


République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Akli Mohand Oulhadj de Bouira

Faculté des Sciences et des Sciences Appliquées

Département d'Informatique

Mémoire de Licence

en Informatique

Spécialité : *GSI*

Thème

Optimization in Software-Defined Networking

Encadré par

— DJOUABRI Abderrezak

Réalisé par

— ZIANE Mohamed Badr

2020/2021

Remerciements

Louange a ***ALLAH*** *par la grâce duquel les bonnes choses se réalisent, je remercier DIEU le tout puissant, miséricordieux, de m'avoir donné la force, la santé, la volonté et le courage d'accomplir ce modeste travail*

Je voudrais dans un premier temps remercier, mon encadreur ***M.Abderrezake DJOUABRI****, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion.*

Je voudrais aussi d'exprimer ma reconnaissance envers les amis et collègues qui m'ont apporté leur soutien moral et intellectuel tout au long de ma démarche

Je remercie également toute l'équipe pédagogique de l'université de Bouira.

Enfin, j'adresse mon vifs remerciement aux membres du jury qui ont accepté d'évaluer mon travail.

Dédicaces

A mes chers Parents,

A mes chers frères et sœurs,

A tous mes chers(e) amis(e)

Ziane Mohamed Badr

Table des matières

Table des figures

Liste des tableaux

Liste des abreviations

API	Application ¨Programming Interfaces
SBI	Southbound Interface
NBI	Northbound Interface
JVM	JAVA Virtual Machine
LAN	Local Area Network
WAN	Wide Area Network
NAT	Network Address Translation
NOS	Network Operating System
OF	OpenFlow
ONF	Open Networking Foundation
OVSDB	Open VSwitch DataBase
QOS	Quality Of Service
SDN	Software Defined Network
FIFO	First In First Out
OVS	OpenVSwitch
NFV	Network Function Virtualization
VLAN	Virtual LAN
POCO	Pareto Optimal Controller Placement
CPP	Controller Placement Problem
MHNSGA	Multi-start hybrid non-dominated sorting genetic algorithm
CGA-CC	Clustering-based Genetic Algorithm with Cooperative Clusters

Introduction generale

With the continuous advancements in information technology, the Internet has become a large-scale complex infrastructure that has changed people's styles of work, study and life. The complexity of the traditional Internet makes its configuration and management difficult for network administrators when considering network dynamics and diverse application needs. There is therefore an urgent need to design and develop a new network architecture capable of managing the network dynamically and flexibly. The programmable network (Software Defined Networking) is proposed to overcome the weaknesses of the traditional network. As a new network paradigm, SDN revolutionizes network technology by breaking the fundamental idea of traditional networks.

SDN has developed as a network paradigm that extends the adoption of virtualization technologies from servers to networks. Although SDN is still in its infancy, a number of SDN-based network devices and software are being deployed by SDN network operators, including major industry players such as Google's B4, Microsoft's SWAN, and Cisco "Centric Infrastructure Application".

The main idea of SDN is to decouple the control plane and the data plane. The control plans are merged into a single entity (or more) named controller. As a decision maker, the SDN Controller is able to provide Application Program Interfaces (APIs) to top-level applications and allow operators to deploy various network policies flexibly, depending on scenarios and application requirements. Therefore, controllers play an important role in SDN networks.

The representative communication interface of SDN is "OpenFlow", which assumes initially a single controller for simplicity. But with the expansion of the SDN network, a single controller cannot meet all the existing management needs. Having multiple control-

lers improves the reliability of network scalability.

In the present work, we study the controller placement problem (CPP) and are looking to find the optimal location and number of controllers in an SDN network.

This thesis is organized as follows :

The first chapter, titled "Introduction to SDN" is about SDN Architecture, its different components and how it works, Also the various available SDN controllers, the challenges and the benefits, then we went through the SDN use cases to end with the Network Function Virtualization.

The second chapter , titled "Challenges for SDN (CPP)" we talked about the main challenges and the related works that tried to cover these challenges, The performance metrics in SDN (controller placement problem -CPP-) and the objective of controller placement problem.

The third chapter, titled "Comparative analysis between works on CPP" contains the controller placement problem and a comparative analysis between works on CPP.

The fourth chapter, titled "Implementation of one of the challenges related to SDN" was about the implementation of Pareto Optimal Controller Placement and the different Topologies Used in Mininet network emulators.

Chapitre 1

Introduction to SDN

1.1 Introduction

The Internet has become an important part of everyday life. From searching via Google to buying from Amazon to keeping up with friends on Facebook, it is a part of people's daily lives. Behind each of these multibillion dollar companies are networks that have to adapt to ever-changing needs. Because the Internet becomes more invaluable than ever, it is important for it to be more accessible and easier to maintain. The networks, which make up the backbone of the Internet, need to adapt to changes, without being hugely labor intensive in hardware or software changes [1]. SDN is the solution that separates the control plane from the underlying network data plane for both efficient data transport and fine-grained control of network management and services. In this chapter, we will present and introduce in a general way the different concepts of SDN and how it works.

1.2 Software defined networking (SDN)

Software-defined networking (SDN) is an architecture that abstracts different, distinguishable layers of a network in order to make networks agile and flexible. The goal of SDN is to improve network control by enabling enterprises and service providers to respond quickly to changing business requirements.

In a software-defined network, a network engineer or administrator can shape traffic from a centralized control console without having to touch individual switches in the network. A centralized SDN controller will direct the switches to deliver network ser-

vices wherever they're needed, regardless of the specific connections between a server and devices.

This process is a move away from traditional network architecture, in which individual network devices make traffic decisions based on their configured routing tables. SDN has had a role in networking for a decade now, and its evolution and roles have continued to evolve [2].

1.2.1 SDN architecture

A typical representation of SDN architecture comprises three layers : the application layer, the control layer and the infrastructure layer.

The application layer, not surprisingly, contains the typical network applications or functions organizations use. This can include intrusion detection systems, load balancing or firewalls. Where a traditional network would use a specialized appliance, such as a firewall or load balancer, a software-defined network replaces the appliance with an application that uses a controller to manage data plane behavior.

The control layer, represents the centralized SDN controller software that acts as the brain of the software-defined network. This controller resides on a server and manages policies and the flow of traffic throughout the network.

The infrastructure layer, is made up of the physical switches in the network.

These three layers communicate using respective northbound and southbound application programming interfaces (APIs). For example, applications talk to the controller through its northbound interface, while the controller and switches communicate using southbound interfaces, such as OpenFlow – although other protocols exist [2].

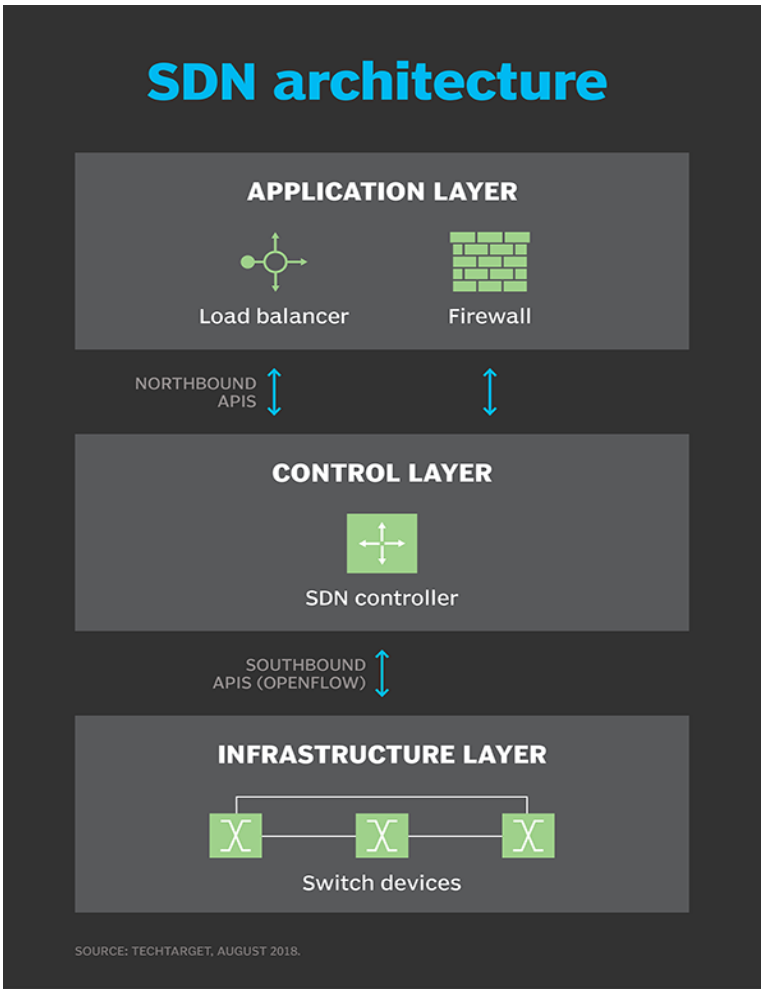

FIGURE 1.1 – A graphic representation of SDN architecture [2]

1.2.2 SDN components

The main components used in SDN are southbound application programming interfaces, SDN controller, and northbound application programming interfaces. Efficient utilization of network resources and protocol management is achieved by the SDN [3]. Therefore, the same OpenFlow devices can be used as and converted as firewall, router, or load balancer or any other application as per the needs. SDN architecture is easy, less costly, easily manageable, scalable, and flexible, and support of the dynamically changed networks is provided by the SDN [4].

Southbound API

For the purpose of communication between the forwarding device and controller, an interface is used and is called as southbound interfaces. Two popular protocols used in

southbound interfaces are ForCES and OpenFlow.

OpenFlow : The OpenFlow paradigm contains three basic parts : pure OpenFlow switch or hybrid switch, SDN controller, and for the purpose of communication a secure connection in between the SDN controller and the OpenFlow switches. An OpenFlow switch maintains a flow table to handle incoming network packets. These OpenFlow switches maintain a flow table to handle particular packets. Each flow table of an OpenFlow switch contains a number of flow table rules in it. Further, each of these flow rules consists of matching fields, actions, and statistics which decides how particularly packets will handle a set of actions which helps to determine the handling of matching packets. In collecting the parameters like how much number of packets is received and the corresponding duration for the traffic flow, these counters are used [5].

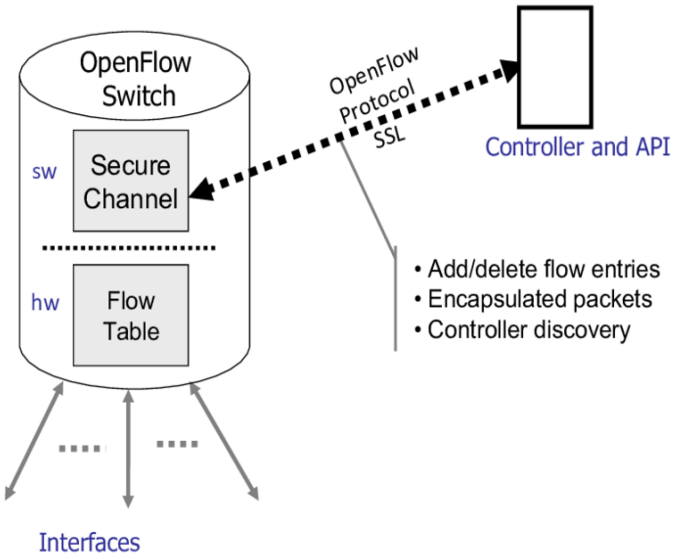

FIGURE 1.2 – The OpenFlow protocol [95]

Northbound API

A northbound interface is an interface that provides a communication between the applications and the controller. Another API called a southbound interface is useful in providing a communication channel between the switches or data paths and the SDN controller [6].

West and Eastbound API

Westbound Application Programming Interface (API) : This interface acts as a channel for providing the interface between SDN management plane and totally different network domains. Network state info and routing choices for every controller are changed through this. Bury domain routing protocols like BGP square measure are used [7]. Eastbound Application Programming Interface (API) : Communication is completed from management plane to non-SDN domains. Depending upon the technology utilized in non-SDN domains, its implementation is proportional [7].

1.2.3 How SDN works ?

SDN encompasses several types of technologies, including functional separation, network virtualization and automation through programmability.

Originally, SDN technology focused solely on the separation of the network control plane from the data plane. While the control plane makes decisions about how packets should flow through the network, the data plane actually moves packets from place to place.

In a classic SDN scenario, a packet arrives at a network switch, and rules built into the switch's proprietary firmware tell the switch where to forward the packet. These packet-handling rules are sent to the switch from the centralized controller.

The switch – also known as a data plane device – queries the controller for guidance as needed, and it provides the controller with information about the traffic it handles. The switch sends every packet going to the same destination along the same path and treats all the packets the exact same way.

Software-defined networking uses an operation mode that is sometimes called adaptive or dynamic, in which a switch issues a route request to a controller for a packet that does not have a specific route. This process is separate from adaptive routing, which issues route requests through routers and algorithms based on the network topology, not through a controller.

The virtualization aspect of SDN comes into play through a virtual overlay, which is a logically separate network on top of the physical network. Users can implement end-to-end overlays to abstract the underlying network and segment network traffic. This microsegmentation is especially useful for service providers and operators with multi-

tenant cloud environments and cloud services, as they can provision a separate virtual network with specific policies for each tenant [2].

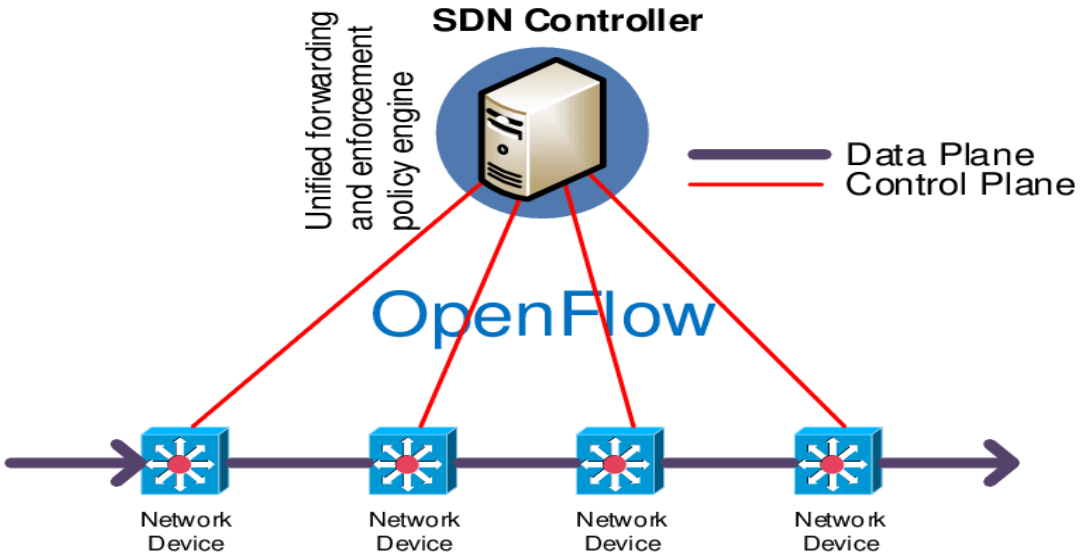

FIGURE 1.3 – SDN network example [96]

1.2.4 Models of SDN

There are four primary types of software-defined networking (SDN), each with its own merits :

1. Open SDN uses open protocols to control the virtual and physical devices responsible for routing the data packets.
2. API SDN uses programming interfaces, often called southbound APIs, to control the flow of data to and from each device.
3. Overlay Model SDN creates a virtual network above existing hardware, providing tunnels containing channels to data centers. This model then allocates bandwidth in each channel and assigns devices to each channel.
4. Hybrid Model SDN combines SDN and traditional networking, allowing the optimal protocol to be assigned for each type of traffic. Hybrid SDN is often used as a phase-in approach to SDN [8].

