivated the development of new *scheduling algorithms* to manage the transmission of flows through a sequence of queues. These algorithms send the flows sequentially. This allows the flows to monopolise all the links of the path they traverse, using their full capacity, subject to the deadline and flow-size constraints. Figure 2.2 shows how four flows are scheduled in a queue for transmission with a full bandwidth. A disadvantage is that these techniques are not appropriate for time-sensitive traffic (i.e., video streaming and VoIP [43]). Flows with higher priorities can be preemptively routed along the paths of other flows with lower priorities. Coupled with this, some applications in DCNs might not require large bandwidth, and can be slow, and thus not saturate the link capacity. Consequently, the link will be underutilized.

2.6.2.1 Methods

To avoid collisions and to achieve efficient power usage in DCNs, the authors in [44] proposed a technique that combined flow preemption and energy-aware routing to reduce energy consumption. Flows were divided into two lists: a sending list and a waiting list. When a new flow entered the DCN, the algorithm checked the flow's priority based on its size. The flow either interrupted an ongoing data transfer and was directly routed to the destination, or was moved to the waiting list based on its assigned priority. Simulations were used to evaluate the performance of their technique and to compare it to other state-of-the-art techniques, such as ECMP. They used the ns-3 network simulator and a Fat-Tree topology. Results suggest that it can be a practical and effective way to make DCNs more energy-efficient.

The authors of [45] were motivated by the importance of implementing exclusion flow routing techniques on different topologies. They proposed a scheduling algorithm based on the priorities of flows, which assigned higher scheduling priority to smaller flow sizes, as in [44]. Experiments were conducted on two types of DCN topologies, BCube and Fat-Tree, using OpenFlow as the south-bound API. Two metrics were used to evaluate the algorithm's performance: the flow arrival rate and the amount of power saved. The results showed an improvement in the utilization ratio of active links, leading to a reduction in power consumption in both topologies, along with the elimination of traffic congestion on active links.

The authors in [46] were motivated by the need to find an optimal subset to optimize power usage. A convex objective function was used to model energy consumption. Two strategies were pursued, an optimal combinatorial algorithm which was composed of two components (Most-Critical-First and Shortest Path (MCF-SP)) and a Random Scheduler (RS). MCF-SP managed flow scheduling based on the weight and the deadline of the flows. All the flows were sorted according to an Earliest Deadline First (EDF) policy. Power usage was optimized by minimizing the transmission rate of the flows per unit of time. By modeling the problem as a convex optimization problem the authors showed that MCF-SP found the optimal solution to the Deadline-Constrained Flow Scheduling (DCFS) problem under the assumption that flows were routed exclusively through a virtual circuit. Conversely, the RS involved relaxing an NP-hard problem, the Deadline-Constrained Flow Scheduling and Routing (DCFSR) problem. Relaxation consisted of finding an approximation based on a

Fractional Multi-Commodity Flow (F-MCF) problem. The power optimization was based on two criteria: a minimum transmission rate criteria for the flows and a usage criteria for the links. The evaluation procedure was conducted using Python, however, the description of the set-up makes comprehensive understanding and reconstruction of the results challenging.

Because the aggregation method was not suitable for network-limited flows, the application generated traffic at a high bit rate, and the flow's throughput depended on the network capacity of the routing path. To overcome this challenge a flow scheduling approach named "Willow" was proposed in [47], which took into consideration both the number of switches involved and the durations of their frame working times. A greedy approximate algorithm was designed that scheduled flows in a real-time manner. The proposed algorithm achieved a network energy consumption saving of 60% of network energy compared to the conventional ECMP scheduling approach.

To overcome the problem of greed in flow selection when finding an energy-efficient path in [46], the authors of [48] proposed an approach that aggregated the flows with a similar deadline into a harmonic flow set, and scheduled them with higher priority, resulting in increased link utilization. The resulting scheduling algorithm was named "BEERS". It ran as a model in the SDN controller. Simulations were run using the OMNeT++ simulator, and results showed improvements in the link utilization and the energy consumption compared with the Exclusive Routing (EXR) algorithm. However, since all the flows competed for links in different periods, it was challenging to form a harmonic flow set at certain points in time, which posed a challenge for the algorithm. The authors extended the BEERs approach and conducted extensive experiments with two types of topology [49]. The results showed that their algorithm could reduce overall energy consumption with respect to traffic volume, and that it could also reduce the average flow completion time.

Conventional methods for reducing power consumption, such as turning off unused switches, can negatively impact network performance. To address this issue, the authors in [50] proposed a dynamic flow scheduling algorithm that balanced the workload on network switches to reduce power consumption. The algorithm considered the flow size, a threshold value that controlled the time delay. The algorithm is evaluated using a simulation-based approach, using OPNET, and the results showed that the algorithm could be an effective solution for reducing power consumption in DCNs without sacrificing performance.

For non-real-time experiment scenarios, a Deep Reinforcement Learning (DRL) based energy-efficient routing algorithm called Deep Q-Network Energy-Efficient

Routing (DQN-EER) was proposed in [51]. The algorithm learnt to select the most Energy-Efficient Paths (EEP) for traffic flows in DCNs. The training of the algorithm was done using a Markov Decision Process (MDP), and Python language was used for programming the algorithm. The results indicated that the algorithm consumes similar energy as the baseline method, CPLEX [ii] solver. However, its calculation time is significantly less than CPLEX, especially in scenarios with a large number of flows. It is important to note that this technique might not be suitable for real-time simulations due to time cost.

Table 2.2 summarizes the flow scheduling methods according to the adopted evaluation criteria and the main objectives.

Table 2.2: Flow scheduling methods for energy efficiency.

Study	Experiments				Objective	QoS	SDN
	Environment	Traffic	Topology	Controller			
Preemptive Flow [44].	NS-3	A	Fat-Tree	-	switch	√	-
Green Data Center [45].	-	A	B-Cube & Fat-Tree	-	switch	√	√
BEERS-1 [48].	OMNeT++	A	Fat-Tree & BCube	-	switch	√	√
Willow [47].	-	A	Fat-Tree	-	switch	√	√
BEERS-2 [49].	OMNeT++	A	Fat-Tree & BCube	-	switch	√	√
FLOWP [46].	OPNET	A	Fat-Tree	-	switch	√	√

2.7 Conclusion

In summary, our investigation reveals that energy optimization techniques currently undergoing active research prominently feature scheduling and flow aggregation methods. This chapter aims to fill a literature gap by providing a comprehensive review of state-of-the-art SDN-based approaches for traffic scheduling and aggregation in DCNs. It analyzes their respective advantages and disadvantages, identifying several key points where further exploration is needed: (1) There is a lack of studies employing the exact method such as a ILP to determine the optimal subset of ports and switches recommended for handling traffic demand, (2) The absence of heuristic algorithms capable of managing multiple flows and dynamically adjusting the subset solution to meet both QoS criteria such as drop packets, and flow demand, and (3) A gap exists in utilizing deep learning techniques for classifying or predicting traffic demand in SD-DCNs, which could significantly contribute to guaranteeing the quality of service for each flow class, and power usage in SD-DCNs. Therefore, future research avenues should focus on leveraging machine learning to optimize traffic

[ii] `https://github.com/aimms/documentation/blob/master/platform/solvers/cplex.rst#cplex`

algorithms and exploring the potential to identify the optimal active subset in real-time for both scheduling and aggregation techniques.

2.8 Key Findings

Question: What are the limitations of existing techniques in terms of power saving, suitability for real-time applications, and the comprehensiveness of power optimization experiments?

Answer: The chapter demonstrates the need to provide the network community with extensive evidence of the effectiveness of formal methods in power optimization for DCNs. It introduces the necessity of a robust monitoring model with a traffic consolidation algorithm to enhance power usage in DCNs, along with ML techniques to identify and manage different types of traffic in DCNs.

Next Step: The upcoming chapter will illustrate how formal methods can be employed to optimize DCNs by turning off unneeded equipment.

3

Formal Methods

Contents

3.1 Introduction

In Chapter 2, we observed that computer networking equipment is designed to accommodate network traffic. However, network devices typically operate at full capacity 24 hours a day, leading to significant energy consumption. For the majority of the time, it has been reported that the DCN is in a low-load state [6]. The problem is that energy consumption of DCN resources, such as links, is wasteful when the utilization level is low. This is because engaging certain equipment in DCN activities, e.g., routing, does not justify the additional cost of the associated energy consumption [27]. Therefore, in this chapter, we propose an ILP model that causes a gradual increase in the number of active links as the traffic demand increases, and to strive for a balance between energy consumption and performance.

Initially, we solved the model using the LinGo solver, and we presented the results described in [25]. Since the results were not promising, this motivated us to explore further using other ILP solvers to enhance the results with more realistic benchmarks, as shown in [24].

We evaluate the proposed model using our introduced tool, NEO-DCN, for DCN optimization, designed to interface with various solvers and run them in versatile combinations on our ILP model to find the optimal solution. The source code and benchmarks are publicly available[i], fostering further research in energy-efficient networking. The chapter reports on experiments under three communication patterns (near, long, and random), measuring runtime and memory consumption in order to evaluate the performance of different ILP solvers.

It is worth noting that a substantial portion of the content in this chapter draws upon the insights presented in the work of [24].

3.2 Related Work

This section delineates the robustness and limitations of recent formal approaches addressing the challenge of decreasing power consumption in the network routing problem.

[i] `https://github.com/kovasz/neO-DCN`

In [27], the authors developed a formal model to compute a minimum-power network subset. The objective function includes two binary variables for each switch and link, with constraints representing Multi-Commodity Network Flow, Power Minimization, and Flow Split. Input parameters for the model encompass the traffic matrix, switch power model, and topology, yielding a subset of the original topology and per-flow route information. Notably, the study primarily concentrates on the manageable number of nodes in the topology, neglecting considerations for network performance. In contrast, our current model calculates the minimum number of links satisfying the traffic matrix, based on the utilization matrix.

The authors in [46] proposed a 0-1 ILP model to minimize the power of idle line cards and integrated chassis by switching them to sleep, under link utilization and packet delay constraints. Meanwhile, the study proposes two heuristic algorithms for the same purpose and reports on experiments executed in two scenarios: synthetic topology and real-life topology named CERNET. In the same context, another study in [52] proposed an ILP model with a multi-objective function to minimize the sum of the energy consumption of switches and links. The study limits the links' maximum and minimum utility to manage the trade-off between the power consumption and the network performance. Nevertheless, neither study presents any experimental result with the ILP model, and the algorithms were experimented outside of any DCN.

The study in [50] proposed a data center scheduling algorithm called FLOWP, besides an ILP formula, with the aim of optimizing power consumption and QoS. The formula considers a minimum threshold for the efficiency of links and switches. The results showed that QoS is improved compared to the approach in [27]. However, similar to [46], the study does not show any experimental result with the ILP model. Experiments are conducted on a heuristic algorithm only.

Our contribution in [25] presented an ILP model with the ability to manage multiple paths to save power consumption and balance the load during over-loaded times. The study conducted experiments with the model using the optimization tool LinGo [41]. The results revealed that LinGo could not find the optimal solution within a reasonable time frame for a high number of flows sent simultaneously. Therefore, it might be necessary to investigate more powerful optimization tools capable of finding solutions within a reasonable time.

The ILP formulations mentioned above aim to tackle challenges related to traffic-aware energy consumption. However, in many cases, the optimal solutions

obtained do not scale well with a large number of links, nodes, line cards, switches, or *flows* [25]. Several of these ILP formulations are specifically designed for predefined DCN topologies, such as fat-tree and Bcubic. In contrast, some are topology-independent. Given these considerations, we were motivated to explore a broad spectrum of state-of-the-art optimization tools and compare their experimental results in the [24].

Our further investigation in [24] demonstrated that solutions could be obtained within a better timeframe using more potent optimization tools. We considered running leading solvers such as GUROBI and SCIP, and we introduced the architecture of our NEO-DCN portfolio solver. Several experiments were conducted on lots of generated traffic in DCNs (such as near and long traffic patterns).

3.3 Problem Modeling

Power consumption of DCN routing mechanisms depends on the mode of operation of the set of switches that forward the flows between sources and destinations. Power consumption of DCN switches can be measured in two ways: *dynamic*, which measures the power consumption of active links, and *static*, which measures the sum of power consumed by components such as chassis, fans, and switching fabric. We adopt the following notation conventions, all the notations are summarized in Table 3.1. A directed weighted graph, $\mathbb{G} = (\mathbb{S}, \mathbb{E})$, models the network topology. The vertex-set is denoted by $\mathbb{S} = \{s_1, s_2, \ldots, s_n\}$, and the edge-set is denoted by $\mathbb{E} \subseteq \{e_{ij} \mid s_i, s_j \in \mathbb{S}\}$. The i-th switch is denoted as s_i and represents an OpenFlow switch. The primary function of each switch is to facilitate the routing of information through the path determined by the network controller. In the graph, $\mathbb{G}$, every edge represents a link, and the link connecting the i-th and j-th switches is identified by e_{ij}. Network links can exist in either an active (ON) or inactive (OFF) state. We use binary variables, denoted by L_{ij}, to indicate the current state of network links. The variable L_{ij} is 1 if the link connecting switches i and j is active. This means that it can transmit packets between two ports. Conversely, L_{ij} is 0 if the link is inactive. In practice, each link consists of two ports, a sending port and a receiving port. Therefore, when designing power-efficient routing the working ports of each link should be considered. Similarly, the variable, ℓ_i, is set to 1 if the switch is active and 0 if the switch is inactive. The Network Power Consumption (NPC)

is given in (3.1).

$$\text{NPC} = S_p \sum_{s_i \in \mathbb{S}} \ell_i + D_p \sum_{e_{ij} \in \mathbb{E}} L_{ij}. \tag{3.1}$$

Equation (3.1) relates the NPC to ℓ_i and L_{ij}, which denote the state of a corresponding switch, s_i, and link, e_{ij}, i.e., whether they are turned on or off. The variables D_p and S_p denote whether the power consumption is dynamic or static, respectively, they are represented as non-negative real numbers, including zero, $D_p, S_p \in \mathbb{R}_{\geq 0}$. In order to capture the flows traversing the topology, the variable $FR\,(f, i, j)$ takes the value 1, which means that the flow f traverses the edge e_{ij}. If e_{ij} is not traversed by the flow f, $FR(f, i, j)$ it set to 0. The link utilization U_{ij} represents the ratio of the size of the flows passing through the edge e_{ij} and the link bandwidth, BW_{ij}. When this ratio is scaled by 100, the utility of a link ranges from 0 to 100%. A link utilization matrix, $\mathbf{U}$, is constructed by considering the utilization of each edge, e_{ij}. The traffic of the DCN is represented by the set of flows $\mathbb{F}$, where each $f \in \mathbb{F}$ is defined as $f = (f.S_r, f.D_s, \lambda_f)$. A flow represents a group of packets which have the same source and destination address, that travel along the same route to reach their destination. The source and destination switches are denoted by $f.S_r$ and $f.D_s$, respectively. Finally, the packet rate of flow f is denoted λ_f and is measured in bits per second.

3.3.1 Optimization Model

In this model, we consider the links as the main energy-saving components in the network. To accommodate the provided traffic, the model uses the set of active links which has the smallest cardinality, $P \in \mathbb{P}$. The optimization model considers the following problem characteristics:

1. The optimization model's parameters refer to a snapshot of the network state for the sake of simplicity. This means that the model considers the network state at a specific moment of time.

2. The model starts with a standard multi-commodity flow problem. The constraints include flow conservation, link capacity, demand satisfaction, and the total number of active links.

3. Splitting a single flow into packets across multiple links in the topology could save energy by increasing overall link utilization. However,

Table 3.1: Conventional names of parameters used in problem formulation.

Parameters	**Definitions**
$\mathbb{S}$	Set of nodes (switches) in the DCN topology, $\mathbb{S} = \{S_1, \ldots, S_n\}$
$\mathbb{E}$	Set of edges where $e_{ij} \in \mathbb{E}$ represents the connection between two switches S_i and S_j.
$f.S_r$	Source switch.
$f.D_s$	Destination switch.
$\mathbb{F}$	Set of flows, where a flow $f = (f.S_r, f.D_s, \lambda_f) \in \mathbb{F}$ is represented by source $f.S_r \in \mathbb{S}$, destination $f.D_s \in \mathbb{S}$, and bit rate $\lambda_f \in \mathbb{N}$.
$\mathbb{P}$	Set of energy-efficient shortest paths, where a path, $P = \{s'_1, \ldots, s'_k\} \in \mathbb{P}$ is represented by a set of switches between the servers of DCNs, where $\forall s'_i \in \mathbb{S}$.
$\mathbf{U}$	Utilization matrix where U_{ij} represents the utilization of the link e_{ij}.
Constant	**Definitions**
BW_{ij}	Adjacency matrix scaled by the bandwidth of the edges $\mathbb{E}$ equal to 1 Gbps.
$T_{ij} \in \mathbb{N}$	Positive-valued variable representing the traffic volume over e_{ij} before including the new flow (previous load state).
Decision Variable	**Definitions**
$FR(f, i, j)$	$\begin{cases} 1, & \text{if flow } f \in \mathbf{F} \text{ passes through link } e_{ij} \in \mathbb{E} \\ 0, & \text{otherwise} \end{cases}$
L_{ij}	$\begin{cases} 1, & \text{if link } e_{ij} \in \mathbb{E} \text{ is active} \\ 0, & \text{otherwise} \end{cases}$

the reordered packets at the destination, due to varied path delays, can degrade the performance. As a result, we incorporate restrictions into our formulation based on the entire flow.

We assume that all the links have the same energy consumption at the beginning. The objective function is defined as

$$\text{minimize} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} L_{ij} \right). \tag{3.2}$$

The objective function in Equation (3.2) minimizes the number of active links, which reduces the energy consumption in turn. We set $L_{ij} = 0$, when there is no physically connection between S_i and S_j. When a physical link does exist, the value of the link is either 0 or 1 depending on the traffic demand. The model also takes into account the correlation between the links and traffic volume. The above objective function works against the following constrains:

1. **Links and Traffic Correlation Constraint:** This constraint considers the correlation between the traffic volume and the links. Therefore, the constraint defines the relationship between the traffic volume T_{ij} and the link state L_{ij} to increase the utility of the link as much as possible, as shown in (3.3).

$$\frac{T_{ij}}{BW_{ij}} \leq L_{ij}, \quad \forall e_{ij} \in \mathbb{E}, \tag{3.3}$$

   Expressing that the traffic volume must not exceed the bandwidth of a link.

2. **Links and Flows Correlation Constraint:** This constraint represents the correlation between links and flows, such that a link should be active if and only if a flow passes through it, as shown in (3.4).

$$FR\,(f,i,j) \leq L_{ij}, \quad \forall f \in \mathbb{F},\ \forall e_{ij} \in \mathbb{E}, \tag{3.4}$$

   Meaning that flows should pass only through active links.

3. **Utility Constraint:** This constraint computes the utility of all the topology's links, and limits the link utility to less than or equal to the link's bandwidth BW_{ij}, thus this constraint acts to limit the value of U_{ij} (the left-hand side of the constraint), by considering the active flows f_{ij} and their packet rate values λ_f, as shown in (3.5).

$$\sum_{f \in \mathbb{F}} FR\,(f,i,j)\ \cdot \lambda_f \leq BW_{ij} - \mathrm{T}_{ij}, \quad \forall e_{ij} \in \mathbb{E}, \tag{3.5}$$

   Where a flow's packet rate is counted according to the directed nature of the network graph.

4. **Path Conservation Constraint:** This constraint installs the path from the source $f.S$ to the destination $f.D$ for each flow f, as shown in (3.6).

$$\sum_{i=1}^{n} FR\,(f, f.S, i) = 1, \quad \sum_{i=1}^{n} FR\,(f, i, f.D) = 1, \quad \forall f \in \mathbb{F}. \tag{3.6}$$

5. **Flow Conservation Constraint:** This constraint guarantees for any flow f that the number of the incoming and outgoing flows of the intermediate switches between the source $f.S$ and the destination $f.D$ should

be equal, in order to avoid packet loss, as shown in (3.7).

$$\sum_{i=1}^{n} FR(f,i,j) = \sum_{i=1}^{n} FR(f,j,i), \quad \forall\ f \in \mathbb{F},\ \forall j \in \mathbb{S}, j \neq f.S, j \neq f.D. \tag{3.7}$$

6. **Network Loop Avoidance Constraint:** In order to enhance the computational performance for finding the minimum number of links between the source switch, $f.S_r$, and destination switch $f.S_r$, we introduce the following constraint. This constraint aims to prevent nodes from revisiting their immediate predecessors in the path, as shown in (3.8).

$$FR(f,i,j) + FR(f,j,i) \leq 1, \quad \forall f \in \mathbb{F},\ i,j \in \mathbb{S}. \tag{3.8}$$

3.4 Problem Solving

Initially, we implement our proposed model in LinGo, which is a high-level language for optimization modeling. LinGo provides a collection of built-in solutions to handle a wide range of optimization problems. Unlike many modeling products, all of the LinGo solvers are directly connected to the modeling environment. Instead of using slower intermediary files, this seamless connection enables LinGo to transmit the issue to the right solver immediately in memory. This direct connection also reduces the compatibility issues between the solver and the modeling language components. LinGo is supported by LINDO Systems Inc. [41]. It is free [ii] and available for students and interested researchers.

We run LinGo on Windows 11 Education N with an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz and 16.0 GB RAM. Our goal is to obtain the optimal solution, which provides appropriate link and flow assignments, so that the traffic demands are satisfied. The inputs to the model include: (1) the topology (fat-tree made up of $k = 4$, we presented in section 2.5.1), where disabled links have their status set to zero; (2) the capacity of the links; (3) the current state of the DCN traffic T_{ij}; and finally, (4) flows $f = (f.S_r, f.D_s, \lambda_f) \in \mathbb{F}$ which are described by their source, destination, and speed rate.

[ii] Researchers interested in using LinGo can send a request email to "sales@lindo.com" to obtain a free license for six months.

$$\begin{pmatrix} \mathbb{G} = (\mathbb{S}, \mathbb{E}) \\ (BW_{ij})_{n\times n}, \mathbb{F}, (T_{ij})_{n\times n} \end{pmatrix} \overset{LinGo}{\rightarrow} \begin{pmatrix} (L_{ij})_{n\times n}, (U_{ij})_n \\ \{FR(f,i,j) \mid f \in \mathbb{F}, e_{ij} \in \mathbb{E}\} \end{pmatrix} \quad (3.9)$$

The output of the optimizer is defined in (3.9). The output of the solver is the paths defined by the active links L_{ij} for each flow and the current link utilization U_{ij} which increases and decreases according to the traffic input matrix, T_{ij}.

