PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
MOHAMED BOUDIAF UNIVERSITY OF MSILA

Faculty of Mathematics and Computer Science

Department Of Computer Science

N°: 61

DOMAIN: Mathematics and Computer Science

FIELD: Computer Science

OPTION: NETWORKS AND TECHNOLOGIES OF AND COMMUNICATION

Dissertation submitted in partial fulfilment of the requirements for the degree of MASTER

By: BENSEFFA YASSINE
BOUDIAF HICHAM

Subject

Multi controller placement in Software Defined Network

Presented publicly to the jury the: 22/07/2023

M^r	SAHRAOUI MOHAMED	University of Msila	Chairman
M^r	LAMRI SAYAD	University of Msila	Supervisor
M^r	BARKAT ABDELBASSET	University of Msila	Examiner

Academic year: 2022/2023

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
MOHAMED BOUDIAF UNIVERSITY OF MSILA

Faculty of Mathematics and Computer Science

Department Of Computer Science

N°: 61

DOMAIN: Mathematics and Computer Science

FIELD: Computer Science

OPTION: NETWORKS AND TECHNOLOGIES OF AND COMMUNICATION

Dissertation submitted in partial fulfilment of the requirements for the degree of MASTER

By: BENSEFFA YASSINE
BOUDIAF HICHAM

Subject

Multi controller placement in Software Defined Network

Presented publicly to the jury the: 22/07/2023

M^r	SAHRAOUI MOHAMED	University of Msila	Chairman
M^r	LAMRI SAYAD	University of Msila	Supervisor
M^r	BARKAT ABDELBASSET	University of Msila	Examiner

Academic year: 2022/2023

DEDICATION

This dissertation is dedicated to our parents

for their encouragement, prayers, motivations and being there.

May God bless them.

Our precious brothers and sisters who've been there

for we in our entire journey.

All of our friends and colleague.

THANKS

We want to express by these few lines of thanks our gratitude towards first of our supervisor, Mr LAMRI SAYAD for his advice and his supervision.

We also want to thank the jury too, for studying and evaluating our work. Finally, we cannot complete this project without expressing our gratitude to all the teachers in the Faculty of Mathematics and Computer, for their dedication and assistance during the years of study.

Thank you all.

TABLE OF CONTENTS

CHAPTER 3: IMPLEMENTATION AND RESULTS

FIGURE LIST

TABLE LIST

ABBREVIATIONS LIST

ACI	Application Centric Infrastructure
API	Application Programming Interface
BOA	Butterfy Optimization Algorithm
CPP	Controller Placement Problem
DFS	Depth-first search
ETSI	European Telecommunications Standards Institute
IDE	Integrated Development Environment
MANO	Management and Network Orchestration
NFV	Network Function Virtualization
PHP	Hypertext Pre-processor
PSO	Particle Swarm Optimization
SDN	Software Defined Network
TLBO	Teaching Learning based Optimization
VBO	Varna Based Optimization
WOA	Whale Optimization Algorithm
PITS	Pareto Integrated Tabu Search

GENERAL INTRODUCTION

The increasing complexity of modern networks, coupled with the need for greater flexibility and agility, has led to the development of software-defined networking (SDN) and related technologies such as network function virtualization (NFV). These approaches enable network administrators to more easily manage and control their networks, and to deploy new applications and services quickly and efficiently.

However, one of the challenges in SDN is the problem of controller placement, or (CPP). The placement of controllers in an SDN architecture can have a significant impact on network performance, as well as on the cost and complexity of the network. In order to optimize the placement of controllers, a number of algorithms and approaches have been developed.

In this project, we will focus on the problem of controller placement and propose a new algorithm for optimizing controller placement in an SDN architecture and optimizing the delay between switches and controllers. We will review related work on this topic, and present our algorithm along with a detailed analysis of its performance. Finally, we will implement our algorithm and evaluate its effectiveness.

This thesis is structured as follows:

- The first chapter presents the basic concepts of network functions virtualization (NFV) including terminologies, history, architecture, and their use cases.
- The second chapter presents the related works on the issue of controller placement, along with a discussion of the specific controller placement problem being studied, the problem formulation system model, and the proposed control plane delay minimization-based controller placement algorithm.
- The third chapter presents the environment of work and Python programming language, IDE develop VSCode. The proposed algorithm implementation and its results are also presented.

CHAPTER 1

Software Defined Network

CHAPTER 1

SOFTWARE DEFINED NETWORK

1. Introduction

Software-Defined Networking (SDN) and Network Function Virtualization (NFV) are two major technological advancements that have revolutionized the field of networking. They have the potential to transform traditional networks by making them more agile, flexible, and programmable.

Together, SDN and NFV offer significant benefits to network operators, including increased network agility, scalability, and reduced operational costs. In this chapter, we will explore the key concepts and technologies behind SDN and NFV, their applications, and their impact on the networking industry.

2. Traditional Network

The architecture of the Internet and computer networks typically involves a variety of network devices, including routers and switches.

Presently, network equipment follows a conventional structure that consists of two fundamental components: the control plan and the data plan (brain and body) respectively. The control plan bears the responsibility of decision-making and process initiation, while the data plan executes the decisions made by the control plan. When a packet arrives on a port of a switch or router, they apply the routing or switching rules that are written into its operating system.[1]

The management of networks and effective implementation of new policies can be difficult, as each device operates with a specific protocol and must be configured appropriately for communication to occur.

Adding a new rule can be challenging due to the lack of standardized procedures, as well as the potential for conflicts between devices on the network.

Additionally, the original intention of network rules was for them to be static, resulting in a static network that is not well-suited for current technological demands.
When network administrators manage and configure one of their network devices, they use a set of predefined command lines that are based on an integrated operating system.

However, managing a large number of these devices can be a major challenge that is prone to errors. This is why traditional networks often suffer from significant gaps in research and

innovation, reliability, scalability, flexibility, and management. As a result, the market can be expensive for businesses.[1]

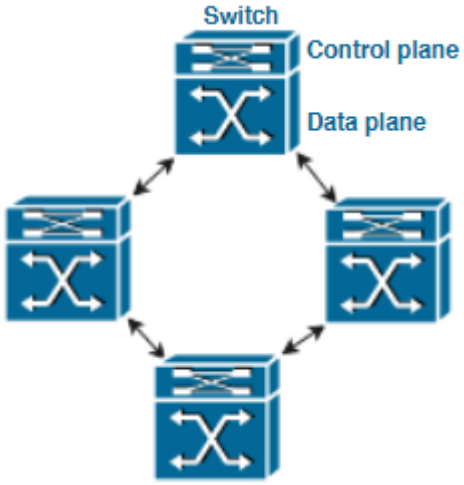

Figure 1.1: Traditional Network [2]

With the advent of the Internet, there has been a significant upsurge in networking infrastructure and the emergence of new technologies such as social networks, the cloud, and virtualization. Concomitantly, the demand for networks with higher bandwidth, more extensive accessibility, and greater dynamic management has become a crucial issue.[3]

To address the limitations and challenges of traditional networks, a structure has been proposed, known as SDN, this system separates the network control from the transmission mechanisms, enabling programming and direct control of the network.

When it comes to security, SDN offers superior protection in many ways, thanks to better visibility and the ability to create secure pathways. However, since SDN relies on a centralized controller, any security breach at that single point of entry could leave the entire network vulnerable.[7]

3. Software Defined Network (SDN)

3.1. Definition

SDN is an architecture designed to make a network more flexible and easier to manage, Centralizing management by abstracting the control plane from the data forwarding function in the discrete networking devices.[4]

3.2. Architecture

SDN presents a network architecture where the control plane is completely separated from the data plane, as in figure 1.2:

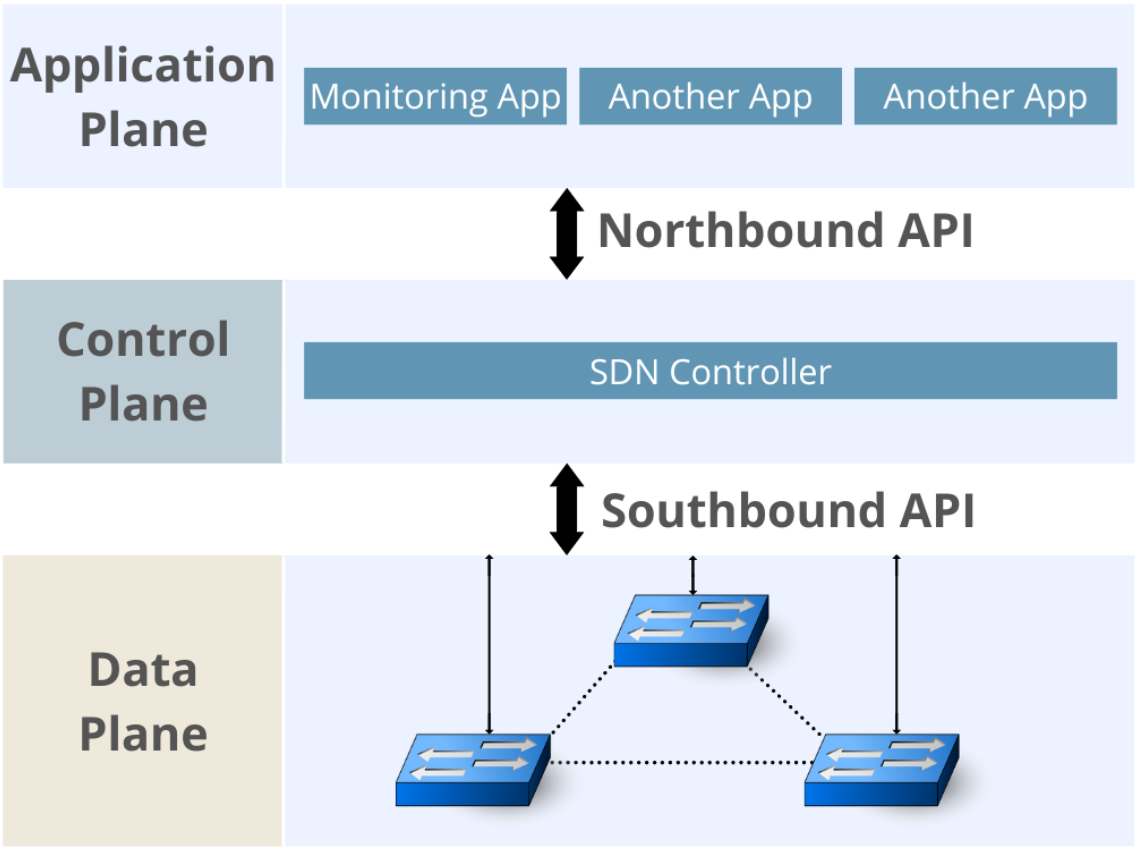

Figure 1.2: SDN Architecture

3.2.1. The components of architecture

As shown in Figure1.2, it shows the main components and interfaces of the SDN architecture.

- Application Plan: As shown in Figure 1.2, the application layer is positioned above the control layer. By utilizing the control layer, SDN applications can conveniently obtain a comprehensive overview of the network's real-time status through Northbound.[5]

The responsibility of the application plane is to establish rules and policies, with applications such as routing, monitoring, load balancing, and firewalls, etc.

- Northbound APIs that enable communication between controllers, applications, and policy engines, making an SDN appear like a single, unified network device.[4]
- Control Plan: SDN controller plane is modelled as the home of one or more SDN controllers that allows to control the data plane (the infrastructure layer) by establishing rules that it will have to follow, automation and enforcement of policy in the switches.[6]
- South band API that facilitate the communication of data between the controller and individual network components (including switches, access points, routers, and firewalls).[4]
- Data Plane: This layer of the network consists primarily of the physical/virtual equipment that is responsible send the packets as per the flow rules received from controller.[6]

3.3. SDN VS Traditional Network

Traditional network design and Software-Defined Networking (SDN) design have different approaches to designing and managing networks. Traditional network design typically uses a distributed architecture with dedicated hardware devices that perform network functions such as routing, switching, and security. Network devices are typically configured manually, which can be time-consuming and error-prone. In contrast, SDN design is based on a centralized architecture where network functions are abstracted from hardware and managed through a controller. SDN design enables automated configuration and management of the network, which makes it more scalable and flexible than traditional networks.

