# Mice Flow Routing in Data Centers based on Software-defined Networking


Universidad
del Cauca

Trabajo de Grado

**Carlos Felipe Amézquita S.**

Advisor: PhD. Oscar Mauricio Caicedo Rendón
Co-Advisor: Msc. Carlos Felipe Estrada Solano

*Departamento de Telemática*
*Facultad de Ingeniería Electrónica y Telecomunicaciones*
*Universidad del Cauca*
*Popayán, Cauca, 2018*

# Mice Flow Routing in Data Centers based on Software-defined Networking

Carlos Felipe Amézquita S.

Trabajo de grado presentado a la Facultad de Ingeniería
Electrónica y Telecomunicaciones de la
Universidad del Cauca para obtener el título de:
Ingeniero en Electrónica y Telecomunicaciones

Advisor: PhD. Oscar Mauricio Caicedo Rendón
Co-Advisor: Msc. Carlos Felipe Estrada Solano

*Departamento de Telemática*
*Facultad de Ingeniería Electrónica y Telecomunicaciones*
*Universidad del Cauca*
*Popayán, Cauca, 2018*

# Acknowledgement

First and foremost, I would like to thank God for my life and my opportunity to continually grow up as a person and as a professional. I want to express my sincere gratitude to my family, especially my parents, for their constant support over the years. They have dedicated their lives to my educational training, and I want to dedicate this work to them. I am also grateful to all my friends and colleagues for being present in each step of my formation process as a person and as a professional. Furthermore, I would like to thank my academic advisor, PhD. Oscar Mauricio Caicedo Rendón and my co-advisor, Msc. Carlos Felipe Estrada Solano for their support in directing and guiding our research work, helping to solve some problems and giving me innovative ideas.

# Abstract

Network operators have used SDN for routing flows in DCNs. However, a significant problem affecting the overall performance of DCNs based on SDN is the delay introduced to latency-sensitive small flows (*i.e.*, mice) by the SDN controllers. Current approaches tackle this problem by compiling and installing paths for mice and elephants dynamically, but this has shortcomings related to the large number of routing rules that the switches must handle, leading to significant delays to mice flows. In this monograph, we introduce MiceDCER, an algorithm that allows the efficient routing, regarding the delay, of mice flows in SDN-based DCNs by assigning internal Pseudo-MAC (PMAC) addresses to the edge switches and hosts. It also installs wildcard rules based on the information carried out by the ARP packets, to indicate the controller the rules it should install on the switches. To test our algorithm, we developed a prototype and conducted the experiments in an emulated topology to compare the results with other routing protocols based on the number of rules. This comparison reveals that MiceDCER significantly reduces the number of rules installed in switches on the topology and, therefore, contributes to reducing the delay in SDN-based DCNs.

# Resumen

Los operadores de red han usado SDN para enrutar flujos en DCNs. Sin embargo, un problema significante que afecta el desempeño general de los DCNs basados en SDN es el retardo introducido en los pequeños flujos sensibles a la latencia (es decir, ratones) por los controladores SDN. Los enfoques actuales abordan este problema compilando e instalando rutas dinámicamente para los elefantes y ratones, pero esto tiene deficiencias relacionadas con el gran número de rutas de enrutamiento que los conmutadores tienen que manejar, llevando a retardos significantes en los flujos ratones. En esta monografía, introducimos MiceDCER, un algoritmo que permite el enrutamiento eficiente en cuanto al retraso de los flujos ratones en DCNs basados en SDN, asignando direcciones Pseudo-MAC (PMAC) internas a los conmutadores de borde y a los hosts. También instala reglas de comodín (*wildcard*) basadas en la información transportada por los paquetes ARP, para indicarle al controlador las reglas que debe instalar en los conmutadores. Para probar nuestro algoritmo, desarrollamos un prototipo y conducimos los experimentos en una topología emulada para comparar los resultados con otros protocolos de enrutamiento basándonos en el número de reglas. Esta comparación revela que MiceDCER reduce significativamente el número de reglas instaladas en los conmutadores en la topología y, por lo tanto, contribuye a reducir el retardo en los DCNs basados en SDN.

# Content

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **AA** | Application-specific Address |
| **AMAC** | Actual MAC |
| **API** | Application Programming Interface |
| **ARP** | Address Resolution Protocol |
| **DCN** | Data Center Networks |
| **DCTCP** | Data Center TCP |
| **ECMP** | Equal-Cost Multi-Path |
| **IoT** | Internet-of-Things |
| **IP** | Internet Protocol |
| **IT** | Information Technology |
| **LA** | Location-specific Address |
| **LAN** | Local Area Network |
| **LLDP** | Link Layer Discovery Protocol |
| **LSB** | Least Significant Bits |
| **MAC** | Media Access Control |
| **MiceDCER** | Mice Data Center Efficient Routing |

| | |
|---|---|
| **MPTCP** | Multi-Path TCP |
| **MSB** | Most Significant Bits |
| **NIC** | Network Interface Controller |
| **NPU** | Network Processor |
| **PMAC** | Pseudo-MAC |
| **QoS** | Quality of Service |
| **RAM** | Random Access Memory |
| **RCP** | Routing Control Platform |
| **RTT** | Round Trip Time |
| **SDN** | Software-Defined Networking |
| **SRAM** | Static Random Address Memory |
| **TCAM** | Ternary Content-Addressable Memory |
| **TCP** | Transmission Control Protocol |
| **TE** | Traffic Engineering |
| **VLAN** | Virtual LAN |
| **VM** | Virtual Machine |
| **VMM** | VM Migration |
| **VoIP** | Voice over IP |
| **WAN** | Wide Area Network |
| **WBS** | Work Breakdown Structure |
| **WLAN** | Wireless Local Area Network |
| **WP** | Work Package |

# Chapter 1

# Introduction

In this chapter, we outline our approach. First, we present the problem statement and describe the objectives followed for the development of this undergraduate work. Second, we present the work packages and activities carried out as well as the schedule followed. Third, we present the resources and budget allocated to the development of the project, as well as the contributions achieved.

## 1.1. Problem Statement

The *Software-Defined Networking* (SDN) is an emerging architecture that separates the control plane from the data forwarding plane (*e.g.*, switches and routers) for enabling a more straightforward network operation from a logically centralized software program, usually known as the controller [1]. This network architecture emerged to tackle the limitations (*e.g.*, scalability and static policies) of traditional networking architectures in satisfying the complex networking needs of today applications. SDN promises to improve the network resource utilization, simplify network management, reduce operating cost, and promote innovation and evolution [2].

Typically, in SDN, when the first packet of a flow arrives at a switch, it is redirected to the controller if the routing rule for handling that flow does not exist in

the network switching table. The controller then compiles the routing logic and dynamically installs the path for the flow by adding routing rules on each switch over the route from source to destination. Subsequent packets from the same flow are not redirected to the controller and follow the installed path. Also, the controller can be programmed for establishing static routing rules on switches for specific routes that flows must follow [1].

SDN has been used for routing different flows in *Data Center Networks* (DCN), mainly focused on the fact that most flows tend to be short-lived and small (*i.e.*, mice), while only very few flows are long-lived and large (*i.e.*, elephants). The mice flows are usually referred to latency-sensitive and bursty applications, such as *Voice Over IP* (VoIP) and search results [3]. These flows are numerous and, thus, they generate many events to be handled by switching devices. The elephant flows are often large transfers like back-end operations and backups. The phenomenon of mice and elephants influences the overall DCN performance [4]: large flows tend to fill network buffers end-to-end, introducing delays to the latency-sensitive small flows that share the same buffers [3].

A lot of research works has addressed the routing of mice and elephant flows by compiling and installing dynamically the path for both mice and elephants from the controller [5]. Here, the problem arises when the first packet from each flow is sent to the controller, introducing the delay of sending the packet. This delay affects negatively the latency-sensitive mice flows [6]. The controller would install routing rules for each received flow, and a large number of flows causes an overload in the controller, increasing the processing time and the delay of flows [7, 8]. The installation of many routing rules in a switch is a problem in DCNs since it increments the time that the switch takes to find the routing rule for a specific flow, generating an scalability issue [5]. This increase in time also increases the delay for flows and may exceed the limited memory of the flow table in switches, which may cause that some flows be lost (dropped) [2].

To tackle the problems mentioned above, other works have proposed to compile and install the path dynamically only for elephant flows [9, 10]. Therefore, these works handle the routing of mice flows by using static rules as in *Equal-Cost Multi-Path* (ECMP). ECMP is an approach that allows including multipath

routing without the need for additional protocols or special configurations, using simple static flow-to-link assignment methods [11]. The implementations considering static rules has some drawbacks such as the high number of routing rules installed on the switches if the rules are per source-destination pairs, the complex task of updating the rules in case of changes in the network, and the inability to set redundant paths for mice flows. Some works reduce the number of static rules by setting ordered IPs in the DCN using IP prefixes, but this is not suitable to provide *Virtual Machine Migration* (VMM). Other works also use scalable [10, 12], distributed [13] or centralized [9] *Traffic Engineering* (TE) techniques (*e.g.,* algorithms, architectures, and protocols) to address the problems related to the routing of mice flows. In general, the related work describes mostly methods to re-route mice flows according to links conditions, but without considering delays caused by an overload in the controller or when sending the packet to the flow table in the controller. It is essential to allow the switch to handle the installation of routing rules in the controller to reduce the delay in the latency-sensitive mice flows [5].

To sum up, it is necessary to find a solution that focuses on coping with the delay on mice flows in DCNs, defining an efficient way on how packets are sent to the SDN controller when the flow arrives, and how routing rules are installed and managed in the network switches. As a result of the above, the following research question is raised:

**How to route efficiently, in terms of delay, mice flows in Data Centers based on Software-Defined Networking?**

## 1.2. Objectives

### 1.2.1. General Objective

Propose a technique for mice flows routing in Data Center Networks based on Software-Defined Networking.

### 1.2.2. Specific Objectives

- Design a technique for routing mice flows in Data Center Networks based on Software-Defined Networking.
- Implement a prototype of the designed technique.
- Evaluate the efficiency of the prototype regarding the delay of mice flows in an emulated environment.

## 1.3. Activities & Schedule

### 1.3.1. Work Structure

To structure and organize the processes during the undergraduate work, we adopted as reference the *Work Breakdown Structure* (WBS). WBS operates by dividing the overall work into more manageable hierarchy structured tasks. Furthermore, WBS is designed by a top-down procedure, decomposing each level from top to bottom into logical groupings of work, allowing lowest-level components to be scheduled, and its costs can be estimated, monitored, and controlled. It is a valuable planning tool that links the goals with the resources and activities in a logical frame [14].

#### Work Package 1. Generate the Initial Knowledge Base

- Review the state of art of the proposal. We carried out a review to check for acquired knowledge, and we compared them with aspects described in the undergraduate proposal.
- Synthesize and expose a summary. We interpreted and summarized a set of central ideas related to the concept of the proposed work.

- Build up the theoretical base. We collected a set of concepts and prepositions to form a perspective which allowed explaining what was going to be performed in the undergraduate work.

**Work Package 2. Design of the Technique**

- Analyze the existing relevant techniques. We have done analyzes to leverage the earlier knowledge and design and build the solution technique according to the obtained in the previous package.
- Define the efficient SDN mice (small) flow routing technique. We have done the definition to determine its following specification and design.
- Specify and design the routing technique. The technique was specified and designed to determine its later implementation.

**Work Package 3. Implementation of the Technique & Evaluation**

- Implement a prototype of the technique proposed. We conducted a proof-of-concept to evaluate the performance of our work regarding the number of rules.
- Evaluate the functionality and efficiency of the prototype regarding the delay. We designed an emulated SDN-based network to prove the efficiency of the prototype.
- Collect data and analyze them. The test results were analyzed and interpreted to check the performance of the prototype.
- Check the results obtained and compare them with techniques such as MiceTrap or PortLand. The test results obtained were compared to conclude that there was an improvement in respect to other routing protocols.

**Work Package 4. Disclosure**

- Elaborate an article. We carried out and submitted a conference paper named “An Efficient Mice Flow Routing Algorithm for Data Centers based on Software-Defined Networking” to present the results of the undergraduate work.
- Elaborate the monograph. We wrote the monograph as a highly detailed and documented study of the work done.

### 1.3.2. Schedule

Table 1.1 presents the schedule followed for developing this undergraduate work. The schedule includes the packets earlier defined.

| ACTIVITIES | MONTHS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **WP1. Generate the Initial Knowledge Base** | | | | | | | | | |
| Revision of the state of art of the proposal | | | | | | | | | |
| Synthesize and expose a summary of a set of central ideas | | | | | | | | | |
| Build up the theorical base by collecting a set of concepts and prepositions | | | | | | | | | |
| **WP2. Design of the Technique** | | | | | | | | | |
| Analyze the existing relevant techniques to leverage the earlier knowledge | | | | | | | | | |
| Define the efficient SDN mice (small) flow routing technique | | | | | | | | | |
| Specify and design the routing technique | | | | | | | | | |
| **WP3. Implementation of the Technique & Evaluation** | | | | | | | | | |
| Implement a prototype of the technique proposed | | | | | | | | | |
| Evaluate the functionality and efficiency of the prototype in an emulated SDN environment | | | | | | | | | |
| Collect and analyze data | | | | | | | | | |
| Check the results obtained and compare them with techniques such as MiceTrap or PortLand | | | | | | | | | |
| **WP4. Disclosure** | | | | | | | | | |
| Elaborate an article | | | | | | | | | |
| Elaborating the monograph | | | | | | | | | |

Table 1.1: Activity Schedule.