1.3 SDN Controllers

Controller in the SDN acts as the vital element. All the information to the data paths or data elements such as network switches/routers is given through southbound application programming interface and information to the applications such as firewall, load balancer, and business logic is achieved through the northbound application programming interface. The SDN controller is situated in the middle of the architecture in between the network elements and the SDN applications [9]. Whole communication between SDN applications and the network devices will pass through the SDN controller. The configuration of the various network devices is achieved by the SDN controller using the most common protocol called as OpenFlow protocol and hence for the flow of network traffic adopts the optimal network path [10].

Controller Scalability

Development of new applications and modifications in the behavior of existing application are much easier with centralized controller. But more requests and events are forwarded to the controller when the size of the network increases, and at particular point, the single SDN controller fails to handle all the requests. The NOX controller is a centralized controller that may be sufficient for a small network, but it could not be sufficient for a large network like data center. We can solve this problem by physically distributing the controller elements while maintaining the global view of the network [11].
Hyper Flow and Kandoo are examples of distributed control planes.

Reactive and Proactive

First, in the reactive approach, it states that, whenever a new network traffic packet enters into the OpenFlow switch, the switch performs the searching operation and finds outs any matching field to the corresponding packet. If there is no flow match, the switch immediately sends that particular packet to the SDN controller. A rule is created by the controller in the flow table. It makes better use of the flow table but each flow requires extra time to set up a flow. The downside is that if the connection between controller and switch is lost, the switch has limited functionality [12]. In the proactive approach, SDN controllers prepopulate network traffic rules added into the flow table of an OpenFlow switch, and thus this packet-in event does not happen. Advantages are : no extra time is

required to set up a flow, and if connection is lost it does not affect the switches.

1.3.1 Various Available SDN Controllers

An SDN controller is also defined as a network operating system (NOS). It allows the abstraction and hence decreasing the complexity of the network is underlying. The main idea was to distinguish between the operating system and network operating system. There is already a significant number of controllers available :
NOX , POX , Floodlight , Ryu , OpenDaylight ,etc.
The information about various controllers, the languages in which they are written, and a brief description are listed in corresponding table [13].

	POX	Ryu	Trema	FloodLight	OpenDaylight
Interfaces	SB (OpenFlow)	SB (OpenFlow) +SB Management (OVSDB JSON)	SB (OpenFlow)	SB (OpenFlow) NB (Java & REST)	SB (OpenFlow & Others SB Protocols) NB (REST & Java RPC)
Virtualization	Mininet & Open vSwitch	Mininet & Open vSwitch	Built-in Emulation Virtual Tool	Mininet & Open vSwitch	Mininet & Open vSwitch
GUI	Yes	Yes (Initial Phase)	No	Web UI (Using REST)	Yes
REST API	No	Yes (For SB Interface only)	No	Yes	Yes
Productivity	Medium	Medium	High	Medium	Medium
Open Source	Yes	Yes	Yes	Yes	Yes
Documentation	Poor	Medium	Medium	Good	Medium
Language Support	Python	Python-Specific + Message Passing Reference	C/Ruby	Java + Any language that uses REST	Java
Modularity	Medium	Medium	Medium	High	High
Platform Support	Linux, Mac OS, and Windows	Most Supported on Linux	Linux Only	Linux, Mac & Windows	Linux
TLS Support	Yes	Yes	Yes	Yes	Yes
Age	1 year	1 year	2 years	2 years	2 Month
OpenFlow Support	OF v1.0	OF v1.0 v2.0 v3.0 & Nicira Extensions	OF v1.0	OF v1.0	OF v1.0
OpenStack Networking (Quantum)	NO	Strong	Weak	Medium	Medium

FIGURE 1.4 – Feature-based Comparison and Selection of SDN Controllers [97]

1.4 SDN Challenges

By decoupling the data and control planes of traditional networks, software-defined networking promises a new generation of low-cost hardware that software development communities can coalesce around to create cost-effective, robust network services tailored to individual needs. However, mature enterprises and networking organizations must overcome several challenges to fully realize SDN's benefits. Following are four common challenges in SDN created by the paradigm shift of software-defined services from traditional hardware-based networking :

1. **Security :** Because the control plane plays such a central function in an SDN architecture, security strategies must focus on protecting the controller and authenticating an application's access to the control plane.

   New services can introduce security threats as programmers and network administrators may unwittingly introduce at-risk code and extend the threat network wide through a centralized or partially distributed controller. Related, SDN's virtual nature can result in the creation of countless network segments, each with its own risk and security requirements.

2. **Scalability :** Since the SDN architecture includes centralized or partially distributed controllers interfacing with data planes on multiple devices, the possibility exists for the controllers to become a network bottleneck. In particular, large networks with volumes of networking requests can overwhelm controllers. As networks grow, the bottleneck tightens and network performance degrades.

   Scalability may be improved with a decentralized control architecture or similar solution, such as split or fully distributed control planes. But such solutions can introduce new obstacles such as convergence and countless control instances to configure and manage.

3. **Interoperability :** For new networks, implementing SDN is fairly straightforward – all network devices are SDN-ready. Transitioning a legacy network to SDN is another story as the legacy network is likely supporting active business and networking systems. Enterprises and most networking environments have to transition to SDN, requiring a period of interoperability with a hybrid legacy-SDN infrastructure.

   SDN and legacy network nodes can operate together, with help from an appropriate

protocol that supports SDN communications while providing backward compatibility with existing IP and MPLS control plane technologies – reducing the cost, risk, and disruption of services while transitioning to SDN.

4. **Performance :** Performance is the greatest issue for all networks. Regardless of how robust, secure, scalable, or interoperable a network is, it's unusable if it lacks performance.

   The separate control and data plan architecture can introduce latency into SDN. In large networks this can build to an unacceptable level of delay, degrading network performance. Related, controller response time and throughput can contribute to poor performance, with the combined effect causing scalability issues.

   The solution for many performance issues in large and growing networks is to push more intelligence to the data plane or move to a distributed control plane architecture of some type. While this can improve SDN performance, it's moving somewhat away from the intent of SDN and replicating traditional networks built on fully distributed intelligent devices. A balance has to be sought where virtualization is maintained without degrading network performance or introducing potential single points of failure [14].

Controller placement problem in SDN

A distinctive feature of software-defined networking (SDN) is a logically centralized control plane realized using multiple physical controllers. The placement of the controllers, the so-called controller placement problem (CPP), is a crucial design issue. It influences network performance parameters such as latency, flow setup time, network availability, load balance of the controllers, and energy consumption [15].

Recently, a variety of solutions have been proposed to tackle the controller placement problem in SDN. The objectives include minimizing the latency between controllers and their associated switches, enhancing reliability and resilience of the network, and minimizing deployment cost and energy consumption [16].

1.5 Benefits of SDN

Decoupling the control plane and the data plane provides better management of the network as well as flexibility by introducing programmability in the network. This structure enables network management to be carried out on the control plane, having no effect on data flows traversing the data plane. A few of the benefits of SDN include :

1. **Enhanced Configurations**

   As network size continually expands and new devices are added to the network, proper configurations need to be implemented for effective network operation. Owing to the heterogeneous nature of network devices, a traditional approach is considered tiresome and fallible. With SDN, the numerous devices are connected to a centralized control plane where configuration and management is carried out from a single point.

2. **Enhanced Performance**

   This is considered the main objective of SDN. Performance optimization is achievable through the centralized control plane, which SDN offers. As such, all challenges pertaining to performance optimization would become manageable. Consequently, traditional issues, which include ; Quality of Service (QoS) support, end-to-end congestion, etc. can be developed and exploited to confirm their efficacy in enhancing system performance (Xia, Wen, Foh, Niyato, Xie,2015).

3. **Reduced Cost**

   With SDN, the centralized control plane is responsible for managing and orchestrating the network, which can be handled by one single controller. In other words, SDN removes network infrastructure management and control away from the network devices and alternatively puts it into software, thereby reducing operating costs.

4. **Encouraging Innovation**

   It is impossible to absolutely foresee and precisely meet the demands of future applications on networks, thereby leading to the deployment of applications, network services and new ideas. It provides a programmable network platform for the purpose of experimenting and carrying out implementations, which in turn enhances innovation [17].

1.6 SDN use cases

Some use cases for SDN include :

1. **DevOps**

   An approach based on software-defined networking can facilitate DevOps by automating app updates and deployments, including automating IT infrastructure components as the DevOps apps and platforms are deployed.

2. **Campus networks**

   Campus networks can be difficult to manage, especially with the ongoing need to unify Wi-Fi and Ethernet networks. SDN controllers can benefit campus networks by offering centralized management and automation, improved security and application-level quality of service across the network.

3. **Service provider networks**

   SDN helps service providers simplify and automate the provisioning of their networks for end-to-end network and service management and control.

4. **Data center security**

   SDN supports more targeted protection and simplifies firewall administration. Generally, an enterprise depends on a traditional perimeter firewall to secure its entire datacenter. However, a company can create a distributed firewall system by adding virtual firewalls to protect the virtual machines. This extra layer of firewall security helps prevent a breach in one virtual machine from jumping to another. Also, SDN centralized control and automation allows the admin to view, modify and control network activity to reduce the risk of a breach to begin with [2].

1.7 Network Function Virtualization (NFV) vs SDN

First, NFV is the virtualization of network components, decoupling them from the hardware appliances that run them in favor of virtual machines. These functions can range from load balancing to firewalls to virtual routing.

Both SDN and NFV rely on the software layer to virtually manage the network. But each solution operates at different levels. They differ in how they separate virtual resources and functions.

The Software Defined Network (SDN) approach separates the control plane (which decides where traffic is sent) from the data plane (which transfers packets to specific destinations) by making the network fully programmable. It does this by relying on switches programmed by an SDN controller that uses a standard protocol like OpenFlow. This solution allows network administrators to manage network services with software that centralizes its programming and offers a much faster and more user-friendly configuration.

In the Network Functions Virtualization (NFV) approach, network functions based on physical appliances are virtualized via a hypervisor, allowing the network to grow without the need to add dedicated devices. This solution separates network control functions such as routers, firewalls, load balancers from the hardware layer and allows these more advanced features to run from a virtual machine. All virtual machines are orchestrated by a single hypervisor. When the hypervisor controls the network functions, the services can be run on standard servers [18].

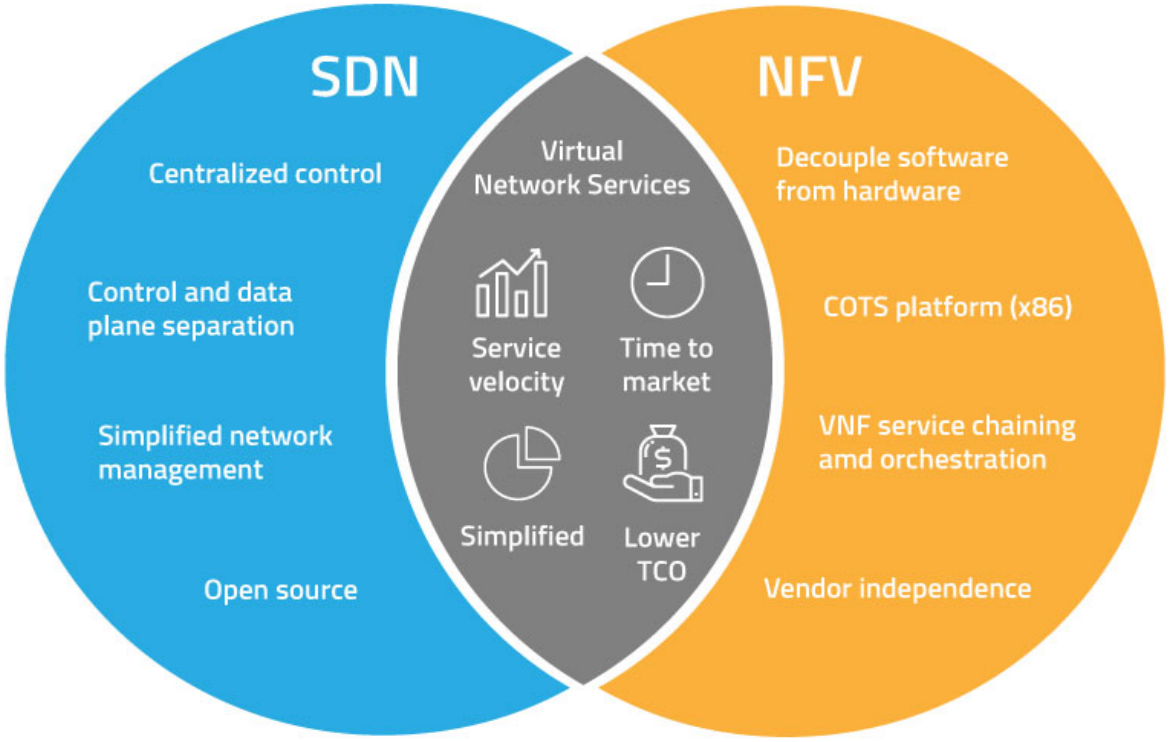

FIGURE 1.5 – Relationship and differences between SDN and NFV [98]

1.8 Conclusion

In this chapter, we have presented in a general way the different concepts of SDN and how does it work. We also mentioned the different models of SDN Controllers and a general view to SDN challenges .

