ALGERIAN DEMOCRATIC AND POPULAR

REPUBLIC MINISTRY OF HIGHE REDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY KASDI MERBAH OUARGLA

FACULTY OF NEW INFORMATION AND COMMUNICATION TECHNOLOGIES

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

MASTER THESIS

Network Administration and Security

BY: AMMOUR MOHMMED CHIKH & DEBBAKH FADIA

THEME

PERFORMANCE EVALUATION OF SOFTWARE DEFINED –NETWOEK (SDN) CONTROLLER

Evaluation Date : 15/06/2023

<u>**COMMITTEE MEMBERS:**</u>

Mr. KAHLESSENANE FARES	**SUPERVISOR**	**UKM OUARGLA**
Mr. KHALDI AMINE	**COMMITTEE CHAIR**	**UKM OUARGLA**
Mr. MECHALIKH CHARAF EDDINE	**REVIEWER**	**UKM OUARGLA**

ACADEMIC YEAR: **2022/2023**

Acknowledgment

First and foremost, praises and thanks to the God, the Almighty, for His showers of blessings throughout my research work to complete the research successfully.

we also thank our parents, who encouraged and motivated us to reach this level of study, we would like to express our deep and sincere gratitude to our research supervisor

' KahleleSennan Farse'

Likewise, we extend our respectful thanks to the members of the jury, who have done the honor to participate in this jury and to examine this work.

A special thanks to our family members and friends for always standing by us. Finally, we would like to thank all those who have helped us from near or far during all our studies and in the preparation of this dissertation, our deep gratitude and respect.

Dedication

I dedicate this research to all those who have inspired and supported me throughout my academic journey.

To my mother ' Mounira ', my father ' Slimane', my sister ' Hana ' , who have been a source of love, support, and belief in my abilities. Your support has ignited my enthusiasm to pursue this research.

To my supervisor, ' KahleleSennan Farse' , thank you for your guidance, expertise, and valuable advice throughout this research, and for motivating us.

To my cousins 'Hanadi ', 'Abla', 'Asma'and 'Nerimane', thank you for the great moments we shared together.

To my friends and colleagues, thank you for your companionship and unwavering support. Your presence has made this journey more enjoyable, and your encouragement has lifted me during challenging times.

Thank you all

Debbakh Fadia.

إهداء

بسم الله الرحمن الرحيم

البداية الحمد لله حمدا كثيرا مباركا طيبا فيه الذي منحني القدرة والفرصة للوصول إلى هذا الإنجاز. بفضله، تجاوزت التحديات وتحققت أهدافي.

أود أن أعبِّر عن امتناني العميق لوالديّ، فهما كانا دعمًا لا يقدر بثمن طوال هذه الرحلة. بفضل حبهما وتشجيعهما، استطعت تحقيق حلمي والوصول إلى هذه اللحظة المهمة في حياتي.

أشكر أسرتي وأحبائي على وقوفهم إلى جانبي ودعمهم المتواصل. كانوا رمزًا للحب والسعادة في حياتي.

أيضًا، أود أن أعرب عن امتناني لأساتذتي ومعلميَّ، الذين ألهموني وقدموا لي المعرفة والإرشاد. بفضلهم، تعلمت وتطورت بشكل كبير

كما اريد ان أغتنم الفرصة للترحم على أستادي العزيزين باباحمو محمد الناصر و بن مير عبد القادر رحمة الله عليهم

كما أشكر كل من ساهم في رحلتي التعليمية بطرق مختلفة . شكرًا لكل من شجعني وساندني وألهمني في طريقي نحو النجاح.
وأخيرًا، إلى نفسي، أنا الذي واجهت التحديات وتجاوزت الصعاب. أشكرك على الإصرار والتفاني في تحقيق هذا الهدف
بفضل الله وبدعم الأهل والأحباب والمعلمين، أصبحت هذه المذكرة حقيقة. أدعو الله أن يجعلها خطوة أولى في رحلة مستقبلية مليئة بالتحقيقات والإنجازات.

بكل الشكر والامتنان،
محمد الشيخ عمور

Abstract

Traditional networks have relied on a distributed control plane, where decision-making and forwarding functions are tightly coupled within individual networking devices. However, this approach poses challenges in terms of network management, scalability, and fault tolerance .

From this context , the concept of Software-Defined Networking (SDN) networks emerged , which is that networking approach that separates the control and data planes to enable centralized management of network traffic and configurations. This allows network administrators to easily configure, manage and monitor network resources. One key protocol that enables dynamic routing decisions in SDN is OpenFlow **.**

Hence , this project proposes to evaluate the performance and fault tolerance capabilities of SDN controllers using Mininet, HPE-VAN SDN Controller, ONOS, and Opendaylight. The study will emulate SDN environments to test and evaluate SDN networks without disrupting existing network infrastructure. The project will investigate the impact of various network parameters, including topology, traffic patterns, and controller placement, on SDN performance. The findings of this study will help network designers and operators identify optimal configurations that maximize performance and fault tolerance in SDN networks. Overall, this project will provide insights that can inform the deployment and management of SDN networks.

keywords: SDN , Openflow ,opendaylight, Mininet, HPE-VAN SDN controller

ملخص

الشبكات التقليدية اعتمدت على طراز التحكم الموزع، حيث تكون وظائف اتخاذ القرار وإعادة التوجيه مرتبطة بشكل وثيق داخل أجهزة الشبكة الفردية. ومع ذلك، يواجه هذا النهج تحديات فيما يتعلق بإدارة الشبكة وقابلية التوسع وتحمل الأخطاء.
من هذا السياق، ظهر مفهوم الشبكات المعرفة بالبرمجيات (SDN)، وهي نهج الشبكات الذي يفصل طبقات التحكم والبيانات لتمكين الإدارة المركزية لحركة المرور وتكوينات الشبكة. يتيح ذلك لمسؤولي الشبكة تكوين وإدارة ومراقبة موارد الشبكة بسهولة. أحد البروتوكولات الرئيسية التي تمكّن اتخاذ قرارات التوجيه الديناميكي في SDN هو OpenFlow.

وبالتالي، يقترح هذا المشروع تقييم أداء التسامح مع الخطأ لشبكات المعرفة بالبرمجيات (SDN) باستخدام Mininet و HPE-VAN SDN Controller و Opendaylight. سيحاكي الدراسة بيئات SDN لاختبار وتقييم شبكات SDN دون التأثير على بنية الشبكة الحالية. سيستقصي المشروع تأثير مختلف المعلمات الشبكية، بما في ذلك الهيكل التوبولوجي، وأنماط حركة المرور، على أداء SDN. ستساعد نتائج هذه الدراسة مصممي الشبكات والمشغلين على تحديد التكوينات الأمثل التي تعزز الأداء وقدرة التحمّل من الأعطال في شبكات SDN. بشكل عام، سيوفر هذا المشروع نظيرات مفيدة يمكن أن تساعد في تنفيذ وادراة شبكات SDN .

الكلمات المفتاحية: الشبكات المعرفة بالبرمجيات ،مينينت، HPE-VAN SDN Controller، **Opendaylight**

Résumé:

Les réseaux traditionnels reposent sur un plan de contrôle distribué, où les fonctions de prise de décision et de transfert de données sont étroitement couplées au sein des dispositifs individuels du réseau. Cependant, cette approche présente des défis en termes de gestion du réseau, de scalabilité et de tolérance aux erreurs .

Dans ce contexte, le concept de réseaux de Software-Defined Networking (SDN) est apparu, qui est une approche de mise en réseau qui sépare les plans de contrôle et de données pour permettre la gestion centralisée du trafic réseau et des configurations. Cela permet aux administrateurs réseau de configurer, gérer et surveiller facilement les ressources réseau. Un protocole clé qui permet des décisions de routage dynamique dans SDN est OpenFlow **.**

Par conséquent, ce projet propose d'évaluer les performances et les capacités de tolérance aux pannes du contrôleur SDN en utilisant Mininet, HPE-VAN SDN Controller et Opendaylight. L'étude émulera des environnements SDN pour tester et évaluer les réseaux SDN sans perturber l'infrastructure réseau existante. Le projet examinera l'impact de divers paramètres réseau, notamment la topologie, les schémas de trafic et le placement du contrôleur, sur les performances du SDN. Les conclusions de cette étude aideront les concepteurs et les opérateurs de réseaux à identifier les configurations optimales qui maximisent les performances et la tolérance aux pannes dans les réseaux SDN. Dans l'ensemble, ce projet fournira des informations qui peuvent éclairer le déploiement et la gestion des réseaux SDN **.**

Les mots clés: SDN , Openflow ,opendaylight, Mininet, HPE-VAN SDN controller

LIST OF FIGURES

LIST OF TABLES

CONTENTS

List of keywords

OSPF : Open Shortest Path First .

RIP :Routing Information Protocol.

IGRP: Interior Gateway Routing Protocol.

BGP: Border Gateway protocol.

IS-IS: Intermediate System to Intermediate System.

LSAs : Link-State.

ONF: Open Networking Foundation.

OVSDB: OpenvSwitch Database.

SDN: Software Defined-Network .

ODL: Opendaylight

LISP: Locator / Identifier Separation.

ONOS: Open Network Operating System.

API : Application Programming Interface.

GENERAL INTRODUCTION

Software-Defined Networking (SDN) is an innovative approach to network architecture that aims to enhance network control and management by separating the control plane from the data plane. In traditional networks, the control plane, responsible for making decisions about how data packets are forwarded, resides within individual networking devices such as switches and routers. However, SDN centralizes the control plane in a software-based controller, allowing for more flexibility, programmability, and agility in managing network resources .

One of the key advantages of SDN is its ability to improve fault tolerance in network environments. Fault tolerance refers to a system's ability to continue operating properly even in the presence of component failures or network disruptions. By leveraging the centralized control plane, SDN enables efficient fault detection, isolation, and recovery mechanisms .

In an SDN architecture, the controller has a global view of the network, including information about network topology, traffic patterns, and device statuses. This visibility allows the controller to monitor the network continuously and detect faults or anomalies promptly. When a fault is detected, such as a link failure or a device malfunction, the controller can take immediate action to mitigate the impact and maintain network functionality .

SDN provides several fault tolerance mechanisms to address various types of failures. For instance, in the case of link failures, the controller can dynamically reroute traffic by updating the forwarding rules in the network devices to avoid the affected links. This rerouting can be based on pre-defined backup paths or computed in real-time by considering the current network conditions .

Moreover, SDN enables the implementation of proactive fault tolerance measures. The controller can employ techniques like redundancy and load balancing to distribute network traffic across multiple paths or devices. This redundancy ensures that if a failure occurs, alternative paths or devices are available to handle the traffic, preventing a single point of failure and maintaining uninterrupted connectivity .

Additionally, SDN allows for efficient network reconfiguration and resource allocation. In the event of a failure, the controller can dynamically reassign network resources, such as bandwidth or computing power, to compensate for the loss and maintain optimal performance. This adaptability and responsiveness contribute to the overall fault tolerance of the SDN environment .

Overall, SDN's centralized control plane and programmability enable effective fault tolerance mechanisms, including fault detection, isolation, recovery, proactive measures, and dynamic resource management. By leveraging these capabilities, SDN enhances network resilience, minimizes downtime, and improves the overall reliability of modern networking infrastructures .
In this project , we will conduct a comparative study between different SDN controllers. To accomplish this ,we have divided the into three chapter :

The first chapter, mention the significance of SDN (Software-Defined Networking), including its features and advantages, purpose, architectural design, and how to effectively perform routing in SDN.

Secondly, this chapter highlights the importance of fault tolerance in sdn .

The final chapter of this project will focus on the technical side of our topology. It will cover the design, implementation, and realization of our proposed topology .

Chapter 1

SDN NETWORk

1 Introduction

SDN is a networking architecture that separates the control plane from the data plane.
The SDN architecture has been adopted by several large companies and universities namely , Google and Stanford. On the other hand , manufacturers such as Cisco ,HP and Juniper now offer SDN solutions that make it possible to manage data centers. [1]
In SDN, the control plane is centralized and implemented as a software application running on a separate server called the controller. Routing in SDN involves determining the best path for network traffic based on topology and traffic load. The controller uses routing protocols, such as OSPF and BGP, to exchange routing information with network devices and calculate the optimal routes. The controller then programs the forwarding tables of network devices to ensure traffic is forwarded along the chosen path, which can be dynamically adjusted in response to network conditions .

1.1 SDN Definition

The term SDN was originally coined to represent the ideas and work around OpenFlow at Stanford university , Stanford ,CA,USA[2].

Software-Defined Networking (SDN) is a network architecture approach that separates the control plane, which makes decisions on how data should be transmitted, from the data plane, Which physically transmits the data.

The key idea behind SDN is to centralize the control of network devices and allow administrators to manage and configure the network through software, rather than manual configuration of individual devices.

This enables greater network flexibility , programmability, and automation, making it easier to implement new network services and change network behavior on the fly. Additionally, SDN can also help to improve network visibility and security, as well as reduce operational costs.

1.2 Objective of SDN:

Software-defined networking (SDN) is an approach to network management and control that emphasizes the centralization of network intelligence and the separation of the data plane (where data packets flow) from the control plane (where network policies are defined).

The objective of SDN is to make network management more flexible, agile, and scalable by enabling network administrators to programmatically control the behavior of the network using software, rather than configuring each network device manually.

This allows for more efficient network management, easier troubleshooting, and faster deployment of new services.

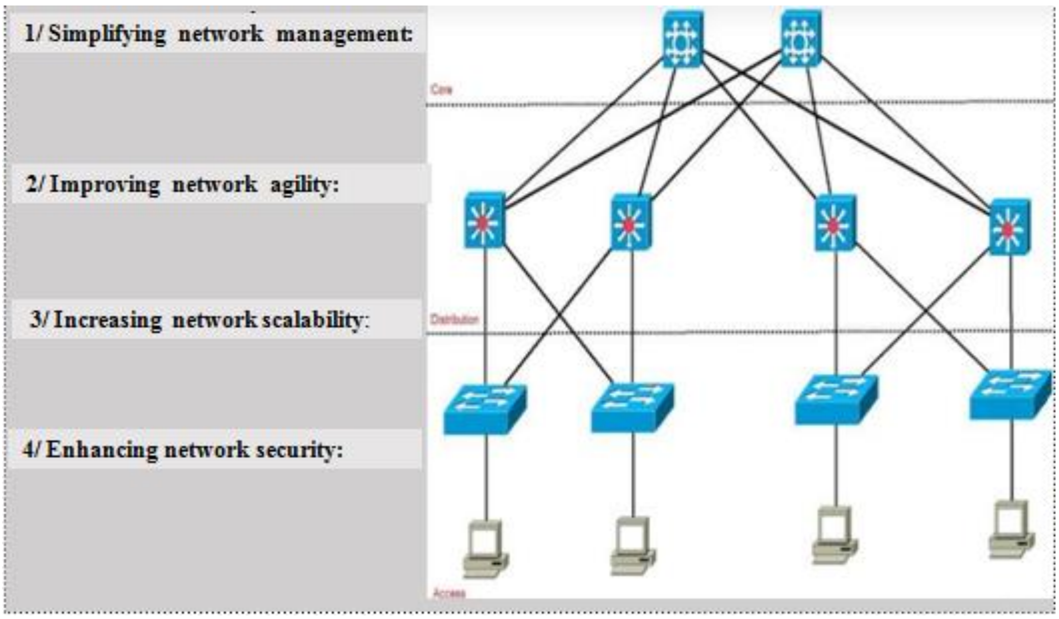

1/ Simplifying network management:
2/ Improving network agility:
3/ Increasing network scalability:
4/ Enhancing network security:
Core
Distribution
Access

Figure1. 1:The benefits of sdn.

1.2.1 Some specific objectives of SDN include:

a. ***Simplifying network management:*** SDN can make it easier to manage complex networks by providing a single point of control for configuring and monitoring network devices.

b. **Improving network agility:** By separating the control plane from the data plane, SDN enables network administrators to quickly adapt to changing network conditions and deploy new network services.

c. **Increasing network scalability:** SDN can help to scale networks more easily by allowing administrators to define and enforce network policies in a centralized manner, rather than configuring individual devices.

d. **Enhancing network security:** SDN can improve network security by enabling administrators to centrally monitor and control network traffic, and by providing fine-grained control over network policies.