## 1.4. Resources & Budget

Table 1.2 and Table 1.3 present the aspects related to the budget spent during the development of this undergraduate work, considering a proposed value for the point in 2018 being 13,598 COP. This value is defined by the National Government of Colombia.

| Researcher | Dedication (h/week) | Weeks | Points | Point Value 2018 |
|---|---|---|---|---|
| Advisor | 1 | 36 | 2.5 | $13,598 |
| Co-advisor | 1 | 36 | 2.2 | $13,598 |
| Student | 40 | 36 | 1.5 | $13,598 |

Table 1.2: Reference Values for Project Budget.

| ITEMS | Sources | | TOTALS |
|---|---|---|---|
| | Student | Department | |
| **Human Resources** | | | |
| Advisor | – | $ 1´223,820 | $ 1´223,820 |
| Co-advisor | – | $ 1´076,962 | $ 1´076,962 |
| Student | $ 29´371,680 | – | $ 29´371,680 |
| **Technical Resources** | | | |
| Personal Computer with Internet Access | $ 1´450,000 | – | $ 1´450,000 |
| Printer | $ 850,000 | – | $ 850,000 |
| Servers | – | $ 1´600,000 | $ 1´600,000 |
| **Various Resources** | | | |
| Paper, Ink, Disks | $ 90,000 | – | $ 90,000 |
| | $ 31´761,680 | $ 3´900,782 | $ 35´662,462 |
| **AUI (20%)** | $ 6´352,336 | – | $ 6´016,240 |
| **TOTAL** | $ 38´114,016 | $ 3´900,782 | $ 42´014,798 |
| **PERCENTAGE** | 90.72% | 9.28% | 100% |

Table 1.3: Resources and Budget.

## 1.5. Contributions

The following work is framed in the research line of Telecommunication Advanced Services of the Telematics Department. Contributions are listed below:

- We designed and built a technique to determine its implementation for efficient routing of mice flows on SDN-based DCNs.
- We implemented a prototype of the routing technique of mice flows for evaluating its performance regarding the number of rules.
- We evaluated the functionality and efficiency of the technique for mice flow routing, to collect and analyze data and check the results obtained.

## 1.6. Organization

The rest of this document has been divided into chapters described below:

- Chapter 2 presents the **Data Centers based on Software-Defined Networking**, describing the relevant topics concerning our research. These topics include SDN, DCN, and TE.
- Chapter 3 presents the **Related Work** that describes the research works closer to our proposal.
- Chapter 4 introduces **MiceDCER** by presenting motivation, exposing an overview, and describing our algorithm and prototype in detail.
- Chapter 5 presents the **Evaluation** of our work. We define the testbed to evaluate the proposed algorithm. Subsequently, we evaluate and analyze MiceDCER and compare the results with other routing protocols regarding the number of rules.
- Chapter 6 presents **Conclusions and Future Work**. We provide the conclusions of our approach and expose ideas for future work.

# Chapter 2

# Data Centers based on Software-Defined Networking

In this chapter, we present a background related to our approach. First, we describe the architectures of SDN and DCN. Second, we present a classification of TE techniques. Third, we describe how network operators use TE methods for improving the performance of SDN-based DCNs.

## 2.1. Software-Defined Networking

The *Software-Defined Networking* (SDN) is an emerging paradigm that innovates in the design and management of computer networks. Although this technology may seem recent, this paradigm is part of a long history of efforts to make these networks more programmable, gaining significant attraction in the networking industry over the past few years [15]. The main feature of SDN is that it aims to separate the network control plane from the data forwarding plane, providing user applications a centralized view of the distributed network states [2]. It also facilitates programmability in changing the overall characteristics of network elements and allowing quicker management, configuration, and resource optimization with dynamic, proprietary-free programs. [16].

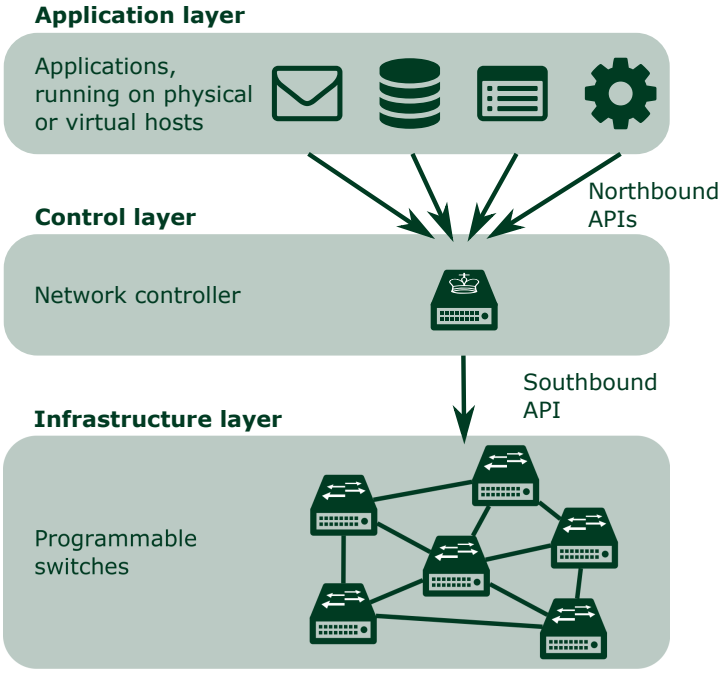



Figure 2.1: SDN Architecture.

Figure 2.1 presents an overview of the SDN architecture, which consists of three layers that interact and communicate between them [17]. SDN architectures usually use a single logically-centralized controller, but may make use of multiple controllers in large-scale or wide-area region networks [2]. The layers interact between them through the use of *Application Programming Interfaces* (API) [17] These interfaces provide access to network services (e.g., routing, security, and access control) and facilitate the achievement of several commercial objectives related to network management. Several SDN deployments oriented to flow management (e.g., Onix [18], BalanceFlow [19], and Beacon [20]) come with their own set of APIs [2].

On the bottom of the SDN architecture is the data forwarding layer (i.e., infrastructure layer), which consists of switching devices (e.g., switches, routers, and load-balancers) that make up the network topology [17]. The switching devices are responsible for collecting and storing temporarily network status and sending them to the controllers. They also process flow packets based on routing rules provided by a controller [17]. The topology in SDN can employ programmable

OpenFlow switches, providing access to the forwarding plane of a switch over the network and enables software applications to look up and forward packets among the switches, routers or other devices that make up the network [2].

In the middle of the architecture is the control layer, which the SDN controller operates. It communicates with the data forwarding layer via South-bound Open APIs, such as the OpenFlow protocol [2]. The control layer uses network policies to regulate the network states in either a centralized or distributed manner. These policies can be updated from time to time to react to the current flow activities [2]. The separation of the control plane from the data forwarding plane allows to control the entire network, and routing control and topology control are the primary functions of the control layer in SDN architecture [21].

The application layer, on top of the control layer, hosts and manages the SDN applications [2]. This layer communicates with the control layer through North-bound Open APIs, which give the applications access to the data collected from the network [22]. These applications can make the network system choose decisions based on the information of network status, and carry out these decisions by configuring switching and routing devices through the use of these APIs [17]. SDN commonly uses the OpenFlow protocol in its architecture, to enable network operators to treat flows better than traditional networks [16]. This protocol provides access to the forwarding plane of a switch over the network and enables switch programs to perform packet lookups and forward them among the network [2].

Figure 2.2 shows the architectural design of an SDN switching device, which consists of two logical components for the data plane and the control plane. In the data plane, the switch processor (e.g., XLP, XScale and NPUs) performs packet forwarding according to the rules imposed by the control layer. In the control plane, the switching device communicates with the controllers at the control layer to receive the rules for packet forwarding and link tuning, and store them in its local memory (e.g., TCAM and SRAM). Unlike conventional switching devices, SDN switches do not run routing protocols, which is a task given to the SDN controller, and they gather and report network status and forward the packets according to the imposed rules [17]. SDN enables network switches for intelligent and dynamic

traffic control [23], allowing network operators and administrators an efficient use and provisioning of network resources [24].

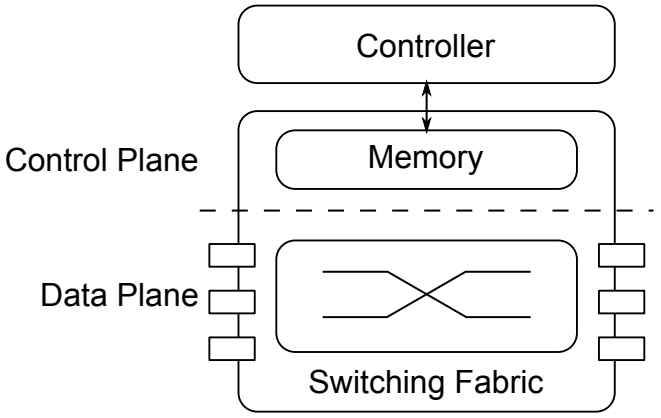



Figure 2.2: Switching Device Model in SDN: a two-layer logical model consisting of a processor for data forwarding and onboard memory for control information.

In SDN, network operators can easily and quickly manage, configure, and optimize network resources with dynamic, automated and proprietary-free programs written in the architecture [16]. It also provides a faster application deployment and delivery, improves the value of data center virtualization, increases resource flexibility, and provides greater cloud integration, solving the challenges associated with networks poorly-suited to the needs of dynamic technology (*e.g.*, security, and interoperability) [25]. SDN enhances the communication of network applications and allows them to share information. An example is the *Routing Control Platform* (RCP), which improves and optimizes the functionality of Kandoo, an SDN architecture with two levels of controllers, and has the ability to reduce the configuration status on the routers, minimizing errors and management complexity, allowing to deploy application services faster than in legacy network architectures [21].

The separation of planes allows SDN to provide efficient use of resources and a more flexible resource provisioning, and network operator employ it for many network applications, including Quality of Service (QoS) management, resource utilization, anomaly detection, traffic engineering, and so on [16]. SDN also im-

proves the performance and scalability of networks, and also allows it to centralize its functions in the control plane [21]. With its extensive capabilities of network programming, SDN can reduce or even remove the limitations of the network infrastructure and architecture to improve network efficiency.

## 2.2. Data Center Networks

The *Data Center Networks* (DCN) are the primary infrastructures for the delivery of cloud services, playing an important role to meet the needs demanded by the *Internet-of-Things* (IoT) applications [26]. Data centers, which are a set of servers, network devices, and other support elements, were recently promoted by network administrators as a cost-effective infrastructure for storing large volumes of data and hosting large-scale service applications [4]. Being a critical factor in the improvement of the network performance, DCNs are the interconnection fabric of the servers and switches within data centers [27]. The network topology, the routing or switching equipment, and the protocols used by the network can define the connections in a DCN. There are several types of DCN architectures, which are divergent in cloud computing support, topology and networking techniques. DCN architectures are also different in dealing with scalability issues [28].

For a better study on the behavior of these networks, DCN architectures split into two main groups: switch-centric and server-centric [26]. Server-centric DCN architectures consist primarily of commodity switches and servers with multiple *Network Interface Controller* (NIC) ports. This group can split into two subgroups: Dual-port and multi-port. Dual-port server-centric DCNs (e.g., FiConn and GQ), require only two NIC ports per server and can be constructed using commodity servers with a small number of NIC ports. Multi-port server-centric DCNs (e.g., Dcell, BCube, and MDCube), require a more significant number of NIC ports when increasing the number of servers, making necessary the use of specialized servers for deploying these architectures [29].

Switch-centric DCNs (e.g., FatTree [30], ElasticTree [31], and VL2 [32]) focus primarily on the network topology, which is often built in tree-based architectures, as

shown in Figure 2.3. These architectures distribute the switches in three operation layers: the core layer, the aggregate layer, and the edge layer, each of which features switches with interconnection intelligence [29]. Thus, the routing intelligence in these DCNs resides amongst the switches, with the servers behaving only as computational nodes. The connection in switch-centric DCNs is made using only server-to-switch and switch-to-switch links, rather than server-to-server links as in server-centric DCNs [27]. The switches have dense interconnections, which provide a more significant number of redundant paths between any given source and a destination edge switch, meaning that the excessive signature of the higher layer links can be significantly mitigated [33].

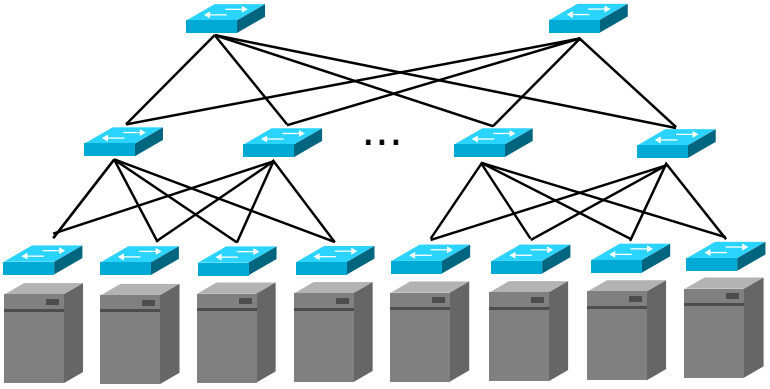


Figure 2.3: Canonical Tree DCN Architecture.

Switch-centric DCNs feature a hierarchical configuration of the network topology, meaning that traffic destined to servers must go to the switches in the core layer of the network, with usually feature large buffers [33]. These architectures feature different designs which came to resolve many issues (e.g., agility, load balancing, and power consumption) that existed in conventional data centers [34].