We will compare the design of the types of networks: traditional networks and software-defined networking (SDN).

3.3.1. Enterprise

The term "enterprise network" refers to the IT framework employed by medium and large-scale businesses to establish a connection among users, devices, and applications. The objective is to facilitate the achievement of organizational goals by ensuring the dependable and secure delivery of interconnected digital services to employees, partners, and customers.[8]

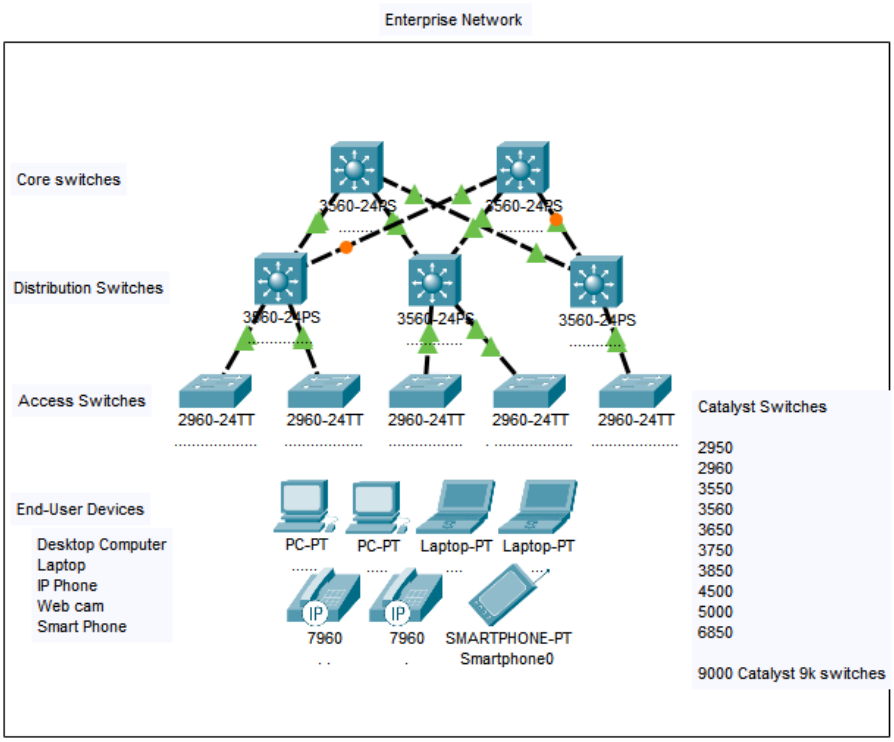

Figure 1.3: Enterprise Network Design

- Cisco SD-Access uses a controller engine to automate network segmentation, enabling granular control over network access and security.

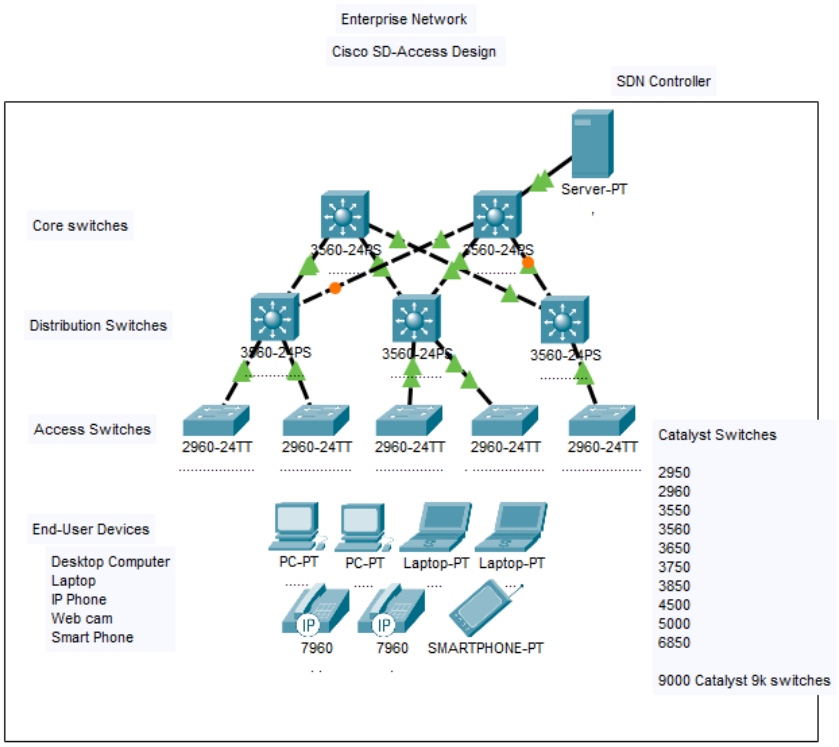

Figure 1.4: Cisco SD-Access Fabric Design

3.3.2. Datacentre Network

Essentially, a datacentre is a physical facility utilized by businesses to store their essential applications and data.[9]

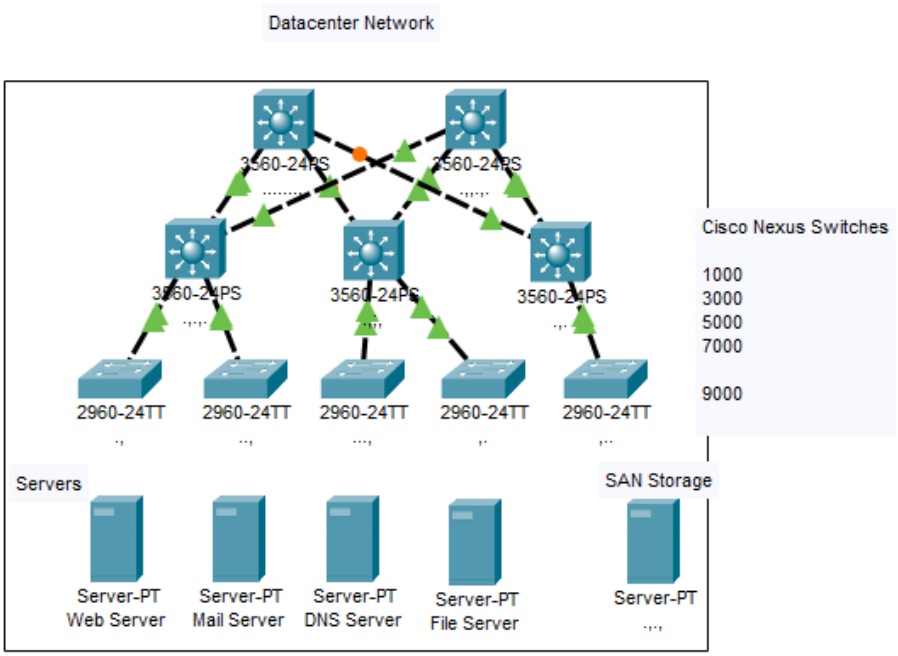

Figure 1.5: Datacentre Network Design.

- "SD Datacentre ACI Design," on the other hand, is a specific approach to datacentre network design that uses Cisco's Application Centric Infrastructure (ACI) technology.

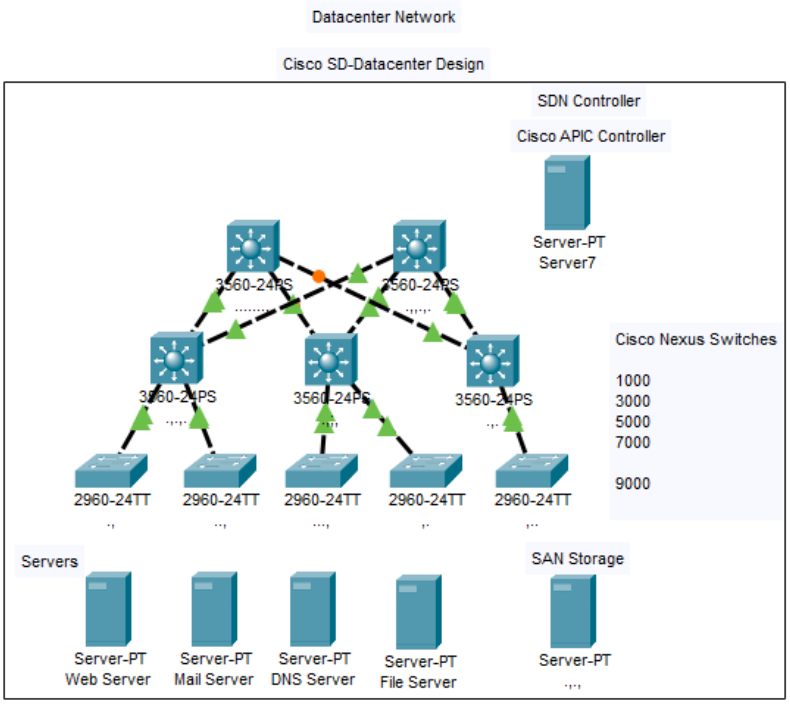

Figure 1.6: Cisco SD Datacentre ACI Design.

3.3.3. WAN (Wide Area Network)

A wide-area network (WAN) is comprised of several local-area networks (LANs) or other networks that interconnect with each other. Simply put, a WAN is an interlinked network of networks, with the Internet the world's largest WAN.[10]

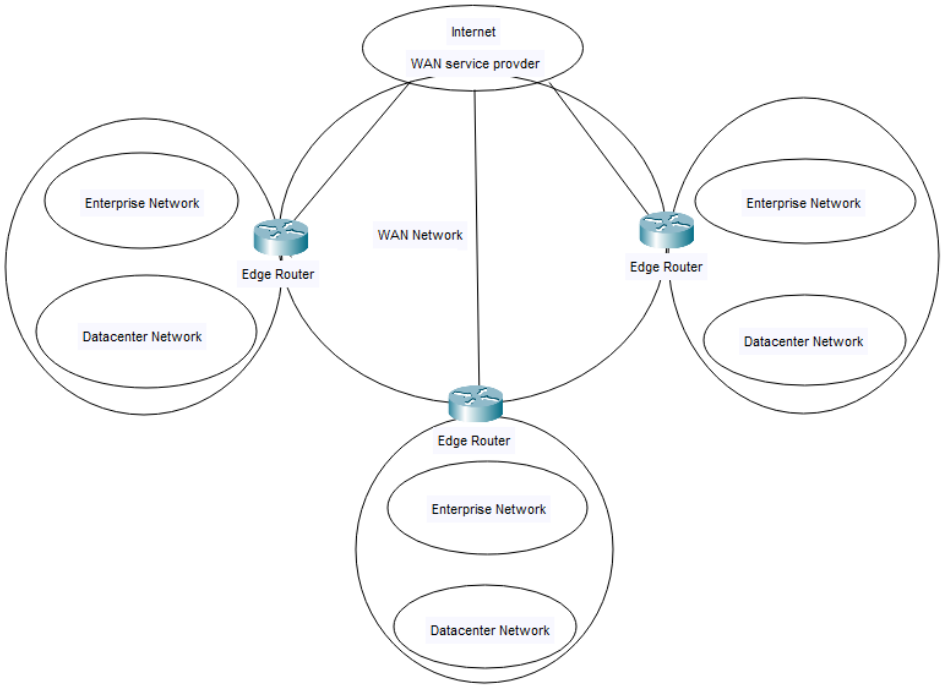

Figure 1.7: WAN Network Design.