Regular network optimization models determine the optimal (shortest, fastest, cheapest, etc.) route from the source to the destination at a given time. In contrast, our model can determine more paths, $\mathbb{P}$, at a time. In addition, it works like a "multi-layer" network optimization model, where the model finds not only one optimal route for one flow with a source and destination but for multiple input flows at a given time. It defines optimal paths set, $\mathbb{P}$, for all the input flows, as well as managing the current state of the network. It gets the current traffic load as an input in parallel with new flow demands, which are then considered when determining the optimal paths for the next round.

We injected different burst sizes and numbers of random flows into the DCN in order to evaluate our approach. The results in Table 3.2 show the global optimal solutions for each status. The number of flows appears in the second column. The size of the model (variables and constraints) increases in parallel with the number of flows. In this state, the model runtime is enormous when there are greater than ≈100 flows active at a time, because of the large number of integer variables.

3.5 LinGo Result Discussion

Although the ILP model presented above can calculate optimal multi-path routes and manage the current network status, obtaining the optimal solution using the optimization toolkit LinGo proved to be computationally expensive when dealing with large network sizes and a high number of flows. Specifically, accommodating only 120 bursts of flows in a fat-tree topology with $k = 4$ took more than 160 minutes.

Figure 3.1 along with Table 3.2 illustrates the relationship between the number of flows, the number of active links, and runtime. The left figure reveals that LinGo struggles to find a solution within a reasonable timeframe when handling

Table 3.2: LinGo test analysis.

Case	No. of flows	Objective value	Total variables	Integer variables	Total constraints	Memory (kB)	Runtime (s)
1	120	35	156912	155616	319249	51179	9891.79
2	110	32	142656	141360	290341	46542	275.24
3	100	32	130992	129696	266689	42747	219.28
4	90	33	118032	116736	240409	38531	452.84
5	80	30	105072	103776	214129	34315	45.53
6	70	31	90816	89520	185221	29678	49.00
7	60	29	77856	76560	158941	25463	12.38
8	50	29	66192	64896	135289	21247	16.89
9	40	28	51936	50640	106381	17031	4.37
10	30	28	38976	37680	80101	12858	3.22
11	20	24	26016	24720	53821	8599	2.22
12	10	19	14352	13056	30169	4805	1.34
13	5	17	7872	6576	17029	2697	0.79

more than 100 simultaneous flows, resulting in a dramatic increase in runtime. Conversely, the right figure demonstrates a proportional increase in the number of active links with the number of flows.

It is worth noting that, initially, when the number of flows is below 30, the optimizer tends to increase the number of links in an approximately linear correlation with the number of flows injected into the network. This observation is attributable to the limitation on the number of active links, which varies depending on the sources and destinations. As the number of active links increases, the optimizer begins to consolidate multiple flows and reduce the activation of new passive links.

These findings have motivated us to explore alternative ILP solver tools. Our objective is to enhance the current results by seeking more powerful optimization tools capable of efficiently finding optimal solutions for a higher number of flows within a reasonable timeframe, especially when applied to more realistic benchmarks.

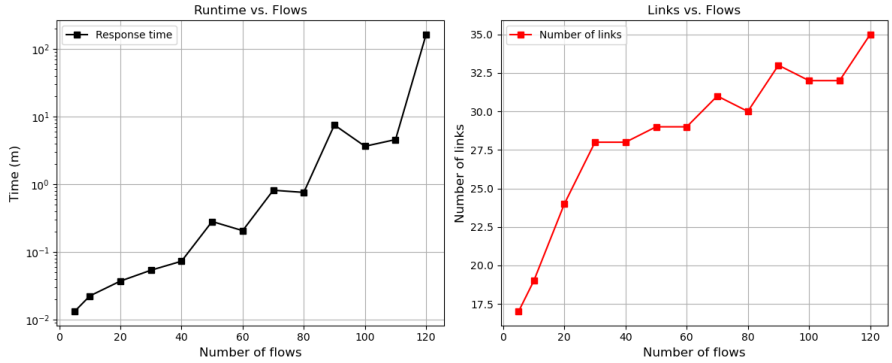


Figure 3.1: Correlation between the number of flows and the runtime of LinGo, and between the number of flows and the number of active links, respectively.

3.6 ILP Solving Tools

In this and the following sections we present and examine more leading problem-solving tools. Google's OR-TOOLS [53] is an open-source toolkit for solving optimization problems in general. Via the Python package `ortools`, one can access several optimization tools, including the following ones. GUROBI [54]

provides one of the most powerful commercial solvers for a wide range of optimization problems, including ILP problems, and it is free to use for academics and students. CP-SAT[iii] is a constraint programming solver that uses SAT methods, and it is part of the OR-TOOLS package. SCIP [55] is one of the fastest non-commercial solvers for MILP and, also, an open-source framework for integer programming. CBC (Coin-or Branch and Cut) is an open-source MILP solver [56]. We will run GUROBI, CP-SAT, SCIP, and CBC from Python code inside our portfolio solver NEO-DCN, as detailed in Section 3.7.

3.7 NEO-DCN Portfolio Solver

The proposed ILP model has been implemented in our tool NEO-DCN for the purpose of comparison, which is a variant of our open-source tool NEO in [57] that we adapted to our DCN model. NEO-DCN is publicly available at `https://github.com/kovasz/neO-DCN`.

NEO-DCN is a portfolio solver, meaning that it can execute different ILP solvers, which were mentioned in Section 3.6, in parallel. The parallel execution is implemented by instantiating `ProcessPool` from the `pathos.multiprocessing` [58] Python module, which can run jobs with a non-blocking and unordered map.

The OR-TOOLS package provides two solver interfaces that we can use for ILP solving: (1) the `MPSolver` interface for MILP solvers such as GUROBI, SCIP and CBC, and (2) the `CPSolver` interface for Google's Constraint Programming solver CP-SAT. Both interfaces allow adding ILP constraints in the form (3.10):

$$\text{solver.Add}\,(w_1 * x_1 + \cdots + w_n * x_n <= c) \tag{3.10}$$

where each w_i and c is an integer, and x_i a Boolean variable.

Note that although the interface allows to use relational operators other than $\leq$, NEO-DCN translates all constraints to "AtMost" constraints for normalization purposes.

For NEO-DCN, we introduced a JSON input format to read data about the configuration of the network, as well as the current configuration of flows. The benchmark files used in our experiments apply this input format as shown in Listing 3.1, and they can be found in the repository of NEO-DCN as well.

[iii] `https://developers.google.com/optimization/cp/cp_solver`

Listing 3.1: JSON format for the DCN model.

```
{
     "number of switches": ...,
     "links": [
         {
             "source": ...,
             "destination": ...
        }
         ...
     ],
     "flows": [
         {
             "source": ...,
             "destination": ...,
             "packet rate": ...
         }
         ...
     ]
 }
```

The integer properties `source` and `destination` represent the index of the corresponding switches, i.e., integers in $[1, n]$ where n is the number of switches. The `packet rate` property is also a positive integer.

3.8 Benchmarks

The communication patterns affect performance and power consumption [59]. Based on the fact that the traffic in a data center swings between peak traffic (e.g., at daytime) and low traffic (e.g., at nighttime) [6, 59], the traffic matrix for a DCN follows the sine wave in (3.11).

$$\text{Traffic Rate } = \frac{1}{2} \cdot \text{max rate} \cdot (1 + \sin(t)) \tag{3.11}$$

This thesis explores three types of the sine-wave traffic matrix: near, long, and mixed (i.e., random). The benchmarks build upon what we described in Section 3.3.1. Each benchmark is a snapshot of the DCN at the time interval t_i , $1 \leq i \leq m$. This means that each benchmark captures the state of the traffic matrix at a specific time.

Near traffic pattern: The traffic is restricted between the servers that reside in the same PODs (Point of Delivery) of the topology, i.e., the servers that are connected through the access layer switches only. The benchmarks aggregate the flows to a minimum number of links inside the same POD.

Long traffic pattern: The traffic is restricted between the servers that reside in different PODs of the topology, i.e., the servers that are connected through the edge, aggregation, and core layer switches. The model saves less power due to balancing the load between switches and using multiple paths to keep the QoS at an acceptable level, depending on the utility of links at that time.

Random traffic matrix: The traffic matrix for this pattern is a mixture of the above patterns, in order to explore how many links we can save with a random sine-wave pattern.

3.9 Experimental Results

In our experiments, we are dealing with the real-world DCN topology in Section 2.5.1 (the topology containing 20 switches and 16 hosts, and numerous links between those nodes). The bandwidth of each link is uniformly set to 1 Gbps.

The ILP solvers that we mentioned in Section 3.6 were run on the benchmark instances with a wall clock time limit of 1200 seconds. LinGo was run as a Windows desktop application for comparative purposes to assess its effectiveness on benchmarks other than the random one solved in Table 3.2, while the other ILP solvers as part of NEO-DCN on Linux. During the experiments, we measured the runtime of the solvers and, also, monitored the memory consumption of solvers by using the memory profiler `mprof`.

Since LinGo ran on Windoes, we considered the comparison inconclusive and unfair, leading us to exclude the amount of memory used by LinGo from the results.

3.9.1 Near Traffic Pattern

All mentioned solvers converge to the optimal solution for this traffic pattern very fast. They reduce the topology in Figure 2.1 to a minimum-tree DCN topology in Figure 3.2 by setting all unneeded links to off-state. Figure 3.2 shows the four scenarios of the near sine-wave pattern. In the first scenario, we burst roughly 1 Gbps distributed over 20 flows from servers A, B to C, D, i.e, directional traffic. The optimizer aggregates all flows in six links. On the other hand, in the second scenario, the number of flows are increased to 30 and the traffic to 1.3 Gbps. The topology changes because the model balances the traffic over the links, and the number of links becomes 8 instead of 6. In the third scenario, part of the traffic is bidirectional, while we keep the traffic volume at 1.3 Gbps. We burst the same number of flows, 30, distributed as follows: (1) 10 flows from servers A to C, (2) 10 flows from B to D, and (3) 10 flows in a reverse direction, from server C to A. The optimizers output the minimum number 12 by aggregating as many flows as possible in one path. In the fourth scenario, we keep all the characteristics of the third scenario, except that we increase the traffic volume to roughly 2 Gbps by sending new traffic from server D to B. As a result, the minimum number of active links becomes 14.

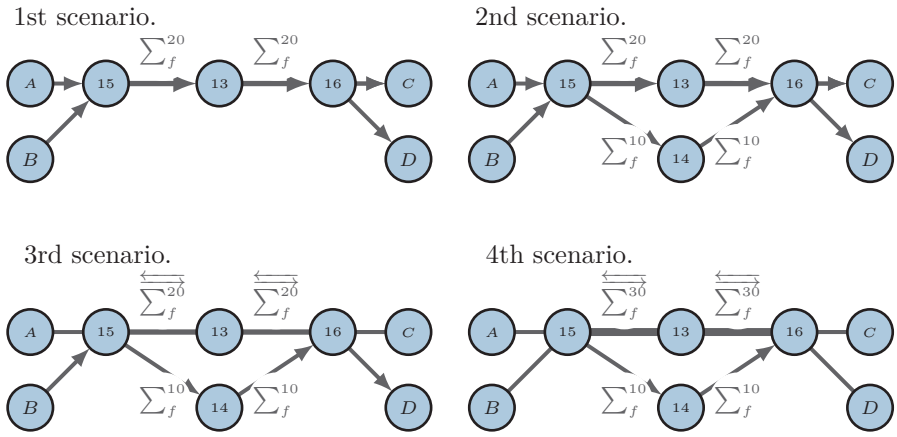


Figure 3.2: Four near-traffic scenarios, where directed links indicate flow direction. Undirected links represent bidirectional flows.

Table 3.3 shows some of the results reported by the ILP solvers, including the optimum value (i.e., minimum number of active links). In the table, the

runtime of each solver is given in seconds. The results show that the benchmark instances in all scenarios can be solved in reasonable time by any of the solvers. However, GUROBI and CP-SAT outperform the other solvers by almost 1 order of magnitude on this benchmark.

Table 3.3: Runtimes (s) of ILP solvers for near traffic pattern.

Scenario	**Flows**	**Opt.**	LinGo	GUROBI	CP-SAT	SCIP	CBC
1	20	6	2.11	0.44	0.42	0.63	0.53
2	30	8	4.21	0.53	0.53	0.73	3.03
3	30	12	3.21	0.53	0.52	0.93	0.62
4	40	14	5.01	0.73	0.72	1.33	3.43

Additionally, Table 3.3 shows fluctuations in runtimes for the CBC optimizer compared to other solvers like CP-SAT and Gurobi. These fluctuations may stem from differences in their optimization algorithms and implementation strategies. CBC utilizes branch-and-cut algorithms, whose performance can vary based on problem characteristics such as size and complexity. In contrast, CP-SAT and Gurobi employ different optimization techniques: constraint programming and mixed-integer programming, respectively. These solvers might be more appropriate for solving the proposed model, leading to more stable runtime behaviors.

3.9.2 Long Traffic Pattern

In this benchmark, we separate the servers from different PODs into two groups: sender and receiver. Besides that, we set the number of flows to a constant value of 24. Then, we gradually increase the traffic volume roughly from 1 Gbps to 5 Gpbs, which were captured in 14 benchmark instances, and we burst them into the DCN. The idea behind this benchmark is to demonstrate how the subset of active links changes according to the traffic demand when the number of flows is constant. Table 3.4 shows the traffic volume in Gbps for each benchmark instances, and the corresponding optimum values (i.e., minimum number of active links).

While 6 active links are sufficient to use for the initial time interval (representing low traffic), one needs to activate the 48 links in the topology for the last time interval (representing high traffic). In the table, one can definitely see higher runtimes than those in the near-traffic experiment. LinGo, SCIP, and CBC

even timed out (TO) for some of the benchmark instances, due to the higher level of difficulty of solving, caused by a high load of traffic.

The solvers' runtimes are visualized in Figure 3.3. Notice that the vertical axis, that represents the runtimes in seconds, is log-scaled. Up to the 7th time interval, when the traffic volume is 2.7 Gbps, all the solvers can find the optimum in a short time and, in particular, GUROBI and LinGo seem to provide a stable performance. For a higher volume of traffic, however, all the solvers loose efficiency very rapidly, except for CP-SAT, which keeps a surprisingly stable performance all the way to the very last time interval.

Table 3.4: Runtimes (s) of ILP solvers for long traffic pattern.

Time Interval	**Traffic**	**Opt.**	LinGo	GUROBI	CP-SAT	SCIP	CBC
1	0.91	6	2.81	0.53	0.42	0.62	0.63
2	0.92	12	2.74	0.43	0.43	0.62	0.63
3	1.5	18	2.93	0.53	1.93	0.62	1.73
4	2.15	26	2.96	0.53	28.88	1.32	3.13
5	2.3	28	2.97	0.53	22.57	25.28	19.56
6	2.4	30	2.85	0.53	17.45	54.63	5.53
7	2.7	32	2.73	0.53	12.25	14.76	11.35
8	3	36	27.73	1.74	16.76	146.02	101.21
9	3.4	36	10.83	2.44	11.35	29.31	80.16
10	3.6	40	597.18	141.96	17.69	122.61	406.6
11	3.9	40	566.44	162.26	13.05	951.43	TO
12	4.2	44	TO	425.5	36.6	291.98	TO
13	4.5	44	TO	144.51	28.28	TO	TO
14	4.8	48	TO	575.49	45.01	TO	TO

The same Figure 3.3 visualizes the memory consumption of the different ILP solvers. Recall that we could not apply memory profiling to LinGo. As the chart shows, all the solvers consume a moderate amount of memory for each benchmark instance. Most importantly, for CP-SAT, which was proved to be the fastest solver on this benchmark, memory consumption seems to be constant-like.

3.9.3 Random Traffic Matrix

In the benchmark with random traffic matrix, we inject different burst sizes and random flows into the DCN. The generated benchmark instances consist of

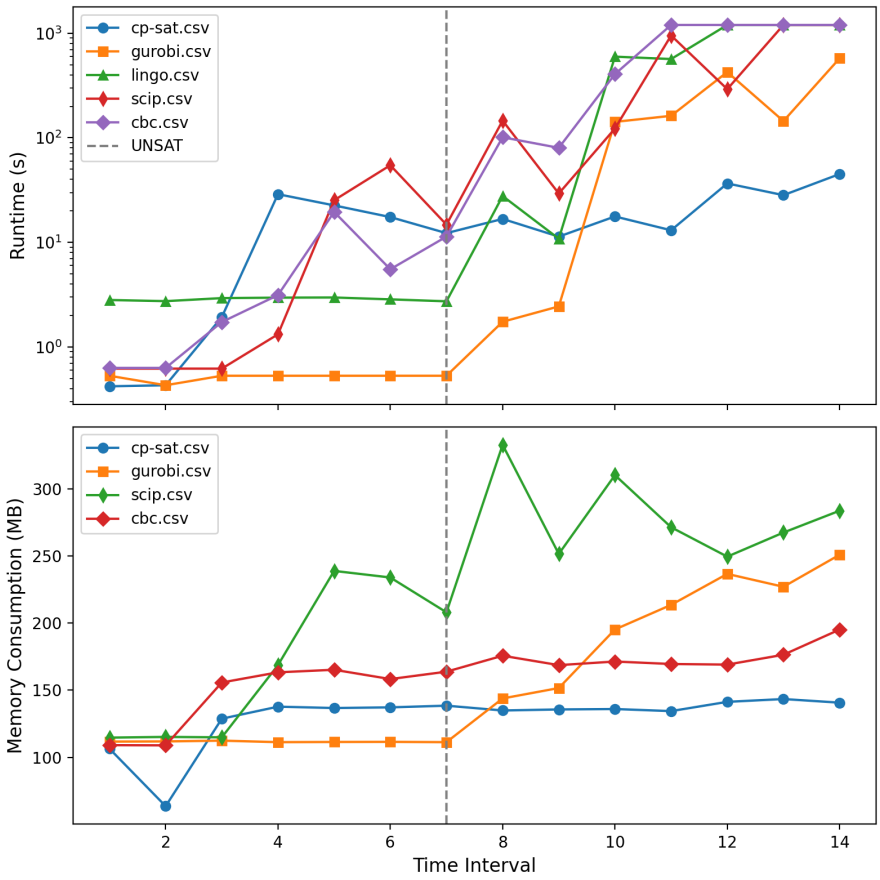


Figure 3.3: The upper plot shows the runtimes of ILP solvers for the long traffic pattern (note a value of 10^3, it means TO), while the lower plot displays the memory consumption of ILP solvers for the same pattern.

5, 10, 20, ..., 200 flows, respectively, where each flow f is given by a random source host $f.S$, a random destination hosts $f.D$ and a random packet rate λ_f.

Table 3.5 gives several details for each benchmark instance. The number of flows gradually increases from 5 to 200. Note that benchmark instances up to 110 flows are satisfiable (SAT), while the ones above that are unsatisfiable (UNSAT), meaning that there does not exist any solution for them. For the SAT instances, the table shows the optimum values (i.e., minimum number of active links).

Table 3.5: Runtimes (s) of ILP solvers for random traffic.

Flows	**Traffic**	**Result**	**Opt.**	LinGo	GUROBI	CP-SAT	SCIP	CBC
5	0.08	SAT	15	0.79	0.24	0.24	0.22	0.24
10	0.21	SAT	19	1.34	0.23	0.33	0.32	0.44
20	0.49	SAT	24	2.22	0.33	0.63	0.53	0.74
30	0.61	SAT	28	3.22	0.53	1.23	0.83	0.83
40	0.92	SAT	28	4.37	0.63	1.63	1.13	1.14
50	1.06	SAT	29	16.89	1.13	3.13	25.68	11.75
60	1.35	SAT	29	12.38	1.54	5.54	7.74	158.31
70	1.35	SAT	31	49	4.54	22.47	185.03	206.83
80	1.67	SAT	30	45.53	3.85	62.25	97.99	158.70
90	1.86	SAT	33	452.84	32.2	83.36	TO	TO
100	2.42	SAT	32	219.28	6.85	45.02	1160.16	329.72
110	2.25	SAT	33	275.24	34.92	TO	TO	481.98
120	2.71	UNSAT		TO	1.93	TO	7.44	42.51
130	8	UNSAT		TO	2.14	TO	8.25	41.3
140	7.71	UNSAT		TO	2.24	TO	10.25	69.56
150	7.84	UNSAT		TO	2.34	TO	9.15	78.88
160	8	UNSAT		TO	2.64	TO	7.44	63.04
170	8	UNSAT		TO	2.83	TO	10.55	116.14
180	7.81	UNSAT		TO	3.04	TO	9.35	136.88
190	7.99	UNSAT		TO	3.14	TO	16.16	87.49
200	8	UNSAT		TO	4.65	TO	13.66	115.13

Table 3.5 and Figure 3.4 show the solvers' runtimes for each benchmark instance. Notice that the vertical axis of the time chart is log-scaled.

Only GUROBI is able to solve all the benchmark instances under the time limit. LinGo and CP-SAT times out on all the UNSAT instances, while all the other solvers are able to recognize the UNSAT case in quite a reasonable timeframe. CBC and SCIP time out on 1 and 2 SAT instances, respectively, which consist of a high number of flows. This is due to differences in their optimization

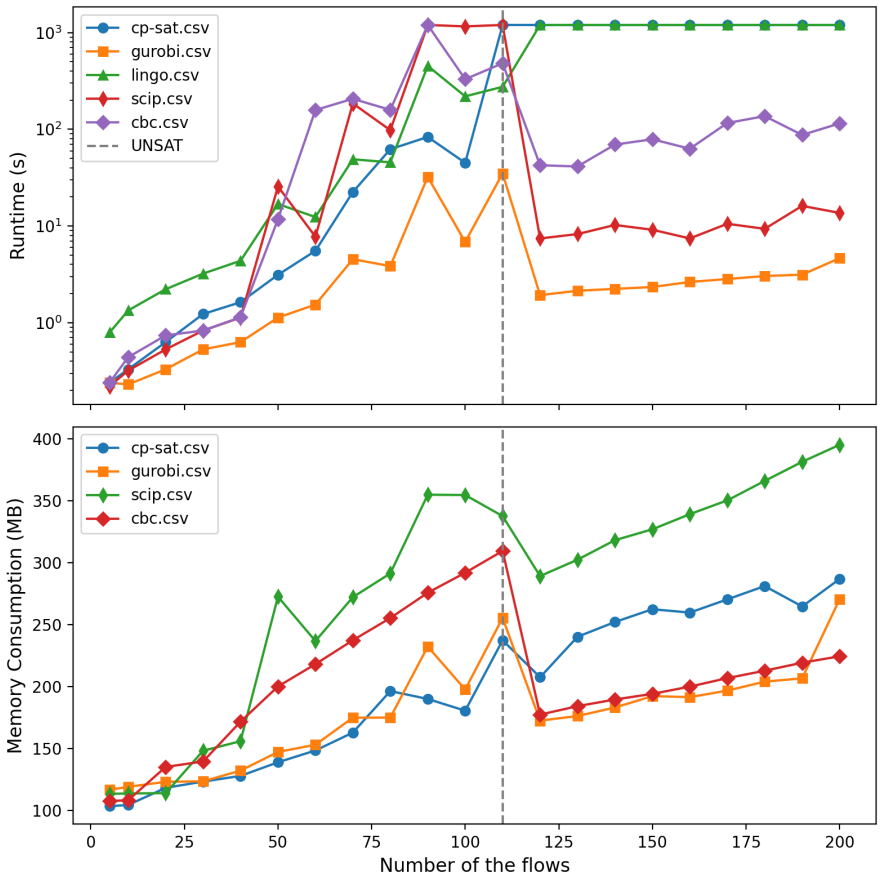


Figure 3.4: The upper plot shows the Runtimes of ILP solvers for random traffic, while the lower plot displays the Memory consumption of ILP solvers for the same pattern.

algorithms and implementation strategies. Therefore, some instances present more challenges for the solvers than others, based on the optimization algorithm techniques employed by each solver.