In the next chapter, we will continue with a presentation detailed of Challenges for SDN

Chapitre 2

Challenges for SDN (CPP)

2.1 Introduction

The new network requirements are rising with internet scalability and expansion of its coverage. The traditional network cannot support recent needings, so the Software Defined Network (SDN) makes the network more programmable, flexible, and controllable, though the scalability is the most important challenge on which the researchers are working in SDN. Multiple controllers are a necessity of current SDN, so the optimum number of controllers and their placement is a problem called controller placement problem (CPP). There are defined different functions to optimize scalability, reliability, and others in SDN which consider various metrics in recent research papers. This optimization problem is NP-hard [19].

In this chapter, we will present and introduce one of SDN challenges (controller placement problem) and all related works to this challenge.

2.2 SDN Security Challenges

SDN introduces a new networking paradigm, and its impact is in the form of a new framework, new components, structural layers, and interfaces. SDN brings with it new security challenges beyond those existed in traditional networks. As SDN decouples the control plane from the data plane, the technology brings with it new sets of components, interfaces, as well as many new security issues. Security challenges in SDN can be divided based on its three layers : the data plane, the control plane, and the application

plane. The data plane can suffer from various security threats such as malicious OpenFlow switches, flow rule discovery, flooding attacks (e.g., switch flow table flooding), forged or faked traffic flows, credential management, and insider malicious host. The application plane inherits security challenges such as unauthorized or unauthenticated applications, fraudulent role insertion, lack of authentication methods, and lack of secure provisioning. The control plane faces several security issues related to centralized SDN controller, communication interfaces, policy enforcement, flow rule modification for modifying packets, controller-switch communication flood, system level SDN security challenges (related to lack of auditing accountability mechanisms), and lack of trust between the SDN controller and third-party applications [9]. Since the control plane in the SDN architecture acts as the heart of this virtual network infrastructure, security vulnerabilities on this layer can cause failure to the entire virtual network architecture.

Scott-Hayward et al. presented a comprehensive analysis of the security challenges of SDN [27]. Security challenges associated with the SDN framework by affected layer/interface are categorized as follows :

1. **Application Layer**

   Unauthorized access is through the unauthenticated application. Malicious applications may introduce fraudulent rule insertion. Configuration issues arise from lack of policy enforcement.

2. **Control Layer**

   Unauthorized access can be introduced through unauthorized controller access and unauthenticated application. Data modification is introduced in the form of flow rule modification to modify packets. Malicious application

2.3 Cloud Security Challenges

While there are many security concerns in cloud computing, Cloud Security Alliance (CSA) released twelve critical security threats specifically related to the shared, on-demand nature of cloud computing for cloud computing with the highest impact on enterprise business [5] :

1. Data Breaches A data breach is an incident in which sensitive, protected, or confi-

dential information is released, viewed, stolen, or used by an individual who is not authorized to do so.

2. Weak Identity, Credential, and Access Management Data breaches and enabling of attacks can occur because of a lack of scalable identity access management systems, failure to use multifactor authentication, weak password use, and a lack of continuous automated rotation of cryptographic keys, passwords, and certificates
3. Insecure APIs (Application Programming Interface) Provisioning, management, orchestration, and monitoring are all performed using a set of software user interfaces (UIs) or application programming interfaces. These interfaces must be designed with adequate controls to protect against both accidental and malicious attempts to circumvent policy.
4. System and Application Vulnerabilities System vulnerabilities are exploitable bugs in programs that attackers can use to infiltrate a computer system for stealing data, taking control of the system or disrupting service operations.
5. Account Hijacking This is a significant threat, and cloud users must be aware of and guard against all methods such as phishing, fraud, and exploitation of software vulnerabilities to steal credentials.
6. Malicious Insiders A malicious insider threat to an organization is a current or former employee, contractor, or another business partner who has authorized access to an organization's network, system, or data and intentionally misused that access in a manner that negatively affected the CIAAA of the organization's information system
7. Advanced Persistent Threats (APTs) These are a parasitical form of cyber-attack that infiltrates systems to establish a foothold in the computing infrastructure of target companies from which they smuggle data and intellectual property.
8. Data Loss Data stored in the cloud can be lost for reasons other than malicious attacks. An accidental deletion by the cloud service provider or a physical catastrophe such as a fire or earthquake can lead to the permanent loss of customer data.
9. Insufficient Due Diligence An organization that rushes to adopt cloud technologies and chooses cloud service providers (CSPs) without performing due diligence ex-

poses itself to a myriad of commercial, financial, technical, legal, and compliance risks.

10. Abuse and Nefarious Use of Cloud Services Poorly secured cloud service deployments, free cloud service trials, and fraudulent account sign-ups via payment instrument fraud expose cloud computing models such as IaaS, PaaS, and SaaS to malicious attacks. Malicious actors may leverage cloud computing resources to target users, organizations, or other cloud providers
11. Denial of Service (DoS) Denial-of-service attacks are attacks meant to prevent users of a service from being able to access their data or their applications by forcing the targeted cloud service to consume inordinate amounts of finite system resources so that the service cannot respond to legitimate users.
12. Shared Technology Issues Cloud service providers deliver their services by sharing infrastructure, platforms, or applications. The infrastructure supporting cloud services deployment may not have been designed to offer strong isolation properties for a multi-tenant architecture (IaaS), re-deployable platforms (PaaS), or multi-customer applications (SaaS). This can lead to shared technology vulnerabilities that can potentially be exploited in all delivery models.

2.4 Controller Placement Problem

CPP is one of the critical problems faced in the realm of SDN when multiple controllers are deployed in the management of the networks as well as the enforcement of network policies in such controllers. It deals with finding the required feasible or optimal number of controllers and their location in the SDN networks to meet various performance and QoS requirements [1]. Generally, most CPP is modeled as a graph, G = (V, E, U) topology where V is the set of n switches, E is the set of edges (physical links) among switches or controllers and U is the set of k controllers [15]. Studies on CPP are centered on designing techniques to solve for the value of k and the relation given by U $\rightarrow$ V, which for instance, is the shortest path latencies between each pair of nodes when minimizing latency as the only objective in the objective function [12], [15]. The formulation is an optimization problem that is either data-driven [16] or metricdriven formulated to find the minimum or maximize cost, the optimal number of controllers, switch-controller (SC)

latency, controller synchronization time or hybrid metrics or multi-objective optimization problem about controller location or placement [1]. The model allows searching for the number, type, and location of controllers [1], [9] which are the key to network performance and QoS [2], [3]. CPP was first considered by Heller et al. [17] as a non-deterministic polynomial (NP)-hard problem and is similar to the facility location problem [2], [3]. Solving CPP is challenging and requires proper planning and good decisions making to achieve optimal location and satisfactory results [2]. For multiple controllers placement, several factors influence the SDN performance such as reliability, controller load balancing, latency, operational costs, and response time of events [1], [2], [6], [8], [10]. For instance :

1. LATENCY Latency is one of the core factors considered in CPP. It depends highly on the distance between nodes in a network which is critical to packet propagation. Two latencies are important : propagation and processing. Propagation latency is the response time among the controllers or switches which is influenced by the distance between them and the processing latency is highly influenced by the controller's load. In other words, the optimality of the latencies influences both Controller-to-Controller (CC) and SC delays respectively. Moreover, processing latency increases as the flows from several switches increases, and whenever the SC latency exceeds a threshold, the overall latency of the network increases greatly. This, therefore, affects the network availability in terms of responding to network events or push forwarding commands to switches on time.
2. CONTROLLER LOAD BALANCING An increase in the number of switches controlled by a controller increases the controller's load as well. If a load of a controller exceeds its processing ability, it could result in queueing delays or new request not served from the switches since it has only the capacity to manage a limited number of switches' request at the same time [1], [8]. That is, the controller fails to process requests, and the communication overhead of SC or CC increases, therefore, it is important SC assignment is well-balanced. However, finding the best placement with minimum controllers and switches assignment is a difficult task when load balancing is considered [14].
3. FAULT TOLERANCE In the SDN, each switch is assigned to a controller and a controller's failure affects the link connecting the controller and switch since no requests the switch will be received or processed at the controller. Accordingly,

link failure negatively impacts the controlled switches and restrict some functions of the control plane whenever the SC connection is lost. The switch receives no new routing instructions and all packets are dropped [8]. Thus, the best placement that minimizes the number of controllers while ensuring reliability is important.

4. OPTIMAL NUMBER OF CONTROLLERS In this perspective, an increase in the number of switches also increases the network complexity to meet various network performance requirements. That is, having several unplanned switches assigned to a controller brings about increase complexity on the controller since it has to process several requests within a given time interval. This poses a great challenge when determining how many controllers should be deployed and their placement to increase network performance and QoS. However, several approaches such as traversal searching to find optimum performance numbers is considered time-consuming [8].
5. COST Optimally and merely placing the controllers in a large scale network topology has a huge impact on the overall expenditure. That is, finding the optimal number of controllers to deploy involves considering all possible costs such as budget limitations, purchasing, repairing, and maintenance costs [1]. The terms CAPEX and OPEX are considered in the SDN controller placement when minimizing the total cost [27]. Thus, finding the best number of controllers to be deployed that minimizes cost is an important factor.
6. INTER–CONTROLLER COMMUNICATION In the SDN, CC or inter-controller communication is maintained via state synchronization to achieve global consistency. In this case, controllers communicate when they wish to pass a message to switches controlled by others. Therefore, such communication impacts the performance of the end-to-end communication between different switches controlled by different controllers [8] and is a critical factor in the perspective of CPP.

   In general, during CPP that involves identifying the required number of controllers and their locations, the key considerations are the performance metrics [5], [10], [12].The metrics are used to evaluate the quality of the different controller placement in the network and are critical to its performance, QoS, efficiency, scalability, and reliability [5], [10], [12].

The SDN controller receives network policies from the logical functions like automated

security management and general applications and load balance via northbound interface, in addition to the administrator management via management interface. In general, the management interface is an internal interface to SDN controller and may be more trustable than northbound interface. Two new logical functions designed by this paper are automated security management which supports SDN controller detecting and mitigating security attacks automatically, and network security monitor which supports administrators in monitoring security status of the entire SDN network.

The SDN controller synchronizes flow entries with switches through southbound interface to route the flows correctly. The SDN controller may exchange some control information of the switches on the border with another SDN controller through east-west interface.

The SDN controller is designed to provide security services mentioned in section II to solve common security issues with some changes and to handle some new security challenges. To provide such security services, the SDN controller comprises two kinds of modules. One kind of modules is new and designed or modified by this paper, such as a policy module, a characteristic of policy module, a role of policy creator module, a security privilege level of role module, a packet scan detection module, a policy conversion module, a flow table manager, flow tables, a flow entry synchronization module, a network security monitor support module. Another kind of modules is to reuse existing security mechanisms, such as an authentication and authorization module (with supporting network API authorization), an integrity and confidentiality module, a system security management module, a log and audit module, and a trusted execution environment.

The policy module is configured to receive one or more policies (including network policies, especially security policies) from its northbound interface (i.e., from automated security management module, general applications module, and Load Balance module) and its management interface (i.e., Administrator Management module). In order to guarantee that received policies are secure, policy creators (e.g., automated security management module, general applications, load balance module, and administrator management module) are authenticated and authorized by authentication and authorization module of the SDN controller.

Moreover, confidentiality and integrity checks are provided by integrity and confidentiality module for these policies in transportation.

2.4.1 ANALYSIS OF CPP STRATEGIES

Recently, several approaches have been proposed and developed to solve SDN-based CPP and each approach has its own set of objectives which are either to minimize or max imize or both [3]. Existing approaches in terms of controller allocation and the optimal number of controllers have been classified as shown in Fig. 2. Fig. 2 present a simple yet informative classification of CPP based on several factors or metrics that influence SDN performance and QoS as well as how such metrics are optimized [3], [15], [23].

As shown in Fig. 2, latency is considered the core influencing factor in the CPP due to the criticality of message exchanges involving SC and CC in the SDN. Albeit there are several network latencies, propagation, and processing latencies are the important latencies critical to CPP [15], [23], [60]. Moreover, in a multi-domain network setting, propagation latency is composed of two types : average latency (avg latency) and the worst-case latency (worst latency) [15], [58]. Both SC avg latency [8], [17], [31], [58], [59] and SC orst latency [2], [28], [34]) are considered within a given network domain while (CC avg latency) [35], [44], [50], [59] is considered for inter- controller communication between domains [15].

Accordingly, SC avg latency constitutes the average packet transmission delay between SC in the SDN and its formulation is shown in Equation 1, while SC worst latency is the maximum transmission delay between SC as formulated in Equation 2 [15], [23].

Moreover, reliability involves measures set in place to ensure that in the event failure, there is no single point of failure and the communication between controllers and switches is not interrupted. This is the basis of its criticality in SDNbased CPP [15], [23] and several approaches exist to improving reliability such as multiple controllers [7], [15], multiple control-path [15], [23], [36]–[38], [54], and minimizing control-path [15], [33], [39].

While the multiple control-path ensures that there exist at least two disjoint paths linking a controller and a switch, multiple controllers ensure a switch is linked to several controllers to eliminate a single point of failure as experienced with a centralized controller. Moreover, minimizing control-path involves a reduction in the physical unit of the control-path, thereby improving reliability. These approaches are based on the probability of failur [15] and Equation 5 presents the formulation for multiple controllers where (v, u) indicates control-path availability probability.

Lastly, when multi-objectives [6], [50] are considered in the controller placement,

though conflicting in nature, two or more metrics are considered. Albeit, there is no fixed number of metrics to optimize, it all depends on the researcher's objective. In this case, the problem is formulated as shown in Equation 9 [15], [23].

In the same vein, during multiple controllers' placement, an influencing factor such as the cost is considered from two perspectives : deployment [1], [40], [41] and energy consumed in the network [15], [44]. Cost in terms of deployment is tackled by adopting a location-allocation strategy that minimizes the overall cost of controllers, switches, and their running expenses given as CAPEX and OPEX [3], [23]. Equation 7 shows the problem formulation for deployment cost (Dcost) where Cx, Cy, and Ccin are the cost of controllers, controllers, and switches connection and inter-controller connection respectively [15].

