Security Enhancement of Open Flow Networks

Graduation Project is submitted to the Electronics and Communications Department in partial fulfilment of the requirements for the Degree of Bachelor of Electrical, Electronics and Communications Engineering

Prepared by

Khaled Elsayed Ahmed

Supervised by

Prof. Dr. Ihab Abdelwahab Ali

2024

Acknowledgments

We would like to express our sincere gratitude to God for giving us the opportunity and ability to complete this project. Many thanks to our supervisor, **Prof. Ihab Abd El Wahab Ali**, for his scientific guidance and useful advice. We are very grateful to our families and loved ones for their constant support and encouragement throughout our academic journey. We hope that this project will serve as a useful reference and source of inspiration for future students and researchers in this field.

Table of Contents

List of figures

List of tables

CHAPTER 1: A DEEP UNDERSTANDING OF SDN NETWORKS AND THEIR IMPLEMENTATION

INTRODUCTION

The rapid evolution of networking technologies has led to the emergence of Software-Defined Networks (SDNs), revolutionizing the way we design, manage, and secure modern networks. SDNs separate the control plane from the data plane, enabling centralized management and programmability. OpenFlow, as a widely adopted communication protocol, plays a crucial role in facilitating the communication between the control and data planes in SDN environments.

However, the adoption of SDNs also introduces new security challenges. The unique architecture of SDNs, with its centralized control and programmable interfaces, creates new vulnerabilities, particularly to denial-of-service (DoS) attacks. These attacks can target the controller, switches, or communication channels, potentially disrupting network operations and compromising availability. The need for robust security measures to protect these networks from a wide range of cyber threats is more critical than ever.

This project, titled "Security Enhancement of OpenFlow Networks," delves into the significance of bolstering security in OpenFlow-based SDN environments. It addresses the vulnerabilities inherent in SDN architectures and explores key measures and strategies to ensure the integrity, confidentiality, availability, and resilience of these networks. By implementing and evaluating various security enhancements, this project aims to fortify SDN architectures against potential vulnerabilities and attacks, particularly focusing on mitigating DoS threats.

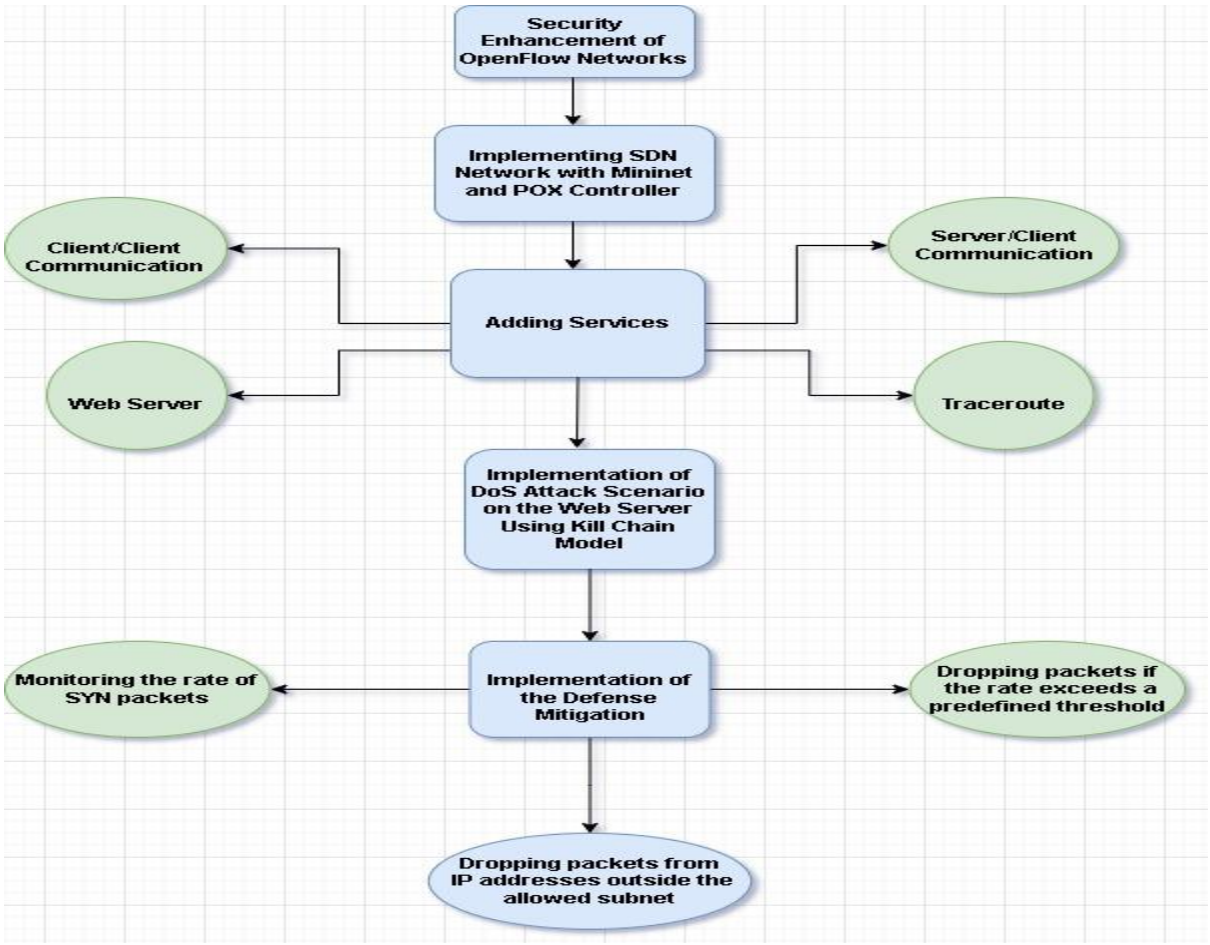


Figure 1.1. Project overview

Software-Defined Networking (SDN) Definition

Software-Defined Networking (SDN) is a relatively new paradigm aimed to solve the limitation of traditional IP networks. First, it solves the vertical integration problem by separating control logic (network control plane) form underlying hardware, responsible for data forwarding. Second, control logic is moved to a logically centralized controller (or network operating system), simplifying network configuration and policy enforcement. A simplified view of this architecture is shown in Figure 1.2. Important to mention that logically centralized models do not signify physically centralized systems. Instead, industry SDN controllers have physically distributed control planes, keeping, nevertheless, centralized logical structure.

The separation of the control plane and the data plane requires a well-defined API (Application Programming Interface) between switches and SDN controller, as depicted on Figure 1.2. The most

recognizable of these interfaces now is OpenFlow. According to it OpenFlow switch can have one or several tables of packet-handling rules (flow tables), each of them matches a subset of traffic and can perform actions like dropping, forwarding, modifying on the incoming packets.

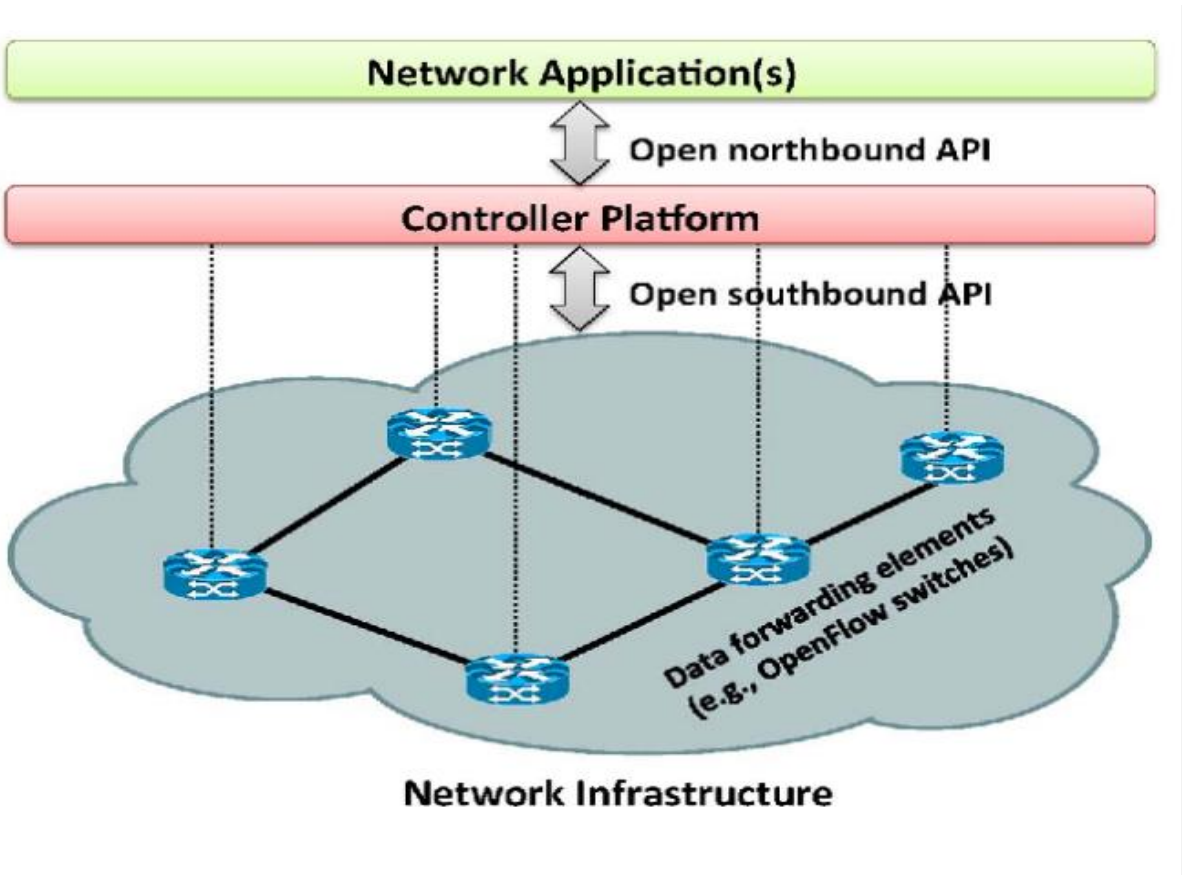


Figure 1.2. SDN architecture

With different sets of rules, installed by controller, an OpenFlow switch can behave like a router, switch, firewall, load balancer, traffic shaper, and any other middlebox. In result of application software-defined networking principles can me separated concerns of defining network policies, their implementation in hardware, and traffic forwarding. This separation is the key to desired flexibility, because network control is now broken into tractable pieces, which allows us to create new abstractions in networking, simplify network management and introduce innovations.

SDN pillars/Attributes

The term SDN (Software-Defined Networking) was originally coined to represent the ideas and work around OpenFlow at Stanford University. The original definition of term SDN refers to a network architecture where the forwarding state of data plane is managed by a remote-control plane, separated from the former. The networking industry, in many cases, shifted from original view, calling SDN

anything involves software. In this work we will refer SDN as described in, as a network architecture with four pillars:

1. The control and data planes are decoupled. Control functionality is removed from network devices that will become simple (packet) forwarding elements.

2. Forwarding decisions are flow-based, instead of destination-based. A flow is broadly defined by a set of packet field values acting as a match (filter) criterion and a set of actions (instructions). In the SDN/OpenFlow context, a flow is a sequence of packets between a source and a destination. All packets of a flow receive identical service policies at the forwarding devices. The flow abstraction allows unifying the behavior of different types of network devices, including routers, switches, firewalls, and middleboxes. Flow programming enables unprecedented flexibility, limited only to the capabilities of the implemented flow tables.

3. Control logic is moved to an external entity, the so-called SDN controller or Network Operating System (NOS). The NOS is a software platform that runs on commodity server technology and provides the essential resources and abstractions to facilitate the programming of forwarding devices based on a logically centralized, abstract network view. Its purpose is therefore like that of a traditional operating system.

4. The network is programmable through software applications running on top of the NOS that interact with the underlying data plane devices. This is a fundamental characteristic of SDN, considered as its main value proposition.

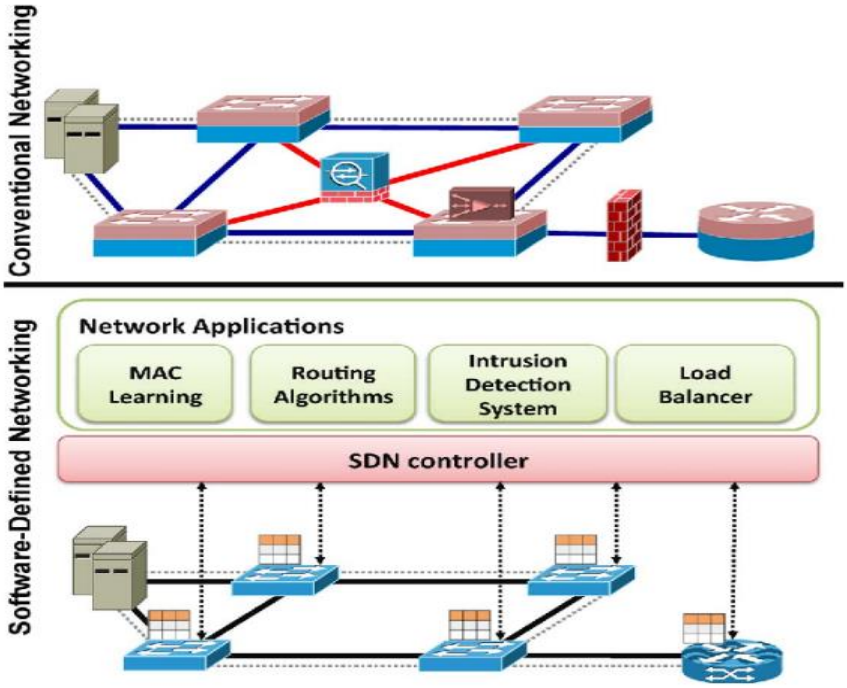


Figure 1.3. Traditional network vs SDN

SDN ARCHITECTURE

Notably, the logical centralization of the control logic has several additional benefits. First, it allows to modify network policies in simpler and less error-prone way by means of high-level languages and software components, comparing to low-level vendor specific device configurations. Second, a control program can automatically react to changes of the state of network and maintain the high-level policies in any circumstances, including device/link outages and spikes of the data traffic. Third, the centralization of the control logic in a controller gives global knowledge of the state of the network, and therefore simplifies the development of sophisticated networking functions, services and applications. Following the SDN concept introduced by Scot Schenker (A researcher at the International Computer Science Institute who's enhance the SDN technology) an SDN can be defined by three fundamental abstractions: (i) forwarding, (ii) distribution, and (iii) specification.

i. The forwarding abstraction should allow any forwarding behavior desired by the control program (network application), at the same time hiding details of the underlying hardware. OpenFlow is one realization of such abstraction, which can be considered as an equivalent to a "device driver" in an operating system.
ii. The distribution abstraction should protect SDN applications from the problems of distributed state, making the distributed control problem a logically centralized one. Its realization requires a common distribution layer, which in SDN resides in the NOS. This layer has two essential

functions. First, it is responsible for installing the control commands on the forwarding devices. Second, it gathers information about the forwarding layer (network devices and links), to offer a global network view to network applications.

iii. The specification abstraction allows a control program (network application) to express the desired network behavior without about implementation of that behavior. This goal can be reached with virtualization solutions, or network programming languages.

These abstractions allow us to convert the abstract configurations that the applications express based on a simplified, abstract model of the network, into a physical configuration for the global network view exposed by the SDN controller. Figure 1.4 depicts the SDN architecture, concepts and building blocks. As was mentioned before, the tight binding between control and data planes makes the deployments of new networking features (for example, routing protocols) very hard, due to the fact that we need to modify all networking devices. Therefore, as was mentioned before, new features are introduced mostly by means of new middleboxes, which are, in turn, vendor dependent and hard to configure. Resulting topology becomes more complex and creates additional hurdles in implementing new functionality. In contrast, SDN separates the control plane from the network devices and becomes an external entity: the network operating system or SDN controller.

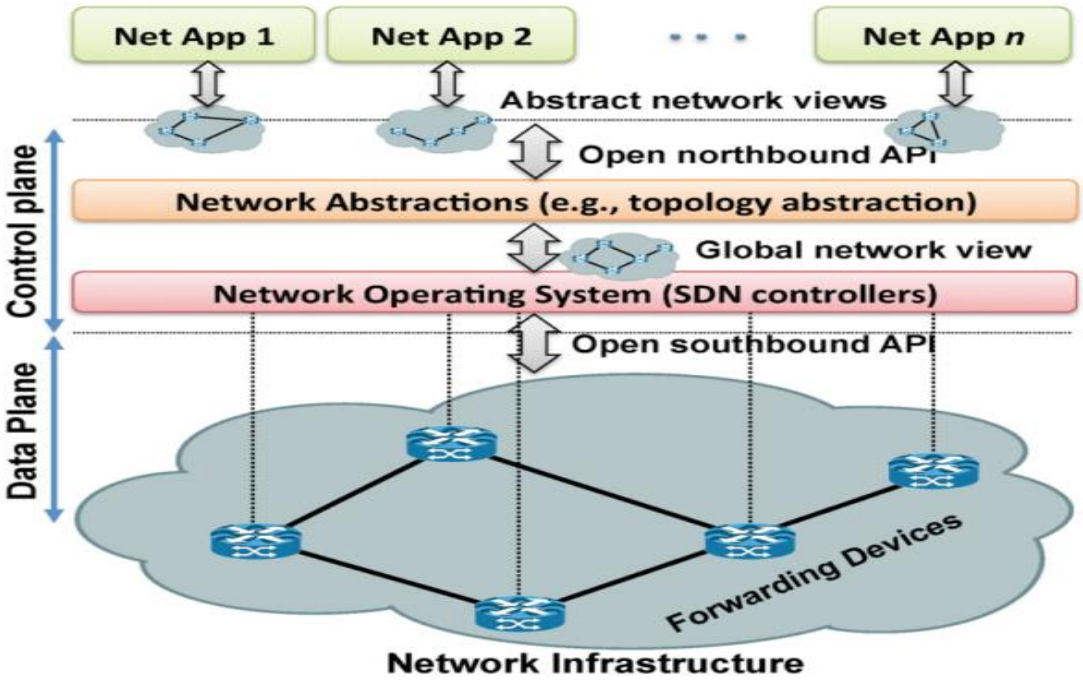


Figure 1.4. SDN Abstraction

This approach has several advantages:

• Provides a simple and cheap network testbed for developing SDN applications. It becomes easier to program these applications because the abstractions of the control platform and network programming languages can be moved and shared.