Regarding runtime, GUROBI outperforms all the other solvers by 1 or 2 orders of magnitude, especially when comparing to LinGo that was used as an underlying solver in section 3.4, which presented in [25].

The memory consumption we have recorded is shown in Figure 3.4. GUROBI, as the fastest solvers for the current benchmark, consumes a moderate amount of memory.

3.10 Result Discussion

The comparative experiments have been conducted on a wide range of traffic benchmarks for three different communication patterns: (1) the near traffic pattern results show that GUROBI and CP-SAT outperform the other solvers regarding runtime for most traffic instances; (2) the long traffic pattern results show that above a traffic volume of 2.7 Gbps all the solvers dramatically loose efficiency, except for CP-SAT, which keeps a good performance and roughly constant memory consumption; (3) the random traffic results show that, for most of the traffic instances, GUROBI outperforms the other solvers regarding both runtime and memory. We can conclude that, for most of the benchmark instances, most of the solvers outperform LinGo regarding runtime. Consequently, it was definitely worth experimenting with those solvers, with GUROBI and CP-SAT in particular, as part of our new contribution in [24].

3.11 Conclusion

In conclusion, the proposed model effectively selects the active subset of the topology to minimize power consumption. However, the translation of this idea into real-world implementation represents the next milestone in our thesis, aiming to make our concept applicable in practical scenarios. To address this challenge, we introduce a heuristic algorithm in the following chapters with the same goal of minimizing the number of active links in DCNs, thereby enhancing efficiency in real-time scenarios. This algorithm is designed to overcome challenges encountered by the ILP model, particularly in scenarios

with multiple and complex traffic flows, where high runtimes were observed. This is achieved through the greedy selection of the optimal path at specific moments in time.

3.12 Key Findings

Question: How can mathematical optimization approaches be employed to improve power usage in DCNs?

Answer: The chapter demonstrates that formal methods can be employed to minimize the active subset of the topology, including ports, links, and switches, to meet the current traffic demand. This involves turning off the remaining idle devices in DCNs, leading to improved power usage [24]. However, the next crucial step is to transition this idea into real-world implementation.

Next Step: The upcoming chapter will illustrate how the integration of SDN and DCNs in one paradigm results in an effective real-time approach to optimizing network resources.

4

SD-DCN Paradigm and Simulation Tools

Contents

4.1 Introduction

After our gentle introduction in section 1.2, in this chapter, we delve deeper into the structure of the SDN paradigm to outline its advantages in optimizing power consumption in SD-DCN structure. We also introduce our testbed, which will be used in the subsequent chapters, to avoid any repetition.

Therefore, this chapter involves the following clarifications: (1) illustrating the layers of SD-DCN structure, (2) explaining OpenFlow theory, which covers the components and basic functions of the OpenFlow switch, and the OpenFlow switch protocol used to manage an OpenFlow switch from a remote OpenFlow controller, and (3) detailing the construction of our testbed, which we will use to test our algorithms in the following chapters.

4.2 SD-DCN Paradigm

SDN addresses the inherent challenges posed by the static architecture of traditional networks, which are decentralized and complex. In contrast, contemporary networks demand greater flexibility and simplified troubleshooting. SDN proposes the centralization of network intelligence in one component by dissociating the forwarding process of network packets (data plane) from the routing process (control plane). The control plane, often comprising one or more controllers, is considered to be the brain of the SDN network, housing comprehensive intelligence models.

This separation of the control and data (Forwarding) planes is a key aspect. The control plane is relocated to a centralized location and operates on a general-purpose computer. This transformation enables direct programmability of network control and abstracts the underlying infrastructure for applications and network services. The OpenFlow protocol serves as a foundational element in constructing SDN solutions, thanks to its role in interpreting the rule-routing policy between the SDN controllers and OpenFlow switches.

Hence, the integration of SDN and DCN into the concept of SD-DCN technology represents an approach to cloud computing that emphasizes streamlining network management and facilitating programmatically efficient network configuration. This strategy is designed to enhance network performance, monitoring, and power consumption within the context of DCN, as will be illustrated

in the current thesis. The structure of SDN is depicted in Figure 4.1. The Figure illustrates the two bounds of the SDN controller, and they are (1) the Northbound Application Programming Interface (API) provides a set of APIs used to manage the network. It allows communication between the SDN controller and applications. The Northbound API enables higher-level applications to interact with the SDN controller, allowing them to request and configure network resources based on statistical information and other network-related data, and (2) the Southbound API (OpenFlow Protocol in our case) allows the SDN controller to instruct these devices on how to forward network traffic based on the control decisions made at the controller level. More detailed insights into these interfaces will be explored in the following sections.

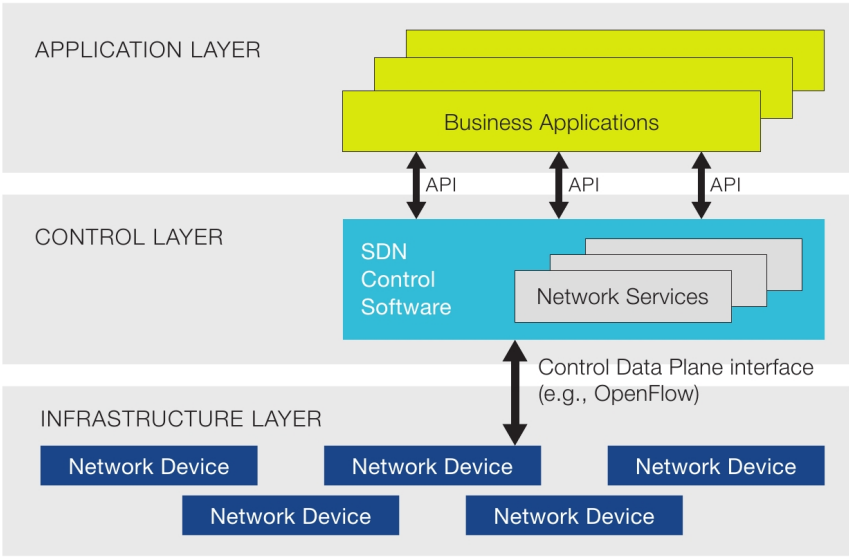


Figure 4.1: Software-Defined Network Architecture. This architecture is retrieved from [60].

4.2.1 Network Infrastructure

The infrastructure layer comprises OpenFlow switches and servers that construct the topology of the DCN, featuring 4 PODs, 16 servers, and 20 switches, as described in Section 2.5.1. In the remainder of this section, we will delve further into OpenFlow theory to acquire a deeper understanding of how the

OpenFlow switch operates in forwarding packets within the switches of the DCN's topology.

Figure 4.2 illustrates the general structure of an OpenFlow Switch, which includes one or more secure channels connecting to one or more SDN controllers. Additionally, it incorporates one or more flow tables that the switch uses to look up and forward incoming packets by specifying the appropriate output ports [61].

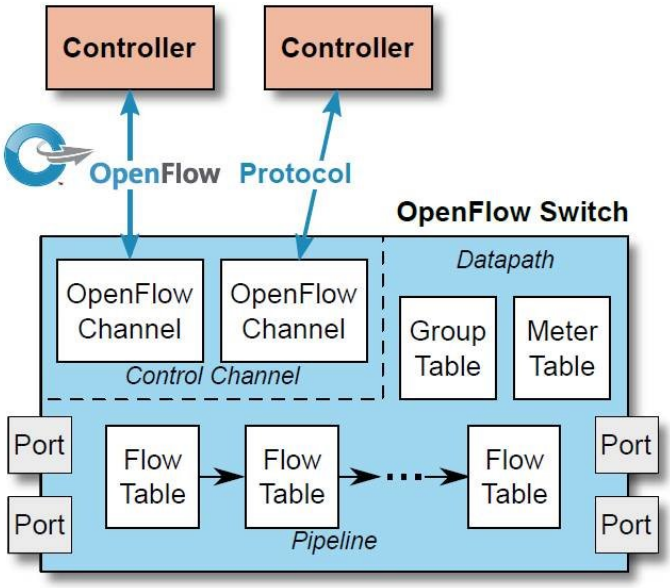


Figure 4.2: OpenFlow switch structure. This architecture is retrieved from [61].

Each flow table in an OpenFlow switch comprises a set of flow entries. The controller directs and manages these flow entries by deleting, updating, adding, and installing new ones using the secure OpenFlow protocol channel connection. Each flow entry consists of match fields, a set of instructions (actions), and counters, as shown in Figure 4.3.

Matching process in OpenFlow SDN involves comparing incoming packets against rules in the flow table of an OpenFlow-enabled switch. The matching process initiates at the first flow table and may extend to additional flow tables in the pipeline. Flow entries match packets in priority order, with the first matching entry in each table taking precedence. The matching process is done based on part or all of the flow entry match fields: MAC address source and destination, IP address source and destination, TCP and UDP port source and

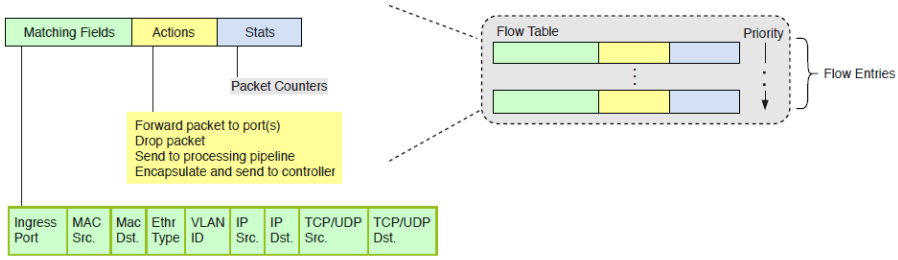


Figure 4.3: OpenFlow flow entry structuree. This architecture is retrieved from [62].

destination, and so on. If a matching entry is found, the instructions (actions) associated with the specific flow entry are executed, such as sending the packet to the appropriate output port [19].

There are two approaches of installing rules (flow entries): reactive and proactive. In the reactive approach, when no match is found in a flow table, the OpenFlow switch sends a "Packet-In[i]" message to the controllers over the OpenFlow channel. The controller responds with a "Modify Flow Table[ii]" message to install a new rule for forwarding packets. In contrast, in the proactive approach, the controller sends the rules directly as pre-defined, constant rules [63].

Moreover, various OpenFlow messages are exchanged between the controller and the OpenFlow switch, including Feature Request, Packet Out, and OpenFlow Statistics Request types (flow stats, port stats, table stats, aggregate flow stats, etc.). In the upcoming chapters, we will delve into the details of some of these messages, utilizing them to develop new models for routing and monitoring [64].

[i]The ofp_packet_in event is fired when the controller receives an OpenFlow packet-in message (`OFPT_PACKET_IN / ofp_packet_in`) from a switch, indicating that a packet arriving at a switch port has either failed to match all entries in the table, or the matching entry included an action specifying sending the packet to the controller. In addition to the usual OpenFlow event attributes, this event includes the port number on which the packet arrived and the parsed raw packet data [63].

[ii]The ofp_flow_mod event is fired by the controller to install a new flow entry in the OpenFlow Switch. These messages include a wealth of information, encompassing matching and action attributes of the flow entry [63].

Note: In the next chapter, this thesis specifically adopts the reactive routing algorithm to minimize the active topology subset and enhance power usage efficiency. Consequently, we rely on "Packet-in" and "Modify Flow Table" messages to manage the routing technique. Additionally, we employ port state messages to build a robust monitoring system.

4.2.2 Control Layer

The design and development of the OpenFlow protocol have been accomplished with the introduction of a new controller platform system. Various types of SDN controllers have emerged and evolved over time. We summarize them in Table 4.1. The controller acts as a middleware between the Apps representing the optimization algorithms for managing and monitoring the dynamic behavior of the network, and customizing work policies across the forwarding devices.

Table 4.1: Various SDN controllers have been developed by different companies.

Controllers	Developers	Languages	Open-Source
Ryu [65]	Nippon	Python	Yes
OpenDaylight [66]	Linux Foundation	Java	Yes
Beacon [67]	Stanford Uni.	Java	Yes
Trema [68]	NEC	Ruby/C	Yes
Floodlight [69]	Big switch	Java	Yes
POX [63]	Nicira	Python	Yes
Maestro [70]	Rice Uni.	Java	Yes
NOX [71]	Nicira	C++	Yes

In the current thesis, we employ the SDN controller to implement management policies in response to network activity. Specifically, we use the POX/RYU as the SDN controller. The choice of the POX/RYU controller is motivated by its suitability for quickly developing solution prototypes [72] and the ability to use Python as a northbound interface to manage the underlying topology, which falls within the scope of our experiences. The standard OpenFlow protocol [19] is employed as a southbound interface to define communication between the POX controller and the infrastructure's elements.

4.2.3 Application Layer

The application layer operates atop the SDN Controller, overseeing the management of the infrastructure layer. Utilizing the northbound interface, the Apps request and gather network statistics to effectively manage and manipulate the provided services. The output of these Apps is translated into instructions, which are then sent to manage the infrastructure layer (forwarding elements) through the southbound protocol. As outlined by [73], these Apps can be categorized into five types: (1) Monitoring, (2) Traffic engineering, (3) Mobility and wireless, (4) Security, and (5) Business applications. All are designed to run large data centers. Therefore, this environment serves as a hub for research, innovations, new ideas, and the development of new services, all implemented in software on the SDN controller.

At this point in the thesis, we introduce standard models that are available to researchers to use, which will be utilized in the subsequent chapters as detailed below.

4.2.3.1 Topology Discovery and Parser model

This module is responsible for detecting the operational elements of the infrastructure layer, switches, and links. Topology discovery is crucial for the network controller as it enables the construction of a comprehensive network view. We utilized the standard POX discovery module. To represent network information as a *graph* for easy management and manipulation, the NetworkX tool [74] was employed.

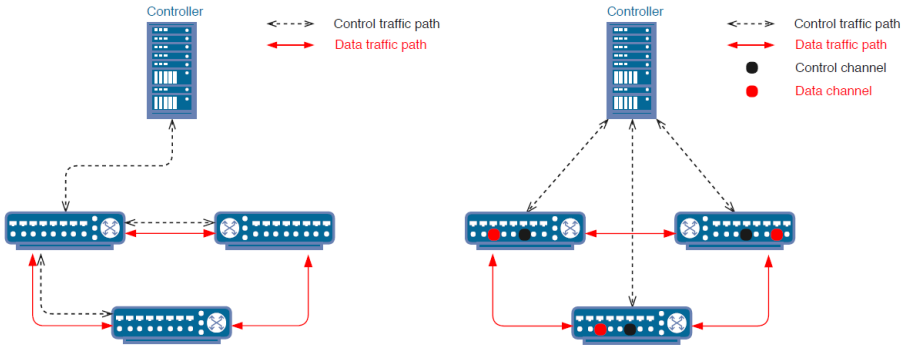
4.2.3.2 Keepalive model

This component prompts POX to send periodic echo requests to connected switches, addressing two key issues. Firstly, certain switches, including the reference switch, may assume that an idle control connection indicates a loss of connectivity to the controller and may disconnect after a period of silence. This behavior is likely flawed, and the component helps maintain the connection. Secondly, in the event of a loss of network connectivity to the switch, sending

echo requests and tracking their responses provides a bound on the time it takes to notice the loss [63].

4.3 Mode of Communication

In this section, we present two types of communication between the data plane devices and the SDN controller to highlight which one is better to deploy in this thesis.



(a) In-band control (b) Out-of-band control

Figure 4.4: Types of connections between OpenFlow switches and the control [75].

4.3.1 In-Band Mode

In this mode, both the traffic of the data plane and the control plane share the same connection or medium, as depicted in Figure 4.4. Consequently, this communication mode does not require any dedicated control port channel. While it offers cost advantages, it introduces a potential drawback:

(1) The control plane and data plane compete for the same resources. As noted by Sharma [76], this competition may adversely affect the performance of the DCN since both planes share bandwidth resources, leading to additional delays.

(2) Furthermore, there are security implications to consider. In this configuration, any attack exposes both traffic and control data, posing a risk of discovery, as highlighted by Braun [77].

Consequently, this mode is not widely adopted in SDNs [78], especially in SD-DCNs, which manage and store much more secure and critical information.

4.3.2 Out-of-Band Mode

This mode separates the traffic of the data plane and the control plane, as depicted in Figure 4.4. In this configuration, the control plane traffic moves through a new secure channel between the OpenFlow switch and the SDN controller. Although this approach increases costs due to the need for dedicated physical equipment, such as a specific port channel and link, it enhances both network delay and security concerns [76]. According to the literature, despite the increased cost compared to the In-Band mode, the Out-of-Band mode is more widely used to ensure high performance [79, 80]. Therefore, in the current thesis, we will adopt the Out-of-Band mode.

4.4 Computer Simulation

To implement our proposed algorithms in the next chapters, computational software systems are employed. The simulation employs suitable methods to examine and model the validity of our algorithms. Currently, network simulations serve as essential tools for IT consultants, researchers, and developers to verify network performance and behavior against new ideas, solutions, algorithms, protocols, and service applications. These simulations help demonstrate the performance by modeling the interaction of network equipment, such as links and switches.

In a study [81], the necessary steps to conduct a successful simulation are outlined. Figure 4.5 illustrates these steps, providing a flowchart to ensure correct simulation procedures. Currently, there are both commercial and open-source tools available for evaluating proposed solutions in SDNs, categorized into two main types, as we will discuss in the next subsections.

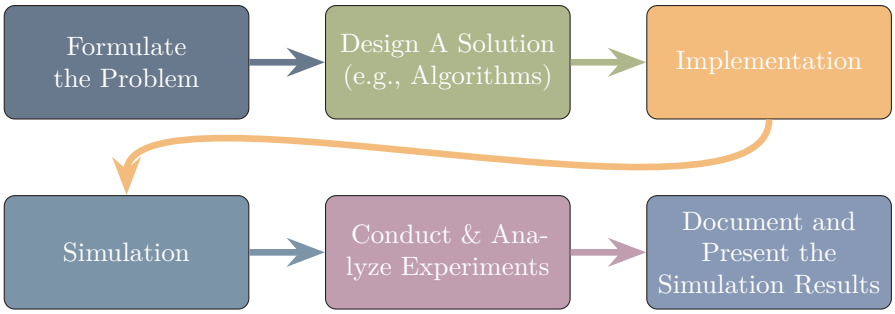


Figure 4.5: Idea realization via simulation

4.4.1 SDN Simulation-Based Testing

The primary goal of any simulation is to analyze the outcomes of algorithms or ideas. SDN is seen as a promising technology for network management now and in the near future. This has led to the emergence of various types of network simulations. The OpenFlow protocol, a crucial component of SDN, enjoys widespread support from different simulators such as Network Simulation-3 (NS-3) [82], FS-SDN [83], and OMNET++ [84], alongside other simulation tools specifically crafted for prototyping and evaluating novel concepts.

Despite the widespread use of these simulations, certain limitations have been identified, particularly regarding diversity and scalability. These issues have been discussed in earlier research [85]. In recent years, concerted efforts have been made to address and mitigate these problems, as explored in [86].

4.4.2 SDN Emulation-Based Testing

The alternative approach for analyzing networks with new algorithms and ideas involves the use of emulation tools. The key distinction here lies in their ability to offer a close approximation of real-world experiments while striving for a balance between the expensive costs associated with physical equipment and the limitations of simulation tools.

There are both proprietary and open-source emulation tools. An example of a proprietary emulation tool is EstiNet [87], which supports both emulation

and simulation environments. However, it is not open source, not free, and presents challenges in terms of adding features and modifying functions. On the other hand, a notable example of a free and open-source emulation tool is the Mininet emulator [88]. Therefore, we employed it in the validation process of the proposed framework in the next chapters.

Mininet is versatile, supporting both real and synthetic network topologies and enabling the execution of real kernel and application code on a single machine.

Mininet is widely adopted in the research community for various reasons, including:

1. Support for custom and real topologies with easy connection and implementation.
2. Compatibility with the OpenFlow protocol as a southbound API.
3. A flexible control plane that seamlessly connects with various types of controllers, including POX, using the default IP (127.0.0.0) and port (6653).

After closely examining the structure of SDN and various computer simulation methods, we are now ready to introduce our testbed, which will be the focus of the next chapter. This marks the final step in the current chapter of our thesis

4.5 Proposed Testbed

Figure 4.6 illustrates our proposed testbed, which will be utilized in the following chapters. Presenting it here is intended to preempt any potential redundancy in the subsequent chapters.

The figure delineates the two interfaces of the SDN controller:

1. The Northbound API (Python) provides a set of APIs used to manage the network. One application we will leverage is Topology Discovery and Parser, which we presented in Section 4.2.3.1.
2. In the control layer, we will deploy either the POX or RYU controller based on the needs of the algorithm.

3. The Southbound API (OpenFlow Protocol in our case) enables the SDN controller to instruct DCN's devices on how to forward network traffic based on control decisions, optimizing power usage. The topology of DCNs follows a fat-tree architecture, as previously presented in Section 2.5.1. Each host is connected to an OpenFlow Switch. Given the focus of the proposed system on out-of-band configuration, each switch is linked to the controller through a secure OpenFlow channel.

4. Lastly, the Mininet emulation connects to the SDN controller using the southbound API channel. Mininet is used to emulate the Fat-tree topology and to control the traffic injected by D-ITG at each server node. Furthermore, it facilitates the emulation of OpenFlow switches, allowing for testing and validation of the SDN controller's behavior under various scenarios

In order to generate traffic that is similar to the traffic delivered on DCNs, we used the Distributed Internet Traffic Generator (D-ITG) [89]. D-ITG is a tool that can be used for producing a realistic packet-based network traffic by accurately emulating the workload of real world traffic [90].

D-ITG is flexible enough to generate traffic patterns according to these distributions at both macroscopic and microscopic scales. Compared to other traffic generation tools, D-ITG is considered to be more reliable [89]. A range of types of traffic can be generated by D-ITG, for example, TCP, UDP, DNS and VoIP traffic. The topology and traffic manager will be deployed inside the Mininet emulator.