Overall, the objective of SDN is to create a more programmable, dynamic, and cost-effective network architecture that can support the evolving needs of modern applications and services.

1.3 SDN Architecture :

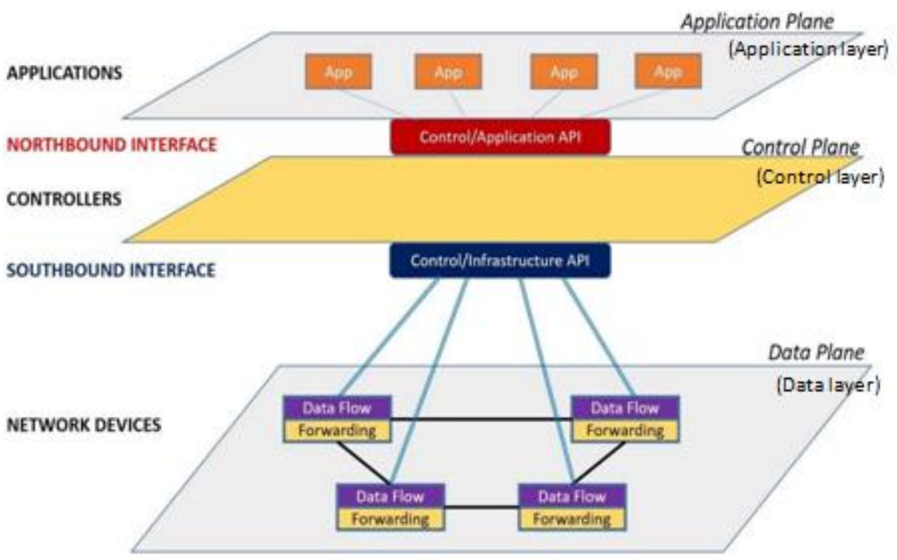


Figure1. 2:SDN Architecture .[3]

Software-defined networking (SDN) is an architecture that separates the network control plane from the data plane, allowing for centralized control of network infrastructure and automated network management.

The following is a detailed explanation of the components of an SDN architecture :

1.3.1 Application Layer:

The application layer provides a user interface for network administrators to manage the network . It includes network management tools, monitoring and reporting tools, and other software applications that control and monitor the network.

1.3.2 Control Layer:

The control layer is responsible for managing the network infrastructure , including switches, routers, and other network devices. It uses a controller, which is a centralized software component that communicates with the network devices and manages the flow of data through the network. The controller receives information from the network devices and sends instructions back to them, which allows for centralized control of the network .

1.3.3 Southbound APIs:

The southbound APIs are the interfaces that connect the controller to the network devices. These API sallow the controller to communicate with the devices, configure them, and collect information a bout the network. There are several southbound APIs, including OpenFlow , NETCONF, and OVSDB.

a. ***Openflow protocol :*** The Open Flow protocol is a communication protocol used to facilitate the management and Control of network traffic flows in a software- defined networking (SDN) environment. It was developed by the Open Networking Foundation (ONF) and is now widely used in many SDN deployments.

The OpenFlow protocol is designed to separate the control plane from the data plane in network switches and routers. The control plane is responsible for managing network traffic, while the data plane is responsible for forwarding data packets. By separating these two planes, the OpenFlow protocol enables network administrators to centrally manage and control the flow of traffic in the network.

OpenFlow uses a standardized set of instructions, called flow rules, to control how traffic is forwarded through the network. Flow rules are defined by the network administrator and are programmed in to the OpenFlow-enabled switches and routers. When a packet arrives at a switch or router, it is matched against the defined flow rules, and the appropriate action is taken based on the match.

The OpenFlow protocol is typically implemented using a centralized controller, which communicates with the OpenFlow-enabled switches and routers in the network. The controller provides a unified view of the network, allowing administrators to manage traffic flow, enforce policies, and respond to network events in a centralized and programmatic way:

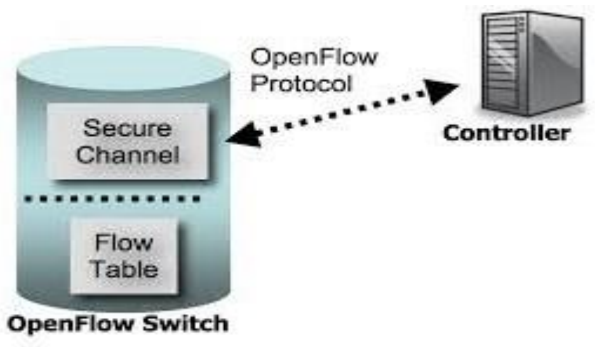


Figure1. 3:OpenFlow protocol . [4]

OpenFlow is a protocol that allows packets to be forwarded based on multiple criteria, such as IP address, MAC address, VLAN, and other addresses. It can be considered a reliable alternative to IP routing. Since its initial release, the OpenFlow protocol has undergone rapid improvements and updates, with numerous new versions introduced. Here are the versions of OpenFlow and their definitions :

- **OP V1.0:** The initial version of OpenFlow, which provided basic packet forwarding and modification capabilities.
- **OP V1.1:** Added support for Quality of Service (QoS) and Group Tables, which allowed for more flexible packet forwarding and prioritization.
- **OP V1.2:** Added support for IPv6, MPLS, and multiple tables, which enabled more advanced traffic engineering and management.
- **OP V1.3:** Introduced features such as flow statistics, packet-in packet-out messages, and support for multiple controller connections.
- **OP V1.4:** Added support for hybrid switching, where OpenFlow and traditional networking can coexist in the same network.
- **OP V1.5 :** Added support for advanced features such as multiple flows within a single packet and improved support for wireless networks.
- **OP V1.6 :** A working draft that aims to provide even greater flexibility and programmability, with features such as dynamic table resizing and support for more complex actions and conditions.

b. ***Openflow devices:*** OpenFlow devices are network devices that support the OpenFlow protocol. OpenFlow is a communication protocol that allows a centralized controller to manage and configure network devices, such as switches and routers, In a software-defined networking (SDN) environment.

An OpenFlow device, such as an OpenFlow-enabled switch ,is capable of forwarding packets based on instructions received from a controller. These instructions can include actions such as forwarding packets to a particular port, dropping packets, or modifying packet headers. In an SDN architecture, the controller has a global view of the network and can use this view to make intelligent decisions about how to forward traffic.

OpenFlow devices and the OpenFlow protocol enable network administrators to easily configure and manage complex networks, and to implement dynamic and flexible network policies. They are widely used in data centers, campus networks, and other large-scale networking environments.

c. ***OVSDB protocol :*** OpenvSwitch Database (OVSDB) protocol is a network management protocol used in Software-Defined Networking (SDN) architecture to manage and configure OpenvSwitch (OVS).SDN is a network architecture that separates the control plane and data plane, providing amore flexible and programmable approach to network management.

OVS is a software-based virtual switch used in virtualized environments, which provides the data Plane functionality in SDN. The OVSDB protocol provides a standard mechanism for managing and configuring OVS, enabling better integration with other network management tools and systems. The OVSDB protocol in SDN architecture operates on two main components, the OVSDB server and the OVSDB client.

- **OVSDB server:** The OVSDB server manages the OVS data base and provides configuration and management services to the OVSDB client. The OVSDB server communicates with the OVS using the OpenFlow protocol, which provides a standardized way of controlling the data plane.
- **OVSDB client:**

The OVSDB client is responsible for communicating with the OVSDB server to manage the OVS database. It sends queries to the OVSDB server to retrieve information from the database , and it sends configuration commands to modify the database's contents.

The OVSDB protocol is based on the JSON-RPC protocol, which operates over a secure transport raye such as SSL/TLS. It defines a set of operations that can be used by the client to query, modify, and delete the OVSDB database contents. These operations include "select,""insert," "update," and "delete."

The OVSDB protocol's schema defines the data model used by the OVSDB server, including information about the data types, tables, columns, and relationships between them.

The OVSDB database includes tables such as "Bridge,""Port,""Interface," and "Controller," which store configuration and operational data for the OVS switch.

The OVSDB protocol in SDN architecture allows for the creation, modification, and deletion of virtual switches, ports, and interfaces .It also allows for the creation of virtual networks ,which are useful in creating isolated network topologies for specific applications or tenants. The OVSDB protocol enables dynamic reconfiguration of the virtual network, making It possible to make changes to the network without disrupting its operation.

Overall, the OVSDB protocol is a critical component of SDN architecture ,providing a standardized mechanism for managing and configuring virtual switches in a virtualized environment. This enables network administrators to create, modify, and delete virtual switches and network configurations, making network management more efficient and flexible.

```
                          +---------------------- -+
                          |    Control &           |
                          |     Management         |
                          |      Cluster           |
                          +-----------------------+
                               | OVSDB          \
                                                 \ OpenFlow
                              | Mgmt              \
                              |                    \
+==============================================+
| +---------------+       +---------------+    |
| |               |       |               |    |
| | ovsdb-server  |-------| ovs-vswitchd  |    |
| |               |       |               |    |
  +---------------+       +---------------+    |
|                                 |            |
|                         +---------------     |
|                         |Forwarding Path     |
|                         +---------------     |
+==============================================+
```

Figure1. 4 :OVSDB Architecture.[5]

1.3.4 Data Layer:

The data layer is responsible for handling the actual data that flows through the network . It includes the network devices, such as switches and routers, which forward packets of data to their intended destinations. The data layer also includes the data plane, which is responsible for processing and forwarding data packets.

One way to implement the data layer in SDN is through the use of a table of flows. A table flows is a data structure used by SDN controllers to manage network traffic flows. It consists of a set of rules that define how network traffic should be processed by the switches in the network.

When a new flow is detected in the network, it is sent to the SDN controller, which consults the table flows to determine how the flow should be processed. The controller then adds a new rule to the table of flows, specifying how the flow should be handled in the future.

The table flows can be organized in different ways, depending on the needs of the network. For example, it may be organized by the source and destination IP addresses ,the type of traffic, or other network characteristics.

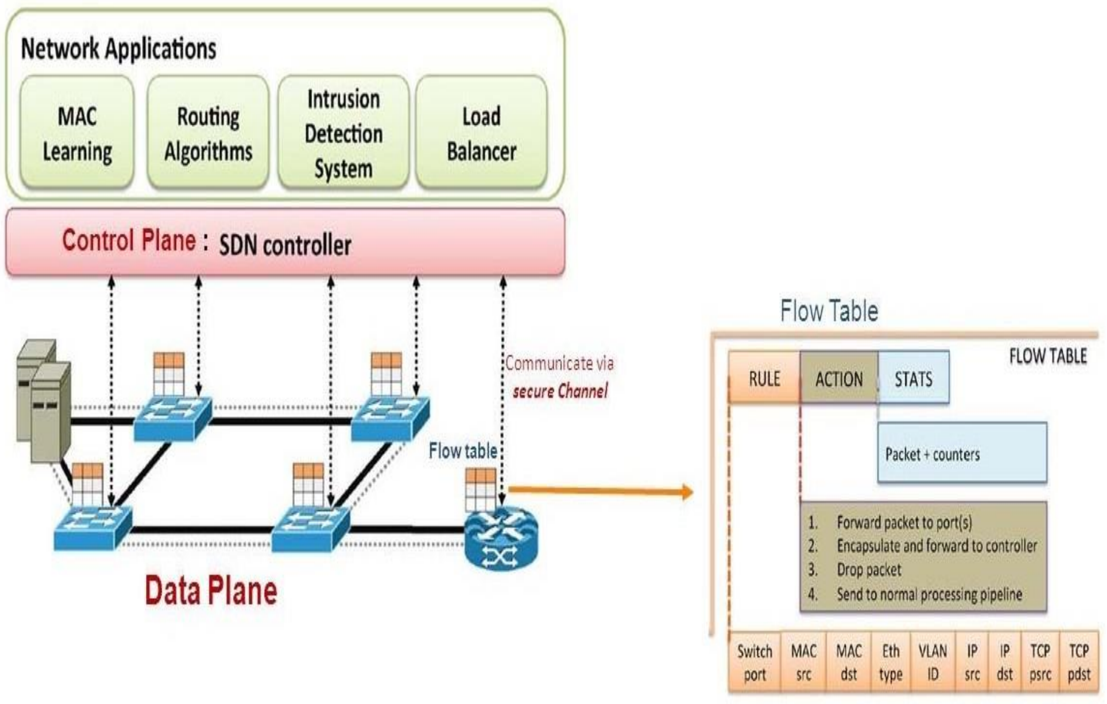


Figure1. 5 : OpenFlow Activated on SDN device.[6]

1.3.5 Northbound APIs:

The northbound APIs are the interfaces that connect the controller to the applications and services that use the network. These APIs allow the applications to communicate with the controller and access network information. The northbound APIs are typically RESTful APIs that use standard web protocols, such as HTTP.

1.3.6 Network Services Layer:

The network services layer provides additional network services, such as firewalls, load balancers , and intrusion detection systems. These services are integrated in to the SDN architecture and can be easily configured and managed through the controller.

In summary, an SDN architecture separates the control and data planes of the network , allowing for centralized control and automated management. The architecture includes the application layer, control layer, southbound and northbound APIs, data layer, and network services layer.

1.4 SDN controller type :

There are several types of SDN(Software-Defined Networking) controllers available, each with its own specific features and capabilities. Here are some of the most common types:

a. ***OpenFlow Controllers:*** These are the most widely used SDN controller sand are designed to support the OpenFlow protocol. They manage network traffic by programming networks witches and routers to forward packets according to rules defined in a central controller.

b. ***ONOS Controllers:*** The ONOS (Open Network Operating System) controller is an open-source SDN controller that supports both OpenFlow and non-OpenFlow protocols. It provides a scalable and high-performance platform for managing large-scale networks.

c. ***Ryu Controllers:*** The Ryu controller is another open-source SDN controller that supports OpenFlow. It provides a flexible framework for developing custom network applications and can be used for a wide range of network management tasks.

d. ***OpenFlow Controllers:*** These are the most widely used SDN controller sand are designed to support the OpenFlow protocol. They manage network traffic by programming networks witches and routers to forward packets according to rules defined in a central controller.

e. ***Floodlight Controllers:*** The Floodlight controller is an open-source SDN controller that supports both OpenFlow and non-OpenFlow protocols. It is highly extensible and can be customized to support a wide range of network applications.

f. ***NOX Controllers:*** The NOX controller is an open-source SDN controller that was one of the earliest controllers developed for OpenFlow. While it is no longer actively maintained, it is still used in some legacy SDN deployments.

g. ***POX Controllers:*** The POX controller is another open-source SDN controller that supports OpenFlow. It is highly modular and can be used to develop custom network applications.

There are also many commercial SDN controllers available, such as the Cisco Application Centric infrastructure (ACI) controller and the VMware NSX controller. These controllers are typically designed to work with specific vendor hardware and software and may offer additional features and functionality.