• All applications can use the same information (the global network view), making more effective policy decisions using control plane software modules.

• These applications can take actions (i.e., reconfigure forwarding devices) from any part of the network, eliminating problem of choosing the location for the new functionality.

• The integration of different applications becomes more straightforward. For example, load balancing and routing applications can be used simultaneously, with load balancing decisions having precedence over routing policies.

Here is another simple graph to understand the separation process between layers and to study every block in the architecture presented by The Open Networking Foundation (ONF) who is taking the lead in SDN standardization.

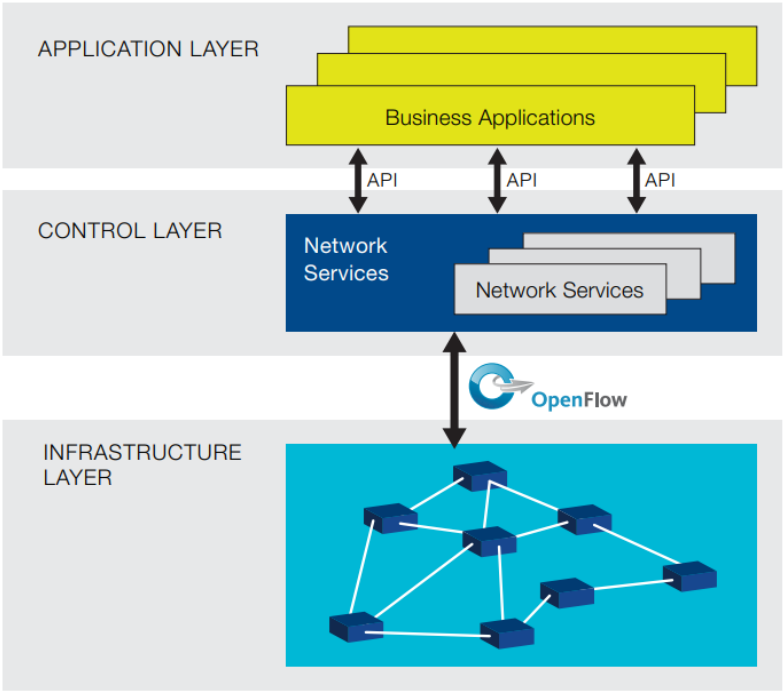


Figure 1.5. ONF/SDN architecture

The ONF/SDN architecture consists of three distinct layers that are accessible through open APIs:

• The Application Layer consists of the end-user business applications that consume the SDN communications services. The boundary between the Application Layer and the Control Layer is traversed by the northbound API.

• The Control Layer provides logically centralized control functionality that supervises the network forwarding behavior through an open interface.

• The Infrastructure Layer consists of the network elements (NE) and devices that provide packet switching and forwarding.

• The interaction between the control plane component and forwarding devices is known as the southbound interface. The data plane's connection with the control element is formalized by the communication protocol.

• The northbound API which allows communication between the management and control layers via northbound interfaces. The network topology, provisioning tasks, and configuration retrieval are all available through this northbound interface. Through the provisioning northbound API, it also provides network, loop avoidance, path calculation, security, and routing, as well as fault management.

After explaining the SDN architecture let's discuss the real-life benefits of it to see the big difference between it and the traditional network and how it achieves greater network efficiency, scalability, and agility.

1.3 SDN BENEFITS

Here are some real-life benefits of SDN technology:

1. Network Virtualization and Multi-Tenancy: SDN allows for network virtualization, enabling the creation of multiple virtual networks on a shared physical infrastructure. This is particularly beneficial in cloud environments and service providers, as it allows for secure multi-tenancy, isolation, and customized network services for different users or applications.

2. Data Center Automation: SDN simplifies data center management by automating repetitive tasks and providing centralized control. Data center operators can provision and manage network resources programmatically, leading to faster service delivery, improved resource utilization, and reduced operational costs.

3. Dynamic Traffic Engineering and Load Balancing: With SDN, traffic can be dynamically routed, and load balanced across the network based on real-time conditions and policies. This enables efficient utilization of network resources, reduces congestion, and optimizes performance for different applications and services.

4. Rapid Service Deployment and Innovation: SDN's programmable nature allows for the rapid deployment of new services and applications. Network administrators can define service requirements through software, eliminating the need for manual configuration of individual network devices. This agility fosters innovation and enables organizations to respond quickly to changing business needs.

5. Network Service Chaining: SDN facilitates the chaining of network services such as firewalls, intrusion detection systems, and load balancers, by directing traffic through a predefined sequence of network functions. This simplifies service deployment, improves security, and enhances the overall performance and reliability of the network.

6. Enhanced Network Visibility and Troubleshooting: SDN provides granular visibility into network traffic and performance through centralized monitoring and analytics. Network administrators can gain real-time insights into network behavior, identify bottlenecks or anomalies, and troubleshoot issues more efficiently, leading to improved network reliability and reduced downtime.

7. Software-Defined WAN (SD-WAN): SDN technology has been instrumental in the development of SD-WAN solutions. SD-WAN simplifies the management and operation of wide area networks by decoupling the control and data planes, enabling centralized management and policy-based routing. This results in cost savings, improved application performance, and enhanced network agility.

8. Cloud Networking: SDN is well-suited for cloud environments, allowing for dynamic provisioning and orchestration of network resources. It enables seamless integration between virtualized network functions and cloud platforms, enhancing the scalability, flexibility, and security of cloud-based services.

9. Internet of Things (IoT) Support: SDN provides a scalable and flexible infrastructure to support the growing number of IoT devices. By enabling centralized management and control, SDN simplifies

IoT device onboarding, enhances security, and ensures efficient communication and resource allocation for IoT applications.

10. Network Resilience and Disaster Recovery: SDN offers improved network resilience through features such as automatic network reconfiguration and fast failure detection. In the event of a network failure or disaster, SDN allows for rapid reconfiguration and redirection of traffic, minimizing service disruptions and improving overall network availability.

11. Policy-Based Network Management: SDN enables policy-based network management, where network administrators can define and enforce policies that govern network behavior. Policies can be based on factors such as application requirements, user roles, or security needs. This approach simplifies network management and ensures consistent application of policies across the network.

12. Scalability and Elasticity: SDN provides scalability and elasticity by separating the control plane from the data plane. This separation allows for the dynamic allocation of network resources based on demand. As network requirements change, resources can be scaled up or down, providing flexibility and cost optimization.

13. Improved Network Efficiency: SDN enhances network efficiency by optimizing traffic flow and reducing network congestion. With centralized control, traffic can be intelligently routed based on real-time conditions, such as available bandwidth or network performance metrics. This improves overall network performance and user experience.

14. Service-Level Agreement (SLA) Assurance: SDN enables the implementation of SLAs by allowing administrators to define and enforce quality of service (QoS) parameters. SLAs can specify bandwidth, latency, or packet loss requirements for different applications or user groups. SDN's ability to prioritize and allocate network resources according to SLAs ensures that critical applications receive the necessary network performance guarantees.

15. Simplified Network Provisioning and Management: SDN simplifies network provisioning and management through automation and abstraction. With a centralized controller, network administrators can define network policies and configurations in software, which are then automatically deployed across the network. This automation reduces manual configuration efforts and minimizes the risk of human errors.

16. Seamless Network Integration and Migration: SDN technology facilitates the integration and migration of diverse network technologies and architectures. It allows organizations to adopt new networking paradigms, such as virtualized network functions (VNFs), without disrupting existing network infrastructure. SDN's programmable nature enables the seamless integration of different network components and protocols.

17. Cost Savings and Resource Optimization: SDN can lead to cost savings by optimizing resource utilization and reducing hardware dependencies. With SDN, organizations can leverage commodity hardware instead of relying on expensive proprietary network devices. Additionally, centralized control and automation reduce operational costs by streamlining network management and troubleshooting processes.

18. Vendor-Neutral Solutions and Interoperability: SDN promotes vendor neutrality and interoperability through open standards and APIs. It allows organizations to choose best-of-breed solutions from different vendors, avoiding vendor lock-in. SDN's open nature fosters innovation and collaboration among vendors, leading to a diverse ecosystem of compatible network components.

19. Compliance and Regulatory Requirements: SDN technology can help organizations meet compliance and regulatory requirements more effectively. With centralized control and policy-based management, it becomes easier to enforce security policies, monitor network activity, and generate audit logs. This aids in achieving regulatory compliance and simplifying the auditing process.

20. Future-Proofing the Network: SDN provides a future-proof foundation for network infrastructure. Its programmable nature and flexibility allow for easy adaptation to emerging technologies, protocols, and requirements. SDN's ability to evolve and accommodate new networking innovations ensures that organizations can stay at the forefront of technological advancements.

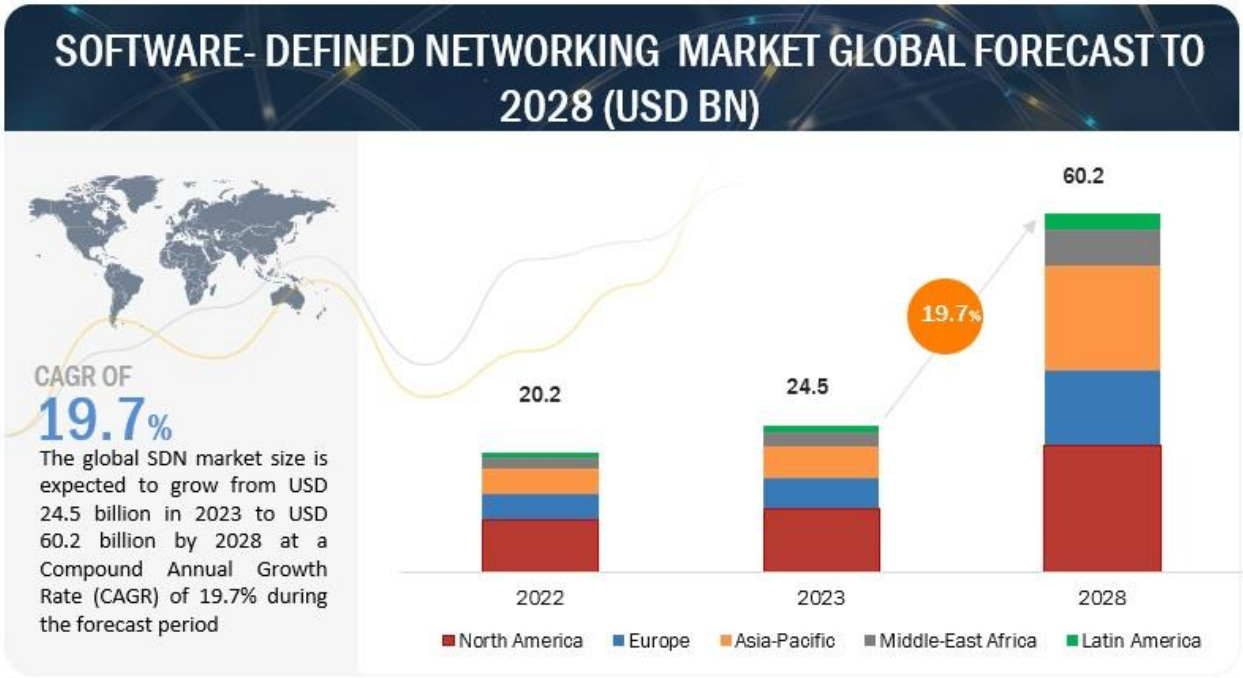


Figure 1.6. Market size from 2022 to 2028

These benefits highlight the versatility and transformative potential of SDN technology, and there are more than that. By leveraging the benefits of SDN, organizations can achieve greater network efficiency, scalability, and agility while reducing costs and simplifying network management.

From here we start talking about NFV (Network Functions Virtualization) to discuss how to implement a virtual and real SDN network with centralized controller or multi controllers as we discussed before.

1.4 NFV (NETWORK FUNCTIONS VIRTUALIZATION)

It is a concept that aims to decouple network functions from dedicated hardware appliances and instead virtualize them on standard servers, storage, and switches. Traditional network functions, such as firewalls, load balancers, and intrusion detection systems, are transformed into software-based virtual network functions (VNFs) that can be deployed and managed in a more agile and dynamic manner.

By virtualizing network functions, NFV offers several benefits. First, it enables organizations to consolidate multiple network functions onto a shared hardware infrastructure, reducing complexity and hardware costs. Second, NFV allows for on-demand provisioning and scalability of network services, enabling rapid deployment and scaling of VNF instances as needed. Third, NFV enhances service agility by facilitating the dynamic chaining and composition of virtualized network functions, enabling organizations to create customized service chains tailored to specific application requirements.

THE IMPACT OF THE OPENFLOW PROTOCOL ON SDN TECHNOLOGY

OpenFlow Protocol:

- Meaning: An open, standardized communication protocol that enables the control of network switches and routers from a centralized controller.
- Function: Facilitates the separation of the control plane (decision-making) from the data plane (packet forwarding) in a network.
- Key Component: Serves as a pivotal southbound interface in SDN architectures, allowing the SDN controller to directly program and manage network devices.

How OpenFlow Helps in SDN Networks:

1. Centralized Control:
    - Establishes a unified point of control for the entire network.
    - Simplifies network management and policy enforcement.
    - Overcomes vendor lock-in by supporting devices from multiple vendors.
2. Dynamic Programmability:
    - Grants SDN controllers the ability to modify network behavior dynamically through software.
    - Enables rapid deployment of new services and features without manual configuration of individual devices.
3. Enhanced Flexibility:
    - Facilitates adaptive traffic engineering and load balancing based on real-time network conditions.
    - Improves network utilization and performance.
4. Simplified Network Management:
    - Abstracts the complexity of the underlying infrastructure.
    - Enables automation of network tasks and reduces operational overhead.
5. Innovation and Experimentation:
    - Provides a platform for testing and developing new network applications and protocols.
    - Accelerates innovation in networking technologies.
6. Security:
    - Provides granular control over traffic flows.
    - Enables proactive security measures and threat mitigation.
7. Virtualization:
    - Supports virtualized network environments, essential for cloud computing and NFV.

1.5 OPENVSWITCH (OVS)

Open vSwitch (OVS) plays a crucial role in Mininet, acting as its virtual switch component. Let's explore its purpose and benefits:

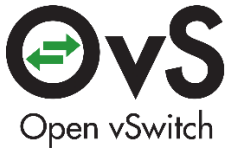


Figure 1.7. OVS

What is Open vSwitch?

- Open vSwitch is a multi-layer virtual switch that runs in user space or kernel space.
- It emulates the behavior of physical network switches within a virtual environment.
- It supports standard switching protocols like OpenFlow, NetFlow, and LACP.

How is OVS used in Mininet?

- Mininet relies on OVS to connect virtual hosts and switches within its simulated network.
- OVS forwards packets between these virtual elements, mimicking real network behavior.
- Users can configure OVS using OpenFlow, allowing them to control traffic flow and implement custom functionalities.

Benefits of using OVS in Mininet:

- Flexibility: OVS offers numerous configuration options, enabling users to design complex network topologies and experiment with diverse protocols and features.
- Speed: Running in user space or kernel space, OVS offers faster performance compared to emulated switches in some situations.
- Openness: Being open-source, OVS is readily available, customizable, and actively maintained by a large community.
- Interoperability: Support for OpenFlow allows seamless integration with various SDN controllers and tools.
- Realism: OVS closely resembles real network switches, making Mininet simulations more realistic and valuable for testing and research.

In summary, Open vSwitch provides a powerful and versatile virtual switch for Mininet, enabling users to build and experiment with complex network scenarios with impressive speed, openness, and interoperability.

1.6 PROJECT IMPLEMENTATION WITH MININET AND POX CONTROLLER

In the context of implementing NFV and SDN, the combination of Mininet and the POX controller provides a powerful platform for network experimentation, prototyping, and testing. Mininet is a network emulator that allows the creation of virtual network topologies using lightweight virtual machines. It enables the simulation of complex network environments, facilitating the evaluation and validation of new network architectures and protocols.

POX, on the other hand, is an open-source SDN controller that provides a flexible framework for developing custom network applications and protocols. By using Mininet and the POX controller together, researchers and network engineers can emulate and evaluate NFV and SDN concepts in a controlled environment. They can create virtual network functions (VNFs) using Mininet's virtual machines and leverage POX's programmability to control the behavior of the network and orchestrate the deployment and chaining of VNFs.

There are different controllers to use with Mininet, but we chose the pox controller to work with.

here are some reasons discusses why we choose it:

1. Flexibility and Customizability: POX provides a highly flexible and extensible framework for developing custom network applications and protocols. It allows you to have fine-grained control over the behavior of your network by writing Python code. This flexibility is particularly valuable if you have specific requirements that may not be easily addressed by other controllers.