Switch-centric DCNs focus to horizontal expansion rather than vertical expansion and these architectures are widely used to achieve high performance and flexibility through their ability to provide better scalability and path diversity [33]. In Clos-tree topology, links between the switches on the core and aggregate layers form a complete bipartite graph as shown in Figure 2.4, and edge switches connect to two aggregation switches as in conventional tree architecture [33].

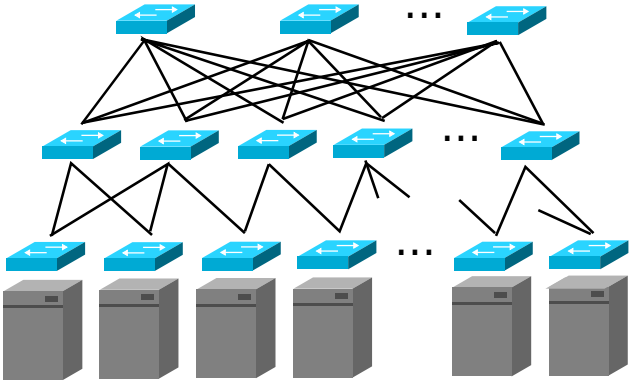


Figure 2.4: Folded Clos-Tree DCN Architecture.

In Fat-tree topology, the number of switch ports determines its size, and can use $k$-port commodity switches uniformly for all layers. As illustrated in Figure 2.5, the topology consists of $k$ *pods*, which are management units interconnected by the $(k/2)^2$ switches that make up the core layer. Each pod consists of $k/2$ edge switches and $k/2$ aggregate switches, and each edge switch, also known as *Top-of-Rack* (ToR), connects to $k/2$ end hosts [11]. Therefore, a $k$-ary FatTree topology is able to support $k^3/4$ hosts [33].

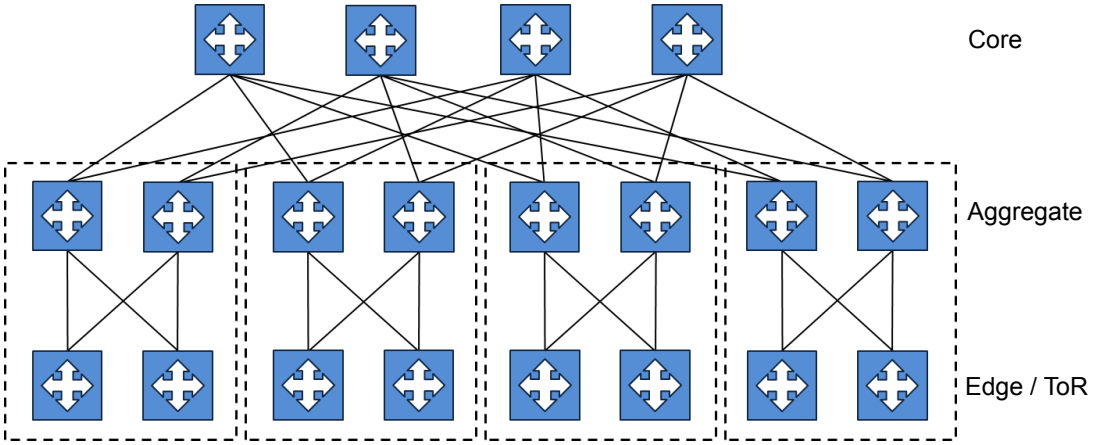



Figure 2.5: A FatTree Topology in a Data Center Network.

In production DCNs, traffic is composed mostly by short-lived and small flows (i.e., mice), but it is contributed mostly by the long-lived and large flows (i.e., elephants)

[30]. In the traffic profile of the data-mining applications and the *Information Technology* (IT) services of larger companies, there are both elephant and mice flows [35]. Mice flows are usually associated with latency-sensitive and bursty applications, such as Voice over IP and search results [36]. The elephants are fewer flows than mice, but they often belong to massive transfers of data, such as in making backups of files [3]. The phenomenon of mice and elephants impacts the overall performance of DCNs: large flows tend to fill the network switch buffers end-to-end, introducing delays to the packets of mice flows that share the same buffers [4].

Existing works deal with the routing of mice and elephant flows in DCNs by dynamically compiling and installing paths for the elephant flows [9, 10], while routing mice flows using static rules provided by multipath routing algorithms, such as the *Equal-Cost Multi-Path* (ECMP). These routing algorithms use multiple paths for sending flows, but their main drawback is that they usually have a longer delay than traditional unicast algorithms. Here, it is noteworthy that an analysis [37] revealed that the correlation between traffic flows in a data center is usually poor.

In most DCN architectures, flow packets are forwarded using *Location-specific Addresses* (LAs) on switches, and *Application-specific Addresses* (AAs) on servers. Therefore, these architectures rely on a directory for AA-to-LA mappings. Switches are not aware of AA addressing, so they use only LAs when forwarding the packets, which are decapsulated at the destination edge switch and delivered to the destination AA server. PortLand (a switch-centric approach), however, uses hierarchical Pseudo-MAC (PMAC) addressing of hosts and VMs for Layer 2 routing. The separation of addressing spaces of switches and servers improves the scalability of protocols such as VL2, as the switches do not have to store forwarding information for a large number of servers. Cloud services instead assign contiguous addresses to VMs placed behind the same edge switch and use wildcard bits for aggregating IP forwarding entries [4].

For typical DCNs, consisting of a two- or three-tier Data Center switching infrastructure, the networks are designed for traffic peak rather than real-time traffic. However, routing algorithms for these architectures are designed primarily for load-balancing and usually are not able to make full use of network resources

[38]. DCN architectures typically face some issues regarding the management and control of physical and virtual resources and the inefficiencies it can lead to [33].

Flow consolidation operations, which allow the direction of DCN flows to a sub-topology with the minimum number of links and switches, does not guarantee full utilization of links [38]. When a sub-topology serves the consolidated flows, not all links work with the same volume of the link capacity. For a set of flows on the same link, the completion time will be the same, regardless of the bandwidth assigned to each flow. Therefore, an exclusive routing must guarantee that each flow will use its path links exclusively, allowing for total link utilization [38].

## 2.3. Traffic Engineering

The *Traffic Engineering* (TE) is an important mechanism to optimize the performance of a data network by dynamically analyzing, predicting, and regulating the behavior of transmitted data [2]. It activates inside the network to perform re-routing and load-balance of flow traffic when an event happens, solving issues such as link overload and improving connectivity [39]. Network managers have exploited TE widely in the past and current data networks, such as ATM and IP/MPLS networks [2], and may solve the problems related to traffic routing [40].

There are several TE techniques, which are also divergent in their operation; traditional TE techniques calculate the paths for routing flows based on simple shortest path routing protocols. These techniques, however, have some issues for network optimization because the traffic in a network is dynamically changing all the time [41]. ECMP, for example, enables and configures the switches with various possible forwarding paths for a given subnet. Therefore, all packets of a flow that arrives are forwarded to the same route, maintaining their arrival order [9]. However, the default segment-routing behavior of exploiting ECMP may lead to several drawbacks, such as higher network resource utilization [40]. They may also present collision of two or more large packet flows with the same output port as shown in Figure 2.6, which may create bottlenecks resulting in a reduction of

the link bisection bandwidth [9].

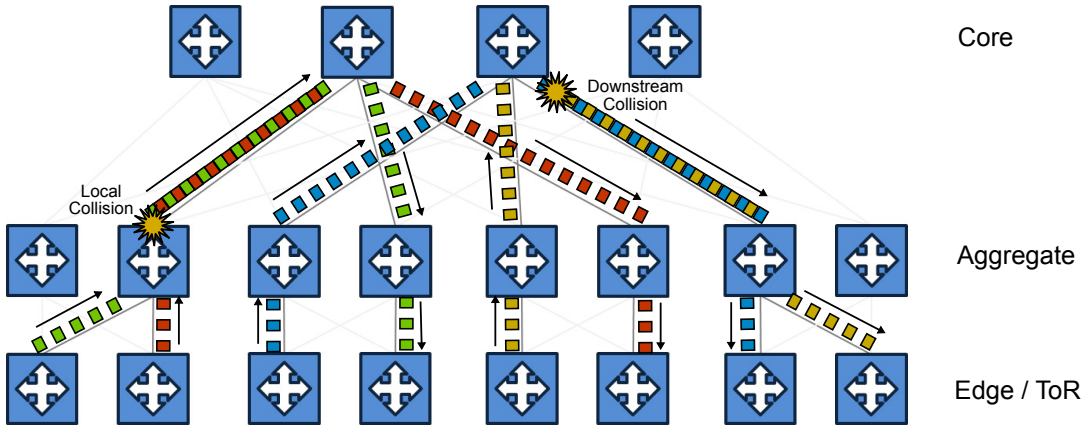



Figure 2.6: Examples of flow collisions in ECMP between the red and green flows and between the blue and yellow flows.

Other TE techniques rely on *Multi-Path Transmission Control Protocol* (MPTCP), which splits traffic evenly by balancing flows across multiple paths, preventing hot spots where congestion occurs [42]. This method allows MPTCP an excellent host-based solution on the transport layer, much like *Data Center TCP* (DCTCP) [11]. However, these techniques lack mechanisms to guarantee that the flows go over different physical network routes [33]. Also, end-users would be required to upgrade to a kernel-code that is compatible with MPTCP [43]. Therefore, an efficient TE algorithm needs to be able to optimize the performance of networks by dynamically allocating bandwidth and changing the route of traffic. Such is the case of TE techniques like Google B4 and Microsoft SWAN [44].

One of the most efficient TE techniques that exist is the Inter-domain SDN, which features the deployment of data plane elements managed by the SDN controller; this enables richer traffic matching that allows a more direct control over the data plane and new applications (*e.g.*, inbound TE, WAN load balancing, and fabric virtualization) [45]. Other techniques based on Inter-domain SDN are usually designed to tackle the challenges to ensure optimal end-to-end QoS for applications [16]. Although there are many proposals of advanced TE techniques in the research literature (*e.g.*, integer programming and multi-commodity flow op-

timization), operators may not have the required knowledge at hand to optimize utilization through these techniques or to improve security [45]. By TE, operators and administrators can achieve the goals of optimizing network performance and network resource utilization [46].

Most TE approaches are limited to a single administrative domain and assume complete knowledge of the underlying substrate topology and the overall network state. However, the rapid development of networks is making network architectures evolve into complex interconnections of domains and layers, delineated by geographic, administrative and economic factors [44]. Existing TE techniques could potentially expose link-flooding attacks, which may also affect severely mice flows. An administrator could exploit TE, and monitor network areas that are persistently affected by link-flooding events, marking such areas as targets, indicating an attack in progress [39]. Since SDN facilitates the carrying out of TE, SDN deployments can be an incentive for operators to start the transition to this new network paradigm. When offering premium services, the SDN controller can reserve some forwarding paths, apply some TE technique and assign a higher priority to traffic related to these services [47].

## 2.4. Traffic Engineering in Data Centers based on Software-Defined Networking

Network operators and administrators use SDN for routing mice and elephant flows in DCNs, since SDN-based switches are becoming more common, and network operators can rely on the functions already provided by SDN [48]. However, the phenomenon of mice and elephant flows impact negatively on the overall performance of SDN-based DCNs: large flows tend to fill the switch buffers end-to-end, introducing delays to the latency-sensitive mice flows that share the same buffers [4]. Moreover, mice flows usually trigger the continuous updating of switching tables when sending the first packet to the controller, which also generates delay [11].

TE is a widely used method for mitigating the events that may negatively affect

mice flows. In SDN-based DCNs, TE methods are necessary to utilize the link bisection bandwidth efficiently. DCN applications also demand TE for achieving routing reliability, load-balancing, and energy-efficiency [49]. TE techniques vary depending on the nature and application of the network, and they are oriented to minimize common issues such as congestion, end-to-end delay, packet loss, energy consumption, and resource utilization [50].

Several TE approaches distinguish between mice and elephant flows in SDN-based DCNs. These techniques dynamically compile and install paths for the elephant flows only, while routing mice flows using static rules provided by baseline methods (e.g., ECMP and VLB) [9, 10]. However, these techniques have some drawbacks in performance, such as the complexity of updating the routing rules continuously in case of dynamic changes in the network state and the difficulty to have redundant paths. Also, these works tend to create a high number of source-destination routing rules, especially in more extensive networks. A switch with many routing rules is a severe scalability problem in DCNs because it increases the time the switch takes to find the routing rule for a specific flow. Such increase in time may cause the loss (dropping) of flows.

Other TE approaches propose to dynamically compile and install the path for both mice and elephants from the SDN controller to tackle with some of the problems presented above [5]. However, these approaches also present some issues. The switch sends the first packet of each incoming flow to the SDN controller, introducing a delay which negatively affects latency-sensitive mice flows [6]. Furthermore, since the controller installs routing rules for each incoming flow, a large number of incoming flows can overload the controller, increasing the processing delay [7]. Therefore, these proposals also present the same scalability problems faced by the switches with many routing rules.

In summary, existing TE approaches offer alternatives to re-route mice flows considering the load on links, the limited memory capacity of switches, and the location of network devices in the topology. However, although there are some TE techniques designed for SDN, they are usually not optimal to meet the requirements of the network. For hybrid SDN networks, SDN-TE services have been designed considering the fixed shortest paths of non-SDN environments, but these

services only handle traffic on packet granularity, without QoS consideration [49].

## 2.5. Final Remarks

We have observed that SDN can address the limitations of traditional network architecture and that DCNs are the main infrastructures in meeting the demands of current network applications. We also observed that network operators use SDN for routing mice and elephant flows in DCNs. However, the performance of SDN-based DCNs is affected by the delays introduced in the latency-sensitive mice flows. Several TE techniques have been developed to overcome these problems, but they present shortcomings related to the large number of routing rules that the switches must handle, leading to significant delays on mice flows. Therefore, a solution is necessary to, first, reduce the delay caused to mice flows by a large number of routing rules in the switches and, second, to avoid sending the first flow packet to the controller.