1.4.1 The function of an Openflow controller

The OpenFlow controller serves as a fundamental component of the control layer in SDN architecture. It is responsible for communicating with switches and can integrate the application layer. The main function of the controller is to manage flow entries in switches, allowing it to dynamically change the flow path by adding, deleting, or modifying flow entries. The controller

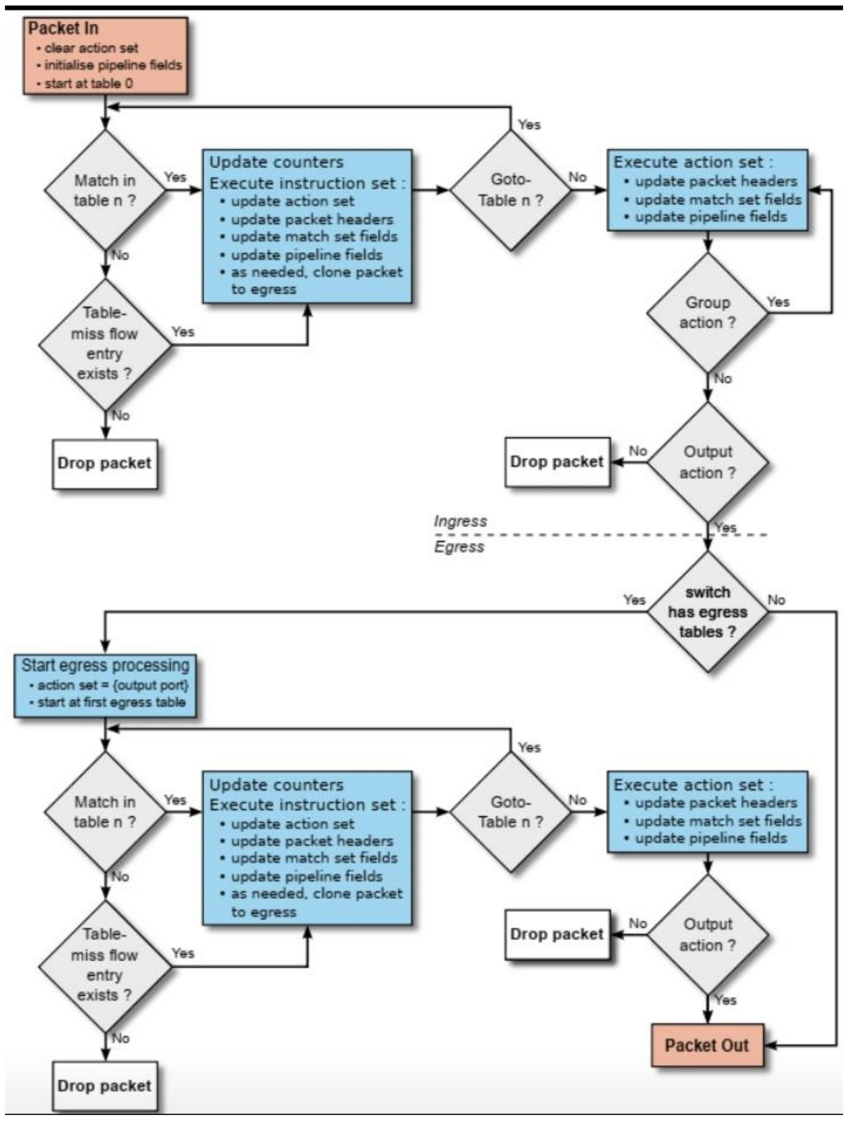


Figure1. 6 :The flowchat detailing packet flow through an Openflow switch .[7]

In modern implementations, controllers often provide a Representational State Transfer (RESTful) API, enabling easy access and interaction with the controller by the application layer. Packet In, Packet Out, and Flow Mod are three major OpenFlow messages :

1. ***Packet In:*** This message is sent from a switch to the controller. It occurs when the switch receives a packet that does not match any existing flow entries in its flow table .
2. ***Packet Out:*** This message is sent from the controller to the switch. It is used to instruct the switch to forward a specific packet or generate a packet for transmission .
3. ***Flow Mod:*** This message is used to add, modify, or delete flow entries in the flow table of a switch. The controller sends Flow Mod messages to the switch to define or update the flow rules that determine how the switch handles incoming packets .

1.4.2 Openflow switch components :

the flow tables in an OpenFlow switch are used to store flow entries and determine how packets are processed and forwarded. Each flow table contains a set of flow entries, which consist of matching fields, counters, and instructions .

When a packet arrives at the switch, it is initially processed by table 0. The flow entries in table 0 are checked sequentially based on their priority, and the packet is matched against the fields specified in each entry. If a match is found, the instructions associated with the matching entry are executed. These instructions can include actions such as forwarding the packet to a specific port, dropping the packet, modifying packet headers, or sending the packet to another table for further processing .

If a flow entry in table 0 specifies a directive to continue processing in another table, the packet is redirected to the specified table. This allows for more complex processing and decision-making based on the requirements of the network. The subsequent tables can have their own flow entries and perform further matching and processing based on the redirected packet .

The use of multiple tables in an OpenFlow switch enables the implementation of various network policies and forwarding behaviors. The flow tables can be populated and modified by the SDN controller through the OpenFlow protocol, allowing for dynamic control and management of packet flows within the network .

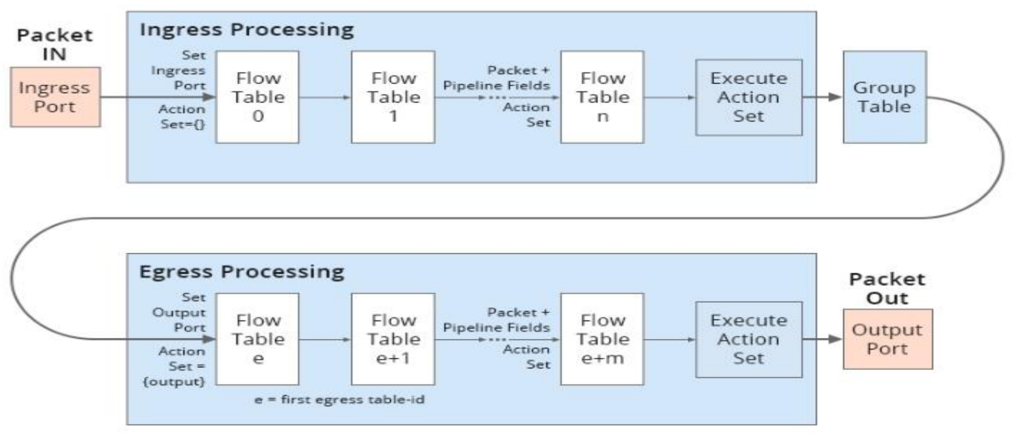


Figure1. 7:The control flow in the openflow protocol.[8]

1.4.3 Flow entire :

Of network traffic and routing, flow types can be classified as proactive and reactive. Here's an explanation of proactive and reactive flow types:

1. ***Reactive :*** when the first packet of a flow is received, it triggers the controller to insert flow entries into the switch's flow table. The switch efficiently utilizes its flow table, as each flow only requires a small additional setup time. However, if there is a loss of connection between the switch and the controller, the switch's utility becomes limited. Nevertheless, the fault recovery process in such a scenario is relatively simple
2. ***Proactive :*** in this scenario, the flow tables in the switch are pre-populated by the controller, eliminating the need for additional flow setup time. The controller inserts aggregated or wildcard rules into the flow tables, which reduces the overhead associated with individual flows. Additionally, with proactive flows, the processing load on the controller is reduced as queries are not sent to the controller for every flow. In the event of a connection loss between the switch and the controller, the traffic is not disrupted .

1.4.4 Openflow message

1. ***Asynchronous messages*:** Asynchronous messages are messages sent from the switch to the controller without a specific request from the controller. These messages are used to notify the controller about changes in the network or switch state, allowing the controller to update its information accordingly. Examples of asynchronous messages include Packet-in, Flow-removed, Port-status, and Role-status messages .

2. **Symmetric messages :** Symmetric messages are messages that are exchanged in both directions between the controller and the switch. These messages are used to establish and maintain the communication between the controller and the switch, as well as to detect and report any connection issues. Unlike asynchronous messages, symmetric messages are sent without explicitly requesting them from the other party. Examples of symmetric messages include Hello, Echo, and Error messages **.**

1.5 Comparison of SDN networks and traditional network :

Software-Defined Networking (SDN) and traditional networks differ in several key ways:

a. ***Centralized control:*** In an SDN network, the control plane is centralized and decoupled from the forwarding plane, which enables programmatic control and management of the network. In traditional networks, the control and forwarding planes are integrated, making network management more complex and difficult to automate.

b. ***Flexibility:*** SDN networks are highly flexible and programmable, making it easier to configure and manage the network, respond to changing network conditions, and deploy new services. traditional networks are often inflexible and rely on manual configuration, making it difficult to make changes quickly.

c. ***Visibility:*** SDN provides a centralized view of the network, enabling network administrators to monitor network performance, diagnose problems, and make changes quickly. In traditional networks, visibility is often limited, making it difficult to manage the network effectively.

d. ***Security:*** SDN provides a centralized point of control for network security, enabling administrators to easily configure security policies and apply them consistently across the network. In traditional networks, security is often an afterthought, making it more difficult to secure the network effectively.

Cost: SDN networks can be more expensive to implement and maintain compared to traditional networks, but they can offer significant cost savings in the long run due to the benefits of centralized control , flexibility ,and visibility.

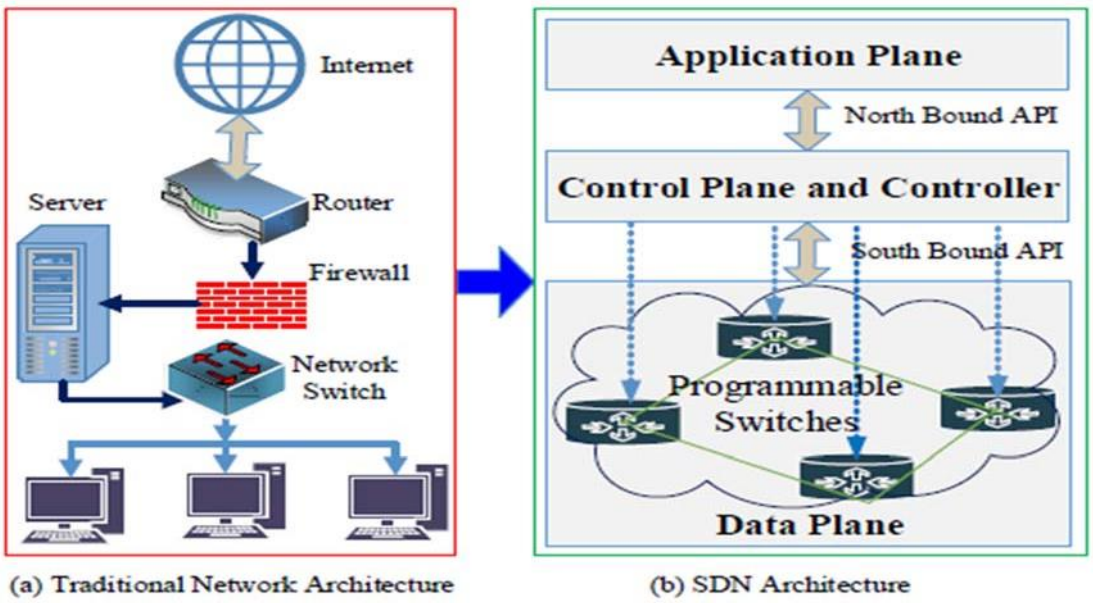


Figure1. 8:Comparison of SDN networks and traditional network.[9]

1.6 Definition routing sdn:

SDN routing , or Software-Defined Networking routing, is a network management approach where the control plane, which determines how network traffic is directed, is separated from the data plane, which handles the actual transmission of data packets. In SDN ,a centralized controller manages and controls routing decisions, allowing for a more dynamic, programmable, and flexible network.

SDN routing enables network administrators to define and manage routing policies through the controller, providing increased agility, scalability, and automation in network management.

SDN routing protocols, such as OpenFlow, provide a standardized interface between the controller and the network devices, allowing for interloperability and vendor-agnostic implementations.

Overall, SDN routing offers a more dynamic, efficient ,and adaptable approach to routing in modern networks.

1.6.1 The advantages of sdn routing over traditional approaches :

SDN routing offers several benefits over traditional approaches to network routing. Some of the key benefits of SDN routing include:

a. ***Centralized control:*** SDN routing allows for a centralized controller to manage and control routing decisions, providing a unified point of control for the entire network. This enables network administrators to define and enforce routing policies from a single location, simplifying network management and reducing configuration complexity.

b. ***Flexibility and programmability:*** SDN routing provides a highly programmable and dynamic network environment. With the ability to define and modify routing policies through the controller, Network administrators can quickly adapt the network to changing requirements and traffic patterns, making it highly flexible and adaptable.

c. ***Scalability:*** SDN routing enables network administrators to easily scale their networks by adding or modifying routing policies through the controller. This allows for efficient network growth and expansion without the need for complex and time-consuming reconfigurations of individual network devices.

d. ***Automation:*** SDN routing allows for automation of network management tasks ,reducing the need for manual configuration of individual network devices. This can result in reduced human error, faster network provisioning, and improved operational efficiency.

e. ***Cost-effective:*** SDN routing can offer cost savings through reduced network complexit.

1.7 Sdn routing protocols :

There are several SDN routing protocols that are commonly used in Software-Defined Networking environments. These protocols define the communication and interaction between the central controller and the network devices, enabling the controller to manage routing decisions. Some popular routing protocols include:

a. ***OpenFlow:*** OpenFlow is one of the most widely used SDN routing protocols. It provides a standardized interface between the controller and the network devices, allowing the controller to directly manage the flow of network traffic. OpenFlow allows for fine-grained control over routing decisions and enables dynamic adaptation to changing network conditions.

b. ***Border Gateway Protocol (BGP):*** BGP is a widely used exterior routing protocol that is also used in SDN environments. In SDN, BGP can be used to establish peering relationships between the controller and the network devices, allowing the controller to manage routing decisions for external traffic.

c. **Open Shortest Path First (OSPF):** OSPF is a widely used interior gateway routing protocol within traditional IP networks. In SDN, OSPF can be used as a routing protocol to calculate the shortest path between switches and distribute routing information.

d. ***Intermediate System to Intermediate System (IS-IS):*** IS-IS is a link-state routing protocol that is commonly used in large-scale networks, such as data center networks. In SDN, IS-IS can be used to propagate routing information to the controller, enabling centralized routing decisions.

e. ***Routing Information Protocol (RIP):*** RIP is a distance-vector routing protocol that is used in smaller networks. In SDN, RIP can be used to propagate routing in formation to the controller, allowing for centralized routing decisions.

Open Shortest Path First (OSPF): OSPF is a popular link-state routing protocol that is widely used in traditional networks. In SDN, OSPF can be used to propagate routing information to the controller ,enabling centralized routing decisions.

1.7.1 The different SDN routing protocols:

There are several SDN routing protocols that are used to implement routing in an SDN environment. Here are some of the most common ones:

a. ***OpenFlow:*** This is the most popular SDN routing protocol and is used to control the flow of traffic in an SDN network. It enables the separation of the control plane and data plane, allowing for centralized control of network traffic.

b. ***PCEP:*** [10] The Path Computation Element Protocol is used to compute the best path for network traffic based on network topology, traffic requirements, and other factors. It can be used in conjunction with other routing protocols to optimize network performance.

c. ***BGP-LS:*** [10] The Border Gateway Protocol Link State (BGP-LS) is used to disseminate topology information from the SDN controller to the network devices. It enables the creation of a global network view, which is necessary for effective traffic engineering.

d. ***ODL-L2SW:*** This protocol is used to configure and manage Layer 2 switches in an SDN environment. It provides a standard interface for controlling the forwarding behavior of switches in the network.

e. ***ODL-L3:*** This protocol is used to configure and manage Layer 3 switches and routers in an SDN environment. It provides a standard interface for configuring routing protocols and forwarding behavior.

f. ***ONOS-Routing:*** This protocol is used to implement routing in the ONOS SDN controller. It supports a range of routing protocols, including OSPF and BGP, and provides advanced features such as traffic engineering and network slicing.

g. ***LISP:*** The Locator/Identifier Separation Protocol is used to separate the identity of a device from its location in the network. This enables more efficient routing and mobility management in an SDN environment.

It's important to note that different SDN solutions may use different routing protocols, and some solutions may use a combination of different protocols. Understanding the different routing protocols and their capabilities is essential for implementing effective routing in an SDN environment.

Software-Defined Networking (SDN) is a network architecture approach that separates the control plane from the data plane. This allows for more flexibility and programmability in network management and allows network administrators to centrally manage network resources.

In traditional networking, switches and routers use their own control plane to determine how to forward packets in the network. However, in SDN, the control plane is moved to a centralized software controller, which is responsible for managing network traffic and forwarding packets. This controller communicates with the switches and routers in the data plane using a standard protocol, such as OpenFlow, to program and manage the network.