2. Active Community and Documentation: POX has an active community of developers and users who contribute to its ongoing development and provide support. The availability of a vibrant community means that you can benefit from their expertise, share knowledge, and seek help if encountering any issues during the project. Additionally, POX has comprehensive documentation that can guide you through the implementation process and help you understand its features and capabilities.

3. Integration with Mininet: POX is designed to work well with Mininet, which is a widely used network emulator for creating virtual network topologies. The integration between Mininet and POX is seamless, making it easier to set up and manage your network experiments. This compatibility helps ensure a smooth and efficient workflow for the project.

4. Open-Source Nature: POX is an open-source controller, which means you have access to its source code and can modify it to suit your specific needs. This openness allows you to tailor the

controller to your project requirements, add new features, or fix any bugs you encounter. It also promotes transparency and fosters collaboration within the community.

5. Support for SDN Concepts: POX provides support for various SDN concepts, such as OpenFlow, which is a key protocol used to communicate between the controller and the switches in an SDN environment. POX allows us to define and enforce network policies, manage flows, and control the behavior of the network. It aligns well with the SDN principles and enables experiments with and implement SDN concepts effectively.

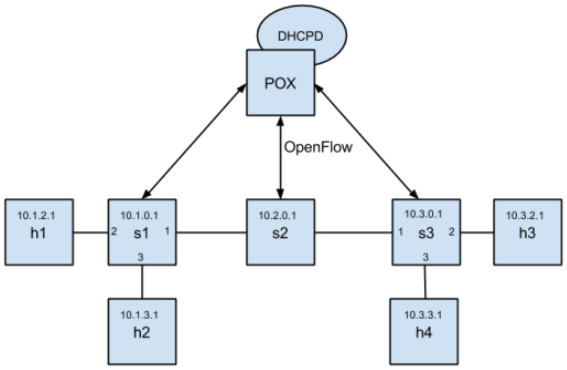


Figure 1.8. The integration between Mininet and Pox controller with OpenFlow protocol

1.7 SETTING THE STAGE FOR SDN EXPLORATION

Software-Defined Networking (SDN) empowers us to break free from the rigid confines of traditional network hardware and embrace programmatic control. To explore the exciting world of SDN, we'll dive into two powerful tools: Mininet and POX. However, before we unleash their potential, laying the groundwork is crucial. This involves establishing a suitable environment – in this case, installing Ubuntu, a versatile operating system that acts as our platform for SDN experimentation.

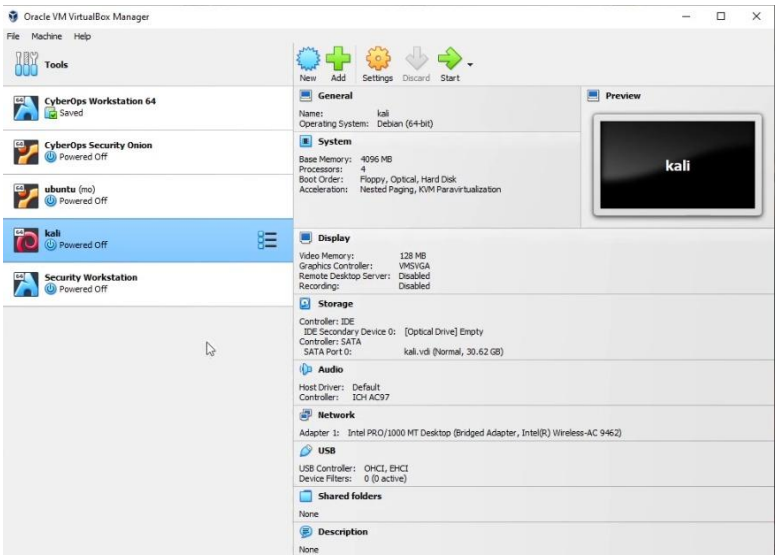
Ubuntu provides a stable, open-source foundation upon which we'll build our SDN playground. It offers robust support for the Python programming language, a key ingredient for both Mininet and POX. With Ubuntu securely in place, we can seamlessly proceed to install and configure these tools, empowering ourselves to delve into the fascinating realm of network programmability.

1.8 INSTALLING UBUNTU

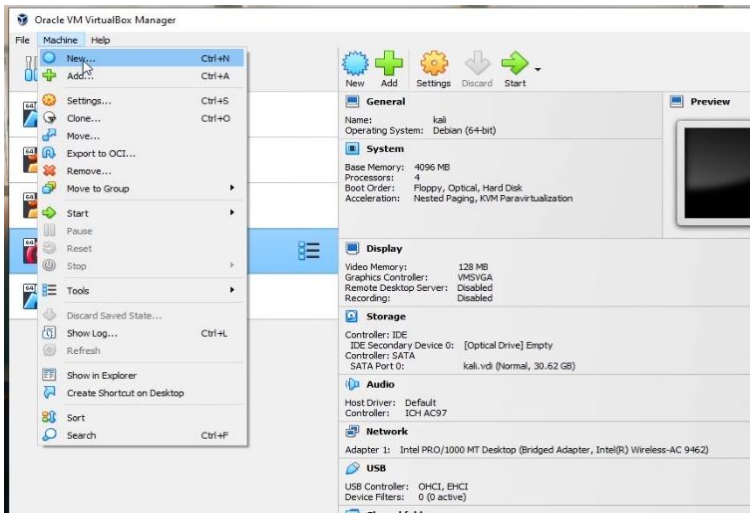
we downloaded the ubuntu-22.04.1-desktop-amd64 version from the official website you can easily download it from here [ubu] or the latest version.it will be downloaded as iso file to setup it we need hypervisor like VMware or VirtualBox. we used VirtualBox to setup ubuntu on it, you can download it from the official website [vbox] .

After downloading VirtualBox and ubuntu lets setup ubuntu on it as follows:

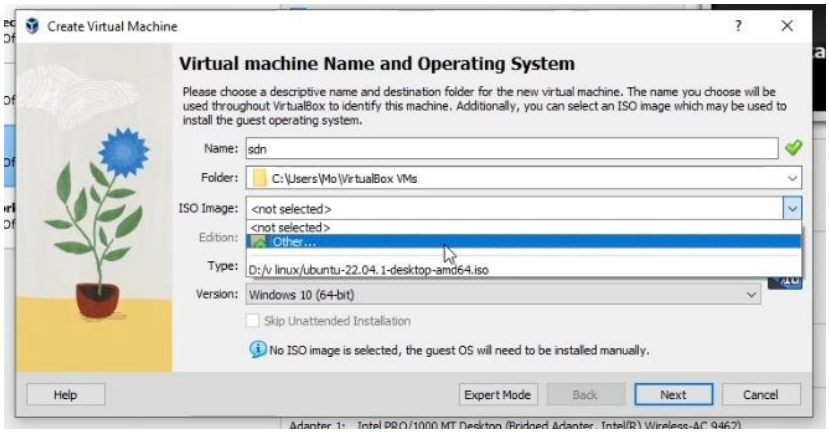
1- Install vbox and it will appear like this.



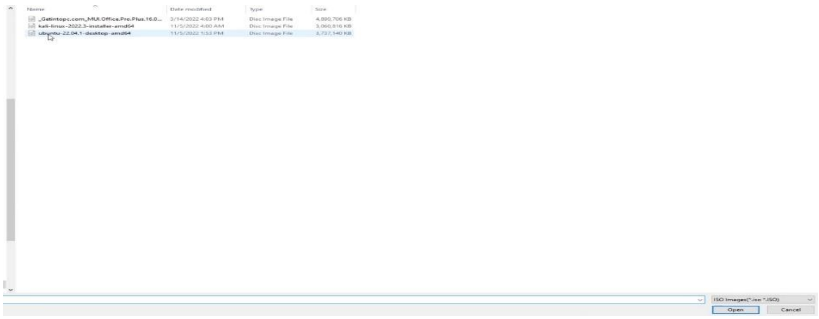
2- Press on Machine on the top left and choose new.



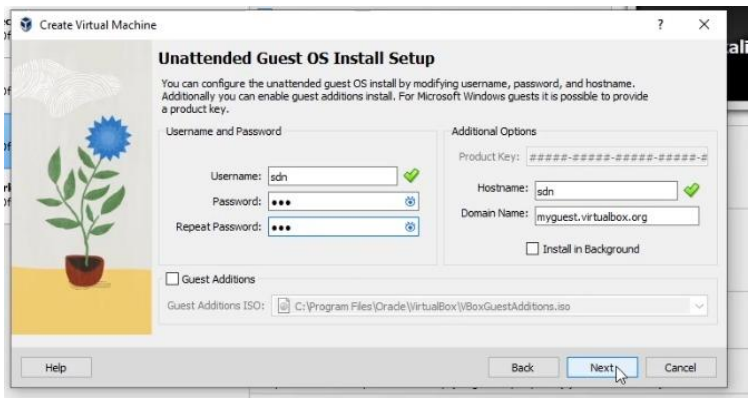
3- Type the name of the machine and choose the iso file that you have downloaded from others.



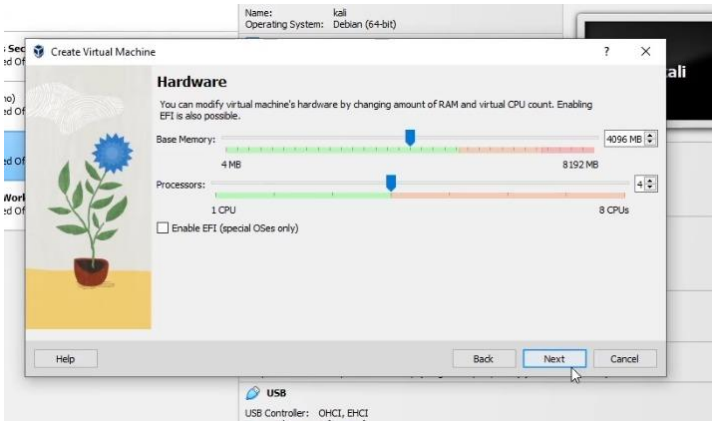
4- Choose the ubuntu iso file and press open.



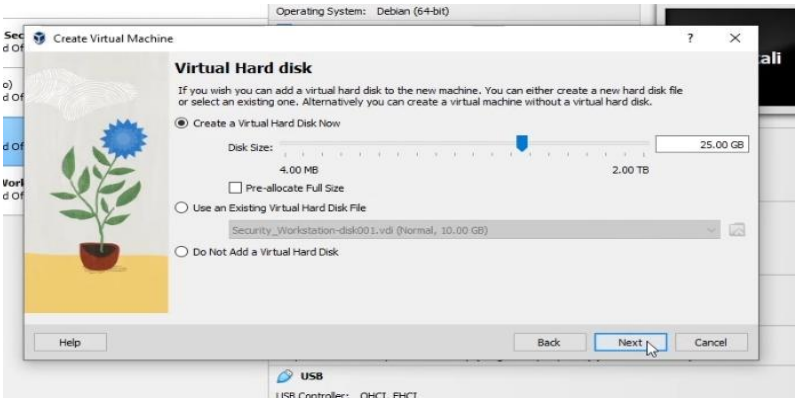
5- Press next and choose your username and password and press next.



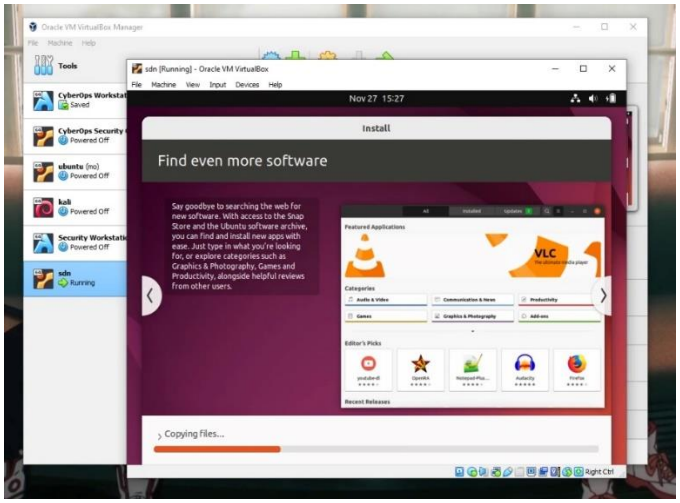
6- Here you choose your hardware resources 4Gb ram and 4cores will be sufficient.



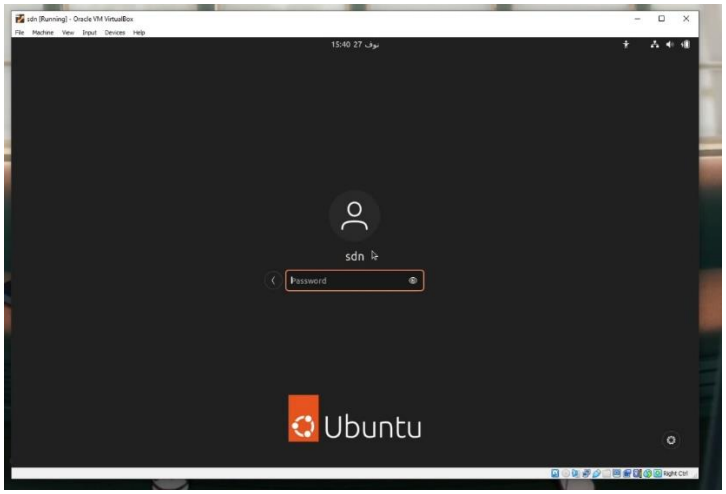
7- After pressing next here you select the disk size 25Gb will not be enough I had to change it to 35Gb later.



8- Press next and finish it will be auto installed as follows.



9- After installing login with the password, you created before.



1.9 INSTALLING MININET AND POX CONTROLLER

1- Start the ubuntu to install mininet by typing the following commands in the terminal: -

$ sudo apt-get install git-core

$ git clone git://githup.com/mininet/mininet

$ sudo apt-get update

$ sudo apt-get install mininet

$ sudo mn

The first command used to install git core to clone repositories(resources) from there.

The second command is used to clone the mininet repository from the link.

Note: After cloning the githup repository of mininet it should be available in /home/<username>.

The third command is used to update the apt database to install the mininet and its dependencies.

The fourth command is used to install the mininet.

Note: Now we can test by run the fifth command →it create the default topo.

2-After installing mininet we installed the controller (POX) by the following command: -

$ mininet/util/install.sh -p

After installing the POX it should be in the directory /home/<username>

Note: to test the POX controller type the next command:

$ python2 pox.py forwarding.l3 It should be up now

3-After installing the mininet and the POX controller, we now can create our topology for the project.

As a start we create a tree topo that includes the following:

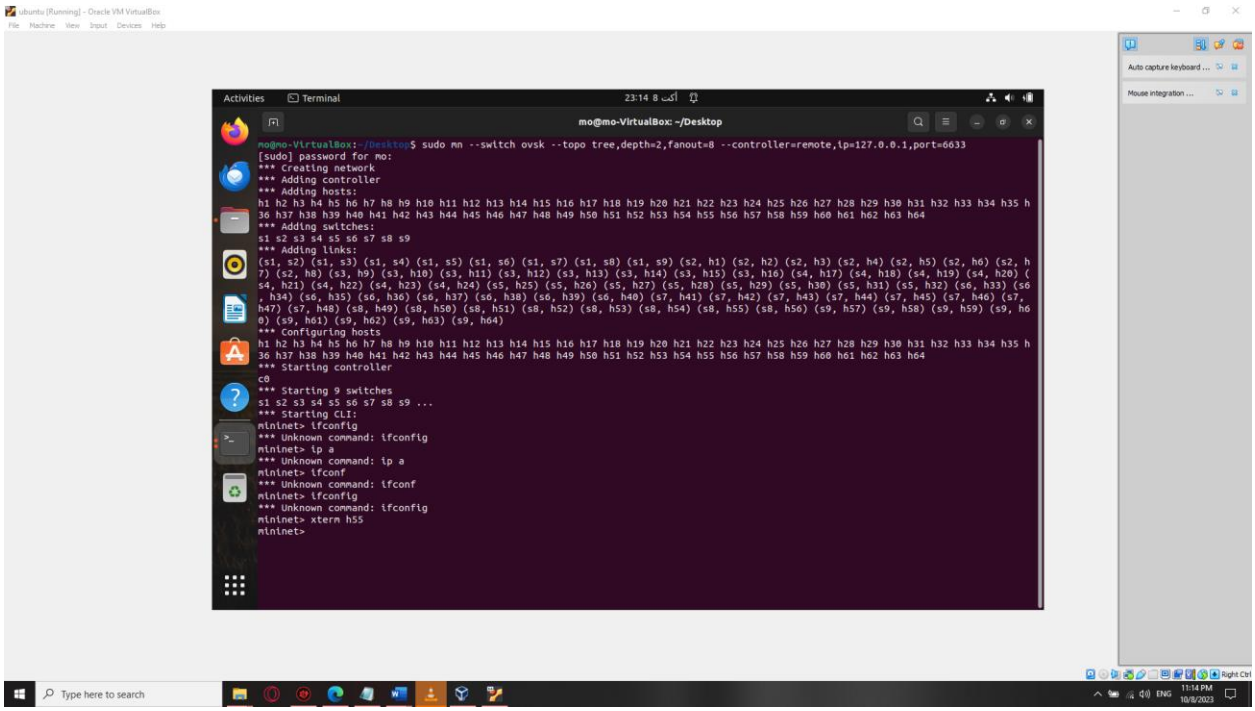
(The pox controller – 9 OpenFlow switches – 64 pc device)

The controller connected to the core OpenFlow switch and the core OpenFlow switch connected to 8 OpenFlow switches each switch connected to 8 pc devices.