In the realm of CPP, several approaches are based on bruteforce, k-center, k-means, k-critical, and other clustering algorithms. TABLE 1 summarizes the algorithm, implementation, and metrics considered in each study

In a similar study, Kobo et al. [43] proposed a CPP for SDWSN intending to optimize a distributed control system for simplicity and application in real-world network deployment. They used both k-means and k-center as well as integrated both CPP and efficient controller re-election

In the same vein, Xiao et al. [55] proposed a network partitioning based CPP for an SDN-based WAN. The authors considered propagation latency, reliability, and load balancing as the important performance metrics and proposed a Spectral clustering Placement Algorithm that achieves the task of large network petitioning into different domains. The goal of the CPP approach was to achieve an optimum number of controllers' selection and their location in the WAN while maximizing the reliability and minimizing latency as well as balancing loads among controllers. The effectiveness of the algorithm was evaluated via simulation using Matlab based framework : Beacon controllers and Cbench as well as topologies from Topology Zoo OS3E Internet2 topology. The results obtained showed the efficacy of the algorithm in solving SDN-based controller placement. The authors also advocated for the need the future. Similarly, Xiao et al. [55] work is extended by Xiao et al. [56] to further address issues of network and domain partitioning in the SDN-based WAN. The authors exploited the benefits of spectral clustering to spectral clustering-based partition and placement algorithms known as a K self-adaptive SDN controller placement

2.5 RELATED SURVEY WORKS

This section presents some of the related works which are separated into three parts : papers on different control architecture, survey papers on CPP outlining the algorithms, problem formulation, the performance metrics, few works that suggested various approaches, and models that are considered in CPP. The discussion is as follows :

Karakus and Durresi [18] conducted a comprehensive survey on SDN-based control plane scalability issues by providing categorization and taxonomy of the state-ofthe-art. Categorization involved two approaches : topology and mechanism-related. The topology-related approaches involved the topology of different architectures, their relationship, and scalability concerns concerning centralized and distributed controllers design.

The mechanism-related involved several mechanisms to optimize controllers and their scalability challenges such as parallel-based and routing scheme-based optimization. The authors further identified the control plane and data plane separation, controller load, and SC communication delay as the key bottlenecks to SDN scalability. Several challenges were also found and reported as well as open problems that need further investigations to ensure more scalability in the SDN such as controller placement, controllers' failure, flow setup latency, state or policy distribution or consistency and so on [18]. Similarly, Bannour et al.

[19] also surveyed on distributed SDN control with a detailed analysis of the state-of-the-art of the distributed controllers' platforms by evaluating their merits and demerits for extensive comprehension. They provided both physical and logical classification as potential guidelines for research and deployment. Discussions were provided on several critical challenges and offered insights into developing and future research trends in distributed SDN control.

Accordingly, scalability, consistency, reliability, and interoperability were identified as parts of the key challenges confronting the design of an efficient and robust distributed SDN controller platforms. Insights offered to counter the challenges include the introduction of major standardization at all levels of CC communication, effective CPP, and knowledge sharing problem. In the same vein, Zhang et al. [20] surveyed recent progress in multiple SDN controllers. They discussed the benefits, detailed design principles, and the architecture as well as the challenges faced by multiple controllers. Moreover, the strengths and weaknesses of several different research work on controller placement and

scheduling were also discussed and analyzed. They presented some future works and research directions. Oktian et al. [21] also surveyed different design approaches to ensure a logically centralized view in multiple distributed controllers. They found that such approaches can be classified into different design choices among SDN adopters and each choice may impact many issues like robustness, scalability, privacy, and consistency. Moreover, they provided an analysis of each of the models in terms of their pros and cons. The findings showed each design produced some features and one can succeed in resolving one issue but may fail in another. They presented the design choice that can be used to construct a distributed controller that overcome all the above issues. In terms of CPP, Wang et al. [22] surveyed state-of-theart solutions to CPP and provided a taxonomy of the CPP based on their objectives such as minimize network latency, maximize reliability, and resilience as well as minimize the cost of deployment and energy consumption. They proposed a new approach to minimize the propagation latency between SC, suggested several types of future research and highlighted some research challenges, and open relevant issues on CPP. Some of the issues identified include the need for an efficient algorithm, multi-objective optimization, cooperation among controllers, and cost awareness.

Similarly, Lu et al. [15] comprehensively surveyed the state-of-the-art of SDN CPP with a focus on the optimization objective. They provided discussions on CPP and classified it into four aspects such as latency, reliability, cost, and multi-objective concerning their objectives. Also provided an analysis of specific algorithms, given the methods and simulations in different application settings. Moreover, several open issues and research challenges for future work on CPP were identified and reported. Some of the issues found include the need for efficient CPP algorithm, cost-awareness, attack-awareness, and virtualized CPP. Accordingly, Singh and Srivastava [23] classified CPP into minimizing, maximizing, and multi-objective approaches. For each approach, they carry out an in-depth analysis of the solutions, their limitations, and suggested several open and future research to bring innovations and reliable CPP. Some of the identified challenges include network partitioning approach to CPP that considers load balancing and all possible failures, application dynamic clustering into CPP, implementation, and deployment large-scale SDN, QoS in function placement problem for network function virtualization, finding the advantages and disadvantages of clustering mechanisms, ILP, Greedy, Heuristics approaches of CPP as well as the efficient solution for large-scale networks with dynamic traffic load

and fault. Das et al. [2] also comprehensively surveyed the SDN-based CPP and discussed various developments in the area.

They classified CPP formulation based on several performance metrics such as latency, QoS, resilience, etc. as well as highlighted on them. Moreover, they summarized the various schemes for solving CPP and their limitations and broadly categorized them into optimal based solutions and heuristic-based suboptimal solutions. They also highlighted CPP application in a wide range of contexts such as mobile/cellular, data-center, large-scale WAN, and nextgeneration networks such as VANETs and 5G. Several open issues and potential research directions were also provided. Furthermore, Kumari and Sairam [46] also performed a comprehensive review of several performance metrics and the features of the existing CPP solutions from the perspective of SDN in terms of wired or wireless networks. They also provided several future research directions such as controller addition and deletion, controller relocating, switch migration issues, CPP for domains like IoT, sensor networks, etc. In a similar work, Hollinghurst et al. [24] also performed an analysis of four different algorithms used for CPP such as the kmeans++, full search, local search, and linear programming (LP). They compared the scalability and the accuracy of the algorithms for the k-median problem. Experiments conducted show the scalability of controllers' placement varied considerably for algorithmic complexity.

Full search algorithms were considered infeasible for large-scale networks, LP algorithms suitable for only restricted network size, while the local search and the k-means++ algorithms were considered the most scalable. In terms of accuracy, full search algorithms were found to be optimal while local search and k-means++ had varied standard deviation and reduced variance respectively. Lastly, Adebayo et al. [25], reviewed four CPP models with a focus on their objectives, strengths, and weaknesses. They presented an analysis of the feature selection methods employed in each model to search for the optimal number to be deployed and their location as well as their impact on the accuracy and convergence time. Findings show that the models were not suitable for real-world applications and concluded that accuracy and complexity constraints can only be met if a meta-heuristic algorithm is used rather than a selective search approach.

Furthermore, albeit three of the models considered traffic conditions, they suffered high convergence time. Thus, high accuracy and low convergence time can be achieved via refinement of the feature selection approach. In order words, optimality via maximizing

accuracy and minimizing convergence time. In a nutshell and according to the authors, the paper provides a stepping stone for optimized controller placement model development for SDWAN. In the above-related works, some of the studies focused on the general issues of multiple controllers in SDN such as scalability, etc. while others focused on CPP approaches that addressed the issues in terms of the impact of different types of metrics or objectives on performance and QoS as well as problem formulation. Also, they provided research directions to address some of the issues in CPP in general. However, this paper performed and in-depth analysis of the most utilized and recent approaches to address SDN CPP.

We specifically focused on the strategies and algorithms used in each case, metrics considered, metrics formulation, the strengths, and possible limitations. Most importantly, we identified various future research directions with a focus on designing a CPP scheme for the SDWSN that is efficient and dynamic. More information about metrics formulation can be found in [2], [15], [22], [23].

2.6 Challenges of Traditional Networking in Future Communication Networks

Usage and demand for technology is growing at a very rapid pace. Networks continue to expand and become more complex as network infrastructure enlarges. In addition, new users and network services are constantly being added to the network. Both new users and network services demand huge network resources, which is increasing exponentially. With regards to increasing size and complexity of networks, traditional approaches for network management would be highly inefficient. This places a serious strain on network operators as they are faced with the task of implementing diverse configurations and keep track of innumerous events on the network.

There has been a massive increase in connected devices over time, going beyond what was assumed could be handled by data communication networks in the near future. For this operation, a solution to combat this substantial growth that can be maintained through a single point is essential. Such a solution must be able to provide services for futuristic needs and demands as well as providing ease of network management. SDN is the key solution to the aforementioned problems. Through the concept of software-defined

networking, network programmability is enhanced and network elements can be remotely managed from a centralized controller.

2.7 Conclusion

In this chapter, we have presented the controller placement problem of SDN controller and all related works to this problem.

We also mentioned the Mathematical Model of CPP and all the different models CPP associated metrics. In the next chapter, we will present and implement one of SDN Challenges.

Chapitre 3

Comparative analysis between works on CPP

3.1 Introduction

By decoupling control plane and data plane, Software-Defined Networking (SDN) approach simplifies network management and speeds up network innovations. These benefits have led not only to prototypes, but also real SDN deployments.

For wide-area SDN deployments, multiple controllers are often required, and the placement of these controllers becomes a particularly important task in the SDN context.

Controller placement is an important problem in software defined networks. Previously, some works addressed the problem by taking into account primary factors, such as latency ,load balance and propagation delay of control paths and capacity of controllers.

In this context, and in order to assist researchers and professionals in these areas, this chapter presents a comparative analysis between works on CPP , In a first phase, we have compared and classified all the works according to their objective, we have selected some work and defined all the characteristics of each work.

3.2 CONTROLLER PLACEMENT PROBLEM

The CPP is a fundamental research problem in SDN. The CPP has come into the picture since 2012. The CPP is not an issue for small- and medium-sized networks because one controller is sufficient to manage these kinds of networks. But in the case of large-sized networks, there would be significant aspects to consider multiple controllers and how to place these controllers in the SDN-based infrastructure. The CPP is a well-known NP-

hard problem and is similar to the facility location problem [58] . Placing the controllers in any of the available locations would only increase the overhead delay of service. This would also lead to a decrease in overall performance of SDN-based infrastructure. The primary objective of the CPP is to place the optimum number of controllers on the best locations of SDN-based infrastructure to achieve better performance.

Figure 1 shows the overall classification of our state-of-the- art of CPP. The CPP is first classified into 2 sections : (1) UCPP and (2) CCPP. The UCPP is further classified into 3 subsections : (a) static traffic-aware UCPP, (b) fault-aware UCPP, and (c) network partitioning–based UCPP. The CCPP is divided into 4 subsections : (a) static traffic-aware CCPP, (b) dynamic traffic-aware CCPP, (c) fault-aware CCPP, and (d) network partitioning–based CCPP. Controller placement problem may be either uncapacitated or capacitated. Literally, uncapacitated means unlimited capacity. All controllers have unlimited capacity in UCPP whereas CCPP refers to different controllers having different capacity. The UCPP and CCPP can be considered as failure-free CPP (FFCPP) and fault-tolerant CPP (FTCPP). In FFCPP, failure scenarios are not considered while focusing on controller placement in the SDN, whereas FTCPP considers failure scenarios.[59]

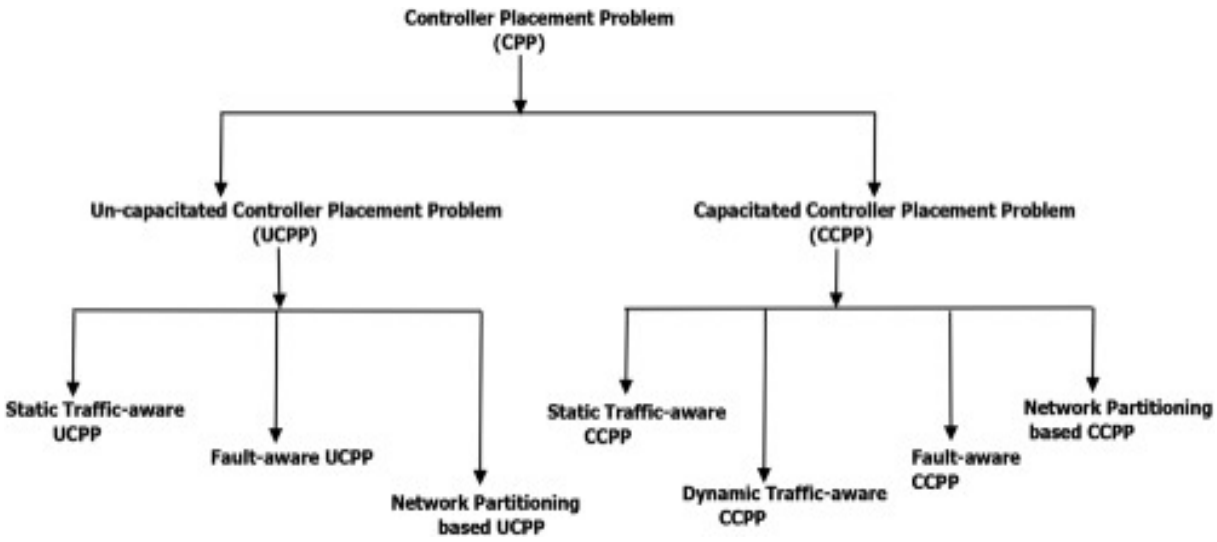

FIGURE 3.1 – Classification of controller placement problem [59]

3.2.1 Uncapacitated CPP

In UCPP, each SDN controller has unlimited capacity, and load on the controllers are ignored. Load on the controllers is a significant factor for the CPP. Some research works have not considered load and capacity of the controllers and assuming that the switches have fixed load. This kind of problem can be restricted to the static environment because the burden on the controllers cannot be distributed to other controllers, which are less loaded.

In 2012, Heller et al [60] have introduced SDN CPP. Here, authors formulated UCPP as a minimum k-median problem and minimum k-center problem to minimize the average and maximum (worst case) switch-to-controller latency, respectively.[59]

Static traffic-aware UCPP

Static traffic means fixed traffic. Here, traffic refers to the number of packets transferred between switches and controllers.

Heller et al [60] present the CPP as to find the set of k controllers and their respective placements so that the average latency between switches and controllers is minimized.

This optimization problem is known as the minimum k-median problem. Heller et al [60] also propose to solve the CPP by minimizing the maximum latency between switches and controllers.[59]

Fault-aware UCPP

A fault-tolerant system is one that functions correctly even in the face of faults. Ensuring the reliability a crucial aspect to achieving better performance of SDN. Fault-tolerant systems are mostly based on the concept of redundancy. For instance, a five nines system would statistically provide 99.999 % availability. Fault-tolerant mechanism removes the single point of failure. There are 3 key features of fault-tolerant systems : (1) replication "provides multiple identical instances of the same system or subsystem, directing tasks or requests to all of them in parallel, and chooses the correct result on the basis of a quorum," (2) redundancy "provides multiple identical instances of the same system and switching to one of the remaining instances in case failure occurs," and diversity "provides multiple different implementations of the same specification, and using them like replicated systems to cope with errors in a specific implementation.Zhang et al [61] take

the initial step to maximize the resilience, and this work assumes that switch to controller latency can be changed in case of failures.