# Chapter 3

# Related Work

This chapter presents a review of the related work according to the network management works that focus on the management of mice flows in SDN or DCN, and expose their differences with our algorithm in Table 3.1. From there we obtain our observations about these works and highlight what our solution needs to fulfill the objectives previously proposed.

## 3.1. Routing in Data Center Networks

There are several works related to routing in DCNs. Next, we describe some of them. In *"Hedera: Dynamic Flow Scheduling for Data Center Networks"* [9], the authors present a scalable, dynamic flow scheduling system that adaptively develops a multi-stage switching fabric to utilize aggregate network resources efficiently so that could be an efficient solution for mice flow routing. Hedera collects flow information from constituent switches, computes non-conflicting paths for flows, and instructs switches to re-route traffic accordingly. Hedera delivers bandwidth improvements with modest control and computation overhead. This system takes advantage of multiple paths in DCN topologies using ECMP, so flow packets all take the same route, maintaining their arrival order. The evaluation of Hedera reveals better results in bandwidth utilization for *Global First Fit* and *Simulated*

*Annealing*[1] than traditional ECMP. It is important to mention that Hedera only improves the forwarding of flows, Hedera does not reduce the delay produced when sending the first packet of the controller.

In *"PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric"* [5], the authors propose an approach called PortLand that supports a *plug-and-play* large-scale DCN in an increasing trend toward migrating applications, computation, and storage into data centers. PortLand is a set of Ethernet-compatible routing, forwarding, and address resolution protocols with a set of routing goals. This approach employs a lightweight protocol to enable switches to discover their position in the topology, assigning internal *Pseudo-MAC* addresses to all end hosts to encode their place in the topology. Furthermore, this approach imposes few requirements on the underlying switch software and hardware, and the authors hope that it enables a move towards more flexible, efficient and fault-tolerant data centers. The evaluation of Portland exposes that it is an excellent fault tolerance approach when supporting VM migration. Limitations of Portland include the lack of methods to handle mice flows when they go to the controller.

In *"DiFS: Distributed Flow Scheduling for adaptive switching in FatTree data center networks"* [11], the authors present an adaptive switch method for Fat-Tree DCNs, which allows switches to cooperate to avoid over-utilized links and find available paths without centralized control. DiFS runs the *Path Allocation* algorithm on each switch to assign flows to all outgoing links and avoid local flow collisions, the *Imbalance Detection* algorithm to monitor the incoming links, and the *Explicit Adaption* algorithm to change the path of the flows and avoid remote flow collisions. DiFS aims to balance flows among different links and improves bandwidth utilization, while avoiding flow splitting and, therefore, setting limits to packet reordering. This protocol focuses on elephant flows, classifying and spreading them as evenly as possible among all links. DiFS is built in a simulated environment to evaluate its performance and compare it with other routing solutions. Results show that DiFS has a better aggregate throughput than that

[1] Both are essential scheduling algorithms that control edge and aggregation switches dynamically. *Global First Fit* assigns a flow to the first path that can accommodate it using a core switch for each flow, while *Simulated Annealing* performs a probabilistic search to efficiently compute paths for flows, assigning a single core switch for each destination host [9].

of ECMP and achieves higher efficiency in avoiding local and remote collisions. However, DiFS does not provide methods for efficient routing of mice flows in DCNs.

In *"Presto: Edge-based Load Balancing for Fast Datacenter Networks"* [13], the authors propose a mechanism that deals with DCN limitations such as latency-sensitive small flows and drawbacks of earlier proposals such as centralized TE, multipath-aware transport, or expensive specialized hardware. Presto utilizes virtual edge switches (vSwitches) to break each flow into discrete units of packets (*flowcells*) and distributes them evenly for an efficient network load balancing. Also, Presto improves throughput, latency, and fairness in the network and reduces the flow completion time tail for mice flows, demonstrating to be a load balancing system that naturally enhances the latency by uniformly spreading traffic in fine-grained units. Note that Presto is built in an open software-based approach to allow simplified network management, and the vSwitch enables functionality aware of the underlying hardware offload features, making it faster. The evaluation of Presto reveals that it is more optimal than ECMP and MPTCP. Limitations of Presto are related to the delay suffered by the mice flows caused by an overload on the network controller.

In *"A source-controlled data center network model"* [12], the authors propose a source-controlled DCN model. This model applies a new type of source routing address (*vector address*) as the packet-switching label. Statistical studies have revealed that a significant amount of mice flows and their quick arrival and departure may cause overhead if all flows rely on one controller for scheduling and routing decision. Employing SDN technology in DCN has full attention for centralized control and high scalability, but the increasing volume reveals several challenges for the controller. This model communicates sender and receiver by sending a packet to the routers, which then perform a table lookup operation to direct the packet. When it arrives the receiver, it performs the same process in the opposite direction. The theoretical and experimental evaluation results of the source-controlled DCN reveals that it has advantages in energy saving and scalability. However, this model does not solve the problems related to the number of routing rules installed on the switches when mice flows are handled.

## 3.2. Flow Routing in Data Centers based on Software-Defined Networking

There are also several works related to the management of mice flows in SDN or DCN. Next, we describe some of them. In *"MiceTrap: Scalable Traffic Engineering of Datacenter Mice Flows using OpenFlow"* [10], the authors present MiceTrap that is a TE approach oriented to routing mice flows. MiceTrap employs mice flow aggregation together with a weighted routing algorithm. This approach spreads the aggregated flows across multiple links based on dynamically computed ratios to balance the load on them. The Elephant Flow Detection Mechanism module separates the elephant flows. The Aggregation Module is implemented on the switches to reduce the number of rules required. The evaluation results of MiceTrap reveal that it takes full advantage of the data center network bandwidth available in conjunction with other applications (*i.e.* MapReduce and Web Search) and achieves better traffic balancing than conventional ECMP. However, aggregating flows do not reduce the delay generated by sending the first packet of each flow to the network controller.

In *"Efficient traffic splitting on commodity switches"* [51], the authors propose an SDN-based algorithm named Niagara that achieves precise traffic splitting while being extremely efficient in the use of the available rule-table space of the switches. Niagara generates wildcard rules to handle and split the flows according to different target weights while minimizing traffic imbalance (i.e., the portion of traffic sent to the wrong next-hop). It allocates these rules to aggregates based on a tradeoff curve of traffic imbalance versus the number of rules, and shares the rules generates across aggregates with similar weights. Evaluation results reveal that Niagara is highly scalable concerning the number of flow aggregates, and achieves a level of traffic splitting that overperforms ECMP. Although using wildcard rules minimizes the time of finding a specific flow rule, Niagara still lacks methods to reduce the delay of the first flow packet.

In *"Mitigating elephant flows in SDN-based IXP networks"* [52] the authors propose a recommendations system named SDEFIX, based on templates for routing

traffic in SDN-based Internet Exchange Points (IXP). SDEFIX enables network operators to handle the identified elephant flows in an IXP network by using SDN rules, applying templates for routing elephant flows and mitigating their effect in mice flows. This system uses elephant flow snapshots to determine the templates to apply and installs the rules to the controller if the operator confirms the generated recommendation. In particular, SDEFIX achieves a proper routing of elephant flows when using the Best Alternative Path (BAP) template, as it selects the best route for elephant flows to reduce their impact on mice flows and improve overall utilization of the IXP network. Results show that SDEFIX achieves small mitigation times even for a large number of elephant flows when using BAP. However, SDEFIX does not provide methods to handle mice flows when sending the first packet to the controller.

In *"CheetahFlow: Towards Low Latency Software-Defined Network"* [53], the authors propose a system which reduces the extra latency in SDN while preserving its flexibility. CheetahFlow uses a support vector machine to predict successive communication pairs, and a flow state manager to detect elephant flows by sampling and sliding window. It also uses mixed routing path search algorithms to reduce the latency in flow routing, and setups wildcard rules to minimize the latency in flow setup. CheetahFlow reduces the cost of collecting statistics from switches by gathering only from edge switches, guaranteeing bandwidth efficiency between the control plane and data plane. CheetahFlow is implemented in an emulated hierarchy topology and evaluated in aspects such as the flow classifier, throughput, latency, and blocking island efficiency. Extensive experiments show that CheetahFlow prominently reduces latency on mice flows without any loss of flexibility of SDN. However, it lacks methods to avoid sending the first flow packet to the controller.

In *"QoS-based distributed flow management in Software Defined Ultra-Dense Networks"* [54], the authors propose a distributed flow management model for Ultra-Dense SDN based on the queuing theory. This model uses a module to divide the incoming flows according to the characteristics of mice and elephants during controller modeling. It sends high-priority mice flows to the head of the queue without interruption, while the other flows are sent to the queue using the

Poisson process, and estimates the mice and elephant flow loads of each controller separately. The model is implemented on an emulated network and evaluated considering the queue waiting times of mice and elephants. Results reveal that this model can decrease the waiting times of mice and elephant flows, as well as the packet losses of the controllers during an outage compared to the conventional distributed controller implementation. However, it only reduces the delay of mice flows during an outage, not during a normal routing operation.

To sum up, the works afore-described present several methods aimed at routing efficiently mice flows in DCNs, dealing, among other aspects, with the limited memory capacity of the switches, the position of the devices in the topology and the available bandwidth in the links. However, it is necessary a solution, like offered by MiceDCER, that involves the allocation of addresses to the hosts, the internal identification of these addresses without having to send the first flow packet to the controller; and the need to install an efficient routing path (*i.e.,* generate fewer rules to route).

Table 3.1 depicts the differences between the related work by considering the article, its functional area (Flow management - F -, Scalability - S - , Data Center solution - D -, SDN integration - I -, Virtualization - V -), if the method described by the work is suitable for mice flow transmission, and the limitations per work in front of this undergraduate proposal.

Table 3.1 reveals that:

- The works [10, 53] focuses on flow detection and scalability to separate the mice flows and avoid delay and latency that could affect their performance.
- The works [10, 9, 5, 51, 53] have some limitations regarding sending the first packet of mice flows from the switch to the network controller.
- The work [13] uses virtualization to address the mice flow tracking, but it is oriented to general flow transmission and network management.
- The publication [5], is not recommended for mice flow transmission as still have some issues related to the delay of these flows, although it is suitable for virtualization.

| Related Work | | | |
|---|---|---|---|
| **Article** | **Functional Area** | **Transmission** | **Limitations** |
| [9] | F,D | - | Controller Delay |
| [10] | F,S | - | Controller Delay |
| [5] | F,V | - | Controller Delay |
| [51] | F,S,I | ✓ | Controller Delay |
| [52] | F,I | ✓ | Handling Methods |
| [11] | F,D | - | Handling Methods |
| [53] | F,D,I | ✓ | Handling Methods |
| [54] | F,D,I | - | Handling Methods |
| [13] | F,D,V | ✓ | Controller Overload |
| [12] | F,D,I | ✓ | Routing Rules |
| This Proposal | F,I,V | ✓ | Number of Rules |

Table 3.1: Related Work

- The works [52, 11, 54] focus on reducing the impact of elephant flows but lacks methods to handle mice flows appropriately.

## 3.3. Final Remarks

After describing our related work and the limitations presented in front of our proposal, we observe that it is needed a solution that provides efficiency regarding the delay of mice flow routing and that supports virtualization while trying to avoid some of the limitations presented in the afore-described approaches. Thus, the purpose of this undergraduate work is to propose a technique that meets the requirements described for mice flows in SDN-based DCNs.

# Chapter 4

# MiceDCER

In this section, we expose our proposed algorithm. First, we present a motivation for our solution. Second, we expose an overview for outlining our proposal. Third, we introduce our algorithm and prototype to detail MiceDCER.

## 4.1. Motivation

We use three experimental deployments to explain how a switch table with a large number of routing rules impacts negatively on the delay of mice flows in DCNs. The first deployment, as shown in Figure 4.1, measures the Round Trip Time (RTT) in an emulated environment based on Mininet 2.2.2 [55], in which a Ryu controller [56] handles an Open vSwitch [57] and two hosts by a single flow-installer algorithm developed in Python [58].

The second deployment, illustrated in Figure 4.2, measures the RTT in a virtual environment, in which the Ryu controller handles an Open vSwitch 2.5.4 and two virtual hosts (connected through a virtual Ethernet interface) by using the algorithm mentioned above.

The third deployment, as shown in Figure 4.3, is to measure RTT in a physical environment, in which the Ryu controller handles (using the same algorithm

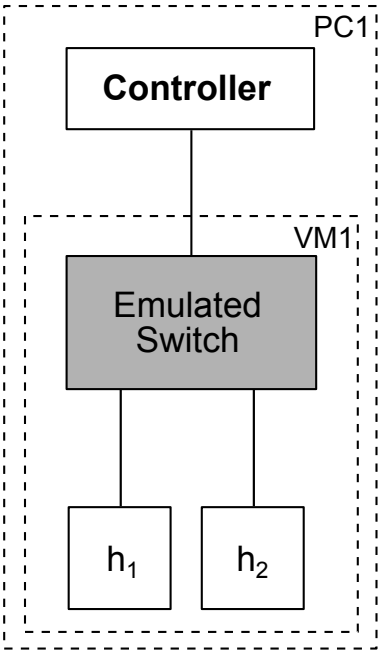



Figure 4.1: Emulated Deployment for RTT Measurement.