4.6 Conclusion

As we conclude this chapter, the significance of the SD-DCN paradigm in achieving Network Resource Optimization (NRO) and dynamically optimizing the network based on load and state becomes evident. This paradigm is particularly crucial for various applications, as it leverages near-real-time information about traffic, topology, and equipment. All the mentioned features establish the SD-DCN paradigm as the most effective way to transition our theoretical framework introduced in Chapter 3 into real-world implementation, a topic further explored in the subsequent chapters. Furthermore, we have established our testbed, ready for conducting experiments in the following sections.

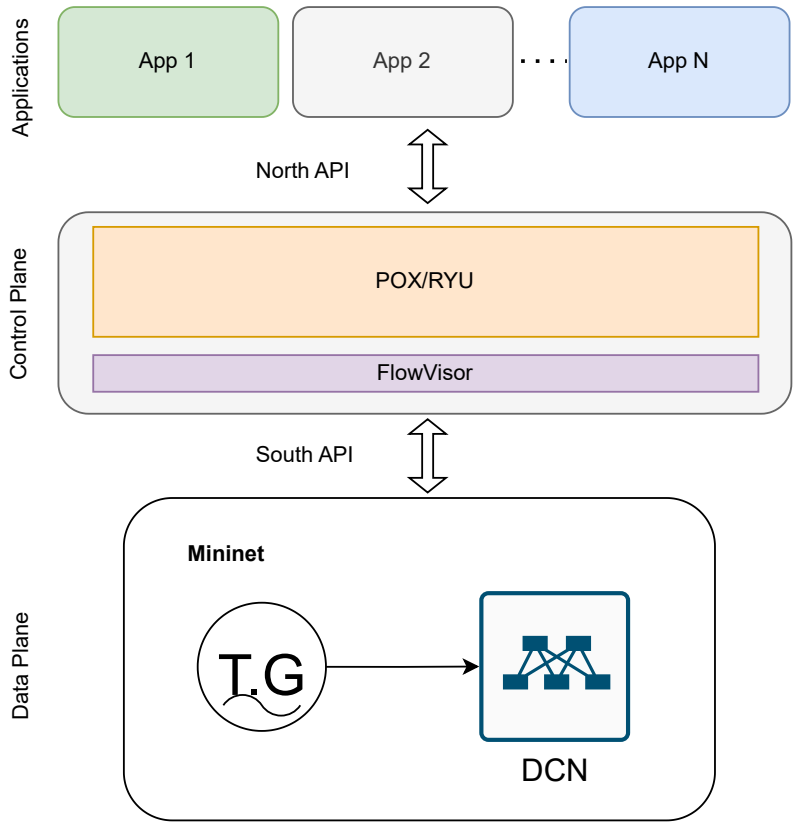


Figure 4.6: SD-DCN testbed structure

4.7 Key Findings

Question: Why is the combination of Software-Defined Networking and Data Center Networks (SD-DCNs) considered a cornerstone for optimizing DCN resources?

Answer: The chapter emphasizes the strength of the SD-DCNs combination by consolidating the network state into a central point. This enables researchers to develop a robust monitoring model and to implement real-time resource optimization, achieving efficient power usage in DCNs [10].

Next Step: The next chapter demonstrates how to develop a heuristic link utility algorithm based on a real-time monitoring system to achieve optimized resource usage.

5

Consolidation Algorithm

Contents

5.1 Introduction

Building upon the findings in Chapter 3, the real-time implementation and emulation of our idea have become crucial for its practical application in an industrial context. In this chapter, we introduce a novel framework designed to minimize power consumption in software-defined DCNs. Central to this framework is the Fill Preferred Link First (FPLF) algorithm, a heuristic link utility-based approach that seeks a balance between energy efficiency and performance.

FPLF is specifically designed to maintain QoS and optimize the energy consumption of DCNs. This is accomplished through continuous monitoring of DCN traffic conditions, utilizing the OpenFlow protocol [19] to acquire real-time topology state and data traffic information. The algorithm selects the most energy-efficient path below a pre-defined threshold, ensuring effective flow quality and power-aware routing.

We demonstrate the effectiveness of FPLF in maintaining satisfactory QoS while reducing power consumption in DCNs, as evidenced by the count of dropped packets, and latency. Therefore, we evaluate our proposed framework using an experimental platform comprising the POX controller and the Mininet network emulator introduced in Chapter 4.

Finally, we benchmark our proposed algorithm against the state-of-the-art approach, the Equal Cost Multi-Path (ECMP) algorithm, to provide a comprehensive comparison and assessment.

It is worth noting that a substantial portion of the content in this chapter draws upon the insights presented in the work of [25].

5.2 Motivated Example

In this section, our focus is on addressing the issue of wasteful energy consumption in scenarios with low link utilization levels. Our proposed approach is designed to dynamically adapt to varying traffic demands, resulting in a gradual increase in the number of active links as demand intensifies. Conversely, when traffic demand is low, we employ a contrasting strategy. The core mission

of the FPLF algorithm is to optimize the utilization of the set of active links and their capacity.

To illustrate our approach in more detail, refer to Figure 5.1. We construct a topology as an example to analyze the performance of the two forwarding methods proposed below. Assuming the links have the same bandwidth, denoted as BW. The source and destination hosts for data flows f_1 and f_2 are C, H, and D, G, respectively. The volume of traffic injected by f_1 and f_2 is denoted as V_1 and V_2, respectively. The flows are assumed to be sent into the DCN simultaneously at $t = 0$.

When flows arrive at 16, the FPLF algorithm checks the link utilization matrix, filling the active links to 90% of their maximum utility. This precaution optimizes performance between the source and destination and prevents links from becoming overloaded and more susceptible to unplanned failures [36], ensuring their long-term use at full capacity. If this threshold is exceeded, new paths are opened. The result is the retention of as many links as possible in the off state. From a routing perspective, the path $16 \rightarrow 13 \rightarrow 2 \rightarrow 9 \rightarrow 12$ is shared between both f_1 and f_2 only if the utility of the links is below the threshold value in the link utilization matrix, $\mathbf{U}$, as shown in Figure 5.1 with blue and red colors.

At $t = 5$, an additional flow is injected into the network. Host A sends flow f_3 to destination H. Assuming the total traffic volume of the flows exceeds the 90% target utilization (i.e., $\sum_{i=1}^{n} V_i > 0.9$), when f_3 arrives at 15[i], FPLF checks the link utilization matrix again and promptly opens a new path $15 \rightarrow 13 \rightarrow 1 \rightarrow 9 \rightarrow 11 \rightarrow 19 \rightarrow 12$ in response. Note that this new path has more hops.

In summary, FPLF categorizes all the links in the first path as *overloaded*, prompting the opening of a new path. The links in this new path are classified as *underutilized*, while all other links are labeled as *idle/unused* links.

5.3 Proposed Method

Similar to the proposed ILP Objective Function discussed in Section 3.3.1, our FPLF algorithm aims to minimize the energy consumption of the DCN. This objective is achieved by maximizing the link utilization, as expressed

[i]OpenFlow switch sends a Packet-In message to the SDN-Controller, where the FPLF algorithm is allocated. The message contains all the flow details.

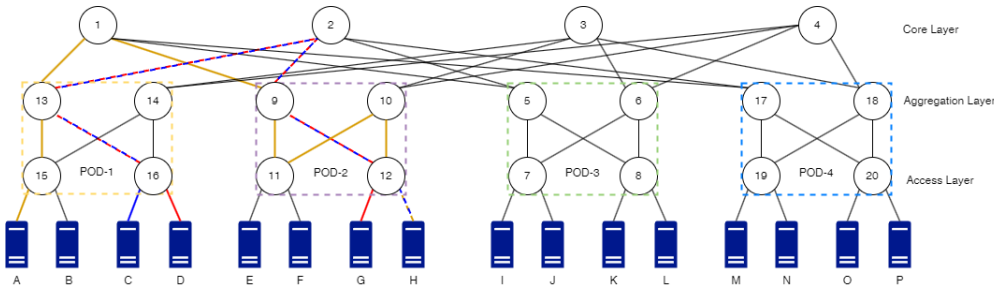


Figure 5.1: Motivational example illustrating the behavior of the FPLF algorithm. In the diagram, , and represent the consolidation of two flows. Meanwhile, , , and represent individual flows in the links.

Table 5.1: Conventional names of parameters used in problem formulation.

Parameters	Definitions
$\mathbb{S}$	Set of nodes (switches) in the DCN topology, $\mathbb{S} = \{S_1, \ldots, S_n\}$.
$\mathbb{E}$	Set of edges where $e_{ij} \in \mathbb{E}$ represents the connection between two switches S_i and S_j.
$f.S_r$	Source switch.
$f.D_s$	Destination switch.
f	A flow $f = (f.S_r, f.D_s, \lambda_f) \in \mathbb{F}$ represented by source, destination and packet rate.
$\mathbb{P}$	Set of energy-efficient shortest paths, where a path, $P = \{s'_1, \ldots, s'_k\} \in \mathbb{P}$ is represented by a set of switches between the servers of DCNs, where $\forall s'_i \in \mathbb{S}$.
$\mathbf{U}$	Utilization matrix where U_{ij} represents the utilization of the link e_{ij}.
$\mathbf{C}$	Link-cost matrix where C_{ij} denotes the cost of the link e_{ij}.
$\mathbf{A}$	Aggregated traffic matrix.
$\mathbf{T}$	Traffic matrix where T_{ij} denotes input traffic of e_{ij}.
BW_{ij}	Adjacency matrix scaled by the bandwidth of the edges $\mathbb{E}$.
k	Number of the pods in the fat-tree topology.
T_{s_i}	Cumulative traffic values that cross S_i during a specific duration of time.
L_{ij}	$\begin{cases} 1, & \text{if link } e_{ij} \in \mathbb{E} \text{ is active} \\ 0, & \text{otherwise} \end{cases}$

in (5.1), which is designed to aggregate traffic as efficiently as possible across a minimum number of links. All the notation used in this chapter is summarized in Table 5.1.

$$\max \left(\frac{\sum_{e_{ij} \in \mathbb{E}} T_{ij}}{BW} \right), \quad \forall e_{ij}. \tag{5.1}$$

The FPLF algorithm ensures that all switches utilize the specified link as long as its bandwidth is underutilized. However, if the link exceeds the threshold value of 90% of the bandwidth, the FPLF algorithm redirects flows to the shortest alternative path using a Dijkstra algorithm.

Before the FPLF algorithm starts, it is essential to have comprehensive statistics on traffic within the DCN. To address this requirement, we developed a robust monitoring model.

5.3.1 Monitoring Model

To ensure the effective functioning of FPLF, we have developed a monitoring model as a prerequisite. This model is designed to collect real-time statistics, specifically the traffic from each node, utilizing the OpenFlow protocol 1.0. The OpenFlow protocol facilitates the exchange of extensive statistics, such as flow, port, and group table statistics, through messages between the SDN controller and OpenFlow switches. These statistics provide the controller with a comprehensive global view of the DCN, as discussed in Section 4.2.1.

In our scenario, where each link consists of two ports, we collect port statistics by periodically sending OpenFlow "port stats request" messages to all switches. Upon receiving the response messages, the monitoring model extracts the bytes and packets cumulative values.

All counters in OpenFlow switches, including Port Counters, Flow Counters, Group Counters, and others, typically maintain cumulative values. These values represent the total count or amount of a specific metric since the counter was last reset or initialized. Consequently, the collection matrix at any given point in time serves as an aggregated traffic matrix. In our case, the aggregated traffic matrix, **A**, is generated during the monitoring phase. Figure 5.2 illustrates how this matrix is generated.

At time t, the controller issues a polling statistics message, and the aggregated traffic represented by $\mathbf{A}(t)$ is temporarily stored until the subsequent time frame $t+1$, when a new traffic aggregated traffic matrix, $\mathbf{A}(t+1)$, is constructed. The controller then accurately calculates the traffic matrix, $\mathbf{T}$, for a specific time frame, as shown in (5.2):

$$\mathbf{T} = \frac{\mathbf{A}(t) - \mathbf{A}(t-1)}{\Delta t} \tag{5.2}$$

Where Δt is the polling period, which is always one second in our case. Hence, we use $\mathbf{T}$ to calculate the link utilization matrix, $\mathbf{U}(t)$, for each period.

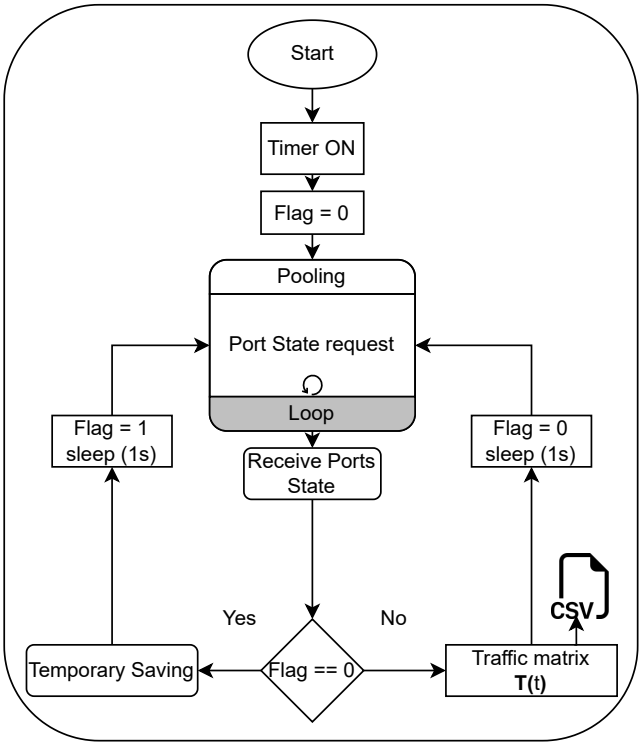


Figure 5.2: Procedure of Traffic Matrix generation.

5.3.2 FPLF-Adaptive Algorithm Components

FPLF incorporates three components. We begin by describing the Link-Utility (LU) calculation component. This component is responsible for calculating the utilization matrix **U**, which stores the utility, U_{ij}, for all the links. We continue by describing the Link-Cost (LC) component, which returns the link-cost matrix **C**. This matrix includes the costs C_{ij} of all the links. Finally, the Fill-Shortest Path (FSP) component forces the forwarding elements to utilize the specified links as long as the bandwidth BW_{ij} bound is not exceeded. The pseudo code of LU component is outlined in Algorithm 1. The input parameters are the flow demand set, $\mathbb{F}$, and the underlay network edge, $\mathbb{E}$, i.e., a Fat-Tree in our case. When a traffic flow $f = (f.S_r, f.D_s, \lambda_f)$ passes through a link, e_{ij}, its corresponding utility U_{ij} is increased by $\frac{\lambda_f}{BW_{ij}}$.

Algorithm 1: Link-Utility.

Input : edge set $\mathbb{E}$, traffic demand $\mathbb{F}$
Output : utilization matrix **U** of all links

```
1 U = 0
2 foreach edge e_ij ∈ E do
3 |   foreach (f.S_r, f.D_s, λ_f) ∈ F passing through e_ij do
4 |   |   U_ij ← U_ij + λ_f / BW_ij
5 |   end
6 end
```

The LU component returns the utilization matrix **U** which describes the links as being overloaded, idle, or under-utilized. The pseudo code of LC is illustrated in Algorithm 2. This algorithm computes the cost value for each link e_{ij}. The input parameters are the utilization matrix **U** and the graph $\mathbb{G}$. According to the utility values U_{ij}, the LC algorithm classifies the links to be either overloaded, idle or under-utilized and it sets a different cost C_{ij} for each link as a result of this classification.

The cost values that are generated by this component are used as an input to the FSP component. The pseudo code for FSP is illustrated in Algorithm 3.

The FSP handles each new Packet-In event (new requests) by identifying the energy-saving path. The input parameters are the edge set, $\mathbb{E}$, link-cost matrix, **C**, and the incoming flows, $(f.S_r, f.D_s, \lambda_f)$.

Algorithm 2: Link-Cost.

Input : utilization matrix **U**, edge set $\mathbb{E}$
Output : link-cost matrix **C**

$\forall\ C_{ij} \leftarrow cost.default()$ ▷ **idle** U_{ij}
foreach *edge* $e_{ij} \in \mathbb{G}$ **do**
  if $U_{ij} > BW^{th}$ **then**
    $C_{ij} \leftarrow cost.high()$ ▷ **overload** e_{ij}
  end
  else
    $C_{ij} \leftarrow cost.low(U_{ij})$ ▷ **underutilization** e_{ij}
  end
end

Algorithm 3: Fill-Shortest Path.

Input : edge set $\mathbb{E}$, flow $(f.S_r, f.D_s, , \lambda_f)$, link-cost matrix **C**
Output : energy-saving path ESP

foreach *new request* **do**
  Run Dijkstra's algorithm [91] to find the shortest path with minimum link costs (C_{ij})
end

The FPLF algorithm fills the preferred links by selecting paths with under-utilization links. As the traffic volume increases, under-utilized links gradually become overloaded links, and the FPLF algorithm compels the FPS component to select idle links to meet the traffic demand. Thus, the FPS component finds the most energy-saving path for the flow, depending on the cost values, (overload, idle, and under-utilization) of the links that are obtained from the LC algorithm. It is worth mentioning that if all links become loaded, the FPLF algorithm shares new traffic demand as overload traffic between the links.

In terms of time complexity, the FPLF algorithm has $O\left(|\mathbb{S}|^2\right)$ complexity as it depends on Dijkstra algorithm to pick paths based on the minimum number of hops.

As a conclusion, the FPLF algorithm utilizes four attraction components, as depicted in Figure 5.3, to generate a network topology subset that satisfies the current traffic demand. Simultaneously, it places all other nodes in the DCN in an OFF state for power saving.

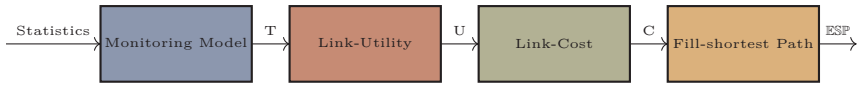


Figure 5.3: The FPLF algorithm's components are interconnected, with the output of each component serving as the input to the subsequent one. The culmination of this process yields the Energy-Efficient Shortest Path as the final output.

5.4 Proposed Framework

The experimental platform is constructed using the Mininet network emulator [88] and the POX controller. The entire emulation process was executed on a single machine (Ubuntu 20.04.2 LTS 64-bit) equipped with a 12-core Intel(R) Core(TM) i7-9750H CPU running at 2.6 GHz and 16 GB of DDR4 RAM. The network topology is imported from the Fast Network Simulation Setup (FNSS) [92], and the FPLF algorithm is implemented using the POX controller as POX modules. To generate network traffic from servers within a DCN, the D-ITG [89] platform is employed. The overall architecture is illustrated in Figure 5.4.

To demonstrate the adaptive primary function of the FPLF algorithm in proportion to traffic demand, we conducted experiments with various scenarios of traffic demand.

5.5 Emulation Setup and Implementation

We delve into exploring the potential energy savings of the FPLF algorithm under both low traffic demand, represented by multiple small-size flows, and high traffic demand, characterized by multiple large-size flows. Our experiments utilize a fat-tree topology with $k = 4$, as illustrated in Figure 5.1.

For our experiments, we deployed a D-ITG transmitter on the terminal of each sender server (A-H) and a D-ITG receiver on the terminal of servers (M-P). Essentially, D-ITG was employed to inject UDP traffic patterns: low traffic and high traffic. The emulation time was set to 2 minutes for the low traffic case and 8 minutes for high traffic. The packet size used for both patterns was 512 bytes. The flow rate for low traffic was a constant 10 packets per second, while high traffic ranged from 10 to 400 packets per second.

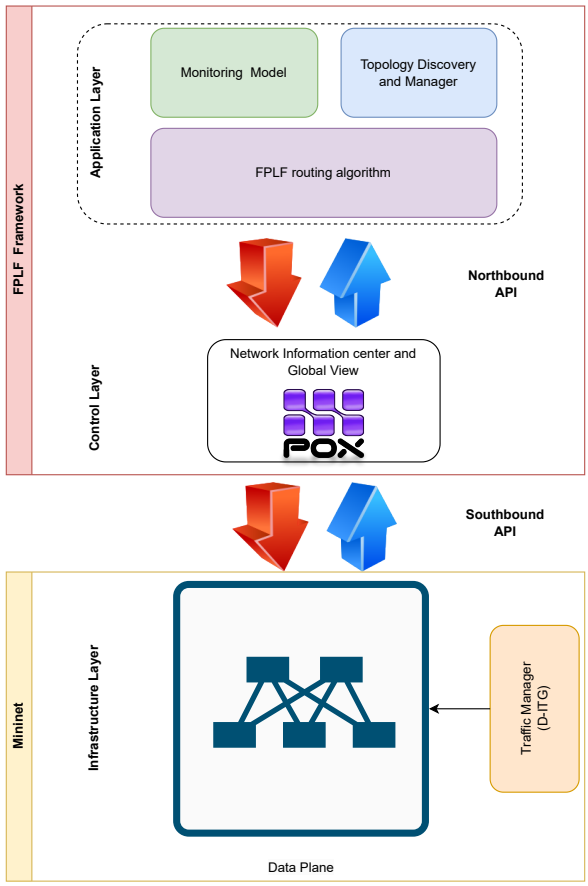


Figure 5.4: Architecture of the proposed framework and its components: the primary contribution is the FPLF algorithm and its traffic-aware optimizer blocks. OpenFlow was utilized on the southbound interface, and POX Python APIs were employed on the northbound interface.

5.5.1 Low Traffic Scenario

We systematically injected the DCN with multiple small flows and recorded the actions of the FPLF algorithm at each step. The start and end times of flows, along with the corresponding FPLF algorithm actions, are summarized in Table 5.2. This table illustrates the events during the first 33 seconds of the low traffic scenario, showcasing the responses of the FPLF algorithm to each burst of flows. The subsequent 87 seconds are not shown due to the absence of events, except for the gradual decrease in the utility of the download link (D-L (2,17)) until the end of the simulation.

The results highlight that the FPLF algorithm optimally directed all traffic to one core switch in the best-case scenario, occurring when the link utility of the paths was below the threshold value, indicative of low traffic demand. This optimization leads to significant energy savings as the FPLF algorithm minimizes the number of utilized core switches to the minimum number of links, achieving an idle:active ratio of 1:$\mathbb{E}_{core}$ (set of core edges). Figures 5.5a and 5.6a illustrate the power-saving paths and the time-varying utilization, respectively, which consistently remains below the threshold value.