In the year 2012, Yan-nan et al [62] have introduced reliability as a placement metric in the SDN environment where a number of controllers are deployed. According to authors, reliability of SDN control network increases the performance of network (network availability). In the large-sized network, the network may fail due to either controller or switch failure or link goes down. To prevent a network from such failure, the controller needs to be fault-tolerant.[59]

Network partitioning–based UCPP

The SDN network is partitioned into several small SDN network domains, and only one controller controls each domain.[63] Network partitioning can reduce the management complexity of resulting partitions of large networks. It could be related to various aspects of the network such as scalability, manageability, privacy, and deployment. Here, we discuss network partitioning based on only latency between switches, not on the traffic load of the switches. There are various techniques (local search, spectral clustering, density-based clustering, and multilevel partitioning) by which network partitioning can be done. [64] FlowVisor has partitioned the large-sized network into several SDN domains, and each domain has its controller. Partitioning the network is based on average-case latency and worst case latency.

Hu et al [65] extended a work by relaxing the previous assumptions (the control paths also traverse the failed physical component) and gave the detailed analysis of reliability for SDN.[59]

3.2.2 Capacitated CPP

In CCPP, load and capacity of controllers are considered during the placement of controllers. There is a chance of controller failure if load and capacity are not taken into account. In other words, we can say that some controllers may have overloaded and some controllers are negligibly loaded. Thus, load balancing is required for the load distribution to controllers having negligible load. Most of the researchers have considered capacity and load of the controller for placing the controllers in SDN. Load on the controller can be measured in terms of the number of switches associated with that controller and a number

of flow initiations per second.

Most popular NOX controller can handle around 30k flow initiations per second.[66] The load and capacity can be taken into account in the following ways during placement of controllers.[59]

Static traffic-aware CCPP

Most of the earlier research works ignore the load balancing between nodes and controllers, which is a major factor for the better performance of SDN. Generally, traffic (latency) between nodes and controllers, ie, static latency is predefined. The load on the controllers and intercontroller latency is considered by Hock et al [67]. This work is based on Pareto-based optimal placement (POCO toolset) in MATLAB to achieve optimal placement of controllers. The POCO provides better load balancing between switch and controllers and balanced placement of controllers in the SDN.[59]

Dynamic traffic-aware CCPP

In changing network traffic scenarios, if switch-to-controller latency is taken as constant, then the result is not good for load balancing among the controllers. To overcome this limitation, dynamic traffic load is considered while placing the controllers on the SDN. Various research works have been done in this category. Controller placement problem in dynamic environments is proposed by Dixit et al [68].

The authors proposed a mechanism for the dynamic environment so that switches can be dynamically assigned to the controller(s). Using this mechanism load can be distributed among the controllers and thereby reducing the chance of controller failure. The authors used Mininet [69] as the experimental test-bed and designed an elastic controller called EstiCon by adding an additional component in the SDN controller.[59]

Fault-aware CCPP

In this section, we analyze the solution of fault-aware CPP where capacity is a constraint. In 2014, Ros and Ruiz et al [70] introduced fault-tolerant CPP in SDN environment. To achieve five nines (99.999%) of reliability, the authors have found a solution where each node on an average needs to establish a connection with 2 controllers or at most 3 controllers. So that, in the case of failure, southbound reliability is assured. To achieve the

aforementioned reliability constraint, authors have proposed a heuristic algorithm where connectivity between controllers and switches act as a surrogate for reliability.

For finding connectivity, the authors find all disjoint paths that help to improve reliability by solving the max-flow problem. The authors provide a mathematical proof of the existence of some nondisjoint paths in the max-flow network, which results in degraded reliability and performance. The extended version of heuristic algorithm [71] forms all set of disjoint paths by analyzing the properties of topology.

Network partitioning–based CCPP

Initially, researchers reported that the response time of the controller depends only on switch-to-controller latency. According to recent claims, the response time of controller depends on both switch-to-controller latency and load on the controller. SDN performance depends on the time of reply between switch and controller, load balancing, and reliability. Controller performance is evaluated by Tootoochian et al [66] and the authors found that the controller handles a limited number of the service requests. If controller gets overloaded and there exists large propagation latency between switches, then partitioning comes into the picture.

In 2013, Bari et al [72] introduced dynamic controller provisioning problem (DCPP) in SDN for the first time. Multiple controllers are deployed in the large-sized network and simultaneously manage and control this network. Optimal DCPP is formulated by integer linear programming (ILP). They have proposed algorithms to minimize the communication overhead as well as flow setup time with the help of integer linear programming.[59]

3.3 Comparative analysis between works on CPP

3.3.1 General classification between works on ccp

In Table 3.1, we list down a general classification between previous works on cpp (target/ article(id))

Target	Article
Cost	Ying Hu et al [73]
	Reza Soleymanifar et al [74]
	Salahi and Hilaire [75]
	Rath et al [76]
Latency	Amit Dvir et al [82]
	V. Huang et al [85]
	P. Xiao [87]
	Jimenez et al [88]
	Yao et al [90]
Latency reliability	Yannan Hu et al [77]
	HU Yan-nan et al [78]
	Alireza Shirmarz et al [80].
Latency & Load balancing	Vivek Srivastava et al [83]
	V.Ahmadi et al [84]
	T. Yuan et al [86]
	Dixit et al [66]
	Rivera et al [91]
Latency & Resilience	T. Das et al [79]
	David Hock et al [81]

TABLE 3.1 – General classification between previous works on cpp

3.3.2 Advanced comparative analysis between works on ccp (latency)

In Table 3.2, we list down a advanced comparative analysis between works on ccp (latency))

Article(ref)	Year	Efficiency	Method	Languages	Static/Dynamic
Jimenez et al [88]	2014	WAN	K- Critcal algorithme	matlab languages	Dynamic
Yao et al [90]	2014	WAN	Linear relaxation	matlab languages	Dynamic
Xiao et al [87]	2016	WAN	K self-adaptive method	matlab languages	Static
Amit Dvir, et al [82]	2018	WWAN	K-Median algorithm.	matlab languages	Dynamic
Victoria Huang et al [85]	2020	WAN	Clustering-based Genetic Algorithm with Cooperative Clusters (CGA-CC)	matlab languages	Dynamic

TABLE 3.2 – Comparative analysis between works on ccp (latency)

3.3.3 Advanced comparative analysis between works on ccp (latency and load balancing)

In Table 3.3, we list down a advanced comparative analysis between works on ccp (latency and load balancing)

Article(ref)	Year	Efficiency	Method	Languages	Static/Dynamic
Dixit et al [66]	2013	LAN	ElastiCon	matlab languages	Dynamic
Rivera et al [91]	2015	LAN	Heuristic Algorithm	matlab languages	Dynamic
V.Ahmadi et al [84]	2018	WAN	Multi-start hybrid non-dominated sorting genetic algorithm (MHNSGA)	matlab languages	Dynamic
T.Yuan et al [86]	2018	WAN	Kuhn-Munkres algorithm	matlab languages	Static
V.Srivastava et al [83]	2021	LAN	Deep Reinforcement Learning strategy	C++/WILL API	Dynamic

TABLE 3.3 – Comparative analysis between works on ccp (latency and load balancing)

3.3.4 Advanced comparative analysis between works on ccp (Cost)

In Table 3.4, we list down a advanced comparative analysis between works on ccp (Cost)

Article(ref)	Year	Efficiency	Method	Languages	Static/Dynamic
Rath et al [76]	2014	LAN	No-0 sum game	matlab languages	Dynamic
Hock et al [92]	2014	LAN	POCO-PLC	matlab languages	Dynamic
Salahi et al [75]	2015	LAN	CPLEX	matlab languages	Dynamic

TABLE 3.4 – Comparative analysis between works on ccp (Cost)

3.3.5 Advanced comparative analysis between works on ccp (Latency reliability)

In Table 3.5, we list down a advanced comparative analysis between works on ccp (Latency reliability)

Article(ref)	Year	Efficiency	Method	Languages	Static/Dynamic
Hu et al [89]	2012	LAN	1-w greedy	matlab languages	Static
Hu et al [94]	2014	LAN	Random placement	matlab languages	Static
Liug et al [93]	2016	WAN	Greedy algorithms	matlab languages	Dynamic

TABLE 3.5 – Comparative analysis between works on ccp (latency and reliability)

3.4 Conclusion

Placement of controller(s) is an important aspect in the large-sized SDN. Efficient controller placement tries to improve the performance metrics like propagation latency, reliability, load distribution, failure resilience, and so on. CPP was introduced in 2012. In the last 5 years, while many researchers have focused on this problem, no related review has been conducted.

In this chapter, first, we have presented the brief introduction of the controller placement problem and . To ensure scalability and reliability, multiple controllers are required for large- scale SDN. Optimum placement of controllers is needed to improve the performance of SDN. Finally we have classified a Reviewed research work of CPP based on their solutions provided by recent researchers.

Chapitre 4

Implementation of one the challenges related to SDN

4.1 Introduction

SDN is physical separation of Control plane from Data plane and control plane is centralized to manage underlying infrastructure. Hence, the SDN permit network administrator to adjust wide traffic flow from centralized control console without having to touch Switches and Routers, and can provide the services to wherever they are needed in the network. As in SDN the control plane is disassociated from underlying forwarding plane, however, susceptible to many challenges like control Placement Problem.

In this chapter, we will present and implement the Pareto Optimal Controller Placement (poco) solution related to Control placement problem .

4.2 Mininet network emulator

There is need to model hosts, switches, links and SDN/OpenFlow controllers. Mininet [99] allows creating topologies of very large scale size up to thousands of nodes and perform test on them very easily. It has very simple command line tools and API. Mininet allows the user to easily create, customize, share and test SDN networks. (Fig. 1)

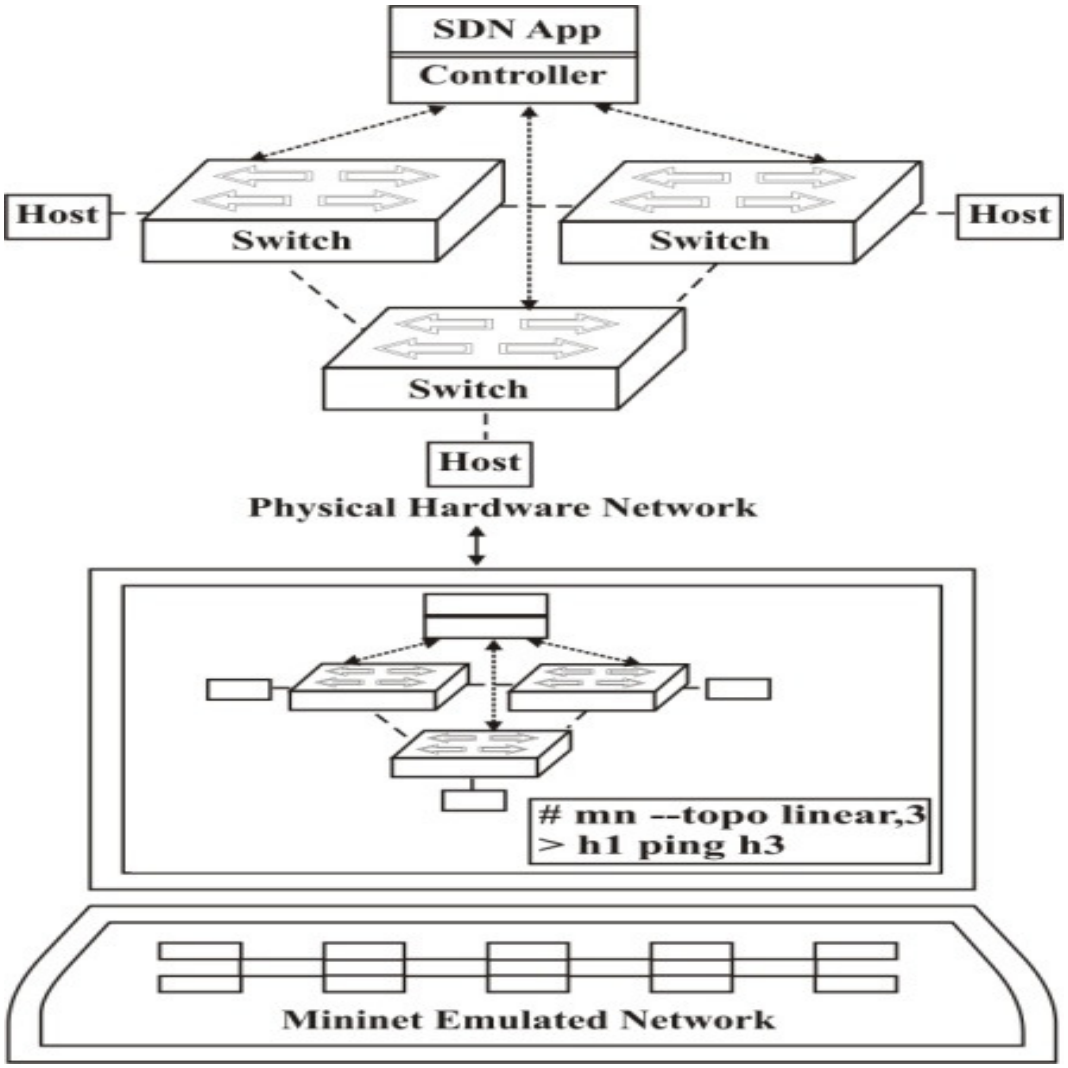

FIGURE 4.1 – Emulating Real Networks in Mininet [105]

Mininet is freely available open source software that emulates OpenFlow devices and SDN controllers. Mininet can simulates SDN networks, can run a controller for experiments [100]. It allows emulating real world network scenarios Couple of SDN controllers are included with in Mininet VM.

4.2.1 Working of Mininet

This network emulator permits you to generate a topology of network which constitutes the SDN controllers, OpenFlow switches, hosts which are virtual, and also various links in between them. It provides a very simple, inexpensive, robust networking test beds for developing and testing OpenFlow-based applications. Mininet is also used for creating complex network topologies. We can also create custom topologies according to our needs.

It also qualifies a simple Python interface for the purpose of creating a network and its various investigation processes. Creation, modification, and testing SDN-based networks are very easy using Mininet emulator tool [101].

Many supporting features of testbed and simulators are also used in Mininet. When the Mininet emulator tool is compared with real costly hardware testbeds available such as VINI, FIRE, GENI, and Emulab, it is quick, cost-free, and always available to reconfigure. Real, unmodified code is run by Mininet when it is compared with the simulators such as EstiNet and ns-3. Connection with realworld networks is easily achieved. Mininet takes only seconds to boot as compared to full system virtualization which takes a long time to boot. Mininet is used for creating large networks consisting of a large number of switches and hosts ; these large networks can also be implemented to real-world networks. It is very easy to install and provides more bandwidth to the network devices attached [102].

