Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunication and Media Informatics

Towards Comprehensive and Effectual SDN Traffic Engineering Framework for Data Center Networks

Ph.D. Dissertation of

Aymen Hasan Rashid Alawadi

Scientific Supervisor:
Dr. Sándor Molnár

May 19, 2022

Declaration

I, undersigned *Aymen Hasan Rashid Alawadi* hereby declare that this Ph.D. dissertation was made by myself, and I only used the sources given at the end. Every part that was quoted word-for-word, or was taken over with the same content, I noted explicitly by giving the reference to the source.

Aymen Hasan Rashid Alawadi May 19, 2022

Abstract

Data Center Networks (DCNs) act as critical infrastructures for emerging technologies. In general, a DCN involves a multi-rooted tree with various shortest paths of equal length from end to end. The DCN fabric must be maintained and monitored to guarantee high availability and better Quality of Service (QoS). Various Traffic Engineering (TE) methods based on SDN (Software-defined networking) have been introduced to solve flow competitions for network resources. Utilizing such methods in a production DCN is risky in terms of throughput losses because of several factors, such as the adopted mechanism of congestion handling and different DCN flow demands (the amount of expected transferred data per second for each flow and could be small and large TCP (Transmission Control Protocol) flows). However, to the best of our knowledge, there is no comprehensive study on investigating the risk analysis of TE methods.

Hence, in the first part of the thesis, we started by investigating the performance of existing SDN-based TE methods, such as Equal-cost multi-path routing (ECMP), Hedera, and PureSDN. The evaluation relies on the Monte Carlo simulation to estimate the value at risk of the large TCP flows (elephant flow) loss rate. The results confirm that the difference between the

comparative methods in elephant flows throughput risks is insignificant, although the adopted flow scheduling was utterly different.

The second part proposes novel hybrid and deployable load-balancing approaches that guarantee the QoS of multi-rooted DCN traffic in symmetric DCN fat-tree topology. We proposed Sieve and Oddlab as SDN-based TE methods. In Sieve, we investigated proactive and adaptive scheduling for flow scheduling strategy based on available bandwidth. To resolve the potential network congestion, a fraction of elephant flow is rerouted to another path regularly. On the other hand, instead of frequent elephant flow rerouting, we define the adaptive flow scheduling by considering the path's residual bandwidth besides active elephant flows to determine the best paths and avoid significant flow rescheduling. Extensive experiments were conducted on a wide range of traffic patterns with synthetic and realistic workloads to prove the feasibility of Sieve and Oddlab without altering any network component, including hosts or switches. The results indicate that both methods significantly improved bisection bandwidth, link utilization, mice flow FCT (Flow Completion Time), average overall FCT, packet loss, round trip delay, and the elephant flow loss rate risk compared to ECMP, Hedera, and PureSDN, respectively. We demonstrate that Sieve and Oddlab operate in low complexity and computational overhead on the SDN controller. Thus, both TE approaches can be deployed in commercial DCNs at a lower cost.

In the third part, we introduce our faulty detection model in Oddlab. This model utilized the same global information from DCN in the adaptive scheduling to detect the faulty links. The procedure of identifying faulty links relies on correlating two events within the DCN: loaded edge switches and underutilized core links. The adaptive flow scheduling model

will utilize the information of the detected links so that the affected links are avoided. We conducted several experiments on realistic workloads to prove Oddlab efficiency on asymmetric DCN topology. The results show that Oddlab can detect faulty links with accuracy and considerably low complexity.

Acknowledgments

I would like to thank everyone that made this thesis possible. I want to thank Dr. Sándor Molnár, my Ph.D. supervisor, for his support, patience, and encouragement throughout my study years. Also, I'm very thankful to my incredible family: my wife Jinan, my son Mohammed Ridha, and my little sweetie daughter Zahraa; without their love, smiles, and support, nothing would be possible. Now, I'm glad that we will enjoy more time together. I'm humbled and feel a deep sense of gratitude towards my parents and other family members for their genuine support and love. Lastly, I would like to thank the University of Kufa – Iraq, the Tempus Public Foundation (TPF) – Stipendium Hungaricum program, and the Department of Telecommunication and Media Informatics in Budapest University of Technology and Economics- Hungary for supporting my Ph.D. scholarship.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"A Research is to see what everybody else has seen, and to think what nobody else has thought."

Albert Szent-Györgyi (1893-1986)

1.1 Preliminary and Background

Today's network enterprises utilize DCN fabrics to handle highly demanded bandwidth applications. DCN applications such as Hadoop, MapReduce, web search, social networks, and e-commerce rely on thousands of servers to sustain high availability, and scalability [22] [54]. DCN plays a vital role as the communication environment for such applications due to multi-paths provided among all the end-hosts (servers) in the network. On the other hand, DCN applications are extremely demanding, where extensive data is transferred within the network. For instance, DCN applications, such as machine learning, data mining, and data analysis, demand high available

bandwidth, fast response, and high availability [65] [103] [67]. Furthermore, the global demand for data transmission is expanding enormously. Industry reports such as the IDC report predicts that the Datasphere (i. e., the created, captured, or replicated data in the industry infrastructures) will rise from 33 Zettabytes in 2018 to 175 Zettabytes by 2025 [76].

Typically, the applications of DCNs produce two types of flows: mice and elephant flows. Most of the flows (i. e., 80%) are Mice flows, the smallest yet shortest-lived TCP flows in the network, and are more sensitive to communication delay [77]. The most massive and long-lived TCP flows, elephant flows, are more affected by the residual link bandwidth. The number of these flows in DCNs is less than that of mice flows, but they deliver most of DCN bytes [11] [77] [69] [46]. In this thesis, we are required to deal with active elephant flows inside the DCN data plane without prior knowledge about the size and duration of the flow. Thus, in terms of elephant flow in byte count, we adopted the definition of elephant flow size of 50 KB based on the observation for the top 10% of the large flows by Benson et al. in [11]. As for the instantaneous elephant flow detection in DCN, we adopted the flow rate (i. e., above 50 kbps) instead of the accumulative flow size as suggested by Roy et al. in [77].

However, many DCN topologies evolved such as hyperx [5], flattened butterfly [49], and fat-tree [26]. For example, fat-tree DCN topology is designed in a symmetric way to achieve high bisection bandwidth thanks to multi-rooted paths between the end-hosts. Hence, all DCN switches need to be remotely connected to the SDN controller, as depicted in Figure 1.1. In this thesis, we use a $K-4$ fat-tree DCN topology with 16 end-hosts. Traditional traffic steering methods such as Equal Cost Multi-Path (ECMP) [36] are based on local nodes to achieve the network operation. Such local

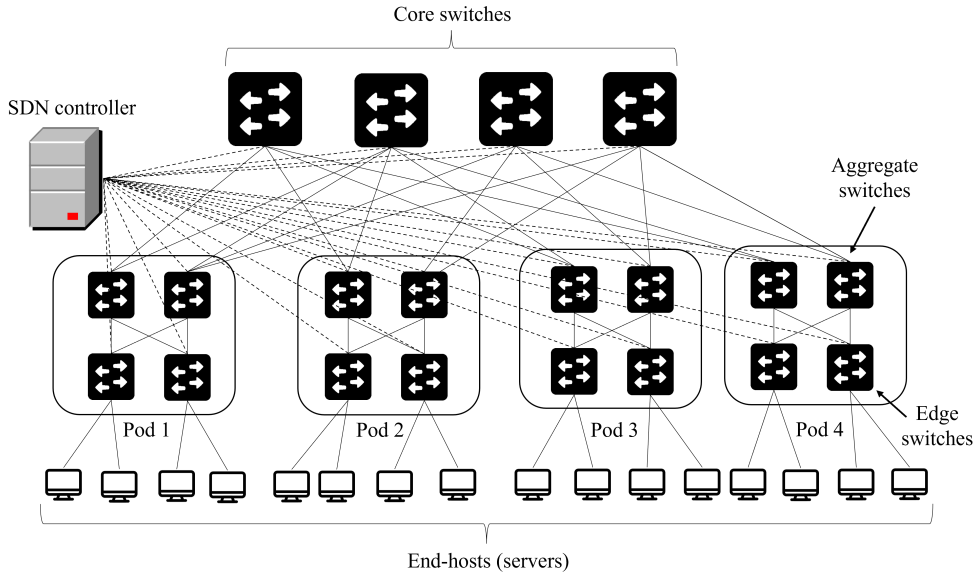


Figure 1.1: $K-4$ fat-tree DCN topology with a central SDN controller.

optimization suffers from local flow collisions that lead to severe congestion, packet loss, and lower network utilization. Therefore, effective traffic scheduling should be applied based on the network's current state to avoid collisions. One of the most critical issues in DCN is the link failures, wherein an average of 40 links are malfunctioning every day in large DCNs [83]. Consequently, it is not easy to maintain the symmetric situation for the duration of the network operation. Essentially, there are two types of link failures, i.e., partial and full failure [29]. The DCN topology turns asymmetric if any one of the failures occurs. A scheme such as ECMP was deployed to hash every flow to a different path to handle traffic congestion in standard DCN operations, considering the complete failure only without watching network stats.

1.1.1 Software-defined networking (SDN)

After SDN was introduced, several solutions and opportunities emerged. The new paradigm brought a new orientation in managing the network traffic where the centralized controller has the property of decoupling the control plane from the data plane for reliable and effective resource handling. Mainly, the SDN concept introduces a new paradigm for network management that includes three planes; application plane, control plane, and data plane (Figure 1.2). In this paradigm, the central controller resides in the control plane and is responsible for network traffic management implemented in the application plane, such as traffic engineering, monitoring, routing, QoS, security, and many other network management applications. The interaction between the control plane and the application plane is delivered via the Application-programming interface (northbound APIs). On the other side, the control plane communicates with the data plane through the OpenFlow southbound APIs (OpenFlow protocol). The OpenFlow [61] protocol is considered an open platform protocol that enables different network and hardware vendors to utilize it in isolate controller traffic from production traffic. One of the main mechanisms of OpenFlow protocol is the ability to write special applications in the switch's flow-table to handle and partition the network's traffic.

The central location of the SDN controller in the network management (Figure 1.2) enables a global view of the network by collecting statistical parameters from the entire network, including information about current flows and port status in the data plane.

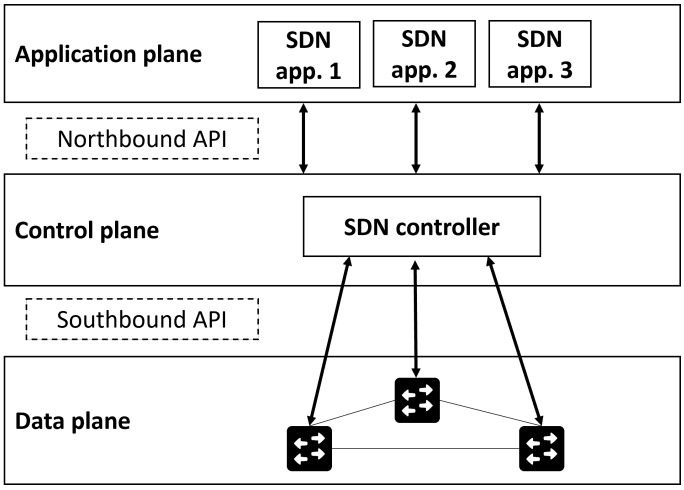


Figure 1.2: SDN (Software-defined networking) architecture.

1.1.2 Fat-tree DCN Topology

In general, fat-tree DCN topology is widely used since it delivers high scalability through the switches' tree structure with high bisection bandwidth. Hence, fat-tree topology has been widely deployed in large production DCN environments such as Facebook DCN fabric [13]. Mainly, the fat-tree topology contains three layers of connected switches: core, aggregate, and edge switches. The DCN end-hosts are directly connected to the edge switch layer. The edge switches are combined with aggregate switches in pods, where each fat-tree DCN contains K pods. Each pod should be connected to $(K/2)^2$ end-hosts, and each switch have $K/2$ ports. The DCN pods are aggregate switches connected to $K/2$ core switches on the upstream side and $K/2$ edge switches on the downstream side. The total number of the end-hosts that fat-tree DCN can support is $K^3/4$. Therefore, the fat-tree achieves high availability between source and destination end-hosts, where there are $(K/2)^2$ equal-cost paths between every pair of them.

In the SDN paradigm, the central controller monitors the ports of each switch in the DCN data plane to effectively handle the traffic flows and make decisions according to the gathered information. Hence, all of the DCN switches need to be remotely connected to the SDN controller, as depicted in Figure 1.1.

1.1.3 SDN Traffic Engineering

SDN paradigm has been proved to deliver efficient Traffic Engineering (TE) solutions compared to the traditional schemes such as Internet Protocol (IP) networks and Multiprotocol Label Switching (MPLS) networks [6]. Ian F Akyildiz et al. in [6], classified the scope of the TE mechanisms applied using the SDN concept, as depicted in Figure 1.3. The main points of the scope are clarified as follows:

(1) **Flow management:** the network flows in SDN concept handled based on entry rules managed by the controller and installed on the data plane (OpenFlow switches). The flow forwarding operation starts after the controller receives the first flow packet (known as *packet_in* request) from the ingress switch. Then, the controller will compute the appropriate path and install the flow entry rules along with the path switches. In this manner, the following packets of the flow or any flow that matches the entry attribute will follow the corresponding entries of the path. Consequently, there is no need to communicate with the controller to handle the flows. However, an overhead could be raised in the control plane and data plane by increasing the number of the managed flows. For instance, the central SDN controllers can handle only a specified number of *packet_in* requests per second. On the

other hand, the Ternary Content-Addressable Memory (TCAM) space limitations of OpenFlow switches affect the flow entries that the controller can handle. Moreover, the operation of installing the flow entry rules takes time. Accordingly, as the network flows increase, the time needed to compute the path and install the entry rules will increase directly. Hence, designing and implementing an effective SDN-based TE method should balance between load balancing and responding latency [6].

(2) **Fault-tolerance:** links and nodes failures are quite often in general production networks. Even the SDN controller is considered the network's central point of failure [7] if exposed to several attacks, such as overwhelming the controller with numerous *packet_in* requests [50] and other forms of attacks such as poisoning network visibility [34]. Therefore, SDN-based methods should continuously detect and recover different network failures to ensure high reliability. One of the solutions came from the *FastFailover* mechanisms introduced in OpenFlow V1.1, which enables the switches to reroute the flow instantly from the failed link without considering the centralized SDN controller. Still, the SDN controller needs to update all the switches with a new path through new flow entry rules. Therefore, the limitation of the switches TCAM should be considered.

(3) **Topology update:** network topology update in SDN refers to update network flow rules due to the planned changes such as modifying network policy rules or unexpected issues such as network congestions handling or failures recovery [6]. The newly added rules might cause further flow delivering delay and thus QoS degrading. Therefore, the

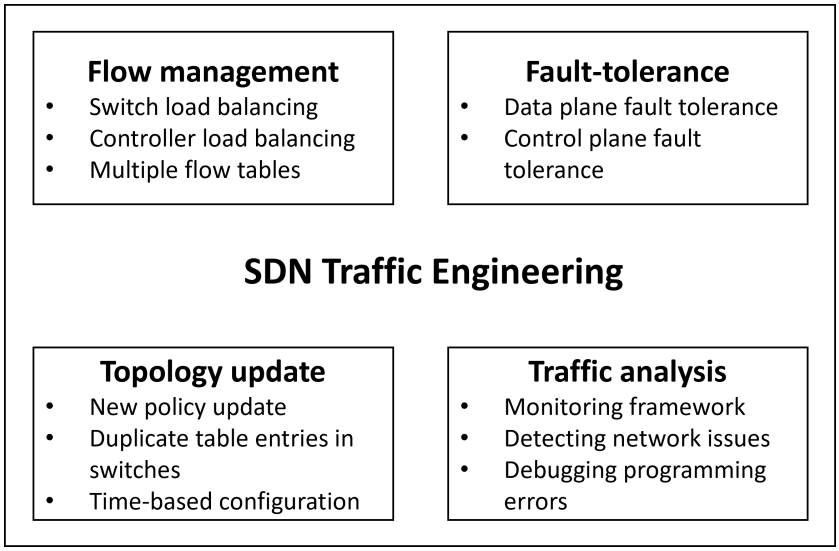


Figure 1.3: SDN traffic engineering classification [6].

process of updating the rules should be performing efficiently without harming other active flows as possible.

(4) **Traffic analysis:** the SDN controller requires different network stats to build flow handling, congestion prediction, and failure detection. Hence, network resource monitoring will include port consumption and flows status in each switch in the data plane. However, increasing the resource monitoring time and scope may produce better flow scheduling, but at the cost of design complexity and controller overwhelming, which lead to response delay. Therefore, the new monitoring mechanism should be aware of the tradeoff between information accuracy and design complexity.

Implementing comprehensive and data computations in real-time are challenging without efficient resource management [22]. SDN plays an essential role in network resource allocation, traffic monitoring, and classification. As stated earlier, the quality of elephant flows is mainly affected by

the available bandwidth of the path. There are many performance evaluation issues associated with the proposed SDN load balancing techniques concerning dealing with the managing of significant data load balancing [27], [75], [51]. The deterministic statistical evaluation approaches are inefficient, especially in a fully stochastic environment. Therefore, in this thesis, the focus will be on investigating the integration of different sources of uncertainty that could affect the overall performance of a load balancing technique on large flows.

In the literature, Sokolov et al. in [87] proposed utilizing OpenFlow protocol statistic information to recognize weak points of the network architecture and predict the potential risks of problems that may occur primarily in the fail-safety task network. The SDN flow table information is considered trivial and inadequate for the applications above the SDN architecture [58]. Therefore, Luo et al. in [58] proposed a context-aware traffic forwarding service for SDN applications assuming a constrained optimization path problem for decision making. The authors suggested two factors, including capacity and cost of a service, that would impact the service composition. They found that cost context can be quantified from other context factors. So, the authors collected the cost and the factors data online to derive their relations by multiple linear regression analysis. The quality of elephant flows is mainly affected by the available bandwidth of the path. Hence, several studies have emerged dealing with load balancing and scheduling issues. Al-Fares et al. in [27] proposed Hedera, a dynamic and central flow scheduling to utilize DCN bisection bandwidth. The authors found the performance of Hedera primarily based on the rate and duration of the flows in the network, but they did not show the future prediction for the blocked elephant flows. Zakia et al. proposed another technique in [105] that relies

on flow priority to find the shortest paths. The authors' evaluated throughput consumption, RTT delay, and packets loss in a fat-tree DCN.

1.2 Research Objectives

In this thesis, we begin by investigating the performance of existing SDN-based TE methods, including ECMP as a standard industrial flow forwarding besides Hedera [27] as a dynamic elephant flow scheduling, and PureSDN [59] as a performance reference for a completely adaptive flow scheduling solution that solely depends on the SDN controller with wildcard forwarding rules. The evaluation relies on predicting the value at risk of the elephant flow loss rate. The studied methods have theoretical and practical benefits. But their effectiveness has not been statistically investigated and analyzed in maintaining the DCN elephant flows. The performance evaluation task motivated us to propose novel DCN SDN-based TE methods that solve the SDN TE challenges discussed in (Section 1.1.3). The main objective is to introduce our solutions based on the OpenFlow SDN concept without altering network components or TCP packets. The thesis is arranged into three main theses.

In the first part of the thesis, we empirically examine the performance and effectiveness of the evaluated TE methods to answer the following questions regarding risk analysis:

(1) What is the predicted loss rate of elephant flows using different TE methods?

(2) What are the risk factors of implementing these methods regarding elephant flow preserving?

Based on the obtained results of Thesis 1, we discover that the wild-card forwarding rule (PureSDN) produces the least elephant flow loss rate compared with the other methods. Besides, we remark that the range of optimization in elephant flow loss reduction is about 18% between the thoroughly hashing-based ECMP and full SDN-based in PureSDN. Hence, developing a flow scheduling method should balance the cost of monitoring and scheduling and the expected elephant flow loss. Therefore, the objective of Thesis 2 is to propose novel approaches based on the SDN paradigm for traffic load balancing and flow scheduling. Thus, we investigate the flow distribution scheme in DCN through predefined flow paths at the DCN edge switches employing ECMP hashing besides the SDN controller. However, our intent in Thesis 2 is to answer the following questions:

(1) How to schedule the flows to improve DCN performance such as mice flow FCT and reduce the elephant flow losing rate.

(2) Does 1:1 sampling between ECMP and the SDN controller necessarily lead to significant packet loss in mice flows and affect the elephant flows throughput due to flow collisions?

(3) Does the additional computation complexity to detect and reschedule elephant flows necessarily lead to a significant delay in the *packet_in* handler?

(4) Can we introduce a deployable, light, and yet effective load balancing technique that works in symmetric and asymmetric DCN topology using the SDN concept without altering network components or TCP packets?

(5) DCN links failure is one of the most critical issues in DCN daily operation [29]. Consequently, it is not easy to maintain the symmetric situation for the duration of the network operation. Therefore, the question is how to schedule the flows so that avoiding the failed links?

The third part is related to faulty links detection in DCN topology. We introduce Oddlab, a novel adaptive scheduling method that considers the healthy paths, available bandwidth, and active elephant flows to find the best paths and avoid significant flow rescheduling. Our objective in Thesis 3 is to propose an algorithm to find and avoid the faulty links without complicated arithmetic operations or executing host or switch altering inside the DCN topology.

1.3 Research Contributions

This thesis proposes novel solutions for TE SDN-based methods. The research solutions are implemented in fat-tree DCN topology, and the results have shown remarkable improvements compared with existing solutions. Furthermore, we have defined our solutions as feasible mathematical models and come up with applicable theorems and algorithms into a production DCN. However, our thesis contributes mainly to the SDN TE presented in Subsection 1.1.3 as follows:

1.3.1 Predicting of Blocking Behavior for Elephant Flows in DCNs Using Monte Carlo Risk Analysis

This thesis empirically designed, implemented, and analyzed a novel quantitative performance evaluation model for existing SDN flow scheduling

and methods used in DCN based on multiple stochastic workloads to predict the elephant flows loss rate Value at Risk. The evaluation estimates the proper probability distribution functions for the proposed risk factors for Hedera, ECMP, and PureSDN. The proposed evaluation model was developed as a Value at Risk analysis model using Monte Carlo simulation. The evaluation included estimating the probability distribution for risk factors based on Anderson-Darling and Kolmogorov Smirnov probability distribution tests. Determining the probability distribution of such TE methods aids in further mathematical analysis of elephant flow handling without the need for additional practical experiments. These risks directly influence the status of DCN applications in terms of flow completion time and throughput. Nevertheless, the TE techniques need to have proper awareness in terms of flow risk analysis instead of accepting the simple average values of the results, especially when the testing samples are not large enough.

1.3.2 Adaptive SDN Load Balancing Framework

In this thesis, we tackled flow management problems (switch load balancing and multiple flow tables) and topology updates of SDN TE. We propose Sieve, in which we leveraged the OpenFlow protocol group table to form a sampling strategy at the edge switches of the DCN topology. The main objective of the flow samplings technique is to reduce the number of installed flow entry rules since a fraction of flows will be handled by proactive paths (ECMP-based), and the SDN controller will manage the rest. In Sieve, we resolve the expected bottlenecks by frequently rescheduling the fraction of elephant flows detected on edge switches to balance the elephant flows among the available paths. Furthermore, we proposed Oddlab (Odds

labels) to avoid the expected bottlenecks forwarding the flows to the least number of active elephant flows. In Oddlab, we maintain the number of installed flow entry rules without tracking and rescheduling the elephant flows. However, extensive experiments were conducted on a wide range of traffic patterns with synthetic and realistic workloads to prove Sieve and Oddlab feasibility without altering any network component (i.e., hosts or switches). The results confirm that Oddlab delivers noticeable improvements in bisection bandwidth, link utilization, packets loss, round trip delay, mice flow FCT, with an average overall reduction in FCT, fewer installed flow entries, with less elephant flow loss rate.

1.3.3 DCN Faulty Links Detection Model Based on Temporal Correlation

In this thesis, we present our contribution to the fault-tolerance and traffic analysis challenges in SDN TE. The proposed procedure correlates two events within the DCN: the loaded edge switches and underutilized core links. Accordingly, the most crucial finding of Oddlab is that in leveraging the global knowledge of the DCN switches statistics on a single and central SDN controller to detect the faulty links and deliver promising results in the asymmetric topologies. Several experiments were conducted on different bandwidth degradation with realistic workloads to prove Oddlab feasibility without altering any network component (i.e., hosts or switches). The results confirmed that Oddlab delivers improvements in asymmetric topology with an average overall reduction in FCT. Oddlab is able to detect, reschedule the affected elephant flows with severe bandwidth degra-

dation in the core links, and avoid the defective paths in the adaptive flow scheduling model.

1.4 Traffic Traces and Communication Patterns used in the Research

We conduct extensive experiments on different traffic patterns with synthetic and realistic workloads of productive DCN (web search and cache jobs [77]). Both workloads are collected and analyzed in the Intra-Datacenter network. In the web search dataset, the distribution ratios will be as follows:

- 0 - 1 kB: 33%
- 1 - 10 kB: 60%
- 10 - 300 kB: 7%

As for the cache workload, the data size distribution is as follows:

- 0 - 0.1 kB: 18%
- 0.1 - 10 kB: 38%
- 10 - 300 kB: 7%
- 1 - 10 MB: 10%

The traffic scenarios are conducted in symmetric and asymmetric DCN $K-4$ fat-tree topology. To implement the test in the mininet environment, each end-host in DCN started to transmit samples of the workloads (16

flows for each dataset) to another end-host based on the same traffic patterns described earlier. The flows are initiated based on the Poisson process with a given mean value to simulate realistic traffic between the DCN end-hosts. Note that the Poisson process is leveraged to simulate discrete and independent events (the event's occurrence does not affect another one), where the mean time between the events is known, but the exact arrival time is unknown. Thus, to simulate empirical workloads that could be produced into a realistic DCN environment, Alizadeh et al. [10] suggested that the flows arrive according to a Poisson process. However, this process is widely employed in literature to model the flows' arrival between the DCN end-hosts, a presented in the following works [27], [82], [17], [47], [98], [53], [55], and [9].

In our traffic emulation, the events are not coinciding. Nevertheless, they are initiated throughout the entire interval of the experiment, where 512 flows were transmitted through 126 sec. for each traffic pattern.

We leverage the same communication pattern applied to evaluate Hedera performance [27], which was introduced in [26]. The generated communication pattern consists of random and staggered probabilities patterns. The generated communication pattern consists of random and staggered probabilities patterns according to the following details:

(1) `Random`: every end-host transmits traffic to any other end-host in the DCN with uniform probability.

(2) `Staggered probability` ($Edge\_p$, Pod_p): every end-host transmits traffic to another host in the same edge with probability ($Edge\_p$), to the same pod with probability (Pod_p) and with other pods in the DCN with probability ($1 - Edge\_p - Pod\_p$).

1.5 Organization of the Thesis

The organization of this thesis is as follows. In Chapter 2, we present an introduction to risk analysis, flow scheduling risks, our objectives in this area, including an overview of some related studies, and our research methodology followed by results and discussion. In Chapter 3, we introduce our research motivation, including a literature review on TE methods, followed by the design concepts of the adaptive flow scheduling of Sieve and Oddlab, including experiments results and discussion. In Chapter 4, we discuss some related studies in faulty links detection in DCN followed by analyzing, implementing, and evaluating the Oddlab faulty link detection experimental results, respectively. Finally, the conclusions and future research directions are described in Chapter 5.

Chapter 2

DCN Flow Scheduling Risk Analysis

"The most sophisticated risk analysis methods used in an organization are often applied to low-level operational risks, whereas the biggest risks use softer methods or none at all."

Douglas W. Hubbard

2.1 Introduction

In this theses, we propose novel solutions related to the SDN TE methods. The achieved results from the proposed performance evaluation model led us to present new flow scheduling solutions using SDN controller and OpenFlow protocol in multi-rooted DCN concerning Flow Management, Fault-tolerant, Topology update, and Traffic monitoring.

The proposed model is based on stochastic DCN traffic behavior to predict the elephant flow loss rate by applying Monte Carlo risk analysis approach on different well-known TE methods with multiple traffic patterns. The general procedure for the proposed performance evaluation includes estimating the risk analysis for the investigated SDN TE methods based on the Value at Risk (VaR) technique that relies on the Monte Carlo simulation process. Mainly, elephant flows are affected by the amount of available bandwidth granted by the TE method on the DCN path. As we explained earlier, this type of flow carries most of the traffic within the DCN, so elephant flows are usually affected by the methods applied to steer the flows. As a result, the adopted flow scheduling mechanism will inevitably affect the required service performance that DCN promise to provide in the Service- Level Agreement (SLA).

This thesis contains two sub-theses. In the first part of Thesis 1, we introduce the required analysis to estimate the probability distribution (uncertainty) of the path available bandwidth, including a margin of bandwidth measurement error. The second part includes the Monte Carlo simulation to generate future prediction behavior of the elephant flow loss rate with the Value at Risk analysis to the obtained results.