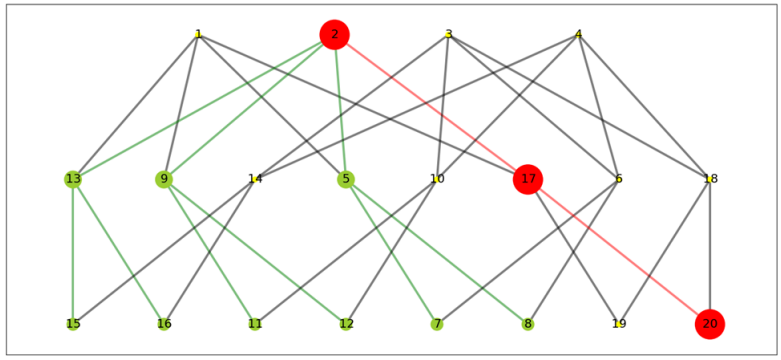
5.5.2 High Traffic Scenario

Similar to the first scenario, we compiled the characteristics of flows and all the scenario events in Tables 5.3 and 5.4, respectively. These tables provide an overview of flow patterns, durations, and total events in the high traffic scenario, along with the responses of the FPLF algorithm to each burst of flows.

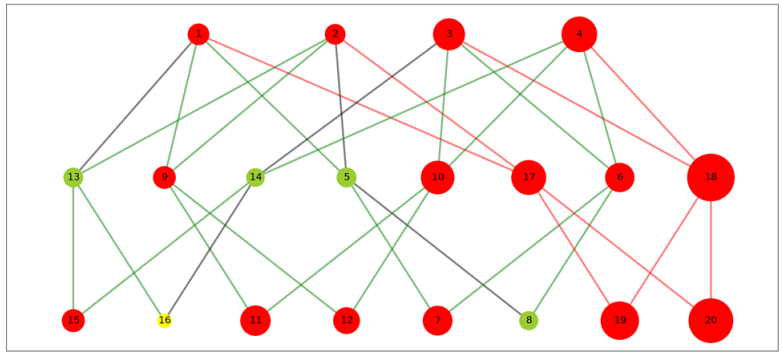
In the high traffic experimental scenario, we gradually increased the traffic until the link utilization value exceeded the threshold. The FPLF algorithm responded to this change by balancing the traffic among core switches to meet the traffic demands. This scenario can be considered as the worst-case scenario due to lower energy savings. Figures 5.5b and 5.6b illustrate the load balancing paths and the link utility of core switches, respectively, which consistently exceeded the threshold value.

Table 5.2: Low traffic scenario events that were generated and had server P as the traffic's destination.

Source	Number of Flows	Start Time (s)	End Time (s)	D-L Utility (2,17)	FPLF-Action	Number of the Core Switches
A, B	Single/Multiple small	0	100	increased from 0 to 1.7	Install-ESP	1
C, D	Single/Multiple small	3	105	increased from 1.7 to 0.22	Install-ESP	1
E, F	Single/Multiple small	10	115	increased from 0.22 to 0.25	Install-ESP	1
G, H	Single/Multiple small	15	120	increased from 0.25 to 0.37	Install-ESP	1
I, J	Single/Multiple small	20	120	increased from 0.37 to 0.51	Install-ESP	1
K, L	Single/Multiple small	33	120	fluctuated between 0.51 and 0.47	Install-ESP	1

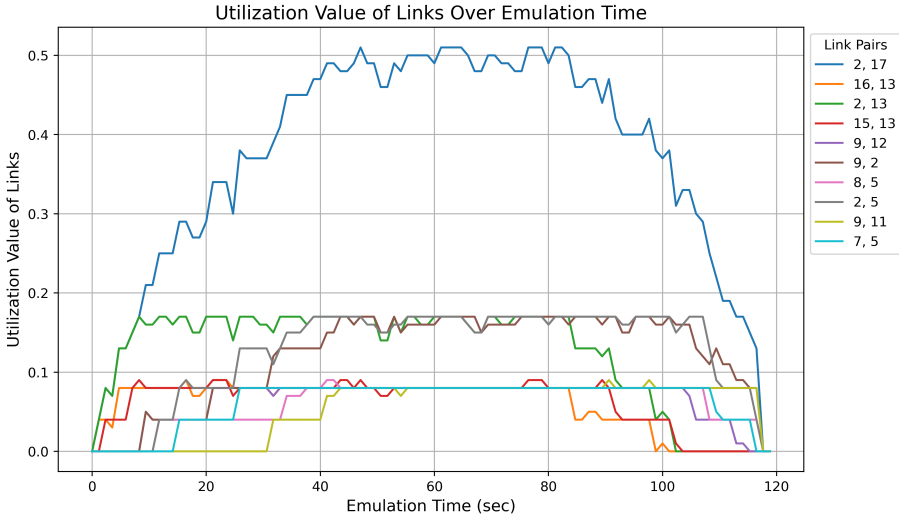


(a) Low traffic load

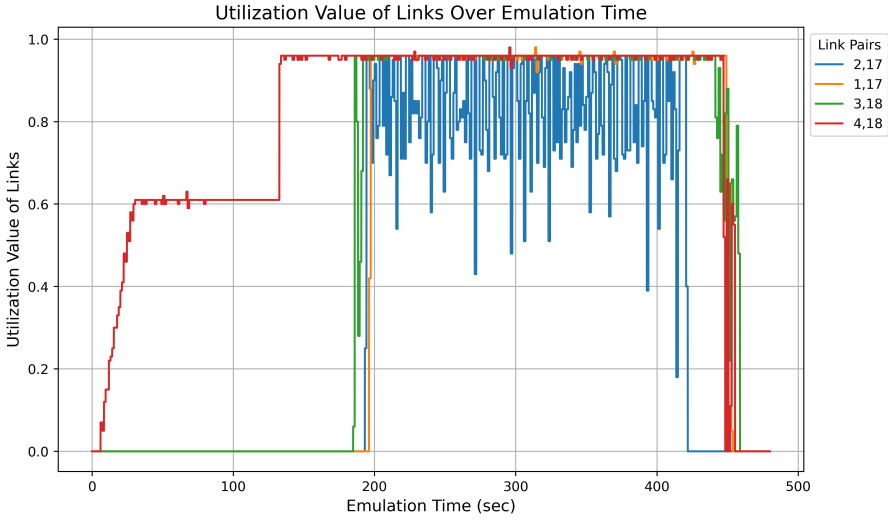


(b) High traffic load.

Figure 5.5: Forward (–), downward (–), and unused (–) paths are illustrated along with the cumulative traffic which is being served by DCN nodes: • > 10 Mb, • 500 kb–10 Mb, • < 500 kb.



(a) Low traffic utility.



(b) High traffic utility.

Figure 5.6: The link utilization is illustrated for both scenarios. The utility is less than the threshold for low traffic and it exceeds the threshold for high traffic scenarios.

Figure 5.6b clarifies that the FPLF algorithm distributes the traffic according to the number of available core switches. Members of the download links set {(1, 17), (2, 17), (3, 18), (4, 18)} all exceeded the threshold value, except for the link (2,17), whose link utilization value fluctuated (illustrated in blue).

In summary, as the demand continued to increase, the FPLF algorithm filled all available links, and power savings became minimal because every link in the network was utilized.

5.6 Result Discussion

This section delves into the traffic distribution dynamics facilitated by the FPLF algorithm in both scenarios. The cumulative traffic values presented in Figure 5.5a shed light on how switches are utilized throughout the emulation, contingent upon link utility.

In the first scenario, sub-path nodes $2 \rightarrow 17 \rightarrow 20$ accumulate substantial values, courtesy of the energy-efficient path identified by the FPLF algorithm. This underscores FPLF's ability in selecting a singular download path capable of managing all injected flows. Moving on to the second scenario, as traffic steadily escalates, Figure 5.5b depicts varying cumulative traffic values for core switches $\{T_{s_2} < T_{s_1} < T_{s_3} < T_{s_4}\}$, aligned with the link utilization values illustrated in Figure 5.6b. Notably, core switch number 4 and the download switches (i.e., switches along the downward path) register heightened cumulative values. This outcome results from the FPLF algorithm's initial choice of them as a single energy-saving path in the first 3 minutes of the simulation. Subsequently, the algorithm progressively opens new paths to accommodate the increased injection traffic, averting any potential network congestion events.

Finally, it is worth noting that in our experiment, the allocation of link resources, such as bandwidth, is unfair because the UDP protocol is a best-effort protocol. However, in the case of TCP traffic, a fair sharing scheme based on TCP's fairness algorithm ensures that if k TCP sessions (k flows) are opened, they must share the same bottleneck link with a bandwidth of R, with each flow expected to have an average rate of R/k. Consequently, we can conclude that applying FPLF to consolidate different flows into one link aims at optimizing utility and conserving power, which may lead to congestion. With TCP traffic, fairness among flows is guaranteed, unlike UDP traffic, which operates on a best-effort principle.

Table 5.3: High traffic flow details.

Source	Destination	Flows Description
A,B,C	M	Between 0 to 11 s, started with multiple small flows, after 140 s from the time of simulation, burst with large flow, i.e., high traffic
D,E,F	N	Between 0 to 11 s, started with multiple small flows, after 190 s from the time of simulation, burst with large flow, i.e., high traffic
G,H,I	O	Between 9 to 30 s, started with multiple small flows, after 190 s from the time of simulation, burst with large flow, i.e., high traffic
G,K,L	P	Between 21 to 28 s, started with multiple small flows, after 190 s from the time of simulation, burst with large flow, i.e., high traffic

Table 5.4: High traffic event details.

Events Time (s)	D-L Utility (1,18)	D-L Utility (3,18)	D-L Utility (2,17)	D-L Utility (4,18)	FPLF-Action	Number of the Core Switches
0 TO 136	0	0	0	0 increased to 0.6	Install-ESP	1
136 TO 190	0	0	0	0.6 increased to 0.96	Install-ESP	1
190 TO 200	0	0.3 increased to 0.96	0	0.96	Install-ESP	2
200 TO 202	0	0.96	0 increased to 0.96	0.96	Install-ESP	3
202 TO 205	0 increasing to 0.96	0.96	0.96	0.96	Install-ESP	4
205 TO 440	0.1.8 fluctuated between 0.96	0.96	0.96	0.96	No action	4
440 TO 466	0	0.96	0.96	0.96	No action	3
466 TO 470	0	0.96 decreased to 0.57	0.96	0.96	No action	3
470 TO 486	0	0.57 decreased to 0	0.96 decreased to 0	0.96 decreased to 0	No action	0

5.7 Performance Evaluation

To assess the performance of the proposed method, we conducted a comparison with the ECMP[ii] approach, as described in [93]. Two key metrics were considered: (1) power consumption rate and (2) number of utilized links.

The FPLF algorithm demonstrated superior performance over ECMP in both low and high traffic scenarios. In low traffic cases, as illustrated in Figure 5.7, the FPLF algorithm reduced the number of links by implementing energy-saving paths within the DCN switches. The number of utilized links and switches is expected to increase proportionally with growing traffic to meet QoS targets. This is evident in Figure 5.8, showcasing the FPLF algorithm's low latency with a sudden spike in traffic compared to the ECMP algorithm, effectively accommodating the traffic surge.

While the FPLF algorithm returns 16 paths with 27 links, ECMP, in comparison, returns 16 paths with 30 links. This observation suggests that the FPLF algorithm is more traffic-aware. As there are four downward links from core switches (as shown in Figure 5.5), bursty high traffic from all other servers fills these four links. Adding any new upload link becomes redundant, leading the algorithm to settle on 27 links.

Minimizing the number of links reduces the power used by turning off the ports and switches it connects. The percentage of saved links is computed according to (5.3) and (5.4).

$$\text{Active Links} = \sum_{e_{ij} \in \mathbb{E}} L_{ij}, \tag{5.3}$$

$$\text{Links saved} = \left(1 - \frac{\text{Active Links}}{|\mathbb{E}|}\right) \cdot 100\%. \tag{5.4}$$

According to the above metrics, the FPLF algorithm exhibits the best energy saving in all traffic volume scenarios. The least of the best energy saving is 15.625% in high traffic and 65.625% in low traffic, while the state-of-the-art algorithm recorded a 6.25% energy saving in both scenarios. Moreover, the energy-saving ratio in (5.5) between the power consumption of both ECMP

[ii] Equal-cost multi-path routing (ECMP) is a routing strategy where packet forwarding to a single destination can occur over multiple best paths with equal routing priority.

and FPLF in terms of active links under the same traffic conditions is 10% in high traffic and 63.3% in low traffic.

$$\text{Energy-saving ratio} = \left[1 - \frac{\text{FPLF Active Links}}{\text{ECMP Active Links}}\right] \cdot 100\%. \tag{5.5}$$

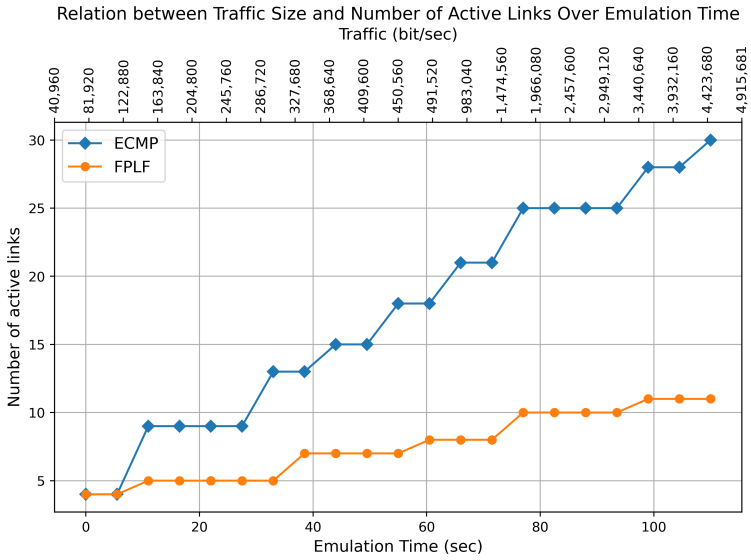


Figure 5.7: DCN link usage under low traffic load scenarios.

5.7.1 QoS Criteria

This study considered two critical QoS criteria: (1) the algorithm response time, i.e., latency and (2) the number of dropped packets.

Figure 5.8 illustrates that the FPLF algorithm exhibits a rapid response (i.e., low latency) to unexpected changes in traffic over time. This behavior can be attributed to the robust monitoring system, reflecting the instantaneous network state, which enables the FPLF algorithm to take prompt actions that align with the traffic demand at specific moments. This was particularly evident when the traffic reached its maximum value of 20 Mbps. The FPLF algorithm stabilized with a fixed number of links (27) at emulation time 130 s, continuing until the end of the experiment. In contrast, ECMP stabilized only after 190 s,

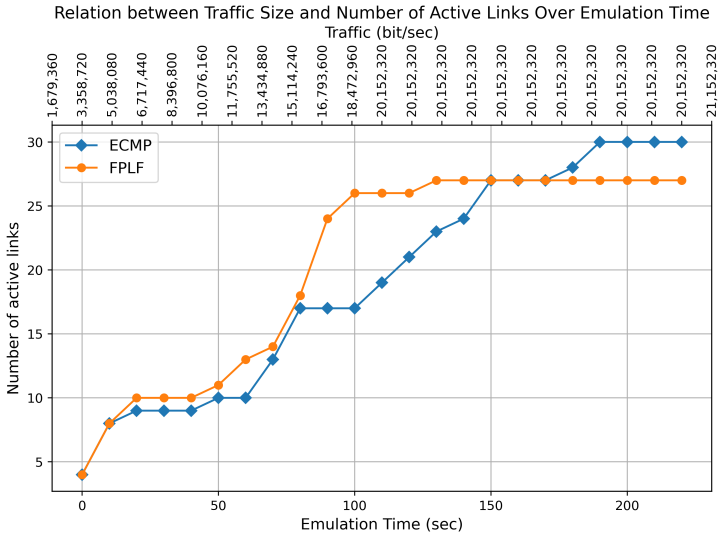


Figure 5.8: DCN link usage under high traffic load scenarios.

signifying a delay of 70 s before achieving the same QoS as the FPLF algorithm. The same observation was recorded in the low-traffic scenario, as shown in Figure 5.7. The FPLF algorithm stabilized with a fixed number of links (11) at emulation time 100 s, while ECMP continued to increase until the end of the simulation.

On the other hand, because FPLF addresses power consumption by consolidating flows in a single link, this may affect the rate of packet arrivals at the destination. Therefore, a crucial metric describing the FPLF performance in *high traffic* scenarios is the number of dropped packets.

During the power consumption level testing, DITG decoder logs were used to record the results in Tables 5.5.

Overall, it can be concluded from the above tables that FPLF has a lower percentage of dropped packets compared to ECMP, indicating that the threshold value helps mitigate congestion events.

Table 5.5: Number of dropped packets for the ECMP and FPLF algorithms.

Destination Server	Algorithm	Total Time (S)	Total Received Packets	Packets Dropped	Packets Dropped %
M	ECMP	429	58,614	128	0.22%
	FPLF	414	58,360	152	0.26%
N	ECMP	326.6	53,329	157	0.29%
	FPLF	322	56,705	117	0.21%
O	ECMP	335.6	53,413	206	0.38%
	FPLF	330	59,671	115	0.19%
P	ECMP	439.7	59,053	415	0.70%
	FPLF	423	62,706	114	0.18%

5.7.2 Evaluation Conclusion

Based on the results reported so far, the following conclusions can be drawn. The FPLF algorithm is suitable for deployment in DCNs that serve various traffic patterns, accommodating both low and high traffic loads. For instance, during periods of low traffic, such as night times, the FPLF algorithm demonstrates the potential to save up to 65.625% of the total energy consumed, leading to reduced network operating costs. On the other hand, during periods of high usage, such as rush hours, the FPLF algorithm efficiently distributes traffic load among switches with low latency, ensuring smooth transitions between traffic patterns and maintaining network performance.

5.7.3 Limitations

As a result of these experiments, we discuss the disadvantages of the FPLF algorithm. Being a heuristic algorithm, it is unlikely to achieve the optimal solution. Additionally, in its current form, the FPLF algorithm cannot distribute the traffic load between switches based on the type of traffic constituting the flows, such as DNS, Ping, voice, video, and so on. Figure 5.9 illustrates core switch 4 in the proposed framework forwarding different flows from the aggregation layer without awareness of fault-tolerance values for those flows. Consequently, this might lead to unfair sharing of switches, as the FPLF algorithm could combine low or high fault-tolerance flows on one switch.

To address this limitation, we propose exploring the integration of machine learning classification techniques. For example, leveraging features used in approaches like [94] and classifiers in [95] could lead to a more equitable sharing of switches based on the characteristics of different flows. As for the FPLF algorithm's time complexity, potential reductions could be achieved if the algorithm searched subgraphs for paths instead of the entire graph, following a variant of the approach presented in [72]. However, a challenge lies in ensuring the obtainment of paths from multiple sub-graphs.

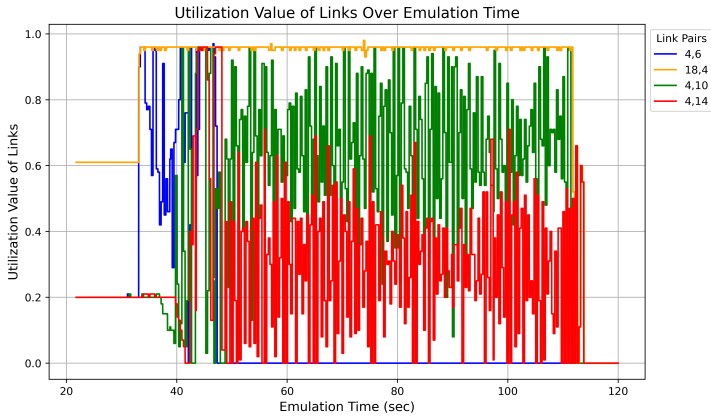


Figure 5.9: Different time-tolerance flows sharing one core switch.

5.8 Conclusion

In this chapter, we introduced the FPLF algorithm, an energy-efficient solution aimed at finding the right balance between saving power and maintaining performance in SDN-based DCNs. We transitioned from the theoretical concepts discussed in Chapter 3 to a practical approach applicable in the real world, focusing on efficiently conserving power in SD-DCNs while minimizing delays. FPLF's adaptability in selecting the best paths at specific times makes it a promising solution.

FPLF utilizes link utility information, taking advantage of SDN controllers and the OpenFlow protocol to identify subsets of switches that meet QoS requirements.

Additionally, we assessed the performance of the FPLF algorithm in a real DCN topology with varying traffic volumes. Using the Mininet emulator and the POX controller, FPLF outperformed the existing ECMP routing algorithm, demonstrating its effectiveness in balancing high traffic levels across the DCN topology. We observed significant energy savings, reaching up to 10% in high traffic and an impressive 63.3% in low traffic scenarios.

Looking ahead to our future work, we will focus on enhancing the FPLF algorithm's flow estimation model. Our goal is to develop a new model that not

only improves FPLF's performance but also analyzes the impact of the model on energy savings and scalability. We plan to leverage community detection and path anatomy techniques [72] in this endeavor.

5.9 Key Findings

Question: How can a traffic-aware approach be adopted to reduce power consumption in SD-DCNs?

Answer: The chapter proposed an adaptive routing algorithm for efficient power consumption in SD-DCNs. The proposed approach has demonstrated the ability to reduce the number of active links between OpenFlow switches in DCNs with low traffic, leading to a reduction in power consumption and operational costs [25].

Next Step: The upcoming chapter will illustrate how Machine Learning methods can be employed to classify *seven* types of DCNs traffic.

6

Traffic Classification

Contents

6.1 Introduction

In the preceding chapter, we introduced a novel routing strategy designed to enhance power efficiency in SD-DCNs. This strategy can blindly balance traffic flows between the core switches under high traffic conditions, without considering the specific services of the flow classes. However, certain classes, such as VoIP, video streaming, and Real-Time Protocol Applications (RTPA), can suffer from increased delays when mixed with other types of traffic. Consequently, we infer that such an algorithm could potentially impact the performance of real-time protocol applications. This realization prompted us to prioritize services for these specific classes (e.g., find an alternative path that meets their requirements), necessitating the development of a robust monitoring system capable of effectively identifying and rerouting such traffic to paths that meet their requirements. The central component of this monitoring system is the traffic classifier. Such a classifier plays a pivotal role in enhancing the performance of SD-DCNs. Therefore, this chapter proposes learning algorithms to classify a wide range of traffic in SDN environments. This work represents a preliminary step towards optimizing QoS for flow classes within the framework of SD-DCNs.

This is accomplished by creating a dataset that encompasses various types of network traffic, including Video, Voice over IP (VoIP), Gaming, and Internet Control Message Protocol (ICMP). We evaluated the performance of several Machine Learning techniques, such as Linear Regression, Support Vector Machine, and Neural Network, and the results were presented in [26]. Moreover, we discussed the integration of classification into the power consumption model, aiming for further advancements in this research area.

It is worth noting that a substantial portion of the content in this chapter draws upon the insights presented in the work of [26].

6.2 Traffic Classification Methods

Precise traffic classification is one of the essential networking functions that allows network operators to manage the network resources in an efficient manner. Traffic classification is particularly beneficial for solving QoS related problems,

which are associated with a wide range of computer networks issues such as power consumption in data centers.

Nowadays, data center networks are considered to be the backbone of many Internet applications and services in different forms such as multimedia and gaming.