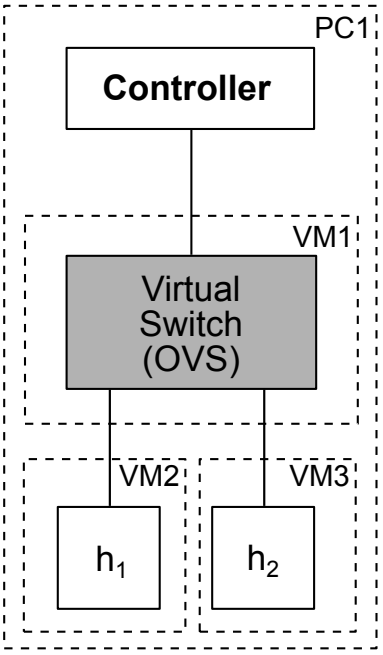



Figure 4.2: Virtual Deployment for RTT Measurement.

that the other deployments) an HP 2920 switch that, in turn, handles communication between two physical hosts through an OpenFlow VLAN interface. We used computers with Intel Core i5 2.40GHz (4 cores) processor, 3GB RAM, and Lubuntu 16.04 LTS operative system to run the emulated environment, the Open vSwitches, and the Ryu Controller.

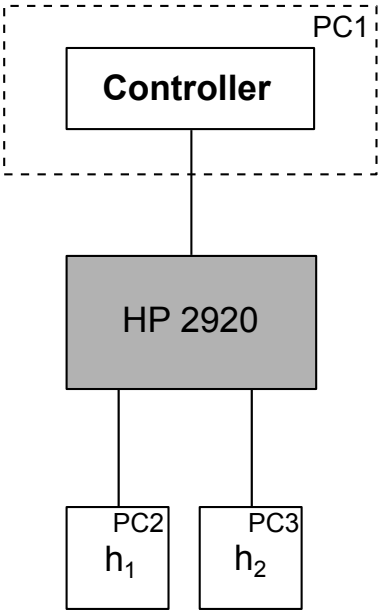



Figure 4.3: Physical Deployment for RTT Measurement.

In the three deployments, we measure the average RTT for each number of rules $r \in R = \{1000, 2000, ..., 16000\}$, where two rules (*i.e.,* working rules) are intended to communicate the hosts, and the remaining are non-working rules. We installed $T$ times the $r$ flow rules in the switch, setting up the working rules at the beginning, at the middle, and at the end of the switch routing table for each $t \in T$, and took the RTT measurement $N$ times for each $t \in T$. The experiments were conducted by pinging from a host to another one. In particular, we measure the average RTT in the first host after receiving the reply packets from the switch.

For each deployment afore-described, Figure 4.4 depicts the results for $T = 30, N = 30$ when the working rules are at the beginning, middle, and ending of the switch table. These results reveal that the average RTT for the emulated de-

ployment stays within a range of $[0.08 - 0.1]$ ms regardless of the number of rules installed. Similarly, the average RTT for the virtual deployment stays at ∼1.3ms. This higher RTT value is caused by the virtual links used to connect the virtual machines. For the physical deployment, there is a significant change. First, the RTT increases (from ∼0.8ms to ∼2.1ms) when the number of rules grows. Second, the location of the rules also impacts the RTT. When the working routing rules are at the beginning of the switch table, the RTT is ∼1.1ms. If the rules are the end, the RTT is ∼2.1ms.

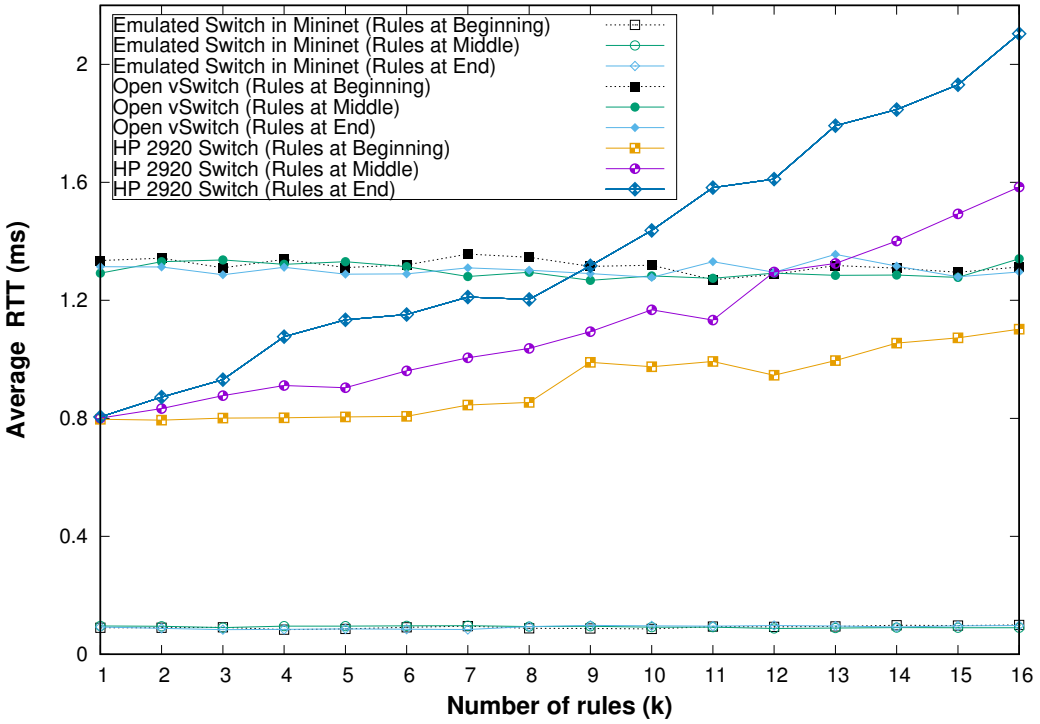



Figure 4.4: Average RTT for different number of rules installed on the switch. Here $T = 30$ and $N = 30$.

From the experiments above, we learned that, first, RTT in emulated and virtual deployments remains nearly constant while varying the number of installed rules. Second, in a physical deployment, however, the amount of limited memory of switches implies that the number of installed rules influences the delay of the packets [2] [59] of mice flows. Thus, we can state that the number of rules is an

issue that needs to be addressed to reduce the delay of packets in mice flows in DCNs based on SDN.

## 4.2. Overview

We present an algorithm named MiceDCER, focused on reducing the delay of mice flows in DCNs. In particular, MiceDCER addresses three issues:

- The allocation of addresses to the hosts.
- The internal identification of these addresses without having to send the first packet of a flow to the controller.
- The need to install an efficient routing path.

Moreover, the controller also needs to know the possible routes for the flows to install the required rules on the switch tables. When a switch is inserted into the topology, the controller executes an internal method called *Topology Discovery*. This method checks which switches are connected and how they connect to each other, giving the controller an overview of the topology.

Figure 4.5 presents the process to generate and install rules in MiceDCER that aims at tackling the delay of packets in the mice flows. The MiceDCER process involves the following tasks: Generate, Define, Install, and Update. It is noteworthy that these tasks are executed after the SDN controller obtains the topology (Topology Overview).

- ***Generate Address.*** MiceDCER generates the PMAC of the receiver edge switch (i.e., the switch that received the flow intercepted by the controller) from the Topology Overview. This task is composed of two others. The first one generates the PMAC of the edge switches based on their position in the topology, while the second task stores the new PMACs in a table, associating them with the corresponding actual MAC (AMAC) of each switch.

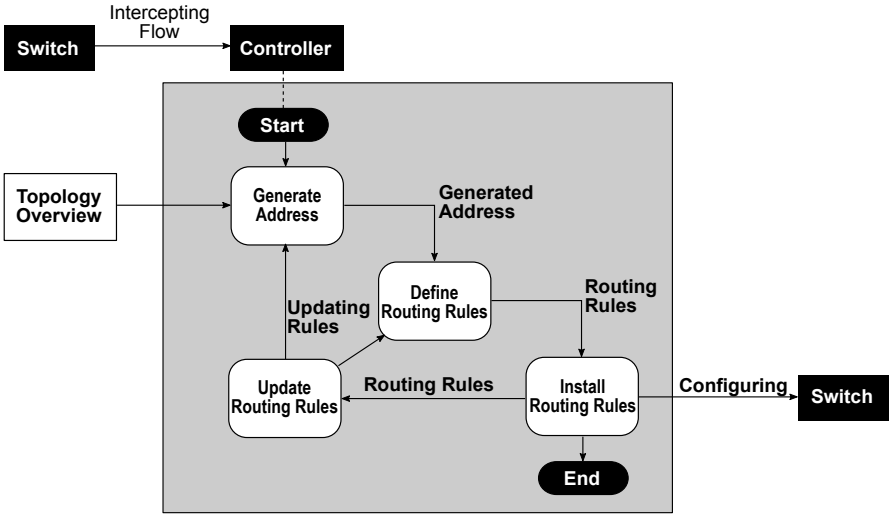



Figure 4.5: Process to generate and install rules in MiceDCER.

- ***Define Routing Rules.*** Here, the MiceDCER algorithm (see Algorithm 1) provides the generated PMACs of the switches to the controller for defining the set of rules to install $R_{ins}$. This definition is performed by considering the source MAC and IP addresses of the messages, as well as the input port of the switches.
- ***Install Routing Rules.*** MiceDCER instructs to the controller to install the rules $R_{ins}$ in the edge switches.
- ***Update Routing Rules.*** When a significant change in the network occurs, the controller updates the defined rules $R_{ins}$. This updating may imply the generation of new PMACs or the definition of new rules.

It is noteworthy that in MiceDCER, we are assuming that we are not going to install ARP routing rules on the core or aggregate switches, but that they will flood their ports when they receive ARP messages. Each request ARP message is expected to be answered with a reply message, which is determined by the value of its Operation Code field on the ARP packet data. This value is essential when it comes to the management of ARP messages required to install the rules on the switches.

Each core switch will have a rule that will connect to each *pod* or networking group. The aggregate switches will have a wildcard group rule that provides a connection with the core layer, and a rule for each link they have with the edge layer. The installation of rules in the edge switches is more complicated than it is in the upper layers because the PMAC-AMAC conversion is performed on the edge layer. When the packet comes from the host, the switch rewrites the source MAC field with its associated PMAC, and before sending the packet to its final destination, it rewrites the destination MAC field with the host AMAC before routing it through its respective port.

## 4.3. Algorithm

- ***Inputs.*** Our algorithm receives as input an intercepted ARP message which contains the primary data $A = \{opcode, eth\_src, eth\_dst, mac\_src, in\_port\}$, where $opcode$, $eth\_src$, $eth\_dst$, $mac\_src$, and $in\_port$ are the Operation Code, Source IP Address, Destination IP Address, Source MAC, and the switch input port, respectively. While the input port does not make part of the ARP data of the message, the algorithm still uses this value for generating rules. MiceDCER also receives a set of edge switches $T = \{sw_i | sw_i = (id, pmac)\}$ from the Topology Overview. Here, $id$ represents the defined ID of the switch $sw_i$, while $pmac$ is the PMAC that the algorithm assigns on $sw_i$. It is important to highlight that the defined ID (i.e., *datapath ID*) for each $sw_i \in T$ is a 64-bit defined field, where the 48 least significant bits (LSB) correspond to the switch MAC, while the 16 most significant bits (MSB) depend on the implementation of the switch, which varies with each model. The algorithm also has an ARP stale time threshold $\Theta$ which determines how often the connection between the two hosts should be checked.
- ***Outputs.*** The outputs of MiceDCER are the PMACs and routing rules to assign to the edge switches. The algorithm provides two tables for storing PMACs: the first one associates the PMACs of the hosts (or virtual machines) according to their IP addresses and AMACs, while the second stores the PMAC headers corresponding to the AMAC of the edge switches.

**Algorithm 1** Rule installation algorithm.

$E$: Set of edge switches (ToR)
$sw$: Message receiver switch $sw \in E$
$A$: Intercepted ARP message data
$I$: List of known IP addresses
$T_H$: Host PMAC addressing table
$T_E$: Edge switch PMAC addressing table
$\Theta$: ARP stale time threshold

**Generate initial rules for edges:**
**for** each $e \in E$ **do**
    $field \leftarrow \{fieldtypes.ARP\}$;
    $actions \leftarrow [ACTION\_CONTROLLER]$;
    $installRule(e, field, actions)$;
**end for**

**Intercepted message management:**
**procedure** MESSAGEMANAGEMENT($sw, A$)
    **if** $A.dst\_ip \notin I$ **then**
        $actions \leftarrow [ACTION\_FLOOD]$;
        $out \leftarrow packetOut(A, actions)$;
        **for** each $e \in E$ **do**
            **if** $e \neq sw$ **then**
                $generateRequests(e, A)$;
            **end if**
        **end for**
    **else**
        **if** $time(A.src\_ip, A.dst\_ip) > \Theta$ **then**
            $checkHostConnection(A.dst\_ip)$
        **else**
            $actions \leftarrow [A.in\_port]$;
            $out \leftarrow packetOut(reply(A), actions)$;
        **end if**
    **end if**
    $sw.sendMessage(out)$;
**end procedure**

**Generate table entries:**
**procedure** GENERATEENTRIES($sw, A$)
    **if** $A.eth\_src \notin T_H$ **then**
        $pmac \leftarrow generatePmac(sw)$;
        add $\{A.eth\_src, pmac\}$ to $T_H$;
        **if** $sw.id \notin T_E$ **then**
            $pmacHeader \leftarrow generateHeader(pmac)$;
            add $\{sw.id, pmacHeader\}$ to $T_E$;
        **end if**
    **end if**
**end procedure**

MiceDCER generates and assigns the PMACs using the form $pod.pos.port.vmid$. Here, $pod$ reflects the pod number of the edge switch, while $pos$ is its position within the pod. $port$ is, from its local view, the port number to which the host connects to the switch, and $vmid$ is the identifier that corresponds to the virtual machine inside the physical machine (or physical hosts on the other side of the bridge). The PMAC header represented in the switch table will have the form $pod.pos.*.*$.