2.1.1 Risk Analysis

In our proposed risk analysis model, we adopted the risk definition introduced by Douglas W Hubbard in [41] (Definition 1):

Definition 1. (Risk) *a measure of uncertainty of possible significant loss (i. e., undesirable outcome).*

The term of the *"quantified"* uncertainty has also defined as follows [41]:

Definition 2. (Uncertainty) *the lack of a particular possible value or answer to an outcome.*

However, these terms have been widely applied in statistics, operation research, economics, public health, etc. In this research, the objective is to transfer the performance evaluation of the TE methods into risk analysis concept to reveal the answer to the following question:

Question 1. *To what extent different TE methods can maintain the elephant flow throughput in the production DCN lifespan?*

To answer this question, we introduce our definitions as follows:

Definition 3. (TE risk) *a measure of uncertainty for elephant flow throughput losses.*

Definition 4. (TE loss) *a measure of exact significant loss of elephant flow throughput.*

However, different TE methods provide different flow scheduling handling. Hence, different available bandwidth values will be produced for each flow. Mainly, elephant flows throughput is affected by the amount of the available bandwidth preserved on the path. Therefore, we came up with the following question:

Question 2. *What is the range (probability) of the available bandwidth produced by each TE method to the elephant flows?*

To answer this question, we conducted extensive experiments on different traffic patterns with synthetic (`Iperf`) and realistic workloads to measure the available bandwidth asserted to the elephant flow by each TE method. So, our definition for the available bandwidth measurement will be as follows:

Definition 5. (TE available bandwidth) *a measure of the probability distribution for available bandwidth produced by the flow scheduling mechanism.*

Our proposed definitions can be formed into a mathematical equation to compute the TE method risk and estimate the uncertainty value of the significant elephant flow losses.

2.1.2 Flow Scheduling Risks

In compliance with the risk definition provided by Douglas W Hubbard in [41] (Definition 1), flow scheduling methods would be risky only if some of its outcomes produce a significant loss. This section will explain the flow scheduling risk factors impacting the elephant flow throughput.

Firstly, we will consider our illustration for the flow scheduling methods regardless of the DCN topology variables, such as type of the DCN topology, link capacity, failures, noise, interference, etc. Secondly, our analysis will assist the person in charge of investing in choosing the appropriate flow scheduling methods for a production DCN.

However, many causes contribute to elephant flow loss, which can be summarized as follows:

(1) **Traffic burstiness phenomena:** In general, data traffic in DCN is proven to be bursty [12] [77]. Hence it cannot be predicted in advance without the aid of other devices such as sFlow [90].

(2) **Elephant flows uncertainty:** Elephant flows sizes, demands, arrival time, or even their numbers are not known. Therefore, any flow scheduling mechanism will attempt to avoid elephant flow collisions heuristically. Then, losses are highly expected.

(3) **How to detect elephant flows inside DCN?** Different flow scheduling methods proposed different mechanisms, such as the link capacity threshold adopted by Hedera [27], and consumption rate at the end-hosts in Mahout [19], etc. These methods have pros and cons in the rate of flow detection speed, and therefore, the response speed for rescheduling decisions is delayed.

(4) **Statistic gathering, computation cost, and topology update:** Different SDN methods apply various techniques to gather DCN information by using various information with different polling intervals. On the other hand, the computation cost and complexity to handle the gathered information and decide the best path for the *packet_in* requests cause significant delays in handling numerous flows, including the elephant flows. The SDN controller requires time for topology update, including installing or modifying the computed paths' flow entry rules all over the DCN switches from end to end. Accordingly, uncertainty and losses are present and highly expected to occur.

(5) **Flow congestion handling:** Traffic congestion is highly anticipated to happen in DCN. Essentially, to handle congestions inside the DCN fabrics, many SDN TE methods tend to reschedule elephant flows to other available paths. Once again, all the needed steps to handle the flows will be repeated to each elephant flow, including statistic gathering, best path computation, and topology update.

Consequently, in this research, the focus will be on investigating the integration of different sources of uncertainty that would affect the overall performance of the SDN-based TE methods on the expected performance of elephant flows.

2.1.3 Research Objectives

This work introduces a stochastic performance evaluation model for estimating the loss rate and risk analysis of the elephant flows produced under different SDN-based TE with fat-tree topology.

In this research, fat-tree topology is used in constructing the primary network environment; since it is considered one of the essential topologies in building efficient, scalable, and cost-effective DCNs. A $k-4$ fat-tree DCN interconnects topology tested in a mininet environment with fixed links capacity from host to switch and switch to switch links with 10 Mbps each (Figure 1.1). However, the research objectives are:

(1) Incorporating the different sources of uncertainty that affect the overall performance of the load balancing, flow scheduling and flow congestion control algorithms.

(2) Finding a proper probability distribution function for the elephant flow available bandwidth measured by each TE method, including a margin of bandwidth measurement error.

(3) Since the problem factors follow a probabilistic scheme, we derive our problem in Monte Carlo simulation to estimate the loss uncertainty by generating the predicted losing samples of the TE methods.

(4) We examine the achieved results through qualitative analysis with Value at Risk (VaR) and statistics to estimate the loss uncertainty and provide the required risk analysis model.

2.2 Literature Review

Empirical studies can be achieved either by qualitative or quantitative methods for gathering and analyzing data. The quantitative approach includes collecting numerical data and analyzing it using statistical methods. In contrast, the qualitative approach mainly collects the data in text, image, or sounds and analyzes it using imprecise measurements [86]. In SDN networks, OpenFlow protocol provides many statistical and numerical information about the monitored network regarding the flows and packets passed through any monitored port in the flow table. Sokolov et al. in [87] have suggested using this information to identify weak points of the network architecture and predict the potential risks of problems, especially in fail-safety tasks in the network. The SDN flow table information is considered trivial and inadequate for the applications above the SDN architecture [58]. Therefore, Luo et al. in [58] proposed a context-aware traffic forwarding service for SDN applications assuming a constrained optimization path problem for decision making. The authors suggested two factors (capacity and cost of service and) that would impact the service composition, and they found that cost context can be quantified from other context factors. So, the authors collected the cost and factors data online to derive their relation by multiple linear regression analysis. As for elephant flow management, several studies have emerged dealing with load balancing and scheduling issues such as [94], [75], and [51]. Long et al. proposed LABERIO [57], based on the real-time bandwidth utilization rate and max-min policy for path selection. The deterministic statistical evaluation approaches are inefficient, especially in a fully stochastic environment such as a large DCN and elephant flow traffic balancing. Liu et al. [58] introduced a frame-

work to allow adaptive multi-path routing of elephant flows in DCNs under changing load conditions. This solution employs a NOX controller, which negatively affects the performance. Similar to Mahout [19], it detected elephant flows at end-hosts. Still, it monitors TCP socket buffer at end-host to mark flows exceeding a predefined threshold so that elephant flows are forwarded based on a weighted multi-path routing algorithm that installs better paths in switches. Correspondingly, Hedera delivers mice flows based on ECMP by default. However, it employs link load as the only metric for rerouting decisions.

Similarly, You-Chiun Wang and Siang-Yu You applied the group features of OpenFlow in [101] to propose a framework for managing the routes in DCNs by checking link loads so that the framework distributes flows among different paths to balance the loads. This framework provides no distinguishing between elephant and mice flows. Still, when the congestion occurs on a link, the framework selects a backup flow with the most considerable traffic demand, which means in practice, most probably it will be an elephant flow. Nevertheless, it does not provide any measurements about the impact on mice flows. Wang et al. in [97] presented TSACO, which detects elephant flows by OpenFlow and sFlow, then forwards them according to an adaptive multi-path algorithm that handles mice flows differently. TSACO computes the available bandwidth and delay of paths and splits an elephant flow over multiple paths with considerably enough free bandwidth to balance the load. In contrast, it sends mice flows on the remaining computed flows whose delay characteristics are suitable. As a result, TSACO provides better throughput for elephant flows and shorter delay for mice flows than ECMP and weight ECMP. Al-Fares et al. in [27] utilized Monte Carlo simulation to investigate the effect of the hash-based ECMP

method on the bisection bandwidth. The results show that the bisection bandwidth was affected significantly (by an average of 60.8% of the total bandwidth) when increasing the number of flows per host. The authors revealed their intuition without further details about the assumed Monte Carlo simulation factors and adopted equation.

In terms of elephant flow prediction, Bezerra et al. in [13] analyzed the actual DCN dataset from Facebook DCN fabric to model an elephant flow detection framework on a short-term basis. The proposed method relies on FARIMA and a Recurrent Neural Network and it has been evaluated based on descriptive statistics and inferences of the flows. Different versions have appeared to OpenFlow protocol, and each one comes with specific optimizations and updates. Leonardo C Costa et al. in [18] introduced a performance evaluation framework to measure the performance gain between OpenFlow version 1.0 and version 1.3 on software and hardware switches. The study concluded that despite OpenFlow 1.3 presents new features such as multi-table and increases the number of matches in the tables, researchers should be aware that most of these features are still limited in commercial hardware switches. However, increasing the number of matches would affect the final performance significantly.

To the best of our knowledge, in literature, the concept of risk analysis has not been investigated to determine the significant elephant flow losses produced from the static and adaptive TE methods, not to mention the uncertainty level of the loss and the factors affecting it.

2.3 Research Methodology

Our proposed framework for predicting the elephant flow loss rate combines the uncertainty of the available bandwidth of different TE methods with various elephant flows' demands and durations. Figure 2.1 represents the flowchart and the required steps conducted to achieve the prediction framework. The proposed model consists of selecting the appropriate SDN TE methods to investigate and evaluate the uncertainty behaviors of the fat-tree DCN in balancing elephant flow. Then, Anderson-Darling (AD) and Kolmogorov Smirnov statistic (K-S) hypothesis testing methods were applied upon the obtained uncertainty samples to get the appropriate probability distribution function for each of them. Next, Monte Carlo simulation was utilized along with Value at Risk (VaR) analysis to predict the significant losses for the tested elephant flows resulting upon using the TE methods in a fat-tree topology.

As mentioned beforehand, various TE methods will be tested to examine the different flow scheduling handling on elephant flow predicted loss. The tested methods are based on the switch load-balancing techniques, including ECMP, Hedera, and PureSDN. The hash-based ECMP method [36] only scatter the flows among the available paths without considering the congestions, Hedera [27] only reschedule elephant flows when reaching a certain threshold of capacity. Finally, PureSDN [59], an entirely dynamic flow scheduling method based on SDN controller, to find the shortest and best available bandwidth for every network flow.

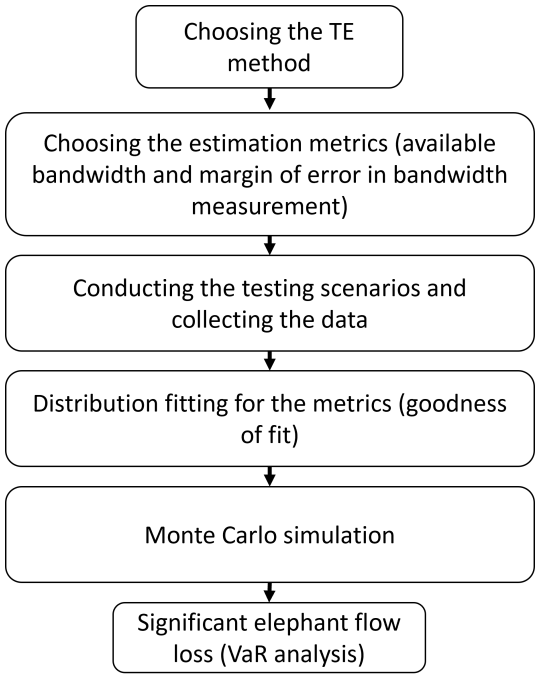


Figure 2.1: Flowchart of the elephant flow loss prediction framework.

2.3.1 Collecting and Normalizing the Uncertainty Data

We conducted extensive experiments on different traffic patterns with synthetic and realistic workloads of productive DCN such as web search and cache jobs [77] workloads. The generated communication pattern consists of random and staggered probabilities patterns. To implement the test in the mininet environment, each end-host in DCN started to transmit samples of the workloads (16 flows for cache jobs dataset and 15 for web search) to another end-host (excluding hosts 1 and 16, which will be used as measurement hosts) based on the same traffic patterns described earlier. The flows are initiated based on the Poisson process with a given mean value to simulate realistic traffic between the DCN end-hosts. Accordingly, 465 flows were transmitted through 62 sec. as the traffic simulation time for

each traffic pattern. To measure the obtained available bandwidth on each test, we set an `Iperf` TCP flow between host 1 and host 16 lasts for 20 sec. in the DCN. The reason behind choosing these end-hosts is to expose the elephant flow to pass through the total bisection bandwidth of the DCN topology. Then, the measured bandwidth of each test will be considered as the available bandwidth offered on the full bisection bandwidth by each tested TE method. Throughout our tests, the amount of traffic passing through the full bisection bandwidth varies according to the adopted traffic patterns (random and staggered). Hence, the measured available bandwidth will be highly be affected. This scenario, however, is similar to the traffic of applications such as MapReduce and Hadoop, which requires extensive bisection bandwidth to maintain their all-to-all shuffle communication [94]. We formed five traffic patterns to generate more samples including: `random`, `stag0.1_0.2`, `stag0.2_0.2`, `stag0.3_0.3`, `stag0.4_0.3`, and `stag0.5_0.3`. Each traffic pattern has been replicated ten independent times for each TE method. To illustrate, the `stag0.4_0.3` implies that 40% of choosing hosts will be in the same edge switch ($Edge\_p$) and 30% will be inside the same pod (Pod_p), and the rest will be from other pods over the full discretion bandwidth in a percentage of 30% ($100 - Edge\_p - Pod\_p$). Figure 2.2 depicts the illustration of `stag0.4_0.3` traffic pattern example in fat-tree DCN topology.

To estimate the probability distribution of the risk uncertainties, we will utilize the Goodness of fit procedure. But before that, specific steps will be conducted upon the obtained data as follows:

(1) We normalize all obtained samples of the available bandwidth by a standard scale to adjust the discrete probability distribution properly.

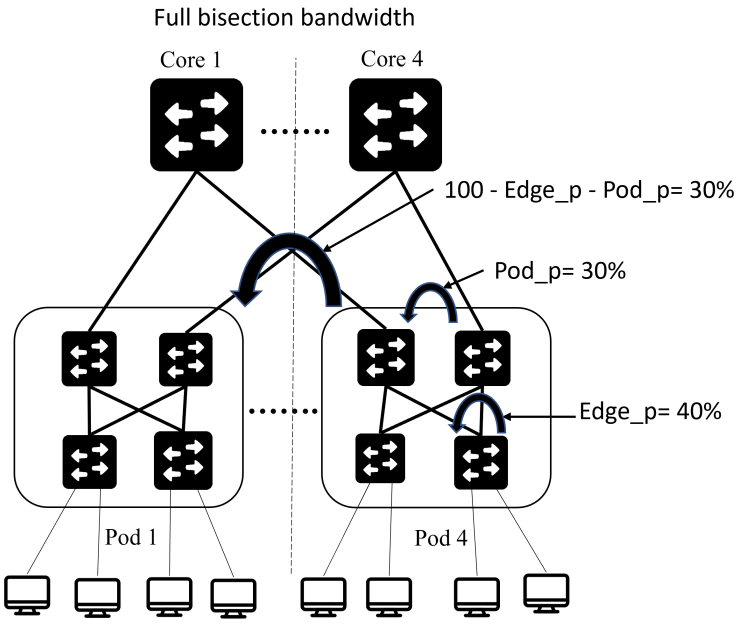


Figure 2.2: The example of `stag0.4_0.3` traffic pattern is illustrated in fat-tree DCN topology.

(2) To determine the margin of bandwidth measurement error, we use the arithmetic samples standard deviation (σ) of the available bandwidth samples mean ($\bar{x}$) for each traffic pattern compared with all available bandwidth values x_{ja}, with ja as an index for each value. At the same time, n is the number of measured samples. Note that we consider the maximum value of the calculated standard deviation since it will indicate the possible upper and lower bounds for the worst-case evaluation as shown in Equation 2.1.

$$\sigma = \sqrt{\frac{1}{n}\sum_{ja}^{n}(x_{ja} - \bar{x})^2} \tag{2.1}$$

2.3.2 Goodness of Fit

The goodness of fit test defined the appropriate probability distribution functions for the available bandwidth besides the margin of bandwidth measurement error. For this sake, we utilized EasyFit professional [23], a specialized statistical tool to test the collected data. K. Schittkowski introduced EasyFit as a software system for data distribution fitting in dynamic systems in [81]. In literature, EasyFit has been widely used to examine the probability distribution for different data types in various fields, such as construction management, finances, trading, biometeorology, operations research, climatology, and transportation management, as appeared in the following works: [24], [85], [89], [40], [60], [39], [38], [93], [70], and [2].

However, since the collected data is in the discrete domain, we chose to use the Kolmogorov Smirnov statistic test (KS) as a hypothesis test to assess the distribution of the data [1]. KS test is a non-parametric test mainly used to compare the distance between the empirical data samples and a specific class of well-known reference probability distributions, as shown in Equation 2.2.

$$D_n = \sup_x |F_n(x) - F(x)| \tag{2.2}$$

Where $\sup_x$ represents the supremum (i. e., least upper bound [78]) distances between the $F_n(x)$, the cumulative distribution function of the observed samples, and reference distribution functions $F(x)$ of an ordered data.

A null hypothesis testing has been performed to accomplish the KS testing using two verifying levels ($H0$ and $H1$). The first value ($H0$) is identified when the tested data specify the distribution, and $H1$ is recognized when

the data does not follow the distribution. To achieve the desired distribution, KS assumes a significance level (α (0.01, 0.0.2, 0.05, etc.)) and compares the tested statistics (D_n) with some of the critical values of the well-known distribution. The hypothesis of the measured distribution will be rejected if the value of (D_n) exceeds the critical value at a significant level.

P-value based on the KS test helps to identify the level when the null hypothesis is rejected. This value indicates a threshold for the significant level ($H0$) to accept all values less than the P-value. For instance, when the P-value = 0.025, the null hypothesis will take all the significance levels less than the P-value, i.e., 0.01, 0.02, 0.05, and reject the higher levels [74].

For more reliable testing in obtaining the probability distribution results, we utilized Anderson-Darling (AD) test, a hypothesis testing to evaluate the distribution of the collected data. Anderson-Darling is defined as (AD^2) as shown in Equations 2.3 and 2.4.

$$AD^2 = -nd - SA\ , \tag{2.3}$$

Where SA is the summation part:

$$SA = \sum_{ia=1}^{nd} \frac{2ia-1}{nd} \left[\ln(Fu(Y_{ia})) + \ln\left(1 - Fu(Y_{nd+1-ia})\right)\right]. \tag{2.4}$$

Where nd is the sample size, ia is the samples index in ascending order, Fu is the cumulative distribution function of the compared distributions, and Y_ia is the ordered data. The null hypothesis testing was performed where $H0$ was identified when the tested data specified the distribution, and $H1$ was identified when the data did not follow the distribution. To come up with the desired distribution, AD assumes significance level (α

(0.01, 0.02, 0.05, etc.)) and compares the test statistics AD^2 with some of the critical values of the most widely used distribution. The hypothesis of the tested distribution will be rejected if AD^2 is greater than the critical value at a significant level.

2.3.3 Probability Distribution Results

Table 2.1 shows the results of conducting KS null hypothesis testing on the throughput measurements of the algorithms. The throughput of ECMP, Hedera, and PureSDN followed the Geometric distribution (GD) based on P-value, and the acceptable critical value was (α= 0.02). However, GD is a discrete probability distribution representing the probability of the success number of independent trials, i.e., Bernoulli trials [74].

Furthermore, the samples of the available bandwidth have been tested under AD as shown in Table 2.2. The results confirm that all samples follow the Geometric distribution based on the values of AD statistic (AD^2) and the obtained critical values estimated on the AD critical value (α = 0.02).

TE method	KS critical values	P-Value	Distribution
ECMP	0.19267	0.16057	Geometric
Hedera	0.19267	0.03506	Geometric
PureSDN	0.19267	0.02187	Geometric

Table 2.1: KS test values for the available bandwidth samples.

TE method	AD critical values	Statistic (AD^2)	Distribution
ECMP	3.2892	1.6337	Geometric
Hedera	3.2892	2.7018	Geometric
PureSDN	3.2892	2.7171	Geometric

Table 2.2: AD test values for the available bandwidth samples.

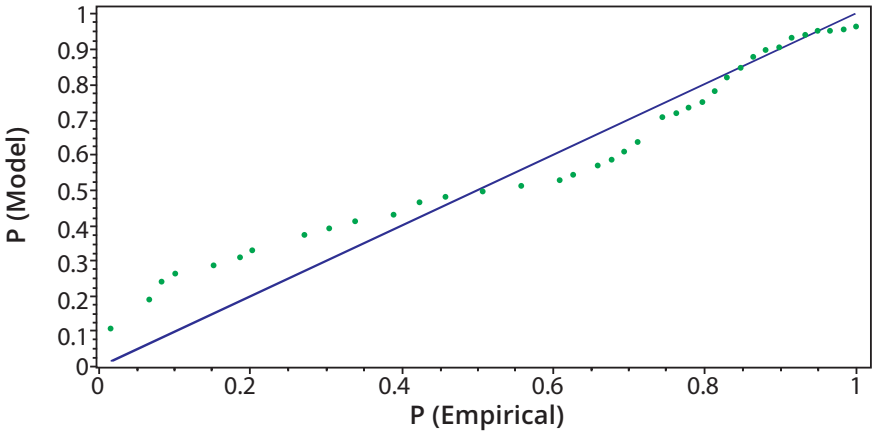


Figure 2.3: P-P plot of the Hedera available bandwidth value distribution fitting with the geometric distribution.

We applied probability mass function of GD to generate the samples of the available bandwidth (AV) with a GD (i. e., $AV \backsim GD(pv)$) for Monte Carlo simulation model where the probability value (pv) of Hedera = 0.03155, ECMP = 0.03861, and PureSDN = 0.02281, respectively.

Figures 2.3, 2.4, and 2.5 show the P–P plots (probability–probability) that prove how the empirical values of the available bandwidth of the studied TE method are close to the data set of the geometric distribution.

As for the risk factor (margin of bandwidth measurement error), the testing showed that the margin of error for the methods follows the simple Discrete Uniform distribution (D-U).

Accordingly, we generate random samples of the error factor (ER) for the tested TE methods within a finite set, where every value has an equal probability in the interval [a_r, b_r], where a_r and b_r are the maximum and minimum values of the samples, respectively, and $a\_r \leqslant ER \leqslant b\_r$. In Monte Carlo simulation model, we utilized the following normalized values as ap-

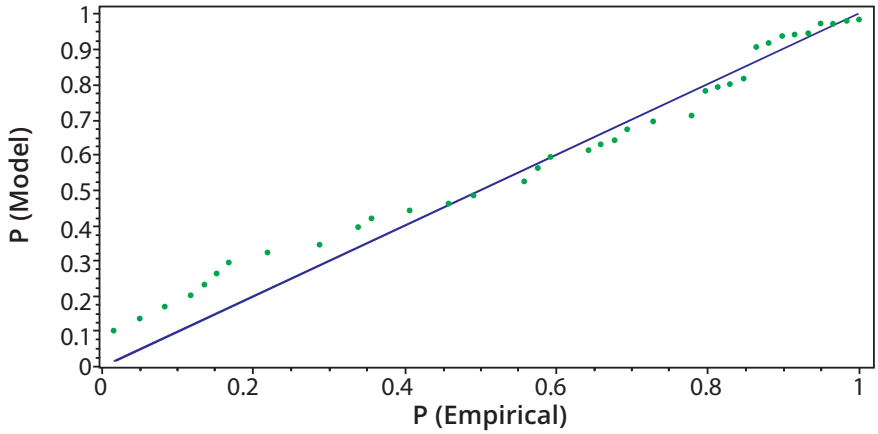


Figure 2.4: P-P plot of the ECMP available bandwidth value distribution fitting with the geometric distribution.

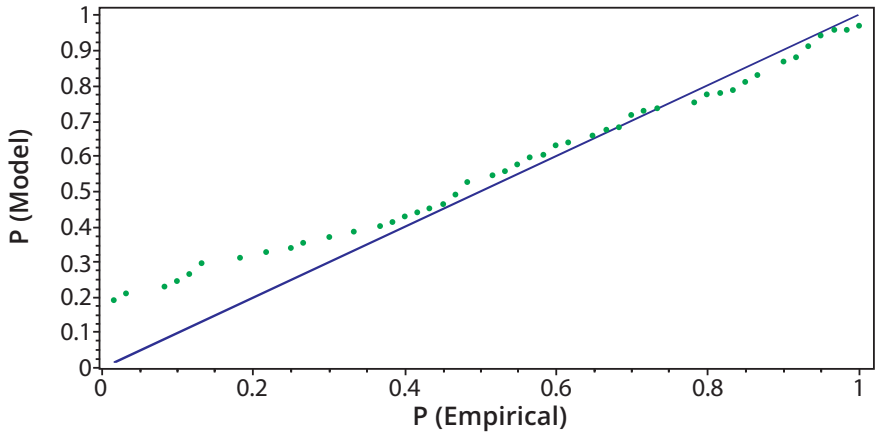


Figure 2.5: P-P plot of the PureSDN available bandwidth value distribution fitting with the geometric distribution.

peared empirically, ECMP = [10, 34], Hedera = [15, 37], and PureSDN = [27, 42].

However, in addition to using the obtained probability distribution of the elephant flow in the Monte Carlo simulation, finding such distributions allows for further mathematical analysis of elephant flow handling without the need for additional experiments.

2.3.4 Monte Carlo Simulation

Back to our definition of elephant flow loss uncertainty (Definition 3 TE risk), we utilized the Monte Carlo simulation strategy to simulate the uncertainty of the elephant flow loss rate for each TE method based on the obtained empirical factor risk (i. e., constrains). The Monte Carlo approach is a technique used to reproduce the stochastic behavior or assess a set of uncertainty inputs of a deterministic model. Typically, it is impossible to predict and determine all possible outcomes of a black box system [95] [41]. We utilized the Monte Carlo simulation process to generate multiple predicted scenarios by estimating the probability distribution of the stochastic input parameters. The simulation process recurred hundreds or thousands of times to produce possible scenarios or solutions with different probabilities. Hence, we leveraged the simulation to predict the elephant flow loss uncertainty for each TE method and estimate the significant loss for the flows by calculating the value at risk. For this sake, we applied the generated samples of the available bandwidth beside the estimated error factors as the Monte Carlo simulation model inputs. Besides, our fundamental equation (Equation 2.5) assumes various elephant flow demands and durations for more realistic results.

$$LO = Du \times (D\_el - (AV + ER)) \tag{2.5}$$

Where LO is the predicted loss uncertainty, Du is the different durations of the evaluated elephant flows, D_el is the elephant flow demands, as shown in Table 2.3, Av is the available bandwidth, and ER is the error factor variable.

Elephant flow	**Demand** D_el **(MBps)**	**Duration** Du **(sec.)**
Large	1.25	100
Normal 1	0.75	85
Normal 2	0.5	65
Small	0.12	45

Table 2.3: Elephant flows evaluation parameters.

Note that the evaluation parameter (D_el) is suggested based on how TE methods detect and reschedule elephant flows. For instance, Hedera reschedules elephant flows when reaching 10% of the total link capacity. Thus, we use the most considerable elephant flow demand with 1.25 MBps, which would be transferred at a rate of more than 10% of the total link capacity (10 Mbps). As for the duration parameter Du, it has been suggested to simulate different flow durations the suggested elephant flow demands at a maximum of 100 sec. [77].

However, the empirical analysis is verified within one million Monte Carlo simulation realizations. Thereafter, the simulation results will provide the whole estimation for the tested data, as shown in Table 2.4. Our simulation model generates two types of samples (i. e., passed and lost samples) based on the simulated parameters and the risk factors. Our analysis considers the lost samples to estimate the uncertainty value of the elephant flow blockage rate measured in MBps, which describes the network

losing rate in the DCN life span. However, the ECMP method achieved the worst loss probability (62.83%) since it does not provide any kind of handling for elephant flows scheduling; this increases the chance of flows collisions. In Hedera, however, the elephant flow detection and rerouting process decreased losses to some extent by 56.76% only. In the fully SDN method (PureSDN), the loss probability becomes much better by 44.43%. Since the measure elephant flow will be guaranteed the best available bandwidth path. But at the expense of the mice flows, in the case of the wildcard rule, or at the expense of increasing the pressure on the SDN controller to handle each flow individually.

The testbed environment includes a $K-4$ fat-tree DCN topology ruining in mininet as a real-time network emulator with the OpenFlow protocol 1.3. The topology has 16 hosts connected to 24 OpenFlow switches with a link capacity set to 10 Mbps. The DCN deployment has been performed into a commodity PC with an Intel Core i5-8400 2.80 GHz CPU, 16 GB RAM running Ubuntu 16.04. The Monte Carlo simulation has been implemented as an *R* script and runs into a Windows 10 PC with an Intel Core i7-5500U 2.40 GHz CPU and 12 GB RAM.