As a part of the on-going project on minimizing the power consumption of data center networks using SDN, we recently introduced the FPLF routing protocol [25]. In FPLF, we designed a heuristic algorithms for energy-aware routing which was implemented using SDN. Since the data and control planes are decoupled in SDN, a global view of the data plane forwarding elements is possible, which makes network management and monitoring easier. In essence, FPLF reduces the usage of data plane links by fitting the incoming traffic flows to the smallest number of links.

Although FPLF is capable of reducing the power consumption in data center networks, it is expected that such gains would come at the cost of reduced QoS for delay sensitive flows like video, VoIP and online game. Therefore, traffic classification is crucial for FPLF so that the class of service information can be identified and become available to the routing and forwarding scheme.

Traffic classification can be achieved by three techniques, these are: (1) port-based, (2) deep packet inspection (3) and Machine Learning [96].

The traffic of modern services is often encrypted or not even associated with specific ports which makes both port-based and deep packet inspection approaches not very effective [97]. ML-based techniques have been widely explored to surpass the limitation of the traditional classification methods. In this chapter, we contribute a lightweight real-time traffic classification model based on ML techniques. We have implemented and evaluated the proposed method in an SDN environment and shown its effectiveness in accurately classifying a wide range of traffic applications in an online manner.

6.3 Related Work

This section provides a summary of recent research, with a focus on supervised machine learning (ML) methods, including Support Vector Machines (SVM), Random Forests (RF), Gradient Boosting (GB), and Neural Networks (NN).

In [98], the authors developed a system for collecting and selecting a set of flow metrics in SDN to achieve accurate traffic categorization. They employed Principal Component Analysis (PCA) and a Genetic Algorithm (GA) to identify the best features for traffic categorization, with SVM used as the classification algorithm. For validation, the authors conducted experiments with three layers of switches and hosts. The study demonstrated that PCA and GA achieved classification accuracy rates of over 91% and 88%, respectively.

However, it is important to note that the study focused on identifying the optimal subset of flow features and tested only one type of ML algorithm. Additionally, real-time testing was not conducted. Nevertheless, the developed system can be valuable for analyzing Distributed Denial of Service (DDoS), File Transfer Protocol (FTP), and Video Streaming traffic.

The authors in [99] introduced an SDN-based traffic categorization architecture. The OpenFlow protocol is used to collect the statistics from the forwarding elements for the purpose of feature extraction where that includes the first incoming five packets, interval arrival time, packet size, source and destination MAC/Port/IP, and flow duration. PCA was used to determine the best components in order to eliminate feature redundancy. An ensemble learning approach is applied in [99], in which three essential classifiers cast a weighted vote on the classification outcome. Random Forests (RF), Extreme Gradient Boosting (EGB), and Stochastic Gradient Boosting (SGB) were adopted. The reported accuracy of the experimental results was 90%.

The main limitation of this approach is that it relies on port number and payload inspection. This approach is not effective due to the dynamics of modern traffic.

The authors in [100] suggested using ML algorithms to recognize network flows. Four machine learning classifiers were employed, including: (1) Feedforward Neural Network, (2) Multilayer Perceptrons (MLP), (3) Levenberg-Marquardt, and (4) Naive Bayes. This method used full-flow data to estimate features such as five-tuple information, the average number of packets, and the average bytes in a flow. Data was collected through instant chatting, video, FTP, Hypertext Transfer Protocol (HTTP), streaming, and peer-to-peer protocols. These four categorization approaches achieved an accuracy of 95.6%, 97%, 97%, and 97.6%, respectively.

This approach must capture data from the whole flow, and such limitations may make it impossible to use early traffic classification.

The authors in [101] introduced Deep-SDN for predicting traffic flows in short response time. Deep-SDN has been tested by using Moore dataset [i] and not in an online fashion.

The approach in [95] demonstrated how features for video traffic can be constructed by explicitly modeling traffic time series. In this case, a QoS metric is modelled as a series of falling exponentials. This type of passive feature modelling approach could easily be extended to traffic classification instead of congestion detection, which was the target application in the paper.

Finally, due to the *dynamic and encrypted nature* of today's communication, traditional approaches such as identifying traffic based on port number and payload inspection are no longer effective. This work attempts to categorize the traffic flow based on its size and rate using supervised ML algorithms. It is worth to mention here that we have collected the data according to the accumulative value of packets and bytes that pass through the forwarding elements.

6.4 D-ITG Dataset

To evaluate the proposed classification methods, we meticulously constructed a dataset using a synthetic traffic generator. Initially, we set up a small-scale network topology with the Mininet emulator [88], which included two hosts, one switch, and one central controller. Each host in the network was connected to an OpenFlow switch [19], and the OpenFlow switch was securely connected to a Ryu controller through an OpenFlow channel, as visualized in the Figure 6.1.

In the application layer, we developed two crucial models: (1) a monitoring model[ii], designed to capture instantaneous flows, and (2) a collecting model, tasked with archiving traces for subsequent division into training and testing datasets.

The traffic flows were generated by the D-ITG generator tool [89]. D-ITG replicated the statistical properties of six traffic applications, including DNS, Telnet, Ping, Gaming, Quake-3, and VoIP. Additionally, we simulated video

[i] Moore dataset consists of eight kinds of traffic classes, and each flow included 249 feature attributes [102]

[ii] We utilized the same model discussed in Section 5.3.1. The distinction lies in solely collecting the aggregation matrix, $\mathbf{A}$, without incorporating additional functions.

traffic using VLC[iii] to establish real-time video streaming between a VLC server and client. As the traffic passed through the network switch, the monitoring model sent data to the collecting model, which extracted and archived the statistics of the arrival flows.

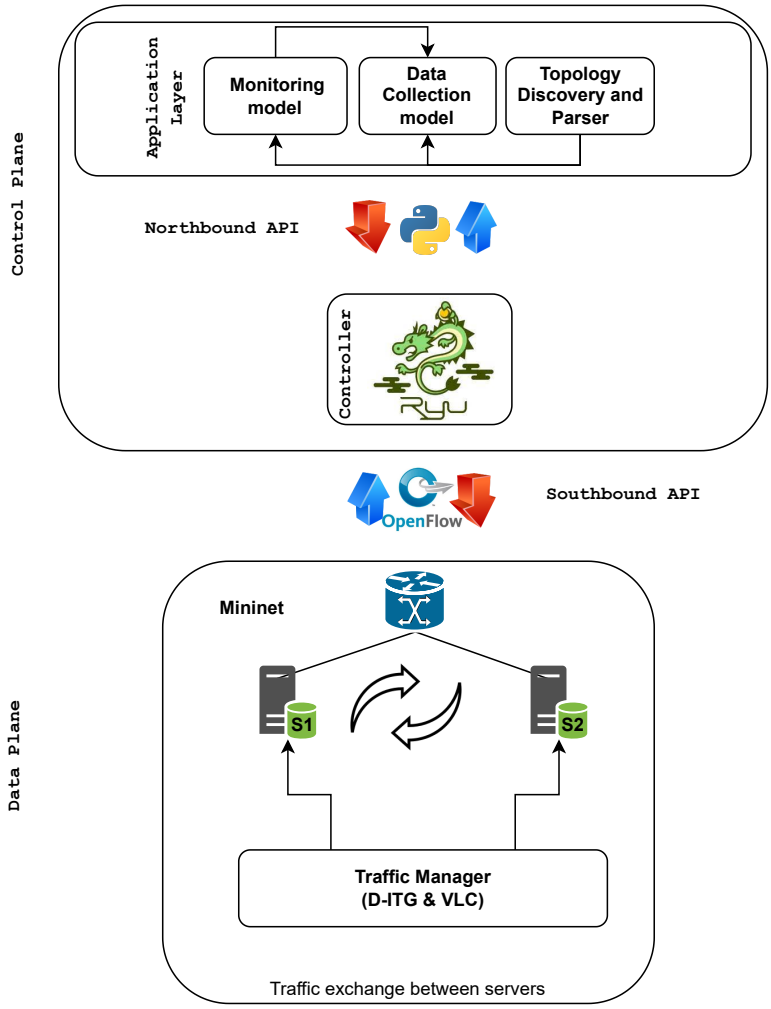


Figure 6.1: The structure of the SDN network in the data collection phase, including the monitoring and the collecting models. Python is used on the northbound interface, and OpenFlow is used on the southbound interface.

The primary function of the collecting model is to derive new features from

[iii]VLC is a free and open source cross-platform multimedia player and framework that plays most multimedia files, and various streaming protocols.

the raw data received every second from the monitoring model, represented by aggregation traffic, **A**. This raw data encompasses the aggregation matrix of packets/bytes in both the forward and backward directions between the source and destination.

These raw data are then used to construct a flow object with the following features:

1. Forward Packets.
2. Forward Bytes.
3. Delta forward Packets: the cumulative number of packets observed since the last forward flow was seen.
4. Delta forward bytes: the cumulative number of bytes observed since the last forward flow was seen.
5. Forward Instantaneous Packets per Second.
6. Forward Instantaneous Bytes per Second.
7. Forward Average Packets per Second.
8. Forward Average Bytes per second.
9. Reverse Packets.
10. Reverse Bytes.
11. Delta Reverse Packets.
12. Delta Reverse Bytes.
13. Reverse Instantaneous Packets per Second.
14. Reverse Instantaneous Bytes per Second.
15. Reverse Average Packets per second.
16. Reverse Average Bytes per second.

We ran each traffic application individually for approximately 150 minutes, resulting in up to 17,000 observations and 16 features for each experimental flow.

6.5 Classification Models

This section presents the implementation of three ML models, namely logistic regression, SVM, and Neural Network (NN) with different types of solvers. All models were trained using the Python programming language and the scikit-learn library except NN using TensorFlow, and executed on a workstation with the following specifications: Windows 10 with a 12-core Intel(R) Core(TM) i7-9750H CPU running at 2.6 GHz and 16 GB of DDR4 RAM ran these models.

The test of the models is performed on a portion of the dataset (i.e., test data). It is worth mentioning here that we dropped the features of "Forward Byte," "Forward Packets," "Reverse Byte," and "Reverse Packet." This is due to the cumulative nature of those features. They may have significantly different values from those that occur in the training phase. For instance, while those features start counting from zero in the training phase, they may start from another initial value in real traffic, which depends on the dynamic behavior of the network. Therefore, ML models might be negatively affected by the misleading information of such features. We aim to ensure the model's robustness and generalizability to real-world scenarios.

For the purpose of evaluation, we employed the following evaluation metrics to assess the model performance, including precision, recall, and F1-score. These metrics are commonly used in the evaluation of machine learning models and are particularly useful for assessing the performance of traffic classification models. Next, we provide the definitions of these metrics:

Precision (Pr): It is the ratio of correctly classified positive flows (TP) to all the flows classified as positive (TP+FP), as shown in (6.1).

$$Pr = \frac{TP}{TP + FP} \tag{6.1}$$

Recall (Rc): It is the ratio of correctly classified positive flows (TP) to all actual positive flows (TP+FN), as shown in (6.2).

$$Rc = \frac{TP}{TP + FN} \tag{6.2}$$

F1-score (F): It is a harmonic combination of precision and recall into a single

measure, as shown in (6.3).

$$F = 2 \cdot \frac{Pr \cdot Rc}{Pr + Rc} \tag{6.3}$$

6.5.1 Logistic Regression model

To construct a Logistic Regression (LR) model, the input is the dot product between a weight vector $\theta \in \mathbb{R}^n$ and an input vector $\mathbf{x}$ (i.e., input training data $x_k \in \mathbb{R}^n$) plus a bias b_k. And the output is the probability $\mathbf{y}_k \in \mathbb{R}$ to select the class k that maximizes (6.4), see [103]:

$$p\left(\mathbf{y}_k = 1 \mid \mathbf{x}\right) = \frac{\exp\left(\theta_k \cdot \mathbf{x} + b_k\right)}{\sum_{j=1}^{N} \exp\left(\theta_j \cdot \mathbf{x} + b_j\right)} \tag{6.4}$$

where, N is the number of classes.

For the purposes of regularization, feature selection, and providing better long-term predictions, we use Ridge Regression (i.e., ℓ_2 Regularization) and Lasso Regression (i.e., ℓ_1 Regularization) [104, 105]. We retained the default value of the inverse of the regularization strength (C), which is 1.0.

Table 6.1: Description of the Logistic Regression models

LR model number	Penalize level	Solver type	Iteration number	Accuracy (%)	Time (sec) of convergence
1	Penalty function/ L2	saga	25	75.31	03.56
2	Penalty function/ L1	liblinear	40	28.23	11.24
3	Penalty function/ L2	newton-cg	18	75.29	08.49
4	Penalty function/ L2	sag	19	75.60	01.67

Table 6.1 shows the results of the models' training based on the training time and accuracy value. We observed that models with solvers sag, saga, and newton had approximately the same accuracy, but different convergence times. On the other hand, the liblinear solver has inferior accuracy compared to the above ones. In Figure 6.2, the confusion matrix of the sag model helps to identify where the model is unsuccessful in accurately determining the target. We observed an apparent misclassification of game, telnet, quake-3, video, and ping traffic as DNS traffic. Furthermore, there is a significant number of the game traffic overlap with voice traffic. This is reasonable as both of their traffic

sources are interactive types of traffic. This is reflected in the precision value for each class of traffic, in the first and second columns of the classification report in Table 6.2.

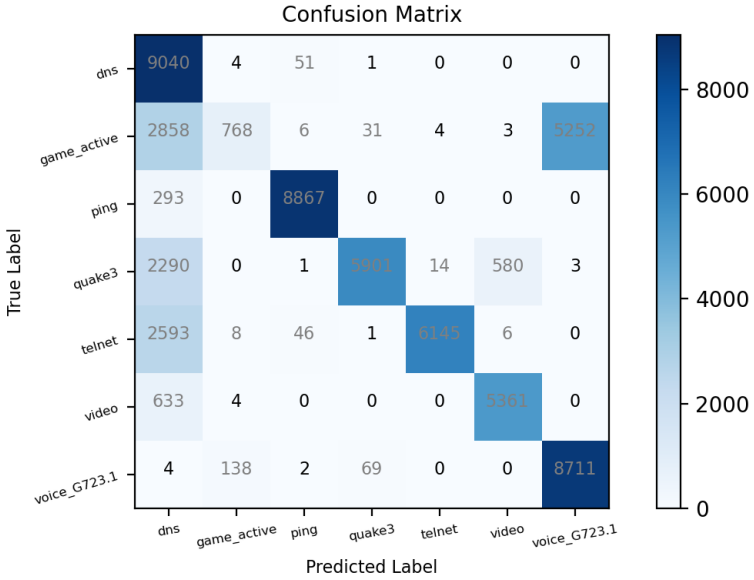


Figure 6.2: Confusion matrix for the LR-SAG model.

The report details precision and recall values, with telnet traffic exhibiting the highest precision and DNS and voice traffic registering the lowest. This discrepancy arises from the misclassification of various traffic types as DNS or voice traffic. Specifically, DNS traffic attains the maximum recall, while game traffic records the minimum due to misclassifications as either DNS or voice traffic.

The F-score, as presented in Table (6.2), offers a comprehensive assessment of our model's performance by combining precision and recall. Notably, ping traffic achieves the highest F-score, reflecting its commendable precision and recall scores.

6.5.2 SVM Model

SVMs represent a highly potent and versatile Machine Learning model, proficient in executing both linear and nonlinear classification, regression tasks,

Table 6.2: Classification report for the LR-SAG model.

Traffics	Precision	Recall	F1-score	Support
dns	0.51	0.99	0.67	9096
game	0.83	0.09	0.16	8922
ping	0.99	0.97	0.98	9160
quake3	0.98	0.67	0.80	8789
telnet	1.00	0.70	0.82	8799
video	0.90	0.89	0.90	5998
voice	0.62	0.98	0.76	8924
Accuracy			0.75	59688
Macro avg	0.83	0.76	0.73	59688
Weighted avg	0.83	0.75	0.72	59688

and even *outlier* detection. SVMs stand out as one of the most widely adopted models in the field of Machine Learning, making them an essential study for anyone interested in this domain [106].

SVMs demonstrate exceptional suitability for classifying datasets that are complex yet small or medium-sized. Their robustness and effectiveness contribute to their popularity, especially in scenarios where accurate classification is crucial.

Like in LR model, we train the SVM model with input training data $x_k \in \mathbb{R}^n$ and output label data $y_k \in \mathbb{R}$ with $(N = 7)$ classes mentioned in the Table 6.2 to construct the Support Vector Classifier (SVC) in the form [107]:

$$y(x) = \text{sign}\left[\sum_{k=1}^{N} \alpha_k y_k \psi\left(x, x_k\right) + b_k\right] \tag{6.5}$$

where α_k is a positive real constant, $\psi\left(x, x_k\right)$ represents the kernel, and b_k is the bias. In order to get the best decision boundary between classes, we use four different kernels as follows:

1. Linear kernel:

$$\psi\left(x, x_k\right) = x_k^T x$$

2. Polynomial-kernel:

$$\psi\left(x, x_k\right) = \left(x_k^T x + r\right)^d$$

3. Radial Basis Function (RBF) kernel:

$$\psi\left(x, x_k\right) = \exp\left\{-\left\|x - x_k\right\|_2^2 / \sigma^2\right\}$$

4. Sigmoid kernel:

$$\psi\left(x, x_k\right) = \tanh\left[\gamma x_k^T x + r\right]$$

where r, d, σ and γ are the kernel parameters used to maintain the separation line between classes.

According to Table 6.3, the sigmoid kernel reported significantly lower accuracy compared to the other kernels. Despite the linear kernel achieving 80.64% accuracy, its performance in online real tests cannot be guaranteed due to the potential nonlinearity of the data. Thus, the use of both sigmoid and linear kernels may not be the optimal choice.

While polynomial and RBF kernels are generally preferred for capturing non-linear hyperplanes and separating high-dimensional classes, the polynomial kernel achieved lower accuracy compared to the RBF kernel and consumed more time than other kernels. The inverse of the regularization strength (C) value is employed to balance regularization and control the trade-off between margins and misclassification terms. It is noteworthy that misclassifications in the polynomial kernel increase with higher trade-off C parameters, leading to larger margin hyperplanes and making classification more challenging in this model.

The most noteworthy result from the data is that the RBF kernel achieved higher accuracy in less time compared to the other models. This finding highlights the efficiency of the RBF kernel in achieving superior accuracy within a shorter time frame.

The evidence of the mentioned outcome is vivid in the confusion matrix for the RBF kernel, depicted in Figure 6.3, where the majority of predicted classes were accurately identified. Nevertheless, akin to the LR model, Figure 6.3 suggests a notable overlap among game, telnet, quake-3, and ping with DNS traffic, potentially resulting in misclassification.

This observation finds support in Table 6.4, revealing that DNS, game, and voice attained the lowest precision and F-score. Thus, the challenge in correctly labeling instances of DNS and game traffic is evident in the diminished precision and F-score values.

The inherent overlap among these traffic categories in the SVM model with the RBF kernel points to difficulties in precise differentiation.

Table 6.3: Classification report for the SVM models

SVM model number	Kernel type	Trade-off parameter (C value)	Accuracy (%)	Training Time (sec)
1	Linear	1	80.64	82
2	Polynomial	3	72.77	144
3	RBF	1	85.35	48
4	Sigmoid	1	62.36	107

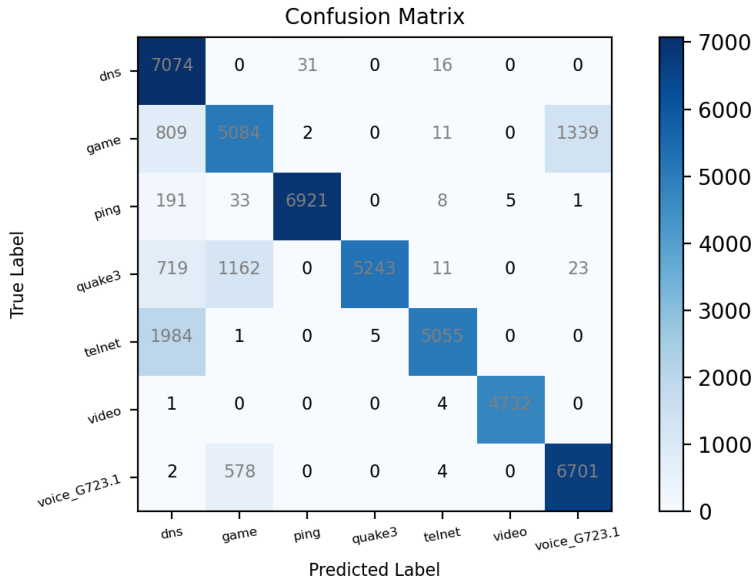


Figure 6.3: Confusion Matrix for the SVM (RBF kernel) model.

6.5.3 Neural Network model

Neural Networks form the cornerstone of Deep Learning, demonstrating versatility, power, and scalability. They stand out as an ideal solution for addressing vast and intricate Machine Learning tasks. Examples include the classification of billions of images, as seen in platforms like Google Images, the facilitation of speech recognition services, exemplified by Apple's Siri, and the provision of personalized video recommendations to millions of users daily on platforms such as YouTube.

Table 6.4: Classification report for SVM (RBF Kernel) model.

Traffics	Precision	Recall	F-score	Support
dns	0.66	0.99	0.79	7121
game	0.74	0.70	0.72	7245
ping	1.00	0.97	0.98	7159
quake3	1.00	0.73	0.85	7158
telnet	0.99	0.72	0.83	7045
video	1.00	1.00	1.00	4737
voice	0.83	0.92	0.87	7285
Accuracy			0.85	47750
Macro avg	0.89	0.86	0.86	47750
Weighted avg	0.88	0.85	0.86	47750

In order to construct a NN model for the classification task, we stack 7 sequential layers as shown in Table 6.5.

Table 6.5: Description of the NN model.

Layer type	**Output Shape**	**Param #**
Normalization	(None, 12)	25
Dense	(None, 24)	312
Dense	(None, 48)	1200
Dropout	(None, 48)	0
Dense	(None, 28)	1372
Dense	(None, 14)	406
Dense	(None, 7)	105
Total params: 3420		
Trainable params: 3395		
Non-trainable params: 25		

The model consists of the following layers:

(1) Layer normalization [108] to scale and shift input values with zero mean and unit variance. The layer normalization statistics are computed over all the input values $x_1, \ldots, x_n$ as follows:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

(2-3) Densely connected layers of size 24 and 48 in the figure, respectively, using the Rectifier Linear Unit (ReLU) activation function $f(x) = \max(0, x)$.

(4) A dropout layer with 50% dropping rate, for the sake of regularization and to avoid overfitting.

(5-6) Densely connected layers of size 28 and 14, respectively, using ReLU.

(7) A densely connected layer of size 7, followed by a softmax activation (6.6), which outputs one of the $N = 7$ class labels.

$$\sigma(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{N} \exp(x_j)} \tag{6.6}$$

During training, we use the sparse categorical cross entropy loss function to cope with this multi-class classification problem where labels are provided as integers.