1.8 The tools needed to setup an sdn test environment

Setting up an SDN (Software-Defined Networking) test environment requires several tools and components to simulate and test different aspects of SDN networks. Here are some essential tools that may be needed:

a. ***SDN Controller:*** A software-defined networking controller is a key component of an SDN network. It provides the centralized control and management of the network, and facilitates communication between the network devices and the applications or services that run on top of the SDN network .The controller selected for this performance test are : Ryu[11],ONOS[12] and OpenDayLight [13]

b. ***Virtualization Software:*** Virtualization software allows for the creation of virtual networks and Virtual devices to simulate an SDN environment. Tools like Mininet, and EVE-NG can be used to create virtual switches ,routers, and hosts that can be connected to an SDN controller for testing and experimentation.

c. ***SDN-Compatible Network Devices:*** SDN-compatible switches ,routers, or access points are needed to build an SDN test environment. These devices should support protocols like OpenFlow, Which is the most common protocol used in SDN networks for communication between the controller and the network devices. Popular SDN-compatible devices include switches from vendors such as Cisco, HP, and Juniper[14].

1.9 Bandwidth Control :

The orchestration and SDN control software is also in control of every media flow. [15]This means that it can effectively allocate the required amount of bandwidth to each media flow, based on its specific needs and priorities. Additionally, the software can anticipate future bandwidth requirements for scheduled productions, allowing for efficient and proactive allocation of network resources. Overall, this results in a more streamlined and optimized network infrastructure for managing media flows.

1.10 Protection :

The orchestration and control software has a deep understanding of the network's structure and how data flows through it. With this knowledge, the software can dynamically create and manage multiple network paths, or path diversity, to protect against potential failures or disruptions in the network. By distributing traffic across multiple paths, the software can effectively ensure that media flows can continue uninterrupted, even in the event of a network outage or other issues. Overall, this allows for a more resilient and reliable network infrastructure for managing media flows .

1.11 Supported Architecture:

SDN (Software-Defined Networking) can adapt to different network architectures with ease, as long as the appropriate orchestration and control software is in place. Unlike traditional automated routing, which may struggle to handle certain network topologies, SDN's flexibility allows it to manage network structures such as star, dual-star (pseudo spine-leaf), or true leaf spine without any compromises. This means that the SDN infrastructure can be tailored to the specific needs of the network, without being limited by its underlying architecture. Overall, this results in a more efficient and adaptable network infrastructure for managing media flows .

1.12 Drivers:

In an SDN-controlled network, the need for drivers for every piece of equipment is eliminated because the control is managed within the network. This allows for easy and cost-effective integration of new equipment into the production set-up.

Traditionally, each piece of equipment would require a specific driver to be installed, which could be time-consuming and costly. With SDN, equipment can be seamlessly integrated into the network without the need for additional drivers, which simplifies the process of expanding or updating the production set-up.

Overall, this results in a more flexible and agile network infrastructure for managing media flows, with reduced barriers to incorporating new equipment into the production environment .

1.13 Conclusion :

SDN is a network architecture that separates the control and data planes in devices, allowing centralized management, automation of tasks, and improved performance and security. It is becoming a crucial tool for modern networks, particularly in datacenters, transforming the way they are designed, deployed, and managed. SDN provides flexibility and programmability, utilizing various routing protocols such as OpenFlow, BGP, IS-IS, RIP, OSPF, making it easy to manage and automate tasks. The separation of the control plane from the data plane enhances security and scalability, making SDN a preferred option for modern networks. Its continued development and evolution offer vast possibilities for network optimization and innovation.

Chapter 2
Fault tolerance in SDN NETWORk

2 Introduction

Eliminating complex control functions leads to improving the efficiency of routing, in addition to enabling new strategies for network management.

However, failures and disruptions can occur in SDN environments, impacting the availability and reliability of network services. Fault tolerance is a crucial aspect of SDN design that focuses on mitigating the impact of failures and ensuring uninterrupted network operations.

The primary objective of fault tolerance in SDN is to create a resilient network infrastructure that can adapt to failures and continue providing reliable services. By integrating fault tolerance mechanisms into the SDN architecture, organizations can enhance network robustness, improve service availability, and reduce the time required to recover from failures.

In the following discussion, we will explore some key fault tolerance techniques commonly employed in SDN environments, highlighting their significance and impact on network reliability.

2.1 Dealing with error

Systems inherently encounter a range of problems due to both flexible and rigid hardware defects ,which can which can lead to service interruptions . It is essential to differentiate among three terms: "defect," "error," and "failure." These concepts are interrelated, as depicted in the diagram .

A fault is an irregularity that occurs when the operational state of a network strays from its anticipated or standard condition. It acts as the fundamental trigger for errors, such as software defects or malfunctions, which can be linked to human activities or disturbances in power provision. The propagation of errors leads to unique or multiple failure scenarios within the system, resulting in service disruptions.[16][17]

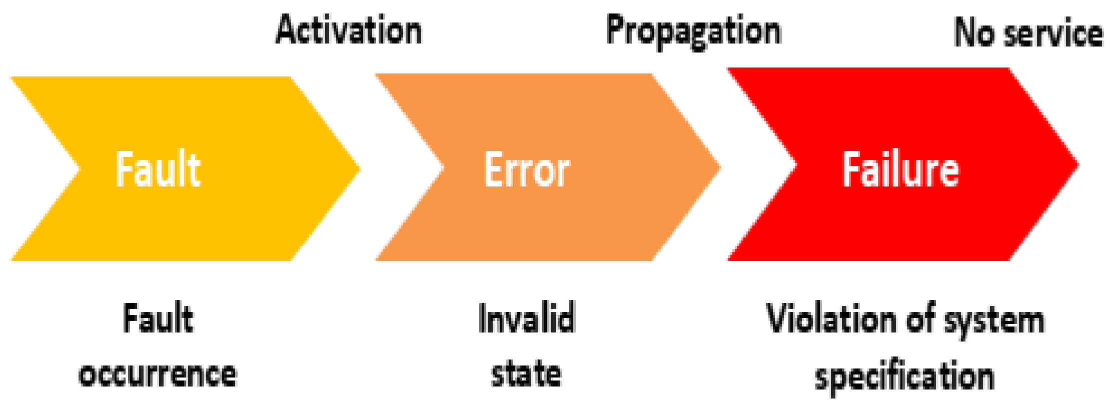


Figure 2. 1 :The relationship betwenn fault ,Error ,and failure.[18]

2.2 Definition fault tolerance

Fault tolerance is a widely used term in the fields of networking and computer science. It describes a mechanism that enables systems to recover from failures and prevent service interruptions when a system malfunction occurs. Essentially, it refers to the system's capability to provide reliable services despite the presence of unreliable components. One way to achieve fault tolerance is by implementing redundant backup elements for each unreliable component. These backup elements are designed to take over in the event of a failure, ensuring continuous and uninterrupted service delivery .

2.3 The Importance of fault tolerance in sdn

Fault tolerance is crucial in SDN because it helps ensure that the network remains operational even in the face of failures. In traditional networking, failures can often cause widespread outages that can be difficult and time-consuming to troubleshoot and fix. With SDN, fault tolerance can be built in at the design level, making it easier to recover from failures.

Real-world examples of network failures abound, from natural disasters to cyber attacks. Without fault tolerance, these types of events can have catastrophic consequences for businesses and organizations. By implementing fault tolerance in SDN, you can help protect your network from these types of events and ensure that your business operations continue uninterrupted.

2.4 Defintion fault tolerance in sdn

Fault tolerance in Software-Defined Networking (SDN) refers to the ability of the network infrastructure to withstand and recover from failures or disruptions without significant impact on service availability and performance. It involves the implementation of mechanisms and strategies that detect, isolate, and recover from faults, ensuring uninterrupted network operations and maintaining reliability.

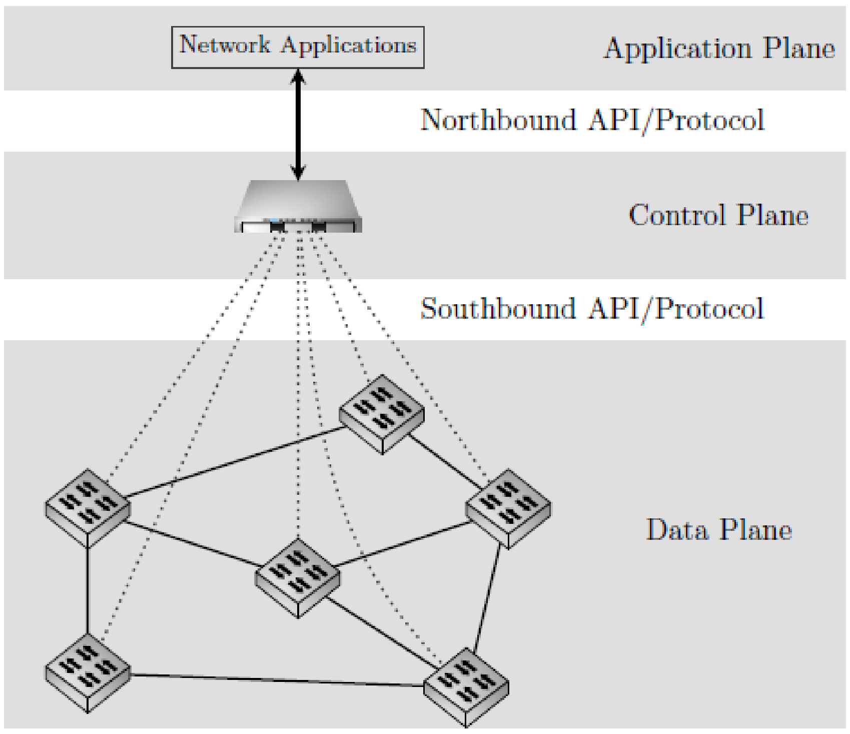


Figure 2. 2 :Structure of a SDN network.[19]

2.4.1 The Fault Tolerance Problem

The issue of fault tolerance is significant in networking as it ensures uninterrupted operation of networks, even in the presence of link or device failures. The network should be capable of swiftly and transparently handling faults, causing minimal service disruption. Due to the separation of data and control planes, fault tolerance needs to be addressed separately for each plane. In the context of Software-Defined Networking (SDN), fault tolerance must be considered for both the data plane (switches and links) and the control plane (controllers and controller-switch link)

This paper primarily focuses on fault tolerance in the data plane, while providing an overview of fault tolerance in the control plane .

2.5 Fault Tolerance in the Data Plane

The functionality of software-defined networks heavily relies on the proper operation of the controller, which manages the control logic in the data plane. However, fault tolerance can be implemented in software-defined networks at three different levels: the data plane, control plane, and application plane .

When it comes to fault tolerance in the data plane, it shares similarities with traditional network architectures. However, the dynamic nature of software-defined networks has necessitated the reimagining of fault detection and recovery methods for links to ensure fault tolerance. The primary emphasis is on swiftly identifying faults .

The introduction of the software-defined networking model has shifted the complexity from network devices to control plane elements. Consequently, the computational load on data plane network elements has been significantly reduced. This allows for more flexible execution of applications in the data plane, resulting in improved Quality of Service (QoS), Quality of Experience (QoE), and the ability to meet the requirements of next-generation networks. For instance, the OpenFlow protocol offers rapid fault detection mechanisms and simplified configuration of alternative paths on switches. In this way , the failure recovery delay is reduced , the packet loss rate is minimized , and a high effective transport rate is achieved in the network . [20]

2.5.1 Reacting to topology changes

when considering fault tolerance in SDN-based networks, it is important to weigh the potential benefits of rerouting traffic to a better path against the overhead caused by reconfiguring the network .

In a dynamic SDN network, where links are constantly being added or removed, the network's topology is continuously changing. It is desirable for traffic to take a near-optimal path through the network, but there is always a possibility that the chosen path may fail. In such cases, a failover algorithm is used to select the next best path for restoring network function .

Finding the optimal failover path can be computationally expensive and time-consuming, especially for large networks. As a result, it may be acceptable to restore network function by applying a suboptimal path initially, rather than waiting for the calculation of the optimal path. The focus is on getting the network back to a functional state quickly, and the suboptimal path can be optimized later when there is more time .

However, it is important to consider the overhead caused by traffic traversing a suboptimal path. This overhead can include increased latency, reduced throughput, or inefficient resource utilization. In some cases, rerouting the traffic to a lower-overhead path may be beneficial in terms of reducing the costs associated with traversing the network .

Nevertheless, reconfiguring the network to implement a new path also introduces overhead. This overhead includes the time and resources required to update the network's forwarding rules and propagate the changes across the network. Therefore, any approaches to fault tolerance should carefully evaluate whether the benefits of rerouting traffic outweigh the overhead caused by network reconfiguration .

In summary, fault tolerance in SDN-based networks should consider the trade-off between rerouting traffic to optimize path selection and the overhead introduced by reconfiguring the network. The decision to change existing paths should only be made if the benefits of rerouting outweigh the costs associated with network reconfiguration .

2.5.2 Minimizing overhead

Paris et al. [21] propose an approach that addresses the balance between optimal paths and the frequency of network reconfiguration in SDN-based networks. Their approach consists of two sub-mechanisms: rapid handling of failures by rerouting to alternative paths and a mechanism for path optimization .

When a link or device failure occurs in the network, the first sub-mechanism focuses on quickly restoring network functionality by calculating backup paths on demand. The priority is to find an alternative path as quickly as possible, even if it is suboptimal. A shortest-path algorithm is typically used to calculate these alternative paths. The primary goal at this stage is to ensure the network is operational, and path optimization is deferred to the second sub-mechanism .

The second sub-mechanism, known as the Garbage Collection of network resources, focuses on path optimization. It involves periodically analyzing and optimizing flow allocations in the network. An iterative algorithm is employed to converge towards the optimal solution. This optimization process takes into account the availability of new links and the repair of failed links. If network changes open up shorter paths, the garbage collection mechanism may reroute traffic accordingly. However, rerouting is only performed if the optimization gained from the new path outweighs the overhead caused by the necessary network reconfiguration .

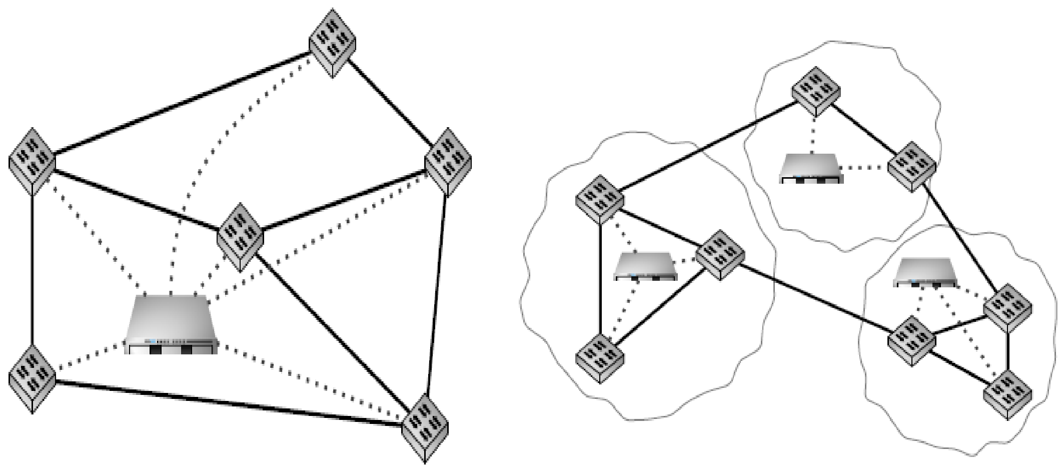
In a static network, paths would naturally converge towards the optimal solution. However, failed links and devices introduce suboptimal paths, leading to additional overhead and deviating further from the optimal solution. The path optimization mechanism helps counteract this deviation by periodically analyzing and optimizing the flow allocations in the network .

Paris et al.'s approach combines rapid restoration of network paths after failures with periodic path optimization. By prioritizing quick recovery through suboptimal paths and periodically optimizing paths using the Garbage Collection mechanism, they aim to strike a balance between network functionality and the overhead caused by reconfiguration .