TE method	**Elephant flow loss uncertainty**
Hedera	56.76%
ECMP	62.83%
PureSDN	44.43%

Table 2.4: Distribution statistics of the tested methods.

2.3.5 Distribution Shape Analysis

To perform the Value at Risk (VaR) analysis for the obtained results, we present the histograms of elephant flow loss uncertainty for each TE method

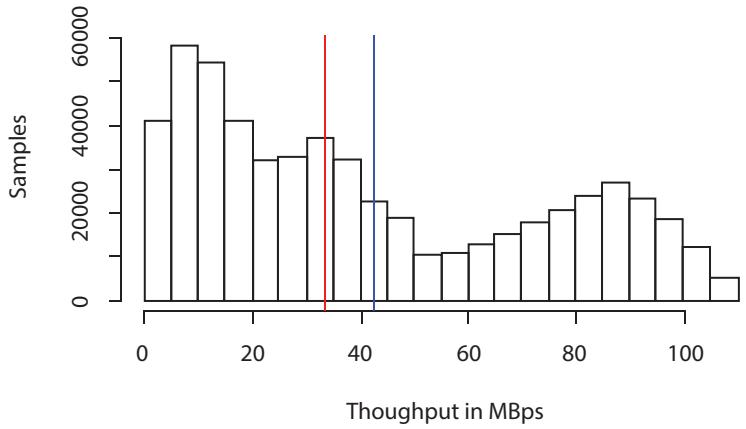


Figure 2.6: Histogram of Monte Carlo simulation for Hedera elephant flow loss rate.

to figure out the variations of the values. The analysis outcome of the model is represented by histograms (Figures 2.6, 2.7, and 2.8) of the predicted elephant flow loss uncertainty produced from employing each TE method.

The current histograms do not follow a particular type of known probability distribution. Still, we can indicate that they have a heavy left-hand tail and unsteady proceed to the long right-hand tail. Common distribution shape measurements were calculated for a better understanding of the loss rate behavior, such as skewness and kurtosis, as shown in Table 2.5.

TE method	**Mean**	**Median**	**Skewness**	**Kurtosis**
Hedera	42.16	33.15	0.5026271	-1.130097
ECMP	44.95	35.70	0.4955835	-1.144383
PureSDN	37.93	29.75	0.4590237	-1.191083

Table 2.5: Descriptive statistics of the tested methods histograms.

The calculated mean values (blue line) precede the median values (red line); the skewness values indicate that all methods follow positive skewness and are right-skewed. In this case, the right-hand tail of the histograms

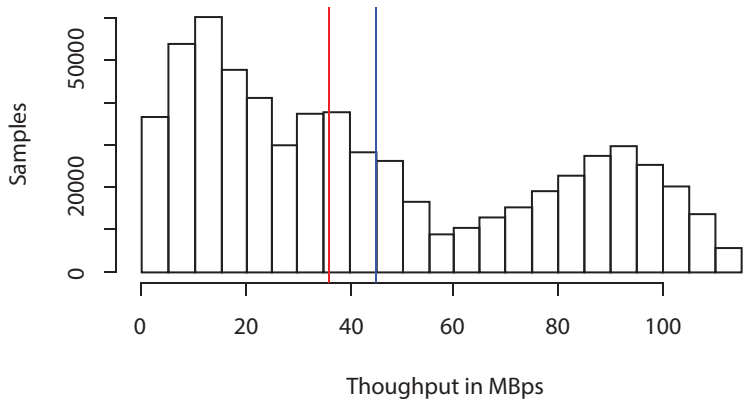


Figure 2.7: Histogram of Monte Carlo simulation for ECMP elephant flow loss rate.

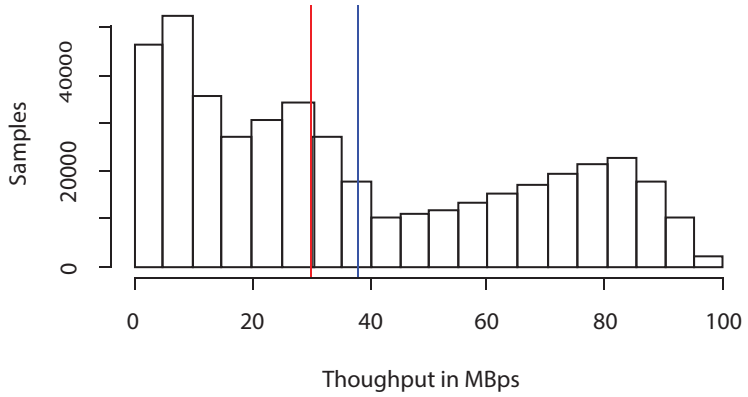


Figure 2.8: Histogram of Monte Carlo simulation for PureSDN elephant flow loss rate.

will be longer than the left-hand tail, which means most of the data will be on the left-hand tail. Consequently, the length of the tail may affect considering the average value as the expected value of the loss rate. We obtained kurtosis degrees for each histogram to identify which one produces more outlier values. We found that the histograms follow the platykurtic distribution since the kurtosis is negative compared with the Normal distribution. Hence, the expected behavior for the algorithms is to produce fewer outliers values.

2.3.6 Value at Risk (VaR) Analysis

Monte Carlo simulation is considered the best method to estimate the value at risk [45]. We utilized VaR to indicate the expected significant blocked elephant flow rate based on the Monte Carlo simulation results. Even though the histograms and the statistics provide comparative information about the behavior of the model and the loss rate prediction, Value at Risk (VaR) analysis could deliver a more profound analysis based on some confidence level [91]. Note that the blocked rate of elephant flow represents the number of bytes per second throttled in the DCN fabrics DCN adopted such TE method.

In this research, the chosen confidence level was 95% since outlier results would appear with a more significant percentage. We calculated the confidence level by considering the quantile function (Equation 2.6).

$$VaR = -\mu_{nv} + \phi^{-1} \times (1 - u_c)\sigma_{nv} \tag{2.6}$$

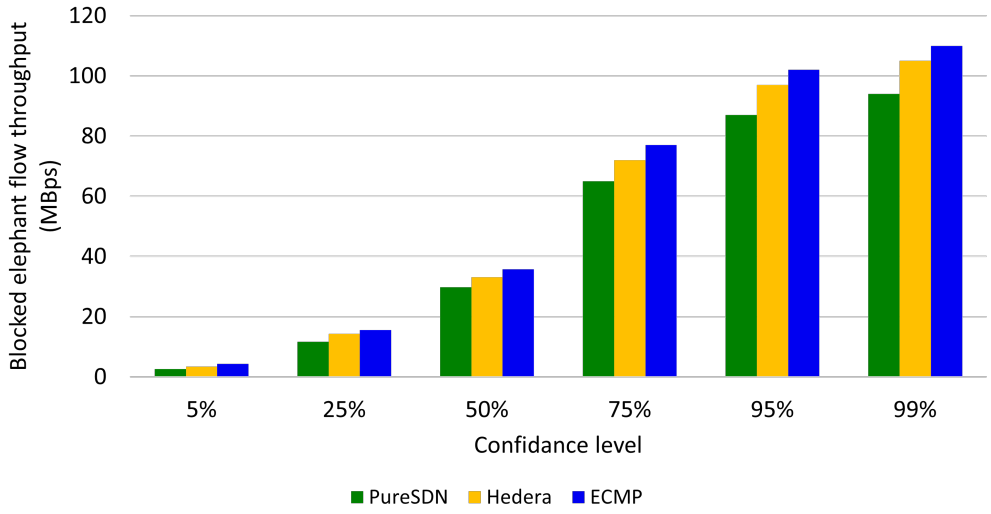


Figure 2.9: Different confidence levels of VaR analysis.

Where μ_n is the mean of the prediction values (nv), ϕ is the standard Normal distribution function, σ_n is the standard deviation of the values, and $(1 - u_c)$ is used for the chosen confidence level.

As depicted in Figure 2.9, the loss rate in PureSDN is the lowest, with 87 MBps compared with Hedera and ECMP, which were 97 MBps and 102 MBps, respectively. Mainly, these values represent the maximum loss rate value of the elephant flows over the DCN production lifespan.

2.3.7 Risk as Expected of Elephant Flows Loss

Now, we answer the main question (**Question 1**): *"To what extent can different TE methods maintain elephant flow throughput during the production DCN lifespan?"*. To this end, we apply the obtaining results of the uncertainty and significant losses into a single concept "*expected loss of the event*" introduced in [41]. In our term of analysis, the expected loss is defined as the uncer-

tainty of the elephant flow loss times the predicted significant loss. Table 2.6 shows the expected elephant loss for the examined TE methods.

TE method	Loss uncertainty	Significant loss	Expected loss
Hedera	56.76%	97 MBps	55.05 MBps
ECMP	62.83%	102 MBps	64.08 MBps
PureSDN	44.43%	87 MBps	38.65 MBps

Table 2.6: The expected elephant loss for the examined TE methods.

In our research, however, there are two types of analysis as seen by decision-makers, "risk neutral" and "risk averse," as suggested by Douglas W Hubbard in [41]. As proved in the Monte Carlo simulation, the elephant flow loss event is highly expected DCN. This loss directly influences the status of DCN applications in terms of flow completion time and throughput. Therefore, we adopted a risk-averse analysis definition that reveals the highly expected elephant flow loss by employing the TE method for flow scheduling in the DCN. Thus, the decision-maker should be aware of those risks. On the other hand, the risk-neutral analysis of the uncertainty and losses will be defined as expected outcomes with a certain average weight.

Table 2.6 even remarks that the range of optimization in elephant flow loss reduction uncertainty is about 18% between the thoroughly hashing-based ECMP and full SDN-based in PureSDN. Hence, developing a flow scheduling method should balance the cost of monitoring and scheduling and the expected elephant flow loss.

2.4 Conclusion

This thesis proposed a novel risk analysis framework for elephant flows scheduling inside the DCN based on existing SDN-based TE methods. We

empirically designed, implemented, and analyzed a risk analysis framework for existing TE methods based on multiple stochastic workloads and traffic patterns to predict the value at risk of the elephant flow loss rate. The evaluation considered the proper probability distribution functions for the proposed risk factors of the loss rate for Hedera, ECMP, and PureSDN. The proposed evaluation model has been built based on Monte Carlo simulation as a value at risk analysis model. The evaluation included estimating the probability distribution for risk factors based on the Kolmogorov Smirnov and Anderson-Darling tests. Finding the probability distribution of such algorithms helps further mathematical analysis regarding elephant flow handling without conducting more practical experiments. The probability of elephant flow losses showed that Hedera achieved 56.76% with 97 MBps maximum expected loss, ECMP is 62.83% with 102 MBps, and PureSDN is 44.43% with 87 MBps, respectively. Hence, the design and development of the TE methods required proper awareness in terms of elephant flow risk analysis. Besides, the analysis built upon real DCN workloads making it suitable to provide a quantitative benchmark for any TE method. Furthermore, our proposed analysis contributed to the term of DCN SLA by predicting the adopted flow scheduling productivity regarding elephant flows.

However, more research is needed to investigate the elephant flow behavior for different TE methods such as Markov approximation strategy [33] [32], in addition to machine learning different methods [80] [107].

Chapter 3

Adaptive SDN Load Balancing Framework

"Everything is theoretically impossible, until it is done."

Robert A. Heinlein (1907-1988)

3.1 Introduction

In contemporary DCN, the research community has focused on flow scheduling and traffic load balancing leveraging SDN and OpenFlow protocol [19] [96]. Nevertheless, several issues need to be addressed when applying SDN solutions to DCN load balancing. For instance, the central SDN controllers can handle only a specified number of *packet_in* requests per second. On the other hand, the TCAM (Ternary Content-Addressable Memory) space limitations of OpenFlow switches affect the flow entries that the controller can handle. As for decision making, frequent elephant flow rerouting may

degrade the TCP performance through packet reordering, not to mention the increased complexity of the flow scheduling calculation. Furthermore, the DCN flows' demand cannot be predicted in advance to obtain a suitable route without the cooperation of other devices such as sFlow [90], or even the end-hosts [44].

Elephant flows typically try to make full use of the link capacity. Consequently, mice flows could suffer from real-time latency [31]. According to research, any delay in the response time of DCN applications has a significant impact on the user experience [69]. For example, by increasing the flow delay by 400 ms, Google discovered that daily searches were reduced by 0.6 percent [15].

The general design of the DCN network topologies includes multi-rooted trees that provide multiple paths between each pair of hosts. Accordingly, the challenge here is to identify the suitable path for the flows according to the current load of the path and avoid traffic congestion and potential conflicts. As proved in Chapter 2, the ECMP or even Hedera scheduling will cause traffic congestion with a significant elephant flow loss risk and bursty DCN traffic.

The first part of our work is to permit the elephant flows to use total bisection bandwidth before entering the DCN edge switches to the predefined congestion state. This thesis explores how the flow-based load balancing scheme delivers better Flow Completion Time (FCT) with a minimum elephant flow loss rate. Therefore, the Thesis questions are:

(1) How to schedule the flows to improve DCN performance, such as mice flow FCT and reduce the elephant flow losing risk.

(2) Does 1:1 sampling between ECMP and the SDN controller necessarily lead to significant packet loss in mice flows and affect the elephant flows throughput due to flow collisions?

(3) Does the additional computation complexity to detect and reschedule elephant flows necessarily lead to a significant delay in the *packet_in* handler?

(4) Can we introduce a deployable, light, and yet effective load balancing technique that works in symmetric and asymmetric DCN topology using the SDN concept without altering network components or TCP packets?

(5) How to secure the DCN bisection bandwidth even with defective links?

Methods have been already appeared in the literature to deal with load balancing problems and flow scheduling dilemmas. Still, they lacked a complete picture of elephant flows losing rate, or even based on altering network packets or components to detect elephant flows or failed links. To address these questions, we divided the thesis into two sub-theses.

3.2 Research Motivation

DCN topology is typically designed with multi-rooted layers and multiple paths between each pair of hosts [27]. As a result, finding a suitable path is difficult because avoiding congestion necessitates taking into account the current load besides potential flows conflicts. The motivation example, depicted in Figure 3.1, shows a $K-4$ Fat-tree DCN contains shuffle communication patterns regardless of the flow size. Two senders, H1 and H3, con-

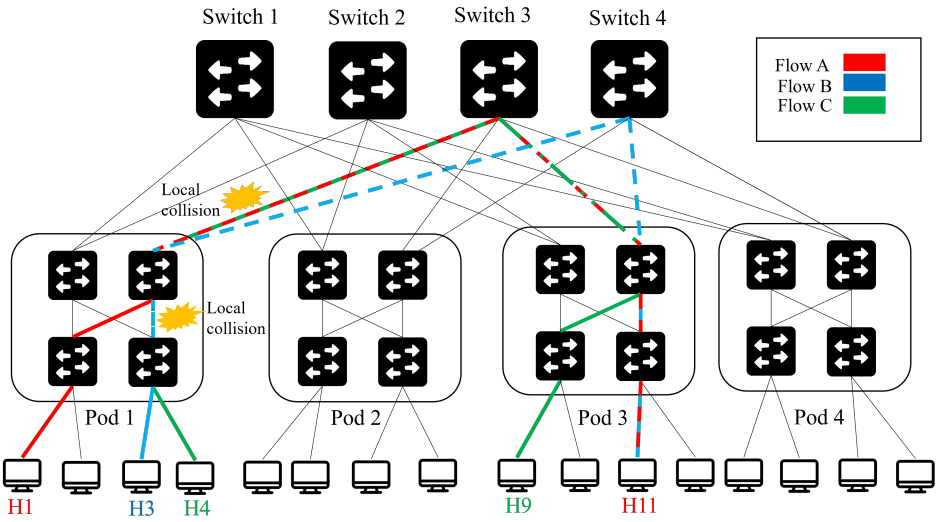


Figure 3.1: Motivation example on flow collisions inside $K-4$ fat-tree DCN explains flow collisions yielded as a result of static hashing based routing employed by ECMP.

nected to pod 1 initiate two mice flows to H11 in pod 3. It was assumed that there was an elephant flow in the background transferring data from H4 to H9. Using ECMP only in such a scenario results in several collisions, as illustrated in Figure 3.1, because it employs scheduling based on static hashing of the packet header where congestion occurs due to overburdened links. This sort of congestion will profoundly diminish QoS [15] of mice flows. As a result, deploying applications in DCN using ECMP does not ensure QoS due to data loss and re-transmissions. Flow contentions and bottlenecks, on the other hand, are unavoidable [90]. Hence, rescheduling flows from one path to another solely based on flow bandwidth consumption, as in Hedera [27], may result in another type of congestion and flow completion delay. Therefore, improving QoS in DCN necessitates taking into account the characteristics of both flow classes and the network situation.

3.3 Literature Review

We organize the studies that appeared in the flow scheduling literature into two categories. One for the central solutions resides in the control plane, while the other for the studies that involve the DCN data plane in the operations of flow scheduling.

3.3.1 Central Solutions

Several studies have emerged dealing with flow scheduling in DCN. Known solutions such as Hedera [27] and Mahout [19] worked on rescheduling elephant flows based on central SDN controller, but with different elephant flow detection mechanisms. Devoflow proposed in [20] provides a flow control mechanism in DCNs by rerouting elephant flows with more significant sizes than 1 MB. However, Devoflow did not show the elephant flows scheduling effect on the FCT measurements of mice flows. MiceTrap [92] is a mice flows detection and scheduling algorithm based on finding underutilized paths. Nevertheless, the authors did not consider elephant attributes and their impacts on mice flow completion time. Yazidi et al. [102] classified links into hot and cold links to reschedule the detected elephant flows to the least congested links. Still, the controller overhead of the proposed mechanism is remarkably high since it requires monitoring the demands of all DCN flows. L2RM framework proposed in [101] to balance the load in fat-tree DCN topology. L2RM controller maintains three tables for each switch to save routes information, traffic loads, and flow entries. L2RM invokes a rerouting mechanism on a threshold of 25% of the utilization deviation value. L2RM treats both flows classes similarly.

3.3.2 Distributed Solutions

Distributed solutions aim to overcome the control overhead. In this context, a fine granularity has been introduced by flowlet, a data unit defined in [48] as a packet burst. Flowlet is leveraged in CONGA [9] to obtain optimal flow scheduling in leaf-spine DCN by applying leaf switches to feedback on congestion metrics. Nevertheless, CONGA requires custom switches based on Application Specific Integrated Circuit (ASIC). Wang et al. [100] employed end-hosts to detect elephant flows to be scheduled by the controller. The remaining flows on the switches, on the other hand, are scheduled by ECMP. End-hosts are used by BLEND [55] to track all outgoing elephant flows and to select paths for the remaining flows locally. Modifying the kernel of the end-host is not a viable solution. Afek et al. [3] proposed a sampling technique that makes use of the OpenFlow group feature. The authors manage to preserve the flows information into a data structure in the controller to follow the total counter of each flow. Besides, a unique flow entry for a prospect elephant flow will be established after passing a predefined threshold. As an effect, the controller will be concerned with more processing.

Tang et al. [90] proposed a flow classification model for both mice and elephant flows by sampling *packet-in* packet of each flow to recognize aggregated flow category. Accordingly, such a solution is presented to misclassification. Hermes [108] is a congestion-aware load balancing technique. Hermes proactively schedules flows when congestion or failure occurs. For congestion detection, the method relies on ECN (Explicit Congestion Notification) and RTT (Round-Trip Time). Despite the fact that Hermes is deployable because it does not require any hardware changes, all DCN end-hosts must participate in the sensing method. As a result, the sensing approach

cannot be implemented without end-host visibility. CAPS [37] is an end-host-based congestion-aware technique. The solution includes three modules: a packet encoder and decoder on each DCN host and a packet spraying module on the ToR switch based on Random Packet Spraying (RPS). Traffic flows are divided into mice and elephant flows, with elephants being scheduled to specific paths using ECMP and mice being scatted to all available paths based on RPS. Aside from RPS capable switches, this technique necessitates changes to end-hosts. Luopan [99] is a distributed congestion aware approach based on sampling routing paths between ToR switches to direct flowcels to the least congested path. The method operates at flowcel granularity with a 64KB threshold. Therefore, the end-host NIC must be managed in order to generate flowcels.

Path sampling, on the other hand, is accomplished through packet propping, which necessitates TCP/IP modification. In [53], Levi et al. proposed an elephant flow detection and rerouting method. The DCN topology path diversity is utilized to reroute elephant flows on less congested shortest and non-shortest paths. The method is primarily based on Hedera's strategy for scheduling initial flows via proactive paths (ECMP). Following that, elephant flows exceeding 10% of the link capacity are rerouted based on a congestion threshold. The best path is then chosen based on the path congestion rank and the estimated path delays. Although the proposed method improved elephant flow latency, it did not investigate the average overall FCT.

3.4 Our SDN TE Solutions

In this research, we investigate the possibility of introducing a deployable, light, yet effective load balancing technique that works based on the SDN concept only without altering the network components or TCP packets. We propose Sieve, in which we leveraged the OpenFlow protocol group table to form a sampling strategy at the edge switches of the DCN topology. The main objective of the flow samplings technique is to reduce the number of installed flow entry rules since a fraction of flows will be handled by proactive paths (ECMP-based), and the SDN controller will manage the rest. In Sieve, we resolve the expected bottlenecks by frequently rescheduling the fraction of elephant flows detected on edge switches to balance the elephant flows among the available paths. Furthermore, we propose Oddlab, a novel SDN TE solution to detect and avoid the expected faulty links in DCN. In Oddlab, we adopted the idea to schedule the flows to the least number of active elephant flows without frequent elephant flow rerouting. In Oddlab, we maintain the number of installed flow entry rules without tracking and rescheduling the elephant flows. As a result, Oddlab is designed to be served efficiently in both symmetric and asymmetric fat-tree DCN topology.

However, we define the deployment cost of SDN-based flow scheduling solutions by the needed amount of altering (i.e., hosts or switches) or adding (i.e., sFlow sampling or distributed controllers) to the main components of the DCN. In Sieve and Oddlab, we only leveraged the global network visibility at a single and central SDN controller to fulfill effectual flow scheduling solutions, including faulty link awareness. Thus, the estimated cost of deployment is equal to a single SDN controller and commodity OpenFlow switches hardware.

3.5 Sieve Framework Design Aspects

This section discussed the Sieve main design aspects, starting with problem formulation and Sieve model description.

DCN is modelled as directed graph $G = (V, E)$, where V is defined as set of the nodes $V = \{v_0, v_1, ..., v_n\}$ and E as set of the directed edges $E = \{e_0, e_1, ..., e_n\}$. Network flows are routed between every source $s \in V$ and target $t \in V$ with a path $P = (v_0, v_1, ..., vn), \forall v \in V$. A directed graph defined the flow network as $G = (V, E, c)$, where each edge $(u, v) \in E$ has capacity $c(u, v)$. The network flow problem can be classified into two types based on their demands (i. e., commodities): single commodity and multi-commodity. There are only single sources and targets (s, t) in single commodity flow, where $s, t \in V$, and $s \neq t$. On the other hand, multi-commodity flow contains set of commodities originated from set of (s_i, t_i), where $(s_i, t_i) \in V$.

In Sieve, however, we define two significant problems, including initial flow scheduling and elephant flows rescheduling to resolve the predicted bottlenecks on DCN edge switches:

Definition 6. *Problem* (Sieve Adaptive Flow Scheduling (SAFS)): *Given a multi-commodity DCN flow $G = (V, E, b)$, where each edge $(u, v) \in E$ has available bandwidth $b(u, v)$, the goal is to choose the path with minimum statistic costs from V_s to V_t that contains the maximum residual bandwidth.*

Note 7. *the minimum statistics costs refer to the obtained residual bandwidth that costs only one loop over the monitored edges per polling interval (P_r).*

Definition 8. *Problem* (Sieve Elephant Flow Rescheduling (SEFR)): *Given a multi-commodity DCN flow $G = (V, E, El\_e)$, with a certain number of active*

elephant flows $El\_e(u,v)$ at the edge switches, the objective is to resolve the path bottleneck by detecting and rescheduling a set of elephant flows each with the source and the destination (with flow rate $\geqslant$ 50 Kbps) to other available paths from V_s to V_t.

3.5.1 Problem Formulation

In general, the routing problem in flows assignment should satisfy four constrains: (i) the throughput of all flows routed on a link should not exceed its capacity $c(u,v)$; (ii) the number of flows entering a node v equals the flows that exit from the same node; (iii) A flow must leave its source node completely; (iv) A flows must enter its target node completely. The load balancing dilemma is based on scheduling set of flows $fe_i(u,v)$ over the edge (u,v) among all links' capacity $c(u,v)$ evenly, as shown in link utilization $U(u,v)$ Equation 3.1. Were $(u,v) \in E$, kc the set of commodities defined by $kc_i = (s_i, t_i, D_i)$, s_i the source, and t_i the target, D_i the demand for the commodity i.

$$U(u,v) = \frac{\sum_{i=1}^{kc} fe_i(u,v).D_i}{c(u,v)} \tag{3.1}$$

The obvious solution for this problem is accomplished by minimizing the maximum utilization for the links U_{max}. The demands D_i for each commodity is not consistent, and it is costly to monitor it to deliver better utilization. Since there is a cost $t(u,v).f(u,v)$ when scheduling and monitoring each flow on (u,v). Consequently, we intend to minimize the flow scheduling cost f_cost (Equation 3.2) by employing the ECMP flow hashing method and the SDN controller based on the power of two choices concept [63].

$$f\_cost = Min \sum_{(u,v)\in E} (t(u,v) \sum_{i=1}^{ks} fe_i(u,v)) \tag{3.2}$$

However, the process of choosing a particular path $p = (v_1, v_2, ..., v_{np})$, among np set of paths by the SDN controller for each pair of nodes should satisfy the following constraints: (i) finding the path p with minimum edge count (link e) $\forall v \in V$ between sources and targets (s, t) in single commodity flow, where $s, t \in V$, and $s \neq t$. (ii) the throughput of every link e should not be less than link physical capacity C_e (Equation 3.3), where l_e is the current link load, and C_e is the physical link capacity; (iii) maximum path P_b residual bandwidth from source to target (Equation 3.4).

$$\frac{l\_e}{C\_e} < C\_e \tag{3.3}$$

$$P\_b = Max(C\_e - l\_e) \tag{3.4}$$

To reduce the elephant flow rescheduling cost, only a fraction of them will be rescheduled from path P_1 to path P_2. The condition of choosing the nominated number of elephant flows relies on the following:

- Number of all installed elephant flows (El_p_1) with the byte size ($flow\_net \geq 50$ Kbps) on path $El\_p_1 > 0$.
- A load of the current path C_p1 should be larger than the congestion threshold $Thr > 25\%$ of link capacity, while the load of other path $C\_p2 < 25\%$ of link capacity (C_e).

Different link occupation thresholds have been measured in our joint work (Sieve [104]) based on the number of elephant flow detection mes-

sages (`OFPFlowStatsRequest`) and the number of the rescheduling failure message. This procedure has been taken to not overwhelm the SDN controller by the number of elephant flows that must be rescheduled. Thus, the threshold of 25% was found to be the most suitable value to deliver better mice flow FCT with a reasonable number of rescheduled elephant flows. On the other hand, finer granularity (such as 10%) would produce less elephant flow rescheduling messages, but at the expense of high controller overhead monitoring numerous elephant flows. Nevertheless, higher thresholds (i. e., 50% and 75%) will decrease the total framework efficiency by reducing the number of elephant flows managed by the controller.

However, the number of the rescheduled elephant flows will be calculated in Equation 3.5, as proposed by Maiass in our Sieve joint work [104].

$$Num\_redir\_EFlows = E\_count \times \frac{E\_u}{E\_c} \tag{3.5}$$

Where E_count represents the number of all elephant flows on the edge port, E_u is the current port utilization, and E_c is the physical edge port capacity. We illustrat the procedure of estimating the rescheduled elephant flows as suggested by Maiass in Algorithm 1, lines (5-13) [104].

3.5.2 Adaptive Elephant Flow Rescheduling solution

In this thesis, we investigated the flow distribution scheme in DCN through predefined flow paths at the DCN edge switches employing ECMP hashing besides the SDN controller. Throughout conducting extensive experiments on a wide range of traffic patterns with synthetic and realistic DCN workloads. We found that the proposed method delivers noticeable improvements in bisection bandwidth, moderate link utilization. An average

overall reduction in FCT, packet loss, average delay, and finally a better elephant flows losing rate compared with standard ECMP [36] in addition to dynamic SDN scheduling method such as PureSDN [59] and Hedera [27]. The main reason behind choosing flow sampling is that in addition to being simple to achieve in the DCN switches, it speeds up the time response of the DCN to the growing number of applications flows.

As for the adaptive flow scheduling by the SDN controller, Sieve efficiently schedules DCN flows based on the available bandwidth of network ports without altering the network components to improve the Flow Completion Time (FCT) of the mice flows. Figure 3.2 depicts the Sieve architecture implemented in the SDN control plane and data plane.