We can create this topo by running the following command →

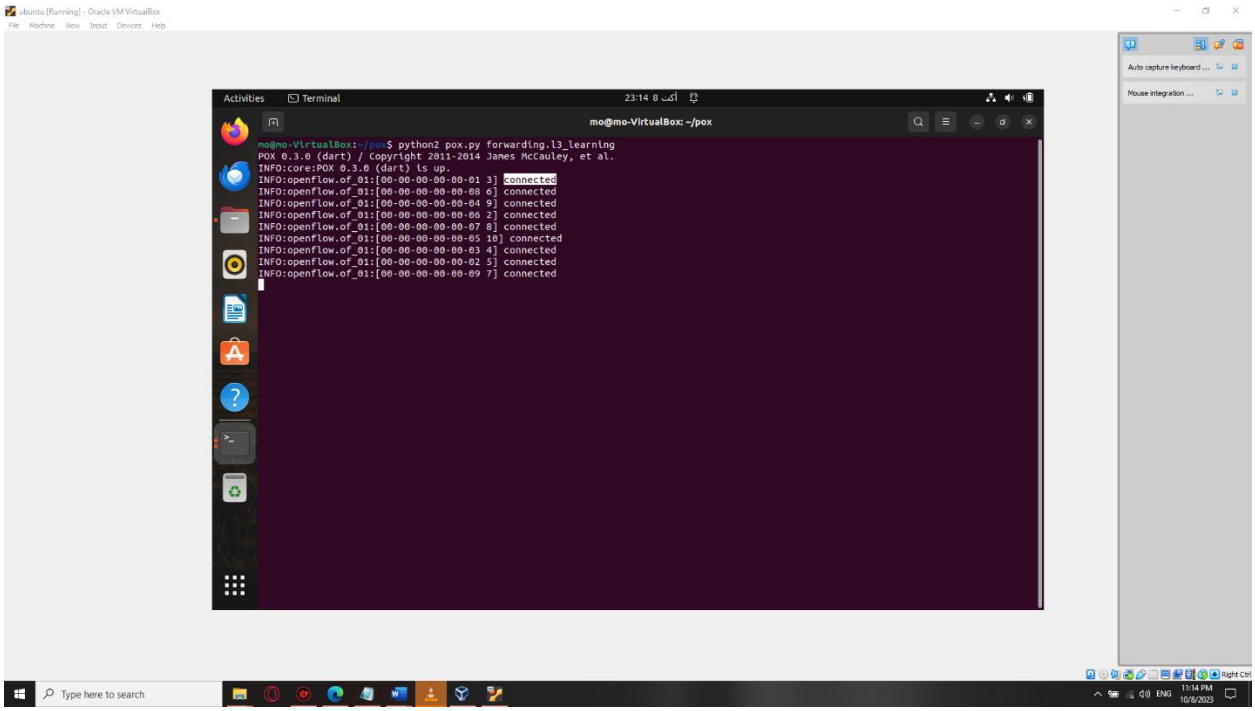
$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --

controller=remote,ip=127.0.0.1,port=6633



Before creating the topo make sure that the controller POX is running(listening) by running the following command in this directory (home/pox) :

$ python2 pox.py forwarding.l3_learning



Now the POX is listening, when running the topo command, the topo created in the mininet and the pox captured 9 OpenFlow connections, that means that the topo is working perfectly.

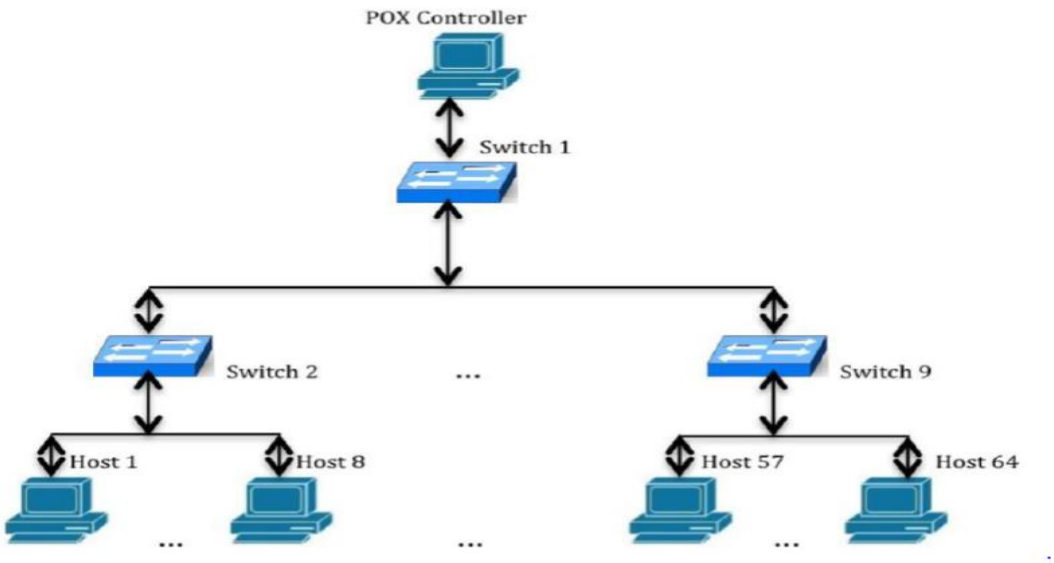


Fig 1.9 Openflow connection through topo

1.10 THE FIRST UPDATED TOPOLOGY

We did some updates to the previous topo to make it better than before. We reduced the topology to reduce the number of nodes in the network for a good performance and increase one controller (c1) to work as a data backup or a data replication for the essential one (c0) until the topology becomes as follow:

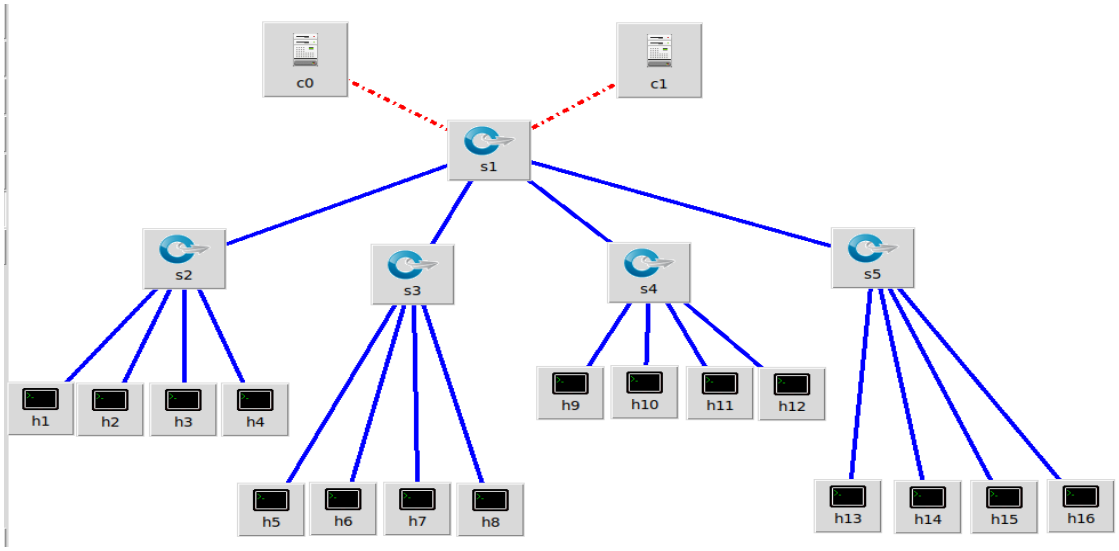


Figure 1.9. The first updated topology

About the topology:

As we see here in the topo, the essential controller (c0) is connected to the core OpenFlow switch and the controller (c1) works as a mirror (data replication) to the c0 controller. If the c0 drops the c1 is still working as a (data replication).

And the core OvS switch is connected to four OvS switches, everyone connected to four hosts. This topo is called tree topology.

Reasons for Updating the Network Topology

1. Redundancy and Reliability:

- Initial Topology: The initial setup with a single POX controller posed a single point of failure. If the controller failed, the entire network would be affected, leading to a complete disruption of services.

- Updated Topology: Introducing two POX controllers enhances redundancy. If one controller fails, the other can take over, ensuring continuous network operations and increased reliability.

2. Scalability and Manageability:

- Initial Topology: Managing a network with 9 OpenFlow switches and 64 PC devices can be complex and difficult to scale, especially in a lab environment or for educational purposes.

- Updated Topology: Reducing the number of switches to one core switch and four access switches, along with 16 hosts, simplifies network management and makes it easier to scale and troubleshoot.

3. Efficiency and Performance:

- Initial Topology: The larger network with more switches and devices could lead to increased latency and reduced performance due to the higher number of hops and potential bottlenecks.

- Updated Topology: The streamlined topology reduces the number of hops and potential points of congestion, improving overall network performance and efficiency.

4. Focus on Security:

- Initial Topology: A larger network can introduce more security vulnerabilities and makes it harder to monitor and secure every component effectively.

- Updated Topology: With a more manageable number of switches and hosts, security measures can be more effectively implemented and monitored, providing a more secure environment for testing and development.

5. Cost and Resource Optimization:

- Initial Topology: Deploying and maintaining a larger network requires more resources, including hardware, power, and administrative overhead.

- Updated Topology: Reducing the number of devices and switches optimizes the use of resources, lowering costs and making it feasible to maintain and operate the network efficiently.

6. Practicality for Research and Development:

- Initial Topology: The complexity of the larger network might be impractical for focused research and development, particularly in a controlled environment.

- Updated Topology: The updated, simpler topology is more practical for testing specific security enhancements and network configurations, providing a controlled environment that better suits research objectives.

These changes help create a more robust, manageable, and efficient network environment, better suited for exploring security enhancements and addressing the unique challenges posed by SDN architectures.

How we implement this topology:

1- open the terminal and type the following commands in this directory Home/pox to open the two controllers (c0, c1):

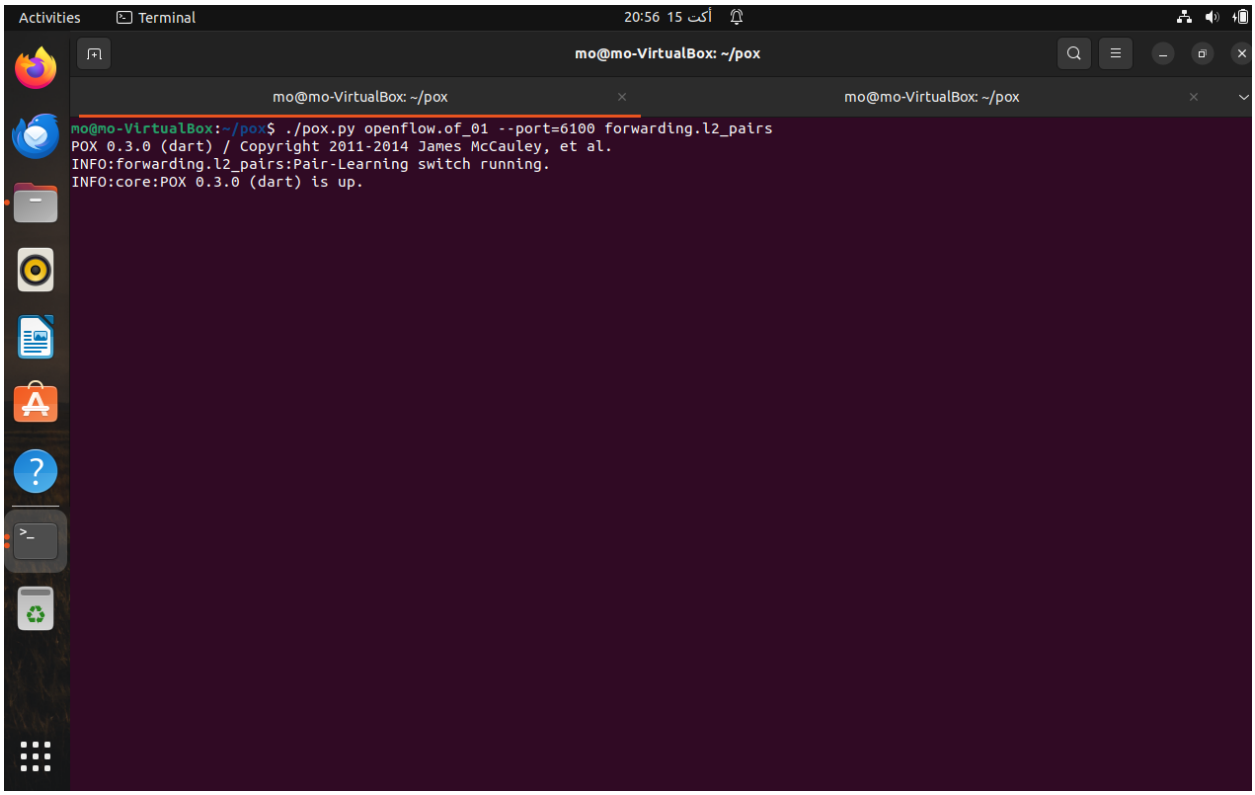
$ ./pox.py openflow.of_01 --port=6100 forwarding.l2_pairs

$ ./pox.py openflow.of_01 --port=6101 forwarding.l2_pairs

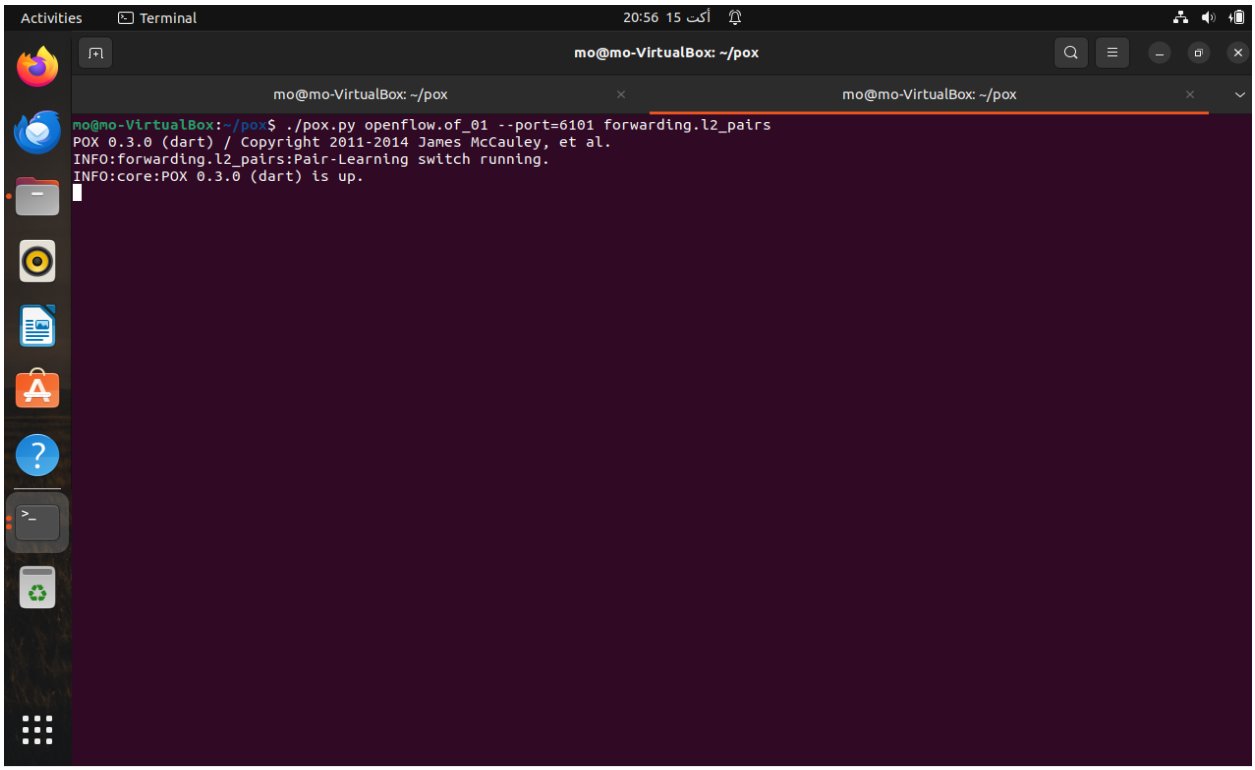
The first command opens the c0 pox controller with OpenFlow version one protocol running on it at the port "6100" and listening to the forwarding layer of the network and captures any change on it.