As an optimization technique for gradient descent, we run the experiments with the Stochastic Gradient Descent (SGD) method as well as the Adaptive Moment Estimation (Adam) algorithm.

Table 6.6 shows the results of the NN models' training and validation, including the accuracy values and training time.

Table 6.6: Classification report for the NN models.

NN model number	Dropping rate	Optimizer type	Epoch number	Accuracy (%)	Time (sec) of convergence
1	0.5	Adam	10	89	31.00
2	0.5	SGD	10	79.94	32.00

According to Table 6.6, we observed that both SGD and Adam Optimizers had approximately the same convergence time, but different accuracy. Therefore, we employ the Adam Optimizer to test our approach in real-time action in Section 6.6. Figure 6.4 shows the confusion matrix for the model using Adam Optimizer. There is a slight overlap between games, voice, quake-3 and DNS traffics. Furthermore, we observed a significant number of telnet flows categorized as DNS traffic. This is reflected in the F-score values for each class of traffic, as shown by the classification report in Table 6.7.

On one hand, the minimum precision value was reported in the case of DNS traffic. On the other hand, the minimum recall value was observed in the case of telnet traffic, attributed to the misclassification of numerous flows as DNS traffic.

As a result, the report highlights that game, telnet, and DNS traffic exhibited lower F-score values compared to other types of traffic.

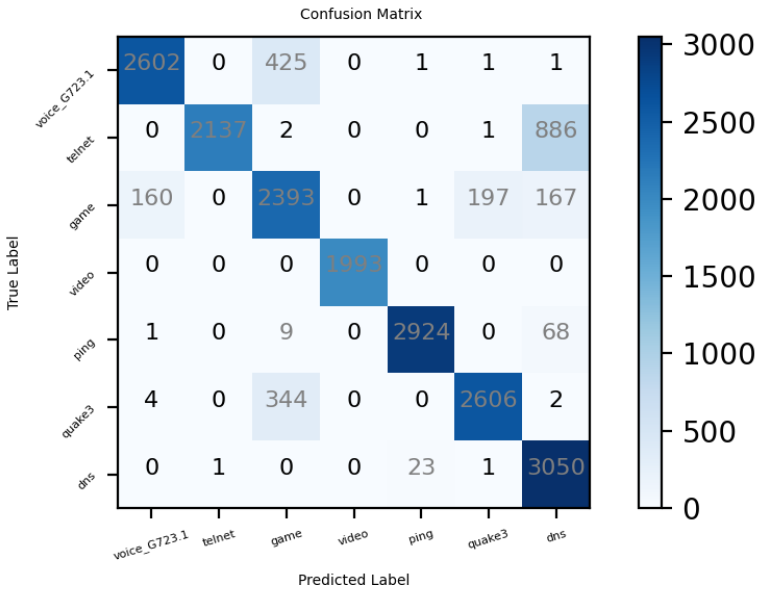


Figure 6.4: Confusion Matrix for the NN model.

Table 6.7: Classification report for the NN (Adam Optimizer) model.

Traffics	Precision	Recall	F-score	Support
voice	0.94	0.86	0.90	3030
telnet	1.00	0.71	0.83	3026
game	0.75	0.82	0.79	2918
video	1.00	1.00	1.00	1993
ping	0.99	0.97	0.98	3002
quake-3	0.93	0.88	0.90	2956
DNS	0.73	0.99	0.84	3075
Accuracy			89	20000
Macro avg	0.91	0.89	0.89	20000
Weighted avg	0.90	0.89	0.89	20000

In preparation for the subsequent online evaluation in Section 6.6, we serialized and stored the NN model as the best model using the Hierarchical Data Format (HDF5). This choice of storage format ensures efficient retention and easy retrieval of the models. In the subsequent testing phase, we invoked these serialized models within the suggestion framework. This step exposed the models to unseen data, allowing us to assess its performance on new traffic types, similar to those used during the model training process.

6.6 Online Testing

In this section, we delve into the evaluation of the NN model for predicting the labels of newly arriving flows in a real-time network scenario. To accomplish this, we seamlessly integrated the NN model with the SDN controller, alongside monitoring and topology discovery, as depicted in the online workflow illustrated in Figure 6.5. Subsequently, we simulated each traffic type for approximately 10 minutes, initiating a burst of 1000 flows for each traffic type.

The testbed was configured using a Linux 20.04 LTS machine with 8 GB RAM and a Core i7 processor. We established the network topology using the Mininet emulator, with Ryu serving as the adopted network controller.

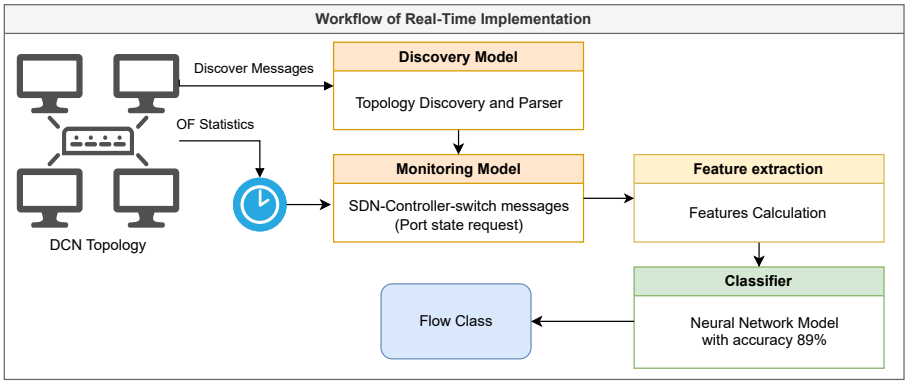


Figure 6.5: Workflow of the real-time implementation of the Neural Network classifier in an SDN environment.

Figure 6.6 illustrates the accuracy percentages for each traffic class. Notably, the classification model achieves 100% accuracy in predicting quake-3 traffic. Similarly, it excels in predicting DNS, video, and ping traffic classes, with accuracies reaching up to 99%, 97%, and 93%, respectively.

However, challenges arise in the classification of VoIP traffic, where the model fails to classify up to 14% of the injected traffic. Likewise, the model struggles with telnet traffic, misclassifying up to 35% of generated telnet traffic. The lowest performance is reported in predicting game traffic, with an accuracy of up to 13%.

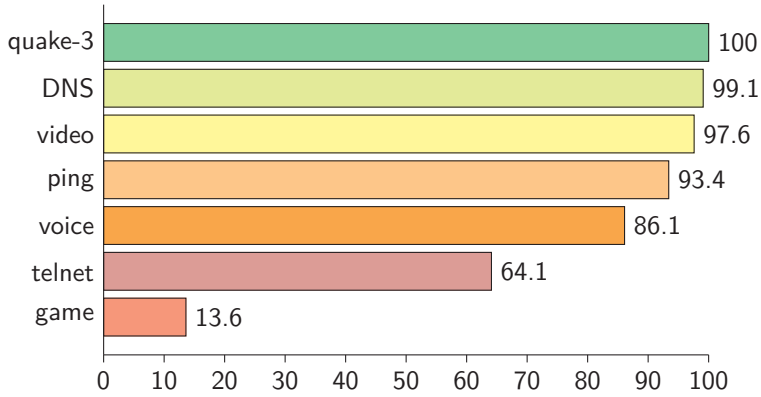


Figure 6.6: Result of SDN testing

6.7 Discussion

In this section, we compared the results obtained during the training and validation phases with the outcomes of the online testing for the NN model (with Adam optimizer).

During online testing, the most challenging scenario was encountered with the game traffic class, as depicted in Figure 6.7. This aligns with the validation phase, where game traffic exhibited the lowest F-score of 0.79. Conversely, online testing yielded the best results for both quake-3 and video traffic, with minimal or no misclassifications. This corresponds to the validation phase, where both quake-3 and video traffics demonstrated high F-score values of 0.90 and 1, respectively.

The impact and correlation were consistent in the case of telnet traffic, where online testing resulted in 64.1%, and validation yielded an F-score of 0.83.

In this work, we propose a lightweight approach for classifying/predicting network arrival traffic based solely on the packets' bit rate, eliminating the

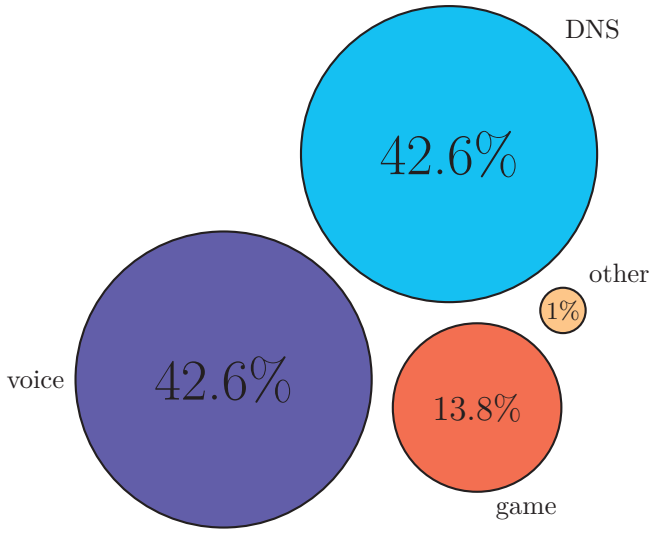


Figure 6.7: SDN testing on game traffic

need for further inspection, such as decapsulation of packet headers to retrieve additional data. This approach is particularly well-suited for secure traffic and sophisticated networks.

On the other hand, this approach presents a significant challenge, as all derived features rely on the size and speed of flow packets.

While the proof-of-concept design performs well, it does have limitations. In some cases, interruptions in flows due to network incidents can impact the accuracy of the model. This occurs, especially in scenarios where the features measure the average packet size. Flow interruptions may lead to decreased values for these features, consequently reducing the accuracy of the proposed model.

6.8 Limitations

While our approach effectively utilizes MAC addresses as host identifiers in SDN, it encounters a limitation related to parallel traffic between the same hosts. This limitation arises due to our exclusive reliance on MAC addresses for host identification. To address this, future enhancements could involve incorporating functionalities at higher network layers, such as the transport and

network layers. For instance, exploring the inclusion of ports or IP addresses may provide a more robust solution for handling parallel traffic between the same hosts.

These limitations underscore potential areas for refinement and expansion in future iterations of the lightweight classifier. Despite these constraints, the proposed classifier offers promising advantages to enhance performance in applications like real-time hotspot detection, congestion monitoring, and power optimization when integrated with SDN.

6.9 Conclusion

This chapter demonstrated the promise of applying machine learning approaches to the problem of traffic classification for SDNs. Three machine learning models LR, SVM and NN were implemented and evaluated in the context of classifying network traffic of different classes based on 16 selected features. We used the D-ITG in order to built a synthetic traffic dataset for this purpose. The offline testing results revealed that the LR with sag achieved an accuracy of 75.6%, the SVM with RBF achieved 85.35%, and the NN with ADAM optimizer achieved 89%. After testing the three models, we decided to incorporate only the NN model into an SDN implementation to facilitate further online measurements. Four metrics were employed in the evaluation process: accuracy, precision, recall, and F-measure.

As a future plan, we intend to combine the proposed model with our FPLF algorithm [25] in order to optimize the power consumption of software-defined data center networks and wireless networks [2, 3].

6.10 Key Findings

Question: How can we predict the class of service for each flow in SD-DCNs to enhance flow quality?

Answer: This chapter demonstrated the ability of supervised machine learning techniques in classifying seven types of flow classes without using packet investigation methods, relying solely on OpenFlow statistics [26].

Next Step: The upcoming chapter will explore *open issues* and *future directions* of the current thesis.

7

Future Directions and Summary

Contents

7.1 Introduction

This chapter introduces a mathematical scheduling model designed to minimize power consumption while ensuring that flow delay metrics remain below a specified threshold. The future objective is to evaluate its performance and draw comparisons with the model proposed in Chapter 3. Subsequently, we explore open issues and potential future directions that can be built upon the ideas presented in the current thesis.

A promising avenue for future exploration involves employing multiple controllers to enhance computational resources and the scalability of the SD-DCNs. However, this expansion must be approached with mindfulness towards power consumption concerns. Additionally, there is a prospect of integrating deep learning techniques to address the overhead problem in SD-DCNs. Utilizing learning algorithms to analyze traffic patterns and predict future traffic demand can assist SD-SDNs controllers in making real-time adjustments and reconfiguring network devices within their control domain to optimize power usage and reduce consumption.

Finally, this chapter concludes with a concise summary of the main points discussed in the thesis..

7.2 Flow Scheduling Model

In this section, we present the objective function and constraints that mathematically formulate an ILP model that optimizes power consumption in DCNs using flow scheduling techniques. It's worth noting that the suggested scheduling model operates under the same three assumptions that are applied to the consolidation model discussed in Section 3.3.

The goal we seek to achieve with the objective function in (7.1) is to minimize power consumption while maintaining network performance metrics. Prior to defining the objective function we define the role of its constituent components in Table 7.1.

The objective function computes a weighted sum of the variables L_{ijk} for all

Table 7.1: Parameters of the flow scheduling model.

Parameters	Definitions
L_{ijk}	$\begin{cases} 1, & \text{if link } e_{ij} \in \mathbb{E} \text{ is active during timeslot } k \\ 0, & \text{otherwise} \end{cases}$
$FR(f,i,j,k)$	$\begin{cases} 1, & \text{if flow } f \in \mathbb{F} \text{ is scheduled on link } e_{ij} \in \mathbb{E} \\ & \quad \text{during timeslot } k. \\ 0, & \text{otherwise} \end{cases}$
$C_f \in \mathbb{N}$	Bandwidth required by flow f in Mbps
$La_f \in \mathbb{N}$	Latency tolerance of flow f in microseconds
$\alpha_f \in \mathbb{N}$	Starting timeslot of flow f
$\omega_f \in \mathbb{N}$	Ending timeslot of flow f, $\alpha_f \leq \omega_f$
$la_{ijk} \in \mathbb{N}$	Latency of link e_{ij} during timeslot k
$p_{ij} \in \mathbb{N}$	Power consumption of link e_{ij} in watts

active time slots and the power consumption p_{ij} for each link.

$$\min : \sum_{i=1}^{n} \sum_{j=1}^{n} \left(p_{ij} \cdot \sum_{k} L_{ijk} \right) \tag{7.1}$$

The above objective function is subject to the following constraints to guarantee network performance: bandwidth, latency, etc.

1. Bandwidth constraint.

$$\sum_{f \in \mathbb{F}} C_f \cdot FR(f,i,j,k) \leq BW_{ij}, \quad \forall e_{ij} \in \mathbb{E}, \ \forall k.$$

   The total bandwidth usage of each link should not exceed its maximum bandwidth capacity BW_{ij} during any timeslot k.

2. Latency constraint.

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k} la_{ijk} \cdot FR(f,i,j,k) \leq La_f, \quad \forall f \in \mathbb{F}.$$

   The sum of the latencies of all links used by a flow f must not exceed its latency tolerance La_f.

3. Workload constraint.

$$\sum_{f \in \mathbb{F}} FR(f,i,j,k) \leq 1, \quad \forall e_{ij} \in \mathbb{E}, \ \forall k.$$

Each timeslot k can have at most one flow scheduled on a link e_{ij}.

4. Flow activation constraint.

$$FR(f,i,j,k) = 0, \quad \forall f \in \mathbb{F},\ \forall e_{ij} \in \mathbb{E},\ \forall k \notin [\alpha_f, \omega_f].$$

The decision variables $FR(f,i,j,k)$ are set to 0 for each flow f outside of its specified time interval between α_f and ω_f. α_f represents the timeslot at which the flow f is allowed to start its transmission. ω_f represents the timeslot by which the flow f must be completely served or finished with its transmission.

5. Links and flows correlation constraint.

$$FR(f,i,j,k) \leq L_{ijk}, \quad \forall f \in \mathbb{F},\ \forall e_{ij} \in \mathbb{E},\ \forall k.$$

Flows can only pass through active links, enforcing L_{ijk} to be in the ON state.

Similarly, the Path conservation constraint and the Flow conservation constraint from Section 3.3.1 have to be adapted, simply by quantifying over the timeslots k.

Solving the ILP model with these constraints results in an optimal flow scheduling policy that minimizes power consumption and keeps the traffic latency at an acceptable level.

7.3 Open Issues and Discussion

While the SDN paradigm has presented solutions to many existing network issues, such as those related to power consumption, further investigation and verification are required for SDN-based Data Center Network power consumption, especially under various workload circumstances.

Implementing power optimization approaches based on SDN may potentially increase the *response time* of the controller. This is due to the continuous monitoring of the network and decision-making processes aimed at optimizing power consumption while maintaining network performance. Such overhead could introduce delays in the controller's response time.

Additionally, the process of finding the *optimal subset* of active links and switches can be resource-intensive, demanding significant *computational resources*. This resource allocation for optimization purposes may lead to a performance loss in a DCN, as the resources are diverted from critical network functions. Furthermore, certain approaches necessitate the use of *heuristics or approximation algorithms*, as seen in the current thesis, typically providing only suboptimal solutions.

Therefore, prior to the implementation of power optimization approaches based on SDN, it is crucial to carefully evaluate the potential trade-offs between power consumption, network performance, and computational resources. Further investigation may be necessary to identify the optimal balance between power optimization and network performance in a specific DCN. Subsequent research incorporating these variables will be essential. Moreover, exploring the integration of machine learning models and implementing multiple SDN controllers within SD-DCNs holds potential advantages that could enhance network performance.

In the following, we will suggest new directions to deal with such problems in the future.

7.3.1 Multiple Controller SDN

Using multiple controllers can potentially help mitigate the problem of increased response time in power optimization approaches based on SDN. By distributing the workload across multiple controllers, the overall response time can be reduced, improving network performance. Moreover, multiple controllers can potentially improve the efficiency of finding the optimal subset of active links and switches. Different controllers can be responsible for different parts of the network, and they can work together to find the best solution. This can help to reduce the computational demand and improve the speed of optimization. However, the synchronization of multiple controllers in an SDN environment can be achieved through various methods, such as: (1) *Consistency Protocols* ensure that all controllers have the same view of the network. (2) *Event Notifications* are used by controllers in order to communicate with each other. For example, the OpenDaylight (ODL) controller provides a notification service that allows controllers to subscribe to events and to receive notifications when the state of the network changes. (3) *Replication*: Controllers can replicate/duplicate their state and share this copy with other controllers in the network. For

example, the Floodlight controller uses a master-slave replication mechanism to synchronize the state of multiple controllers. By using one or more of these methods, multiple controllers in an SDN environment can synchronize their work and ensure the efficient operation of the network. Therefore, when considering the use of multiple controllers, it is important to carefully evaluate the potential benefits and drawbacks, striking a balance between overhead and power consumption. Subsequently, the network architecture and control algorithms should be designed accordingly.

7.3.2 Machine Learning-Based Approaches

Machine Learning is a promising approach for solving a wide range of computer networks problems. ML-based techniques can be used to optimize power consumption in SD-DCN. By analyzing network traffic patterns and predicting network demand, ML algorithms can help SDN controllers make real-time adjustments and reconfigure the network to optimize the energy usage. One avenue worth exploring in research involves utilizing ML to *predict the network demand* for different time periods, such as hourly or daily intervals [23]. Based on these predictions, the SDN controller can adjust the routing of network traffic to reduce power consumption during periods of lower demand.

Other ML algorithms can be used to *analyze the behavior of individual network components*, such as switches or servers, to identify patterns that indicate when these components are not being used efficiently [109]. This information can then be used by the SDN controller to adjust network configurations and optimize power usage.

Finally, using ML algorithms to classify network traffic in real-time, SDN controllers can make more informed decisions about network traffic routing and resource allocation, leading to improved network performance and efficiency besides efficient power usage [26]. Overall, ML-based approaches have the potential to significantly reduce power consumption in DCNs, which is an important consideration for organizations looking to improve their energy efficiency and to reduce their carbon footprint.

7.3.3 DCNs with Adaptive Link Speed

In the current objective function in 3.3, the formula does not account for the speed levels of the links (switches, ports) but rather focuses solely on their ON/OFF state. However, the formula could be extended to control port speed for optimization purposes by introducing a new decision variable, L_{ij}^{h}, where h represents the speed of the link. Furthermore, we are currently working on optimizing power usage by controlling the speed of the ports using formal methods. We have considered three speed levels: 10 Mbps, 100 Mbps, and 1 Gbps. We have also obtained fresh results. Such a technique may increase the efficient usage of power in DCNs because ports consume less power at lower speed levels.

7.4 Thesis Points

In this section, we summarized the key theses derived from the previous chapters, categorized under three main areas: ILP for power usage modeling, heuristic algorithms for real-time emulation, and machine learning algorithms for mitigating the effects of consolidation techniques.

7.4.1 Thesis 1: Power Usage Modeling in DCNs Using a Consolidation ILP Model

1. **Problem Statement**: The escalating power usage in DCNs is a critical concern. Traditional network structures face significant limitations in optimizing power consumption, necessitating advanced methodologies.

2. **Approach**: This thesis focuses on solving the power optimization problem in DCNs using ILP formulations.

3. **Key Findings**: The ILP model was implemented and solved using various ILP solvers, including LinGo, GUROBI, CP-SAT, SCIP, and CBC. Comprehensive experiments were conducted on different traffic patterns (near, long, and random) using the tool NEO-DCN. The results showed that for the near traffic pattern, GUROBI and CP-SAT outperformed

other solvers in terms of runtime. The long traffic pattern highlighted the challenges posed by increasing traffic volumes, with CP-SAT maintaining stability even in high-demand scenarios. The random traffic matrix experiment further emphasized the superior performance of GUROBI in terms of both runtime and memory consumption across a diverse range of traffic instances. Overall, the findings suggest that GUROBI is a robust and efficient solver for the power optimization problem in DCNs, providing valuable insights for future research and practical implementations.

7.4.2 Thesis 2: Real-time Emulation to Efficient Power Usage in SDN Environment

1. **Problem Statement**: Efficient power usage in SDN environments requires real-time solutions that can adapt to varying traffic conditions while maintaining performance.

2. **Approach**: This thesis explores the emulation setup and implementation of the Fill Preferred Link First (FPLF) algorithm in software-defined data center networks.

3. **Key Findings**: The FPLF algorithm was tested on a fat-tree topology using Mininet and the POX controller. Experiments demonstrated that in high-traffic scenarios, the FPLF algorithm achieved up to 10% energy savings, while in low-traffic situations, the algorithm realized an impressive 63.3% energy savings compared to the existing ECMP routing algorithm. These results underscore the algorithm's adaptability and significant energy savings in various traffic scenarios. However, the FPLF algorithm does have limitations, particularly in its ability to estimate flows accurately and equitably distribute them. Additionally, it does not prioritize specific types of traffic, such as VoIP, video streaming, and Real-Time Protocol Applications (RTPA), which can suffer from increased delays when mixed with other traffic types. This limitation highlights the need for further enhancements in flow estimation models and the development of mechanisms to ensure equitable flow distribution and priority handling of real-time traffic.