At the DCN edge switches, we leveraged a distributed flow sampling procedure by adopting a flow sampling technique (1:1). In this manner, a portion of the flows will be forwarded through proactive paths leveraging ECMP, and the rest of the flows are scheduled using the adaptive scheduling method of the Sieve SDN controller. Periodically, the controller monitors the congested edge switches looking for elephant flows to reschedule some of them to other underutilized paths.

3.5.3 Flows Distribution at DCN Edge Switches

It is not feasible to leverage the SDN controller to compute a suitable path for each flow in the network. We sample the incoming flows based on OpenFlow SELECT group type with two buckets so that the *packet_in* request of a flow is either scheduled directly to the ECMP predefined path or sent to the SDN controller. The select group can be defined with weighted buckets, and each bucket can perform specific actions on the switch port (i.e., drop

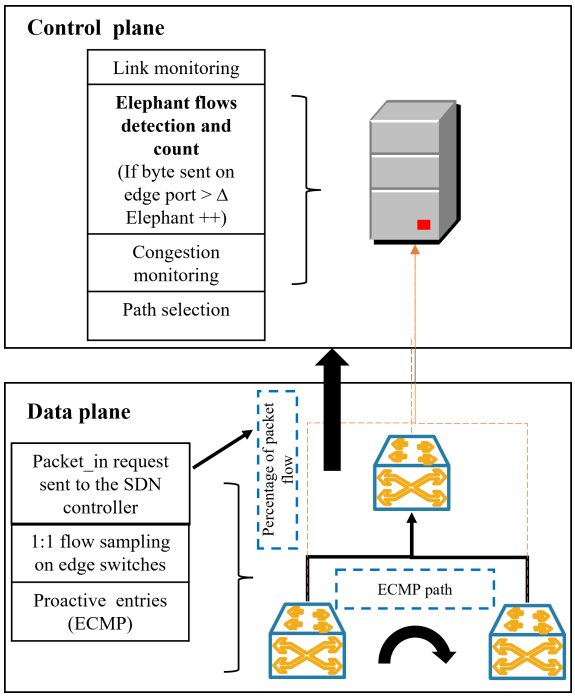


Figure 3.2: Sieve SDN architecture.

or forward the packets). As depicted in Figure 3.3, identical weights have been set for the proactive path (ECMP table) and control handling.

At edge switches, we defined multiple flow entries including, direct flow entries and polling flow entries. The direct flow entries contain the target end-host's IP address and the direct output port that connects the host directly to the edge switch. In contrast, the polling flow entry contains the information of (*Ip_src, Ip_dst, transport_src_prt, transport_dst_prt, and action: the output of edge switch port*). Subsequently, if the incoming packet did not match any existing flow entries at the edge switch, it would be forwarded dynamically to the controller or the ECMP table with uniform probability.

Theorem 9. *Sieve 1:1 flow distribution at the DCN edge switches produces even distribution between proactive paths (ECMP) and SDN controller.*

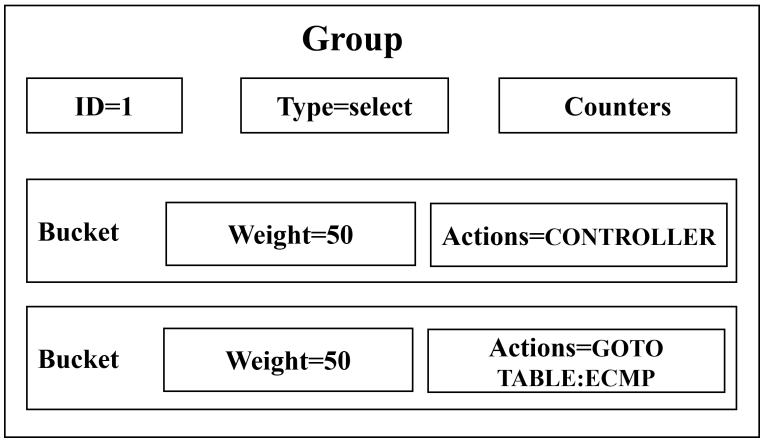


Figure 3.3: The sampling group entry at the edge switches.

Proof. To prove that the hashing-based packet allocation at the edge switches produces an even flow distribution, we apply SUHA (Simple Uniform Hashing Assumption) on the proposed sampling mechanism. As a result, a flow cannot be allocated to both buckets at the same time. Equation 3.6 presents the probability of the hashing function.

$$Pr(h(xs) = iv) = \frac{1}{z} \tag{3.6}$$

Where $h(xs)$ is the hashing function, iv is the index value ($iv \in [0, z-1]$), and z is the hashing locations (i.e., in our case z = 2). Accordingly, the upper bound of the flows allocated to ECMP bucket can be calculated as in Equation 3.7 using Boole's inequality.

$$Pro(ECMP \quad receives \geq w \quad flows) \leq \binom{z}{w} \frac{1}{z^w} \tag{3.7}$$

The binomial combination $\binom{z}{w}$ represents the subset of the hashed flows where every flow from set w will be sent to the ECMP bucket with the probability of $\frac{1}{z^w}$, as proven in [64] for the Balls-and-Bins model. Therefore, the flows will be evenly distributed.

□

3.5.4 Edge Sampling Performs Under a Finite Systems

At the edges, we adapted two identical bucket-forwarding decisions. Accordingly, the incoming flows are independently hashed to the ECMP or SDN controller uniformly. In particular, the flows are served based on the first-in-first-out manner (FIFO) with an exponential distribution for the service time of flows. This phenomenon is associated with the supermarket model discussed in [63]. Hence, we utilize this model to provide a precise and systematic analysis of our flow sampling model behavior when the number of flows tends to be significant ($f \to \infty$). We assume the following parameters: the flows arrive at the DCN edge switches following a Poisson process with a rate of λns, at the number of available forwarding choices (i. e., servers) on the system ns =2 (SDN and ECMP), with $dc = 2$, the sampling decisions. To obtain the expected waiting time of the flows beside the maximum queue length, Kurtz's theorem for large numbers and Chernoff-like bounds [63] are applied as follows:

Theorem 10. *The anticipated time that the flow can wait at the DCN edge to get forwarding service, whether with ECMP or Controller ($dc = 2$) over the period $[0, T]$ is limited to:*

$$\sum_{f=1}^{\infty} \lambda i^{\frac{dc^f - dc}{dc-1}} + O(1) \tag{3.8}$$

Where $O(1)$ depends on T and ($\lambda i = \lambda ns$) states and represents the error bound between the system's state with a fixed number of the forwarding choices ns and when it goes to infinity. Nonetheless, empirical simulations revealed the same behavior even for small ns over time intervals as reported in [63].

Theorem 11. *The maximum initial queue length on the DCN edge switches with ($dc = 2$) over the period $[0, T]$ would be equals to:*

$$\frac{\log \log ns}{\log dc} + O(1) \tag{3.9}$$

Our proposed method presents an exponential improvement in the waiting time and the queue length. As a result, the ends of the queues will be reduced doubly exponentially rather than singly exponentially when $dc = 1$ [63]. As a result, the central controller will not be overwhelmed with a vast number of *packet_in* requests. In this manner, the controller can handle the flows precisely, and thus multiple applications can be performed in the control plane without excessive burden.

3.5.5 ECMP Proactive Scheduling

ECMP proved to be a fast flow scheduling technique since it spread the flows across all the available paths without considering the path status. Therefore, flows collisions frequently occur, causing packet losses with significant delays in the flow completion time. We reduce such collisions and

congestions by using the SDN controller's efficient adaptive flow scheduling.

For ECMP path implementation, we defined the SELECT group table with two buckets on the DCN switches' upstream side. For instance, on the side of edge switches, when connected to the aggregate switches, we established two buckets with actions (OUTPUTPORT:1 and OUTPUTPORT:2) using identical bucket weights to balance the incoming flows between the upstream ports evenly (Figure 3.3). Hence, the same procedure applied to aggregate switches' ports connected to the core switches on the upstream ports (i.e., port 1 and port 2). Note that in fat-tree DCN topology, the DCN's downstream side traffic cannot be balanced since the link will be directly connected to the destination end-host through the core switch, aggregate switch until the edge switch. Accordingly, we defined flow entries with fixed priority values for the directly connected subnetworks on the DCN downstream ports (i.e., port 3 and port 4).

3.5.6 Detecting and Rescheduling Elephant Flows

To resolve the DCN bottleneck, Sieve tries to reschedule only a fraction of elephant flows found the utilization ratio of any edge switch port below a predefined threshold (25% of the link capacity) based on the obtained information of `OFPPortStatsRequest` function (Algorithm 1, line 2). Accordingly, `OFPFlowStatsRequest` invoked to detect and count the elephant flows that will be rescheduled. Firstly, only flow entries with data transfer speed $\geqslant$ 50 Kbps [11] will be counted as an elephant flow. Then, the redirected flows will be counted based on the current port utilization (E_u), as shown in Algorithm 1, line 13. Therefore, the relationship of rescheduled elephant

flows (num_redir_EFlows) with the port utilization E_u will be direct; as the utilization increases, the number of selected flows from that port increases.

Note 12. *The adopted ECMP proactive scheduling reduces the number of redirected elephant flows. Thus, reducing the burden on the SDN controller and diminishing the impact of elephant flow reschedulings such as TCP reordering and packet losses.*

Next, each elephant flow will be redirected according to the `get_best_Path` function to ensure that each flow will get the best available bandwidth path (Algorithm 1, lines 17 - 33). Finally, the new paths will be installed from end to end across the DCN topology.

However, this solution discovers just a portion of the total DCN flows so that a fraction of the flows are not sampled. Assuming that network flows arrive at the DCN edges. According to the adopted flow sampling mechanism, a portion of these flows will be scheduled based on ECMP, and the rest will be scheduled based on Sieve adaptive scheduling. Let CF represent the cumulative flow size for each flow in the network, t' is the operation time interval [0, t'], and NF is the total number of the network flows.

Algorithm 1: Detect and reschedule elephant flows in Sieve [104].

Data: *G= (V, E), F_BW, Th, E_u, min_bw= E_c, dpid_list, max_bw= 0, k, shortest_p= { }, PR, EF_list= { }, Paths= { } , Portid_list*
Result: *best_p= [ ]*

```
foreach P_r do
    Function OFPPortStatsRequest(dpid_list):
        if (E_u_ij < Th and i ∈ pdid_list and j ∈ Portid_list) then
            Reschedule (i, j, E_u_ij):
    Function OFPFlowStatsRequest(i):
        Reschedule (i, j, E_u_ij):
        E_count = 0 for e in F_list: do
            flow_net = flow_byte / flow_duration
            if flow_net ≥ 50 Kbps and Output_port == j then
                EF_list ← (f_info)
                E_count++
            num_redir_EFlows= E_count × E_u(i,j) / E_c(i,j)
            if num_redir_EFlows > 0 then
                for f in (0, num_redir_Eflows) do
                    get_best_Path (G, i, f_info, E_u_ij, num_redir_EFlows)
    Function get_best_Path(G, i, F_info, U_BW_ij, num_redir_Eflows):
        for P in Shortest_P: do
            if (link (P[i] P [i+1]) != j) then
                Paths ← (P)
            else
                continue
        best_p = get_best_Path_by_Bw(i, G, Paths, E_u_ij)
        return best_p
    Function get_best_Path_by_Bw(i, G, Paths, E_u_ij)):
        min_bw= E_c
        max_bw= E_u_ij
        for P in Paths do
            min_bw = bottleneck_of_path (G, P, min_bw)
        if (min_bw > max_bw and min_bw - E_u_ij > 1Mbps) then
            max_bw = min_bw
            best_p = P
    if (best_p) then
        return best_p
    else
        "No path met the conditions"
```

Theorem 13. *The cumulative flows size sampled by the controller is approaching the half of the total size of flows in the DCN as ($t' \to \infty$):*
i.e., $\lim_{t' \to \infty} nc(t')\ cc(t') \to \frac{NF}{2}\ CF$, *where $nc(t')$ is the number of sampled flows by the controller over the time interval [0, t'], and $cc(t')$ is the cumulative flow size for each flow sampled by the controller over the time interval [0, t'].*

Proof. Mitigating the load on the controller is our aim so that it will not be overwhelmed by samples. Let us assume a probabilistic scheme by applying the strong law of large numbers. In this context, each flow has a probability of being sampled by the controller (p_c) or scheduled by ECMP (p_e). To denote the event of sampling the first packet by the controller, we use $I\_c = 1$ with probability p_c; in contrast, $I\_e = 1$ with probability p_e indicates the event of ECMP based scheduling. Thus, the expected number of the sampled flows by the controller is $E[I\_c] = p\_c$, and similarly the expected number of the flows scheduled based on ECMP is $E[I\_e] = p\_e$. Note that the variances of these values are equal to Equation 3.10 and Equation 3.11, respectively:

$$V[I\_c] = p\_c(1 - p\_c) \tag{3.10}$$

$$V[I\_e] = p\_e(1 - p\_e) \tag{3.11}$$

The cumulative size of the sampled flows resulting from the proposed sampling process is, regardless of whether these flows are mice or elephants, represented in Equation 3.12 for cumulative flows sampled on the controller (cc) and Equation 3.13 for cumulative flows sampled on ECMP (ce). Where ne is the number of sampled flows by ECMP, nc is the number of sampled flows by controller, and D denotes the flow demand:

$$cc = \sum_{f=1}^{n_c} I\_cD_f \tag{3.12}$$

$$ce = \sum_{f=1}^{n_e} I\_eD_f \tag{3.13}$$

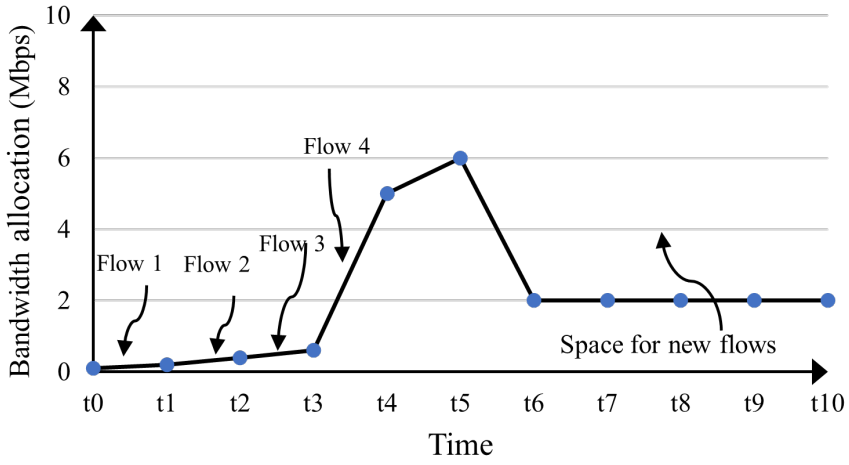
By applying the strong law of large numbers, the expected number of the network flows on each hashing decision (i. e., ECMP or controller) will tend to become closer to the expected value and distributed equally over them. Hence, the the total cumulative size of the flows (i. e., $CF = cc + ce$) is determined by the number of sampled flows $\left(\frac{NF}{2}\right)$. Since the majority of DCN bytes come from elephant flows [77]. Therefore, Sieve can manipulate half of the elephant flows transferred data between DCN end-hosts. Thus, flow conflicts and congestion due to the sole dependence on ECMP can be mitigated. As a result, the number of flow entries managed by the controller will be reduced to half in long-term operation, regardless of rescheduled elephant flows.

□

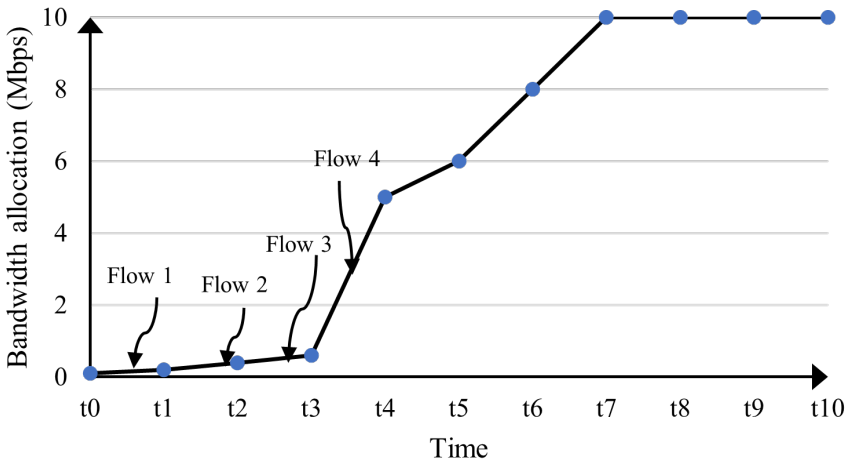
Theorem 14. *Active elephant flows monitoring and redirecting based on the edge switches links is enough to produce less packet loss and reduce the FCT of the mice flows.*

Proof. The many-to-one connection is a pervasive communication pattern inside the data center network. This phenomenon essentially occurred inside the aggregate layer switches when multiple edges attached to a single switch in the aggregate layer and produced overflow queening congestions. The elephant flows consume most of the bandwidth of the link. Accord-

ingly, in each link, the max-min allocation in TCP congestion control tries to maximize the amount of allocated bandwidth to the elephant flows at the expense of the mice flows.



(a) Sieve adaptive scheduling.



(b) Hedera (Global First Fit).

Figure 3.4: Example of the difference between the adopted elephant flow rescheduling procedure on link X in Sieve using link bandwidth occupation, in Figure 3.4 a, and employing flow size as threshold utilized in Hedera as depicted in Figure 3.4 b .

Now, assume two links (X and Y) receive several mixed flows with arrival rate according to the Poisson process (λ). For illustration, the link X will be demonstrated with four flows (three mice flows and one elephant flow). The first three flows arrived and served in the first three sec.; then, the elephant flow begins the slow starting phase at the third second as depicted in Figure 3.4 (a and b).

In the case of Sieve, we define polling rates $p1$, $p2$, ... to monitor the load on the edge switches. Once the occupation rate on an edge switch port of link X below 25% of the total capacity, the controller starts to detect elephant flows on it. Therefore, during the polling rates $p1$ and $p2$, the controller only saves the total load values on X into the network graph. During $p3$, the occupation reaches the threshold, so the elephant flow detection module discovers the elephant flow, and it tries to reschedule it to another link with better bandwidth, i.e., Y, during $p4$ as shown in Figure 3.4 a. Consequently, some bandwidth will be available for the current and upcoming mice or elephant flows.

On the other hand, Hedera monitors all flows at edge switches to detect the elephant flows. The elephant flows detection is based on exceeding 10% of link capacity. Therefore, in one Gbps capacity link, the elephant flow must occupy more than 100 Mbps. Hence, in the co-existing of many elephant flows whose consumptions are below 10% of the link capacity for each, the link will be saturated before elephant flows are appropriately detected and rescheduled. Thus, mice flows will suffer from latency, and their FCT will increase as a result. As depicted in Figure 3.4, every mice flow scheduled by ECMP after $t7$ will be exposed to significant delay or even be lost since the elephant flows have already overtaken the link capacity. Furthermore, Sieve invokes the elephant flow rescheduling only when a

link occupation approaches a predefined threshold of its capacity. Besides, we sample a portion of the flows. As a result, the monitoring and detection burden is reasonable. In contrast, Hedera monitors the consumption of each flow to invoke the rescheduling procedure. □

3.6 Oddlab: Adaptive flow scheduling based on the active elephant flows inside DCN topology.

To achieve better FCT, Sieve [104] attempted to reroute the fraction of elephants on an edge switch when this switch reaches a specific load threshold to a different path based on the residual bandwidth. In this mechanism, however, the new path could also have some new arrived flows that will grow in the accumulative size to be new elephant flows, which will produce further congestion and flows contention. Roy et al. [77] found that the long-lived TCP flows (elephant flows) in Facebook DCN are not so heavy over long periods; hence, frequent elephant rerouting could bewilder to various TE approaches. Moreover, one of the most crucial issues in DCN fabrics is the links failures [29]. Consequently, it is not easy to maintain the symmetric situation for the duration of the network operations. In this thesis, we propose Oddlab, a heuristic and dynamic TE approach to load balance the traffic of DCN based on a centralized SDN controller. Oddlab differs significantly from our previously flow scheduling method (Sieve) in two main aspects.

(1) In addition to flow sampling at edge switches of the DCN (1:1 sampling between ECMP and SDN controller), We use the number of active flow entries relating to the elephant flows [59] besides the residual bandwidth the path to define the best path. Accordingly, we achieve fewer installed flow entries by edge sampling and forward the incoming flow to the less overloaded path, even if not the shortest one.

(2) Oddlab detects and avoids the faulty links inside the DCN and reroutes the elephant flows from the affected paths to deliver high availability.

However, Oddlab includes several stages. The controller starts to learn the topology and indicates the shortest paths between the edge host of the DCN that connects directly to the end-hosts. Then, periodically, the controller monitors the DCN links to determine residual bandwidth and the number of the installed flow entries active elephant flows. Compared to Sieve [104], Oddlab leverages the number of effective elephant flows inside the DCN to predict the future state of the path. Therefore, Oddlab eliminates the need to redirect elephant flows frequently to achieve better FCT values.

The overall aim of Oddlab is to deliver a deployable, light, and yet effective load balancing framework that works based on SDN in symmetric and asymmetric DCN topology without altering network components or TCP packets. We will introduce our proposed model for detection the faulty links in Chapter 4.

In Oddlab, we defined our problem for the initial flow scheduling to avoid elephant flow rerouting as follows:

Definition 15. *Problem* (Oddlab Adaptive Flow Scheduling (OAFS): *Given a multi-commodity DCN flow* $G = (V, E, b, El\_e)$, *where each edge* $(u, v) \in E$ *has*

available bandwidth $b(u,v)$ besides the number of active elephant flow $El\_e(u,v)$, the goal is to choose the path from V_s to V_t that contains the minimum number of elephant flow with maximum residual bandwidth.

3.6.1 Oddlab: Problem Formulation

The process of choosing the path should be efficient so that no rerouting is needed frequently to handle the congestions. Thus, in Oddlab and based on the multi-commodity problem, we aim to find the least congested path so that the maximum flow cumulative throughput $total\_thr$ for each non-fixed demand D_i commodities kc, as accomplished in Equation 3.14.

$$total\_thr = Max \sum_{i=1}^{ks} D_i \tag{3.14}$$

The process of path p choices by the SDN controller for each pair of nodes satisfying the following constraints: (i) the throughput l_e/C_e of every link e between aggregate and the core switch layer should not be less than (potential link failure ratio) h_th (Equation 3.15), where l_e is the current load, and C_e is the physical link capacity; (ii) the path should have the minimum number elephant flows P_el out of all available paths' elephant flows El_e (Equation 3.16); (iii) maximum path P_b residual bandwidth from source s_i to target t_i (Equation 3.17).

$$\frac{l\_e}{C\_e} \leqslant h\_th \tag{3.15}$$

$$P\_el = Min(El\_e) \tag{3.16}$$

$$P\_b = Max(C\_e - l\_e) \tag{3.17}$$

The first constraint is to detect the potential link failure at core links so that the path with this link will be avoided [1]. This restriction is based on the DCN utilization status, determined by the edge switch load odds ratio. Secondly, instead of estimating each incoming flow's demand (D_i), Oddlab will filter the path with the least number of elephant flows. So that the elephant flow collisions are avoided as possible. The third constraint states that the chosen path p has the maximum residual bandwidth among all the available paths p_n.

3.6.2 Complexity Analysis

Both problems of initial Sieve and Oddlab (SAFS in Definition 6 and OAFS in Definition 15) try to find the best path for the incoming flows. Hence, we combined them as a related problem to prove them as NP-complete problems. We need to construct a special case of the problems with a particular structure and reduce it to a known NP-hard problem. The other problem (SEFR) will be considered through the actual data transferring period $[\tau, 1 + \tau]$.

Theorem 16. *Both SAFS and OAFS are NP-complete.*

Proof. Both problems have the same decision problem: For SAFS, given $G = (V, E, b)$, is there a set of paths (SET) between V_s and V_t that maintains the best bandwidth b for each flow demand earlier than t? Given such a set of paths, it will be easy to verify their available bandwidth b and obtain the

[1] Oddlab link failure definition and detection model is introduced in Chapter 4

maximum value among them so that the flow can avoid the most congested path. Similarly, for OAFS, given $G = (V, E, b, El\_e)$, is there a set of paths (SET) between V_s and V_t that maintains the maximum residual bandwidth b besides minimum elephant flows El_e for each flow demand earlier than t? Given such a set of paths will simplify the verification process of obtaining and comparing the maximum residual bandwidth besides the minimum active elephant flows values for each path to get the best available path. All these operations can be done in nondeterministic polynomial time. Hence, both SAFS and OAFS are in the class of NP.

Now we prove that both SAFS and OAFS are NP-hard. The special case of both problems is related to the problem of single-sink unsplittable flow problem (SSUSF), which is proved to be an NP-hard problem in [84]: given a network graph $G'(V', E')$ with limited bandwidth capacity on each edge with a certain amount of demands D_j would like to be routed from source node s_i to destination to a unique destination sink node t. The goal is to maximize the flow route profit r_j without exceeding the edge bandwidth capacity constraints so that each routed flow will be placed at the best available path. Tow maximization standards have been identified in the SSUSF problem: (i) maximizing the total number of the routed demands and (ii) maximizing the throughput of the satisfied routed demands. Our proof is based upon the SSUSF throughput instance to find a different set of paths from to t with the maximum bandwidth that is not exceeding the edge capacity for each flow demand (i. e., $r_j = d_j$). Likewise, finding a solution for the SSUSF throughput instance can solve both SAFS and OAFS with the same structure. This problem conversions can be done in polynomial time. Thus, SAFS is an NP-complete problem.

Furthermore, the OAFS problem of finding the path that holding minimum active elephant flows and best residual bandwidth is similar to the convex optimization problem: which is a minimizing function $fc(x\_r)$ has the objective to find $x\_r \in$ feasible set CR that minimizes $fc(x\_r)$ subject to other constraint functions [14]. This mathematical problem is proved to be an NP-hard problem in general [72]. We can transfer the OAFS problem into a convex optimization problem in polynomial time. Suppose we can find a set of different paths from source node s to destination sink t in a graph $G(V, E)$ with minimum active elephant flows (convex function) and bounded by the maximum residual bandwidth of each path (constraint function). In that case, we can assign the best available path with minimum flow scheduling cost. Since both functions can be done in polynomial time, therefore, OAFS is an NP-complete problem.

□

Theorem 17. *SEFR is NP-complete.*

Proof. The decision of the SEFR problem is: Given a multi-commodity DCN flow graph $G = (V, E, El\_e)$, with a certain number of active elephant flows $El\_e(u, v)$ at the edge switches, is there a strategy to select at least fraction of these flows and reschedule them to other available paths from V_s to V_t successfully within time slot $[\tau, 1 + \tau]$? For each rescheduled elephant flow, a list of all available paths between the V_s and V_t each with a residual bandwidth. This process can be verified by checking if the new reserved bandwidth can handle the new flow without affecting the current flows within the time interval. Then, it will be easy to count and reschedule only a fraction of the flows from the bottlenecked port based on the number of total

elephant flows found on the bottlenecked edge to its residual bandwidth and total capacity. Thus, SEFR in the NP class.

SEFR problem is the same as the NP-hard problem (Vehicle Rescheduling Problem (VRSP)), which is defined based on graph $G(V, E)$ to find a new route for the vehicles of capacity Q that minimize the total scheduling cost from V_s to V_t [88]. Suppose we know a solution for the VRSP by naming the new feasible path that can handle the rerouted vehicles and minimize the total scheduling cost; we have the solution for SEFR with a particular case and vice versa. This conversion can be done in polynomial time; therefore, SEFR is in the NP-hard problem, and since SEFR is in the class of NP, it is NP-complete.

□

3.6.3 Oddlab SDN-based Adaptive Model

One of the Oddlab objectives is to identify the potential faulty links based on DCN traffic only. To achieve such a mission, we observe and correlate two subsequent events inside DCN between the edge switches (traffic source) and core switches (intermediate nodes). The first event includes the DCN utilization state estimation, calculated based on the traffic passing through both upstream ports of the edge switches. The second event involves presence of core switches links within an underutilized state (i. e., passing throughput less than the predefined threshold). We leave the details of detecting the faulty links to Chapter 4.