The second command does the same job but, in the port, "6101" and the c1 pox controller here work as a mirror of c0. We got this commands from GitHup here the link(https://github.com/mininet/mininet/issues/851)

So now the controllers are up and wait for the topo to be created:



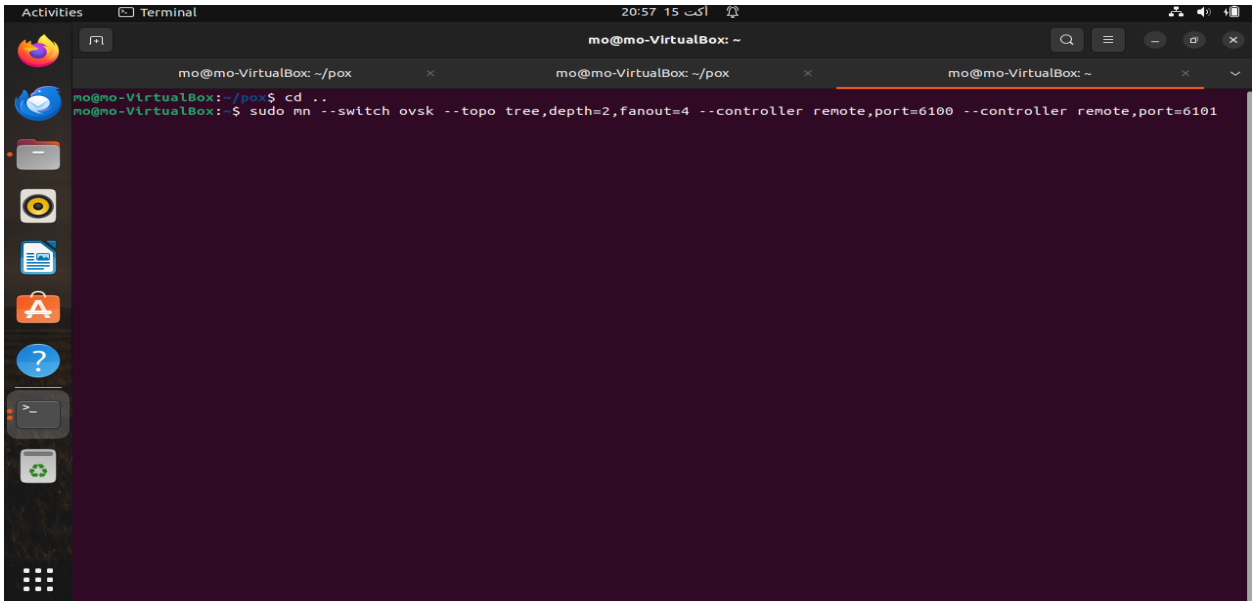
C0 is up



C1 is up

Now we start to create our topo by opening a new terminal window and type the following command :

$ sudo mn --switch ovsk --topo tree,depth=2,fanout=4 --controller remote,port=6100 --controller remote,port=6101



1-The sudo: gives the mininet a root permission to create the topo

2- --switch ovsk : stands for creating switches named open vSwitches.Its a production quality, multilayer virtual switch licensed under the open source **Apache 2.0** license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag). In addition, it is designed to support distribution across multiple physical servers like VMware's vNetwork distributed vswitch or Cisco's Nexus 1000V. To See full feature list (https://www.openvswitch.org/features/)

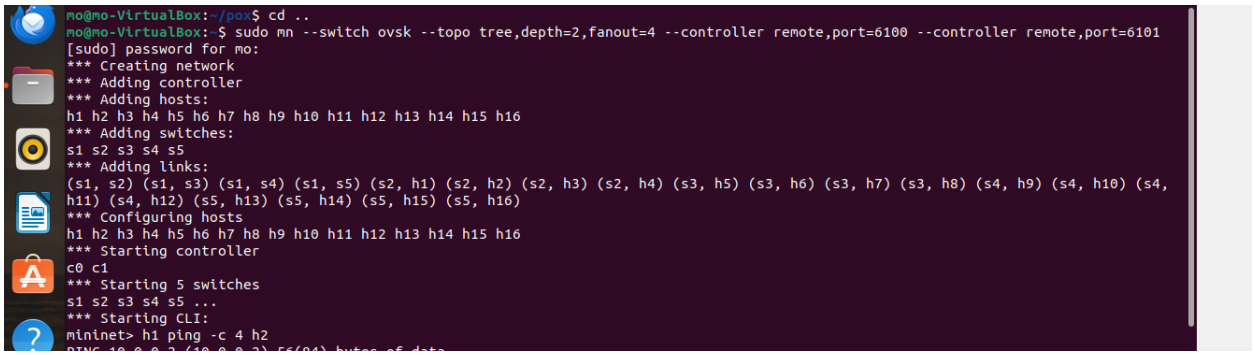
for more information about ovsk switches go to the official sit here (https://www.openvswitch.org)

3- --topo tree,depth=2,fanout=4 : the topo type is tree and having a depth of two and fanout of four, that's means it has a core switch(s1) connected to four switches (s2,s3,s4,s5) ,each one connected to for pc devices.

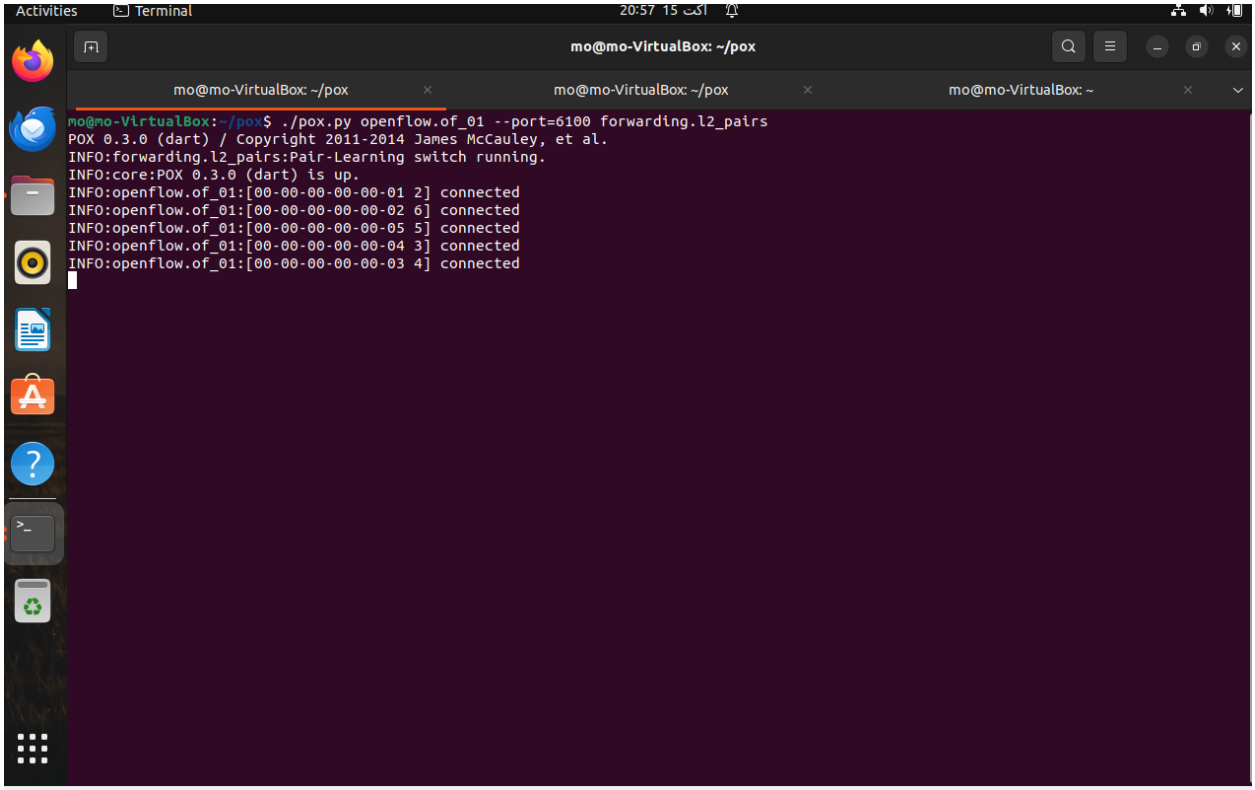
4- --controller remote,port=6100 : this tells the Mininet that there is a connected remote pox controller working on port "6100" with the default ip "172.0.0.1".

5- --controller remote,port=6101 : this tells the Mininet that there is a connected remote pox controller working on port "6101" with the default ip "172.0.0.1".

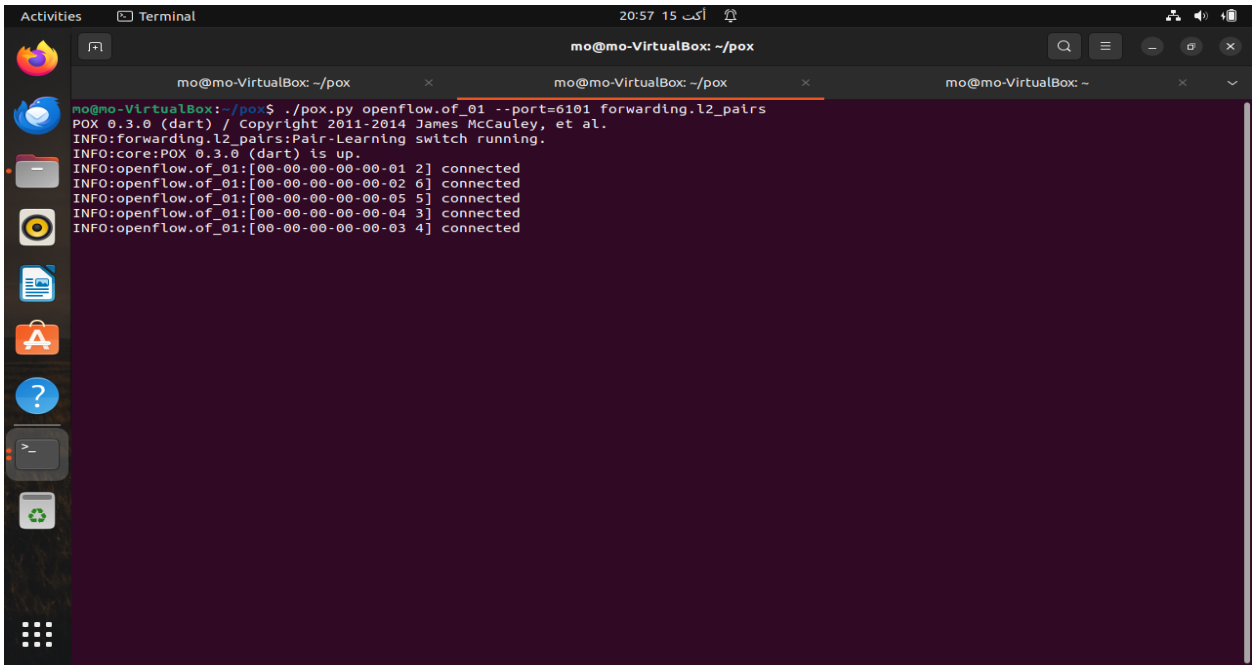
After running the mn command the topo created successfully as follows :



Also the two controllers works and starting monitoring as follows:



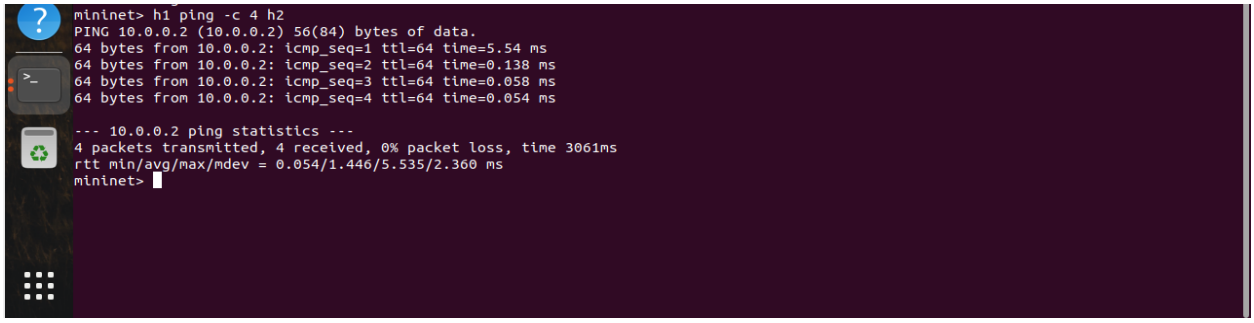
C0 connected



C1 connected

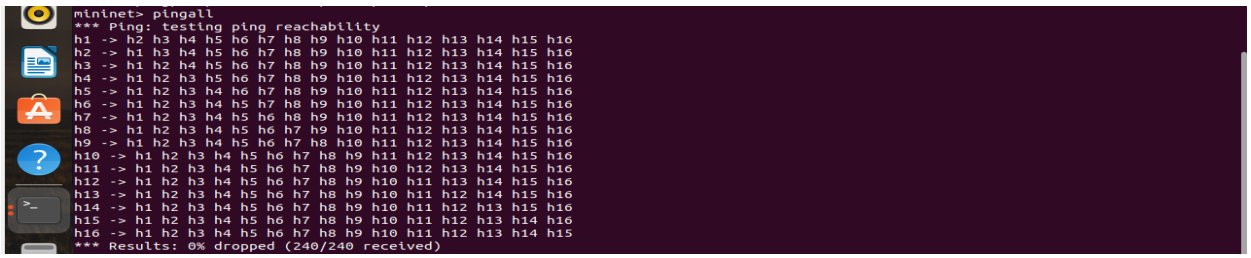
Now after the topo and the two controllers are working let's check some commands:

1- First let’s do a ping from h1 to h2 (send 4 packets)

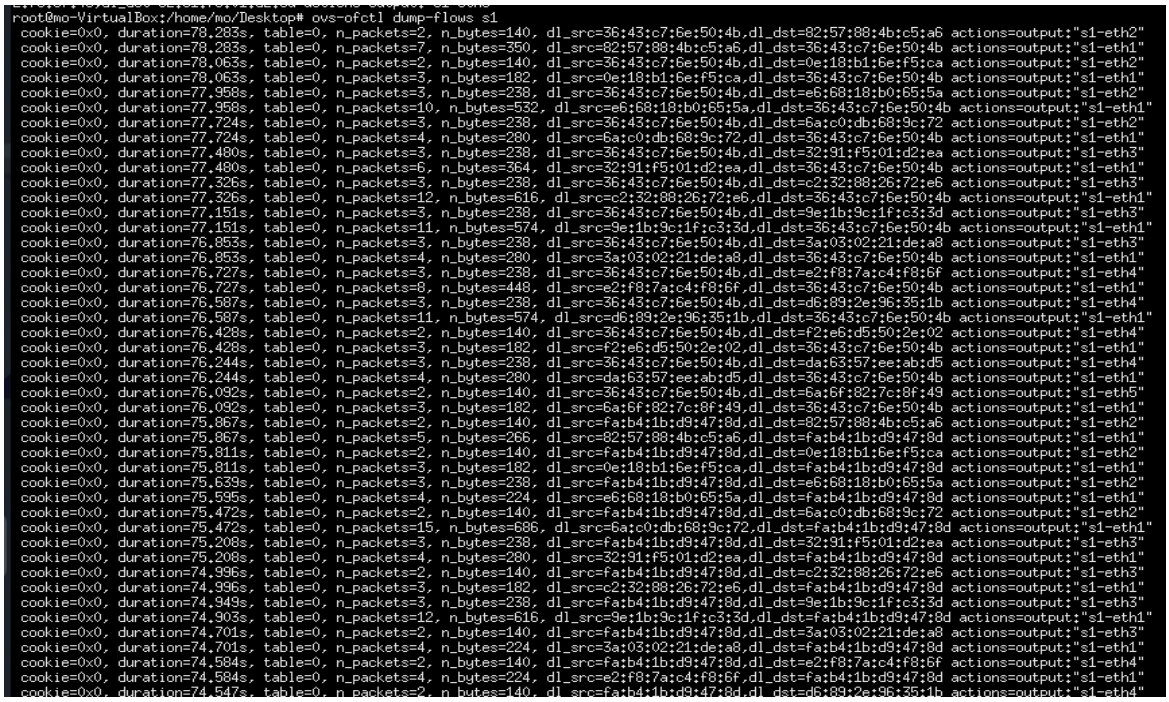


As we see here all four packets received

2- lets ping on all devices

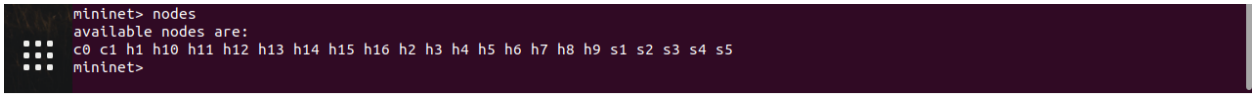


All the devices are reached to others



S1 flow rules after pingall and there are more flow rules.

3- lets check the nodes



4- lets see all links

All links are connected successfully

CHAPTER 2: ENHANCEMENT OF SDN COMMUNICATION SERVICES BY APPLYING DIFFERENT STRATEGIES

2.1 SERVER/CLIENT COMMUNICATION

Overview

To make sure of the performance of the SDN network we should do some tests to our network by running any service in the network like FTP (file transfer protocol)

The File Transfer Protocol (FTP) is a standard communication protocol used for the transfer of computer files from a server to a client on a computer network. FTP is built on a client and server architecture using separate control and data connections between the client and the server of the ftp protocol configured,

and needs an authentication between client and server by a plain text username and password which will be set in the server.