- ***Procedures.*** The algorithm procedures are: generation of initial rules for the edge switches, intercepted message management, and generation of table entries.

- ***Generation of initial rules for the edge switches.*** MiceDCER initially installs the routing rules for the edge switches with the ARP field type, to allow the controller to intercept the ARP messages that arrive at the switch. The algorithm then performs the following procedure to install the rules on the switch tables.

- ***Intercepted message management.*** If the controller does not know the IP destination address of the intercepted ARP request message, the controller indicates the receiver switch (i.e., the switch that received the message intercepted by the controller) to flood (i.e., sends the packet to all ports excluding the input port), and instructs the other edge switches to flood with requests. If the controller knows the IP address, it sends a reply ARP message back to the source host, or it checks if the destination host is still connected after the ARP stale time has passed.

- ***Generation of table entries.*** If the source IP address of the intercepted message does not exist in the host PMACs table, MiceDCER proceeds to generate the PMAC and insert the entry into the table, associating it with the source IP address. In this procedure, if the defined ID of the receiving switch is not in the switch PMACs table, the algorithm generates the PMAC header to add the entry, avoiding to carry out this process again if the same switch receives the flows of several directly connected hosts.

## 4.4. Prototype

MiceDCER relies on information obtained from ARP messages for reducing the number of rules installed by the controller. The algorithm uses the AMAC-PMAC association to rewrite the packet fields with the PMAC generated from the host when the switch receives the packet from the source or when sending it to its destination. MiceDCER is similar to PortLand, in that it assigns PMAC addresses to the end hosts according to their position in the topology [5]. However, unlike PortLand, MiceDCER assigns the PMACs before the hosts start communicating, minimizing the delays in the transmission of mice flows.

Unlike other proposals, MiceDCER is complemented by the *Topology Discovery* method to give the controller an overview of the topology. Once this discovery finishes, we check the connections between switches with LLDP packets to determine which layer they belong to in the topology, and we then run our algorithm to populate switching tables with routing rules. Finally, unlike Hedera [9] or Presto [13] that focus on forwarding and splitting flows, respectively, MiceDCER focuses on the establishment of the most viable routes for mice flows, because it uses group rules to select the outgoing port when routing the flow packet to the upper layers.

Algorithm 2 illustrates the primary process of MiceDCER, which is the installation of the PMAC rules for IP packets in the routing tables of the edge switches in the topology. It installs a rule on Table 0 for each PMAC-AMAC association, where the switch rewrites the source MAC field of the flow packet with the host PMAC before sending it to Table 1. This second table has another installed rule, where the switch rewrites the destination MAC field back with the host AMAC, before sending the flow packet to the output port connecting to the destination host. The IP rules have a lower priority than the ARP rules and ensure the proper connection between the hosts in the topology.

We implement MiceDCER on the top of a Python-based SDN controller. In particular, we use the libraries offered by the Ryu 4.23 framework to implement the Algorithms 1 and 2. This framework supports OpenFlow and ARP that are the fun-

**Algorithm 2** Python code for installing the rules based on PMAC.

```
1: def install_rule_pmac(self, dp, amac, pmac, port):
2:     ofp_parser = dp.ofproto_parser
3:     # Install rule for source MAC change on table 0
4:     match = ofp_parser.OFPMatch(eth_type = ether_types.ETH_TYPE_IP,
           eth_src = amac)
5:     actions = [ofp_parser.OFPActionSetField(eth_src = pmac)]
6:     self.add_flow_table(dp, 20, 1, match, actions)
7:     self.logger.info("Rule installed on datapath (%s) [Table 0] : "+
           "\n : [IPv4] Src : %s -> eth_src = %s, Table 1",
           dp.id, amac, pmac)
8:     # Install rule for destination MAC change on table 1
9:     match = ofp_parser.OFPMatch(eth_dst = pmac)
10:    actions = [ofp_parser.OFPActionSetField(eth_dst = amac),
11:        ofp_parser.OFPActionOutput(port)]
12:    self.add_flow(dp, 10, match, actions, 1)
13:    self.logger.info("Rule installed on datapath (%s) [Table 1] : "+
14:        "\n : [IPv4] Dst : %s -> eth_dst = %s, Table 1",
15:        dp.id, pmac, amac)
```

damental protocols of MiceDCER. The MiceDCER implementation is available in [60].

## 4.5. Final Remarks

We observe that after realizing experiments on three different deployments, we found that the delay increases when installing many routing rules in a physical environment. We also realize that is necessary to address several issues when focusing on reducing the delay of mice flows, and that we can use an internal controller method to find which switches are connected and how they connect between them. Finally, we observe that through AMAC-PMAC association, wildcard rules and group rules, we can reduce the number of rules installed in each of the switches in the topology.

# Chapter 5

# Evaluation

This chapter presents the evaluation of our proposed solution. First, we describe the testbed of MiceDCER. Second, we expose the process done by the algorithm to install the flow rules on the switch tables. Third, we compare the results obtained with other routing protocols to check that it reduces the number of rules.

## 5.1. Testbed

We evaluate our algorithm analytically to verify if it reduces the delay significantly in the flow packets compared with other routing protocols based on IP or MAC addresses. For this evaluation, we set up an emulated environment as shown in Figure 5.1. To carry out the evaluation, first, we check the initialization of the topology with MiceDCER to make sure it installs the correct rules; we use a Fat-Tree topology since it is the most common topology used in DCNs. Second, we calculate the number of switches for each layer in our topology, and the number of rules installed by the three evaluated solutions namely, MiceDCER, MAC routing, and IP routing.

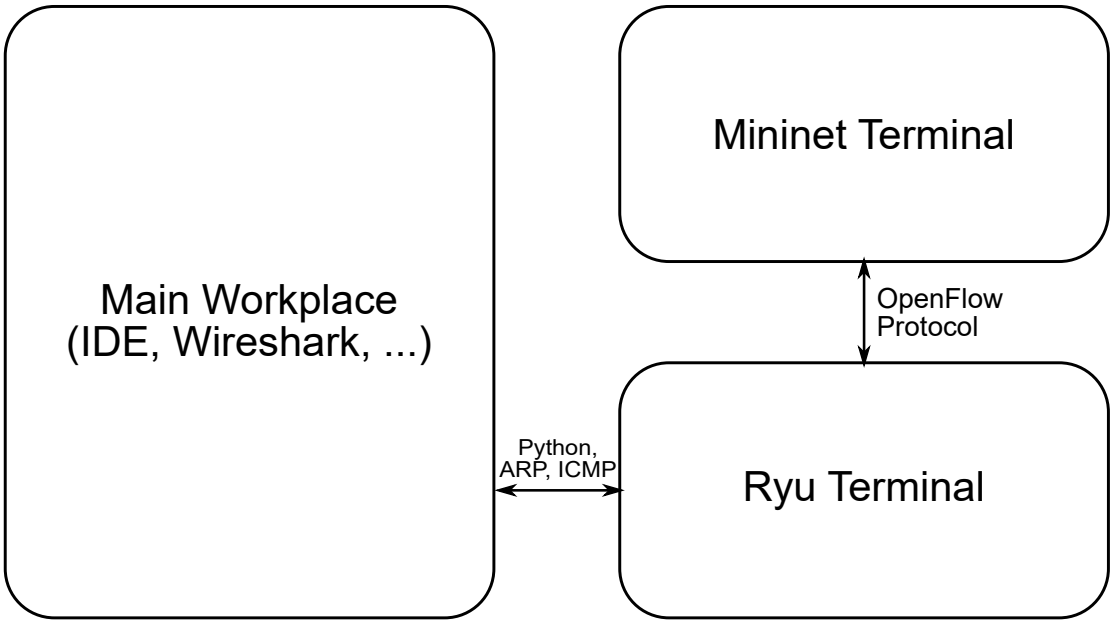



Figure 5.1: MiceDCER Testbed.

## 5.2. Results

In this section, we present the evaluation results of MiceDCER. First, we describe the process of initializing the topology. Second, we calculate the number of routing rules installed per switch for each of the three layers in the topology and compare the results obtained with other routing proposals.

### 5.2.1. Topology

Figure 5.2 illustrates the interception of messages ARP that MiceDCER performs. Our algorithm uses LLDP (*Link Layer Discovery Protocol*) packets to indicate the internal position of the switches in the topology. When the controller (*i.e.,* Ryu) starts running and activates the option of observing the links, MiceDCER automatically installs the rules that allow the controller to intercept the ARP packets at the edge switches. After running *Topology Discovery* to check the switches and the connections between them, MiceDCER assigns the pod and position values of the edge switches to encode their position within the topology.

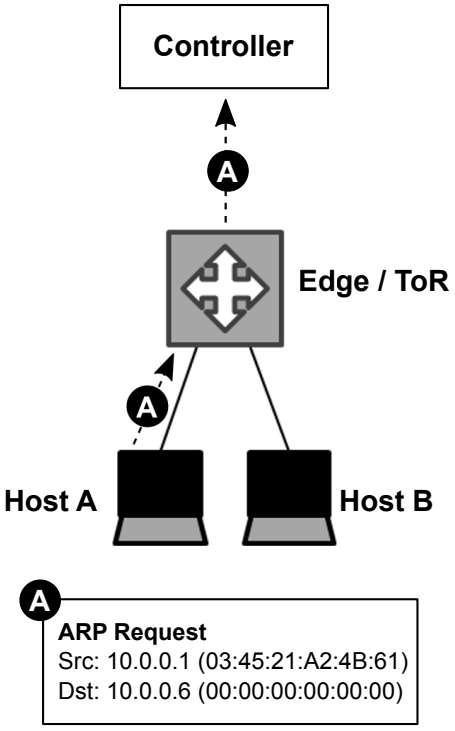



Figure 5.2: Interception of ARP.

### 5.2.2. Number of Rules

We first calculate the total number of routing rules that MiceDCER installs in the switches within the topology. The total amount of rules per switch (including group rules) for $k$-ary FatTree topology can be obtained by using the Equations (5.1), (5.2) and (5.3) for core, aggregate, and edge layers respectively:

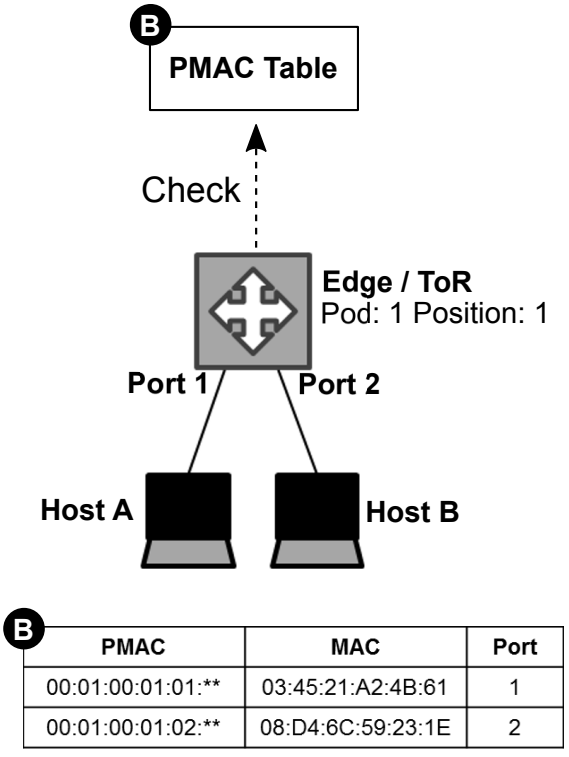
$$C_{DC} = 3 + k \tag{5.1}$$

$$A_{DC} = 5 + \frac{k}{2} \tag{5.2}$$

$$E_{DC} = 6 + 2\sum_{h=1}^{k/2} M_h \tag{5.3}$$

Each of the $k^2/4$ core switches connects with each of the $k$ pods. Each of the $k^2/2$ aggregate switches has a wildcard rule which routes to the core layer through a group table and a rule for each of the $k/2$ connections with the edge switches. Each of the $k^2/2$ edge switches has two routing tables, where the first table contains the rules for matching the source AMAC, and the second table has the rules for matching the destination PMAC. The number of rules installed for each table is equal to the number of VMs $M_h$ for each host $h$ connected to the rack using a bridged adapter, as the VMs send the packets with their own MAC. All the switches also have extra rules for table-miss action and ARP management.

Figure 5.3 illustrates the PMAC-AMAC association that MiceDCER carries out. The edge switches rewrite the source MAC field with the host PMAC when receiving the packet, and rewrite the destination MAC field back to the host AMAC before sending the packet out.




| PMAC | MAC | Port |
|---|---|---|
| 00:01:00:01:01:** | 03:45:21:A2:4B:61 | 1 |
| 00:01:00:01:02:** | 08:D4:6C:59:23:1E | 2 |

Figure 5.3: PMAC-AMAC Association.