3.6.4 Oddlab Adaptive Flow Scheduling

The adaptive method enhances the scheduling performance by avoiding hashing collisions and choosing the best paths with minimum flow entries for active elephant flows besides better residual bandwidth. Consequently, we define the flow entries with 5-tuple $<$ `IP protocol, src port, src IP, dst port, and dst IP` $>$ based on OpenFlow protocol. The Oddlab controller reacts in two ways to the *packet_in* requests based on the estimated *Odds* value of the edge switches. Accordingly, when the DCN is not fully utilized (i,e., $Odds \leqslant 1$), the *packet_in* reaction regarding the *best_path* will be based on finding the path with less installed elephant flows and high available bandwidth as shown in Algorithm 3. In this case, the $maxfnum\_of\_path$ from Algorithm 5 and $bottleneck\_of\_path$ in Algorithm 2 will be invoked to indicate the best available path between end-hosts (*best_path*). To estimate the number of the active elephant flows, flow entry statistics obtained from the `EventOFPFlowStatsReply` function will be collected and stored into the graph $G = (V, E)$. Later, only flow entries with data transfer speed $\geqslant$ 50 Kbps [59] will be counted as an active elephant flow entry ($fnum$). To obtain the active elephant flows, we manage to calculate each flow byte consumption ($flow\_byte$) and divide it by the duration of that flow ($flow\_duration$) [59]. So that each link $e \in G(V, E)$ will have the number of active elephant flows ($fnum$). In this step, the path containing the minimum active flows will be considered the *best_path*. Next, as explained in Algorithm 2, the free bandwidth values ($free\_bw$) of all paths gathered and stored into the graph $G = (V, E)$ based on the `OFPPortStatsRequest` function. Finally, the adaptive flow scheduling method will choose the path with the least active elephant flows and

the lightest loaded as the *best_path*. This method will ensure that each flow (regardless of the flow demand) will get the best possible path (*best_path*), but not necessarily the shortest. Consequently, most flows will not be agglomerated into a single short path. Accordingly, in Oddlab, the flow load balancing equation is as follows (Equation 3.18):

$$U(u,v) = \frac{\sum_{i=1}^{kc} fe_i(u,v)}{mf(u,v).(mb(u,v))} \tag{3.18}$$

Where mf presents the $maxfnum\_of\_path$ between u and v node, and mb is the $bottleneck\_of\_path$ value between the same nodes.

Compared with Sieve, Oddlab leverages the number of effective elephant flows inside the DCN to predict the future state of the path. Therefore, Oddlab eliminates the need to redirect elephant flows frequently to achieve better FCT values.

3.7 Complexity Evaluation

We estimate the time and space complexity of Oddlab and Sieve by considering the worst-case scenarios regarding DCN density (number of monitoring switch ports k) and the average number of elephant flows $|El\_e|$ inside the DCN traffic. Our main proactive method built upon ECMP hashing is determined locally at the data plane level of DCN switches. Therefore, the ECMP time complexity ECMP is $O(1)$. In Oddlab, the primary adaptive traffic scheduling algorithm depends on the collected port states to determine the best paths (`EventOFPFlowStatsReply` and `OFPPortStatsReply`). Since we choose not to reroute the elephant flows in this step, the time com-

Algorithm 2: Handling of port information for DCN switches.

Data: *G=(V,E) , path, min_bw, P_r, dpid_list, speed*
Result: *mini_bw_link*

```
healthy_links[dpid][prt_no] = [ ]
foreach P_r do
    Function OFPPortStatsRequest(dpid_list):
        Function save_free_bw(dpid, prt_no, speed):
            free_bw =capacity-speed
            G[j][i][j][i+1](bw) ← free_bw
            return G[j][i][j][i+1](bw)
        return G[j][i][j][i+1](bw) (src_prt) (dst_prt)
        Function min_bw_links(G(V, E), path, min_bw):
            // To estimate the path bottleneck.
            for i in len(path)-1 do
                bw = G[j][i][j][i+1](bw)
                mini_bw_link = min(bw, min_bw)
        return mini_bw_link
```

Algorithm 3: Oddlab adaptive flows scheduling.

Data: *G=(V,E) , src_IP, dst_IP, src_prt, dst_prt, min_bw = capacity, max_bw = 0, max_fnum* = 0, *k, shortest_p*= { }, global(*Odds, af_links, proac_ports*)
Result: *best_path*= [ ], *failure_alert*

```
while Odds ⩽ 1 do
    for i in (0, k) do
        if shortest_p [i]= shortest_paths (G, src_IP, dst_IP) then
            for j in shortest_paths do
                max_fnum = maxfnum_of_path (G, j, max_fnum)
                fnum_of_paths (src_IP, dst_IP) ← max_fnum
                min_fnum_path = min(fnum_of_paths (src_IP, dst_IP,max_fnum))
            max_bw_of_paths = 0
            for fnum in fnum_of_paths do
                min_bw = bottleneck_of_path (G, j, min_bw)
                if min_bw > max_bw then
                    max_bw = min_bw
                best_path = fnum
return best_path
```

Algorithm 4: Path flow number estimation.

Data: *G=(V,E)* , *path*, *max_fnum*, global(af_links)
Result: *max_flownum*

```
for i in len(path)-1 do
    src_prt = G[j][i][j][i+1](src_prt)
    dst_prt = G[j][i][j][i+1](dst_prt)
    fault_src_itr = G[j][i][j][i+1](fault_src)
    fault_dst_itr = G[i][j][i][j+1](fault_dst)
    fnum = G[j][i][j][i+1](fnum)
    // This condition will avoid detected links when fault
       iteration reaches the fault_iter threshold.
    if (i in af_links and src_prt in af_links[j] and dst_prt in af_links[i] and fault_src
     ⩾ fault_iter and    fault_dst ⩾ fault_iter) then
        max_flownum = ∞
    else
        max_flownum = max(fnum, max_fnum)
return max_flownum
```

plexity is $O(k^2)$. In contrast, the time complexity of Sieve will be $O(|El\_e|(\frac{k}{5}) + k)$. Since Sieve constantly monitors the elephant flows on the upstream ports of edge switches and tries to reschedule them to another available path.

As for the space complexity, the controller memory should maintain the following variables, including the number of active elephant flows and the residual bandwidth of each port. Hence, Oddlab space complexity is ($O(k^2)$. On the other hand, Sieve space complexity holds information of the detected elephant flows on the upstream ports of each edge switch ($\frac{k}{5}$) besides the obtained ports stats (i. e., residual bandwidth) to find the proper path. So, Sieve space complexity would be ($O(|El\_e|(\frac{k}{5}) + k)$).

Moreover, we proved that the sampling process with two buckets on edge switches reduced the controller overhead by half with the means of eliminating the *packet_in* requests to the controller at a time interval of ($t \rightarrow$

∞) [104]. Additionally, we adopted a commercial DCN introduced in [11] with 10,000 end-hosts and 1,000 flows per second for each host to prove the simplicity and effectiveness of the proposed method when increasing the number of DCN end-hosts. However, within 2 ms as the median arrival time for each flow, we anticipate that the Oddlab controller can handle $\frac{10{,}000 \times 1{,}000}{2} = 5{,}000{,}000$ (i.e., 2,500,000 *packet_in* requests only) [104]. This is considerably less than the number of flows a single controller can manage per second (more than 12 million requests [25]). Consequently, Sieve and Oddlab implementation is highly achievable in hardware such as NetFPGA OpenFlow switches [27] owing to uncomplicated arithmetic operations and less overhead.

Note that Sieve analysis shows that the DCN information update in fat-tree $K-4$ topology took up to 35.84 μsec. from sending the static request (`OFPPortStatsReply`) until receiving and processing it [104]. In contrast, Oddlab will double this period since we collect throughput and active elephant flow numbers inside the DCN (`EventOFPFlowStatsReply` and `OFPPortStatsReply`), but without the extra complexity regarding frequent elephant flow redirection in the case of symmetric DCN topology.

3.8 Experimental Results

Sieve and Oddlab operate as a python application within the Ryu [79] controller utilizing mininet [62] as a real-time network emulator to establish a multi-rooted $K-4$ fat-tree DCN (e.g., Figure 1.1) besides OpenFlow protocol 1.3.1 as a communication protocol. The testbed topology includes 16 hosts interconnected to twenty 4-ports OpenFlow switches and four pods with four core switches. To maintain the connection precision in the real-

time environment in symmetric DCN topology, we set the link capacity through the DCN to 10 Mbps. As illustrated in Figure 1.1, the controller is connected to each DCN switch to collect the required network statistics to maintain the network flows. Our DCN deployment has been implemented using a commodity PC with an Intel Core i5-8400 2.80 GHz CPU, 16 GB RAM running Ubuntu 16.04.

Extensive experiments were conducted on a wide range of traffic patterns with synthetic and real workloads to prove the proposed methods (Sieve and Oddlab) feasibility without altering any network component (i.e., hosts or switches). We demonstrated that Sieve and Oddlab deliver noticeable improvements than existing hash-based ECMP flow scheduling, Hedera, and PureSDN in bisection bandwidth, link utilization, packets loss and Round-trip Times (RTT), overall average FCT, with less elephant flow loss risk.

3.8.1 Average Bisection Bandwidth

One of the most important features of the multi-rooted topology is to grant full bisection bandwidth among the connected end-hosts. But without an efficient flow scheduler, the DCN bisection bandwidth will decrease dramatically as it appears in ECMP due to flows collisions [27]. In the bisection bandwidth experiment, we compare the obtained results to the ideal network situation using “NonBlocking” where a central switch is connected directly to all end-hosts.

We conducted two benchmark tests. The first one includes the average bisection bandwidth and link utilization test. The second test is designated for the FCT examination. We accomplished the tests on the 16 hosts

of the DCN topology, where flows were generated using TCP `Iperf` [42] according to the traffic patterns described earlier. We repeated the average bisection bandwidth test for ten independent runs for more reliable and realistic results. Each run lasted for 60 sec., during which the average bisection bandwidth is accumulating using `BWN-NG (Bandwidth-NG)` [16] on the edges downstream as implemented in Hedera [27] and [59].

As shown in Figure 3.5, the obtained results confirm that the static hashing method (ECMP) achieved the lowest throughput due to the flows collisions produced from scheduling multiple elephant flows at the same path. Oddlab outperforms Hedera, ECMP, and Sieve in `random` and `stag0.1_0.2` where most of the traffic (i.e., 70% in `stag0.1_0.2`) will be among different pods in these patterns, which implies that most of the flows will benefit from the bisection bandwidth granted by the multi-rooted DCN topology. Conversely, Sieve sustains an average bisection bandwidth in overall patterns compared with other adaptive elephant flow rescheduling methods such as Hedera due to the frequent elephant flow rescheduling strategy. However, we remark that the average throughput in the rest patterns is slightly improved since the resource contention and the collision rate decreases when the traffic is within the same edge switch or in the same pod. Still, the entirely based SDN method (PureSDN) outperforms in `random` and `stag0.1_0.2` patterns but at the high controller overhead expense. Since it depends on the central controller for the adaptive flow scheduling, from another aspect, Oddlab scheduling depends on central iterative optimization with the help of ECMP, which makes the proposed method operate with less burden on the controller and delivers a notable improvement in average bisection bandwidth.

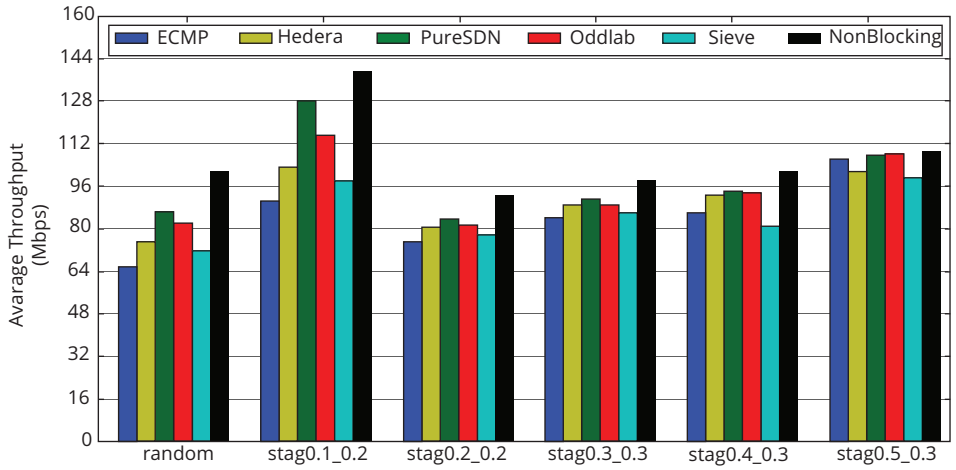


Figure 3.5: Average bisection throughput for the five TE methods under different traffic patterns.

3.8.2 Link Utilization

It refers to the average DCN link consumption compared to the actual capacity. In this aspect, we will determine how the proposed scheduling methods will achieve moderate link utilization so that links will not become prone to congestions. Figure 3.6 illustrates the commutative link utilization for Sieve and Oddlab compared with the other scheduling algorithms under different traffic patterns. Obviously, the majority of the links were underutilized in the case of ECMP.

For instance, the `random` and `stag0.1_0.2` patterns for ECMP showed that 50% of the links (y-axis) were occupied with less than 20% (x-axis) of the total capacity due to entirely random flow scheduling and flows' collisions. Compared with other methods, Oddlab performs at the medium level in various traffic patterns. A closer look at `stag0.1_0.2` pattern, we can anticipate that 70% of links were occupied by 75% of capacity in Hedera, 76% in Sieve, 82% in Oddlab, and 90% in PureSDN. Interestingly, thanks to

the frequent elephant flow rerouting, we remark that the link utilization in Sieve had not reached the peak consumption remarkably in the whole traffic patterns. For instance, 90% of the links in `random` pattern occupied 95% of the total capacity compared with the other methods where the utilization reached 100% of the capacity. We conclude that most of the links in the case of PureSDN reached the overutilized situation; that is what makes it prone to more congestion, thus increasing the flows completion time.

3.8.3 Flow Completion Time (FCT)

It denotes the efficiency of the flow scheduler algorithm to deliver different flows rapidly. For Sieve and Oddlab, we will show the results of the overall Average Flow Completion time (AFCT) beside FCT values for the mice flows for each end-hosts.

In Figure 3.7 (`random` pattern), we notice that Sieve achieved 23 %, 18,9 %, and 53,8 % less completion time compared to ECMP, Hedera, and PureSDN, respectively.

For Oddlab, we will show the results of the AFCT of the transferred flows for each end-hosts. As shown in Figure 3.7 (`random` pattern), Oddlab reduces the AFCT up to 30%, 25.7%, 62%, and 5% compared to ECMP, Hedera, PureSDN, and Sieve, respectively. Notwithstanding, PureSDN relies on the wildcard flow entry rules for TCP flows, whereas flows that belong to the same source and destination IP addresses will be scheduled on the same path. Hence, the selected path in the PureSDN case will be highly saturated. On the other hand, mice flows are known to be the most affected flows by transmission delay. Figure 3.8 shows that both Sieve and Oddlab

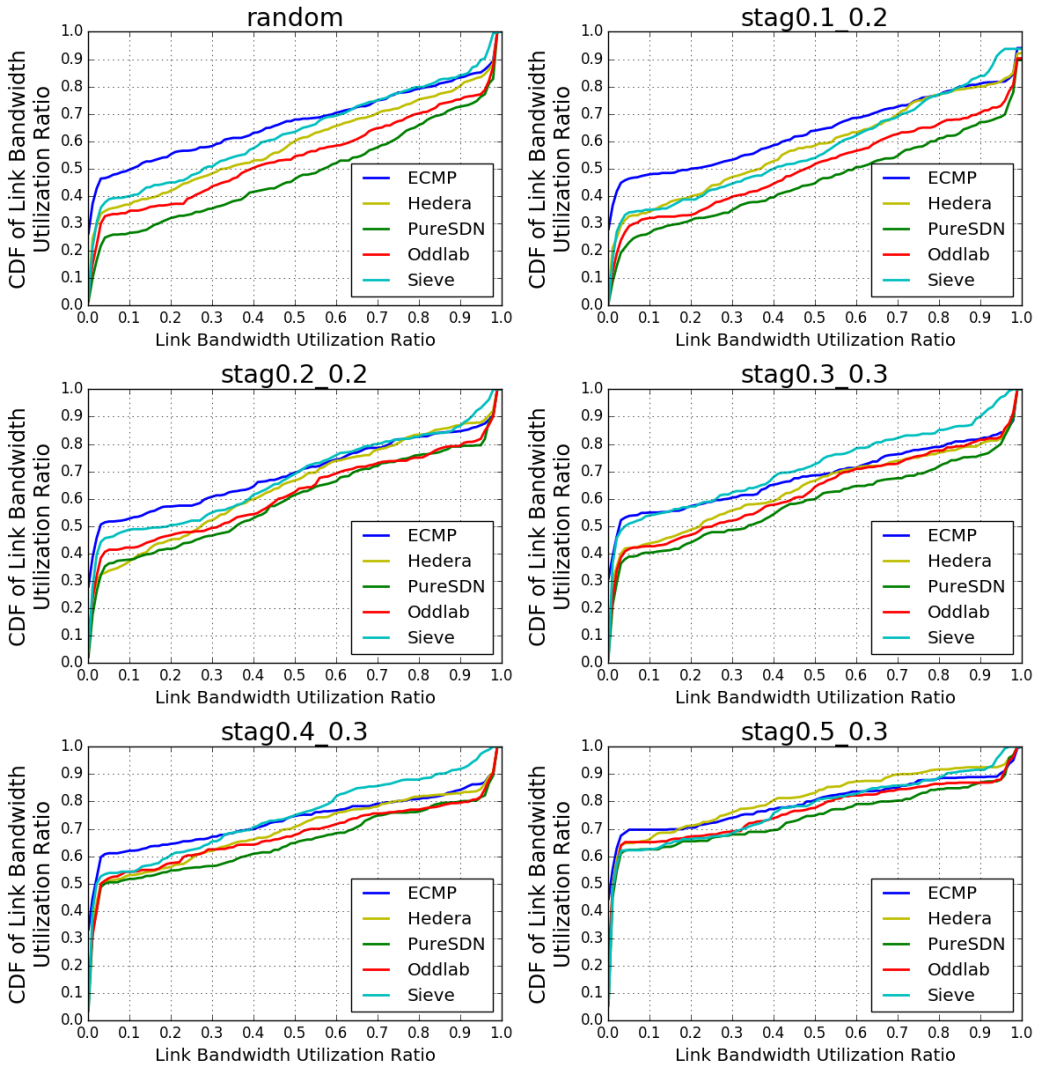


Figure 3.6: CDF of link bandwidth utilization for the five TE methods under different traffic patterns.

significantly reduce FCT values for the mice flows within a accumulative flow size $\leqslant$ 50 KB [77].

We remark that Hedera and ECMP perform almost close to each other. Since Hedera scheduling is based on ECMP for mice flows, only reroute elephant flows to the best path once the flows reach 10% of the link capacity. Consequently, the flow collision is inevitable, which necessarily delays the arrival of the flows remarkably. As for Sieve results, the frequent elephant flow rerouting reduces the delay for the mice flows. This procedure, will affect the arrival time of elephant flows. Not to mention the newly added flow entry rules into DCN switches to manage the rerouted elephant flows. In the Oddlab case, initially, the adaptive scheduling tries to avoid congested paths without the necessity for elephant flow rerouting. It can be noted in Fig. 3.7 that ECMP and Hedera outperform in average FCT reduction in the traffic patterns `stag0.2_0.3 and stag0.4_0.3` where most of the traffic is inside the same pod. In this case, Oddlab may choose farther paths based on shorter path conditions, which may affect the flows' arrival time.

To sum up, the symmetric DCN topology results conclude that the proposed flow scheduling of Oddlab outperforms in delivering better average throughput, moderate link bandwidth utilization, and reduced average overall FCT.

3.8.4 Packets Loss and Round-trip Times (RTT)

It indicates the number of lost packets due to flow congestions and collisions within the DCN fabrics. In Sieve and Oddlab, we adopted edge flow sampling and proactive paths built upon ECMP hashing. We conducted the packet loss experiment as illustrated in [59]. This experiment aimed

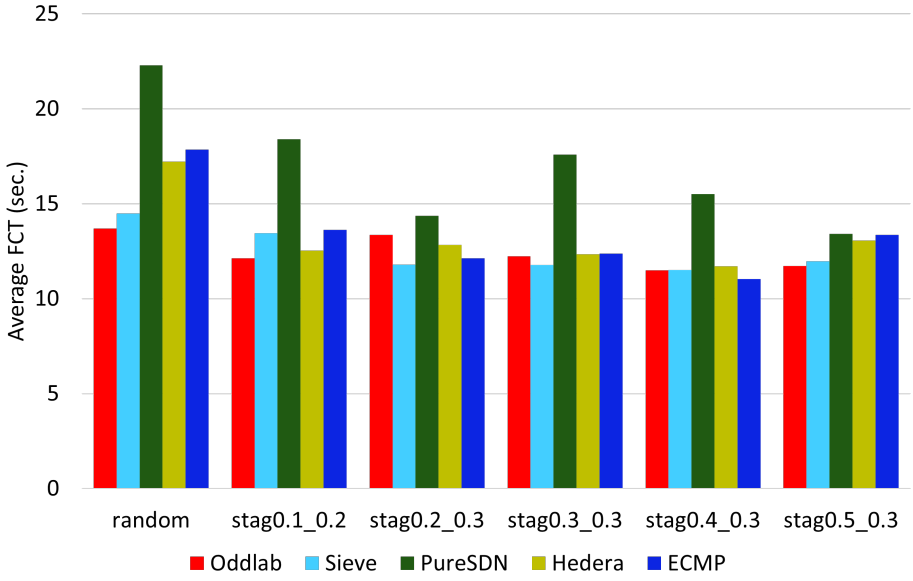


Figure 3.7: Average overall FCT for the five TE methods under different traffic patterns.

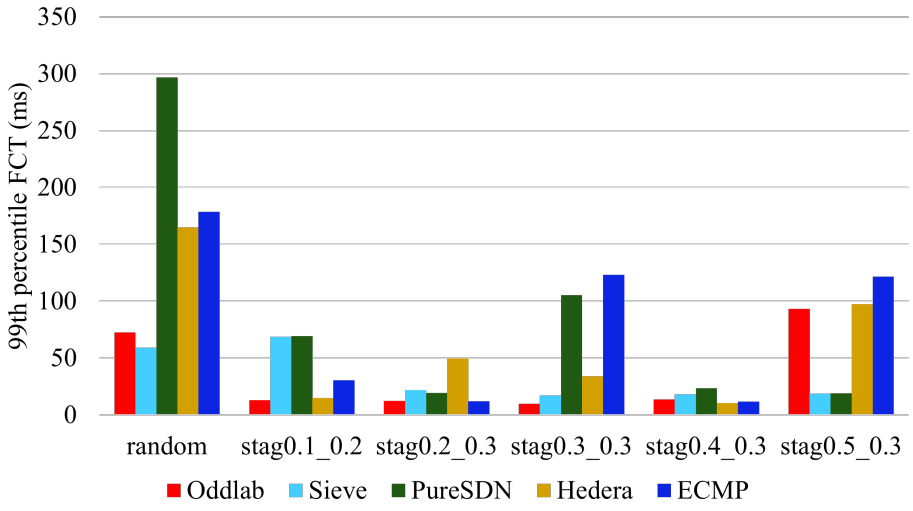


Figure 3.8: The 99^{th} percentile FCT for the five TE methods under different traffic patterns.

to flood all DCN links by elephant flows initiated from each end-host based on the defined traffic patterns to examine how the different TE methods will handle the flows to overcome packet losses. To investigate packets losses, we employed the Linux ping command. Ping relies on sending various Internet Control Message Protocol (ICMP) echo requests packets (600 successive packets in our experiment) between each DCN end-hosts and waiting for the ICMP echo replies. Then, ping records the packets loss besides a statistical summary of the results in each end-host, including minimum, maximum, the mean round-trip times (RTT), and standard deviation of the mean. This experiment included RTT measurement to show the end-to-end responsive delay (signal time plus the amount of time to acknowledge that signal in the receiver) within the DCN.

In this experiment, the network is overwhelmed with elephant flows to represent the worst-case scenario into a production DCN. We repeated the experiment 10 rounds for each method, and we reported the averaged results of the successive packets loss and RTT as shown in Figure 3.9 and Figure 3.10, respectively.

As shown in Figure 3.9, the packets loss results demonstrate that the PureSDN method achieved less loss since ICMP packets have been scheduled adaptively in all experiment runs. We remark that Oddlab outperforms Hedera and ECMP in case connections spanned all DCN topology layers as depicted in (`Stag0.1_0.2 and Stag0.2_0.3`). On the other hand, despite edge flow sampling, Oddlab outperforms Sieve, Hedera, and ECMP, respectively, in the number of lost packets almost in all patterns. The reason is that, in Oddlab, we did not reschedule the flows regularly as adopted in Sieve. Alternatively, the adaptive flow scheduling model initially determines the best available path based on the DCN states (active elephant

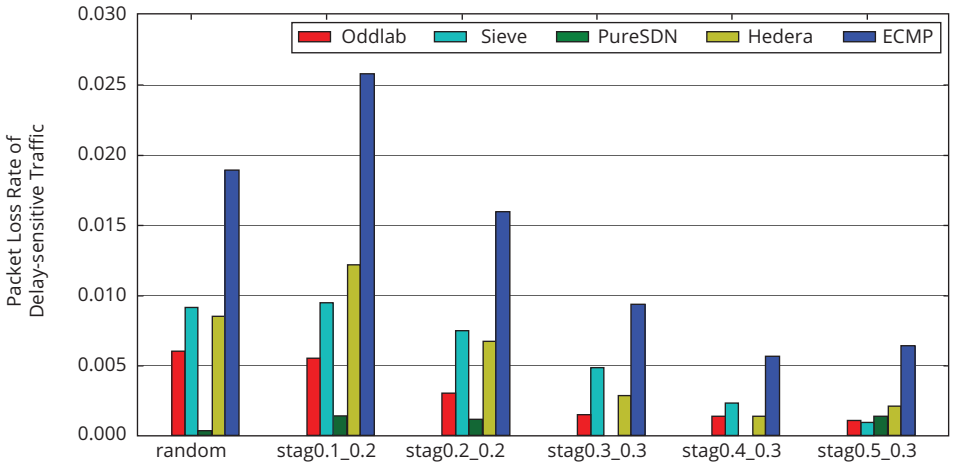


Figure 3.9: Average packets loss for the five TE methods under different traffic patterns.

flows and residual bandwidth). In Sieve, the results are similar or better than Hedera (e.g., `Stag0.1_0.2 and Stag0.5_0.3`) and outperform ECMP in all traffic patterns. In Sieve, we only considered TCP elephant flows in rescheduling decisions since ICMP and User Datagram Protocol (UDP) packets are sensitive cannot be reordered at the destination host. Nonetheless, the packets loss in Sieve is still tolerable.

Furthermore, we illustrate the obtained RTT values reported by `ping` tests in Figure 3.10. The results confirm that the DCN in Oddlab is the least congested compared with the other methods (except PureSDN), and ECMP is the most congested DCN due to the flows' collisions. The results confirm that Sieve got the slightest RTT values than Hedera since it reschedules elephant flows earlier than Hedera, which remains until the flow reaches 10% of the capacity.

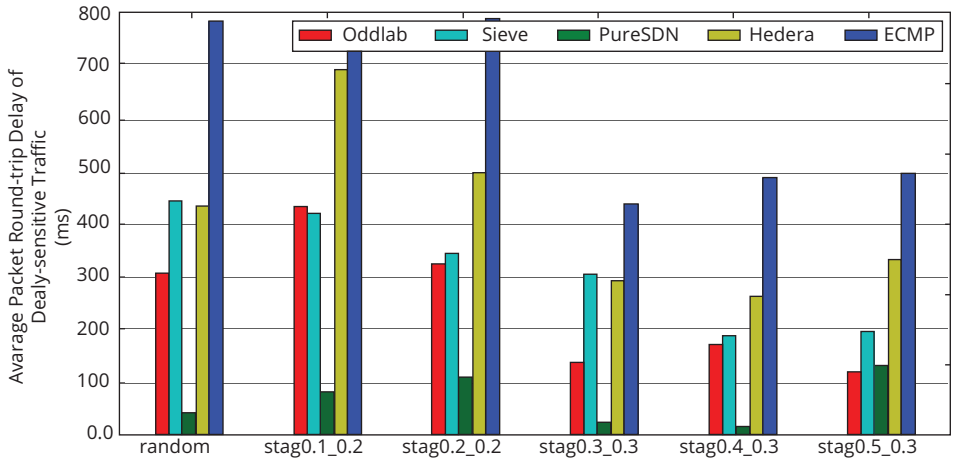


Figure 3.10: Average round trip delay for the five TE methods under different traffic patterns.

3.8.5 Sieve and Oddlab Risk Analysis

Elephant flows have been handled differently in Sieve and Oddlab. This evaluation metric utilizes our proposed risk analysis framework (as presented in Chapter 2) to indicate the amount of the elephant flows blocked rate exposed to risk when such methods are in charge of flow scheduling in a productive DCN.

- **Estimating the proper probability distribution functions**: The analysis results show that the available bandwidth of Sieve and Oddlab followed the Geometric distribution (G-D) with the probability value (pv) of Sieve = 0.02281, Oddlab = 0.02826, respectively. Figure 3.11 and 3.12 show the P–P plots (probability–probability) that prove how the empirical values of the available bandwidth for Sieve and Oddlab for the obtained available bandwidth values.