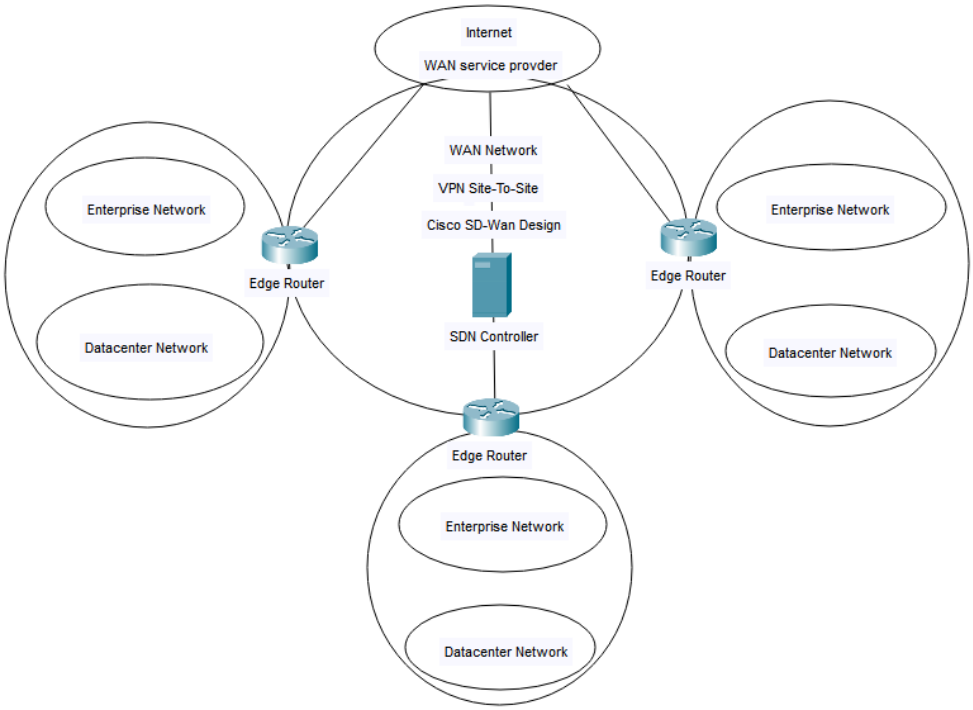

Figure 1.8: Cisco SD-Wan Design (SD-W Design).

- WAN design is often complex and time-consuming, requiring expertise in networking and a thorough understanding of the organization's needs. SD-WAN design is simpler and more automated, allowing organizations to set policies and rules that automate traffic routing and security management. Overall, while WAN design is a traditional approach to connecting geographically dispersed locations, SD-WAN design is a more modern, automated, and software-defined approach that offers additional benefits and flexibility.

3.4. Controller

3.4.1. Definition

The core element of an SDN architecture, that enables centralized management and control, automation, and policy enforcement across physical and virtual network environments.[4]

3.4.2. Types of controllers

There are several SDN controllers, such as:

- **Nox:** Was the first OpenFlow controller released in early 2008, it's open source and written on C++ and has been used by many researchers.[11]
- **Pox:** Is similar to NOX controller. POX is an sdn controller that enables rapid network development and prototyping.[12]
- **Beacon:** A Java-based high performance and multithreaded OpenFlow controller released in early 2010.[11]
- **Floodlight:** Is an open source SDN controller. It is an enterprise-class, Java-based OpenFlow controller. It works with both physical switches and virtual switches that use the OpenFlow protocate.[12]

4. Network Function Virtualization (NFV)

4.1. Definition

Network functions virtualization (NFV) replaces physical networking hardware with virtual machines, which are powered by a hypervisor to enable routing, load balancing, and other networking services. [13]

4.2. Architecture

The NFV architecture by ETSI is shown in Figure 1.9:

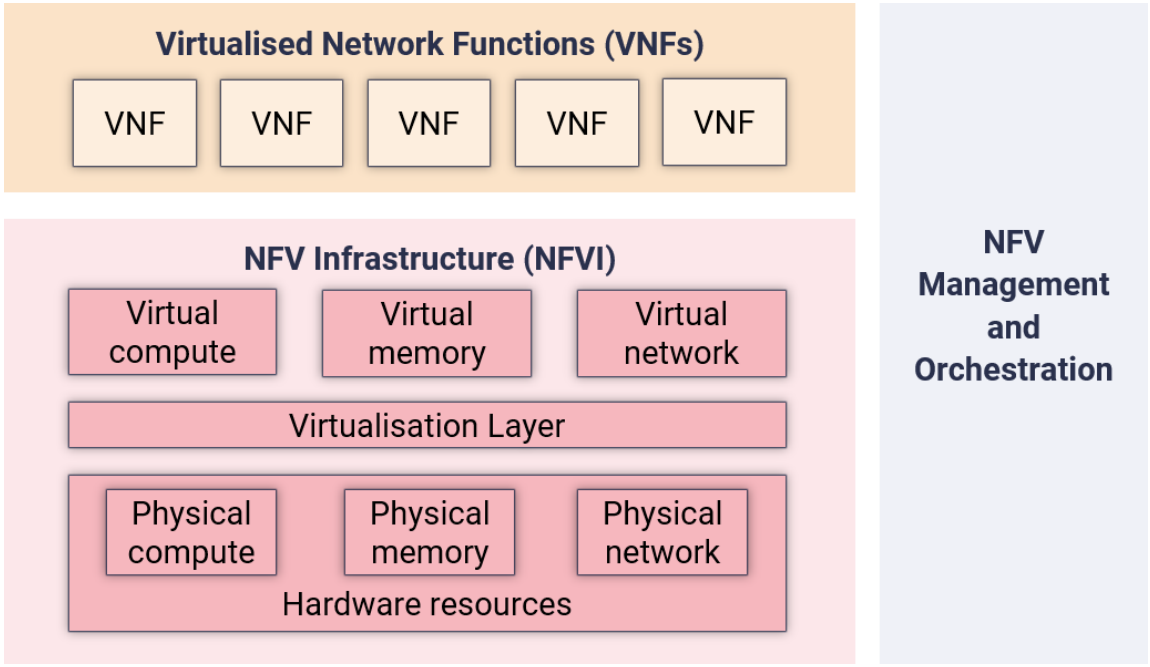

Figure 1.9: NFV architecture by ETSI.

The NFV architecture consists of:

- **Virtual Network Functions (NFV):** Network functions virtualisation are operating in one or more virtual machines that are built on top of the hardware networking infrastructure. These VNFs include routers, switches, SD-WAN, firewalls and a range of other network services that can be obtained as software from vendors like Cisco, Juniper Networks.[14]
- **Network Functions Virtualization Infrastructure (NFVI):** The NVFI is based on low cost, standardized x86-based computing hardware and software-hypervisors, and consists of a hardware interface, a virtualization layer between hardware and software and a virtual interface (storage, network,).[14]
- **NFV Management and Network Orchestration:** The European Telecommunications Standards Institute (ETSI) working group has developed a framework called NFV Management and Network Orchestration (MANO), which facilitates the end-to-end management of network services.[14]
- **Network Functions Virtualization Infrastructure (NFVI):** The NVFI is based on low cost, standardized x86-based computing hardware and software-hypervisors, and

consists of a hardware interface, a virtualization layer between hardware and software and a virtual interface (storage, network,).[14]

- **NFV Management and Network Orchestration:** The European Telecommunications Standards Institute (ETSI) working group has developed a framework called NFV Management and Network Orchestration (MANO), which facilitates the end-to-end management of network services.[14]

4.3. Benefits of network functions virtualization

The deployment and management of networks is made simpler by NFV, as it shifts network functions into software that can run on standard industry-grade hardware and be managed remotely from anywhere within the operator's network. This transition has numerous advantages for the network operator,[15] including:

- Reduced space needed for network hardware
- Reduce network power consumption
- Reduced network maintenance costs
- Reduced maintenance and hardware costs

4.4. Disadvantages of network functions virtualization

- NFV requires a realignment of processes to manage traditional and virtual infrastructures simultaneously.
- NFV is complex and difficult for many operators to deploy on a large scale due to the scope of the architecture and the number of distinct components.[16]

4.5. Comparison between SDN and NFV

NFV separates network services from hardware appliances, while SDN takes it a step further by separating control functions such as routing, policy definition and applications from the network's forwarding functions.

With SDN, a single point of control enables network programming, allowing the network to adapt quickly to varying workloads. This technology can be applied to both physical and virtual networks, but virtual networks can operate without SDN. Both NFV and SDN are based on virtualization technology.[15]

5. Conclusion

In conclusion, Software Defined Networking (SDN) represents a paradigm shift in networking that allows for the separation of the control plane from the data plane, providing greater flexibility, agility, and scalability. This is achieved through the use of a controller that manages network resources and programmable switches that provide granular control over traffic flows.

Additionally, SDN enables the use of Network Functions Virtualization (NFV), which allows for the virtualization of network functions. This eliminates the need for dedicated hardware for each network function, reducing costs and simplifying network management.

Overall, SDN and NFV offer significant benefits for modern networking.

CHAPTER 2

Controller Placement Problem

CHAPTER 2

CONTROLLER PLACEMENT PROBLEM

1. Introduction

Programmable networks (SDN) are a new paradigm that decouples the control plane and data plane, simplifying network management and improving its scalability. The controller in the control plane manages switches by providing them with rules that dictate their packet forwarding behaviour.

In a large-scale network, proper placement makes the best use of the existing network connectivity between switches. A quick response and reliable connection between the switch and its associated controller are key points for SDN networks. A single controller is difficult to control all switches in a large-scale SDN network, as the capacity of the controller is limited and the delay between the controller and switches is very high. Currently, most research aims to deploy multiple controllers in different locations to control the entire data plane.

In this type of architecture, the placement of multiple controllers becomes a critical issue. In this chapter, we present work related to the controller placement problem (CPP) in programmable networks, and then formulate the problem before proposing an algorithm to solve it.

2. Related work

Software-defined networking (SDN) algorithms are designed to optimize various aspects of network performance, such delay minimization, cost, reliability, load balancing, and energy consumption.

Figure 2.1 shows the classification of CPP from the perspective of the optimized goal:

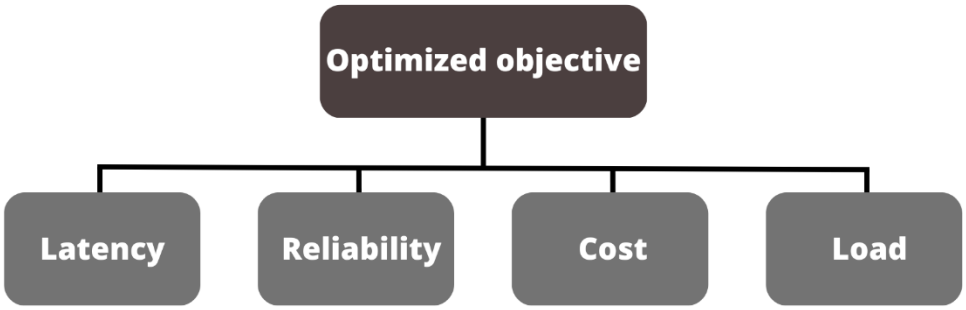

Figure 2.1: Some optimized goal.