7.4.3 Thesis 3: Mitigation of the Effects of Consolidation Technique on the Quality of the RTAP Classes

1. **Problem Statement**: FPLF can blindly balance traffic flows between the core switches under high traffic conditions without considering the specific services of the flow classes. However, certain classes, such as VoIP, video streaming, and Real-Time Protocol Applications (RTPA), can suffer from increased delays when mixed with other types of traffic. Consequently, such an algorithm could potentially impact the performance of real-time protocol applications. This realization prompted us to prioritize services for these specific classes, necessitating the development of a robust monitoring system capable of effectively identifying and rerouting such traffic to paths that meet their requirements. The central component of this monitoring system is the **traffic classifier**, which plays a pivotal role in enhancing the performance of SD-DCNs. Therefore, this thesis proposes learning algorithms to classify a wide range of traffic in SDN environments.

2. **Approach**: This thesis leverages machine learning models for traffic classification within SDN to enable distributed flow according to class type.

3. **Key Findings**: Three machine learning models were implemented: Logistic Regression, Support Vector Machine, and Neural Network. These models were trained and assessed using both synthetic and real traffic datasets. The Neural Network model emerged as a standout performer, exhibiting high accuracy in classifying diverse traffic types, up to 89%. Further integration of the Neural Network model into an SDN environment for real-time online testing indicated commendable accuracy across various traffic classes, though challenges were noted in classifying specific types such as games and telnet. The findings suggest that the proposed machine learning approach shows promising potential for optimizing power consumption within SD-DCNs.

7.5 Conclusion

The escalating power usage in DCNs has emerged as a global concern. Notably, active research is being conducted on energy optimization techniques, with a focus on scheduling and flow aggregation methods. This chapter proposes a traffic scheduling technique that strives for a balance between power optimization and flow latency for DCNs, providing a foundation for further exploration in future investigations.

Additionally, it sheds light on the limitations associated with multi-controller SDN solutions, primarily stemming from performance constraints and how to address them in the future. Future research avenues should consider harnessing the potential of machine learning methodologies to optimize traffic-aware methods for efficient power consumption and improve Quality of Service, etc Moreover, there is a need to explore hybrid solutions that integrate advanced scheduling and aggregation techniques, along with the implementation of multi-controller setups.

8

Acknowledgements

Writing this dissertation has been a journey filled with guidance, encouragement, and support from many wonderful people. I extend my deepest gratitude to my supervisor, Dr. Gergely Kovásznai. His extensive knowledge, attention to detail, and unwavering support have been crucial from the beginning of my research on "Network Power Optimization Techniques" to its completion. Dr. Kovásznai approachability, valuable feedback, and dedication to excellence motivated me to face challenges with determination and thoroughness.

A special place in my heart is reserved for my Mom and Dad. Their unwavering love, constant support, and endless encouragement have been my strength and motivation throughout this journey. Their lessons and values have shaped me, and their faith in me has been a guiding light, pushing me to strive for excellence.

I also express my gratitude to my family for their patience and support throughout my study journey. Special thanks to my wife for being by my side and assisting me during that time.

The financial support from my home country, Iraq, and Stipendium Hungaricum deserves special recognition. Their generosity significantly alleviated the financial challenges of my academic journey, enabling me to concentrate on my research. I also want to extend my sincere gratitude to Science Foundation Ireland, represented specifically by Dr. Ruairí de Fréin, and Dr. Ali Malik, for their collaboration and generous support in funding some research papers during specific phases of my Ph.D. journey.

Mohammed Al-shamarti

Debrecen, Hungary, 2024

References

[1] Penghao Sun, Zehua Guo, Sen Liu, Julong Lan, Junchao Wang, and Yuxiang Hu. Smartfct: Improving power-efficiency for data center networks with deep reinforcement learning. *Computer Networks*, 179:107255, 2020.

[2] Furkan Rabee, Mohammed Nsaif, and Ali Al-Haboobi. Reliable compression route protocol for mobile crowd sensing (RCR-MSC). *Journal of Communications*, 14:pp. 170–178, 02 2019.

[3] Furkan Rabee, A Al-Haboobi, and Mohammed Ridha Nsaif. Parallel three-way handshaking route in mobile crowd sensing (PT-MCS). *J Eng Appl Sci*, 14:3200–3209, 2019.

[4] Statista. Worldwide data created and consumed. `https://www.statista.com/statistics/871513/worldwide-data-created/`, 2023. 21-10-2023.

[5] Javier Arribas Cámara and Vicente Sánchez Jiménez. The european union facing the abyss: legislative review in the face of the energy crisis, 2022. *Journal of Energy & Natural Resources Law*, pages 1–16, 2023.

[6] Beakal Gizachew Assefa and Öznur Özkasap. A survey of energy efficiency in sdn: Software-based methods and optimization models. *Journal of Network and Computer Applications*, 137:127–143, 2019.

[7] Andre Rios, Alberto J Gonzalez, Jesus Alcober, Javier Ozon, and Kayhan Z Ghafoor. Conferencing services in P2P networks: Trends and challenges. *Future Internet Services and Service Architectures*, pages 139–160, 2022.

[8] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. Recalibrating global data center energy-use estimates. *Science*, 367(6481):984–986, 2020.

[9] Ralph Hintemann and Simon Hinterholzer. Energy consumption of data centers worldwide. *Business, Computer Science (ICT4S)*, 2019.

[10] Mohammed Nsaif, Gergely Kovásznai, Ali Malik, and Ruairí de Fréin. Survey of routing techniques-based optimization of energy consumption in sd-dcn. *INFOCOMMUNICATIONS JOURNAL*, 15(SI):35–42, 2023.

[11] Chandrakant D Patel, Cullen E Bash, Ratnesh Sharma, Monem Beitelmal, and Rich Friedrich. Smart cooling of data centers. In *International Electronic Packaging Technical Conference and Exhibition*, volume 36908, pages 129–137, 2003.

[12] D Suleiman, M Ibrahim, and I Hamarash. Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction. In *4th International Conference on Electrical and Electronics Engineering*, volume 12, 2005.

[13] Md Faizul Bari, Raouf Boutaba, Rafael Esteves, Lisandro Zambenedetti Granville, Maxim Podlesny, Md Golam Rabbani, Qi Zhang, and Mohamed Faten Zhani. Data center network virtualization: A survey. *IEEE communications surveys & tutorials*, 15(2):909–928, 2012.

[14] Nisha Chaurasia, Mohit Kumar, Rashmi Chaudhry, and Om Prakash Verma. Comprehensive survey on energy-aware server consolidation techniques in cloud computing. *The Journal of Supercomputing*, 77:11682–11737, 2021.

[15] John T Moy. *OSPF: anatomy of an Internet routing protocol*. Addison-Wesley Professional, 1998.

[16] Charles L Hedrick. Routing information protocol. Technical report, 1988.

[17] Internet Engineering Task Force (IETF). IP Flow Information Export (IPFIX) Protocol Specification. Technical report, RFC 7011, 2013.

[18] Cisco Systems, Inc. NetFlow Services and Solutions Guide. Retrieved from `https://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/3-6/user/guide/cnfguide.html`, 2008.

[19] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74, 2008.

[20] Michael Carter, Camille C Price, and Ghaith Rabadi. *Operations research: a practical introduction*. Crc Press, 2018.

[21] Wayne L. Winston. *Operations Research Applications and Algorithms*. Curt Hinrichs, 1987.

[22] Asaju La'aro Bolaji, Mohammed Azmi Al-Betar, Mohammed A Awadallah, Ahamad Tajudin Khader, and Laith Mohammad Abualigah. A comprehensive review: Krill herd algorithm (kh) and its applications. *Applied Soft Computing*, 49:437–446, 2016.

[23] Duc-Huy Le, Hai-Anh Tran, Sami Souihi, and Abdelhamid Mellouk. An AI-based traffic matrix prediction solution for software-defined network. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE, 2021.

[24] Gergely Kovásznai and Mohammed Nsaif. Integer programming based optimization of power consumption for data center networks. *Acta Cybernetica*, Nov. 2023.

[25] Mohammed Nsaif, Gergely Kovásznai, Anett Rácz, Ali Malik, and Ruairí de Fréin. An adaptive routing framework for efficient power consumption in software-defined datacenter networks. *Electronics*, 10(23):3027, 2021.

[26] Mohammed Nsaif, Gergely Kovásznai, Mohammed Abboosh, Ali Malik, and Ruairí de Fréin. ML-Based Online Traffic Classification for SDNs. In *2022 IEEE 2nd Conference on Information Technology and Data Science (CITDS)*, pages 217–222. IEEE, 2022.

[27] Brandon Heller, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. ElasticTree: Saving energy in data center networks. In *Nsdi*, volume 10, pages 249–264, 2010.

[28] Min Sang Yoon and Ahmed E Kamal. Power minimization in fat-tree SDN datacenter operation. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2015.

[29] Tran Manh Nam, Nguyen Huu Thanh, Ngo Quynh Thu, Hoang Trung Hieu, and Stefan Covaci. Energy-aware routing based on power profile of devices in data center networks using SDN. In *2015 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 1–6. IEEE, 2015.

[30] Daniel Stutzbach and Reza Rejaie. Characterizing the two-tier gnutella topology. *ACM SIGMETRICS Performance Evaluation Review*, 33(1):402–403, 2005.

[31] Saima Zafar, Abeer Bashir, and Shafique Ahmad Chaudhry. On implementation of dctcp on three-tier and fat-tree data center network topologies. *SpringerPlus*, 5(1):1–18, 2016.

[32] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 75–86, 2008.

[33] Vahid Asghari, Reza Farrahi Moghaddam, and Mohamed Cheriet. Performance analysis of modified bcube topologies for virtualized data center networks. *Computer Communications*, 96:52–61, 2016.

[34] Rastin Pries, Michael Jarschel, Daniel Schlosser, Michael Klopf, and Phuoc Tran-Gia. Power consumption analysis of data center architectures. In *Proc. Int. Conf. on Green Communications and Networking (GreeNets)*, volume 51 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 114–124. Springer, 2012.

[35] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review*, 40(1):92–99, 2010.

[36] Ali Malik and Ruairí de Fréin. A proactive-restoration technique for SDNs. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2020.

[37] Xiaodong Wang, Yanjun Yao, Xiaorui Wang, Kefa Lu, and Qing Cao. Carpo: Correlation-aware power optimization in data center networks. In *2012 Proceedings IEEE INFOCOM*, pages 1125–1133, 2012.

[38] Nguyen Huu Thanh, Pham Ngoc Nam, Thu-Huong Truong, Nguyen Tai Hung, Luong Kim Doanh, and Rastin Pries. Enabling experiments for energy-efficient data center networks on openflow-based platform. In *2012 Fourth International Conference on Communications and Electronics (ICCE)*, pages 239–244. IEEE, 2012.

[39] Junhua Ba, Ying Wang, Xuxia Zhong, Sixiang Feng, Xuesong Qiu, and Shaoyong Guo. An SDN energy saving method based on topology switch and rerouting. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–5. IEEE, 2018.

[40] Marcelo da Silva Conterato, Tiago Coelho Ferreto, Fábio Rossi, Wagner dos Santos Marques, and Paulo Silas Severo de Souza. Reducing energy consumption in SDN-based data center networks through flow consolidation strategies. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 1384–1391, 2019.

[41] LINDO Systems Inc. Lingo the modeling language and optimizer. `http://www.lindo.com`, 2020.

[42] Tran Hoang Vu, Pham Ngoc Nam, Tran Thanh, Le Thai Hung, Le Anh Van, Nguyen Duy Linh, To Duc Thien, and Nguyen Huu Thanh. Power aware open-flow switch extension for energy saving in data centers. In *The 2012 International Conference on Advanced Technologies for Communications*, pages 309–313, 2012.

[43] Obinna Izima, Ruairí de Fréin, and Ali Malik. A survey of machine learning techniques for video quality prediction from quality of delivery metrics. *Electronics*, 10(22), 2021.

[44] Yunfei Shang, Dan Li, and Mingwei Xu. Greening data center networks with flow preemption and energy-aware routing. In *2013 19th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN)*, pages 1–6. IEEE, 2013.

[45] Dan Li, Yunfei Shang, and Congjie Chen. Software defined green data center network with exclusive routing. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1743–1751. IEEE, 2014.

[46] Rui Wang, Zhipeng Jiang, Suixiang Gao, Wenguo Yang, Yinben Xia, and Mingming Zhu. Energy-aware routing algorithms in software-defined networks. In *Proc. IEEE Int. Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pages 1–6. IEEE, 2014.

[47] Dan Li, Yirong Yu, Wu He, Kai Zheng, and Bingsheng He. Willow: Saving data center network energy for network-limited flows. *IEEE Transactions on Parallel and Distributed Systems*, 26(9):2610–2620, 2014.

[48] Guan Xu, Bin Dai, Benxiong Huang, and Jun Yang. Bandwidth-aware energy efficient routing with SDN in data center networks. In *2015 IEEE 17th international conference on high performance computing and communications, 2015 IEEE 7th international symposium on cyberspace safety and security, and 2015 IEEE 12th international conference on embedded software and systems*, pages 766–771. IEEE, 2015.

[49] Guan Xu, Bin Dai, Benxiong Huang, Jun Yang, and Sheng Wen. Bandwidth-aware energy efficient flow scheduling with SDN in data center networks. *Future Generation computer systems*, 68:163–174, 2017.

[50] Juan Luo, Song Zhang, Luxiu Yin, and Yaling Guo. Dynamic flow scheduling for power optimization of data center networks. In *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*, pages 57–62. IEEE, 2017.

[51] Zan Yao, Ying Wang, and Xuesong Qiu. Dqn-based energy-efficient routing algorithm in software-defined data centers. *International Journal of Distributed Sensor Networks*, 16(6):1550147720935775, 2020.

[52] Beakal Gizachew Assefa and Oznur Ozkasap. Framework for traffic proportional energy efficiency in software defined networks. In *2018 IEEE Int. Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 1–5. IEEE, 2018.

[53] Laurent Perron and Vincent Furnon. OR-Tools. `https://developers.google.com/optimization/`, 2019.

[54] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. `https://www.gurobi.com/documentation/9.5/refman/`, 2022.

[55] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 8.0. ZIB-Report 21-41, Zuse Institute Berlin, December 2021. `http://nbn-resolving.de/urn:nbn:de:0297-zib-85309`.

[56] John Forrest, Ted Ralphs, Haroldo Gambini Santos, Stefan Vigerske, Lou Hafer, John Forrest, Bjarni Kristjansson, jpfasano, EdwinStraver, Miles Lubin, rlougee, jpgoncal1, Jan-Willem, h-i gassmann, Samuel Brito, Cristina, Matthew Saltzman, tosttost, Fumiaki MATSUSHIMA, and to st. coin-or/cbc: Release releases/2.10.7, January 2022.

[57] G. Kovásznai, K. Gajdár, and L. Kovács. Portfolio SAT and SMT solving of cardinality constraints in sensor network optimization. In *2019 21st Int. Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 85–91, 2019.

[58] Michael M McKerns, Leif Strand, Tim Sullivan, Alta Fang, and Michael A. G. Aivazis. Building a framework for predictive science. *arXiv preprint arXiv:1202.1056*, 2012.

[59] Xiaolin Li, Chung-Horng Lung, and Shikharesh Majumdar. Green spine switch management for datacenter networks. *Journal of Cloud Computing*, 5(1):1–19, 2016.

[60] Software-defined networking: The new norm for networks. `https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/`, Year of Access. Accessed: February 20, 2024.

[61] Salekul Islam, Nasif Muslim, and J William Atwood. A survey on multicasting in software-defined networking. *IEEE Communications Surveys & Tutorials*, 20(1):355–387, 2017.

[62] Duohe Ma, Zhen Xu, and Dongdai Lin. Defending blind ddos attack on sdn based on moving target defense. In *International Conference on Security and Privacy in Communication Networks: 10th International ICST Conference, SecureComm 2014, Beijing, China, September 24-26, 2014, Revised Selected Papers, Part I 10*, pages 463–480. Springer, 2015.

[63] M McCauley. Pox web interfaces. `https://github.com/noxrepo/`, 2012. [Online] Accessed on 1st January 2024.

[64] OpenFlow Switch Specification. Version 1.0. 0 (wire protocol 0x01). *Open Networking Foundation*, 2009.

[65] Nippon. Component-based software defined networking framewor. `https://github.com/faucetsdn/ryu`, 2011-2014. [Online] Accessed on 1st January 2024.

[66] A OpenDaylight. Linux foundation collaborative project. Available online: `http://www.opendaylight.org` (accessed on 18 June 2022), 2013.

[67] David Erickson. The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 13–18, 2013.

[68] Thomas Dietz Thomas. Trema tutorial. 2012.

[69] Project floodlight. `https://github.com/floodlight/floodlight`. [Online] Accessed on 1st January 2024.

[70] Zheng Cai, Alan L Cox, and TS Ng. Maestro: A system for scalable openflow control. Technical report, 2010.

[71] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM computer communication review*, 38(3):105–110, 2008.

[72] Ali Malik, Ruairí de Fréin, and Benjamin Aziz. Rapid restoration techniques for software-defined networks. *Applied Sciences*, 10(10):3411, 2020.

[73] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[74] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[75] Ali Malik Jaber Al-Bdairi. *Fault tolerance enhancement for software defined networks*. PhD thesis, University of Portsmouth, 2019.

[76] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. In-band control, queuing, and failure recovery functionalities for openflow. *IEEE Network*, 30(1):106–112, 2016.

[77] Wolfgang Braun and Michael Menth. Software-defined networking using openflow: Protocols, applications and architectural design choices. *Future Internet*, 6(2):302–336, 2014.

[78] Younghee Park, Hongxin Hu, Xiaohong Yuan, and Hongda Li. Enhancing security education through designing SDN security labs in cloudlab. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 185–190, 2018.

[79] Steven SW Lee, Kuang-Yi Li, Kwan Yee Chan, Guan-Hao Lai, and Yao Chuan Chung. Software-based fast failure recovery for resilient openflow networks. In *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*, pages 194–200. IEEE, 2015.

[80] Reaz Ahmed and Raouf Boutaba. Design considerations for managing wide area software defined networks. *IEEE Communications Magazine*, 52(7):116–123, 2014.

[81] S Chick, P Sánchez, D Ferrin, and D Morrice. How to conduct a successful simulation study. In *Proceedings of the 2003 winter simulation conference*, volume 2003, pages 66–70, 2003.

[82] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[83] Mukta Gupta, Joel Sommers, and Paul Barford. Fast, accurate simulation for sdn prototyping. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 31–36, 2013.

[84] Dominik Klein and Michael Jarschel. An openflow extension for the omnet++ inet framework. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, pages 322–329, 2013.

[85] Vern Paxson and Sally Floyd. Why we don't know how to simulate the internet. In *Proceedings of the 29th conference on Winter simulation*, pages 1037–1044, 1997.

[86] Steven D Elliott et al. Exploring the challenges and opportunities of implementing software-defined networking in a research testbed. 2015.

[87] Shie-Yuan Wang, Chih-Liang Chou, and Chun-Ming Yang. Estinet openflow network simulator and emulator. *IEEE Communications Magazine*, 51(9):110–117, 2013.

[88] 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*, pages 1–6, 2010.

[89] Alessio Botta, Walter de Donato, Alberto Dainotti, Stefano Avallone, and Antonio Pescapé. D-ITG 2.8. 1 Manual. *Computer for Interaction and Communications (COMICS) Group*, pages 3–6, 2013.

[90] Mohammed Ridha Nsaif, Mohammed Falah Abbood, and Abbas Fadhil Mahdi. Detection and prevention algorithm of DDOS attack over the IoT networks. *TEM Journal*, 9(3):899, 2020.

[91] Edsger W Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pages 287–290. 2022.

[92] 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, ICST, Brussels, Belgium, Belgium, 2013. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[93] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, Amin Vahdat, et al. Hedera: dynamic flow scheduling for data center networks. In *Nsdi*, volume 10, pages 89–92. San Jose, USA, 2010.

[94] Ruairí de Fréin. Source separation approach to video quality prediction in computer networks. *IEEE Communications Letters*, 20(7):1333–1336, 2016.

[95] Ruairí de Fréin, Obinna Izima, and Ali Malik. Detecting network state in the presence of varying levels of congestion. In *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, 2021.

[96] Noora Al Khater and Richard E Overill. Network traffic classification techniques and challenges. In *2015 Tenth international conference on digital information management (ICDIM)*, pages 43–48. IEEE, 2015.

[97] Kai-Cheng Chiu, Chien-Chang Liu, and Li-Der Chou. Capc: packet-based network service classifier with convolutional autoencoder. *Ieee Access*, 8:218081–218094, 2020.

[98] Anderson Santos Da Silva, Cristian Cleder Machado, Rodolfo Vebber Bisol, Lisandro Zambenedetti Granville, and Alberto Schaeffer-Filho. Identification and selection of flow features for accurate traffic classification in sdn. In *2015 IEEE 14th International Symposium on Network Computing and Applications*, pages 134–141. IEEE, 2015.

[99] Pedro Amaral, Joao Dinis, Paulo Pinto, Luis Bernardo, Joao Tavares, and Henrique S Mamede. Machine learning in software defined networks: Data collection and traffic classification. In *2016 IEEE 24th International conference on network protocols (ICNP)*, pages 1–5. IEEE, 2016.

[100] Mohammad Reza Parsaei, Mohammad Javad Sobouti, Reza Javidan, et al. Network traffic classification using machine learning techniques over software defined networks. *International Journal of Advanced Computer Science and Applications*, 8(7), 2017.

[101] Ali Malik, Ruairí de Fréin, Mohammed Al-Zeyadi, and Javier Andreu-Perez. Intelligent SDN traffic classification using deep learning: Deep-sdn. In *2020 2nd International Conference on Computer Communication and the Internet (ICCCI)*, pages 184–189. IEEE, 2020.

[102] Yong Wang, Wenlong Ke, and Xiaoling Tao. A feature selection method for large-scale network traffic classification based on spark. *Information*, 7(1):6, 2016.

[103] Ralf Klabunde. Daniel jurafsky/james h. martin, speech and language processing. *Zeitschrift für Sprachwissenschaft*, 21(1):134–135, 2002.

[104] Richard G Baraniuk. Compressive sensing [lecture notes]. *IEEE signal processing magazine*, 24(4):118–121, 2007.

[105] Gary C McDonald. Ridge regression. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(1):93–100, 2009.

[106] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

[107] Xiaolin Huang, Andreas Maier, Joachim Hornegger, and Johan AK Suykens. Indefinite kernels in least squares support vector machines and principal component analysis. *Applied and Computational Harmonic Analysis*, 43(1):162–172, 2017.

[108] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[109] Benoit Steiner, Chris Cummins, Horace He, and Hugh Leather. Value learning for throughput optimization of deep learning workloads. *Proceedings of Machine Learning and Systems*, 3:323–334, 2021.