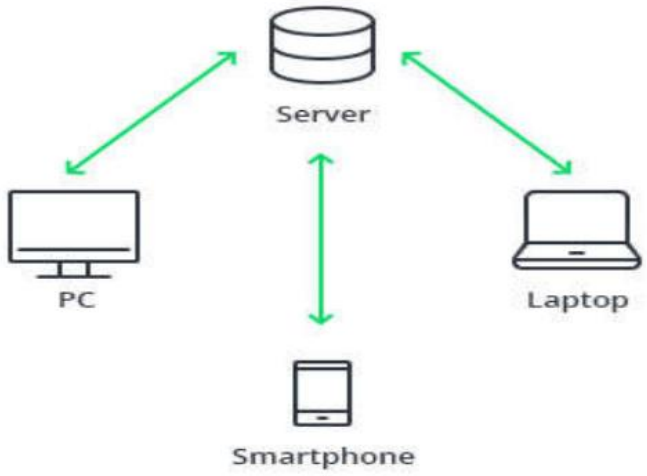
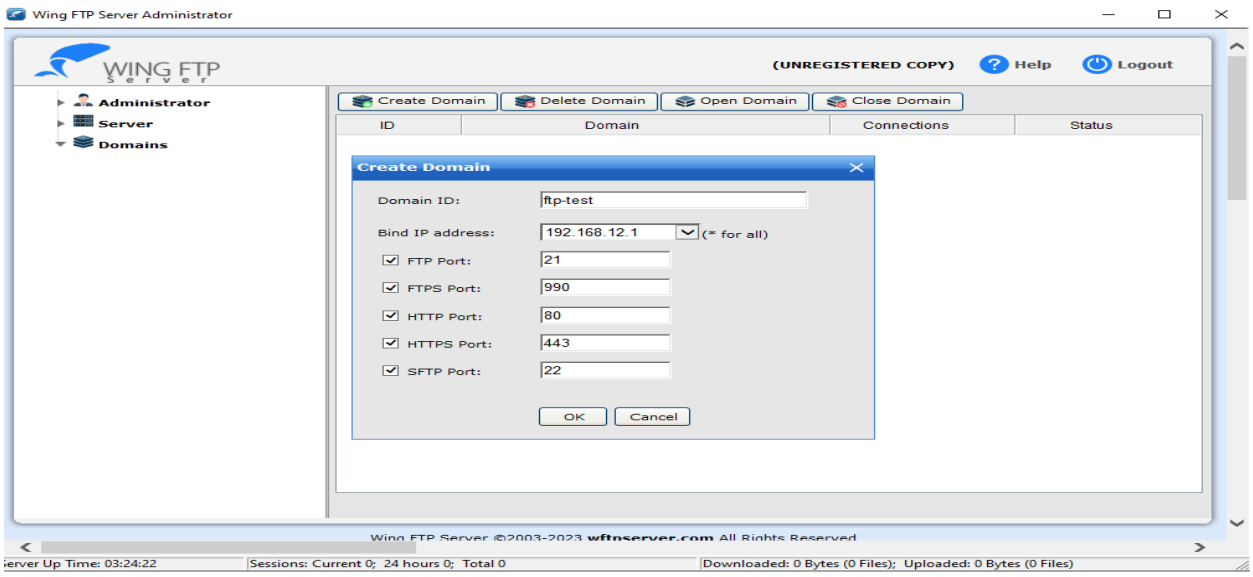


Figure 2.1. SERVER/CLIENT COMMUNICATION

The Implementation.

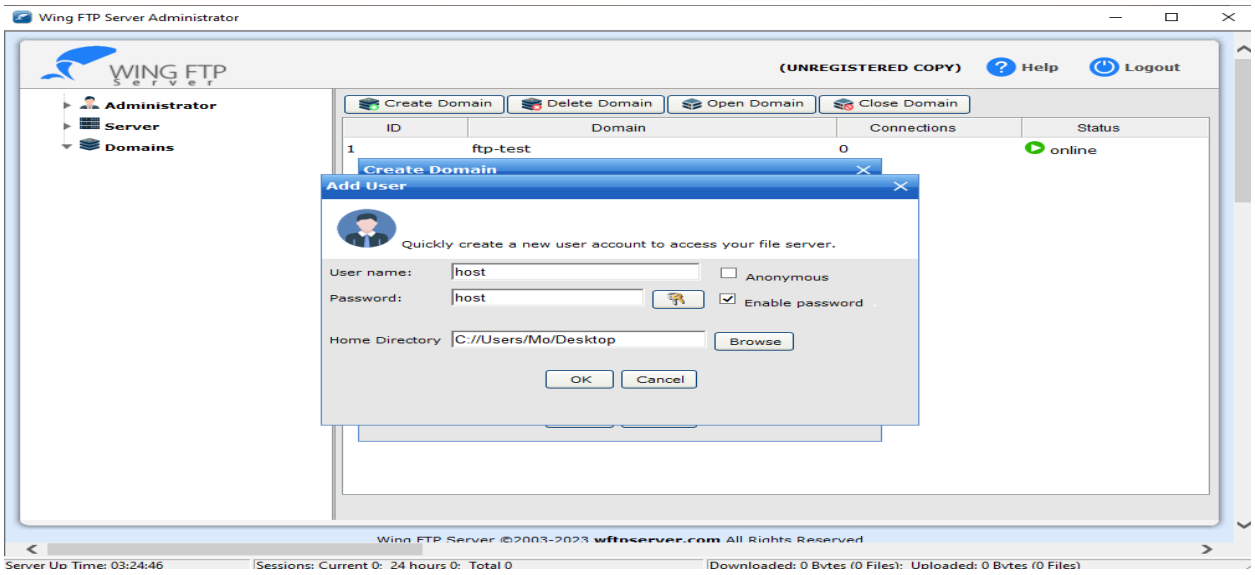
1) first step at all is the installation of the ftp server (Wing FTP Server Admin) on the windows machine:

2) creation of the ftp server.:

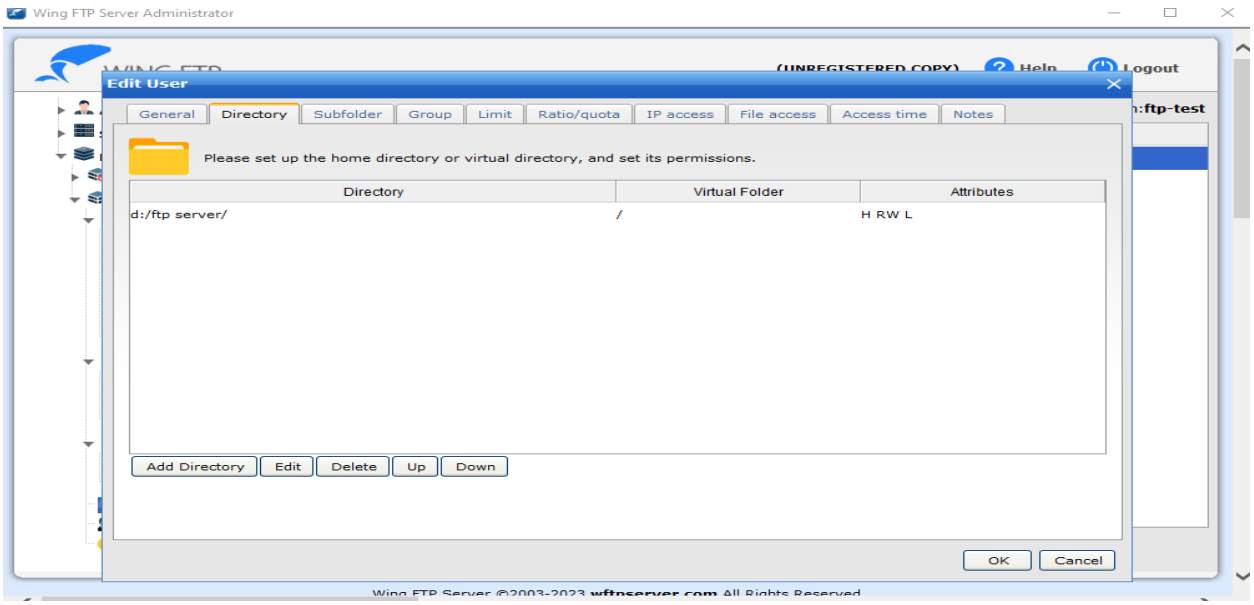


Give it the domain name and the IP of the server and check the service ports and press ok.

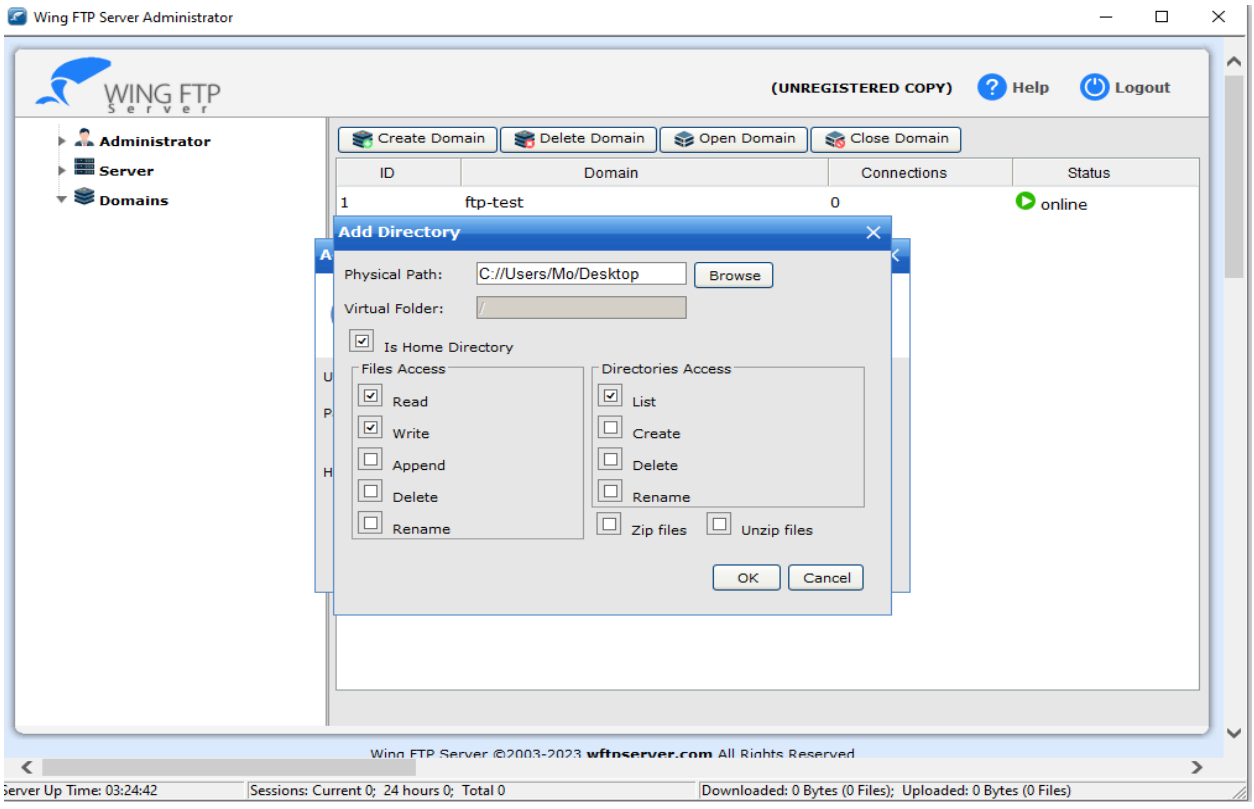
3) Now add a user by adding the name, password(credentials), and the directory:



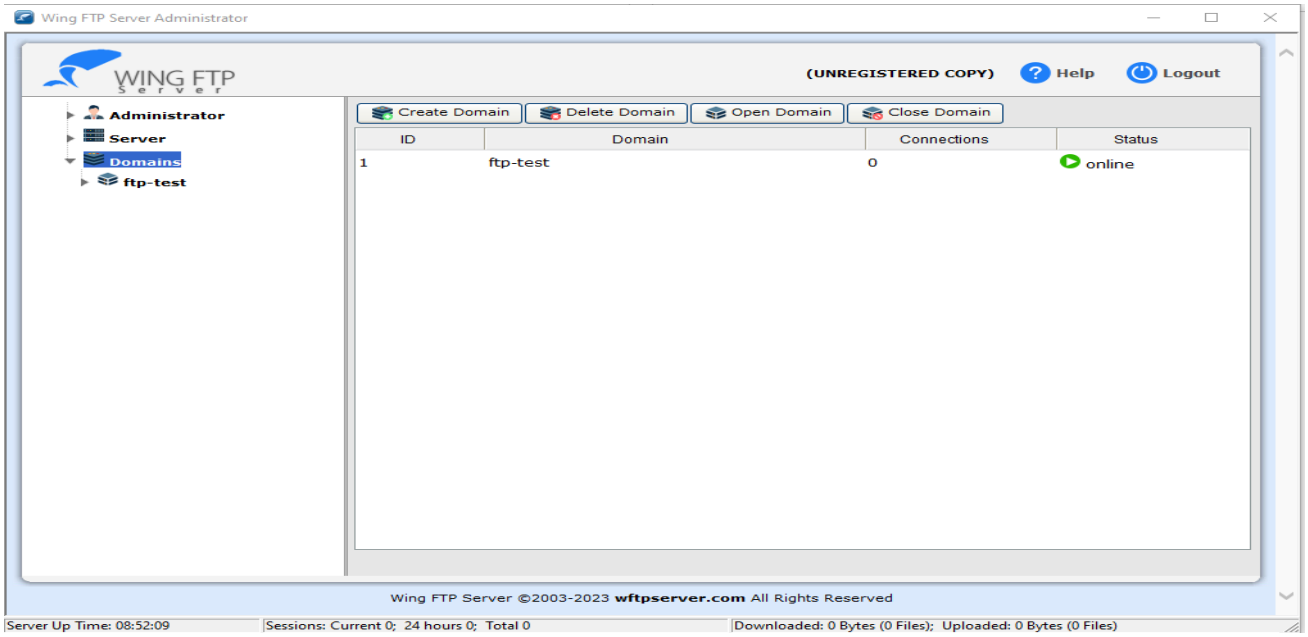
We have to change this directory after attaching some files.



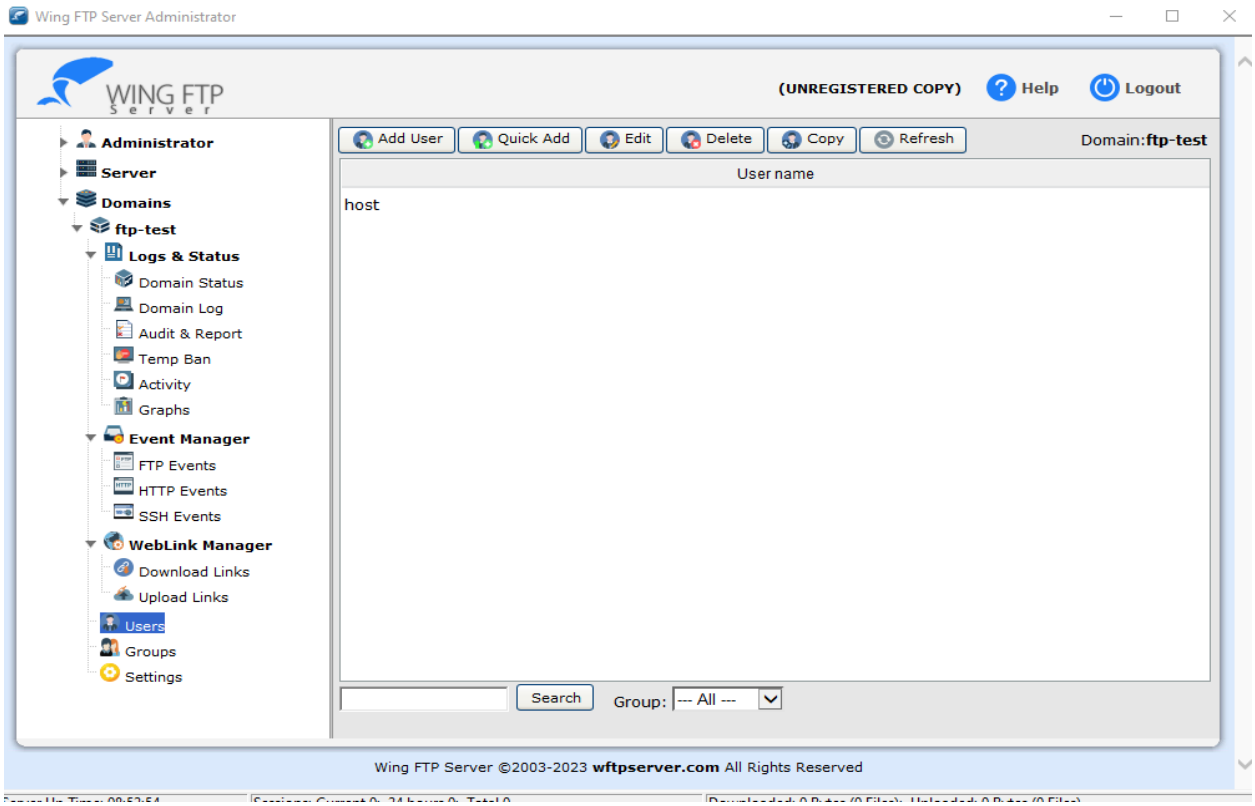
4) give the privileges of the user and press ok:



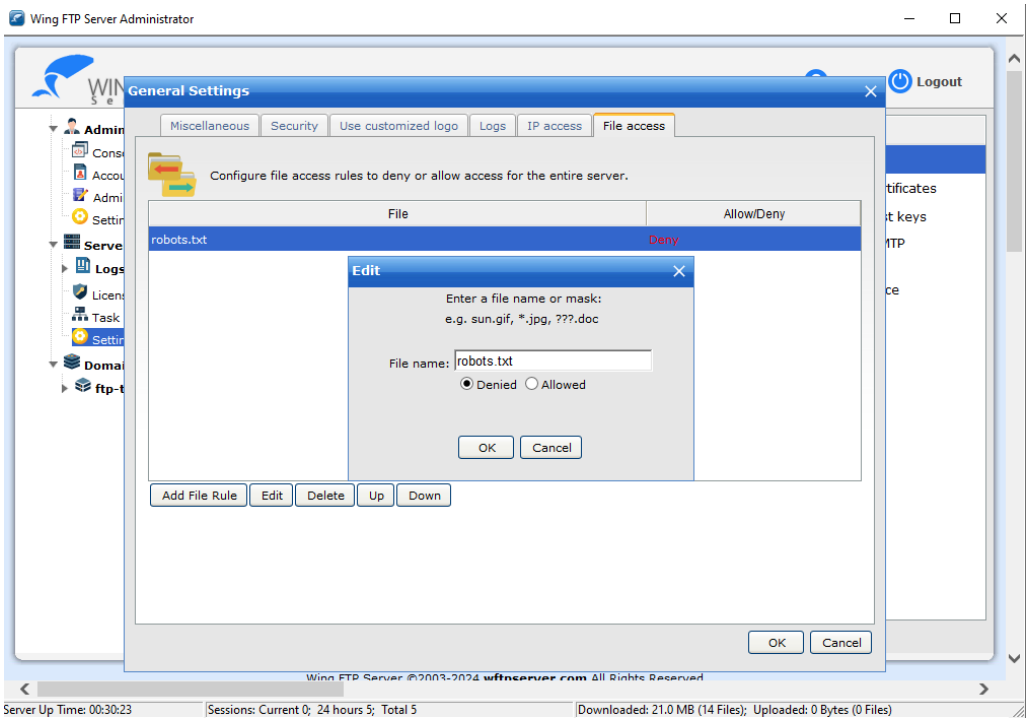
Now we created the user and the domain.



ftp-test domain is online.