We can see some other studies proposed to deploy controllers for different goals:

❖ **Mathematical Model for SDN Controller Placement with Minimizing Latency**

Ashutosh Kumar Singh1, Shashank Srivastava, and Shashwati Banerjea [17] propose a mathematical model for the controller placement problem in SDN. This model simultaneously determines the latency between switches and controllers and considers them as objective functions defined as follows:

- Latency between switch (S) and controller (C)

$$\mathrm{L}^{avgSC}(P) = \frac{1}{n}\sum_{s\in S} \min_{c\in P} d(s,c) \quad (1)$$

- Inter-Controller latency

$$\mathrm{L}^{avgCC}(P) = \frac{1}{P_c}\sum_{c_i,c_j\in P} \min_{c\in P} d(c_i,c_j) \quad (2)$$

- Total latency

$$\mathrm{L}^{avg-latency}(P) = \frac{1}{n}\sum_{s\in S} \min_{c\in P} d(s,c) + \frac{1}{P_c}\sum_{c_i,c_j\in P} \min_{c\in P} d(c_i,c_j) \quad (3)$$

To strike a balance between these conflicting objectives, Ashutosh Kumar Singh1, Shashank Srivastava, and Shashwati Banerjea present a heuristic algorithm called Varna Based Optimization Algorithm (VBO) to minimize the total average latency while considering the load on the switch and capacity of controllers as constraints.

VBO has been applied to several topologies and compared to other algorithms, including (PSO, TLBO, Jaya, BOA, and WOA). The simulation results indicate that in most scenarios and cases, VBO provides better results compared to other algorithm.

❖ Mathematical Model for Resilience, Scalability, and Latency

The article presents a mathematical model that addresses the placement of controllers in software-defined networking (SDN) with multiple controllers, taking into account factors such as resilience, scalability, and inter-plane latency.[18]

Many mathematical steps have been taken with the aim of an objective function:

$$Minimize\ \delta\ +\ K1\frac{\theta^{TOTAL}}{\Delta} + K2\frac{\Pi^{TOTAL}}{\Delta} \qquad (4)$$

The expression (1) is the objective function for an optimization problem that seeks to minimize three factors:

- The scalability factor, represented by δ.
- The total latency experienced by packets in the network, represented by θ^{TOTAL}.
- The total number of links used in the network, represented by Π^{TOTAL}.

According to the findings, the model can maintain scalability (δ) even in the face of failure scenarios by ensuring load balancing among controllers. Although reduced network connectivity can cause latency to increase, this issue can be mitigated by allowing for more possible controller locations.

In essence, the model generates a controller placement strategy that can sufficiently handle multiple failure scenarios.

❖ Optimal Controller Placement and Migration in SDN

The authors of the article [19] address two main issues. The first issue is finding the optimal number of controllers and their placements in the network. For this purpose, the Pareto Integrated Tabu Search (PITS) algorithm has been proposed. The second issue is performing controller migration in case of load imbalance and network link failure.

The migration phase is triggered specifically in two situations: in the event of a link failure or when there is a need to balance the load on the controllers. To initiate the migration, the algorithm examines the controller array to determine if any free controllers are currently accessible. If there are free controllers, the algorithm proceeds to install a controller in the overloaded location, establish connections between the switches and the controller, and update the flow tables accordingly. However, if no free controllers are available, the algorithm then searches for underutilized controllers within the Controller Array.

❖ **Dynamic Computation of Optimal Controller Placement in SDN**

In [20] the authors reveal a novel method for dynamic computation of optimal controller placement, including determining their locations and partitioning switches into clusters for assignment.

The proposed approach formulates the controller placement as an optimization problem, aiming to minimize controller response time, control load, intra-cluster delay, and intra-cluster throughput. A computationally efficient heuristic called DDCP, based on deep reinforcement and learning techniques, is proposed. Experimental results with the ONOS controller demonstrate significant improvements in network performance, including response time and resource utilization.

❖ **Enhanced Quantum-Behavior Particle Swarm Optimization for Optimal Multi-Controller Placement in SDN**

In [21] , the authors reveal an enhanced version of the Quantum-behavior particle swarm optimization (QPSO) algorithm to address its limitations in global search ability. By incorporating a full search history and an improved dimension update strategy, the proposed algorithm aims to improve the performance of QPSO.

The algorithm is evaluated through simulations on various multi-controller placement problems. The results demonstrate that the enhanced algorithm outperforms the traditional QPSO algorithm in terms of performance. The algorithm is specifically designed to determine the optimal number of controllers and minimize latency in the network.

2.1. Synthesis

The related works discussed have shed light on different approaches and algorithms for addressing the controller placement problem in software-defined networking (SDN). The works by Ashutosh Kumar Singh et al. [17] Mohammad Ashrafi et al. [18] and Quanyuan Zhang et al. [21] each proposed novel mathematical models and optimization algorithms to determine the optimal number of controllers, their placements, and the minimization of network latency. Additionally, the work by EL Hocine Bouzidi et al. [20] introduced a dynamic and efficient heuristic approach based on deep reinforcement and learning techniques. These diverse studies highlight the ongoing research efforts to enhance network performance and scalability through improved controller placement strategies. By considering these related works collectively, it becomes evident that significant progress has been made in addressing the controller placement problem, enabling better network optimization and management in SDN environments.

3. Discussion of optimizing Controller Placement for Minimizing Delay

An important consideration in an SDN (Software-Defined Networking) environment with multiple switches is the strategic placement of controllers, which must be studied and optimized. The issue of controller placement involves identifying the most advantageous positioning for these devices within the network as well as determining how to allocate switches to each controller. This entails determining the necessary number of controllers, their optimal locations, and establishing proper association between controllers and switches.

The delay between switches and controllers primarily refers to the delay that occurs when packets traverse the network infrastructure. This delay can be caused by factors such as physical distance, network congestion, or processing time at the switches and controllers.

The interval between switches and controls pertains mainly to the delay that arises during packet traversal across the network infrastructure. Aforementioned delay can be attributed to various factors such as physical separation, network congestion, or processing time at the switches and controllers.

The objective is to tactically position the controllers in the network to minimize delays by creating a strategy that centres on the minimization of delay. This implies situating controllers in areas that are geographically closer to the switches they manage, thereby diminishing the distance packets must traverse to establish a placement strategy for controllers that minimizes delays, various elements must be analysed, such as network topology, traffic patterns, and the desired delay constraints.

The controller placement problem as a control plane delay minimization problem subject to switches and controllers association, In the following section, we will discuss the heuristic method that we employed to address this delay optimization problem.

4. Problem formulation and system model

The network topology of a data plane G (V, E, D) consists of a set of switches " V " where Vi denotes the ith switch, $1 \leq i \leq N$, which N is the number of switches, and a set of bidirectional links "E". and "D" is the delay.

In the article "Control plane delay minimization-based capacitated controller placement algorithm for SDN", we conducted research on the delay between switch Vi and Vj, which is a combination of the transmission delay and the propagation delay. It is defined as:

$$D = D^s + \alpha D^c \qquad (5)$$

Where "D" is the control plane delay of the network and "D^s" denotes the delay between controllers and switches, "α" is Weighting factor for the importance of inter-controller delay and "D^c" is the inter-controller delay.

The delay between controllers and switches, "D^s" in (1) can be calculated as:

$$D^s = \sum_{j=1}^{N} \sum_{i=1}^{N} X_{ij} D_{ij}^s \qquad (6)$$

Where "X_{ij}" denotes the binary association variable between V_j and C_i

The inter-controller delay denoted by "D^c" in (1) and can be calculated as:

$$D^c = \sum_{j=1}^{N} \sum_{i=1}^{N} y_i D_{ij}^c \qquad (7)$$

Where yi denotes the binary controller placement variable of Ci. And we assume that the maximum number of controllers that can be deployed is M.

However, for simplicity and ease of analysis, we consider the combined delay as a single entity.

In the Binary_identifier Matrix, the link between switch Vi and Vj is defined as follows:

$$[\text{Binary\_identifier}]ij = \begin{cases} 1, & if\ there\ is\ a\ link\ between\ Vi\ and\ Vj; \\ 0, & Otherwise. \end{cases} \qquad (8)$$

The delay between switch Vi and Vj in ArrayD Matrix is a natural number ϵN, it is defined as follows:

$$[\text{ArrayD}]ij = \begin{cases} N, & if\ there\ is\ a\ link\ between\ Vi\ and\ Vj; \\ 0, & Otherwise. \end{cases} \qquad (9)$$

We assume that the network topology is known.

The objective of the problem is to determine the optimal locations for controllers to effectively manage the network and optimizing the delay between switches and controllers.

5. CPDM-CCP: Control Plane Delay Minimization-based Controller Placement Algorithm

The heuristic algorithm that combines Djikstra and K-means algorithms, aimed at minimizing the delay while considering various constraints, improve control plane performance, and improve network management in SDN environments.

We present in Figure 2.2 process flow diagram for our algorithm proposed

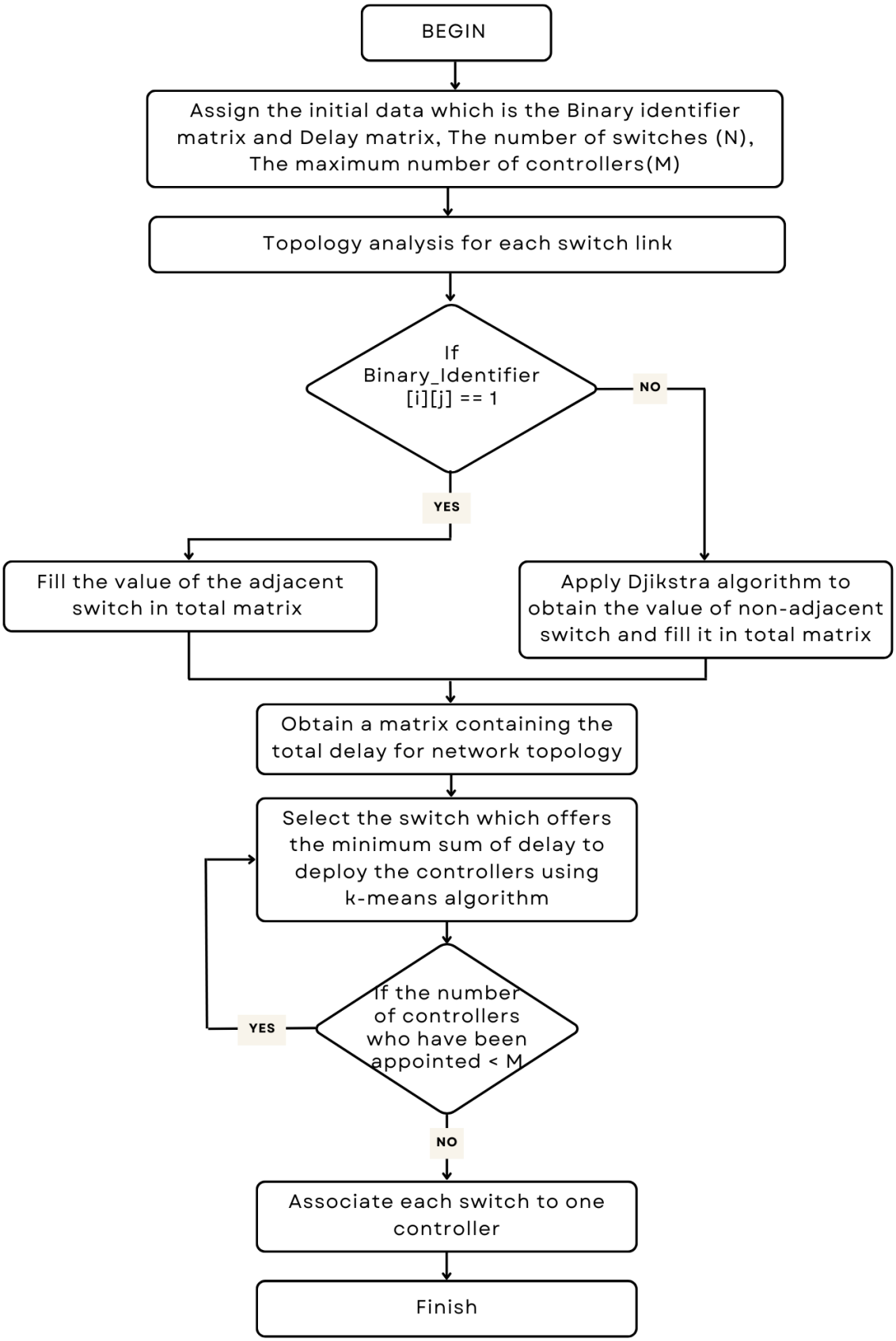

Figure 2.2: Flow Diagram.