  Table 3.1 presents the results of the KS null hypothesis testing on the throughput measurements of Sieve and Oddlab. The throughput test-

ing showed that both methods followed the Geometric distribution (G-D) based on P-value, and the acceptable critical value was (α= 0.02).

Furthermore, results of AD test (Table 3.2) confirm that all samples follow the Geometric distribution based on the values of AD statistic (AD^2) and the obtained critical values estimated on the AD critical value (α = 0.02).

As for the risk factor (margin of bandwidth measurement error), the testing indicated that Sieve and Oddlab followed the simple Discrete Uniform distribution (D-U). Accordingly, we used the following normalized values regarding the a_r and b_r as appeared empirically, Sieve = [13, 50] and Oddlab = [16, 41], respectively.

- **Risk Analysis of Elephant Flows Blocked Rate Predictions Using Monte Carlo Simulation**: We proved by implementing our risk analysis framework that the elephant flows rerouting process in Sieve decreased losses probability by 45.74%, almost similar to the PureSDN result (44.43%). As we illustrated in link utilization criteria, Sieve can reroute elephant flows whenever free bandwidth is available. Oddlab, on the other hand, achieved 52.8 % of elephant flow loss probability. Histograms represent the risk analysis outcomes of the Sieve and Oddlab (Figure 3.13 and Figure 3.14). The mean values (blue line) precede the median values (red line); besides, the skewness values indicate that all methods follow positive skewness and are right-skewed. We obtained kurtosis degrees for each histogram to identify which one produces more outlier values. We found that the histograms follow the platykurtic distribution since the kurtosis is negative compared with the Normal distribution. Therefore, the expected behavior for

the algorithms is to produce fewer outliers values as shown in Table 3.3.

- **Value at Risk (VaR) analysis**: As depicted in Figure 3.15, the Significant loss rate in Sieve is 94 MBps compared with Oddlab, which were 95 MBps at 95% confidence interval. Nevertheless, due to the higher loss uncertainty for Oddlab, it will be more prone for elephant flow losses than the Sieve TE method. These values represent the extreme loss rate value of the elephant flows over the DCN production lifespan.

- **Risk as expected of elephant flows loss**:

  Table 3.4 lists the expected elephant flow loss for Sieve and Oddlab compared with other exiting TE methods such as ECMP, Hedera, and PureSDN, respectively. We remark that Sieve reduces elephant flow losses due to the adapting elephant flow rerouting mechanism. Oddlab, on the other hand, comes in second place, but with the advantage of not rerouting the elephant flows.

TE method	KS critical values	P-Value	Distribution
Sieve	0.19267	0.069	Geometric
Oddlab	0.19267	0.051	Geometric

Table 3.1: KS test values for the available bandwidth samples for Sieve and Oddlab.

TE method	AD critical values	Statistic (AD^2)	Distribution
Sieve	3.2892	2.3427	Geometric
Oddlab	3.2892	2.4552	Geometric

Table 3.2: AD test values for the available bandwidth samples for Sieve and Oddlab.

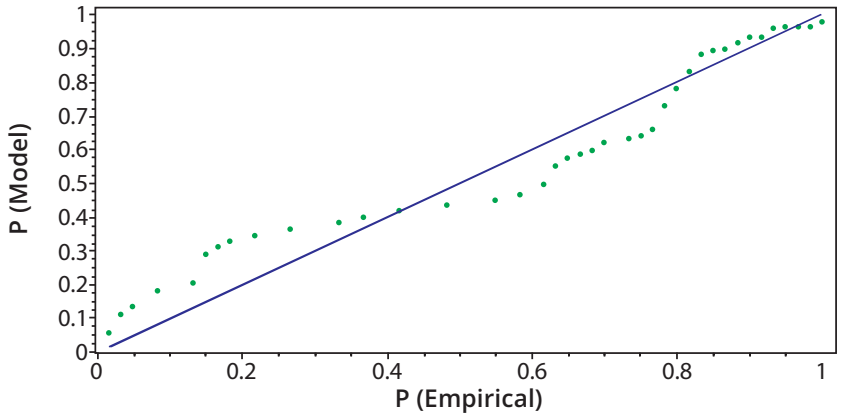


Figure 3.11: P-P plot of the Sieve available bandwidth value distribution fitting with the geometric distribution.

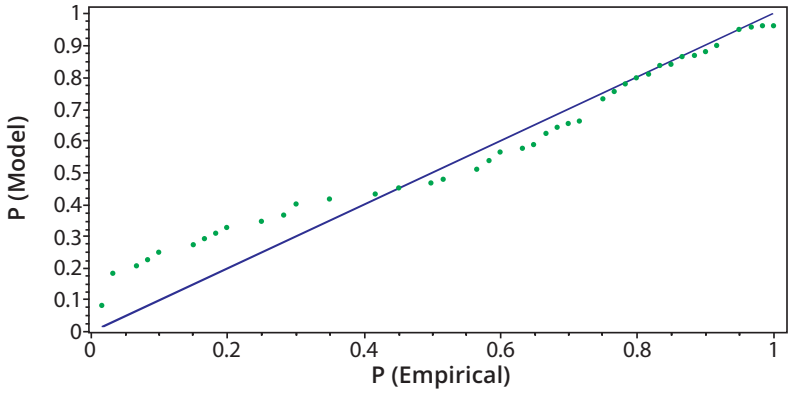


Figure 3.12: P-P plot of the Oddlab available bandwidth value distribution fitting with the geometric distribution.

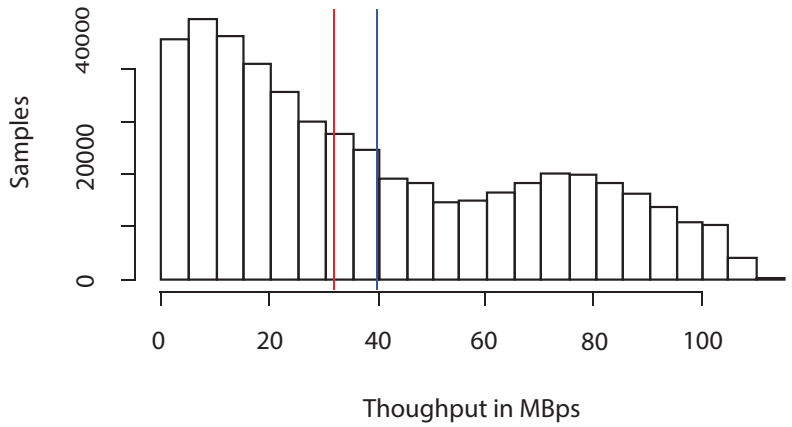


Figure 3.13: Histogram of Monte Carlo simulation for Sieve elephant flow loss rate

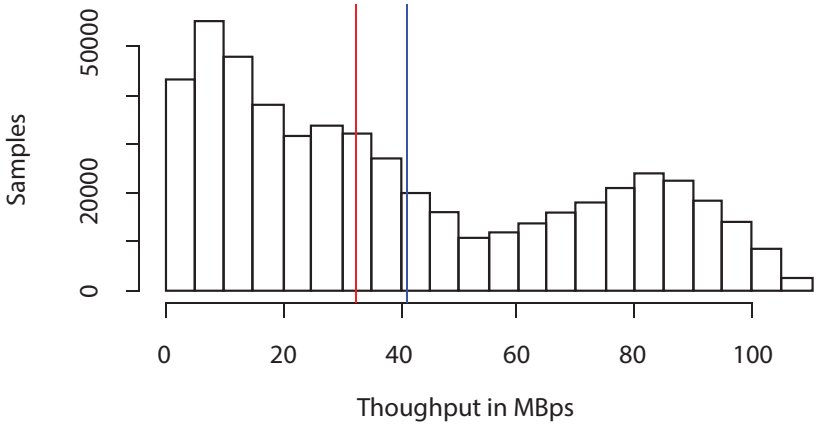


Figure 3.14: Histogram of Monte Carlo simulation for Oddlab elephant flow loss rate.

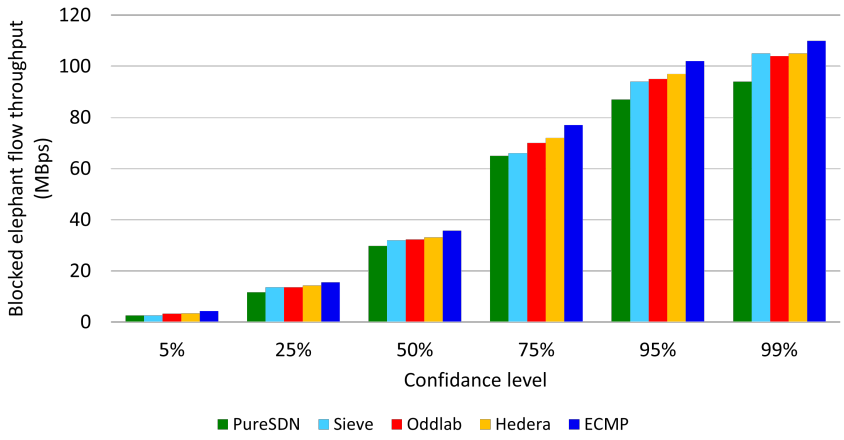


Figure 3.15: Different confidence levels of VaR analysis for the TE methods.

TE method	Mean	Median	Skewness	Kurtosis
Hedera	42.16	33.15	0.5026271	-1.130097
ECMP	44.95	35.70	0.4955835	-1.144383
PureSDN	37.93	29.75	0.4590237	-1.191083
Sieve	39.93	32.00	0.537531	-0.9663874
Oddlab	40.96	32.30	0.503721	-1.107166

Table 3.3: Descriptive statistics of the tested methods histograms.

TE method	Loss uncertainty	Significant loss	Expected loss
ECMP	62.83%	102 MBps	64.08 MBps
Hedera	56.76%	97 MBps	55.05 MBps
PureSDN	44.43%	87 MBps	38.65 MBps
Sieve	45.74%	94 MBps	43 MBps
Oddlab	52.8%	95 MBps	50.16 MBps

Table 3.4: The expected elephant loss for the examined TE methods.

3.9 Conclusion

This thesis proposed Sieve and Oddlab, a novel hybrid and deployable load-balancing approach that guarantees the QoS of multi-rooted DCN traf-

fic in symmetric topologies. We employed the proactive path mechanism besides SDN adaptive scheduling for flows scheduling that considers available bandwidth in Sieve and the active elephant flows inside the DCN to find the best paths in Oddlab. Accordingly, the most crucial finding of Sieve and Oddlab is that in leveraging the global knowledge of the DCN switches statistics on a single and central SDN controller to deliver promising results in symmetric DCN topologies. The performance analysis based on the law of large numbers revealed that the adopted flow sampling mechanism at the edge switches diminishes the SDN controller burden to half, reducing the waiting time for flow handling. Extensive experiments were conducted on a wide range of traffic patterns with synthetic and realistic workloads to prove Oddlab feasibility without altering any network component (i.e., hosts or switches). The obtained result of the average overall Flow Completion time (AFCT) confirmed that Sieve achieved 23%, 18,9%, and 53,8% compared to ECMP, Hedera, and PureSDN, respectively. For Oddlab, the results showed that Oddlab delivered noticeable improvements in bisection bandwidth, link utilization, packets loss, round trip delay, mice flow FCT, with an average overall reduction in FCT up to 30%, 25.7%, 62%, and 5% compared to ECMP, Hedera, PureSDN, and Sieve, respectively.

Furthermore, we applied our proposed risk analysis framework to investigate the blocked elephant flows rate. The results revealed that Sieve decreased the elephant flow losses probability by 45.74% almost similar to PureSDN (44.43%) compared to 52.8% achieved by the Oddlab scheduling procedure. The results also proved that the frequent elephant flows rescheduling can preserve elephant flows throughout, but with an acceptable transfer delay (i. e., average overall FCT). Hence, we conclude that the

proposed methods significantly improve DCN productivity and reduce the burden on the SDN controller.

Chapter 4

DCN Faulty Link Detection Model

"An experiment is a question which science poses to Nature, and a measurement is the recording of Nature's answer."

Max Planck (1858-1947)

4.1 Introduction

DCN is considered the central core infrastructure for data management in evolving infrastructures such as cloud enterprises, social networking services, and e-commerce. To keep the revenue of such investment, DCN service availability must always be maintained in all circumstances[67]. One of the most crucial issues in DCN fabrics is the links failures [29]. Consequently, it is not easy to maintain the symmetric situation for the duration of the network operations. Recently, the annual outage analysis survey from the Uptime Institute for 2021 states in [52] that most organizations (76%) have suffered from certain types of outages in the last three years. Still,

power, software and IT systems, and networks issues are the top causes of these outages by 37%, 22%, and 17%, respectively. Such DCN operation outages may cost the business a massive amount of money that could reach $1 million.

In DCN, the link failure can be classified into two categories based on the failure causes and duration: i.e., hardware and software failures (i. e., partial and complete failure). The most common type is the hardware failure caused by physical faults in line cards. In contrast, software failures are caused by software bugs or IOS hotfixes issues [29]. On the other hand, the hardware failure is known to be a long-lived failure and may require hardware replacement. Simultaneously, the software type is considered short-lived and could be resolved automatically like root guard in spanning tree protocol [29]. Accordingly, in Oddlab, we adopted both definitions of faulty links. However, link overload or some outage (short-lived failures) could occur in DCN core links. Therefore, as illustrated in Subsection 4.5, we treat such events based on the outage duration; whenever the link returns to its normal state, it will be removed from the faulty links list.

The DCN topology becomes asymmetric if any one of the failures occurs. A scheme such as ECMP was deployed to hash every flow to a different path to handle traffic congestions in standard DCN operations, considering the complete failure only without monitoring network stats. State-of-the-art approaches address such challenges by modifying the network components (switches or hosts) to discover and avoid broken connections.

This chapter proposes an SDN adaptive scheduling that considers the healthy paths, available bandwidth, and active elephant flows to find the best paths and avoid significant flow rescheduling. In Oddlab faulty de-

tection model, finding the faulty links begins by labeling the edge switches when the DCN is stressed with substantial traffic among different pods.

4.2 Research Motivation

In Sieve, to achieve better FCT, we attempt to reroute a fraction of elephants on an edge switch when this switch reaches a specific load threshold to a different path based on the residual bandwidth. This mechanism will produce further congestion and contention between the newly arrived flows and DCN topology asymmetricity in case of link failure.

For example, as shown in Figure 4.1, in time instance (T1), an elephant flow A originated from the host (H1) in pod 1 to host (H11) in pod 3. Since the links in core switches are the highly congested layers in DCN, therefore, in (T2), local congestion may occur between aggregation switch (Agg. 2) and (switch 3) in the core layer. Note that a severe bottleneck at the core later may lead to link failure [29]. In Hedera [27] and Sieve, elephant flows in Edge1-Agg.2 links are rerouted to another path in (T3). This decision may drive further congestion and delay currently active flow, particularly to the growing elephant flows. In (T4), elephant flow B originated from H4 in pod 1 to H9 in pod 3. When all the paths to the destination host (H9) are congested, the path containing the link (Agg.2- switch3) may be taken. In this case, TCP congestion control will reduce the number of packets passed through the congested link so that the edge switch may not bypass the rerouting threshold to reroute the congested elephant flow. Accordingly, we note that the current active elephant flows inside DCN should be considered for the incoming flows scheduling decision. This example clearly

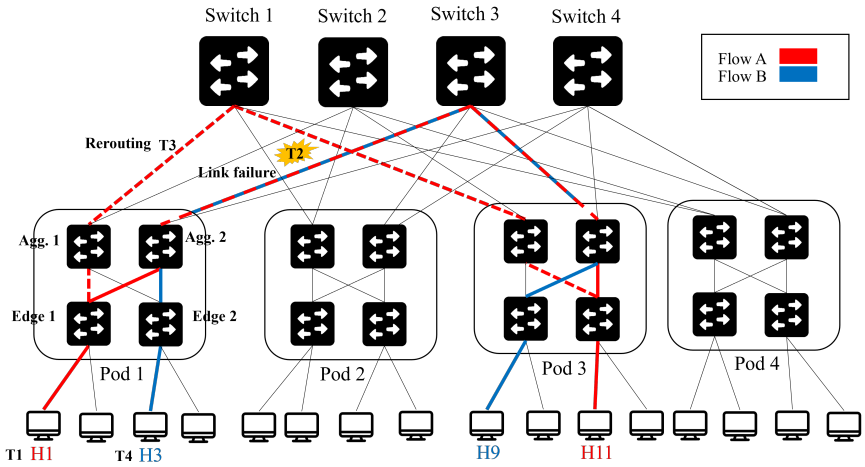


Figure 4.1: Flows collisions and rerouting in fat-tree DCN topology.

explains that the rerouting decision should avoid the possible link failure and avoid such failure in flow scheduling.

4.3 Literature Review

DCN suffers from various uncertainties in its operation, such as the traffic dynamics, failures, and the topology asymmetricity [106]. Many approaches have been proposed to address these challenges. In [9], Alizadeh et al. proposed CONGA, where the concept of flowlet (packet-level granularity) was introduced to achieve optimal flow distribution in asymmetric topology by collecting the congestion feedback from the switches. Flowlet is small chunks of a flow sent dynamically across the switch ports depending on the congestion feedback. This process affects mice flows' FCT and packet reordering in flowlet rerouting on congestion or failure detection. Still, the switch's hardware needs to be altered to provide the congestion feedback. Hermes, a congestion-aware load balancing technique proposed

in [106] works on packets routing and flow scheduling on congestions or failure occurs. In congestion detection, the method depends on ECN (Explicit Congestion Notification) and RTT. Although Hermes is deployable since hardware modification is not needed, all end-hosts in the DCN need to participate in the sensing process. Therefore, the sensing technique is challenging to accomplish without end-host aid. CAPS presented in [37] is an end-host-based local congestion-aware technique. The method includes three main modules: the packet encoder and the decoder on each DCN host, besides Random Packet Spraying (RPS). In CAPS, traffic flows are divided into mice and elephant flows, where elephants are scheduled using ECMP, while mice flows are scattered to all available paths based on RPS. This technique required changes in end-hosts in terms of software besides the availability of RPS switches. End-hosts have also been leveraged in SAPS [44] to handle flow scheduling in asymmetric topology by providing virtual symmetric paths to each flow using SDN controller and group tables. In SAPS, the elephant flows are identified based on the number of sent bytes inside each end-host. Hence, the shim layer of these hosts should be visible. The deployment of such a method is costly and may be restricted in cloud environments. DRILL proposed in [28] as a per-packet scheduling technique that relies on spreading the packets randomly to the shortest and least congested queues based on the power of the two choices approach. The technique has been evaluated under different loads on the leaf and spine switches in Clos DCN topology and required hardware changes at the level of the DCN switches. Recently, FlowFurl has been introduced in [82] as a flow level routing approach that works in an asymmetric DCN topology. The method reroutes the flows based on the ECN-bit state transferred between source and destination over the intermediate switches to

Table 4.1: Comparison of DCN load balancing methods.

Method	DCN changes	Granularity	Failure handler
ECMP [36]	×	Flow	Full failure only
Hedera [27]	×	Flow	PortLand protocol
CONGA [9]	Switches	Flowlet	✓
DRILL [28]	Switches	Packet	×
SAPS [44]	Hosts	Packet	✓
Hermes [106]	×	Packet / flow	✓
FlowFurl [82]	Hosts & switches	Flow	✓
Sieve	×	Flow	×
Oddlab	×	Flow	✓

detect and discover the healthy paths. FlowFurl requires modifications on the end-hosts since it is in charge of flow rerouting, besides the DC's intermediate switches.

Table 4.1 compares the most relevant DCN load balancing methods regarding DCN hardware or software components change (switches / end-hosts), load balancing granularity, and failure resilience.

In this research, we demonstrate the proposed solution (Oddlab) for detecting the faulty links in DCN topology based on the global DCN information of flow scheduling. Thus, we prove our proposed solution to serve DCN in both symmetric and asymmetric topologies.

4.4 Faulty Links Detection Mechanism

To detect and avoid the potential faulty links, we perform three steps on the SDN data plane. In these steps, we establish a temporal correlation between the loading state at the edge switches and the possible link failure at the core switches. The main components and steps of OddLab are

presented in Figure 4.2. The SDN controller represents the control plane, which contains the Oddlab's main functions and periodically collects network information. This information includes ports and flows stats based on OpenFlow protocol. Then, the obtained data was utilized to determine the best paths between every pair of end-hosts in the data plane and label the loaded edge switches. Accordingly, in step 1, the labeling process (red label in Figure 4.2) starts after the SDN controller detects in any monitoring period that there are specific amounts of traffic passing on the upstream side of the edge switches (ports 1 and 2). This step also involves the odds calculation, which indicates the number of edge switches that meet the utilization condition to the number of non-loaded switches to indicate whether the whole DCN is utilized or not. As depicted in Figure 4.2, three edge switches fulfilled the utilization condition, and only one in Pod 4 did not, which means DCN is in utilization state. Whenever the controller determines that most edge switches are sufficiently utilized, step 2 begins by looking for the faulty links whose load level falls beyond 1% of the total link capacity. In step 3, all the elephant flows found on the detected faulty links will be rescheduled to other healthy paths. Later, the adaptive flow scheduling model will be updated so that the incoming flows will avoid the affected paths. Ordinarily, the entirely failed links continue to convey less throughput while DCN is in the loading state (step 1). Consequently, to avoid these links even in the proactive paths, the weighting function of the ECMP hashing at the aggregate switches will be altered in step 4. Lastly, an alert will also be issued, including the affected links' information, so that the DCN administrator can treat the failed links.

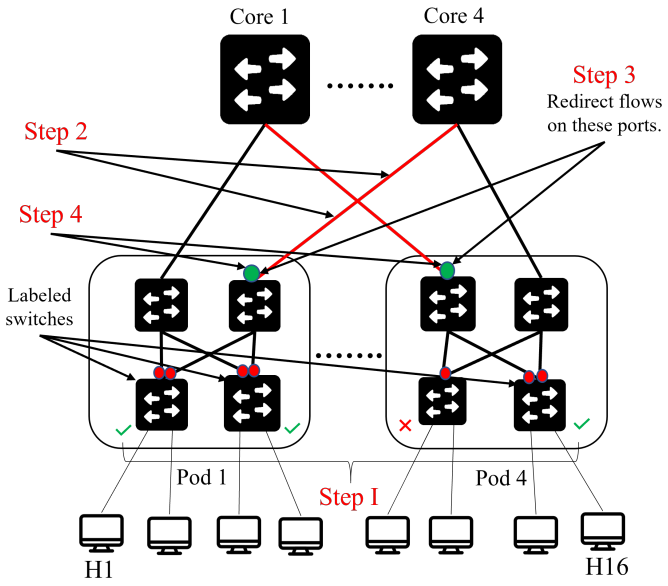


Figure 4.2: Oddlab approach on detecting the faulty links on $K-4$ fat-tree DCN.

4.4.1 Oddlab within the SDN Paradigm

The proposed model was designed within the control plane and data plane of the SDN paradigm as shown in Figure 4.3, we performed Oddlab for flow scheduling and failed links detection into three phases. The first phase resides in the DCN data plane, where flows are sampled based on the (1:1) technique. Then, flow is either handled by the controller or forwarded based on the ECMP hashing technique. The second phase will be in the control plane and contains two main sub-models. The first is for ports stats polling, and the second is for odds calculation and labeling based on the edge switch consumption. In the third phase, the best path calculation and link health checking are estimated based on the network-directed graph's information to define the healthy paths and update the aggregate switch bucket weight in the data plane.

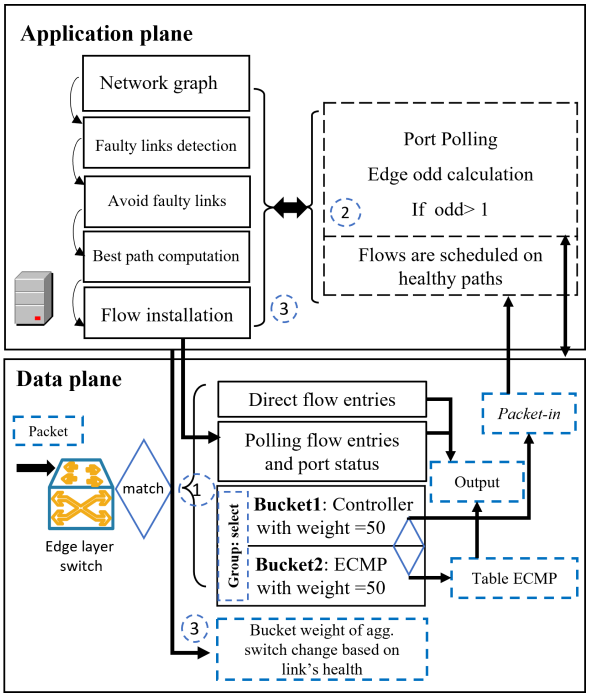


Figure 4.3: Oddlab architecture.

In Oddlab, we define the following problems to detect the faulty links and avoid them in the adaptive flow scheduling mechanisms.

Definition 18. *Problem* Oddlab Faulty Links Detection (OFLD): *Given a multi-commodity DCN flow* $G = (V, E, b, El\_e)$, *where each core switch has links* $(u, v) \in E$, *the goal is to detect the links that do not carry the same amount of traffic when those links are active for a specified duration of monitoring time.*

Definition 19. *Problem* Oddlab Elephant Flow Rescheduling (OEFR): *Given a multi-commodity DCN flow* $G = (V, E, El\_e)$, *where each core switch have links* $(u, v) \in E$, *the objective is to detect the elephant flows from the affected core links and reschedule them at the edge switches (with flow rate* $\geqslant$ *50 Kbps) to other available paths from* V_s *to* V_t.

4.4.2 Complexity Analysis

In this section, we prove the OEFR problem as an NP-complete problem. As we proved our problems (SAFS in Definition 6 and OAFS in Definition 15) by reduction and construction special case of know problems.

Theorem 20. *OEFR is NP-complete.*

Proof. Given $G(V, E, b, El\_e)$ and elephant flows on edge switches in the case of OEFR, the objective is to look for a different set of paths (SET) from V_s to V_t and assign the flows to the best available path based on Definition 15 within time slot $[\tau, 1 + \tau]$. In the OEFR problem, all elephant flows found on the affected links will be rescheduled to another set of healthy paths between the V_s and V_t each with residual bandwidth b and a number of active elephant flows El_e. This process can be verified by checking if the new reserved bandwidth and number of active elephant flows of the chosen path can handle the new rescheduled flow without affecting the current flows on the path within the time interval. Then, it will be easy to count and reschedule the elephant flows from the affected port to another healthy path. These operations can be solved in polynomial time. Thus, OEFR in the class of NP.

The decision of OEFR is similar to the OAFS problem mentioned in Definition 15) but with avoiding the affected core switch ports in the best path decision. Thus, OEFR is NP-hard by definition, and since OEFR are in the class of NP, it is NP-complete.

□

4.5 Spatial-Temporal Correlation to Detect Faulty Links in DCN

Inspired by the fact that network traffic inside DCN is known to have properties such as similarities, periodicity, and correlation [56]. Therefore, we leverage spatial-temporal correlation between two events in the input data space on the edges and cross the DCN over the aggregate and core switches at different time intervals to detect the potentially failed links. Spatial-temporal reasoning in detecting the faulty links shows how the two conditions of the DCN loading state and the underutilized links fit together in the DCN life span.

In this correlation, the spatial-temporal reasoning in detecting the faulty links shows how the two conditions of the DCN loading state and the underutilized links fit together in the DCN operation life span. As shown in Figure 4.4, the controller will monitor and estimate the edge load and links' health based on the predefined thresholds. Whenever both events are correlated in consecutive monitoring intervals (i.e., from $t3$ to $t5$ for short-lived failures), the controller will indicate the affected link as "*unhealthy link*" and redirect all elephant flows found on the upstream port of the link. As depicted at $t6$ interval in Figure 4.4, the correlation is broken, but the link failure threshold is still fulfilled. In this case, the path that contains the detected link will be avoided in the adaptive flow scheduling. Nevertheless, it will be excluded from the detected link list when the link is recovered (i.e., getting regular traffic) from the proactive paths using ECMP. However, we represent our spatial-temporal correlation model in the following assumptions.