The added user.



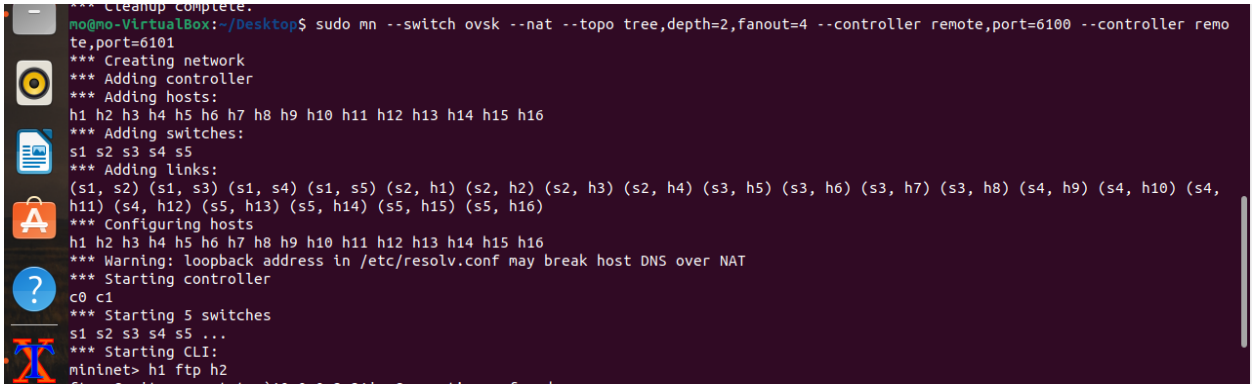
File access.

5)now we open the ubuntu machine to run the network and test our ftp server. but we faced a problem that the Mininet network is private so it cannot see the ftp server or even ping on google so it cannot reach the outside network (WWN).

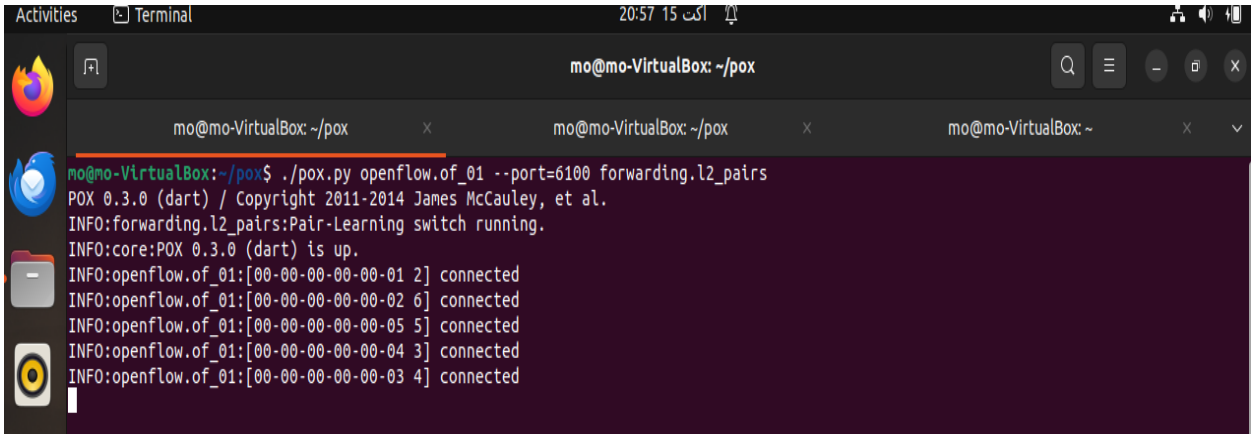
So, to solve this problem we must run the Nat protocol in the network to connect it to the eth0 (NIC):

so, it can be done in the command that runs the topo by adding the NAT argument as follows: -

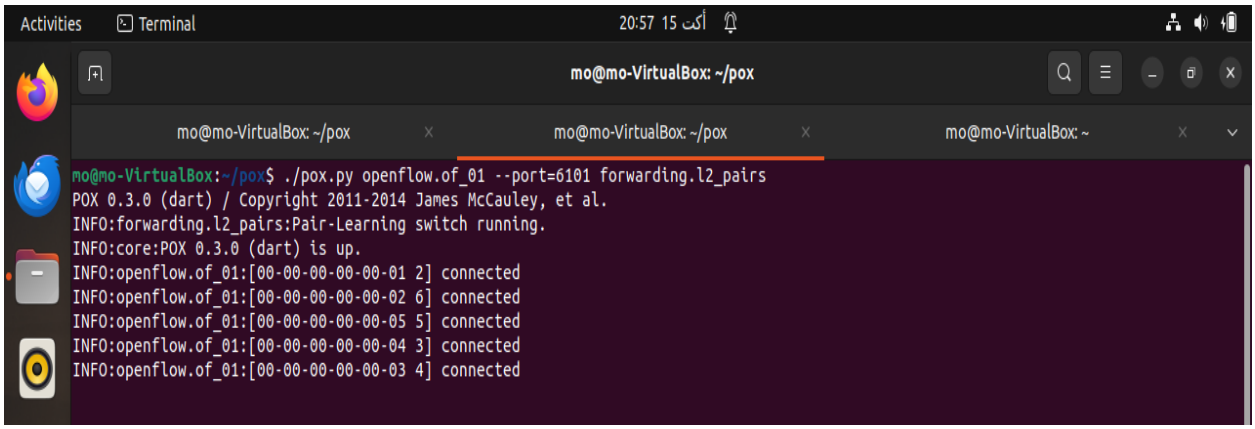
$ sudo mn --switch ovsk --nat --topo tree,depth=2,fanout=4 --controller remote,port=6100 --controller remote,port=6101



6)before creating the topo open the two pox controllers:



The 1st controller



The 2nd controller

7)Now to check that our network is connected to the WWN, ping on google's DNS:

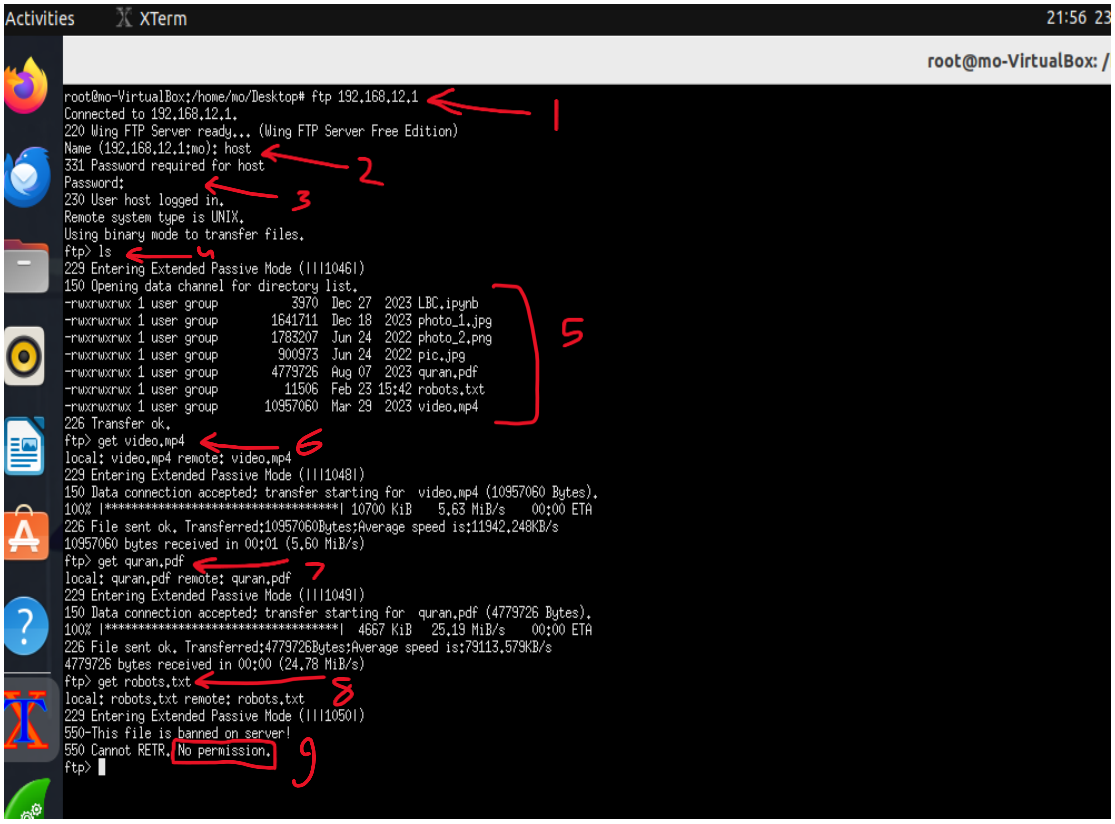
```
mininet> h1 ping -c 4 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=51.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=42.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=46.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=58.7 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3015ms
rtt min/avg/max/mdev = 42.484/49.745/58.685/6.044 ms
mininet>
```

So, it works

8)We now go to check the ftp server. We tried to run the service in the Mininet cli but it doesn't work efficiently:

So, we tried another way by xterm tool (cli) of the hosts. We chose host1 to download the file.

9) On the CLI of h1 do the following:



1- Connect to the server by typing (ftp 192.168.12.1) to request connection with the ip address.

2- Entering the username of the user that we created.

3- Enter the password also.

4- Now we connected successfully to the server and typed *ls* to see the files stored in the server.

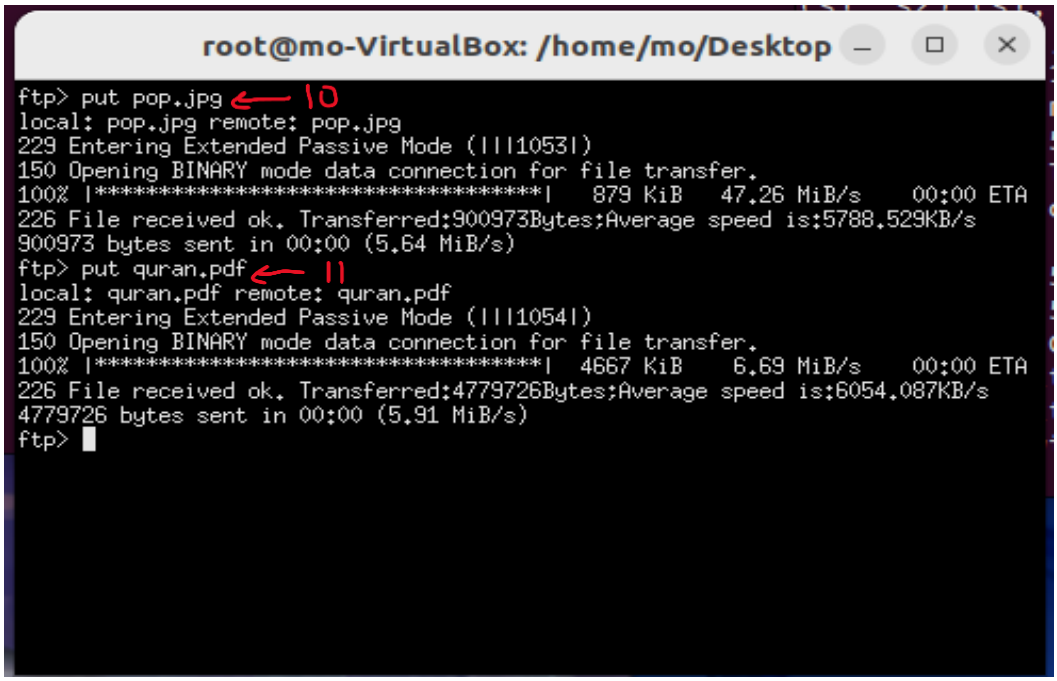
5- The available files stored on the server.

6- Downloading the video by *get* command from the server on the host.

7- Downloading another pdf file.

8- Trying to download the banned file.

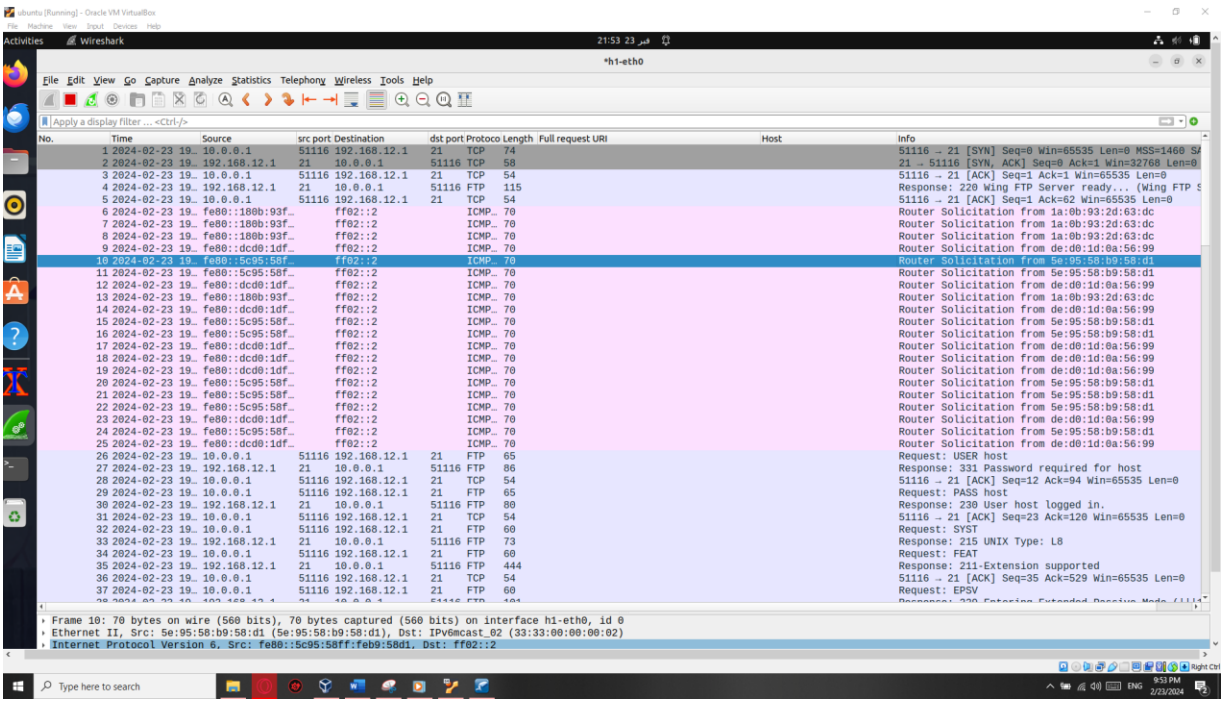
9-No permission as expected.



10- uploading a photo to the server.

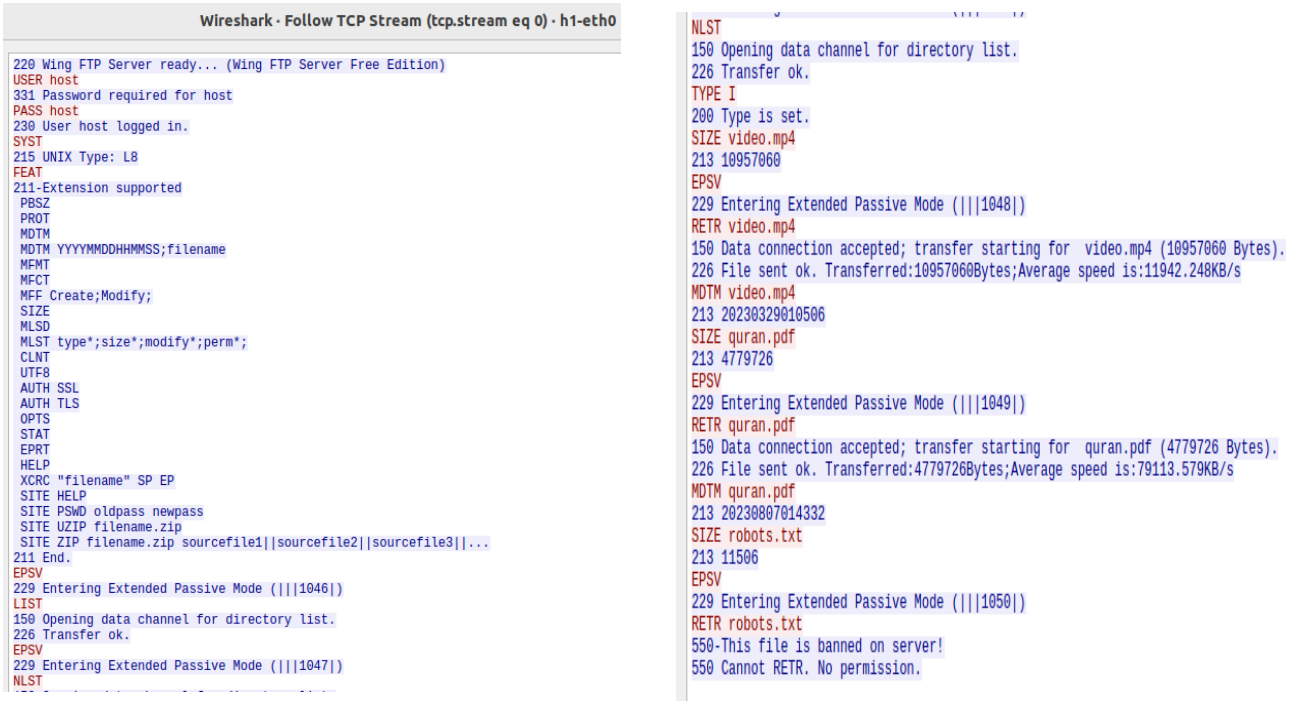
11- uploading another file to the server.