We have simplified the proposed method in steps as follows:

A. Determine the delay between any two adjacent switches in the network.

B. For any two nonadjacent switches, determine the minimum delay by applying the Djikstra algorithm.

C. Calculate the sum of minimum delay, set as the weight between two switches;

D. Determine the first controller, denoted by C1 based on the K-means algorithm. More specifically, for each switch, calculate the sum of the minimum delay with other switches, select the switch which offers the minimum sum of delay to deploy the first controller;

E. Calculate the delay between each switch and C1, select the switch which offers the minimum delay to deploy the second controller, denoted by C.

F. Associate each switch to one controller.

G. Select the switch which offers the minimum delay with C1and C2 to deploy the third controller.

H. The process repeats until the number of controllers is reached.

I. For all the switches in the network, associate one controller with offering the minimum delay under the load constraint of the controller.

5.1. Optimal route selection strategy using the Djikstra algorithm

After considering that the graph is known, and analyzing the structure, we can find two non-adjacent switches Vi and Vj, the delay between which may include multiple hops through intermediate switches and links. This means that the delay calculation becomes more complex and cannot be easily specified or calculated directly. So, we've come to apply Djikstra's algorithm, which is widely used to solve the shortest path problem in weighted directed graph to calculate the optimal path between Vi and Vj, If there is a link between the switches ([Binary_identifier]ij = 1), it is left unchanged, and if there is no link ([Binary_identifier]ij), a new value is set to obtain the minimum delay.

And we get a new array containing the delay between the switches that were not linked before, called D_Prime.

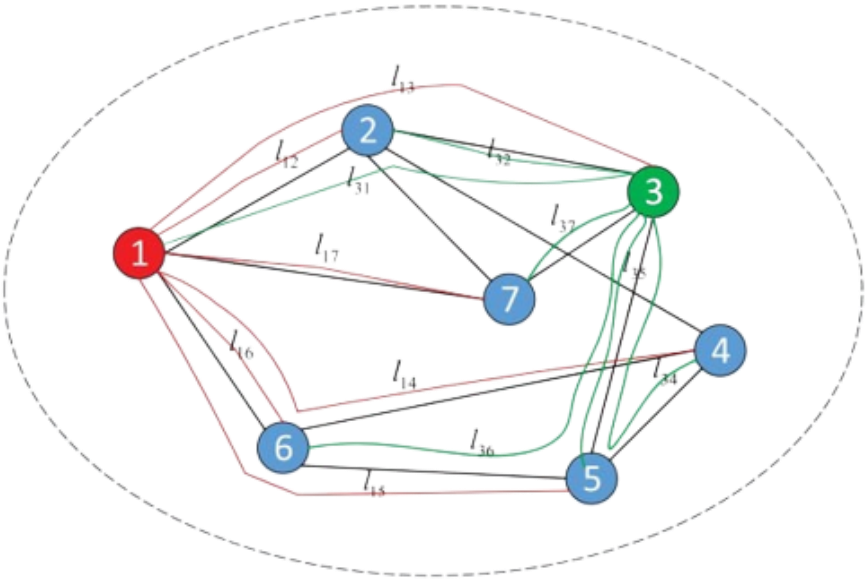

Figure 2.3: Optimal Position selection of controller. [22]

The accompanying pseudocode (see Figure 2.4) illustrates Djikstra algorithm used in our system model to address the problem of no link between switches:

Algorithm 1 Djikstra-based optimal route selection strategy

```
Require: D, αij, N
Ensure: D'
For i=1 to N do:
    For j=1 to N do:
        If αij = 1
            D'[i][j] = D[i][j]
        Else
            Applying Djikstra to find the shortest delay for no adjacent switches
        End If
    End for
End for
Return D'
End Procedure
```

Figure 2.4: Djikstra-based optimal route selection strategy process.

5.2. Controller placement strategy using the K-means algorithm

Given that one controller in the network can be associated with multiple switches, it is possible to establish clusters where the controller acts as the cluster head and the associated switches act as the cluster members. As a result, the network topology of the Software-Defined Networking (SDN) can be seen as a cluster-based structure.

To determine the optimal placement of controllers and formulate an effective strategy for the association between controllers and switches, clustering methods can be employed. In our approach, we have utilized the K-means algorithm, a highly efficient clustering method, to determine the appropriate number of controllers and their respective locations. This has enabled us to devise an optimal controller placement strategy based on the K-means algorithm.

By applying the K-means algorithm, and after construct a total matrix that represents the delay between adjacent and non-adjacent switches in the network, the sum of the minimum delays is calculated from the total matrix, find the index and value of the minimum delay from the sum of minimum delays and select the switch with the minimum delay as the first controller and update the respective and put it in a vector called Yi, we have successfully identified the controllers within the network, with each switch is associated with one controller as the head and its associated switches as members. This approach facilitates efficient management and control of the network, as the controllers can effectively oversee and communicate with their associated switches and to reach the optimal delay.

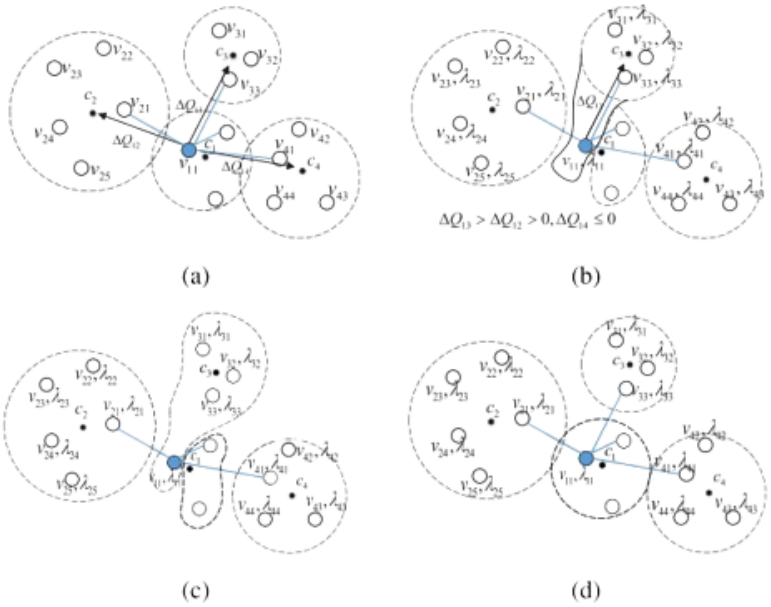

Figure 2.5: Process of finding target community for a node. [22]

The accompanying pseudocode (see Figure 2.6) illustrates K-means algorithm used in our system model to address the controller location mapping problem, and the total number of controllers deployed should be less that or equal to a given value M. This constraint can be expressed as:

$$C = \sum_{i=1}^{N} Y_i \leq M \qquad (10)$$

Algorithm 2 K-means strategy

```
1: Require: M, m, D, D_prime, index_minimum_delay, Value_minimum_delay
2: Ensure: Yi
3: Determine the first controller:
4:     calculate the sum of minimum delays:
5:     Sum_Delay = sum_delay(total_matrix=D_prime+D)
6:     Set "M" the number of controllers and initializes the variable m to 1
7:     While (m<M)
8:         Find the index and value of the minimum delay
9:         Yi[index_minimum_delay] = 1
10:    return Yi
11: End Procedure
```

Figure 2.6: K-means pseudocode strategy.

5.3. Binary association

The binary association refers to the association between a specific controller (indexed as "i") and a specific switch (indexed as "j") in a software-defined networking (SDN) context. The primary objective is to determine which controller should be associated with each switch based on the minimum delay, we assume that each switch can only associate with a unique controller, can be expressed:

$$C = \sum_{i=1}^{N} X_{ij} = 1 \qquad (11)$$

The accompanying pseudocode (see Figure 2.7) illustrates association algorithm used in our system model to address the problem of association between controller and switches.

Algorithm 3 Switch-Controller Association Algorithm

```
1: binary_association[][]=0;
2: Xij[]=0
3:   For i to N do
4:     For j to N do
5:       When Yi[j]==1
6:           Xij[]=D_total
7:  binary_association=min(Xij[i])
8: Return binary_association
9: End Procedure
```

Figure 2.7: Switch-controller association pseudocode

6. Conclusion

In this chapter, we presented an overview of related works and literature on the controller placement problem in SDN. Next, we provided a detailed description of a proposed solution, which uses a Djikstra and K-Means algorithm to solve the shortest path and determine the appropriate number of controllers and their respective locations.

CHAPTER 3

IMPLEMENTATION AND RESULTS

CHAPTER 3

IMPLEMENTATION AND RESULTS

1. Introduction

In the previous chapter, we conducted a review of existing works that address the problem of controller placement in SDN and presented the CPDM -CCP algorithm in detail. In this chapter, we will explain the implementation of the proposed algorithm. We have chosen an object-oriented programming language that is fast and allows manipulation of high-level graphical interfaces has a clear and easy-to-understand syntax, and is open source. For this purpose, we have chosen the Python language.

Finally, we will present the test results to evaluate the algorithm's performance.

2. Development environment

Visual Studio Code is a robust source code editor that can be run on your desktop, and is compatible with Windows, macOS, and Linux operating systems. This editor comes equipped with integrated support for JavaScript, TypeScript, and Node.js, and boasts a diverse collection of extensions that support various languages and runtimes such as C++, C#, Java, Python, PHP, Go and .NET. [23]

We have opted for this IDE due to its user-friendly interface and our familiarity with its functionalities, which have made it an effortless choice for our development needs.

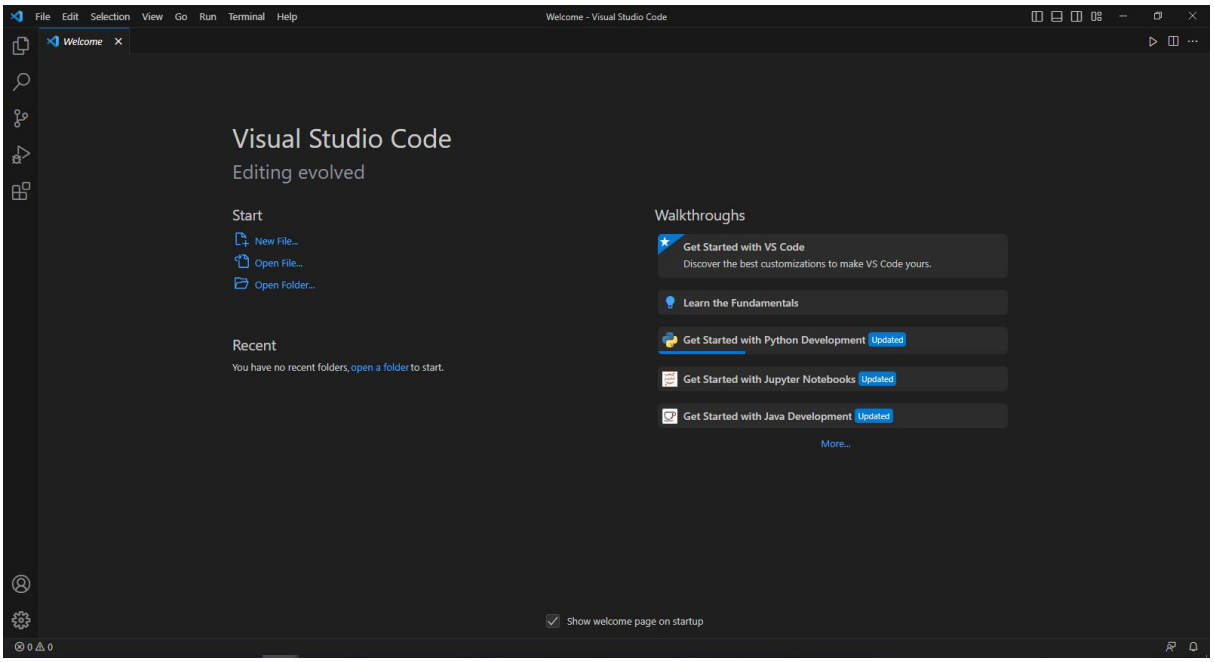

Figure 3.1: Visual Studio Code.