Other approaches install a routing rule depending on the IP address, rather than the MAC of the host or VM [30]. The total amount of rules for IP-based routing, assuming the use of wildcard rules and *Topology Discovery*, is given by the Equations (5.4), (5.5) and (5.6):

$$C_{IP} = 2 + k \tag{5.4}$$

$$A_{IP} = 3 + k \tag{5.5}$$

$$E_{IP} = 3 + \frac{k}{2} + \sum_{h=1}^{k/2} M_h \tag{5.6}$$

The total amount of rules for MAC-based routing is given by Equations (5.7), (5.8) and (5.9):

$$C_{MAC} = \frac{k^3}{8} \tag{5.7}$$

$$A_{MAC} = \frac{k^3}{8} \tag{5.8}$$

$$E_{MAC} = \frac{k^3}{4} \tag{5.9}$$

Figure 5.4 presents the topology elements in a typical Fat-Tree topology when the number of hosts to handle grows. These results reveal that the number of switches grows significantly regarding the number of hosts. Furthermore, $1/5$ of switches in the topology are in the core layer, distributing the remaining switches between the core and aggregate layers.

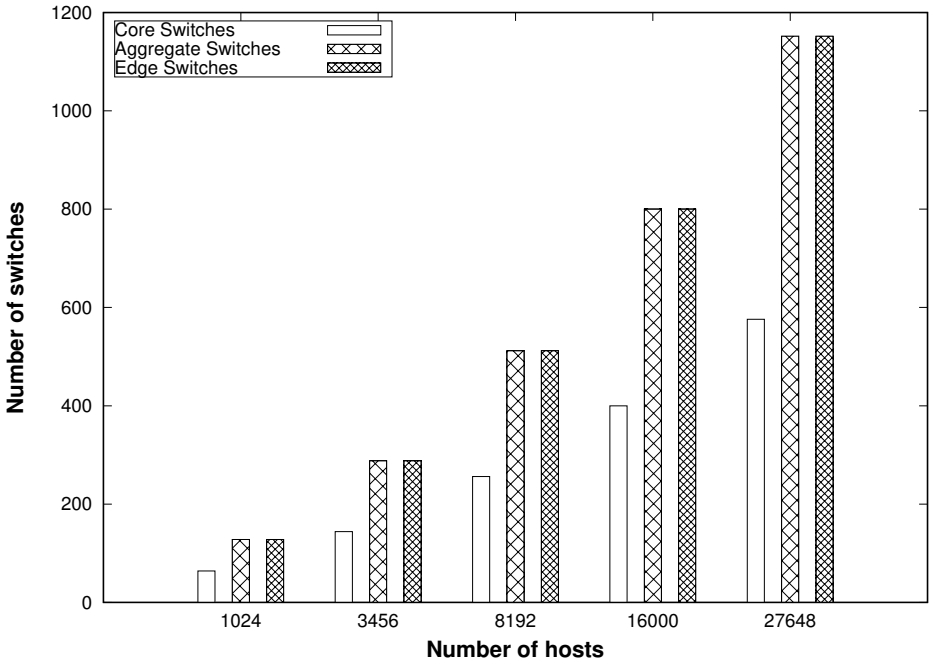



Figure 5.4: Topology Elements.

Figure 5.5 presents the number of rules per edge switch generated by MiceDCER, IP-based and MAC-based routing when the number of hosts grows (recall that the edge switches grows too, Figure 5.4). We assume we are using eight non-bridged VMs per host. Results reveal that MAC-based and IP-based routing installs more rules than MiceDCER (*e.g.,* $\sim$1500 rules per edge switch when using 27648 hosts). Considering these results, we can conclude that MiceDCER reduces the number of rules per edge switch significantly when compared with other routing solutions.

Figure 5.6 presents the number of rules per aggregate switch generated by MiceDCER, IP-based and MAC-based routing when the number of hosts grows. These results also reveal that MAC-based routing installs much more rules than MiceDCER. The IP-based routing installs approximately the double of rules than MiceDCER. Thus, we can conclude that MiceDCER reduces the number of rules per aggregate switch significantly when compared with the MAC-based and IP-based routing.

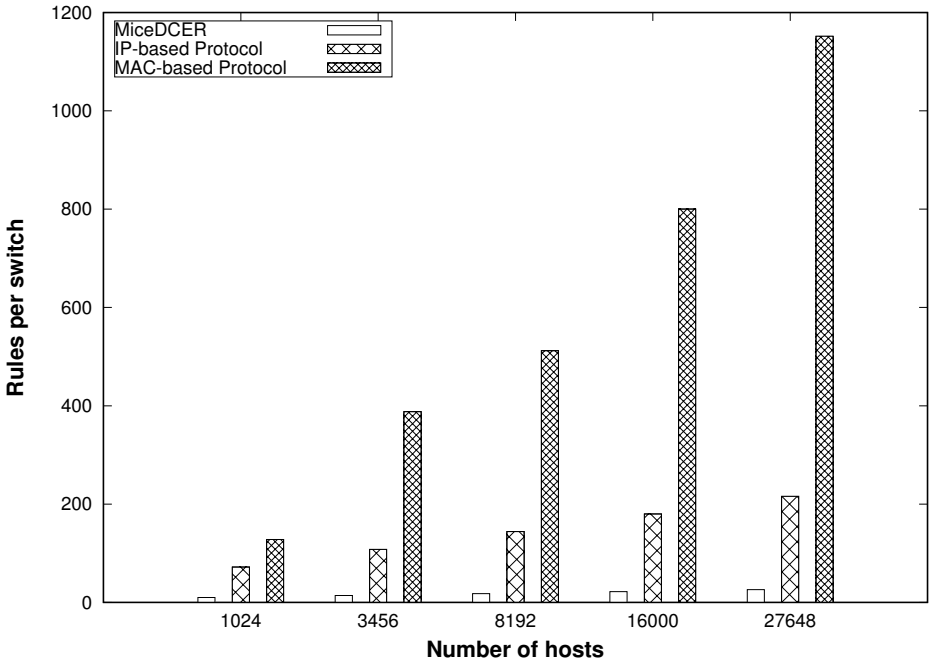



Figure 5.5: Rules on Edge Switches.

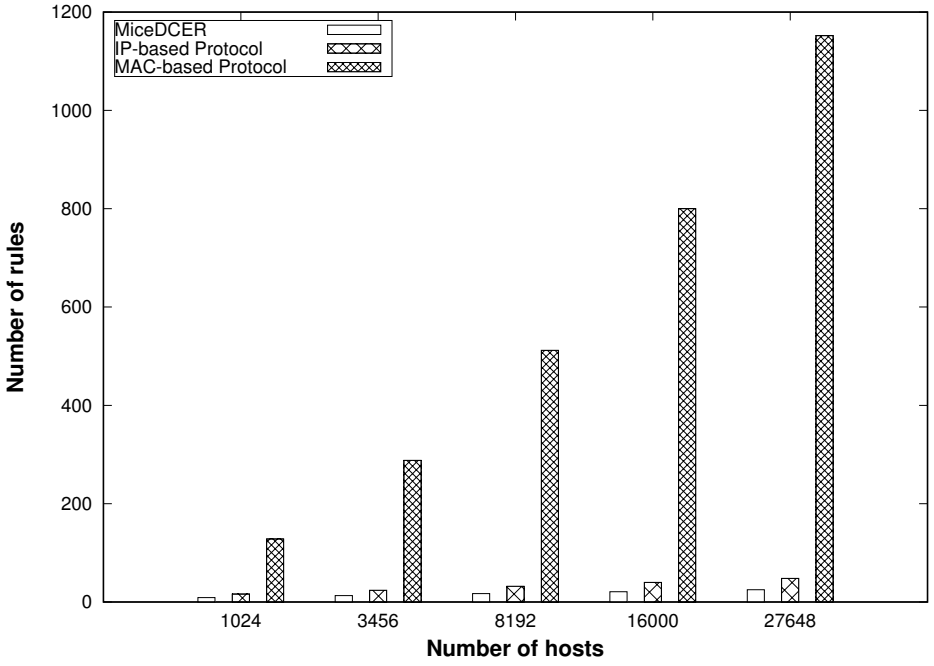



Figure 5.6: Rules on Aggregate Switches

Figure 5.7 presents the number of rules per core switch generated by MiceDCER, IP-based and MAC-based routing when the number of hosts grows. These results reveal again that MAC-based routing installs more rules than MiceDCER. In turn, the IP-based routing installs about the same amount of rules as MiceDCER. We can conclude that MiceDCER reduces or at least generates the same number of routing rules to install in the core switches.

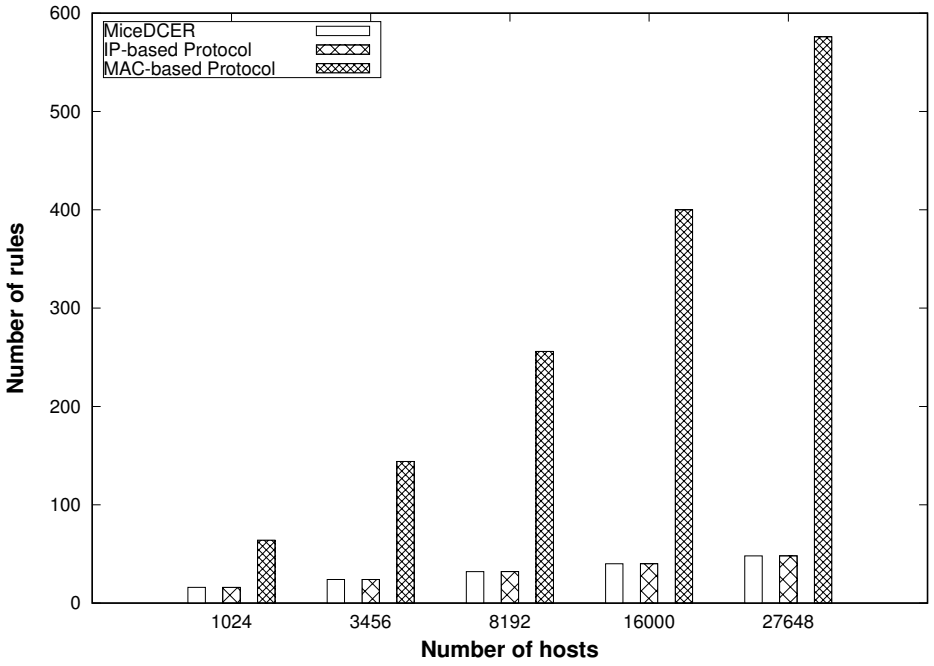



Figure 5.7: Rules on Core Switches.

To sum up, regarding the number of rules to install, MiceDCER always outperforms the performance of MAC-routing in SDN-based DCNs operating a Fat-Tree topology. In turn, the IP-based protocol also installs more rules than MiceDCER, in particular, in the edge and aggregate layers. Considering that MiceDCER installs fewer routing rules in switching tables than traditional routing protocols, we can conclude that it contributes to reducing the delay of mice flows.

## 5.3. Final Remarks

We observed that the methods used by our algorithm contribute to reducing the number of rules on switch tables efficiently. We already demonstrated in Chapter 4 that the number of rules impacts the delay of mice flows so that reducing the number of rules installed contributes to the optimization of the network regarding the delay. We can conclude that MiceDCER is a solution for efficient routing in SDN-based DCNs.

# Chapter 6

# Conclusions and Future Work

In this chapter, we start by answering the proposed research question. Then we present the main conclusions obtained during evaluation. Finally, we outline ideas for future work.

## 6.1. Conclusions

This work presented the proposed solution to answer the research question: **How to route efficiently, in terms of delay, mice flows in Data Centers based on Software-Defined Networking?**

A relevant problem affecting the overall performance of SDN-based DCNs is the delay introduced in latency-sensitive mice flows by the logically centralized controllers when more massive elephant flows tend to fill the network buffers. Several works aimed to resolve this problem, but they present drawbacks such as scalability issues and delays when sending the first flow packet to the controller. Aiming at overcoming this problem, in this work, we present MiceDCER, an algorithm which aims to route efficiently mice flows in Data Centers based on SDN.

In particular, our algorithm, first, installs rules relying on the information obtained from ARP messages. Second, it takes advantage of *Topology Discovery* to identify the position of the switches and install the appropriate rules for improving rout-

ing. Moreover, MiceDCER generates wildcard rules to save memory in switching tables.

By analytical results compared with other routing protocols, we have demonstrated that MiceDCER significantly reduces the number of rules installed in switches and, therefore, contributes to reducing the delay of the latency-sensitive mice flows.

## 6.2. Future Work

According to with the work done for developing this project, we expose some ideas for future work. These ideas are outlined below:

- Evaluate the performance of MiceDCER in other types of networks (*e.g.,* SDWAN and SDWLAN) that follows the SDN paradigm.
- Implement MiceDCER for other SDN controllers to expand further the possibilities of an efficient routing of mice flows in SDN and DCN.

# Bibliography


[1] H. Kim and N. Feamster, “Improving network management with software defined networking,” *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, mar 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6461195/

[2] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, “A roadmap for traffic engineering in SDN-OpenFlow networks,” *Elsevier Computer Networks*, vol. 71, pp. 1–30, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614002254

[3] M. Afaq, S. U. Rehman, and W.-C. Song, “A Framework for Classification and Visualization of Elephant Flows in SDN-Based Networks,” *Procedia Computer Science*, vol. 65, pp. 672–681, 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1877050915028410

[4] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, “Data Center Network Virtualization: A Survey,” *IEEE Communication Surveys & Tutorials*, vol. 15, no. 2, pp. 1–20, 2012.

[5] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, “PortLand : A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric,” *SIGCOMM’09*, pp. 39–50, 2009.