4.2.2 Mininet Commands

• $ sudo mn –h : This is used for displaying a help message describing Mininet's startup option where $ proceeds Linux command that should be typed in root shell prompt.

• $ sudo mn : The default topology is the minimal topology which is a very simple topology that contains OpenFlow switch and 2 hosts. It also creates links between switch and two hosts. This shows the following results :

creating controller
adding controller
adding hosts :
h1 h2
adding switches :
s1
adding links :
(h1,s1) (h2,s1)
configuring hosts
h1 h2
starting controller

c0

starting controller

s1

starting CLI Mininet¿

• Mininet> net : This command shows the following results :

mininet > net

h1 h1-eth0 :s1-eth1 h2

h2-eth0 :s1-eth2

s1 lo : s1-eth1 :h1-eth0

s1-eth2 :h2-eth0

• Mininet > dump

This will show switches and hosts listed.

mininet > dump

< Host h1 : h1-eth0 :10.0.0.1 pid=8683 >

< Host h2 : h2-eth0 :10.0.0.2 pid=8684 >

• mininet > nodes

available nodes are :

c0 h1 h2 s1

mininet > pingall

Ping : testing ping reachability

h1 -> h2 h2 -> h1

Results : 0% dropped (2/2 received) [106]

4.2.3 Different Topologies Used in Mininet

Mininet can support a number of topologies in which we can also create custom topology, some by default created topologies in Mininet such as single, reversed, minimal, linear, and tree which are discussed below.

1. **Minimal 2** hosts and 1 OpenFlow virtual switch are created in this topology. A link is created in between two hosts and switch which is virtual.
2. **Single** This topology contains 1 OpenFlow switch and n number of hosts, and a virtual link is also created between switch and n number of hosts

3. **Reversed** In this topology connection order is reversed as compared to single topology.
4. **Linear** In this topology there is a creation of link between the "n" number of OpenFlow switches and "n" number of hosts. A link between host and each switch and among the switches is also created.
5. **Tree** "n" levels like tree are contained in this topology and 2 hosts are connected [106].

4.2.4 CREATING CUSTOM TOPOLOGIES

Using Mininet, you can easily create a custom topologies [103]. For example creating custom topology having 2 switches and 5 hosts (Fig. 8) needs just writing a few lines of Python [104] code. You can also easily create very complex flexible, robust. You can also configured that topology based on the parameters that are to be pass to it, and reuse that topology for multiple experiments.

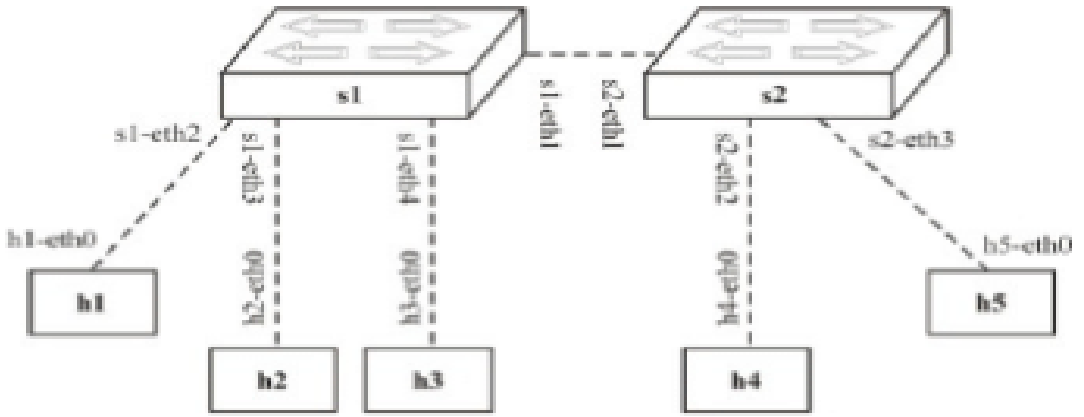

FIGURE 4.2 – Custom topology [105]

In the following Listing 1 contain Python code for creating custom topology having 2 switches and 5 hosts. This topology is run in Mininet by using following command.

Python CustomTopologyPerformance.py

```
from mininet.cli import CLI
from mininet.util import dumpNodeConnections
from mininet.node import CPULimitedHost
from mininet.link import TCLink
class SingleTopologyPerformance(Topo):
        def __init__(self, k=3):
                Topo.__init__(self)
                switch=self.addSwitch('s1')
                linkoptions=dict(bw=10, delay='10ms', max_queue_size=1000, use_htb=True)
                for h in range(k):
                        host=self.addHost('h%s' % (h+1), cpu=.4/k)
                        self.addLink(host, switch, **linkoptions)

def performanceTest():
        topo=SingleTopologyPerformance(k=5)
        net=Mininet(topo=topo, host=CPULimitedHost, link=TCLink)
        net.start()
        print "displaying host connection information"
        dumpNodeConnections(net.hosts)
        print "Testing network Connectivity"
        net.pingAll()
        print "Checking bandwidth between host h1 and h3"
        h1,h3 = net.get('h1','h3')
        net.iperf((h1,h3))
        net.stop()
if __name__ == '__main__':
        setLogLevel('info')
        performanceTest()
```

There are number of classes, functions, methods and variables in the Listing 1

1.Topo : It is a base class for Mininet topologies.

2. Add Host (name, cpu = f) : It is used for adding a host to the topology which contains two parameters. First parameter specifies the name of host and second specifies the fraction of overall system CPU resources that is to be allocated to the virtual host.

3. Add Switch() : It is used for adding a switch to the topology and returns the switch name, for example s1.

4. Add Link (node1, node2, **link options) : It is used for adding a bidirectional link which contains three parameters. The first, second parameter specify the host and switch name respectively and third parameter specify the dictionary that contain number of options such as bandwidth, delay and loss characteristics, with a maximum queue size.

5. start() : It is used for starting your network.

6. stop() : It is used for stopping your network.

7. Mininet : It is used as a main class to create and manage a network.

8. net. hosts : It is used to show all the hosts in network.

9. ping All () : This is used to check connectivity between all nodes.

10. Set Log Level : There are number of Log Level such as info, debug, and output. Info is recommended as it provides useful information.

11. Dump Node Connections () : Dumps connections to/from a set of nodes.

There are basically two classes available such as CPU Limited Host and TC Link that can be used for performance limiting and isolation. There are number of ways that these classes may be used, but simple way is to specify them as the default host and link classes to Mininet(), and then to apply the appropriate parameters in the topology.[105]

4.2.5 The SDN controllers and the Mininet

The SDN principle is the capability to control the packet forwarding process through a unique interface. The controller can center all communications with the programmable network elements and provide a single view of its state by isolating the details of each element.

One of the SDN advantages is exactly this centralized view of the network that makes possible to develop a detailed analysis and to decide how the system should operate.

The Mininet emulator implements connection between switches and different controllers, such as NOX and Floodlight for instance. This possibility allows developers interested in creating and testing controller resources to be able to use Mininet to perform their simulations.[107]

4.2.6 Uses of Mininet

• It is very fast. Creating and starting a basic network takes few seconds.

• You can build custom network topologies. The topology could contain a single switch or thousands of switches representing data center.

• You can run real source code. Any code that can be executed on Linux can also be run on Mininet.

• Mininet can be run on desktop, server, virtual machine (VM), laptop, and in the cloud.

• It is much uncomplicated to use. Mininet experiments can be run and performed by writing simple python scripts. [106]

4.2.7 Limitations of Mininet

• It forces some resource constraints when run on a single system.

• Applications or an OpenFlow switch that does not support Linux is not able to run in Mininet .

Open vSwitch, Indigo, Pantou, Pica8, and Softswitch are some of the mainly used software switches. Hardware switches which are available are of two types : OpenFlow enabled and native switches. Most SDN controllers such as NOX, POX, Beacon, OpenDaylight, and Maestro support OpenFlow version 1.0 [106].

4.3 Implementation of Pareto Optimal Controller Placement (POCO) solution

The POCO-toolset to compute resilient Pareto-based Optimal COntroller-placements is implemented in MATLAB. With an efficient combined use of CPU and RAM, it can evaluate the entire solution space of all controller placements even when resilience is considered. It calculates all considered objectives for each placement so that the best solutions according to the particular requirements can be selected from the entire range of possible placements. The actual decisions which of the multi-criteria optimization goals is most important can thus be postponed to after the optimization process[108].

A) Program Structur :

The implementation of POCO as a set of MATLAB scripts and functions makes POCO easy to use and extensible. An overview of the program structure as well as the contained functions is shown in Figure 4. The functions and scripts can be roughly classified into four groups : (1) input, (2) config, (3) calculation, and (4) output[108].

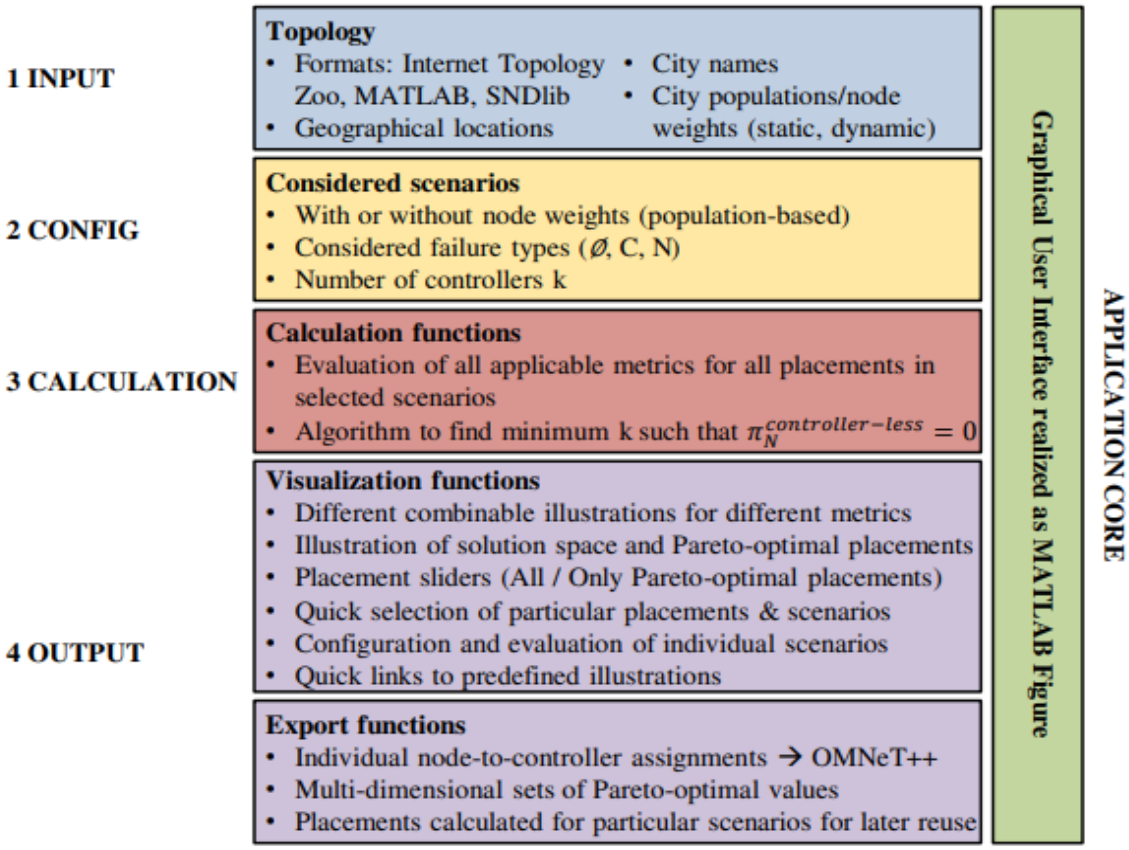

FIGURE 4.3 – Program structure of POCO [108]

B) Graphical User Interface :

To facilitate and improve the user experience of POCO, we now provide a GUI to control POCO and to evaluate and illustrate its results. Figure 2 shows a screenshot of the GUI. The upper part shows the network topology and can visualize different scenarios and options.

The lower part of the GUI allows to compare different placements according to multiple objective functions and to easily select a particular placement. This can be, e.g., the best placement according to a chosen metric, any arbitrary placement or one of the Pareto-optimal placements according to given metrics.

The lower part also shows an evaluation of all placements according to a selection of two arbitrary metrics as it has been done in [109] and highlights the Pareto-optimal results. All calculations necessary to evaluate different placements for different metrics are performed only once – triggered by a menu item – and the results are locally stored

for quick access to speed up the illustration and evaluation process [108].