3. Implementation

In this work, we aim to solve the SDN controller placement problem, building upon the CPDM-CCP approach. We have designed our program in python. In order to represent all the elements and apply the algorithms properly, we had numerous elements, variables, and functions.

Therefore, we will perform experiment with the algorithm presented in the past chapter, and we'll introduce the implementation from the beginning, the constant variables, the process and the results.

Constant variables	**Explanation**
Alpha(i,j)	A binary value, randomly generated from the range {0, 1}, denotes the presence or absence of a switch.
Delay(i,j)	The randomly generated value from the range {1,9}, indicates the delay between any two adjacent switches.
Yi	Binary controller position variable for each specified controller, so that the value of the variable is between {0,1}.

Table 3.1: Constant Variables.

3.1. Generate matrix

The binary_identifier(N) function takes the number of switches (N) as an input and returns a binary identifier matrix (alpha). The matrix alpha is initialized as a square matrix of size N x N. The diagonal elements of the matrix are all equal to 1, indicating a self-connection within the switch. After that, another loop over each pair of distinct row and column indices (i and j) and assigns a random binary value (0 or 1) to alpha[i] [j]. The same value is also assigned to alpha[j][i] to ensure symmetry in the matrix. Finally, the alpha matrix is returned.

```
# Ask the user the number of switchs and controllers
while True:
    try:
        N = int(input("Please enter the number of switches (N) : "))
        M = int(input("Please enter the number maximal of controllers (M) : "))
        break
    except ValueError:
        print("Veuillez entrer un nombre valide.")
```

Figure 3.2: User Input for Number of Switches and Controllers.

The calculate_delay_matrix(alpha) function takes the binary identifier matrix (alpha) as input and returns a delay matrix (delay). The function initializes the delay matrix as a square matrix of size N x N. Then, the diagonal elements of the matrix are all equal to 1, indicating that there is no delay between the switch and itself. If alpha[i][j] is 1, indicating that switches i and j are in contact, a random integer delay value between 1 and 9 is assigned to delay[i][j]. Then the delay matrix is returned.

```
# Fill a binary_identifier matrix with N number of switch
def binary_identifier(N):
  alpha = np.zeros((N, N), dtype=int)
  for i in range(N):
    for j in range( N):
        if i==j:
            alpha[i][j]=1
  for i in range(N):
    for j in range(i+1, N):
        alpha[i][j] = random.randint(0, 1)
        alpha[j][i] = alpha[i][j]
  return alpha
# Fill a matrix of (ArrayD) Delay value
def calculate_delay_matrix(alpha):

    N = len(alpha)
    delay = np.zeros((N, N),dtype=int)

    for i in range(N):
        for j in range(N):
            if i==j:
                delay[i][j]=0
            elif alpha[i][j] == 1:
                # Calculate the delay between switches i and j
                delay[i][j] =random.randint(1, 9)# Replace with your own delay calculation logic
                delay[j][i]=delay[i][j]
    return delay
```

Figure 3.3: Binary_identifier() and Calculate_delay_matrix() functions.

3.2. Unveiling the Main Functions

Once the matrix is generated, we proceed to the phase of determining the most efficient route between the switches in the network. the function is as follows:

Optimal_strategy ()

The function optimal_strategy operates on finding the optimal route between switches in the network. It creates a graph based on the alpha matrix and assigns delay values to the edges. The delay remains between each adjacent switch as is, and if there is a pair of non-adjacent switches, the Djikstra algorithm is applied to find the shortest path, and store the minimum delay in the D_prime matrix.

```
def optimal_strategy(N, Binary_Identifier, D):
    # Initialize a graph object
    graph = Graph(N)
    # Initialize a matrix to store the minimum delays
    D_prime = [[0 for j in range(N)] for i in range(N)]
    # Construct the graph
    for i in range(N):
        for j in range(N):
            if Binary_Identifier[i][j] == 1:
                graph.graph[i][j] = D[i][j]
    # Apply Djikstra's algorithm to each pair of non-adjacent switches
    for i in range(N):
        for j in range(N):
            if Binary_Identifier[i][j] != 1:
                # Reset the graph object
                graph = Graph(N)
                # Construct the graph
                for k in range(N):
                    for l in range(N):
                        if Binary_Identifier[k][l] == 1:
                            graph.graph[k][l] = D[k][l]
                # Apply Djikstra's algorithm to find the shortest path between i and j
                graph.djikstra(i)
                D_prime[i][j] = graph.dist[j]
    return D_prime
```

Figure 3.4: optimal_strategy() function.

Djikstra()

The function Djikstra implements Djikstra's algorithm to calculate the minimum delays between any two no-adjacent switches in a network.

Figure 3.5 shows the code of the function

```
def djikstra(self, src):
    self.dist = [sys.maxsize] * self.V
    self.dist[src] = 0
    sptSet = [False] * self.V

    for cout in range(self.V):
        u = self.minDistance(self.dist, sptSet)
        sptSet[u] = True
        for v in range(self.V):
            if self.graph[u][v] > 0 and sptSet[v] == False and self.dist[v] > self.dist[u] + self.graph[u][v]:
                self.dist[v] = self.dist[u] + self.graph[u][v]
```

Figure 3.5: Djikstra() function.

K-means Algorithm

The algorithm aims to iteratively select controllers based on the minimum delay value, (Figure 3.6) shows the code of the K-means algorithm.

```
# k-means controller placement
#Determine the first controller:
Sum_Delay=sum_delay(total_matrix_delay(D,D_prime))# the sum of minimum delay

index_minimum_delay,min_delay=min_index_delay(Sum_Delay)# Index and the sum of the minimum delay with other switches
delay_moyen=[]
delay_moyen.append(min_delay)
Yi = [0] * N #  the binary controller placement variable FOR each controller selected
Yi[index_minimum_delay] = 1 # selact the first controller

# set the reste of controller
m=1
# the vecter of witch select controller with witch value selected updates to infin to ll be not select in next step
D_controller= update_value_to_infinity(Sum_Delay, index_minimum_delay)
# repeat the same step to determin all controller corespendate
while m<M:
            # determin the index and value of minimun delay to selected controller
            index_minimum_delay,min_delay=min_index_delay(D_controller)
            # fill the vector of Yi with the position of index of delay selected
            Yi[index_minimum_delay]=1
            # witch controller selected call a function to associate each new value selected
            binary_association,val=Association(total_matrix,Yi)
            # Fill each value of delay for each slect of controller
            delay_moyen.append(np.sum(val))
            # update the matrix with the value selected to infini to don't selected for second step
            D_controller= update_value_to_infinity(Sum_Delay, index_minimum_delay)
            # increament the m to  next  item
            m += 1
```

Figure 3.6: K-means algorithm**.**

Association ()

The Association() function calculates the binary association between controllers and switches based on the minimum delay values. It creates a binary matrix representing the associations and computes the values associated with each controller, which is the sum of delay values for switches associated with that controller. Figure 3.7 shows the code of the association function.

```
def Association(total_matrix,Yi):
  #the binary association identifier between the ith controller and the jth switch
  #initialization of Xij to zero
  binary_association = [[0] * N for _ in range(N)]
  Controller_val=[]
  # assocaite the to the each switch with controller offer the min value
  for i in range(N):
        Xij = [0] * N # inisialise vector with zero
        for j in range(N):
            if Yi[j] == 1:# test if the current case is controller
                Xij[j]= total_matrix[i][j] # insert each value of controller
            else:
                Xij[j]=math.inf #insert each value is not controller to infini to not selected
        # select the index of minimum value of controller offre
        inx,val=min_index_delay(Xij)
        Controller_val.append(np.sum(val))
        # charge the value selected in matrix of
        for j in range(N):
             if j==inx and i != j:# Validate Controller Association
                binary_association[i][inx] = 1 # associate the switch i to controller
  return binary_association,Controller_val
```

Figure 3.7: Association() function.

3.3. Visual Displays

```
# Create a NetworkX graph based on the Binary_Identifier
G_1 = nx.Graph() #initializes an empty graph object called G_1
for i in range(len(Binary_Identifier)):
    for j in range(i+1, len(Binary_Identifier)):
        if Binary_Identifier[i][j] > 0:
            G_1.add_edge(i+1, j+1)
# Draw the graph without weights and before add controller
pos = nx.spring_layout(G_1)
color_map = ['lightblue'] * len(G_1.nodes)
nx.draw(G_1, pos, with_labels=True, node_color=color_map, node_size=800)
plt.show()
```

Figure 3.8: Code representing the graph.

This code snippet helps visualize the graph represented by the Binary_Identifier matrix by drawing the graph with nodes displayed as light blue circles and the resulting graph is displayed.

```
#Curve graph
 #Simple data
x = list(range(1, M+1))
y = delay_moyen
# Plotting the graph
plt.plot(x, y)
# Adding labels and title
plt.xlabel('Nombre of controllers ')
plt.ylabel('Total control plane delay (ms) ')
plt.title("Delay minimization for controller problem placement")
plt.xticks(range(1, len(x) + 1)) # Set the x-axis ticks by the number of controllers

# Display the graph
plt.show()
```

Figure 3.9: Code representing the graph

The code is plotting a line graph representing the relationship between the number of controllers and the total control plane delay. The x-axis represents the number of controllers, while the y-axis represents the total control plane delay in milliseconds.

After creating the "Binary Identifier" in document file. we generate this matrix (containing 30 switches) after we extracted as graphical view. The (figure 3.10) show this graph:

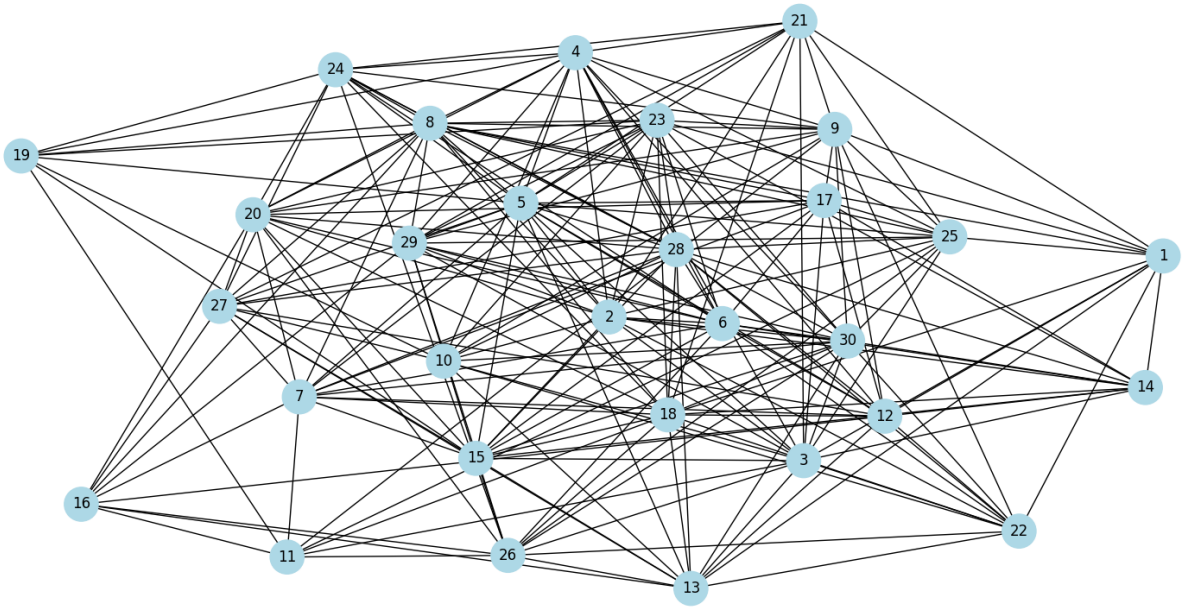

Figure 3.10: Binary Identifier matrix network.