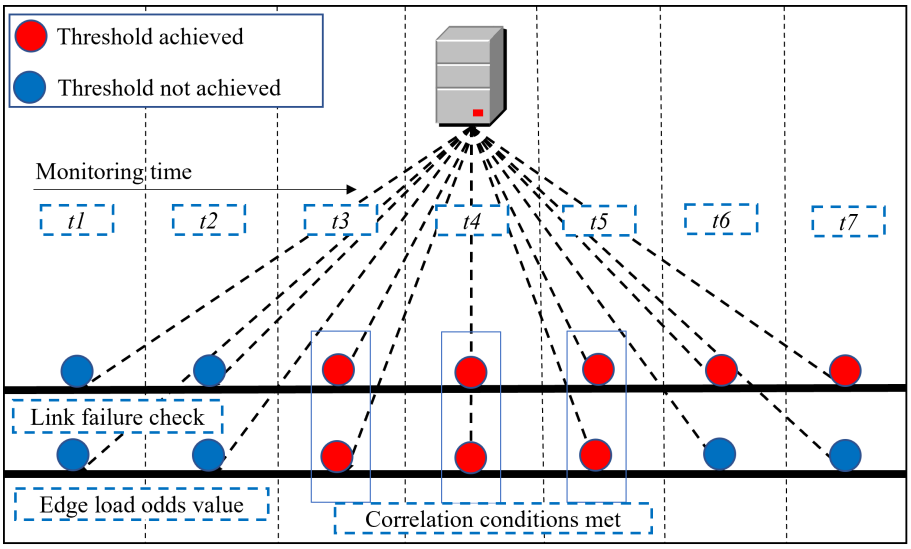


Figure 4.4: The process of spatial-temporal correlation to detect the potential failed links.

Assumption 1. *We observe and correlate two inconsistencies in network events monitored in the same time window between the edge (traffic source) and core switches (intermediate nodes).*

Assumption 2. *Since the DCN core links are very prone to failure [29], the process of checking the failed link is included in each period of DCN monitoring time.* [1].

We introduced the failure correlation with the following parameters:

- The bidirectional core links C_l, which indicates the link in the DCN core layer, where C_l=(e, $health\_thr$), $e \subseteq E$ of the DCN links with underutilized threshold $health\_thr$.
- The DCN utilization stat (*Odds* value) of the edge switches dge_odds.
- The counter of the sequence number S of the correlation parameters C_l and $edge\_odds$.

[1]The detailed information of the faulty links detection will be described in Subsection 4.5.2

Namely, the given parameters C_l and *edge_odds* are represented with uniform distribution [1, 0] based on the defined threshold conditions for each parameter. At the same time, the counter of the sequence number (CD) is defined as a "correlation determinant", where the link failure is assigned based on the counter of the correlation sequence number and the defined threshold (*health_thr*). The proposed determinate CD will hold for the different types of failures (soft and hard failures). In addition to determining the false positive link failure detections resulted from broken failure correlations. The possible detected failed links information is defined into a set R (C_l, *edge_odds*, CD) so that Oddlab scheduler reroutes the affected elephant flows and avoids these links into next flow scheduling.

4.5.1 DCN Utilization State Estimation

We consider the DCN utilization state estimation as confidence evidence that the detected congested links in the core layer are potential failed links with high probability $P(failed\_link|loaded\_DC)$. As depicted in Algorithm 6, Oddlab invokes all ports statistics from the switches periodically using `OFPPORTStats` messages based on the predefined polling rate, ($P_r = 2$ sec.). We leverage the collected ports information in several functions to estimate the DCN loading state, such as finding ports free bandwidth function (*save_free_bw*), finding path bottleneck (*min_bw_links*), and detecting the faulty links $(save\_health)$. Initially, *save_free_bw* function has been defined to store the throughput consumption information ($free\_bw$) of the DCN switch's ports $(dpid, prt\_no)$ in a directed graph $G(V, E)$. As a result, the primary flow scheduling procedure will rely on the reserved graph free bandwidth values (*min_bw_links*) when defining the path bottleneck. Reg-

ularly, this function will ensure that the selected path has the best available bandwidth out of the available paths between each end-hosts.

We utilize the obtained information of port $(dpid, prt\_no)$ consumption $(free\_bw)$ to determine the DCN utilization. For this sake, we only filter the edge switches ports' available bandwidth values on the upstream side (ports 1 and 2). In this step, edge switch would be labeled as loaded L_e if the residual bandwidth value of each port is at least 95% of the link capacity for both ports. Then, the *Odds* value will be calculated based on the number of labeled switches to the number of not labeled switches NL_e (Equation 4.1).

$$Odds = \frac{\sum L_e}{\sum NL_e} \tag{4.1}$$

We can summarize the reasons behind adopting such a threshold (95% of the link capacity) as follows:

(1) The fact is that the DCN traffic tends to be so bursty [12] [77]. Since we employed an equal flow sampling (i. e., 1:1) at the DCN edge switches, it is impossible to assume elephant flows are always transferred under controller monitoring.

(2) This threshold is applied only on the upstream side of the edge switches, and it will ensure early utilization estimation to look for the potential faulty links.

Hence, choosing above this threshold will significantly affect the detection time of the faulty links.

4.5.2 Faulty Links Detection Procedure

When the number of edge switches that fulfills the minimum loading threshold exceeds the number of not loaded switches (i.e., $Odds > 1$), the Oddlab controller will invoke the second initiated reaction for getting the best-path as depicted in (*path_health_check*) in Algorithm 7. The links' health is estimated periodically based on port statistics' information initiated in `OFPPortStatsRequest` together with the $save\_free\_bw$ function as shown in Algorithm 6. Initially, the core switches' links are considered healthy unless their throughput falls below the failure threshold (i.e., $health\_thr \leqslant 1\%$ of the link capacity). To maintain the correlation conditions ($health\_thr$ and $Odds$), the number of consecutive faulty link frequencies will be calculated over the network monitoring duration ($fault\_iter$) as described in Algorithm 6 (line 17). When the correlation is fulfilled, the faulty link information will be saved on the directed graph $G(V, E)$, including (src_prt, dst_prt, $health$, $fault\_iter$). In the other hand, the correlation parameter of the detected faulty links ($fault\_iter$) will be nulled wherever link's throughput grows above the threshold (i.e., $health\_thr > 1\%$ of the link capacity) as illustrated in Algorithm 6 line 25. The information of the truly faulty links, including $dpid$ and src_prt, will be stored into a list (af_links) to be applied as elephant flows rescheduling information (Algorithm 8 line 22).

4.5.3 Elephant Flows Rescheduling

The Oddlab controller will use the af_links list information to reschedule all the congested elephant flows on these links. As illustrated in Algorithm 9 (line 7), the flow information is monitored periodically for all ports to participate in the process of flow scheduling. Accordingly, when the value

of $Odds > 1$, all elephant flows mounted on the upstream side of the aggregate switch's ports ($dpid$ & prt_no) with a flow rate $\geqslant$ 50 Kbps [11] will be rescheduled to other healthy paths. As explained in Algorithm 7, these flows will be redirected from end to end-host based on the flow's source IP address (src_ip) to new healthy paths ($best\_path$). The reason behind choosing end-to-end redirection is to get the shortest healthy path instead of overwhelming more links if we decide to reroute from the aggregate switch. Note that the elephant flows scheduling decision was made to give mice flows more resources to pass through the affected links. Besides, there is no point in rescheduling mice flows since they are tiny and may significantly affect completion time. Finally, the rescheduled flows information is stored into the red_links list with the switch and port number information ($dpid$ & prt_no) so that the port will not be checked in the following flow monitoring round.

4.5.4 Adaptive Flow Scheduling After Detecting the Faulty Links

The identified faulty links in the (af_links) list were avoided by the Oddlab adaptive flow scheduling method. As shown in Algorithm 7 (line 1), the Oddlab second reaction relies on the estimated $Odds$ value ($Odds > 1$). The algorithm primarily guarantees that the determined paths are free of any faulty links in the core switches. Therefore, the health of the path is checked before computing the active flow numbers of the path. For simplicity, we define a high value for $max\_flownum$ whenever a faulty link is detected in any path to estimate the path flow number; thus, the adaptive flow will

not choose that path later in the adaptive flow scheduling, as described in Algorithm 5 (line 8).

As an outcome of the faulty link detection, the flow load-balancing equation is as follows (Equation 4.2):

$$U(u,v) = \frac{\sum_{i=1}^{kc} fe_i(u,v)}{hp(u,v).(mf(u,v).(mb(u,v)))} \tag{4.2}$$

Where hp represents the checked healthy paths between u and v nodes.

Note that even when the value of *Odds* depreciates below 1, the adaptive flow scheduling will avoid the defected paths based on Equation 4.2. In Algorithm 5 (line 7), we defined a statement to check the status of the detected links (af_list) and avoid them by assigning a very large number to the $max\_flownum$ parameter so that the path including that link will not be considered as a $best\_path$.

The following assumption is built upon the observation of Gill et al. [29] on the DCN link failures, where they noticed that the link failures would last about ten minutes for the kind of software failures. In contrast, hardware failure persists longer and often requires a direct intervention to solve the failure. We employed the correlation determinant in the correlation process over information polling intervals (P_r = 2 sec.) to identify the potential faulty links and remove the false positive results as follows:

Assumption 3. *The new state for every core links R (C_l′, edge_odds′, CD) compared with the previously detected set R (C_l, edge_odds, CD) to define the failed link.*

$\therefore R\ (C\_l', edge\_odds', CD) \subseteq R\ (C\_l, edge\_odds, CD)\ \forall\ Core, iff\ edge\_odds' > 1\ \&\&\ C\_l' \leqslant C\_l\ , CD = CD + 1$

Note that whenever the number of loaded edge switches is dropped (i.e., $edge\_odds' < 1$), the primary correlation will break, and the Core links (C_l') will not be checked for failure. The assumption helps identify the false positive detections by tracing the throughput changes on any Core link (C_l') in the case of $C\_l' \geq C\_l$. Therefore, the link (C_l') will be omitted from the potentially failed links set R (C_l, $edge\_odds$, CD), and the correlation determinant will be nulled (CD=0).

The probability of being any Core link e detected as faulty links depend on Assumption 4.

Assumption 4. *Any Core link e determined as a faulty link iff $e \subseteq R$(C_l, edge_odds, CD) and $CD > 1$ for Short-Lived (S_L) failures or $CD \geqslant 5$ for Long-Lived failures (L_L).*

Thus, Assumption 4 can be put into a failure indicator (I_R) to determine the faulty link and what type it belongs to, based on the CD value.

$$I\_R(C\_l, edge\_odds, CD)\,(e)= \begin{cases} \text{S\_L} & \text{iff e}\subseteq R(C\_l, edge\_odds, CD)\ \&\ CD > 1 \\ \text{L\_L} & \text{iff e}\subseteq R(C\_l, edge\_odds, CD)\ \&\ CD \geqslant 5 \\ 0 & Otherwise \end{cases}$$

Then, $dpid$ and src_prt of the detected link are saved into the faulty links set FL($dpid$, src_prt, CD) af_links list to be used as elephant flow redirection information and to be avoided for the upcoming flow scheduling. For long-lived failures, we defined a higher value for $fault\_iter \geqslant 5$ (i.e., the monitoring time is equivalent to $5 \times P_r$). Accordingly, the proactive bucket weight of links that achieves this threshold will be modified; hence, the affected port will not obtain the last equal number of flows. Therefore, the weight value of the affected upstream port is omitted using the OpenFlow `ovs-ofctl mod-group` message ($dpid$, src_prt), as described in Algorithm 8

(line 23). Eventually, an alert is triggered for the detected faulty link, with the link information and whether the link has been excluded from the service (i.e., hardware failure), $failure\_alert$ = (af_links & $proac\_ports$)).

Algorithm 5: Path flow number estimation.

```
   Data: G=(V,E) , path, max_fnum, global(af_links)
   Result: max_flownum
1  for i in len(path)-1 do
2      src_prt = G[j][i][j][i+1](src_prt)
3      dst_prt = G[j][i][j][i+1](dst_prt)
4      fault_src_itr = G[j][i][j][i+1](fault_src)
5      fault_dst_itr = G[i][j][i][j+1](fault_dst)
6      fnum = G[j][i][j][i+1](fnum)
       // This condition will avoid detected links when fault
          iteration reaches the fault_iter threshold.
7      if (i in af_links and src_prt in af_links[j] and dst_prt in af_links[i] and fault_src
        ⩾ fault_iter and    fault_dst ⩾ fault_iter) then
8          max_flownum = ∞
9      else
           max_flownum = max(fnum, max_fnum)
10 return max_flownum
```

Algorithm 6: Handling of port information for links failure detection.

Data: $G=(V,E)$, $path$, min_bw, P_r, $dpid\_list$, $aff\_edge\_list[dpid][prt\_no]$, $speed$, global $(af\_links[dpid][prt\_no])$, global $proac\_ports$
$all\_edges = 8$ edge switches in $k-4$ fat-tree

Result: *Odds*, *af_links[dpid][prt_no]*

healthy_links[dpid][prt_no] = [ ]
foreach P_r **do**
  Function `OFPPortStatsRequest(`$dpid\_list$`):`
    Function `save_free_bw(`$dpid, prt\_no, speed$`):`
      $free\_bw$ =$capacity$-$speed$
      $G[j][i][j][i+1](bw) \leftarrow free\_bw$
      if *dpid in edge_list and prt_no in [1,2] and free_bw* $\leq$ *edge_thr* **then**
        $aff\_edge\_list[dpid][prt\_no] \leftarrow loaded$
      for *d in* $aff\_edge\_list[dpid][prt\_no]$ **do**
        if *prt_no [1,2] is loaded* **then**
          `// Number of affected edge switches.`
          aff+=1
          not_aff=all_edges - aff
        Odds= aff / not_aff
      return *G[j][i][j][i+1](bw), Odds*
      if *dpid or dpid in edge_list and port_no in [1,2]* **then**
        call save_health $(dpid, prt\_no, speed)$
      Function `save_health(`$dpid, prt\_no, speed$`):`
        $dpid\_prt\_health$ = $speed$ / $capacity$
        $fault\_iter$ =0
        if $dpid\_prt\_health \leq health\_thr$ *and* $Odds > 1$ **then**
          if prt_no *in* $healthy\_links[dpid]$ **then**
            $fault\_iter = fault\_iter+1$
            $healthy\_links[dpid][prt\_no] \leftarrow health, fault\_iter, Odds$
            $G[j][i][j][i+1](src\_prt) \leftarrow src\_prt$
            $G[i][j][i][j+1](dst\_prt) \leftarrow dst\_prt$
            $G[j][i][j][i+1](health) \leftarrow health$
            $G[j][i][j][i+1](fault\_iter) \leftarrow fault\_iter$
        if *prt_no in healthy_links[dpid] and dpid_prt_health* $>$ *health_thr and prt_no in af_links[dpid] and prt_no in proac_ports[dpid]* **then**
          `// To delete false-positive faulty links.`
          $fault\_iter$ =0
          $af\_links[dpid].remove(prt\_no)$
    return *af_links[dpid][prt_no], Odds, G[j][i][j][i+1](bw)(health)(fault_iter)(src_prt) (dst_prt)*

Algorithm 7: Oddlab adaptive flows scheduling when DCN is in an overutilization state.

Data: *G=(V,E)* , *src_IP*, *dst_IP*, *src_prt*, *dst_prt*, *min_bw = capacity*, *max_bw = 0*, *max_fnum* = 0, *path_health* = 0, *k*, *shortest_p*= { }, global(*Odds*, af_links, *proac_ports*)

Result: *best_path*= [ ], $failure\_alert$

```
1  while Odds > 1 do
       // In utilized DC, the best path will be determined based
          on the shortest and healthy path with the least elephant
          active flows and high available bandwidth.
2      for i in (0,k) do
3          if shortest_p [i]= shortest_paths (G, src_IP, dst_IP) then
4              for h in shortest_paths do
5                  path_health = path_health_check (G, j, link_health)
6                  if path_health ∀ e ∈ path ==0 then
                       max_fnum =0
7                      path_health (src_IP, dst_IP) ← max_fnum
8                  else
                       max_fnum = ∞
9                      path_health (src_IP, dst_IP) ← max_fnum
10         path_health (src_IP, dst_IP) ← max_fnum
11         for j in path_health do
12             max_fnum = maxfnum_of_path (G, j, max_fnum)
13             fnum_of_paths (src_IP, dst_IP) ← max_fnum
14             min_fnum_path = min(fnum_of_paths (src_IP, dst_IP,max_fnum))
15         for fnum in fnum_of_paths do
16             min_bw = bottleneck_of_path (G, j, min_bw)
17             if min_bw > max_bw then
18                 max_bw = min_bw
19             best_path =fnum
20 return best_path, failure_alert(af_links & proac_ports)
```

Algorithm 8: Path health estimation.

Data: *G=(V,E)* , *path*, *link_health*, P_r, *health_thr*, global*(af_links)*, *fault_iter*
Result: *path_health*

foreach P_r **do**
  health_list = [ ]
  non_health_list = [ ]
  proac_ports= [ ]
  for *i in len(path)-1* **do**
    src_prt = G[j][i][j][i+1](src_prt)
    dst_prt = G[i][j][i][j+1](dst_prt)
    fault_src_itr = G[j][i][j][i+1](fault_iter)
    fault_dst_itr = G[i][j][i][j+1](fault_iter)
    _health_src_prt = G[j][i][j][i+1](health)
    _health_dst_prt = G[i][j][i][j+1](health)
    if *(_health_src_prt >health_thr and fault_src_itr ⩽ fault_iter and _health_dst_prt > health_thr and fault_dst_itr ⩽ fault_iter)* **then**
      health_list ← _health_src_prt)
    else
      if *(_health_src_prt < health_thr and fault_src_itr > fault_iter and _health_dst_prt < health_thr and fault_dst_itr > fault_iter)* **then**
        non_health_list ← (_health_src_prt)
        if *path[i] not in af_links* **then**
          af_links[path[i]] ← [src_prt]
    if *path[i+1] not in af_links* **then**
      af_links[path[i+1]] ← [dst_prt]
    else
      if *dst_prt not in af_links[path[i+1]]* **then**
        af_links[path[i+1]] ← [dst_prt]
    `// The following statement for proactive links changing on the aggr-core switches in case of long-lived failures.`
    if *path[i] in agg_swt and path[i] not proac_ports and src_prt in [1,2] and fault_src_iter ⩾ 5 and fault_dt_iter ⩾ 5* **then**
      ovs-ofctl mod-group (*path[i], src_prt*)
      proac_ports ← (path[i],src_prt)
  if *len(non_health_list)>0* **then**
    path_health= ∞
    else
      path_health= 0
return *path_health*

Algorithm 9: Handling of flow information for DCN switches.

Data: *G=(V,E)*, $dpid\_list$, *path*, *flow_list*, *link_health*, P_r, *health_thr*, global*(af_links)*
Result: *red_links[dpid][prt_no]*

```
foreach P_r do
    Function EventOFPFlowStatsReply(dpid_list):
        for f in in flow_list do
            flow_net = flow_byte / flow_duration
            if flow_net ≥ 50 Kbps then
            fnum[dpid][prt_no] ←fnum[dpid][prt_no]+1
            G[j][i][j][i+1](fnum) ← fnum
            if len (af_links) is-not null and Odds > 1 then
                // Rerouting the congested elephant flows to
                   alternative paths.
                for d in af_links do
                    for p in af_links[d] do
                        if af_links[d][p] ∩ red_links[d][p] is true then
                            continue
                        else
                            dpid, prt_no = d, p
                    for fd in flows installed on (dpid, prt_no) do
                        if fd_load ⩾ 50 Kbps then
                            E_count++
                    for e in E_cont do
                        if e[dpid][prt_no] not in red_links then
                            red_links ← e[dpid][prt_no]
                            Based on src_ip of each flow e, get e_flow_info
                            get_best_path (G, e, e_flow_info) // The
                               congested flows rerouted based on the
                               execution of algorithm 7.
return red_links[dpid][prt_no], G[j][i][j][i+1](fnum)
```

4.6 Complexity Evaluation

In this section, we estimate the time and space complexity of Oddlab by considering the worst-case scenarios when rescheduling the affected elephant flows after detecting the faulty links. In this case, the time complexity will include the information of ports consumptions and the number of active elephant flows, besides the number of redirected elephant flows. Hence, the complexity will be $(O(|El\_e| + k^2))$. As for the space complexity, the controller memory should maintain the following variables, including the number of active elephant flows and the residual bandwidth of each port beside the number of redirected elephant flows of the affected links on the upstream ports of the aggregate switches. For instance, as depicted in Figure 1.1, out of 80 ports in $K-4$ fat-tree DCN switches, only 16 upstream ports connect aggregate and core switches. Hence, the space complexity is $(O(|El\_e|(\frac{k}{5}) + k^3))$.

As explained in Subsection 3.6.4, the best paths are periodically determined based on the DCN edge loading status (*Odds* value). The best paths are frequently optimized based on the acquired network parameters (i.e., residual bandwidth and the number of active elephant flows) from the ports of the DCN switches. The scheduling algorithm does not become complicated even after identifying the utilized switches and *Odds* value; additional iterations will only be appended to the previously gathered parameters to learn the best paths. The added complexity is in the number of redirected elephant flows $|El\_e|$ from the affected paths only when identified faulty links.

4.7 Experimental Results

This section discusses and analyzes the Oddlab flow scheduling results obtained under asymmetric DCN topology, including faulty links detection. Note that we made the data results available publicly on the Oddlab GitHub repository[2].

4.7.1 Performance Under Asymmetric DCN Topology

Asymmetricity in DCN topologies is imperative due to multiple factors such as partial failures. In this experiment, we modify some links bandwidth to achieve asymmetric fat-tree topology. For this sake, we set the capacity of some links between aggregate and core switches to 5 Mbps, while other links remain 10 Mbps [55]. This experiment uses the same workloads based on web search and cache jobs. We noted that the FCT values for the flows will be affected by bandwidth reduction remarkably. Therefore, we assess the performance based on the achieved FCT of the comparative TE methods. This experiment applies in the random traffic pattern to allow enough flows to pass over the core switches. Accordingly, the asymmetric topology effect will be recognized at the end-hosts application layer.

Figure 4.5 shows the achieved reduction of the overall average FCT values. The results show that the Oddlab scheduler can finish the overall flows within 18 sec. as an average FCT, which reduces up to 12%, 8%, and 10% compared with ECMP, Hedera, and PureSDN, respectively. This experiment proves that despite the decrease in the link bandwidth is not notable, the adaptive scheduling of Oddlab achieved a better FCT reduction without complicated flow rerouting.

[2]https://github.com/aymeniq/Oddlab

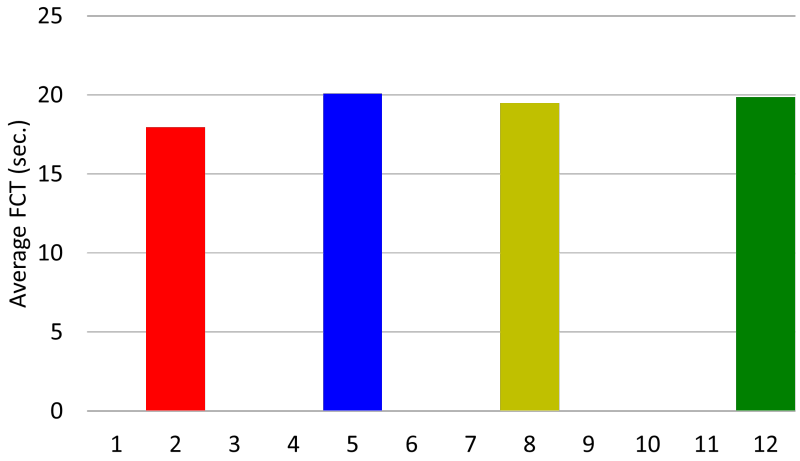


Figure 4.5: Average overall FCT for Oddlab compared with ECMP, Hedera, and PureSDN under asymmetric DCN topology.

4.7.2 Failure Detection

This experiment investigates multiple failures between aggregate and core switch links with severe bandwidth reduction. Accordingly, we set the capacity of specific core links to 0.01 Mbps between switch 1 and pod 1 and between switch 4 and pod 4, as shown in Figure 4.1. Note that the bandwidth threshold was assumed to simplify the faulty link simulation, and the algorithm can be adjusted manually to any required bandwidth severity. Nevertheless, the same experiment of real workloads in asymmetric DCN topology was repeated with random traffic patterns. We set the link fault threshold iteration ($fault\_iter$) to be larger than 1.

Consequently, the faulty links should maintain the same link faulty threshold $health\_thr = 0.01$ Mbps at least two monitoring periods in a row when $Odds > 1$. After detecting the faulty links, all the active elephant flows on the aggregate switch's upstream side will be rescheduled to other healthy paths. Thus, all the incoming flows will still get symmetric paths of the

DCN topology by avoiding the faulty links. The remaining flows at the faulty link are mice flows (i.e., flow rate $<$ 50 Kbps). If the link's productivity increases throughout the slow start of a TCP flow, it will be excluded from the list of affected links (i.e., false-positive case). Nevertheless, if the link remains at the same productivity rate (long-lived failure) with $Odds > 1$ for further monitoring periods, then upstream hashing weight will be altered accordingly. The evaluation of this experiment will be based on the obtained average FCT values for redirected elephant flows after redirection. This experiment conducted five independent simulations since different numbers and sizes of elephant flows are randomly scheduled on the faulty links based on the edge switches flow sampling and the adaptive flow scheduling decisions. We found that only 97 elephant flows were found on the faulty links and recovered based on adaptive flow scheduling during the experiment.

Fig. 4.6 shows the average FCT for the rescheduled elephant flows found on the detected faulty links. The initial FCT refers to the primary determinate FCT value on the end-host application layer for each flow over the faulty links. We found unusual initial outlier FCT values (i.e., more significant than 1000 sec.) for three elephant flows larger than 6 Mbyte in size. So, we omitted them to obtain precise average calculations. As depicted in Figure 4.6, we observe that Oddlab considerably reduces the average FCT for the redirected elephant flows over the initial determined FCT, thanks to detecting the faulty links and the adaptive flow scheduling. After detecting the faulty links, the Oddlab adaptive scheduling model will spread the flows across the available paths without congesting the paths significantly, depending on the proposed adaptive scheduling. Hence, we remark that

the new average FCT for redirected elephant flows is acceptable compared with their sizes.

4.7.3 Discussion and Comparison

In specific TE methods such as Hedera [27], Mahout [19], and Sieve, the elephant flows only rescheduled when reaching a certain threshold or based on the availability of enough bandwidth on other paths. To handle the faulty links issue, Hedera, for instance, adopted PortLand specialized mechanisms [68] for link failure detection and flow rerouting.

DCN end-hosts can indicate early the elephant flows via the initial determined FCT. Still, with the cost of altering all end-hosts to monitor each flow and participating in the routing decisions such as SAPS [44], Hermes [108], FlowFurl [82]. Furthermore, end-host sensing does not provide sufficient information about the affected DCN links, so network administrators cannot efficiently discover the problem. Therefore, we can conclude that the experiment results prove Oddlab feasibility and profits based on SDN controller and OpenFlow protocol.

Regarding detecting faulty links inside DCN topology, Gill et al. [29] correlated links failure logs with the link's earlier observed traffic in a five-minute time window. Although this type of correlation required a link-state memory for each DCN link defined in each monitoring time, the link should not fail because of a preceding traffic decline. For instance, the current state of the link could be in a typical traffic situation due to a routine service drop from the end-hosts. Therefore, multiple false positive detections would occur. On the other hand, Oddlab's correlation process occurred with current and upcoming events in a shorter time (2 sec. of the information polling

rate). Accordingly, when the correlation breaks in any upcoming event, the link will be identified as a false positive and return to normal.

We got some false positive tests in our tests, especially when the simulation tends to the end and there is no more traffic to send so that Oddlab cannot delete the false positive detections. The faulty links detection sensitivity (the true positive rate) was 100% accurate to detect the faulty links. As for the specificity test (the true negative rate), Oddlab was 92.85% specific to identify the correct faulty links (i. e., one false positive link out of 14 regular links).

Although we adopted ECMP paths to overcome the controller overhead, we encountered the dilemma of not identifying the precise number of flows that the edge switch handles. Therefore, we counted the port consumption in edge labeling. Faulty link detection by Oddlab is based on the following observation: "*As far as most edge switches are labeled based on the odds concept, the faulty paths can be detected quickly*". Consequently, the expected time for identifying faulty links relies on the network traffic demands and monitoring time to achieve a high detection precision.

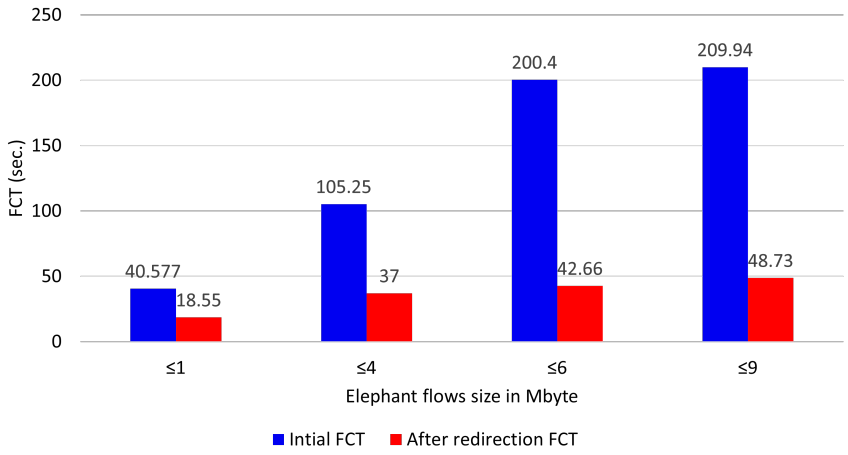


Figure 4.6: Average FCT for the redirected elephant flows after failure detection.