2.5.3 Fault tolerance controller

Ensuring fault tolerance in the control plane is crucial because the availability of the controller is not guaranteed at all times. Controller failures can occur, leaving the network without proper guidance and rendering it in a headless state. In such situations, the network is unable to handle events like incoming packets belonging to unknown flows without the presence of a functioning controller. Therefore, implementing a fault-tolerant control plane becomes essential .

In the following section, we will explore different approaches to achieve a fault-tolerant control plane .



(a) Centralized (b)Centralized

Figure 2. 3 : SDN Control Plane Topologies.[19]

2.6 Fault-Tolerance in the Control Plane

Ensuring fault tolerance in the control plane is crucial because the data plane relies on the controller for its operation. The control plane performs critical tasks such as handling unknown flows and facilitating communication with the application layer. Thus, the controller needs to be operational at all times to ensure proper network functioning .

In SDN, the control plane is based on a centralized controller, which means that the controller acts as a single point of failure. If the controller becomes unavailable, the network may experience disruptions and loss of functionality. Therefore, introducing redundancy in the control plane becomes necessary to enable fast failover in case of a controller fault.

In the following section, we will explore different approaches to achieving fault-tolerant controllers, which aim to minimize the impact of controller failures and ensure continuous network operation.

2.6.1 Control plane topology

There are two common types of control plane topologies in SDN: a single logically centralized controller and a distributed topology.

In the single logically centralized controller topology (Figure 2.3 a), the control plane is managed by a single controller that has a global view of the network. This controller is responsible for making decisions and controlling the behavior of the entire network. This topology is typically used in smaller networks where the load on the controller can be handled effectively.

On the other hand, the distributed topology (Figure 2.3 b) is prevalent in larger networks where a single controller may not be able to handle the scale and complexity of the network. In this topology, multiple controllers are deployed, with each controller managing a specific domain or subset of the network. These controllers can operate in parallel, handling different parts of the network independently. This distributed approach allows for better scalability and load balancing .

One of the advantages of a distributed topology is its inherent fault tolerance. If one controller fails, another controller can take over the responsibility for its domain, ensuring continuity of network control. This redundancy and failover capability help maintain network functionality even in the presence of controller failures.

SDN control planes can be structured with a single logically centralized controller or a distributed topology. The distributed topology, with multiple controllers handling different network domains, offers better scalability and fault tolerance by allowing controllers to take over each other's responsibilities in case of failure .

2.7 Causes of SDN Failure :

There are a lot of reasons , let's focus on two specific reasons that can contribute to the downfall or failure of a sdn:

1. ***Link Failure:*** A link failure refers to the loss of connectivity between two network devices or switches. This can occur due to cable damage, hardware malfunction, or environmental factors. When a link fails in an SDN, it can result in disrupted communication, increased latency, or even complete network partitions .
2. ***Switch Failure:*** A switch failure happens when a network switch malfunctions or becomes unresponsive. This can occur due to hardware failures, power outages, software bugs, or misconfigurations. A switch failure can impact the forwarding of network traffic, leading to connectivity issues or degraded network performance .
3. ***Controller failure*** : A controller failure occurs when the controller becomes unresponsive or experiences a system crash. This can happen due to software bugs , hardware malfunctions , or resource limitations. When a controller fails ,it can result in the loss of network control .
4. ***Controller-switch communication failures:*** Communication failures between the SDN controller and switches can occur due to network connectivity issues, congestion, or controller overload. These failures can result in switches being unable to receive instructions from the controller, leading to misconfiguration or an inability to handle dynamic network changes .
5. ***Network application failures:*** Faults can occur within network applications running on top of the SDN infrastructure. For example, a network application may experience a software bug, resource exhaustion, or failure to handle network events properly. Application failures can impact specific services or functionality provided by the SDN environment .
6. ***Configuration errors:*** Faults can also arise from misconfigurations within the SDN infrastructure. Human errors in setting up flow rules, controller policies, or switch configurations can lead to unintended consequences, network congestion, or incorrect routing decisions .

7. ***Resource limitations:*** Resource limitations, such as insufficient bandwidth, memory, or processing power, can cause performance degradation or failures in an SDN environment. Overutilization of resources can result in network congestion, packet drops, and deteriorated Quality of Service (QoS).

When analyzing fault tolerance in SDN, it's essential to consider these different fault scenarios and develop mechanisms that can detect, recover, and mitigate the impact of these faults to ensure the overall resilience and reliability of the SDN infrastructure

2.8 Fault tolerance mechanisms

Recovery mechanisms in SDN can be categorized based on their scope, which can be local or end-to-end path recovery. Local recovery involves bypassing the failed network component, while end-to-end path recovery bypasses the entire affected path between the source and destination. Two common approaches for recovery are protection and restoration. Protection strategies use preconfigured alternate paths, while restoration strategies establish alternative paths upon fault detection. Restoration introduces additional delay in computing the alternative path .

Protection strategies can be divided into 1:1 path protection and 1+1 path protection. In 1+1 path protection, data is duplicated and sent to both the primary and backup paths, with the duplicated packets ignored at the destination. This approach has a negative impact on the useful transport rate. On the other hand, 1:1 path protection transmits data through the primary path, and if the primary path fails, it is replaced by the pre-established backup path . [22]

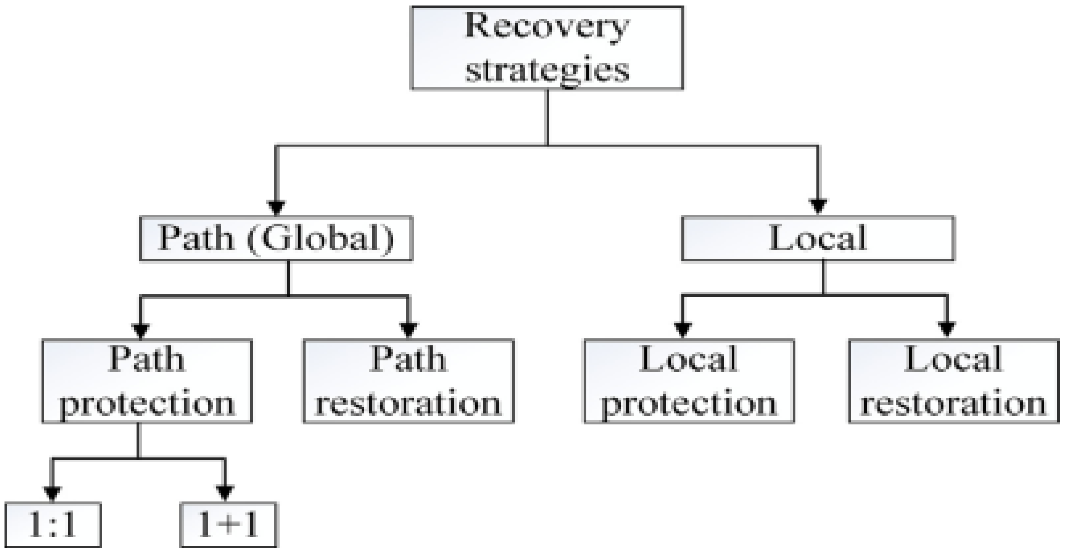


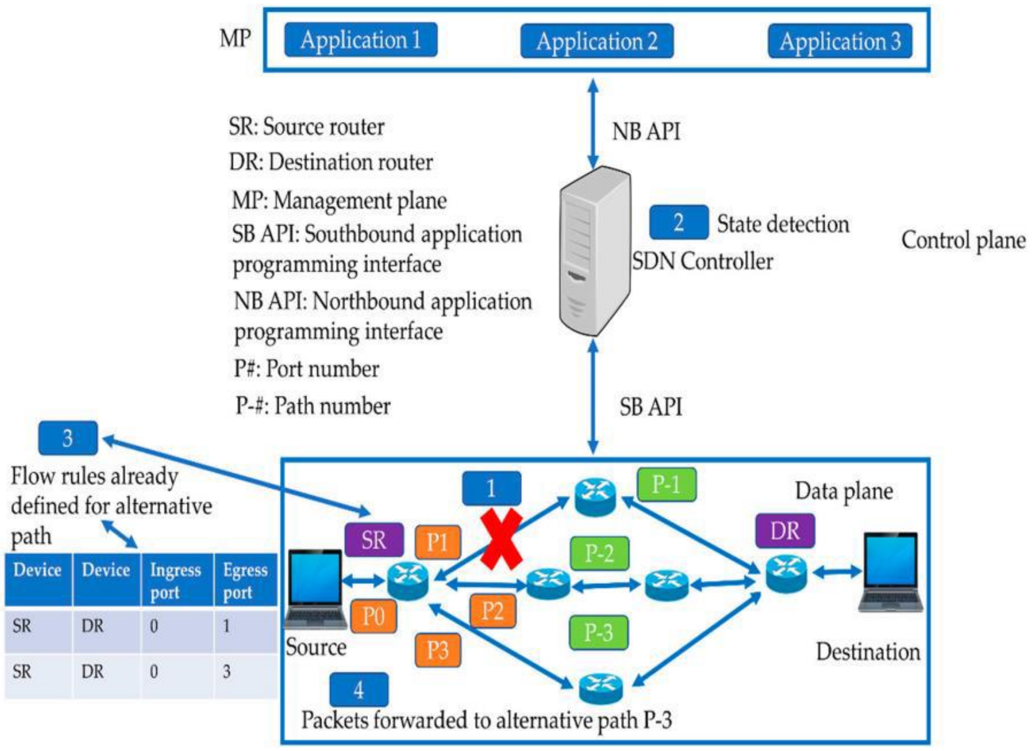
Figure 2. 4 :Mechanisms for distance recovery in networks.[22]

The diverse models of software-defined networks provide a broad array of solutions to tackle the challenges encountered by networks that are hard to overcome using conventional approaches.

The centralization of control plays a crucial role in enabling the technical viability of these solutions. Fault tolerance stands out as a significant challenge across the data, control, and application levels. The primary focus of this study is to address fault tolerance specifically at the data level, as our research centers around exploring fault tolerance mechanisms within this domain.

2.9 Restoration of link failure using proactive approaches

There are two methods for handling link or switch failures: proactive protection and reactive restoration. Proactive protection involves setting up backup paths in advance. When a failure occurs, the affected traffic is promptly and locally rerouted to the alternative path without requiring intervention from the controller. The process is illustrated in Figure 2-3, which demonstrates how link failures are addressed using proactive protection .



Device	Device	Ingress port	Egress port
SR	DR	0	1
SR	DR	0	3

Figure 2. 5: The mechanism in which link failure is handled using proactive protection.[23]

When a link failure occurs in path number 1, the switch already has the necessary information for the alternative path stored in its flow rule table. This allows for seamless redirection of packets to the specified alternative path. Proactive protection, in comparison to reactive restoration, offers faster recovery time as it eliminates the need for consulting the control plane and calculating the alternative path in real-time. The objective is to achieve path recovery within a timeframe of less than 50 milliseconds, which is the specified requirement for acceptable solutions in this domain. However, configuring an alternative path for every primary path may exceed the storage capacity of flow table entries on switches, leading to potential limitations. Additionally, it introduces additional processing overhead for flow matching when dealing with a large number of flow entries. The subsequent sections present and discuss proposed solutions for proactive protection .

2.10 Challenges and Limitations

While fault tolerance is important in SDN, there are also challenges and limitations to implementing it effectively. For one, building a fault-tolerant network can be complex and expensive, requiring specialized hardware and software.

Additionally, achieving fault tolerance in SDN can require trade-offs in terms of performance and efficiency. For example, using redundant network paths can increase latency and reduce bandwidth utilization. It's important to carefully consider these trade-offs when designing a fault-tolerant SDN network.

2.11 Advantages of Fault Tolerance in SDN

Fault tolerance in software-defined networking (SDN) has several advantages. Firstly, it ensures that the network remains operational even when there are hardware or software failures. This is crucial for businesses that rely on their networks to function properly. Secondly, fault tolerance enables the network to adapt to changing conditions and avoid potential faults. This helps to improve the overall reliability and performance of the network.

Another advantage of fault tolerance in SDN is that it allows for faster recovery times in the event of a failure. With redundancy and failover mechanisms in place, the network can quickly recover from any faults and continue providing uninterrupted service. This is particularly important for mission-critical applications where downtime can have serious consequences.

2.12 Conclusion

In conclusion, fault tolerance is of utmost importance in Software-Defined Networking (SDN) for ensuring reliable and uninterrupted network operations. By implementing fault tolerance mechanisms, such as redundant controllers, link redundancy, and load balancing, SDN networks can enhance their resilience to failures and disruptions. Fault tolerance enables service continuity, minimizes downtime, improves scalability. It allows for efficient management and control of the network, even in the presence of failures .

Chapter 3

IMPLEMENTATON OF THE SOLUTION

3 Introduction :

This chapter represents the technical aspect of fault tolerance in software-defined network by showcasing the tools used and the steps involved ,while specifying the protocol used .

3.1 Emulation Environment:

The software environment is made up of the following parts:

VMware :

VMware is a software company that specializes in virtualization and cloud computing technologies. It was founded in 1998 and is headquartered in Palo Alto, California. VMware's products include virtualization, networking, and security management tools, as well as software-defined data center and storage software. Its flagship product is VMware vSphere, which is a server virtualization platform used for implementing and managing virtual machines on a large scale. VMware also offers desktop software compatible with Linux, Microsoft Windows, and Mac OS X. The company is a subsidiary of Dell Technologies.

MiniNet

MiniNet is network emulator, It supports Openflow protocol for integrqtion with an SDN controller .this allows for the efficient execution of small-scale networks with artificial traffic on computers that may not be powerful. The software is free to use and has an open-source license. Creating a network with a large number of devices can be difficult and expensive, which is why MiniNet uses a virtual mode strategy to create prototypes and emulations of technological networks. The software emulates a complete network using only a single system running a Linux kernel. Although the elements of the network such as nodes, switches, routers, and links are created by software, they are designed to behave like real elements. MiniNet's goal is to create virtual networks with ease and speed, which can run nodes, network cores, and virtualized network devices through a simple feature host, such as Linux. The software can emulate different types of network elements, such as nodes, layer 2 switches, layer 3 routers, and links.

3.2 Proposed controllers :

OpenDayLight

ODL is an open-source software-defined networking (SDN) platform that provides a framework for building SDN applications and a centralized point of control for managing network traffic using software. It enables network programmability, automation, and supports a wide range of networking protocols and technologies .

HPE VAN Controller

The HPE Virtual Application Networks (VAN) SDN Controller Software serves as a centralized control point in a software-defined network (SDN) and streamlines tasks such as network management, provisioning, and orchestration. It allows for the delivery of new application-based network services and offers open APIs for developers to create innovative solutions that dynamically link business needs to network infrastructure through custom Java programs or RESTful control interfaces. The HPE VAN SDN Controller is suitable for use in campus, data center, or service provider settings.

ONOS

controller is an open-source SDN controller that supports both OpenFlow and non-OpenFlow protocols. It provides a scalable and high-performance platform for managing large-scale networks.

3.3 Implementation:

In this part, we will present the different phases of the realization of our project:

Installing technologies related to computer networking and virtualization :

In the first phrase of our project , we set up a virtual machine running Linux on VMware and installed Mininet on it .

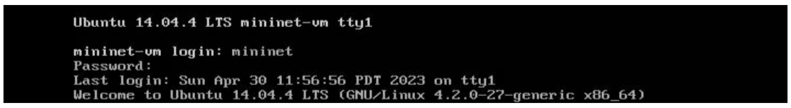
3.3.1 Installing Mininet on VMware:

1. We downloaded and installed the VMware software on our computer .

2. We downloaded the latest Mininet version 23.0.4 VM image from the Mininet website. The VM image is a pre-configured virtual machine with Mininet already installed .

3. We imported the Mininet VM image into VMware. To do this, we opened VMware and went to File > Open, then selected the Mininet VM image file. This created a new virtual machine in VMware with Mininet pre-installed.