4. Results

After we are implementing the algorithm presented in the past chapter, we got the following results:

- The vector represents the sum of delay when connecting with all other.
- The vector representing the placement of the controllers.
- The vector representing the delay after adding each controller.

(Figure 3.11) shows us the result obtained from applying CPDM on a network containing 30 switches, with a value of number of controllers (M) equal to 7.

```
-----------------------------------------------------
The vector that represents the sum is: [126, 124, 134, 126, 131, 148, 121, 122, 132, 146, 152, 120
, 135, 150, 144, 129, 147, 108, 132, 139, 124, 120, 130, 159, 147, 142, 122, 156, 132, 124]
-----------------------------------------------------
controller poition: [0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
 1, 0, 0, 0]
-----------------------------------------------------
Delay value after adding each controller: [108, 83, 60, 53, 50, 48, 40]
```

Figure 3.11: Result obtained from applying the heuristic algorithm.

Each controller with its associated switches is shown in Table 3.2

Controller 01	S4, S25, S26, S29
Controller 02	S5, S15, S17, S20, S24, S30
Controller 03	S19, S23
Controller 04	S6, S9, S16
Controller 05	S28
Controller 06	S1, S3, S10, S11, S13, S14
Controller 07	S21

Table 3.2: The association of the switches with each controller.

(Figure 3.12) shows us the controller placement after applying the proposed algorithm.

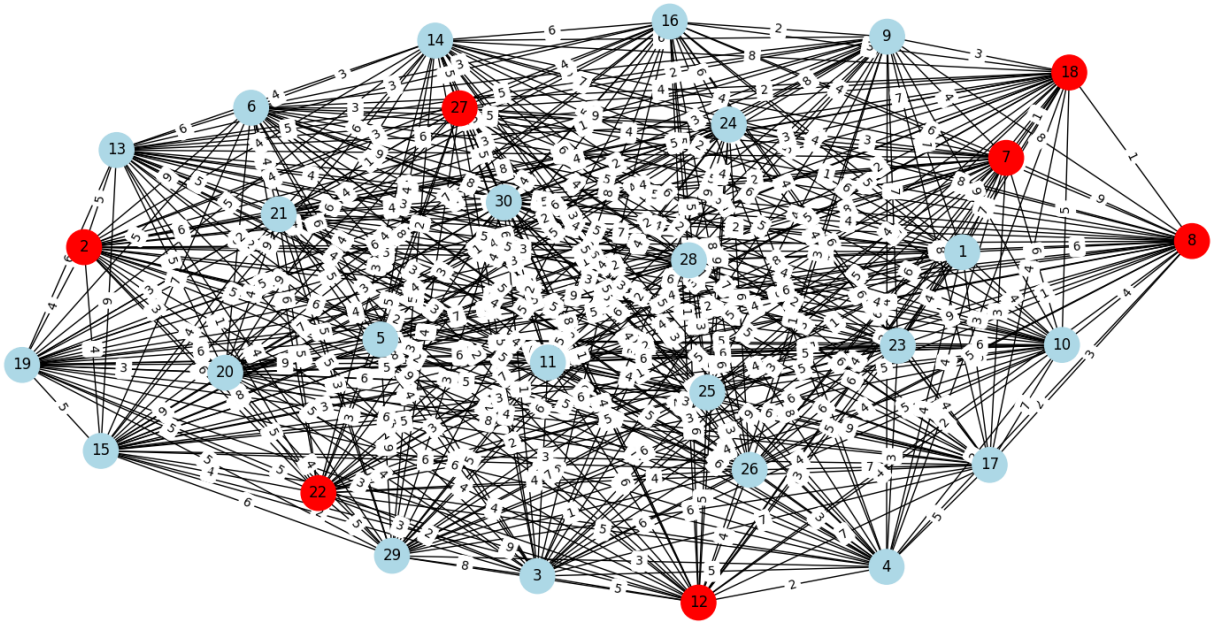

Figure 3.12: Binary Identifier matrix network after applying the heuristic algorithm.

(Figure 3.13) shows us the cluster after setting the controllers.

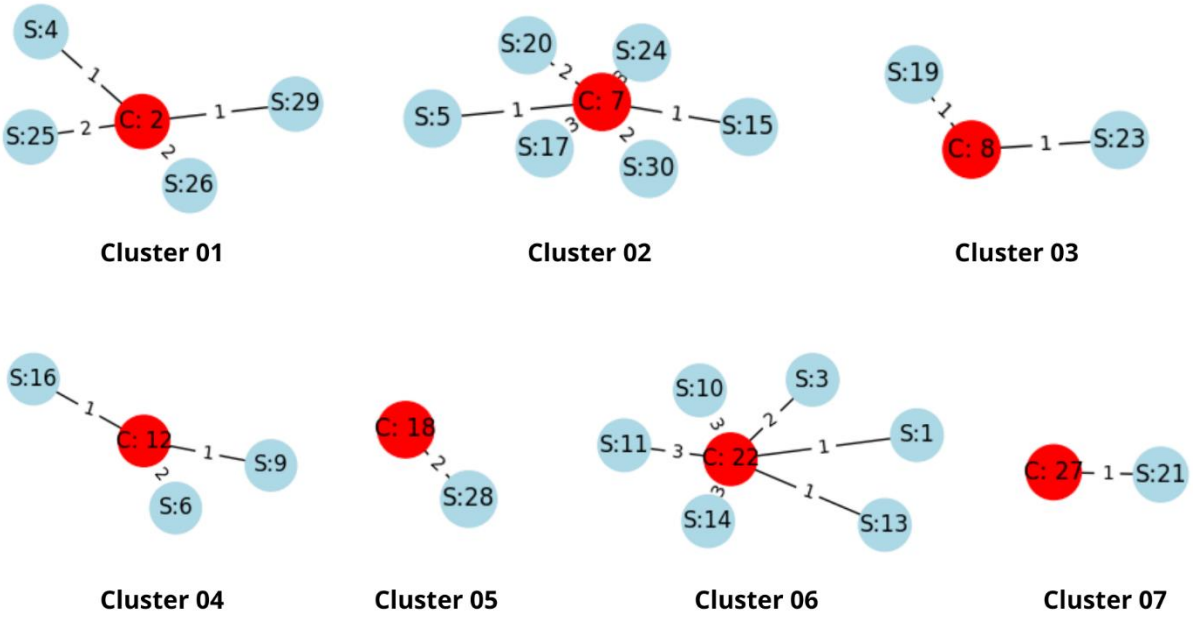

Figure 3.13: Clusters of association controllers.

We obtained the following outcomes for delay optimization:

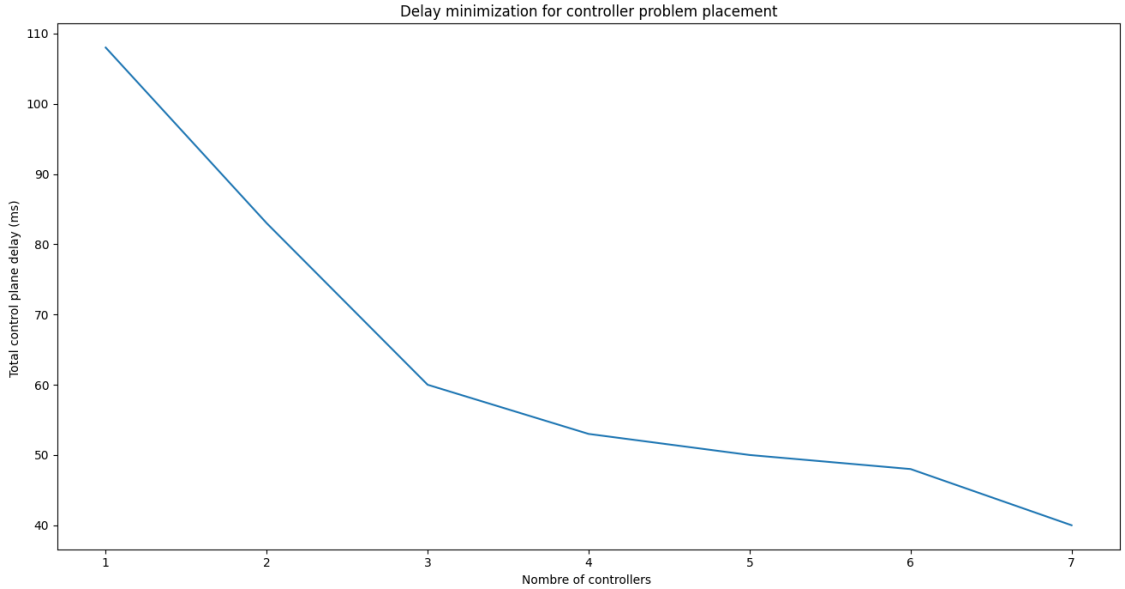

Figure 3.14: Graphical curve for delay optimization with N=30.

The delay caused by adding each controller is depicted on the graphical curve with 30 nodes in Fig. 3.14, which shows the initial sharp drop in delay when the first 3 controllers are added. It then shows a slower decline until controller number 7 is added, making additional controllers unnecessary.

After upgrading the binary identifier matrix network to a network comprising 100 switches and setting the number of controllers (M) to 21, we proceeded to execute the algorithm outlined in the previous chapter. As a result

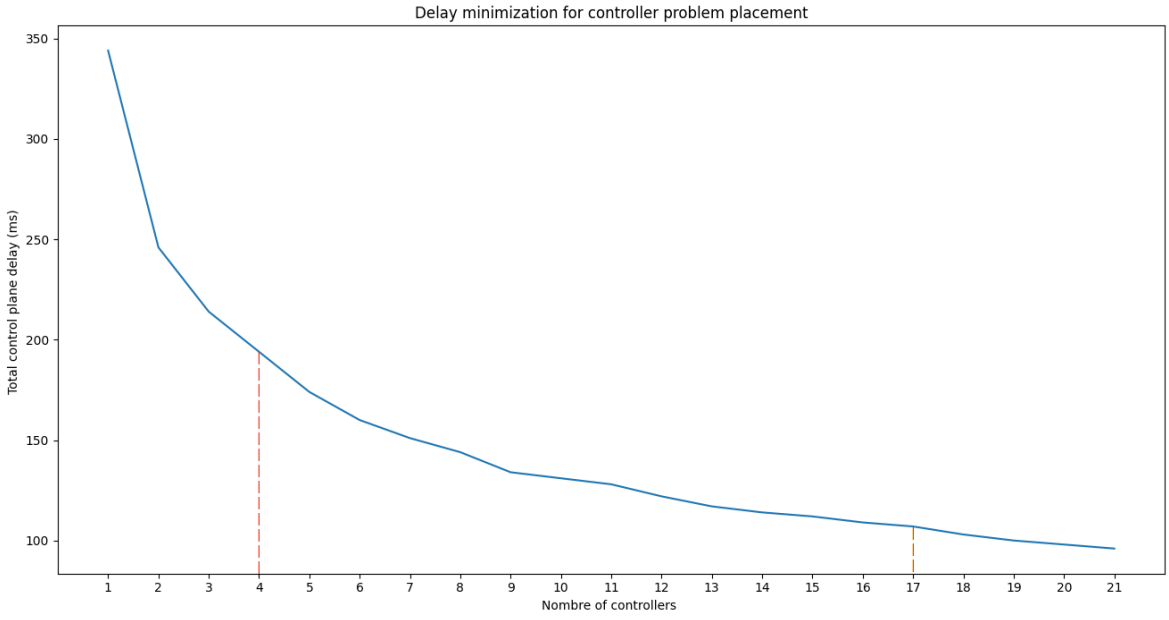

Figure 3.15: Graphical curve for delay optimization with N=100.