[6] B. Nunes Astuto, M. Mendonça, X. Nam Nguyen, K. Obraczka, and T. Turletti, “A Survey of Software-Defined Networking: Past, Present,

and Future of Programmable Networks," *HAL Archive*, vol. 16, no. 3, pp. 1617–1634, 2014. [Online]. Available: https://hal.inria.fr/hal-00825087v5

[7] P. Song, Y. Liu, T. Liu, and D. Qian, "Controller-proxy: Scaling network management for large-scale SDN networks," *Elsevier Computer Communications*, vol. 108, pp. 52–63, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804517302485

[8] A. K. Koohanestani, A. G. Osgouei, H. Saidi, and A. Fanian, "An analytical model for delay bound of OpenFlow based SDN using network calculus," *Journal of Network and Computer Applications*, vol. 96, pp. 31–38, oct 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804517302485

[9] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10)*, vol. 10, pp. 1–15, 2010. [Online]. Available: http://bnrg.cs.berkeley.edu/{~}randy/Courses/CS294.S13/7.3.pdf

[10] R. Trestian, G.-M. Muntean, and K. Katrinis, "MiceTrap: Scalable traffic engineering of datacenter mice flows using OpenFlow." *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM2013)*, pp. 904–907, 2013. [Online]. Available: https://pdfs.semanticscholar.org/5b37/28ceb3ceb86c5cdf3c073220187d5eff3825.pdf

[11] W. Cui, Y. Yu, and C. Qian, "DiFS: Distributed Flow Scheduling for adaptive switching in FatTree data center networks," *Elsevier Computer Networks*, vol. 105, pp. 166–179, 2016. [Online]. Available: https://ac.els-cdn.com/S1389128616301839/1-s2.0-S1389128616301839-main.pdf?{_}tid=8821d2c2-d160-11e7-a28e-00000aab0f01{&}acdnat=1511560007{_}c66cc98783a1bbe3a127b8fb4e1ac40e

[12] Y. Yu, M. Liang, and Z. Wang, "A source-controlled data center network model," *PLOS ONE*, vol. 3, no. 22, pp. 1–16, 2017. [Online]. Available: http://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0173442

[13] K. He, E. Rozner, K. Agarwal, W. Felter, and J. Carter, “Presto: Edge-based Load Balancing for Fast Datacenter Networks,” *ACM SIGCOMM Computer Communication Review - SIGCOMM’15*, vol. 45, no. 4, pp. 465–478, 2015. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p465.pdf

[14] E. Siami-Irdemoosa, S. R. Dindarloo, and M. Sharifzadeh, “Work breakdown structure (WBS) development for underground construction,” *Automation in Construction*, vol. 58, pp. 85–94, 2015. [Online]. Available: http://ac.els-cdn.com/S0926580515001594/1-s2.0-S0926580515001594-main.pdf

[15] N. Feamster, J. Rexford, and E. Zegura, “The Road to SDN: An Intellectual History of Programmable Networks,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, 2014.

[16] M. Karakus and A. Durresi, “Quality of service (qos) in software defined networking (sdn): A survey,” *Journal of Network and Computer Applications*, vol. 80, pp. 200–218, 2017.

[17] J. Chen, X. Zheng, and C. Rong, “Survey on software-defined networking,” *IEEE Communication Surveys & Tutorials*, vol. 17, no. 1, pp. 115–124, 2015.

[18] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, “Onix: A Distributed Control Platform for Large-scale Production Networks,” *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation - OSDI’10*, vol. 10, pp. 1–14, 2010. [Online]. Available: https://www.usenix.org/legacy/events/osdi10/tech/full{_}papers/Koponen.pdf

[19] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, “BalanceFlow: Controller load balancing for OpenFlow networks,” in *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*. IEEE, oct 2012, pp. 780–785. [Online]. Available: http://ieeexplore.ieee.org/document/6664282/

[20] D. Erickson, “The Beacon OpenFlow Controller,” 2010. [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home

[21] Z. Jingjing, C. Di, W. Weiming, J. Rong, and W. Xiaochun, “The deployment of routing protocols in distributed control plane of SDN.” *The Scientific World Journal*, vol. 2014, p. 918536, 2014. [Online]. Available: http://www.hindawi.com/journals/tswj/2014/918536/http://www.ncbi.nlm.nih.gov/pubmed/25250395

[22] O. Salman, I. Elhajj, A. Chehab, and A. Kayssi, “IoT survey: An SDN and fog computing perspective,” *Elsevier Computer Networks*, vol. 143, pp. 221–246, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128618305395

[23] A. Mayoral, R. Vilalta, R. Muñoz, R. Casellas, and R. Martínez, “SDN orchestration architectures and their integration with Cloud Computing applications,” *Elsevier Optical Switching and Networking*, no. 26, pp. 2–13, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.osn.2015.09.007

[24] M. Karakus and A. Durresi, “A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN),” *Elsevier Computer Networks*, vol. 112, pp. 279–293, 2017.

[25] K. Reilly, “Software-Defined Networking – SDN Versus Legacy Network Infrastructure,” 2015. [Online]. Available: http://infrastructuretechnologypros.com/software-defined-networking-sdn-versus-legacy-network-infrastructure/

[26] Y. Lu, Z. Ling, S. Zhu, and L. Tang, “SDTCP: Towards Datacenter TCP Congestion Control with SDN for IoT Applications,” *Sensors*, vol. 17, no. 1, pp. 1–20, jan 2017. [Online]. Available: http://www.mdpi.com/1424-8220/17/1/109

[27] I. A. Stewart, “On the combinatorial design of data centre network topologies,” *Journal of Computer and System Sciences*, vol. 89, pp. 328–348, 2017. [Online]. Available: http://creativecommons.org/licenses/by/4.0/

[28] B. Wang, Z. Qi, R. Ma, H. Guan, and A. V. Vasilakos, "A survey on data center networking for cloud computing," *Elsevier Computer Networks*, vol. 91, pp. 528–547, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138912861500300X

[29] A. Erickson, I. A. Stewart, J. A. Pascual, and J. Navaridas, "Improved routing algorithms in the dual-port datacenter networks HCN and BCN," *Future Generation Computer Systems*, vol. 75, pp. 58–71, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17300134

[30] M. U. o. C. Al-Fares, A. U. o. C. Loukissas, and A. U. o. C. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM*, pp. 63–74, 2008.

[31] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving Energy in Data Center Networks," *Proceedings of the 7th USENIX Conference on Network Systems Design and Implementation (NSDI)*, pp. 2–17, 2010. [Online]. Available: https://www.usenix.org/legacy/event/nsdi10/tech/full{_}papers/heller.pdf

[32] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, S. Sengupta, A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication - SIGCOMM '09*, vol. 39, no. 4. ACM Press, 2009, pp. 51–62. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1592568.1592576

[33] F. P. Tso, S. Jouet, and D. P. Pezaros, "Network and server resource management strategies for data centre infrastructures: A survey," *Elsevier Computer Networks*, vol. 106, pp. 209–225, 2016. [Online]. Available: www.elsevier.com/locate/comnet

[34] A. Hammadi and L. Mhamdi, "A survey on architectures and energy efficiency in Data Center Networks," *Elsevier Computer Communications*,

vol. 40, pp. 1–21, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2013.11.005

[35] C. Networking and Miercom, "The Interaction of Buffer Size and TCP Protocol Handling and its Impact on Data-Mining and Large Enterprise IT Traffic Flows," *Speeding Applications in Data Center Networks*, pp. 1–21, 2016. [Online]. Available: http://miercom.com/pdf/reports/20160210.pdf

[36] N. L. S. da Fonseca and R. Boutaba, *Cloud Services, Networking, and Management*, 1st ed. Wiley-IEEE Press, 2015.

[37] H. Zhu, X. Liao, C. de Laat, and P. Grosso, "Joint flow routing-scheduling for energy efficient software defined data center networks," *Journal of Network and Computer Applications*, vol. 63, pp. 110–124, mar 2016. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1084804516000473

[38] G. Xu, B. Dai, B. Huang, J. Yang, and S. Wen, "Bandwidth-aware energy efficient flow scheduling with SDN in data center networks," *Future Generation Computer Systems*, vol. 68, pp. 163–174, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16303028

[39] D. Gkounis, V. Kotronis, C. Liaskos, and X. Dimitropoulos, "On the Interplay of Link-Flooding Attacks and Traffic Engineering," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 2, pp. 6–11, 2016.

[40] E. Moreno, A. Beghelli, and F. Cugini, "Traffic engineering in segment routing networks," *Elsevier Computer Networks*, vol. 114, pp. 23–31, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128617300063

[41] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu, "Traffic engineering in hybrid SDN networks with multiple traffic matrices," *Elsevier Computer Networks*, vol. 126, pp. 187–199, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138912861730289X

[42] D. Carra, "Controlling the Delay of Small Flows in Datacenters," 2013. [Online]. Available: http://profs.sci.univr.it/{~}carra/downloads/dataCenter{_}openIssues{_}v04.pdf

[43] W. Guo, V. Mahendran, and S. Radhakrishnan, "Join and spilt TCP for SDN networks: Architecture, implementation, and evaluation," *Elsevier Computer Networks*, vol. 137, pp. 160–172, 2018. [Online]. Available: https://doi.org/10.1016/j.comnet.2018.03.022

[44] J. Hua, L. Zhao, S. Zhang, Y. Liu, X. Ge, and S. Zhong, "Topology-Preserving Traffic Engineering for Hierarchical Multi-Domain SDN," *Elsevier Computer Networks*, vol. 140, pp. 62–77, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138912861730289X

[45] V. Kotronis, A. Gämperli, and X. Dimitropoulos, "Routing centralization across domains via SDN: A model and emulation framework for BGP evolution," *Elsevier Computer Networks*, vol. 92, pp. 227–239, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615002480

[46] M.-C. Lee and J.-P. Sheu, "An efficient routing algorithm based on segment routing in software-defined networking," *Elsevier Computer Networks*, vol. 103, pp. 44–55, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128616300871

[47] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and Research Challenges of Hybrid Software Defined Networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 70–75, apr 2014.

[48] A. M. Abdelmoniem and B. Bensaou, "Reconciling Mice and Elephants in Data Center Networks," *2015 IEEE 4th International Conference on Cloud Networking*, 2015. [Online]. Available: https://home.cse.ust.hk/{~}amas/files/Cloudnet15.pdf

[49] B. Dai, G. Xu, B. Huang, P. Qin, and Y. Xu, "Enabling network innovation in data center networks with software defined networking: A survey," *Journal*

*of Network and Computer Applications*, vol. 94, pp. 33–49, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2017.07.004

[50] Z. Alsaeed, I. Ahmad, and I. Hussain, "Multicasting in software defined networks: A comprehensive survey," *Journal of Network and Computer Applications*, vol. 104, pp. 61–77, 2017. [Online]. Available: https://doi.org/10.1016/j.jnca.2017.12.011

[51] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient traffic splitting on commodity switches," *CoNEXT*, pp. 1–13, 2015.

[52] L. A. D. Knob, R. P. Esteves, L. Z. Granville, and L. M. R. Tarouco, "Mitigating elephant flows in SDN-based IXP networks," *IEEE ISCC*, pp. 1352–1359, jul 2017.

[53] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "CheetahFlow: Towards low latency software-defined network," *2014 IEEE International Conference on Communications, ICC 2014*, pp. 3076–3081, 2014.

[54] T. Bilen, K. Ayvaz, and B. Canberk, "QoS-based distributed flow management in Software Defined Ultra-Dense Networks," *Elsevier Ad Hoc Networks*, vol. 79, pp. 105–111, 2018. [Online]. Available: https://doi.org/10.1016/j.adhoc.2018.06.002

[55] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *ACM SIGCOMM*, ser. Hotnets-IX, New York, NY, USA, 2010, pp. 19:1–19:6.

[56] "Ryu Software-Defined Networking Framework." [Online]. Available: https://ryu.readthedocs.io/en/latest/getting{_}started.html

[57] "Open vSwitch." [Online]. Available: https://www.openvswitch.org/

[58] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed SDN controller system: A survey on design choice," *Elsevier Computer Networks*, vol. 121, pp. 100–111, 2017.

[59] J. Jung, K. Kim, and H. Kim, “On the Necessity of VM Migration: Simulation on Datacenter Network Resources,” *Wireless Personal Communications*, vol. 86, no. 4, pp. 1797–1812, feb 2016.

[60] “MiceDCER Implementation Code.” [Online]. Available: https://github.com/cfamezquita/MiceDCER

# Mice Flow Routing in Data Centers based on Software-defined Networking


Universidad
del Cauca

**ANNEXES**

Degree work

**Carlos Felipe Amézquita S.**

Advisor: PhD. Oscar Mauricio Caicedo Rendón
Co-Advisor: Msc. Carlos Felipe Estrada Solano

*Departamento de Telemática*
*Facultad de Ingeniería Electrónica y Telecomunicaciones*
*Universidad del Cauca*
*Popayán, Cauca, 2018*

# Chapter 7

# ANNEX A

Annex A presents the content of our GitHub repository.

| FILE | CONTENT DESCRIPTION |
|---|---|
| micedcer_api.py | It is the Python source code used to build the algorithm used in the evaluation of the prototype. |
| Slide_Operation_CN.pptx | It is the slide show presented for Ph.D. Chadi Assi on August 10, 2018. |

Table 7.1: Content of GitHub Repository.

Table 7.1 presents the content of our GitHub repository, it is available online in URL: https://github.com/cfamezquita/MiceDCER

# Chapter 8

# ANNEX B

The annex B presents the paper developed during the elaboration of my degree work to be published.

- **Carlos Felipe Amézquita Suárez**, Felipe Estrada Solano, Nelson L.S. da Fonseca, Oscar Mauricio Caicedo Rendón. **An Efficient Mice Flow Routing Algorithm for Data Centers based on Software-Defined Networking** IEEE ICC 2019.
    - Status: Submitted
    - Classification: h-index = 56 (Google Scholar)