4. We started the Mininet VM in VMware. We selected the Mininet VM in the VMware interface and clicked the "Start" button. This started the virtual machine and launched Mininet.
5. When Mininet is launched ,It many prompt you to entre some commands such as a password ,username … etc .



```
Ubuntu 14.04.4 LTS mininet-vm tty1

mininet-vm login: mininet
Password:
Last login: Sun Apr 30 11:56:56 PDT 2023 on tty1
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)
```

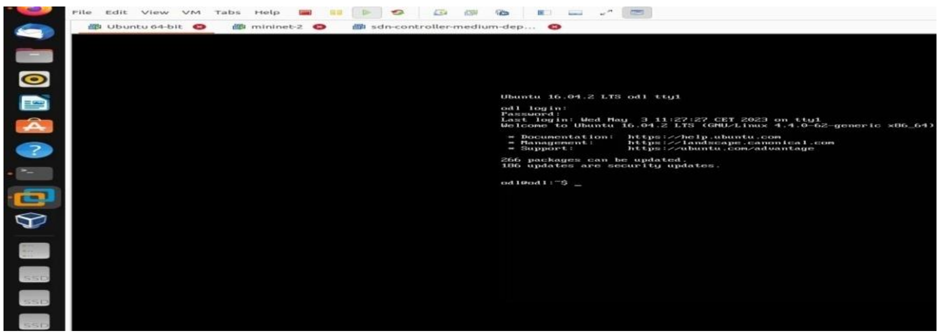
Figure3. 1 : The mininet strated working .

6. We change the IP address because every time we connect to a new network ,the IP address changes .we change it to make it a static IP address.

After starting Mininet , we will install three controllers :

3.3.2 Install opendaylight :

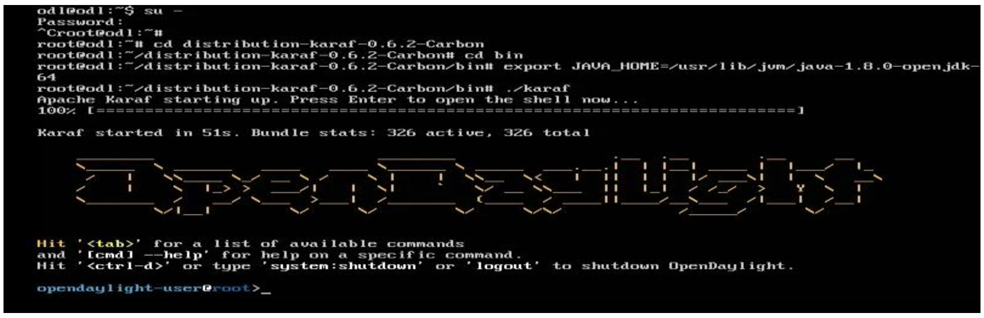
1. First , we download the Ubuntu server version 16.04.2 that supports Opendaylight .
2. We have installed Ubuntu server on VMware .



```
Ubuntu 16.04.2 LTS odl tty1

odl login:
Password:
Last login: Wed May  3 11:27:27 CET 2023 on tty1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-62-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

266 packages can be updated.
186 updates are security updates.

odl@odl:~$ _
```

Figure3. 2 :Installed Ubuntu Server in VMware.

3. After completing the installation of Ubuntu Server , we download and run Opendaylight (odl) v on it .



```
odl@odl:~$ su -
Password:
^Croot@odl:~#
root@odl:~# cd distribution-karaf-0.6.2-Carbon
root@odl:~/distribution-karaf-0.6.2-Carbon# cd bin
root@odl:~/distribution-karaf-0.6.2-Carbon/bin# export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-
64
root@odl:~/distribution-karaf-0.6.2-Carbon/bin# ./karaf
Apache Karaf starting up. Press Enter to open the shell now...
100% [========================================================================]

Karaf started in 51s. Bundle stats: 326 active, 326 total

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>_
```

Figure3. 3 : Opendaylight Controller Launches.

4. Now , use the URL 10.83.0.99:8443.sdn/ui/app/index#oftopology on any web browser as show in Figurer:

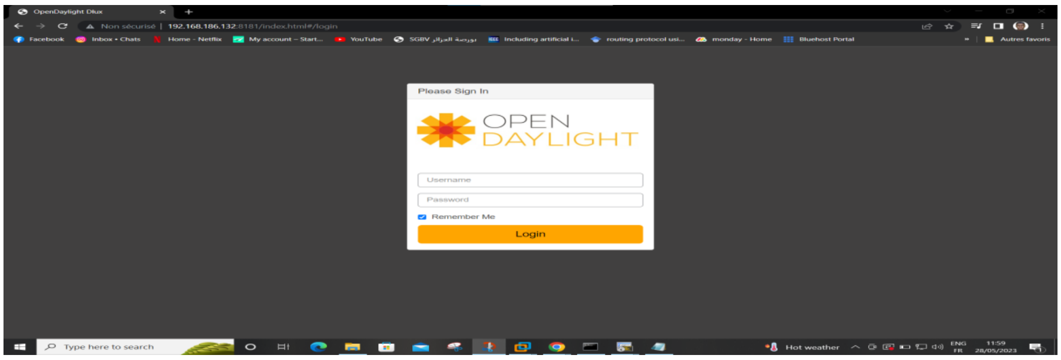


Figure3. 4 :OpenDaylight Controller Launches.

5. To access the Hpe-Van sdn controller ,we should use the default username and password.

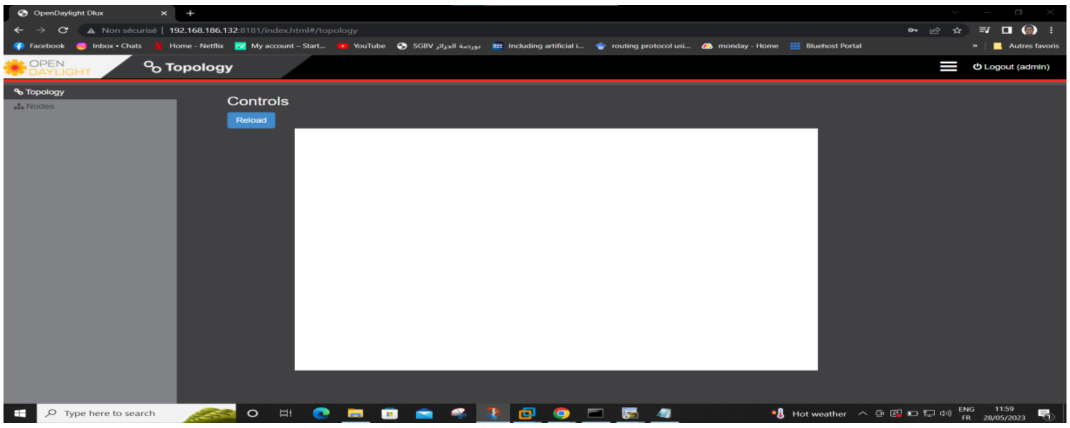
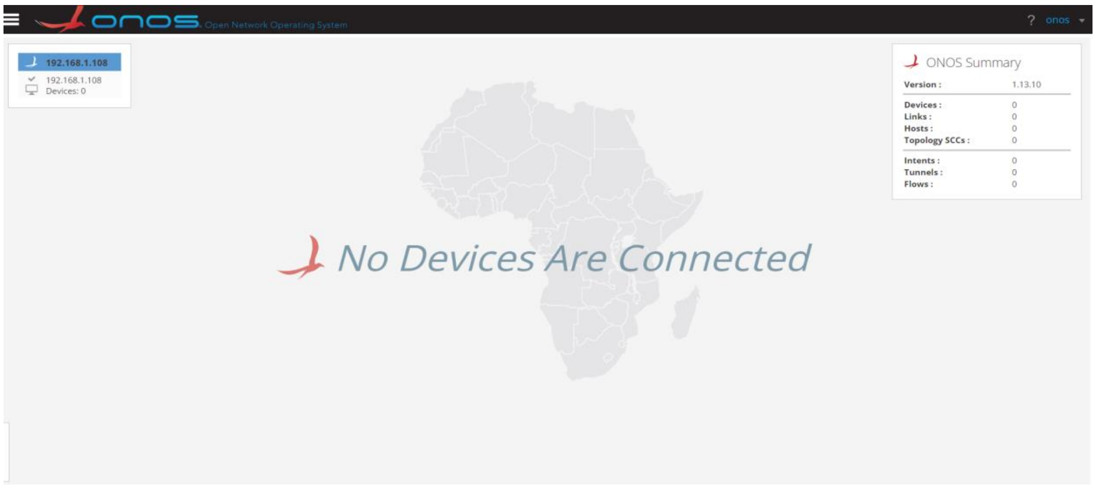


Figure3. 5 : Connecting to the Web Interface of the Opendaylight.

Once Opendaylight is installed successfully on Ubuntu server ,we establish a connection between Opendaylight and Mininet .

3.3.3 Install ONOS :

1. After downloading Ubuntu 16.04.2 .
2. We have installed Onos controller version 1.13.1 on Ubuntu server .



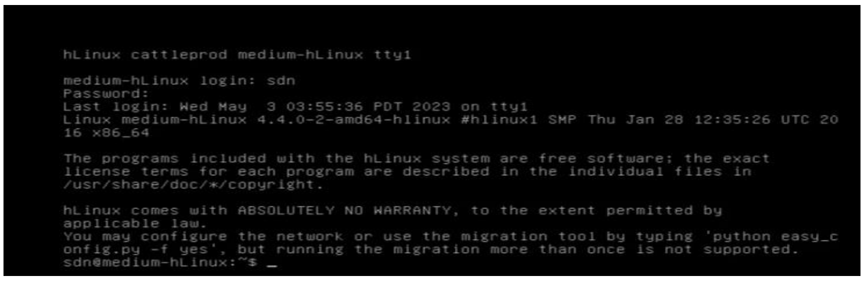
ONOS Summary

Version :	1.13.10
Devices :	0
Links :	0
Hosts :	0
Topology SCCs :	0
Intents :	0
Tunnels :	0
Flows :	0

Figure3. 6: Connecting to the web interface of onos controller.

3.3.4 Install HPe –Van .sdn-Controller :

1. We downloaded the HPe –Van .SDN –Controller installation package
2. We opened VMware software and created a new virtual machine , them downloaded and installed hpe-van sdn controller on it ,and modifying some settings to run the controller .
3. Here ,the installation is complete and the HPe-Van sdn controller service has started .



```
hLinux cattleprod medium-hLinux tty1

medium-hLinux login: sdn
Password:
Last login: Wed May  3 03:55:36 PDT 2023 on tty1
Linux medium-hLinux 4.4.0-2-amd64-hlinux #hlinux1 SMP Thu Jan 28 12:35:26 UTC 20
16 x86_64

The programs included with the hLinux system are free software; the exact
license terms for each program are described in the individual files in
/usr/share/doc/*/copyright.

hLinux comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
You may configure the network or use the migration tool by typing 'python easy_c
onfig.py -f yes', but running the migration more than once is not supported.
sdn@medium-hLinux:~$ _
```

Figure3. 7 :Installation Completed and HPE-VAN SDN Controoller service started.

4. Now , we used the URL 10.83.0.99:8443.sdn/ui/app/index#oftopology on any web browser as show in Figurer:

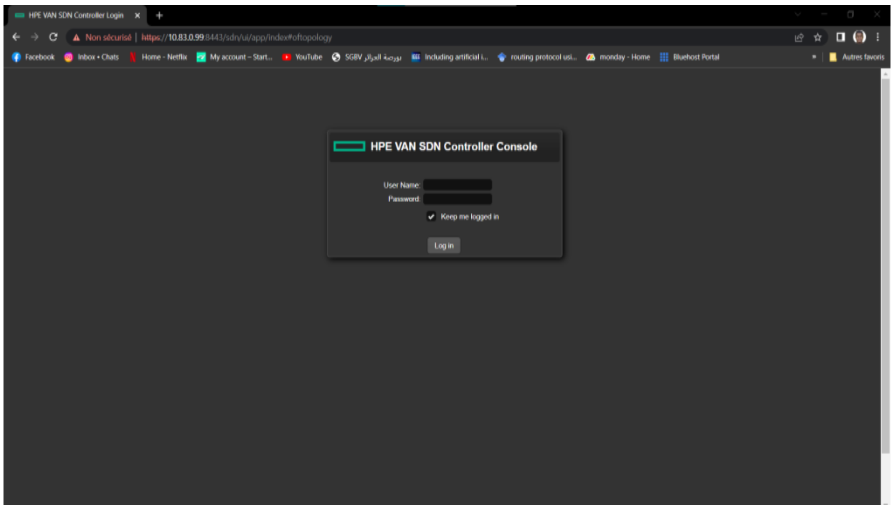

HPE VAN SDN Controller Login
Non sécurisé | https://10.83.0.99:8443/sdn/ui/app/index#oftopology
HPE VAN SDN Controller Console
User Name:
Password:
Keep me logged in
Log in

Figure3. 8 :Web Interface representing the home page of the Hpe-Van sdn Controller .

5. To access the Hpe-Van sdn controller ,we should use the default username and password.

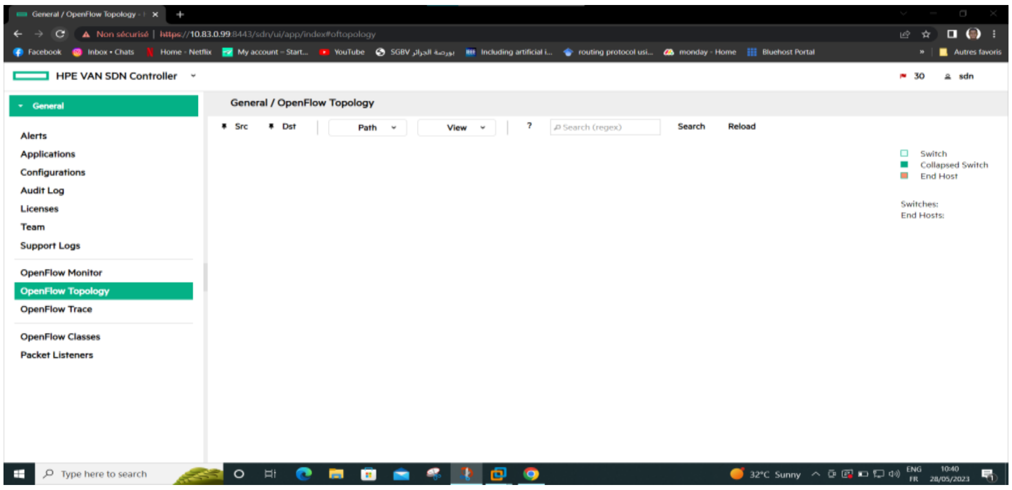


Figure3. 9 : Connecting to the Web Interface of the Hpe-Van sdn controller.

After installing hpe-van sdn controller on VMware ,we need to connect it to Mininet.

3.4 The fundamental command for Mininet :

The fundamental commands for utilizing Mininet :

Directive	Explanation
Sudo –S	The command is used to gain root privileges and does not need to be used before every Mininet command .
Sudo mn- h	The command is used to display the Mininet help menu, providing information and options related to Mininet commands and functionality .
Sudo mn	The command is used to create a Mininet network with the default topology, which is the minimal topology .
Sudo mn-c	The command is used to clear Mininet or undo the effects of a previously executed command .
Mininet>net	The command in the Mininet displays and lists the links that have been formed in the network.
Mininet>links	The command displays detailed information about all the links associated with the nodes in the network.
Mininet>nods	The command displays the available nodes for the default minimal topology in the current Mininet network.
Mininet>pingall	The command in the Mininet CLI triggers a ping operation from each network host to every other network host.
Mininet>h1 ping h2	The command in Mininet, you can continuously check the connectivity between Host h1 and h2.
Mininet> h1 ifconfig	The command in Mininet will display the IP address, broadcast address, and MAC address of Host h1, providing comprehensive network interface information for the host.
Mininet>h1 ip route	The command in Mininet will show the IP routing table of Host h1, providing information about the network paths used by the host to reach various

	destinations .
Mininet>h1 ping –c1 h2	The command in Mininet tests the connection between hosts h1 and h2 by sending a single packet. This command verifies if a successful communication can be established between the two hosts by exchanging a single packet of data .
Mininet>Xterm	The command allows you to connect to the terminal of Host h2, providing you with an interactive session where you can execute various commands and perform operations on Host h2 .
Mininet>exit	The command allows you to terminate or exit from a network that has been created in Mininet .