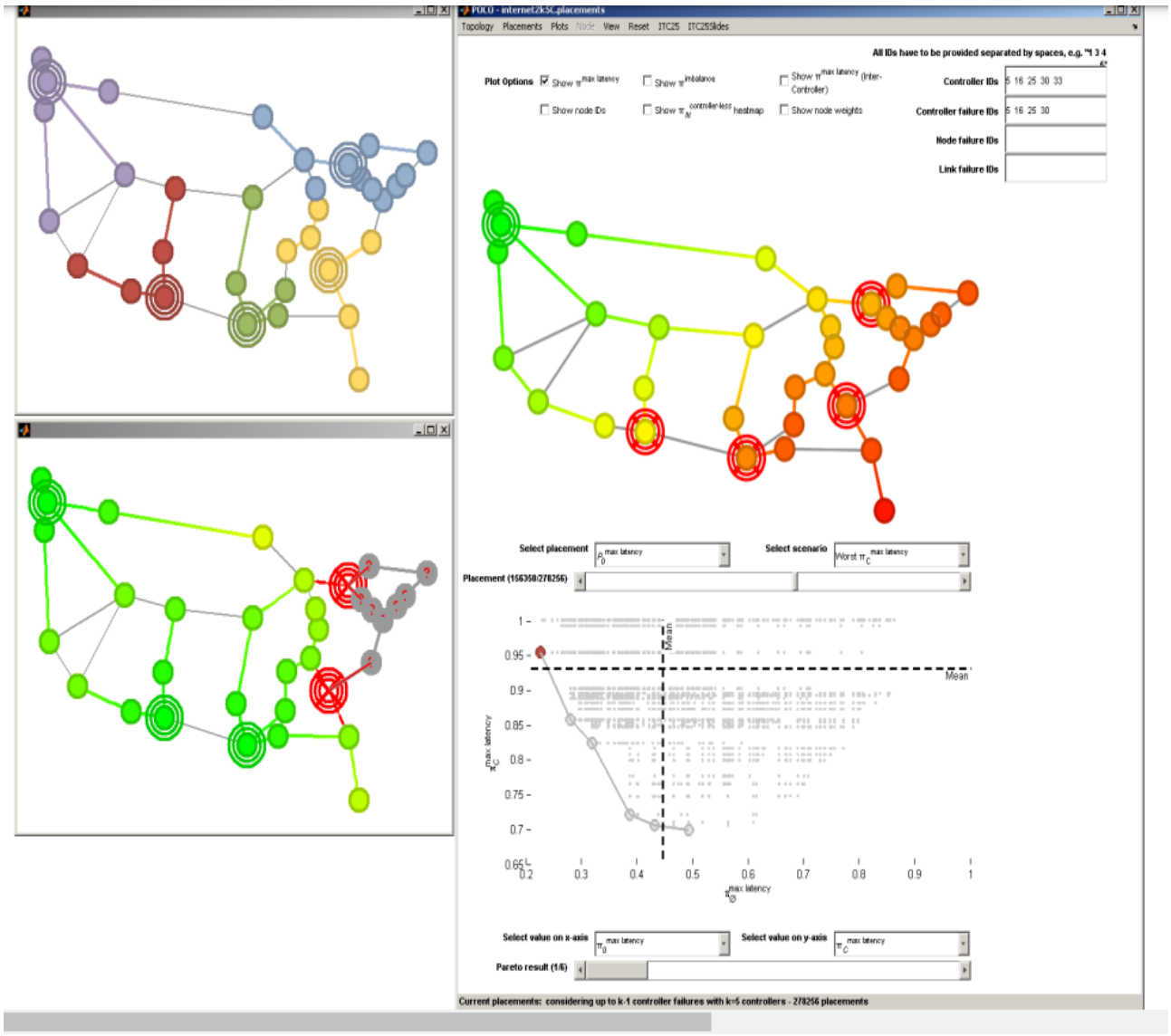

FIGURE 4.4 – Screenshot of the POCO GUI

```
2188  %--------------------------------------------------------------------
2189  % Opens a figure containing information about POCO
2190      function openAbout(hObject, eventdata)
2191          hAboutFig = figure('MenuBar','none','Toolbar','none','HandleVisibility','on','Name', 'About','NumberTitle','off','Position', [(screensize(3)/2 - 250), (screens
2192          imgsrc = strcat('file:/',pwd,'/images/pocologocollage.png');
2193          text = strcat('<html><center><img src="',imgsrc,'" border="0"><br><br><font family="verdana"><span style="font-size:20pt;">POCO: A framework for Pareto-Optimal
2194          je = javax.swing.JEditorPane( 'text/html', text );
2195          je.setEditable(false);
2196          [hcomponent, hcontainer] = javacomponent( je, [], hAboutFig );
2197          set( hcontainer, 'units', 'pixel', 'Position', [0 0 500 600] );
2198          hCloseButton = uicontrol('Parent',hAboutFig,'Units','pixel','Position',[200 25 100 30],'HandleVisibility','on','Visible','on','String','Ok','Style','pushbutton
2199      end
2200
2201
2202  %--------------------------------------------------------------------
2203  % Closes the GUI after asking for permission
2204      function closeGUI(hObject, eventdata)
2205          decision = questdlg('Do you really want to close POCO?','Exit POCO','Yes','No','No');
2206          if strcmp(decision,'Yes')
2207              if ishandle(hMainFigure)
2208                  delete(hMainFigure);
2209              end
2210              if ishandle(hFigurePLC)
2211                  delete(hFigurePLC);
2212              end
2213              if ishandle(inputFig)
2214                  delete(inputFig);
2215              end
2216          end
2217      end
2218
2219
2220      function closePLCFig(hObject, eventdata)
2221          stop(plcPlotTimer);
2222          stop(plcCalcTimer);
```

FIGURE 4.5 – Part code of The POCO GU

4.4 SDN-Controller-Placement using Emulation, case study : Classical "unsupervised" machine learning algorithms (Silhouette and Gap Statistic

This algorithm is used to find the number and the location of controllers in a multi-controllers architecture for SDN networks. It applies a machine learning algorithm named Silhouette and Gap Statistic.These algorithms are classically used to analyze cluster quality through the metric of minimum distances between data points. In the context of controller placement, we leverage these algorithms to find the number of controllers that minimizes overall network propagation latency (i.e. switch-to-switch latency). To find the

best locations for these controllers, we extend a facility location algorithm called Partition Around Medoids algorithm (PAM), with propagation latency (i.e. controller-to-switch latency) as our main objective function. The source code for this part of the experiment is in the folder named Controller Placement.tar.gz.

The algorithm use controller-to-node latency (propagation + queuing +processing latency) as a key performance indicator.

Python libraries used in this project

-igraph : this is a library collection for creating and manipulating graphs and analyzing networks

-matplotlib :a Python plotting library

-numpy : a library that allows manipulation of large multi-dimensional arrays and matrices

See Annex1 for Python classes used in this project.

4.5 Conculsion

With the introduction of Software Defined Networking (SDN), the concept of an external and optionally centralized network control plane, i.e. controller, is drawing the attention of researchers and industry. A particularly important task in the SDN context is the placement of such external resources in the network.

In this chapter, we have introduced a resilient Pareto-based Optimal COntroller placement (POCO) framework that provides the operator of a network with all Pareto-optimal placements.

Conclusion générale

Even that SDN is recognized today as an architecture that opens the network to applications. But We are still far from the initial rigid definition in which it was only a question of dissociating the control and data planes. Openflow is therefore only one component of SDN and the market today seems to be thinking more about solutions that effectively simplify programming and infrastructure.

In this work we have compared the research works on Controller placement problem (CPP). Also we have reimplemented some algorithms already proposed.

In future work, the optimization of several objectives will be considered for the controller placement. It would be interesting to study the issues related to relocating controllers, adding / removing controllers, the challenges of migrating switches and also studying issues related to mobility and CPP for other areas of the business. network such as IoT and sensor networks.

Bibliographie

[1] Fei Hu - Network Innovation through OpenFlow and SDN Principles and Design-CRC Press (2014) page 3.

[2] https ://searchnetworking.techtarget.com/definition/software-defined-networking-SDN

[3] Monaco, M., Michel, O., Keller, E. (2013, November). Applying operating system principles to SDN controller design. In Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks (p. 2). New York : ACM.

[4] Bianco, A., Birke, R., Giraudo, L., Palacin, M. (2010, May). OpenFlow switching : Data plane performance. In 2010 IEEE International Conference on Communications (ICC) (pp. 1–5). Piscataway : IEEE.

[5] Xia, W., Wen, Y., Foh, C. H., Niyato, D., Xie, H. (2015). A survey on software-defined networking. IEEE Communications Surveys and Tutorials, 17(1), 27–51.

[6] Wickboldt, J. A., De Jesus, W. P., Isolani, P. H., Both, C. B., Rochol, J., Granville, L. Z. (2015). Software-defined networking : Management requirements and challenges. IEEE Communications Magazine, 53(1), 278–285.

[7] Jarschel, M., Zinner, T., Hoßfeld, T., Tran-Gia, P., Kellerer, W. (2014). Interfaces, attributes, and use cases : A compass for SDN. IEEE Communications Magazine, 52(6), 210–217.

[8] https ://www.ibm.com/cloud/blog/software-defined-networking

[9] Dixit, A., Hao, F., Mukherjee, S., Lakshman, T. V., Kompella, R. (2013, August). Towards an elastic distributed SDN controller. ACM SIGCOMM Computer Communication Review, 43(4),7–12

[10] Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., Smeliansky, R. (2013).

Advanced study of SDN/OpenFlow controllers. In Proceedings of the 9th Central and Eastern European Software Engineering Conference in Russia (p. 1). New York : ACM.

[11] Tootoonchian, A., Ganjali, Y. (2010). Hyperflow : A distributed control plane for openflow. In Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking (p. 3). Berkeley : USENIX Association.

[12] Fernandez, M. P. (2013b). Comparing OpenFlow controller paradigms scalability : Reactive and proactive. In 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA) (pp. 1009–1016). Piscataway : IEEE.

[13] Brij Gupta, Gregorio Martinez Perez, Dharma P. Agrawal, Deepak Gupta - Handbook Of Computer Networks And Cyber Security$_{P}principlesAndParadigms-Springer(p325)$

[14] https ://www.nojitter.com/4-challenges-lying-wait-sdn

[15] https ://onlinelibrary.wiley.com/doi/10.1002/net.22016

[16] https ://ieeexplore.ieee.org/document/8053474

[17] Proceedings of the International Conference on Industrial Engineering and Operations Management Washington DC, USA, September 27-29, 2018 (p 1680)

[18] https ://ictexpertsluxembourg.lu/ict-cloud/sdn-vs-nfv-differences/

[19] https ://link.springer.com/article/10.1007/s12652-020-02754-w

[20] T. Das, V. Sridharan and M. Gurusamy, quot ;A Survey on Controller Placement in SDN,quot ; in IEEE Communications Surveys amp ; Tutorials, vol. 22, no. 1, pp. 472-503, Firstquarter 2020, doi : 10.1109/COMST.2019.2935453.

[21] A. Dvir, Y. Haddad, and A. Zilberman, "Wireless controller placement problem," in Consumer Communications Networking Conference (CCNC), 2018 15th IEEE Annual. IEEE, 2018, pp. 1–4.

[22] thisis Afrim Sallahi, Optimal placement of controllers in software defined networks, Ottawa-Carleton Institute, May.2014.

[23] thisis Mohammad Ashrafi Habibadi, controller placement in software defined networking, ALGARVE University, 2018.

[24] D. Kreutz, et al., "Software-Defined Networking : A Comprehensive Survey," PROCEEDINGS OF THE IEEE, vol. 103, pp. 1476, JAN 2015.

[25] D. Zeng, C. Teng, L. Gu, H. Yao, and Q. Liang, "Flow setup time aware minimum cost switch-controller association in software-defined networks," in Heterogeneous Networking for Quality, Reliability, Security and Robustness (QSHINE), 2015 11th International

Conference on. IEEE, 2015, pp. 259–264.

[26] J. Lu, Z. Zhang, T. Hu, P. Yi and J. Lan, A Survey of Controller Placement Problem in Software-Defined Networking, in IEEE Access, vol. 7, pp. 24290-24307, 2019, doi 10.1109ACCESS.2019.2893283.

[27] Abha Kumari et Ashok Singh Sairam, A Survey of Controller Placement Problem in Software Defined Networks, 2019.

[28] Bala Prakasa Rao Killi et al., Controller placement in software defined networks : A Comprehensive survey, Computer Networks 163 (2019) 106883, 2019.

[29] Ashutosh Kumar Singh and Shashank Srivastava, "A survey and classification of controller placement problem in SDN", wileyonlinelibrary.com/journal/nem, DOI : 10.1002/nem.2018.

[30] M. T. I. ul Huque, G. Jourjon, and V. Gramoli, "Revisiting the controller placement problem," in Proc. IEEE 40th Conf. Local Comput. Netw. (LCN), 2015, pp. 450–453

[31] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli, "Large-scale dynamic controller placement," IEEE Trans. Netw. Service Manag., vol. 14, no. 1, pp. 63–76, Mar. 2017.

[32] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu, "The SDN controller placement problem for WAN," in Proc. IEEE/CIC Int. Conf. Commun. China (ICCC), Oct. 2014, pp. 220–224.

[33] H. Kuang, Y. Qiu, R. Li, and X. Liu, "A hierarchical K-Means algorithm for controller placement in SDN-based WAN architecture," in Proc. 10th Int. Conf. Measuring Technol. Mechatronics Autom. (ICMTMA), Feb. 2018, pp. 263–267.

[34] B. P. R. Killi and S. V. Rao, "Optimal model for failure foresight capacitated controller placement in software-defined networks," IEEE Commun. Lett., vol. 20, no. 6, pp. 1108–1111, Jun. 2016.

[35] P. Xiao, Z.-Y. Li, S. Guo, H. Qi, W.-Y. Qu, and H.-S. Yu, "A k self-adaptive SDN controller placement for wide area networks," Frontiers Inf. Technol. Electron. Eng., vol. 17, no. 7, pp. 620–633, Jul. 2016.

[36] J. Yu, Y. Wang, K. Pei, S. Zhang, and J. Li, "A load balancing mechanism for multiple SDN controllers based on load informing strategy," in Proc. Netw. Oper. Manage. Symp., Oct. 2016, pp. 1–6.

[37] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller

placement in software defined wide area networks," IEEE Trans. Netw. Service Manag., vol. 15, no. 1, pp. 344–355, Mar. 2018.

[38] Y. Hu, W. Wang, X. Gong, X. Que, S. Cheng, On reliability-optimized controller placement for software-defined networks, China Commun. 11 (2014) 38–54.

[39] B. Liu, B. Wang, and X. Xi, "Heuristics for SDN controller deployment using community detection algorithm," in Proc. 7th IEEE Int. Conf. Softw. Eng. Service Sci. (ICSESS), 2016, pp. 253–258.

[40] K. S. Sahoo, B. Sahoo, R. Dash, and M. Tiwary, "Solving multi-controller placement problem in software defined network," in Proc. IEEE Int. Conf. Inf. Technol. (ICIT), 2016, pp. 188–192.

[41] Y. Jimenez, C. Cervello-Pastor, and A. J. Garcia, "On the controller placement for designing a distributed SDN control layer," in Proc. IFIP Netw. Conf., Jun. 2014, pp. 1–9.

[42] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks," IEEE Trans. Netw. Service Manage., vol. 15, no. 1, pp. 344–355, Mar. 2018.

[43] M. F. Bari et al., "Dynamic controller provisioning in software defined networks," in Proc. IEEE 9th Int. Conf. Netw. Service Manag. (CNSM), 2013, pp. 18–25.

[44] K. S. K. Liyanage, M. Ma, and P. H. J. Chong, "Controller placement optimization in hierarchical distributed software defined vehicular networks," Comput. Netw., vol. 135, pp. 226–239, Apr. 2018.

[45] D. P. Venmani and Y. Gourhant, "Cross-control : A scalable multitopology fault restoration mechanism using logically centralized controllers," in Proc. IEEE 15th Int. Conf. High Perform. Switch. Routing (HPSR), 2014, pp. 57–63.

[46] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for software-defined networks," in Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM), May 2013, pp. 672–675.

[47] Abdelrahman Abuarqoub, A Review of the Control Plane Scalability Approaches in Software Defined Networking,Future internet, 2020.

[48] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li, and T. Li, "Density cluster based approach for controller placement problem in large-scale software defined networkings," Comput. Netw., vol. 112, pp. 24–35, Jan. 2017.