4.8 Conclusions

This thesis introduces the faulty link detection procedure adopted in our proposed TE method (Oddlab). The procedure of identifying faulty links relies on correlating two events within the DCN: loaded edge switches and underutilized core links. Therefore, Oddlab employs the statistics of the DCN switches on a single and central SDN controller to detect faulty links and achieves promising results in both the symmetric and asymmetric topologies. However, extensive experiments were conducted on a wide range of traffic patterns with synthetic and realistic workloads to prove the feasibility of Oddlab without altering any network component, including hosts or switches. The results indicated that Oddlab reduced the average overall FCT up to 12%, 8%, and 10% compared with ECMP, Hedera, and PureSDN, respectively. Oddlab can detect, reschedule the affected elephant flows with severe bandwidth degradation in the core links and avoid the defective paths in the adaptive flow scheduling model. We demonstrated that Odd-

lab functions with low complexity and low computational overhead on the SDN controller. Therefore, Oddlab has the potential to be applied to commercial DCNs at a low cost.

Chapter 5

Conclusions and Future Directions

"Each problem that I solved became a rule, which served afterwards to solve other problems."

René Descartes (1596-1650)

5.1 Thesis Summary

This thesis introduces our novel framework solution for evaluating and solving TE issues utilizing the SDN concept. Mainly, the thesis includes three related theses arranged as follows:

5.1.1 DCN Risk Analysis

The first part of the thesis deals with a risk analysis of existing Traffic Engineering (TE) methods based on the SDN concept. We can summarize the achieved results of this thesis as follows:

- We introduce the concept of risk analysis of deploying the SDN TE methods inside the DCN topology (i. e., fat-tree).
- The evaluation included estimating the loss uncertainty of elephant flows losses based on Monte Carlo simulation.
- We empirically investigated the risk factors of elephant flows based on realistic workloads with traffic patterns to predict the value at risk of the elephant flow loss rate.
- Our evaluation included estimating the probability distribution for risk factors based on the Kolmogorov Smirnov and Anderson-Darling statistical testing methods.
- We studied different existing TE methods such as the hashing-based method (ECMP [36]), adaptive SDN method for elephant flow scheduling (Hedera [27]), beside the fully SDN-based method (PureSDN [59]).
- We showed that the probability of elephant flow losses in Hedera achieved 56.76% with 97 MBps maximum expected loss, ECMP is 62.83% with 102 MBps, and PureSDN is 44.43% with 87 MBps, respectively.
- We proved in the Monte Carlo simulation that the elephant flow loss event is highly expected DCN. As a result, adopting any TE method would directly influence the status of DCN applications in terms of flow completion time and throughput.

5.1.2 Adaptive SDN Load Balancing Framework

In the second part, we propose Sieve and Oddlab based on the SDN concept. The achieved results of these methods can be summarized in the following:

- We investigated the flow distribution scheme in DCN through predefined flow paths at the DCN edge switches employing ECMP hashing besides the SDN controller.

- We employed the adaptive scheduling for flows scheduling that considers available bandwidth in Sieve and the active elephant flows in Oddlab to find the best paths and avoid significant flow rescheduling.

- We formed the adaptive flow scheduling and elephant rescheduling problems for Sieve and Oddlab and proved them as NP-complete problems.

- We proved theoretically and empirically that leveraging the global knowledge of the DCN switches statistics on a single and central SDN controller to deliver promising results in symmetric DCN topologies without altering any network components or TCP protocol.

- The performance analysis of Oddlab based on the law of large numbers revealed that the adopted flow sampling mechanism at the edge switches diminishes the SDN controller burden to half besides reducing the waiting time for flow handling.

- We conducted extensive experiments on a wide range of traffic patterns with synthetic and realistic workloads to prove Oddlab and Sieve feasibility. The results confirmed that Sieve and Oddlab achieved promising improvements in the average overall Flow completion time, mice flow FCT, bisection bandwidth, link utilization, packets loss, round trip delay compared with ECMP, Hedera, PureSDN, and Sieve, respectively.

- We applied our proposed risk analysis framework to predict the elephant flows loss rate. The results showed that Sieve decreased the elephant flow losses probability by 45.74%, almost similar to PureSDN by 44.43% compared to 52.8% achieved by the Oddlab scheduling procedure.

5.1.3 DCN Faulty Links Detection Model

In the third part of the thesis, we present the Oddlab faulty links detection model. The model achieved results can be summarized in the following:

- We used the same global network information for flow adaptive scheduling to identify the faulty links with acceptable computational complexity without modifying other network components such as switches or end-hosts.
- The procedure of identifying faulty links relies on correlating two events within the DCN: loaded edge switches and underutilized core links.
- We use the information of the affected paths to update the adaptive flow scheduling model of Oddlab and modify the proactive paths if required based on the type of the link failure (i. e., long or short failure).
- We proved that Oddlab employs the statistics of the DCN switches on a single and central SDN controller to detect faulty links and achieves promising results in both the symmetric and asymmetric topologies.

5.2 Thesis Applications

The achieved results from the proposed performance evaluation model and flow scheduling solutions contribute directly to the traffic engineering aspect using SDN controller and OpenFlow in DCNs concerning Flow Management, Fault-tolerant, Topology update, and Traffic monitoring.

The outcomes of Thesis 1 (Chapter 2) determined the deployment risk factors of different SDN TE methods regarding the expected loss rate of elephant flows and demonstrated their impact on data center applications productivity. Therefore, the outcomes might apply to all DCN applications, such as Hadoop and MapReduce, that demand high stability and productivity. This evaluation was applied as a baseline to develop new SDN TE methods to solve various issues in DCN TE aspects, as presented in Thesis 2 (Chapter 3) and Thesis 3 (Chapter 4).

The results of Thesis 2 are also essential to discover the flow sampling and frequent elephant flow rerouting at the DCN edge and to what extent the method was effective toward improving DCN Flow Management, Topology update, and Traffic monitoring. Fault-tolerant ability and fewer Topology updates presented in Thesis 3 showed that Oddlab was achieved with significantly low computational power overhead in terms of space and time complexity. Therefore, Oddlab implementation is highly applicable in industrial DCNs due to uncomplicated arithmetic operations and less overhead. Besides, there is no need for any modification to the DNC hosts.

5.3 Future Research Directions

The DCN fabric must be maintained and monitored to guarantee high availability and better QoS by employing an effective traffic engineering (TE) method. Our proposed solutions present a step forward in the direction of producing and evaluating an effectual SDN TE. Still, many challenges are planned to be addressed in the future, including:

- In terms of risk analysis, we plan to simulate the link failures effect of the elephant flow losses rate and DCN productivity.

- With the rapid evolution of cloud services and the Internet of Things (IoT) technologies, the problem of DCN power consumption is getting more severe [35]. For instance, Google consumes 260 million watts to maintain the normal DCN operations [8] [30]. Hence, we intend to propose a framework that estimates the anticipated power consumption of different DCN TE methods.

- The SDN controller's central position is considered the single point of failure in DCN management [66]. An attack such as Distributed Denial-of-Service (DDoS) is widely used to breach the SDN controller [43] and put the whole DCN down. However, several security solutions tend to use distributed SDN controllers [71] [73] [4]. We already proposed an SDN forensic prototype based on distributed SDN controllers as appeared in [7]. Nevertheless, we plan to measure how the proposed measure of the proposed flow sampling can be employed to defend the controller against DDoS attacks.

- We plan to leverage Oddlab in DCN power reduction by switching off specific paths based on the edge switches' traffic demands and odds ratio.

- Further investigation is needed to examine the effectiveness of the proposed method on different DCN topologies, including multi-stage topologies such as the Clos network.

- Oddlab faulty link detection strategy functions only on a three-stage topology and beyond, including the $k-4$ fat-tree topology. Hence, a possible approach to overcome this issue is to correlate other events, such as queue lengths of the paths.

However, more innovative research is needed to study, analyze, and produce new solutions for emerging services such as the IoT, cloud services, fifth-generation (5G) networks, and beyond. For instance, with the fast 5G deployment, IoT applications are expected to generate 163 zettabytes (ZB) of data by 2025 [21]. Although Machine learning strategies (i. e., reinforcement learning) provide promising results, more investigation is needed to produce faster and less complex TE models.

Bibliography

List of Publications

International Journals

(J_1) **Alawadi, A. H.** and Zaher, Maiass and Molnár, Sándor, "Methods for Predicting Behavior of Elephant Flows in Data Center Networks", Infocommunications Journal, Vol. XI, No 3, September 2019, pp. 34-41. DOI: 10.36244/ICJ.2019.3.6. (WOS, Scopus, CS=0.8). (*2.67 point*)

(J_2) Zaher, Maiass and **Alawadi, A. H.** and Molnár, Sándor, "Sieve: A flow scheduling framework in SDN based data center networks", Computer Communications, 171, Elsevier. 2021, pp. 99-111. DOI: 10.1016/j.comcom.2021.02.013. (WOS, IF=3.16, Scopus, CS=5.8). (*2 points*)

(J_3) **Alawadi, A. H.** and Molnár, Sándor, "Oddlab: Fault-Tolerant Aware Load Balancing Technique for Data Center Networks", Annals of Telecommunications journal. Springer Nature. 2021. https://doi.org/10.1007/s12243-021-00898-0. (WOS, IF=1.563, Scopus, CS=3.9). (*6 points*)

International Conferences

(C_1) **Alawadi, A. H.**, and Molnár, Sándor, Risk Analysis of Blocked Rate Predictions for SDN Load Balancing Using Monte Carlo Simulation. In 2019 IEEE Symposium on Computers and Communications (ISCC), pp. 1028-1033. IEEE, 2019. (*3 points*)

(C_2) Zaher, Maiass and **Alawadi, A. H.** and Molnár, Sándor, "Class-based Flow Scheduling Framework in SDN-based Data Center Networks",

In 2020 IEEE International Conference on Computing, Electronics Communications Engineering (iCCECE), pp. 51-56. IEEE, 2020. (*0.99 point*)

Other Journal Publications

(O_1) **Al Awadi, A. H.**, and Belaton, B., "Multi-phase IRC botnet and botnet behavior detection model". International Journal of Computer Applications, 66(15), pp. 41-51, 2013. (*3 points*)

(O_2) Kadhim, H. A., and **AlAwadi, A. H.**, and Sarvghadi, M. A., "Experimental Study of Parallelizing Breadth First Search (BFS) Algorithm". Research Journal of Applied Sciences, Engineering and Technology, 12(4), pp. 465-472, 2016. (*2 points*)

(O_3) **Al Awadi, A. H.**, "Dual-layer SDN model for deploying and securing network forensic in distributed data center". Current Journal of Applied Science and Technology, pp. 1-11, 2017. (*3 points*)

Publication overall score = 22.66 points

References

[1] Hadi Mohammadzadeh Abachi et al. "Statistical discretization of continuous attributes using kolmogorov-smirnov test". In: *Australasian Database Conference*. Springer. 2018, pp. 309–315.

[2] Chintan Advani et al. "A Wi-Fi Sensor-Based Approach for Examining Travel Time Reliability Parameters Under Mixed Traffic Conditions". In: *Transportation in Developing Economies* 6.1 (2020), pp. 1–9.

[3] Yehuda Afek et al. "Detecting heavy flows in the SDN match and action model". In: *Computer Networks* 136 (2018), pp. 1–12.

[4] Suhail Ahmad and Ajaz Hussain Mir. "Scalability, consistency, reliability and security in sdn controllers: A survey of diverse sdn controllers". In: *Journal of Network and Systems Management* 29.1 (2021), pp. 1–59.

[5] Jung Ho Ahn et al. "HyperX: topology, routing, and packaging of efficient large-scale networks". In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. 2009, pp. 1–11.

[6] Ian F Akyildiz et al. "A roadmap for traffic engineering in SDN-OpenFlow networks". In: *Computer Networks* 71 (2014), pp. 1–30.

[7] Aymen Hasan Rashid Al Awadi. "Dual-layer sdn model for deploying and securing network forensic in distributed data center". In: *Current Journal of Applied Science and Technology* (2017), pp. 1–11.

[8] Sultan Alanazi et al. "Reducing data center energy consumption through peak shaving and locked-in energy avoidance". In: *IEEE Transactions on Green Communications and Networking* 1.4 (2017), pp. 551–562.

[9] Mohammad Alizadeh et al. "CONGA: Distributed congestion-aware load balancing for datacenters". In: *Proceedings of the 2014 ACM conference on SIGCOMM*. 2014, pp. 503–514.

[10] Mohammad Alizadeh et al. “pfabric: Minimal near-optimal datacenter transport”. In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 435–446.

[11] Theophilus Benson, Aditya Akella, and David A Maltz. “Network traffic characteristics of data centers in the wild”. In: *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 2010, pp. 267–280.

[12] Theophilus Benson et al. “Understanding data center traffic characteristics”. In: *ACM SIGCOMM Computer Communication Review* 40.1 (2010), pp. 92–99.

[13] Jeandro de M Bezerra et al. “Performance evaluation of elephant flow predictors in data center networking”. In: *Future Generation Computer Systems* 102 (2020), pp. 952–964.

[14] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[15] Jake Brutlag. *Speed matters for Google web search*. 2009.

[16] *BWM-ng - Bandwidth Monitor NG (Next Generation)*. https://www.gnutoolbox.com/bwmng. [Online; accessed 18-October-2021].

[17] Li Chen et al. “Scheduling mix-flows in commodity datacenters with karuna”. In: *Proceedings of the 2016 ACM SIGCOMM Conference*. 2016, pp. 174–187.

[18] Leonardo C Costa et al. “OpenFlow data planes performance evaluation”. In: *Performance Evaluation* 147 (2021), p. 102194.

[19] Andrew R Curtis, Wonho Kim, and Praveen Yalagandula. “Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection”. In: *2011 Proceedings IEEE INFOCOM*. IEEE. 2011, pp. 1629–1637.

[20] Andrew R Curtis et al. “DevoFlow: Scaling flow management for high-performance networks”. In: *Proceedings of the ACM SIGCOMM 2011 Conference*. 2011, pp. 254–265.

[21] Delali Kwasi Dake et al. “Traffic Engineering in Software-defined Networks using Reinforcement Learning: A Review”. In: *International Journal of Advanced Computer Science and Applications* 12.5 (2021).

[22] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[23] Antony Drokin. *Easyfit - Distribution fitting software*. ://easyfit.informer .com/5.6/. [Online; accessed 10-October-2021]. 2010.

[24] Stefan Ekman and Måns Svensson. "Brianaria (Psoraceae), a new genus to accommodate the Micarea sylvicola group". In: *The Lichenologist* 46.3 (2014), pp. 285–294.

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

[26] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. "A scalable, commodity data center network architecture". In: *ACM SIGCOMM computer communication review* 38.4 (2008), pp. 63–74.

[27] Mohammad Al-Fares et al. "Hedera: dynamic flow scheduling for data center networks." In: *Nsdi*. Vol. 10. 8. San Jose, USA. 2010, pp. 89–92.

[28] Soudeh Ghorbani et al. "Drill: Micro load balancing for low-latency data center networks". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017, pp. 225–238.

[29] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. "Understanding network failures in data centers: measurement, analysis, and implications". In: *Proceedings of the ACM SIGCOMM 2011 Conference*. 2011, pp. 350–361.

[30] James Glanz. *Google Details, and Defends, Its Use of Electricity*. The New York Times, 2011.

[31] Albert Greenberg et al. "VL2: A scalable and flexible data center network". In: *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. 2009, pp. 51–62.

[32] Binjie He, Dong Zhang, and Chang Zhao. "Hidden Markov Model-based Load Balancing in Data Center Networks". In: *The Computer Journal* 63.10 (2020), pp. 1449–1462.

[33] Kouji Hirata and Miki Yamamoto. "Data center traffic engineering using Markov approximation". In: *2017 International Conference on Information Networking (ICOIN)*. IEEE. 2017, pp. 173–178.

[34] Sungmin Hong et al. "Poisoning network visibility in software-defined networks: New attacks and countermeasures." In: *Ndss*. Vol. 15. 2015, pp. 8–11.

[35] Peng HongYu et al. "An Improved Energy Saving Strategy for SDN-based Data Center". In: *2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE. 2020, pp. 1371–1376.

[36] Christian Hopps et al. *Analysis of an equal-cost multi-path algorithm*. Tech. rep. RFC 2992, November, 2000.

[37] Jinbin Hu et al. "CAPS: Coding-based adaptive packet spraying to reduce flow completion time in data center". In: *IEEE/ACM Transactions on Networking* 27.6 (2019), pp. 2338–2353.

[38] Jin Huang et al. "Climatology of rainfall erosivity during 1961–2012 in Jiangsu Province, southeast China". In: *Natural Hazards* 98.3 (2019), pp. 1155–1168.

[39] Jin Huang et al. "Regional changes of climate extremes and its effect on rice yield in Jiangsu province, southeast China". In: *Environmental earth sciences* 77.3 (2018), pp. 1–11.

[40] Jin Huang et al. "Spatiotemporal analysis the precipitation extremes affecting rice yield in Jiangsu province, southeast China". In: *International journal of biometeorology* 61.10 (2017), pp. 1863–1872.

[41] Douglas W Hubbard. *The failure of risk management: Why it's broken and how to fix it*. John Wiley & Sons, 2009.

[42] *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*. https://iperf.fr. [Online; accessed 18-October-2021].

[43] Amir Iranmanesh and Hamid Reza Naji. "A protocol for cluster confirmations of SDN controllers against DDoS attacks". In: *Computers & Electrical Engineering* 93 (2021), p. 107265.

[44] Syed Mohammad Irteza et al. "Efficient load balancing over asymmetric datacenter topologies". In: *Computer Communications* 127 (2018), pp. 1–12.

[45] Philippe Jorion. *Value at risk*. McGraw-Hill Professional Publishing, 2000.

[46] Glenn Judd. "Attaining the Promise and Avoiding the Pitfalls of {TCP} in the Datacenter". In: *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*. 2015, pp. 145–157.

[47] Abdul Kabbani et al. "Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks". In: *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 2014, pp. 149–160.

[48] Srikanth Kandula et al. "Dynamic load balancing without packet reordering". In: *ACM SIGCOMM Computer Communication Review* 37.2 (2007), pp. 51–62.

[49] John Kim, William J Dally, and Dennis Abts. "Flattened butterfly: a cost-efficient topology for high-radix networks". In: *Proceedings of the 34th annual international symposium on Computer architecture*. 2007, pp. 126–137.

[50] Diego Kreutz, Fernando MV Ramos, and Paulo Verissimo. "Towards secure and dependable software-defined networks". In: *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. 2013, pp. 55–60.

[51] Yuan-Liang Lan, Kuochen Wang, and Yi-Huai Hsu. "Dynamic load-balanced path optimization in SDN-based data center networks". In: *2016 10th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*. IEEE. 2016, pp. 1–6.

[52] Andy Lawrence. *Annual outage analysis 2021*. Uptime Institute, 2021.

[53] Chen Levi and Michael Segal. "Avoiding bottlenecks in networks by short paths". In: *Telecommunication Systems* 76.4 (2021), pp. 491–503.

[54] Kaiyang Liu, Aqun Zhao, and Jianping Pan. "Data Center Architecture, Operation, and Optimization". In: *Future Networks, Services and Management*. Springer, 2021, pp. 185–212.

[55] Lu Liu et al. "An SDN-based hybrid strategy for load balancing in data center networks". In: *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2019, pp. 1–6.

[56] Yazhi Liu et al. "Load balancing oriented predictive routing algorithm for data center networks". In: *Future Internet* 13.2 (2021), p. 54.

[57] Hui Long et al. "LABERIO: Dynamic load-balanced routing in OpenFlow-enabled networks". In: *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE. 2013, pp. 290–297.

[58] Shibo Luo et al. "Context-aware traffic forwarding service for applications in SDN". In: *2015 IEEE International Conference on Smart City/ SocialCom/ SustainCom (SmartCity)*. IEEE. 2015, pp. 557–561.

[59] Huang Machi. "Research on Data Center Network Traffic Scheduling Strategy Based on SDN". In: *University of Posts and Telecommunications, Chongqing, China*. 2017.

[60] Roger McHaney et al. "Iterative conceptual modeling: A case study in cardiac patient survival simulation". In: *Operations research for health care* 19 (2018), pp. 57–65.

[61] Nick McKeown et al. "OpenFlow: enabling innovation in campus networks". In: *ACM SIGCOMM computer communication review* 38.2 (2008), pp. 69–74.

[62] *Mininet: A realistic virtual network*. http://mininet.org. [Online; accessed 17-October-2021].

[63] Michael Mitzenmacher. "The power of two choices in randomized load balancing". In: *IEEE Transactions on Parallel and Distributed Systems* 12.10 (2001), pp. 1094–1104.

[64] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[65] Tatsuya Mori et al. "On the characteristics of Internet traffic variability: spikes and elephants". In: *IEICE TRANSACTIONS on Information and Systems* 87.12 (2004), pp. 2644–2653.

[66] Seyed Mohammad Mousavi and Marc St-Hilaire. "Early detection of DDoS attacks against SDN controllers". In: *2015 International Conference on Computing, Networking and Communications (ICNC)*. IEEE. 2015, pp. 77–81.

[67] Tuan Anh Nguyen, Dong Seong Kim, and Jong Sou Park. "Availability modeling and analysis of a data center for disaster tolerance". In: *Future Generation Computer Systems* 56 (2016), pp. 27–50.

[68] Radhika Niranjan Mysore et al. "Portland: a scalable fault-tolerant layer 2 data center network fabric". In: *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. 2009, pp. 39–50.

[69] Mohammad Noormohammadpour and Cauligi S Raghavendra. "Datacenter traffic control: Understanding techniques and tradeoffs". In: *IEEE Communications Surveys & Tutorials* 20.2 (2017), pp. 1492–1525.

[70] Badana Ntanganedzeni and Joel Nobert. "Flood risk assessment in Luvuvhu river, Limpopo province, South Africa". In: *Physics and Chemistry of the Earth, Parts A/B/C* (2020), p. 102959.

[71] Mathis Obadia et al. "Failover mechanisms for distributed SDN controllers". In: *2014 International Conference and Workshop on the Network of the Future (NOF)*. IEEE. 2014, pp. 1–6.

[72] Panos M Pardalos and Stephen A Vavasis. "Quadratic programming with one negative eigenvalue is NP-hard". In: *Journal of Global optimization* 1.1 (1991), pp. 15–22.

[73] Kévin Phemius, Mathieu Bouet, and Jérémie Leguay. "DISCO: Distributed SDN controllers in a multi-domain environment". In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE. 2014, pp. 1–2.

[74] Andreas N Philippou, Costas Georghiou, and George N Philippou. "A generalized geometric distribution and some of its properties". In: *Statistics & Probability Letters* 1.4 (1983), pp. 171–175.

[75] Jaafar A Rashid. "Sorted-GFF: An efficient large flows placing mechanism in software defined network datacenter". In: *Karbala International Journal of Modern Science* 4.3 (2018), pp. 313–331.

[76] David Reinsel, John Gantz, and John Rydning. *The Digitization of the World From Edge to Core*. IDC white paper by Seagate, 2018.

[77] Arjun Roy et al. "Inside the social network's (datacenter) network". In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 2015, pp. 123–137.

[78] Walter Rudin et al. *Principles of mathematical analysis*. Vol. 3. McGraw-hill New York, 1964.

[79] *Ryu: Ryu SDN Framework*. https://ryu-sdn.org. [Online; accessed 17-October-2021].

[80] Saim Salman et al. "DeepConf: Automating data center network topologies management with machine learning". In: *Proceedings of the 2018 Workshop on Network Meets AI & ML*. 2018, pp. 8–14.

[81] Klaus Schittkowski. "EASY-FIT: a software system for data fitting in dynamical systems". In: *Structural and Multidisciplinary Optimization* 23.2 (2002), pp. 153–169.

[82] Kapil Sharma and Ram Narayan Yadav. "An adaptive, fault tolerant, flow-level routing scheme for data center networks". In: *Computer Networks* 175 (2020), p. 107235.

[83] Nick Shelly et al. "Destroying Networks for Fun (and Profit)". In: HotNets-XIV. Philadelphia, PA, USA: Association for Computing Machinery, 2015.

[84] F Bruce Shepherd and Adrian R Vetta. "The Inapproximability of Maximum Single-Sink Unsplittable, Priority and Confluent Flow Problems". In: *Theory of Computing* 13.1 (2017), pp. 1–25.

[85] Eric K Simpeh et al. "A rework probability model: a quantitative assessment of rework occurrence in construction projects". In: *International Journal of Construction Management* 15.2 (2015), pp. 109–116.

[86] Dag IK Sjoberg, Tore Dyba, and Magne Jorgensen. "The future of empirical methods in software engineering research". In: *Future of Software Engineering (FOSE'07)*. IEEE. 2007, pp. 358–378.

[87] V Sokolov et al. "A network analytics system in the SDN". In: *2014 International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC)*. IEEE. 2014, pp. 1–3.

[88] Remy Spliet, Adriana F Gabor, and Rommert Dekker. "The vehicle rescheduling problem". In: *Computers & Operations Research* 43 (2014), pp. 129–136.

[89] Ali M Subyani and Nassir S Al-Amri. "IDF curves and daily rainfall generation for Al-Madinah city, western Saudi Arabia". In: *Arabian Journal of Geosciences* 8.12 (2015), pp. 11107–11119.

[90] Feilong Tang et al. "Elephant flow detection and differentiated scheduling with efficient sampling and classification". In: *IEEE Transactions on Cloud Computing* (2019).

[91] Chan Kok Thim, Mohammad Nourani, and Yap Voon Choong. "Value-at-risk and conditional value-at-risk estimation: A comparative study of risk performance for selected malaysian sectoral indices". In: *2012 International Conference on Statistics in Science, Business and Engineering (ICSSBE)*. IEEE. 2012, pp. 1–5.

[92] Ramona Trestian, Gabriel-Miro Muntean, and Kostas Katrinis. "MiceTrap: Scalable traffic engineering of datacenter mice flows using OpenFlow". In: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE. 2013, pp. 904–907.

[93] Jacek Tylkowski and Marcin Hojan. "Threshold values of extreme hydrometeorological events on the Polish Baltic coast". In: *Water* 10.10 (2018), p. 1337.

[94] Bhanu Chandra Vattikonda et al. "Practical tdma for datacenter ethernet". In: *Proceedings of the 7th ACM european conference on Computer Systems*. 2012, pp. 225–238.

[95] J-C Walter and GT Barkema. "An introduction to Monte Carlo methods". In: *Physica A: Statistical Mechanics and its Applications* 418 (2015), pp. 78–87.

[96] Chuan'an Wang et al. "A switch migration-based decision-making scheme for balancing load in SDN". In: *IEEE Access* 5 (2017), pp. 4537–4544.

[97] Chunzhi Wang et al. "An ACO-based elephant and mice flow scheduling system in SDN". In: *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*. IEEE. 2017, pp. 859–863.

[98] Haibo Wang et al. "PrePass: Load balancing with data plane resource constraints using commodity SDN switches". In: *Computer Networks* 178 (2020), p. 107339.

[99] Peng Wang et al. "Luopan: Sampling-based load balancing in data center networks". In: *IEEE Transactions on Parallel and Distributed Systems* 30.1 (2018), pp. 133–145.

[100] Shuo Wang et al. "Flow distribution-aware load balancing for the datacenter". In: *Computer Communications* 106 (2017), pp. 136–146.

[101] You-Chiun Wang and Siang-Yu You. "An efficient route management framework for load balance and overhead reduction in SDN-based data center networks". In: *IEEE Transactions on Network and Service Management* 15.4 (2018), pp. 1422–1434.

[102] Anis Yazidi, Hussein Abdi, and Boning Feng. "Data center traffic scheduling with hot-cold link detection capabilities". In: *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*. 2018, pp. 268–275.

[103] Shui Yu, Xiaodong Lin, and Jelena Misic. "Networking for big data: part 2 [Guest Editorial]". In: *IEEE network* 29.5 (2015), pp. 4–5.

[104] Maiass Zaher, Aymen Hasan Alawadi, and Sándor Molnár. "Sieve: A flow scheduling framework in SDN based data center networks". In: *Computer Communications* 171 (2021), pp. 99–111.

[105] Umme Zakia and Hanene Ben Yedder. "Dynamic load balancing in SDN-based data center networks". In: *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE. 2017, pp. 242–247.

[106] Hong Zhang et al. "Resilient datacenter load balancing in the wild". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017, pp. 253–266.

[107] Junjie Zhang et al. "CFR-RL: Traffic engineering with reinforcement learning in SDN". In: *IEEE Journal on Selected Areas in Communications* 38.10 (2020), pp. 2249–2259.

[108] Jiaqi Zheng et al. "Hermes: Utility-aware network update in software-defined wans". In: *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE. 2018, pp. 231–240.