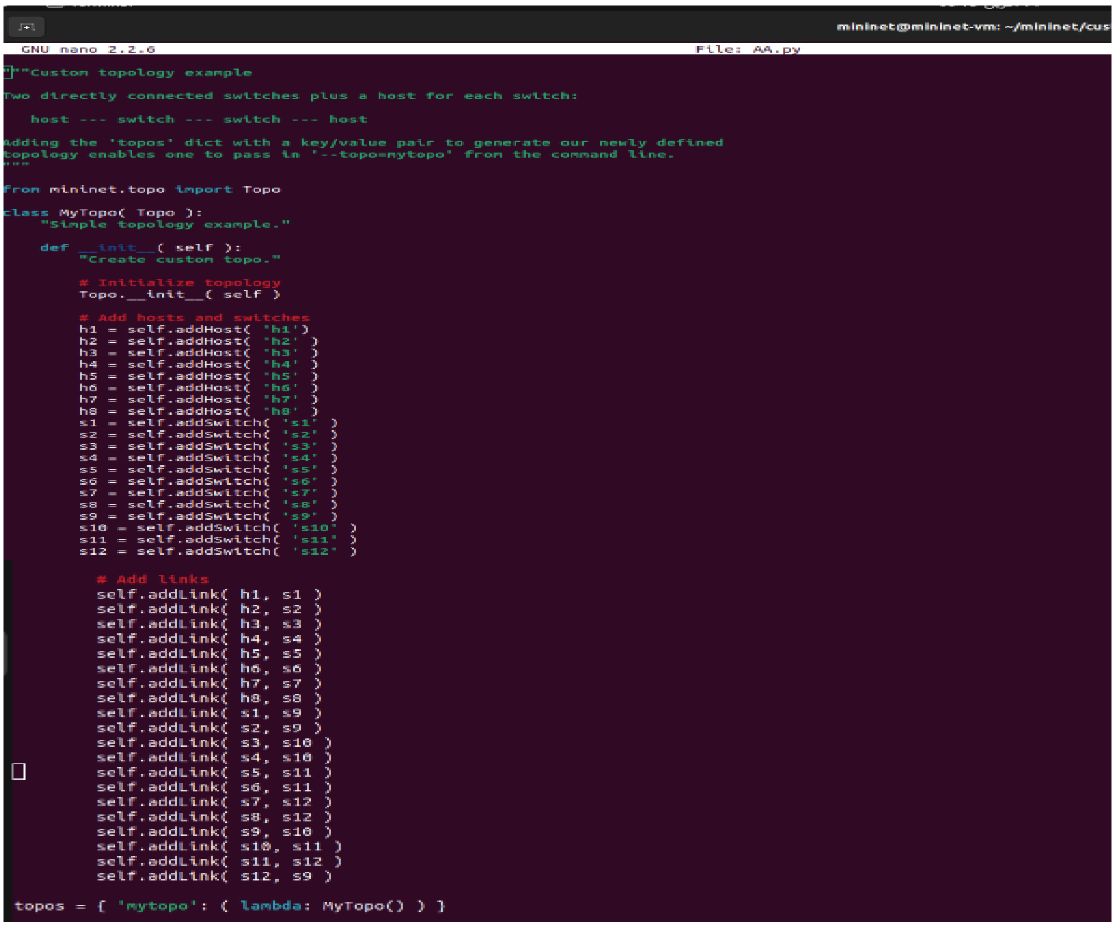
Table 3. 1 : commands one needs to know for utilizing Mininet.

3.5 Topology presentation:

In the first phrase of our project ,we have successfully installed Mininet, hpe-van sdn controller ,and Opendaylight on a Linux system running on VMware.

3.5.1 Topology presentation in mininet:

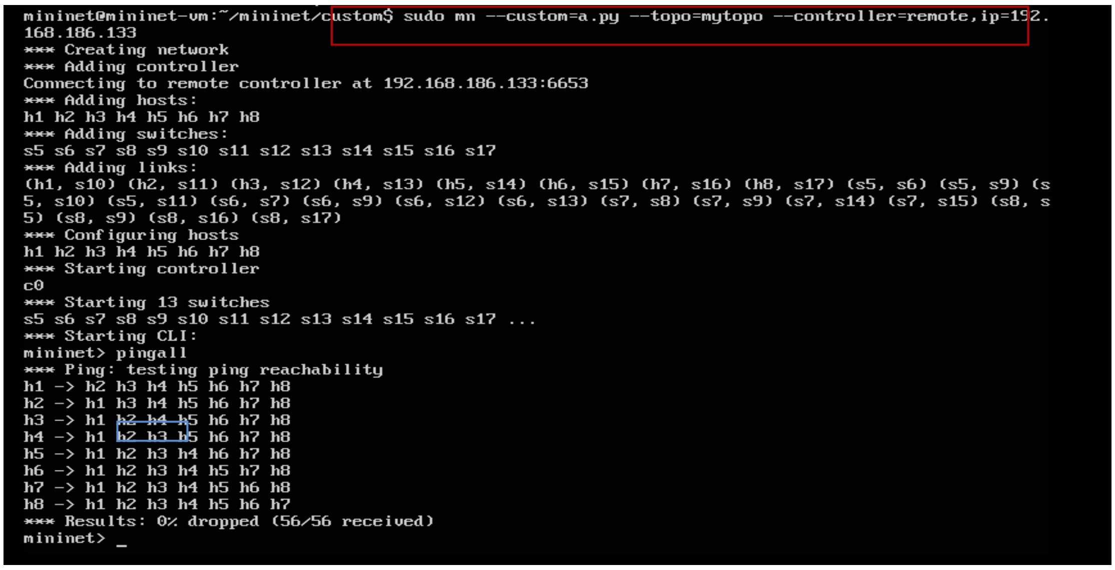
Second phrase : Create a custom topology in Mininet :



```
mininet@mininet-vm: ~/mininet/cus
GNU nano 2.2.6                                    File: AA.py

"""Custom topology example

Two directly connected switches plus a host for each switch:

   host --- switch --- switch --- host

Adding the 'topos' dict with a key/value pair to generate our newly defined
topology enables one to pass in '--topo=mytopo' from the command line.
"""

from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        h1 = self.addHost( 'h1')
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        h4 = self.addHost( 'h4' )
        h5 = self.addHost( 'h5' )
        h6 = self.addHost( 'h6' )
        h7 = self.addHost( 'h7' )
        h8 = self.addHost( 'h8' )
        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )
        s3 = self.addSwitch( 's3' )
        s4 = self.addSwitch( 's4' )
        s5 = self.addSwitch( 's5' )
        s6 = self.addSwitch( 's6' )
        s7 = self.addSwitch( 's7' )
        s8 = self.addSwitch( 's8' )
        s9 = self.addSwitch( 's9' )
        s10 = self.addSwitch( 's10' )
        s11 = self.addSwitch( 's11' )
        s12 = self.addSwitch( 's12' )

         # Add links
         self.addLink( h1, s1 )
         self.addLink( h2, s2 )
         self.addLink( h3, s3 )
         self.addLink( h4, s4 )
         self.addLink( h5, s5 )
         self.addLink( h6, s6 )
         self.addLink( h7, s7 )
         self.addLink( h8, s8 )
         self.addLink( s1, s9 )
         self.addLink( s2, s9 )
         self.addLink( s3, s10 )
         self.addLink( s4, s10 )
         self.addLink( s5, s11 )
         self.addLink( s6, s11 )
         self.addLink( s7, s12 )
         self.addLink( s8, s12 )
         self.addLink( s9, s10 )
         self.addLink( s10, s11 )
         self.addLink( s11, s12 )
         self.addLink( s12, s9 )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Figure3. 10 :Create a custom Topology in Mininet .

In this command running Mininet after topology creat and another command will cause triggers a ping operation from each network host to every other network host .