The delay incurred with the addition of each controller is depicted on the graphical curve with 100 nodes in the Figure 3.15, illustrating the initial steep decrease in delay when adding the first 4 controllers. It then shows a slower decrease until the 17th controller is added, after which the delay stabilizes, making additional ones unnecessary.

After upgrading the binary identifier matrix network to a network comprising 200 switches and setting the number of controllers (M) to 43, we proceeded to execute the algorithm outlined in the previous chapter. As a result:

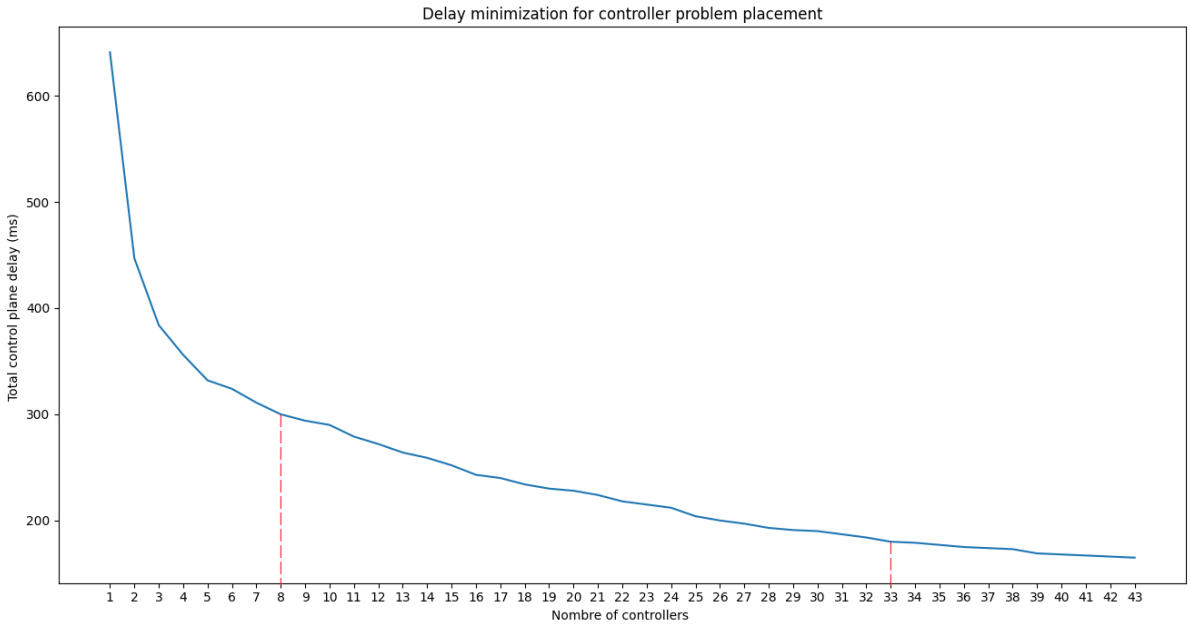

Figure 3.16: Graphical curve for delay optimization with N=200.

The delay incurred with the addition of each controller is depicted on the graphical curve with 200 nodes in the Figure 3.16, illustrating the initial steep decrease in delay when adding the first 8 controllers. It then shows a slower decrease until the 33th controller is added, after which the delay stabilizes, making additional ones unnecessary.

5. Conclusion

In conclusion, Chapter 3 has focused on the implementation and results of the proposed algorithm, which combines Djikstra and k-means methodologies. The primary objective was to determine the positions of the controllers on the network and minimize the delay effectively.

The findings presented in this chapter demonstrate the successful application of the proposed algorithm. Through the utilization of the Djikstra algorithm, which calculate the optimal path between switches, and the k-means algorithm to determine the placement of controllers within the network, a robust solution was achieved. The algorithm effectively

determined the optimal positions for the controllers in the network, leading to the minimization of delay.

The combination of Djikstra and k-means algorithms showcases the potential for integrating different methodologies to tackle complex problems effectively. The successful implementation of this algorithm opens up new possibilities for addressing similar challenges in other network systems.

GENERAL CONCLUSION

Software-Defined Networking (SDN) has emerged as a promising paradigm for managing and controlling network infrastructure. By separating the control plane from the data plane, SDN enables greater flexibility, scalability, and programmability in network management. One crucial aspect of SDN is the placement of multiple controllers, which plays a vital role in ensuring efficient network operation. Our research focused on optimizing the delay between switches by addressing the multi-controller placement problem in SDN and proposed an algorithm to optimize this placement.

The Controller Placement Problem (CPP) is one of the main challenges in programmable networks for enhancing performance. To address this problem, we have chosen the Controller Placement Algorithm based on control plane delay minimization. The algorithm incorporates optimal route selection using Djikstra's algorithm and controller placement strategy using the K-means algorithm.

In this work, we implemented the algorithm using the Python programming language and conducted experiments on different topologies. After inputting the values, we obtained the optimal number of controllers and their respective placements that minimize the delay between switches and controllers.

Overall, future studies in this area can focus on finding the optimal number of the controllers and capacitance mode for them.
Furthermore, examining the integration of SDN and NFV could result in more comprehensive and efficient network solutions..

BIBLIOGRAPHY

[1] M. Altufaili, "Intelligent Network Bandwidth Allocation using SDN", International Journal of Engineering Research & Technology, Vol. 4, 2015, p.493-496.

[2] A. Malik, A.Benjamin and K.Chih-heng, "THRIFTY: Towards High Reduction In Flow Table memory", 2018 Imperial College Computing Student Workshop (ICCSW 2018), Dagstuhl - Germany, 2019.

[3] "The Road to SDN- ACM Queue," [Online]. Available : https://queue.acm.org/detail.cfm?id=2560327. [Accessed 09 March 2023].

[4] "Software-Defined Network," [Online]. Available : https://www.cisco.com/c/en/us/solutions/software-defined-networking/overview.html. [Accessed 12 March 2023].

[5] V. K.Gubrani, M.Schraf, T.V. Lakshman, V. Hilt and E.Morocco, "Abstracting network state in Software Defined Networks (SDN) for rendezvous services", Communications (ICC), 2012 IEEE International Conference, Ottawa, ON, Canada, 2012.

[6] Ould Lamara Sami, Takilt Moussa, "Implementation du SDN dans une structure IP/MPLS", Master, Tizi-Ouzou, 2018

[7] "What is Software-Defined Networking (SDN)?" [Online]. Available: https://www.vmware.com/topics/glossary/content/software-defined-networking.html [Accessed 18 March 2023].

[8] "What is an Enterprise Network" [Online]. Available: https://www.cisco.com/c/en/us/solutions/enterprise-networks/what-is-an-enterprise-network.html [Accessed 22 March 2023].

[9] "What is a Datacentre Network" [Online]. Available: https://www.cisco.com/c/en/us/solutions/data-center-virtualization/what-is-a-data-center.html [Accessed 22 March 2023].

[10] "What Is a WAN? Wide-Area Network" [Online]. Available: https://www.cisco.com/c/en/us/products/switches/what-is-a-wan-wide-area-network.html [Accessed 22 March 2023].

[11] Md. Tariqul Islam, Nazrul Islam, Md. Al Refat, "Node to Node Performance Evaluation through RYU SDN Controller",Wireless Personal Communications, Vol 112, 2020, p 555-570.

[12] Chikhi Mehdi, Djemil Adel, "Etude et implémentation de l'approche Software Defined Network dans un réseau local", Master, Blida, 2019.

[13] "What is network functions virtualization?" [Online]. Available: https://www.vmware.com/topics/glossary/content/network-functions-virtualization-nfv.html [Accessed 01 April 2023].

[14] "Defining The Elements of NFV Architectures" [Online]. Available: https://blog.equinix.com/blog/2019/10/17/networking-for-nerds-defining-the-elements-of-nfv-architectures/ [Accessed 03 April 2023].

[15] "What is Network Function Virtualization (NFV) " [Online]. Available: https://www.blueplanet.com/resources/What-is-NFV-prx.html [Accessed 09 April 2023].

[16] "Benefits of Network Virtualization & Potential Disadvantages" [Online]. Available: https://www.optanix.com/pros-and-cons-of-network-function-virtualization-in-a-nutshell/ [Accessed 12 April 2023].

[17] Ashutosh Kumar Singh1, Shashank Srivastava, Shashwati Banerjea, "Evaluating heuristic techniques as a solution of controller placement problem in SDN", Journal of Ambient Intelligence and Humanized Computing, Vol 31, 2022, P 3761.

[18] Mohammad Ashrafi, Noélia Correia, and Faroq AL-Tam, "A Scalable and Reliable Model for the Placement of Controllers in SDN Networks", 9th International EAI Conference, Faro Portugal, 2018.

[19] G. Ramya, R. Manoharan, "Enhanced optimal placements of multi-controllers in SDN" Journal of Ambient Intelligence and Humanized Computing, Vol 12, 2021, PP 08-10.

[20] EL Hocine Bouzidi, Abdelkader Outtagarts, Rami Langar, Raouf Boutaba, "Dynamic clustering of software defined network switches and controller placement using deep reinforcement learning", computer networks journal, Vol 207, 2022, PP 108-109.

[21] Rong Chai, Qiongfang Yuan, Lei Zhu and Qianbin Chen, "Control plane delay minimization-based capacitated controller placement algorithm for SDN", EURASIP Journal on Wireless Communications and Networking, Vol 10, 2019, PP 1009-1009.

[22] Wen chen, Cong chen, Xueqin jiang, (Member, IEEE), and Leijie liu, "Multi-Controller Placement Towards SDN Based on Louvain Heuristic Algorithm", ieee Xplore, Vol 99, 2018, PP 1-1.

[23] "Documentation for Visual Studio Code" [Online]. Available: https://code.visualstudio.com/docs [Accessed 27 May 2023].

ملخص:

هذا العمل يهدف إلى وضع وحدات التحكم للشبكات المبرمجة بهدف تحديد مواقعها بطريقة تقلل من زمن التأخر بين المحولات وأجهزة التحكم. لتحقيق هذا الهدف تم اختيار طريقة وضع وحدات التحكم اعتمادا على خوارزميتي جيكسترا وخوارزمية التجميع الذاتي وتشير النتائج التجريبية إلى أن الطريقة تقدم أداءً أفضل ويمكن تنفيذها بسهولة لمجموعة متنوعة من التطبيقات.

الكلمات المفتاحية: الشبكات المبرمجة، وحدة التحكم، مشكلة وحدة التحكم

Abstract:

This work aims to place controllers for programmable networks in order to determine their locations in a way that reduces the delay between switches and controller.

To achieve this goal, the method of placing controllers based on both Djikstra's algorithm and K-means algorithm was chosen, and experimental results indicate that this method provides better performance and can be easily implemented for a variety of applications.

Keywords: Software Defined Network, Controller, Controller Problem Placement

Résumé:

Ce travail vise à placer des contrôleurs pour les réseaux programmables afin de déterminer leurs emplacements de manière à réduire le délai entre les commutateurs et les contrôleurs.

Pour atteindre cet objectif, la méthode de placement des contrôleurs basée sur les algorithmes de Djikstra et de K-means a été choisie, et les résultats expérimentaux indiquent que cette méthode offre de meilleures performances et peut être facilement mise en œuvre pour une variété d'applications.

Mots-clés: Réseau programmable, Contrôleur, Le problème de placement du contrôleur