[49] Y. Zhang, N. Beheshti, and M. Tatipamula, "On resilience of splitarchitecture networks," in Proc. IEEE Glob. Telecommun. Conf. (GLOBECOM), 2011, pp. 1–6.

[50] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "Efficient controller placement and reelection mechanism in distributed control system for software defined wireless sensor networks," Trans. Emerg. Telecommun. Technol., vol. 30, no. 6, p. e3588, Jun. 2019.

[51] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li, and T. Li, "Density cluster based approach for controller placement problem in large-scale software defined networkings," Comput. Netw., vol. 112, pp. 24–35, Jan. 2017.

[52] J. Liu, J. Liu, and R. Xie, "Reliability-based controller placement algorithm in software defined networking," Comput. Sci. Inf. Syst., vol. 13, no. 2, pp. 547–560, 2016.

[53] Brandon Heller et al., The controller placement problem. SIGCOMM Comput. Commun. Rev.42, 4 (October 2012), 473–478.

[54] D. Zeng, C. Teng, L. Gu, H. Yao, and Q. Liang, "Flow setup time aware minimum cost switch-controller association in software-defined networks," in Heterogeneous Networking for Quality, Reliability, Security and Robustness (QSHINE), 2015 11th International Conference on. IEEE, 2015, pp. 259–264.

[55] H. K. Rath, V. Revoori, S. Nadaf, and A. Simha, "Optimal controller placement in software defined networks (sdn) using a non-zero-sum game," in World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a. IEEE, 2014, pp. 1–6.

[56] M. Khorramizadeh and V. Ahmadi, "Capacity and load-aware software defined network controller placement in heterogeneous environments," Comput. Commun., vol. 129, pp. 226–247, Sep. 2018.

[57] https ://ieeexplore.ieee.org/document/8203978

[58] Arya V, Garg N, Khandekar R, Meyerson A, Munagala K, Pandit V. Local search heuristics for k-median and facility location problems. SIAM J Comput. 2004 ;33(3) :544-562.

[59] A survey and classification of controller placement problem in SDN. International Journal of Network Management, 28(3), e2018. doi :10.1002/nem.2018.

[60] Heller B, Sherwood R, McKeown N. The controller placement problem. In : Proceedings of the First Workshop on Hot Topics in Software Defined Networks. Helsinki, Finland : ACM ; 2012 :7-12.

[61] Zhang Y, Beheshti N, Tatipamula M. On resilience of split-architecture networks. In : Proceedings of the International Conference on Global Telecommunications (GLOBECOM'11). Kathmandu, Nepal : IEEE ; 2011 :1-6

[62] Hu Y, Wendong W, Gon X, Que X, Shiduan C. On the placement of controllers in software-defined networks. The J China Univ Posts Telecommun. 2012 ;19 :92171-97.

[63] Swamy C, Shmoys DB. Fault-tolerant facility location. ACM Trans Algorithms (TALG). 2008 ;4(4) :51 :1-51 :27.

[64] Zhao Z, Wu B. Scalable sdn architecture with distributed placement of controllers for wan. Concurr Comput : Pract Experience. 2017 ;29(16) :1-9.

[65] Hu Y, Wendong W, Gon X, Que X, Shiduan C. On reliability-optimized controller placement for software-defined networks. China Commun. 2014 ;11(2) :38-54.

[66] Tootoonchian A, Gorbunov S, Ganjali Y, Casado M, Sherwood R. On controller performance in software-defined networks. Hot-ICE. 2012 ;12 :1-6.

[67] Hock D, Hartmann M, Gebert S, Jarschel M, Zinner T, Tran-Gia P. Pareto-optimal resilient controller placement in sdn-based core networks. In : Proceedings of the 25th International Teletraffic Congress (ITC'13) IEEE ; 2013 ; Shanghai, China :1-9.

[68] Dixit A, Hao F, Mukherjee S, Lakshman TV, Kompella R. Towards an elastic distributed sdn controller. In : ACM SIGCOMM Computer Communication Review. Hong Kong, China : ACM ; 2013 : Vol. 43 :7-12.

[69] De Oliveira RLS, Shinoda AA, Schweitzer CM, Ligia PLR. Using mininet for emulation and prototyping software-defined networks. In : Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on IEEE ; 2014 ; Bogota (Colombia) :1-6.

[70] Ros FJ, Ruiz PM. Five nines of southbound reliability in software-defined networks. In : Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking. Chicago, Illinois, USA : ACM ; 2014 :31-36.

[71] Ros FJ, Ruiz PM. On reliable controller placements in software-defined networks. Comput Commun. 2016 ;77 :41-51.

[72] Bari MF, Roy AR, Chowdhury SR, et al.. Dynamic controller provisioning in software defined networks. In : Proceedings of the 9th International Conference on Network and Service Management (CNSM'13) IEEE ; 2013 ; Zürich, Switzerland :18-25.

[73] HU et al. : ENERGY-AWARE CONTROLLER PLACEMENT PROBLEM IN SOFTWARE DEFINED NETWORKS.

[74] Reza Soleymanifar et al. Clustering Approach to Edge Controller Placement in Software-Defined Networks with Cost Balancing, 2020.

[75] Sallahi A, St-Hilaire M. Optimal model for the controller placement problem in software defined networks. IEEE Commun Lett. 2015 ;19(1) :30-33.

[76] Rath HK, Revoori V, Nadaf SM, Simha A. Optimal controller placement in software defined networks (SDN) using a non-zero-sum game. In : Proceedings of the 15th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'14) IEEE ; 2014 ; Sydney, NSW :1-6.

[77] Y. HU,Reliability aware controller placement for Software Defined Networks, 2013.

[78] Y. HU et al. On the placement of controllers in software-defined networks, hu2012.

[79] T. Das and M. Gurusmy, Multi-Objective Control Plane Dimensioning in Hybrid SDN Legacy Networks.

[80] Shirmarz-Ghaffari2021 Article TaxonomyOfControllerPlacementP.

[81] David Hock, Matthias Hartmann, Steffen Gebert, Michael Jarschel, Thomas Zinner, Phuoc Tran-Gia University of Wurzburg, Institute of Computer Science, W urzburg, Germany / Pareto-Optimal Resilient Controller Placement in SDN-based Core Networks.

[82] Amit Dvir, Yoram Haddad,, Aviram Zilberman Jerusalem College of Technology, Jerusalem /// Wireless Controller Placement Problem.

[83] Vivek Srivastava, 2021, Machine intelligence approach To solve load balancing problem with high quality of service performance for multi-controller based Software Defined Network.

[84] V.Ahmadi et al., An adaptive heuristic for multi-objective controller placement in software-defined networks, 2018.

[85] V. Huang et al., A Scalable Approach to SDN Control Plane Management High Utilization Comes With Low$_L atency$, 2020.

[86] T. Yuan et al., Balance-Based SDN Controller Placement and Assignment with Minimum Weight Matching, 2018.

[87] P. Xiao, 2016 Article AKSelf adaptiveSDNControllerPl.

[88] Jimenez Y, Cervello-Pastor C, Garcia AJ. On the controller placement for designing a distributed sdn control layer. In : Proceedings of the IFIP Networking Conference. Trondheim, Norway : IEEE ; 2014 :1-9.

[89] Hu Y, Wendong W, Gon X, Que X, Shiduan C. On the placement of controllers

in software-defined networks. The J China Univ Posts Telecommun. 2012 ;19 :92171-97.

[90] Yao G, Bi J, Li Y, Guo L. On the capacitated controller placement problem in software defined networks. IEEE Commun Lett. 2014 ;18(8) :1339-1342.

[91] Ruiz-Rivera A, Chin K, Soh S. Greco : an energy aware controller association algorithm for software defined networks. IEEE Commun Lett. 2015 ;19(4) :541-544.

[92] Hock D, Hartmann M, Gebert S, Jarschel M, Zinner T, Tran-Gia P. Poco-plc : Enabling dynamic Pareto-optimal resilient controller placement in sdn networks. In : Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS'14). Toronto, ON, Canada : IEEE ; 2014 :115-116.

[93] Liug J, Liu J, Xie R. Reliability-based controller placement algorithm in software defined networking. Comput Sci Inf Syst. 2016 ;13(2) :547-560.

[94] Hu Y, Wendong W, Gon X, Que X, Shiduan C. Reliability-aware controller placement for software-defined networks. In : Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM'13). Ghent, Belgium : IEEE ; 2013 :672-675.

[95] https ://www.researchgate.net/figure/The-OpenFlow-protocol$_fig4_258286578$

[96] http ://www.cjcheema.com/2018/05/what-is-openflow-how-it-is-beneficial-in-sdn/

[97] https ://www.researchgate.net/figure/Comparison-among-the-controllers$_fig2_265845342$

[98] https ://www.redeemsystems.com/sdnnfv.php

[99] Handigol, Nikhil, Brandon Heller, Vimal kumar, Jeya kumar, Bob Lantz, and Nick McKeown. "Reproducible network experiments using container-based emulation." In Proceedingsof the 8th international conference on Emerging networking experiments and technologies, pp. 253–264. ACM, 2012.

[100] Lantz, Bob, Brandon Heller, and Nick McKeown. "A network in a laptop : rapid prototyping for software-defined networks." In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, pp. 19. ACM, 2010.

[101] De Oliveira, R. L. S., Schweitzer, C. M., Shinoda, A. A., Prete, L. R. (2014, June). Using mininet for emulation and prototyping software-defined networks. In 2014 IEEE Colombian Conference on Communications and Computing (COLCOM) (pp. 1–6). Piscataway : IEEE

[102] Fontes, R. R., Afzal, S., Brito, S. H., Santos, M. A., Rothenberg, C. E. (2015, November). Mininet-WIFI : Emulating software-defined wireless networks. In 2015 11th

International Conference on Network and Service Management (CNSM) (pp. 384–389). Piscataway : IEEE.

[103] Mininet at https ://github.com/mininet/mininet/wiki/Introduction-to-mininet.

[104] Python at https ://www.python.org/.

[105] Japinder Singh et al, Mininet as Software Defined Networking Testing Platform.

[106] Brij B. Gupta et al, Handbook of Computer Networks and Cyber Security.

[107] Christiane Marie Schweitzer et al, Using Mininet for Emulation and Prototyping Software-Defined Networks.

[108] David Hock et al,POCO-Framework for Pareto-Optimal Resilient Controller Placement in SDN-based Core Networks.

[109] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. TranGia, "Pareto-Optimal Resilient Controller Placement in SDN-based Core Networks," in ITC, Shanghai, China, 2013.

Annexe 1 : Python classes used in this project

main.py

```
#!/usr/bin/python
#Author: Lusani Mamushiane
# This code does the following:
# Reads a topology gml file using igraph and networkx;
# Computes link latencies;
# Emulates the topology;
# Partitions the network into two domains and instantiate controllers;
# Finds optimal locations to place controllers using a ping procedure
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
import igraph as ig
import networkx as nx
import matplotlib.pyplot as plt
import haversine
import numpy as np
import haversine as hv

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')
    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='192.168.0.180',
                         protocol='tcp',
                         port=6633)
```

haversine.py

```
#!/usr/bin/env python
import math

def distance(source_destination):
    lat1, lon1 = source_destination[0]
    lat2, lon2 = source_destination[1]
    radius = 6371

    dlat = math.radians(lat2-lat1)
    dlon = math.radians(lon2-lon1)
    a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(lat1)) \
        * math.cos(math.radians(lat2)) * math.sin(dlon/2) * math.sin(dlon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    d = (0.5*radius * c)/(10**2) #calculate delay by taking the ratio of distance to speed

    return d
```

ملخص

تمثل الشبكات المعرفة بالبرمجيات تحولًا نموذجيًا نحو مستوى تحكم شبكة مركزي خارجي ومنطقي. تتمثل إحدى المهام المهمة بشكل خاص في معماريات الشبكات المعرفة بالبرمجيات في وضع وحدة التحكم ، وتحديد موضع عدد محدود من الموارد داخل الشبكة لتلبية المتطلبات المختلفة. تتراوح هذه المتطلبات من قيود زمن الوصول إلى تحمل الفشل وموازنة الحمل. في معظم السيناريوهات ، تتنافس بعض هذه الأهداف على الأقل ، وبالتالي لا يتوفر أفضل موضع واحد ويحتاج صانعو القرار إلى إيجاد مقايضة متوازنة. يأخذ هذا العمل بعين الاعتبار وضع وحدات التحكم في الشبكات المعرفة بالبرمجيات. الهدف هو العثور على العدد الأمثل ومواضع وحدات التحكم التي تقلل من زمن الانتقال بين المفاتيح وأجهزة التحكم.في هذه الأطروحة ، نقوم بإجراء تحليل مقارن بين الأعمال على مشكل مكان المراقب ، وكذلك تنفيذ بعض الخوارزميات المقترحة بالفعل.

الكلمات المفتاحية : مشكلة وضع جهاز التحكم، الشبكات المعرفة بالبرمجيات ، الشبكات الافتراضية ، تدفق مفتوح

Abstract

Software Defined Networking (SDN) marks a paradigm shift towards an externalized and logically centralized network control plane. A particularly important task in SDN architectures is that of controller placement, the positioning of a limited number of resources within a network to meet various requirements. These requirements range from latency constraints to failure tolerance and load balancing.In most scenarios, at least some of these objectives are competing, thus no single best placement is available and decision makers need to find a balanced trade-off. This work considers the placement of controllers for Software Defined Networks (SDN). The goal is to find the optimum number and placements of controllers that minimize latency between switches and controllers. In this thesis, we perform a comparative analysis between works on the CPP, as well as an implementation of some algorithms already proposed.

Key words : CPP, SDN, Network virtualization, OpenFlow...

Résumé

Le Software Defined Networking (SDN) marque un changement de paradigme vers un plan de contrôle de réseau externalisé et logiquement centralisé. Une tâche particulièrement importante dans les architectures SDN est celle du placement du contrôleur, le positionnement d'un nombre limité de ressources au sein d'un réseau pour répondre à diverses exigences. Ces exigences vont des contraintes de latence à la tolérance aux pannes et à l'équilibrage de charge. Dans la plupart des scénarios, au moins certains de ces objectifs sont en concurrence, donc aucun meilleur placement n'est disponible et les décideurs doivent trouver un compromis équilibré. Ce travail considère le placement des contrôleurs pour les réseaux définis par logiciel (SDN). L'objectif est de trouver le nombre et les emplacements optimaux de contrôleurs qui minimisent la latence entre les commutateurs et les contrôleurs.Dans cette thèse, nous effectuons une analyse comparative entre les travaux sur le CPP, ainsi qu'une implémentation de certains algorithmes déjà proposés.

Mots clés : Problème de placement du contrôleur, SDN, Virtualisation de réseau, fluxOuvert ...