```
mininet@mininet-vm:~/mininet/custom$ sudo mn --custom=a.py --topo=mytopo --controller=remote,ip=192.
168.186.133
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.186.133:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17
*** Adding links:
(h1, s10) (h2, s11) (h3, s12) (h4, s13) (h5, s14) (h6, s15) (h7, s16) (h8, s17) (s5, s6) (s5, s9) (s
5, s10) (s5, s11) (s6, s7) (s6, s9) (s6, s12) (s6, s13) (s7, s8) (s7, s9) (s7, s14) (s7, s15) (s8, s
5) (s8, s9) (s8, s16) (s8, s17)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 13 switches
s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet> _
```

Figure3. 11 :Running and ping Topology in Mininet .

3.5.2 Topology presentation in OpenDayLight :

Fourth phrase: The following result within the opendaylight :

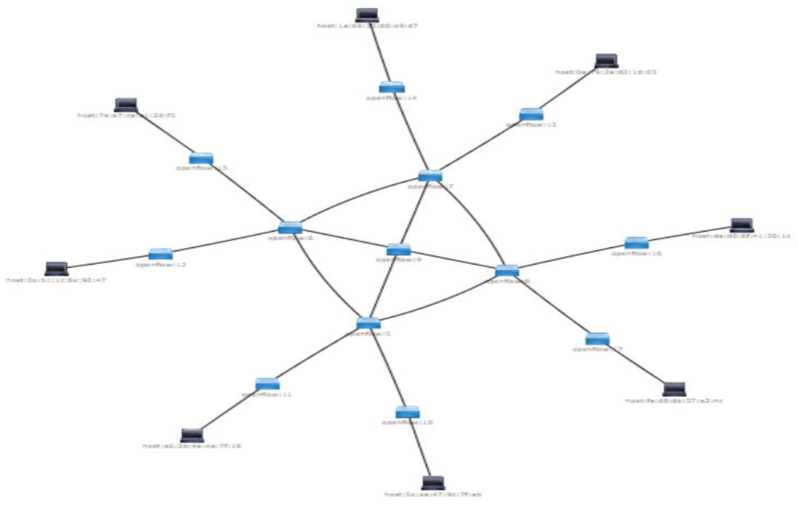
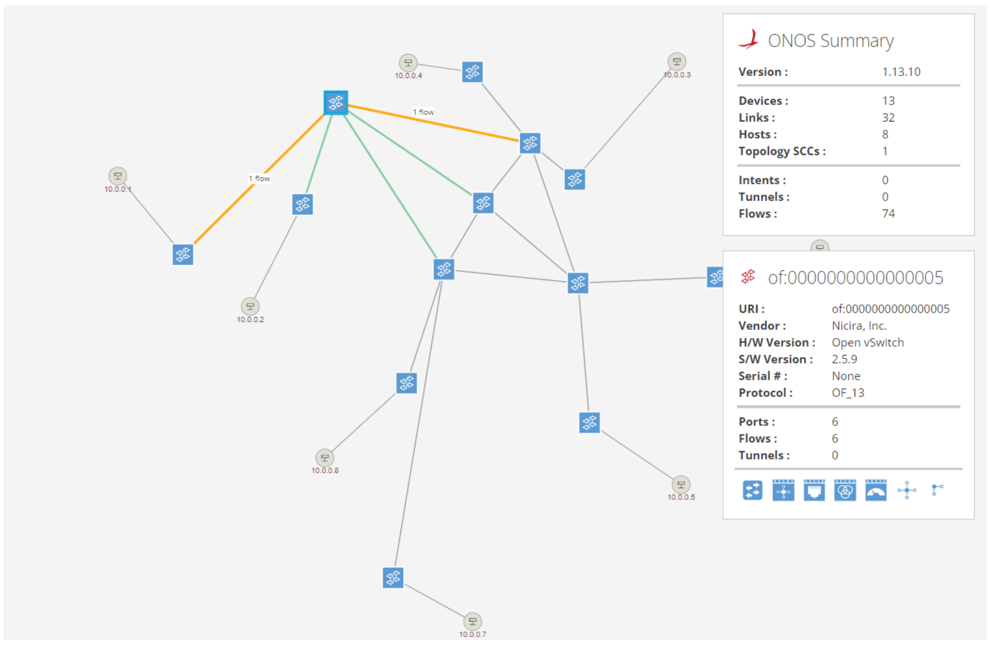


Figure3. 12: Topology Presentation Opendaylight.

3.5.3 Topology presentation in Onos :

This topology in ONOS :



ONOS Summary

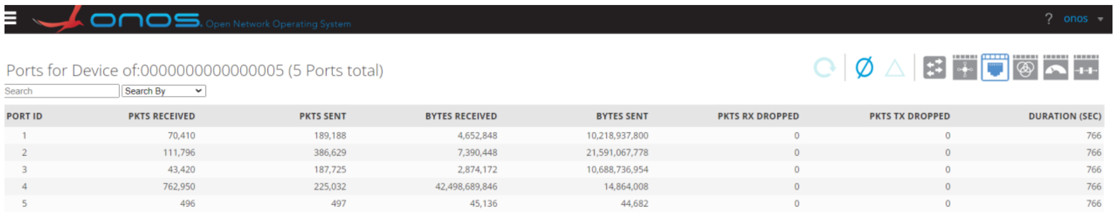
Version :	1.13.10
Devices :	13
Links :	32
Hosts :	8
Topology SCCs :	1
Intents :	0
Tunnels :	0
Flows :	74

of:0000000000000005

URI :	of:0000000000000005
Vendor :	Nicira, Inc.
H/W Version :	Open vSwitch
S/W Version :	2.5.9
Serial # :	None
Protocol :	OF_13
Ports :	6
Flows :	6
Tunnels :	0

Figure3. 13 : Topology presentation in ONOS.

Flow table of Onos :



Ports for Device of:0000000000000005 (5 Ports total)

PORT ID	PKTS RECEIVED	PKTS SENT	BYTES RECEIVED	BYTES SENT	PKTS RX DROPPED	PKTS TX DROPPED	DURATION (SEC)
1	70,410	189,188	4,652,848	10,218,937,800	0	0	766
2	111,796	386,629	7,390,448	21,591,067,778	0	0	766
3	43,420	187,725	2,874,172	10,688,736,954	0	0	766
4	762,950	225,032	42,498,689,846	14,864,008	0	0	766
5	496	497	45,136	44,682	0	0	766

Figure3. 14 : Flow table of Onos .

3.5.4 Topology presentation in Hpe-Van :

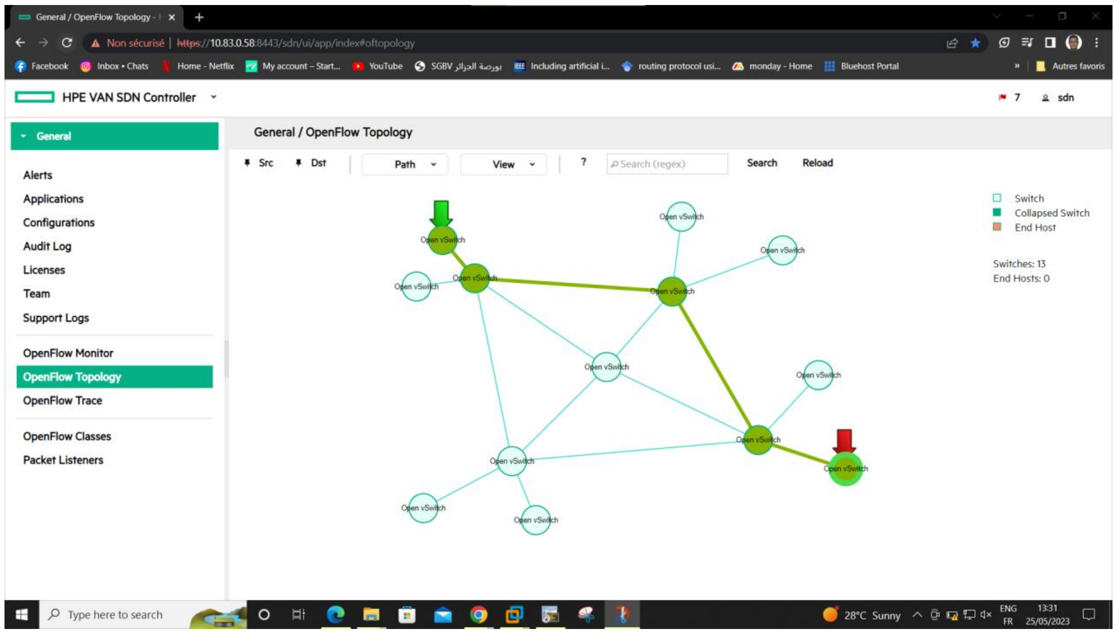
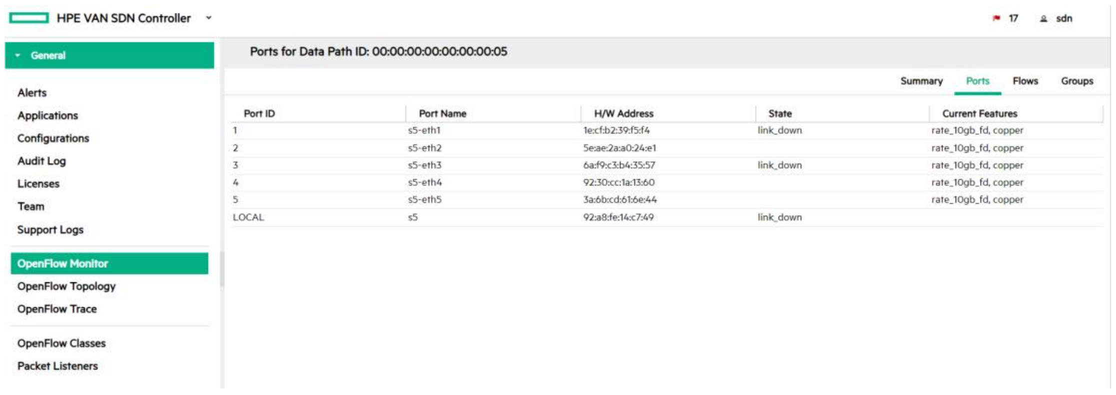


Figure3. 15 : Topology presentation in Hpe-Van.

Flow table of Hpe-Van :



Ports for Data Path ID: 00:00:00:00:00:00:00:05

Port ID	Port Name	H/W Address	State	Current Features
1	s5-eth1	1e:cf:b2:39:f5:f4	link_down	rate_10gb_fd, copper
2	s5-eth2	5e:ae:2a:a0:24:e1		rate_10gb_fd, copper
3	s5-eth3	6a:f9:c3:b4:35:57	link_down	rate_10gb_fd, copper
4	s5-eth4	92:30:cc:1a:13:60		rate_10gb_fd, copper
5	s5-eth5	3a:6b:cd:61:6e:44		rate_10gb_fd, copper
LOCAL	s5	92:a8:fe:14:c7:49	link_down	

Figure3. 16 :Flow table of Hpe-Vane .

3.6 The Test :

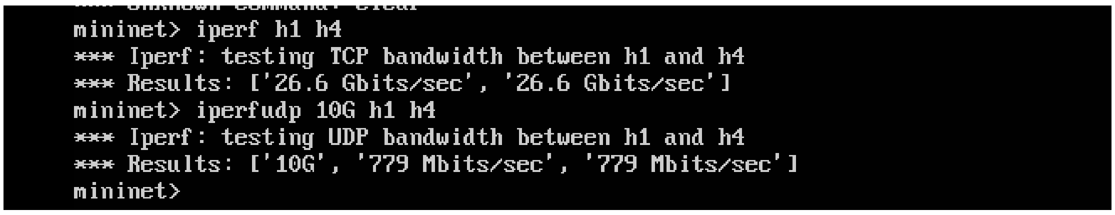
Performance evaluation of software-Defined networks controller:

3.6.1 Iperf toll :

Iperf is a widely used open-source tool that allows network administrators to measure the bandwidth and assess the performance and quality of a network connection between two hosts. It provides a simple yet powerful way to test network throughput and diagnose potential issues.

3.6.2 Time testing in sdn

1. Measuring bandwidth and evaluating network performance between host 1 and host4.



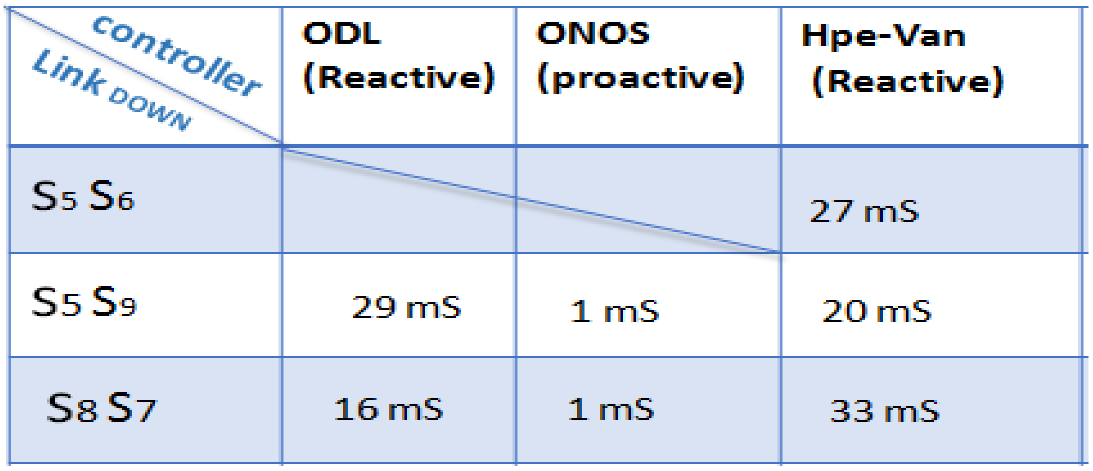
```
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['26.6 Gbits/sec', '26.6 Gbits/sec']
mininet> iperfudp 10G h1 h4
*** Iperf: testing UDP bandwidth between h1 and h4
*** Results: ['10G', '779 Mbits/sec', '779 Mbits/sec']
mininet>
```

Figure3. 17: Measuring the badwidth .

2. When measuring the network bandwidth and analyzing the flow table for each controller, we obtained the following results.

$time = \Sigma(recover\ time - fail\ time)\ /\ number\ of\ affeccted\ flows.$

3.7 Results are shown in a table:



Controller / Link DOWN	ODL (Reactive)	ONOS (proactive)	Hpe-Van (Reactive)
S5 S6			27 mS
S5 S9	29 mS	1 mS	20 mS
S8 S7	16 mS	1 mS	33 mS

Table 3. 2: The results of the comparison betwenn controller response time.

3.7.1 Results were also represented in columns :

1. Columns :

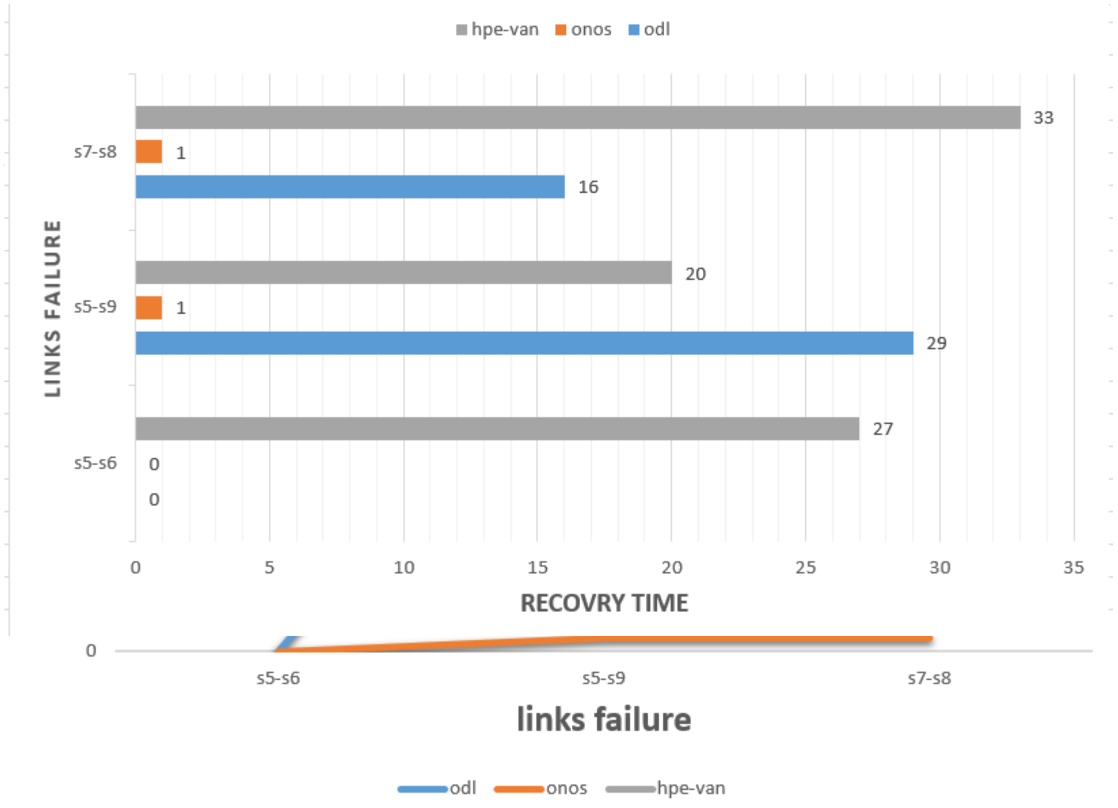


Figure3. 18 : The results of the comparison betwenn controller response time.

Comparison :

In a comparative study between different SDN controllers, when a single link is cut in a topology and a packet is sent, the ODL and ONOS controllers were not affected, but the HPE-VAN controller was significantly affected. However, when multiple links are cut and a packet is sent, both ODL(reactive) and HPE-VAN(reactive) are affected to a greater extent compared to ONOS. It is observed that ONOS does not take much time in selecting an alternate path for packet transmission because it has a routing table to guide it, whereas ODL sends a message to the controller to provide an alternative path, which takes more time .

In conclusion, we have noticed that ONOS (proactive) does not take much time to send the packet and select an alternate path .

3.8 CONCLUSION

In this chapter , we extensively discussed our working environment, explained the structure of our proposed project, and provided a detailed overview of the stages we went through to obtain the results of comparing the time between controllers.

GENERAL CONCLUSION

In our project, aimed explore the potential of Software-Defined Networking (SDN) as a replacement for traditional network infrastructure. The rapid advancement of SDN technology served as the driving force behind our study. The objective of this work is to evaluate the performance of SDN controllers .

To begin, we conducted a comparative analysis between SDN and traditional networks, highlighting the advantages that SDN offers in the business environment. We introduced OpenvSwitch as the main engine driving the functionalities available in SDN .

We then focused on the crucial aspect of fault tolerance in SDN. We explored various techniques and mechanisms to ensure efficient fault tolerance in SDN environments .

After studying , we chose Mininet, a widely-used open-source software that enables the emulation of SDN networks. We employed controllers such as HPe-van, ODL, and ONOS to evaluate their performance in the SDN context. These controllers provided the necessary functionality to manage and control the SDN network .

At last ,Through our experiments, simulations, and performance evaluations, we gained valuable insights into the recovery time of each mentioned controller. This allowed us to observe the differences between them and make informed comparisons .

These points can be considered as potential future work for this thesis :

- Conducting experiments with more complex scenarios would be a logical progression for this research .
- Exploring the integration of the proposed mechanism with interesting machine learning algorithms could lead to enhanced capabilities and improved performance .
- Updating the proposed mechanism to reduce retrieval time is an important aspect to consider .

It is worth noting that software-based knowledge networks still face several challenges. These challenges could include ensuring accuracy and reliability of the knowledge base, addressing scalability issues as the size of the knowledge network grows, and improving fault tolerance to handle errors and failures effectively .

BIBLIOGRAPHY

[1] K. Greene, 'TR10: Software-Defined Networking', *MIT Technology Review*. [online]. Disponible sur: http://www2.technologyreview.com/news/412194/tr10-software-defined-networking/. [Consulté le: 27-août-2019].

[2] D. B. Rawat et S. R. Reddy, " Software Defined Networking Architecture, Security and Energy Efficiency: A Survey", *IEEE Commun. Surv. Tutor.*, vol. 19, nº 1, p. 325□346, Firstquarter 2017.

[3] D. B. Hoang, " Software Defined Networking ? Shaping up for the next disruptive step? ",*Aust. J. Telecommun. Digit. Econ.*, vol. 3, nº 4, nov. 2015

[4]' OpenFlow Switch Specification - PDF ».[online]. Disponiblesur:https://www.opennetworking.org/wpcontent/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[5] B. Pfaff et B. Davie, « The OpenvSwitch Database Management Protocol ». [En ligne]. Disponible sur: https://tools.ietf.org/html/rfc7047. [Consulté le: 27-août-2019].

[6] "Is PCEP/BGP-LS based SDN Approach Ideal Choice for Service Providers?" https://www.keysight.com/blogs/tech/traf-gen/2022/06/28/is-pcepbgp-ls-based-sdn-approach-ideal-choice-for-service-providers

[7] Arif Rubayet "Cross-layer design in Software Defined Networks(SDNs):issues and possible solution" April 2021.

[8] Howard "Open Flow Switch : What Is It and How Does It Work|OpenFlow Switch Components " Updated on Jun 1,2021

[9] "Ryu," [Online]. Available: https :// osrg. github.io/ryu./

[10] "Open Network Operating System (ONOS)," [Online]. Available : https :// wiki .onosproject.org/display/ONOS/Wiki+Home.

[11] Opendaylight," [Online]. Available: https :// www.opendaylight.org

[12] Muhammad Anan,Husnain Bustam "Empowering networking research and experimentation through Software-Defined Networking |Journal of network and Computer Application,2016".

[13] "What Are SDN Controllers (or SDN Controller Platforms)?"January 14,2014.

https://www.sdxcentral.com/networking/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/what-is-sdn-controller/sdn-controllers/

[14] Miriam A.M.Capreetz "An approach for SDN traffic monitoring based on big data technique |Jornal of Network and Computer Applications ,Volume 131,1 April 2019,Pages28-39".
https://www.sciencedirect.com/science/article/pii/S1084804519300244

[15]"SOFTWARE DEFINED NETWORK (SDN) ROUTING".https://techex.co.uk/ipproduction/software-defined-network-sdn-routing.

[16] Van Steen, Maarten, and Andrew S. Tanenbaum. *Distributed systems*. Leiden, The Netherlands: Maarten van Steen, 2017.

[17] Avizienis, Algirdas, et al. "Basic concepts and taxonomy of dependable and secure computing." *IEEE transactions on dependable and secure computing* 1.1 (2004): 11-33.

[18] Hukerikar, Saurabh, and Christian Engelmann. "Resilience design patterns-a structured approach to resilience at extreme scale (version 1.0)." *arXiv preprint arXiv:1611.02717* (2016).

[19] leander Seidlittw ,Cora Perner "Fault tolerance in SDN " chair of netzork architecture and service ".

[20] Jaiswal, Rituka, Reggie Davidrajuh, and Chunming Rong. "Fog Computing for Realizing Smart Neighborhoods in Smart Grids." *Computers* 9.3 (2020): 76.

[21] S. Paris, G. S. Paschos, and J. Leguay, "Dynamic control for failure recovery and flow reconfiguration in SDN," in 2016 12thInternational Conference on the Design of Reliable CommunicationNetworks (DRCN). Paris: IEEE, Mar. 2016, pp. 152–159. [Online].Available: http://ieeexplore.ieee.org/document/7470850/

[22] Thorat, Pankaj, Seil Jeon, and Hyunseung Choo. "Enhanced local detouring mechanisms for rapid and lightweight failure recovery in OpenFlow networks." *Computer Communications* 108 (2017): 78-93.

[23] Ali, Jehad, et al. "Software-defined networking approaches for link failure recovery: A survey." *Sustainability* 12.10 (2020): 4255.