10)Capturing the packets using Wireshark:



As we see TCP uses a three-way handshake to establish a reliable connection and Authentication between the host and the server. After that the process of transferring files starts using ftp protocol.

11) If we follow the tcp stream we'll see all the data (traffic) as a plaintext: because ftp protocol is not secure.

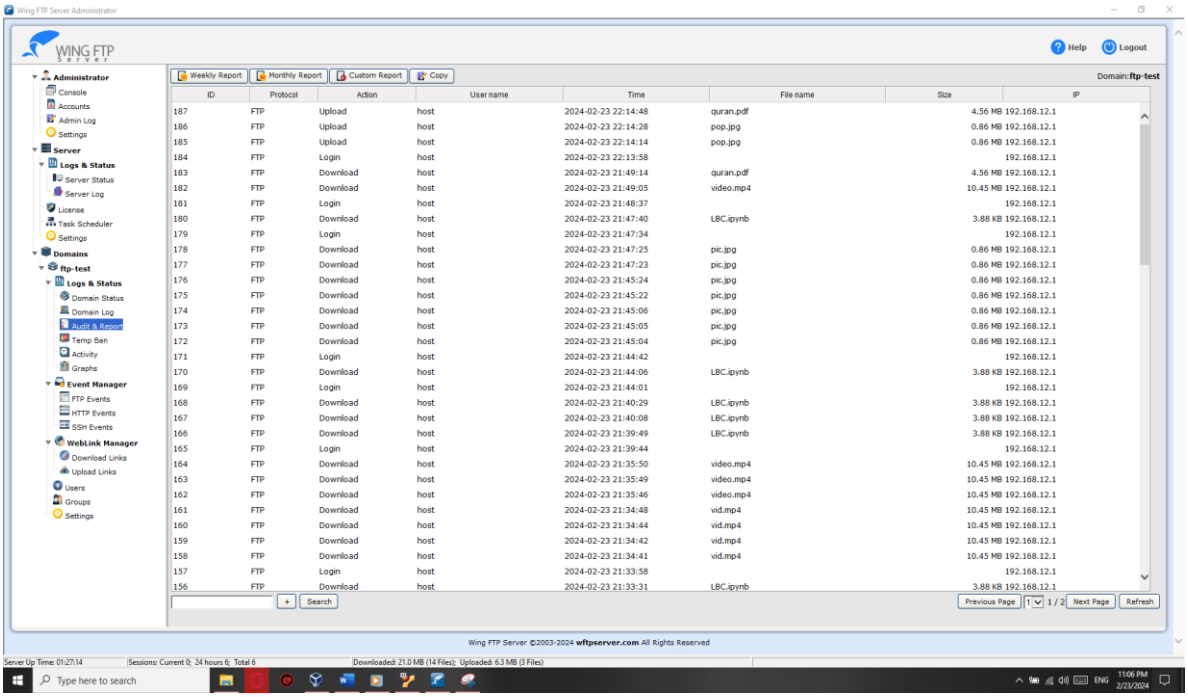


12) Back to the server and see the logs status:

Domain log saved in the server and all sessions.

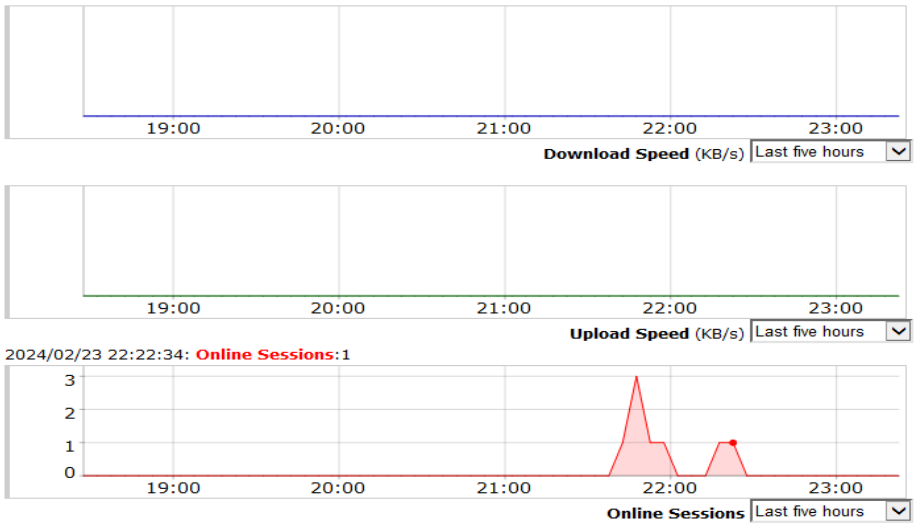
Ex: the last uploaded files to the server.

13- Audit & Reports can also be seen:



Includes (ID,protocol,Action,username,time,file name,size and server ip address)

14- Graphs:



Graphs show the number of online sessions, the download and upload speed.

2.2 CLIENT/CLIENT COMMUNICATION

Point-to-Point Protocol (P2P) is a client-to-client data link protocol used to establish a direct connection between two nodes, unlike FTP which is a client to server protocol.

P2P is a widely used protocol that provides a reliable, efficient, and flexible means of establishing point-to-point connections between devices. It is supported by most operating systems and networking equipment and is used in a variety of applications, including internet access, remote access, and VPNs.

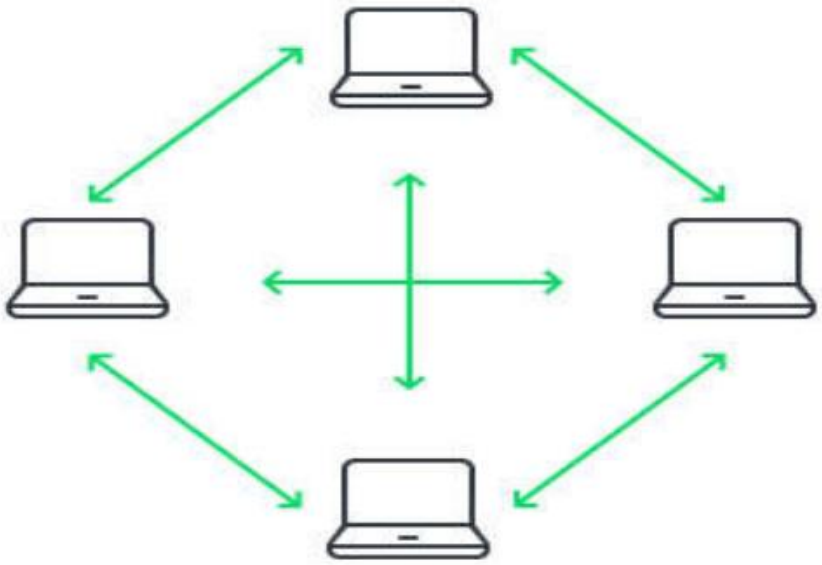


Figure 2.2. P2P COMMUNICATION

Netcat tool

Netcat, also known as "nc" is a powerful and versatile networking utility tool used for reading from and writing to network connections. It provides a simple and straightforward way to create TCP/IP. connections, listen for incoming connections, transfer data, and perform various network-related tasks Netcat can act as both a client and a server, making it useful for a wide range of tasks, such as:

1)Port scanning: Netcat can be used to scan for open ports on a remote host by attempting to establish a connection to different ports and analyzing the response

2)File transfer: It can facilitate the transfer of files between hosts by establishing a connection and sending or receiving data.

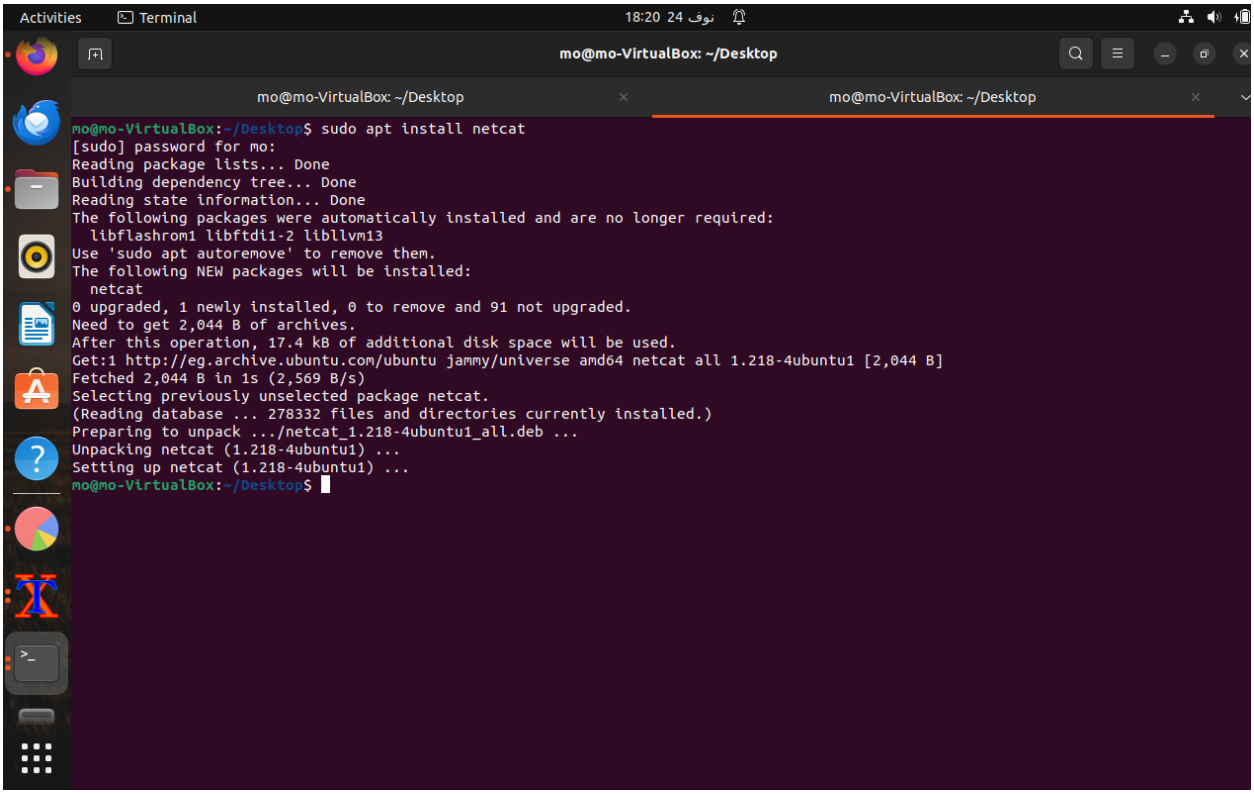
3)Port forwarding: Netcat can forward network traffic between ports or hosts, allowing for advanced network configurations and tunneling.

4)Network testing: It is commonly used for testing network services, checking connectivity, and troubleshooting network-related issues.

5)Remote administration: Netcat can be utilized to execute commands remotely on a remote server or host by establishing a connection and sending commands.

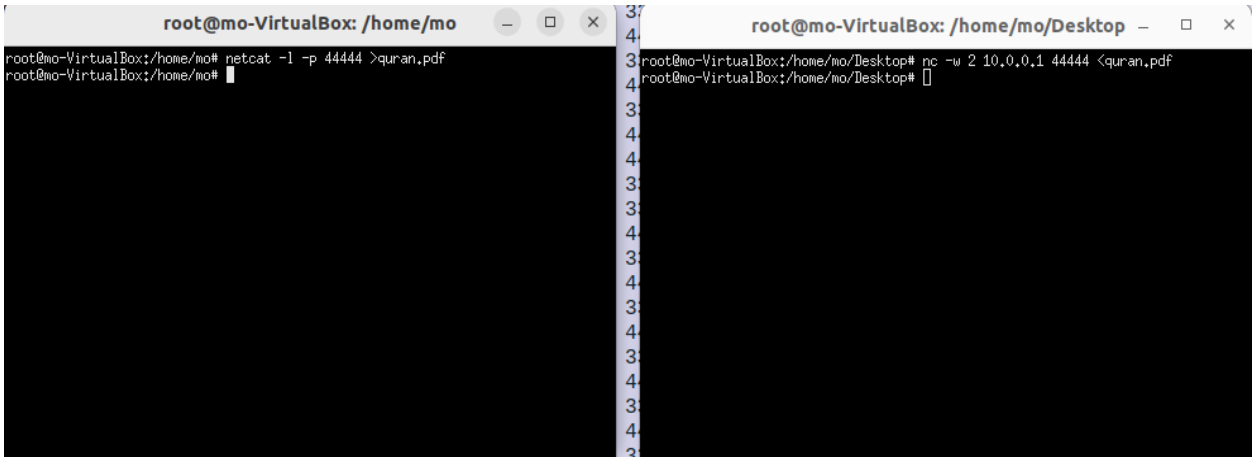
Netcat is highly flexible and can be used in various scenarios due to its ability to handle raw network connections. Its simplicity, combined with its wide range of uses, makes it a popular tool among networks. administrators, security professionals, and enthusiast

nc installation



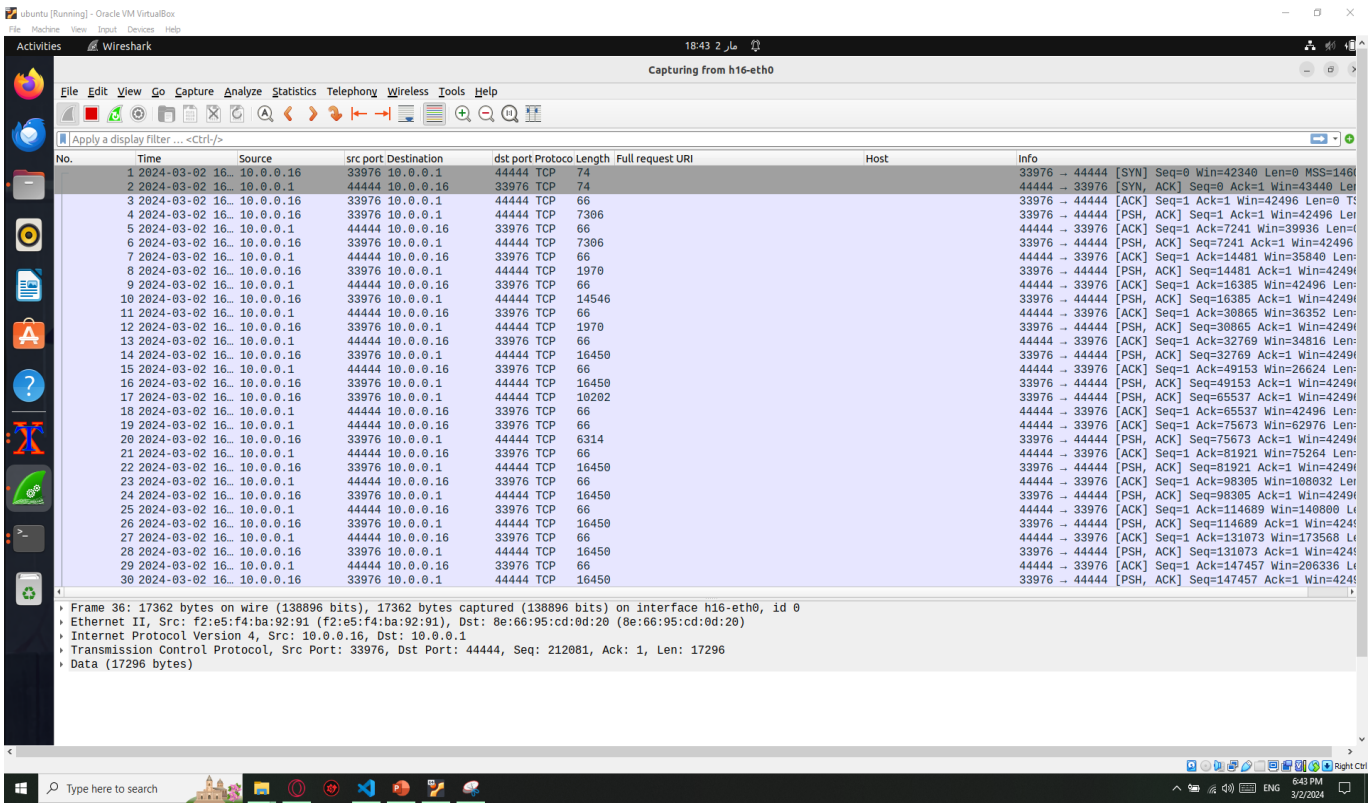
How to use nc tool

1-Open the cli of h1 and h16 by xterm and run the next commands:

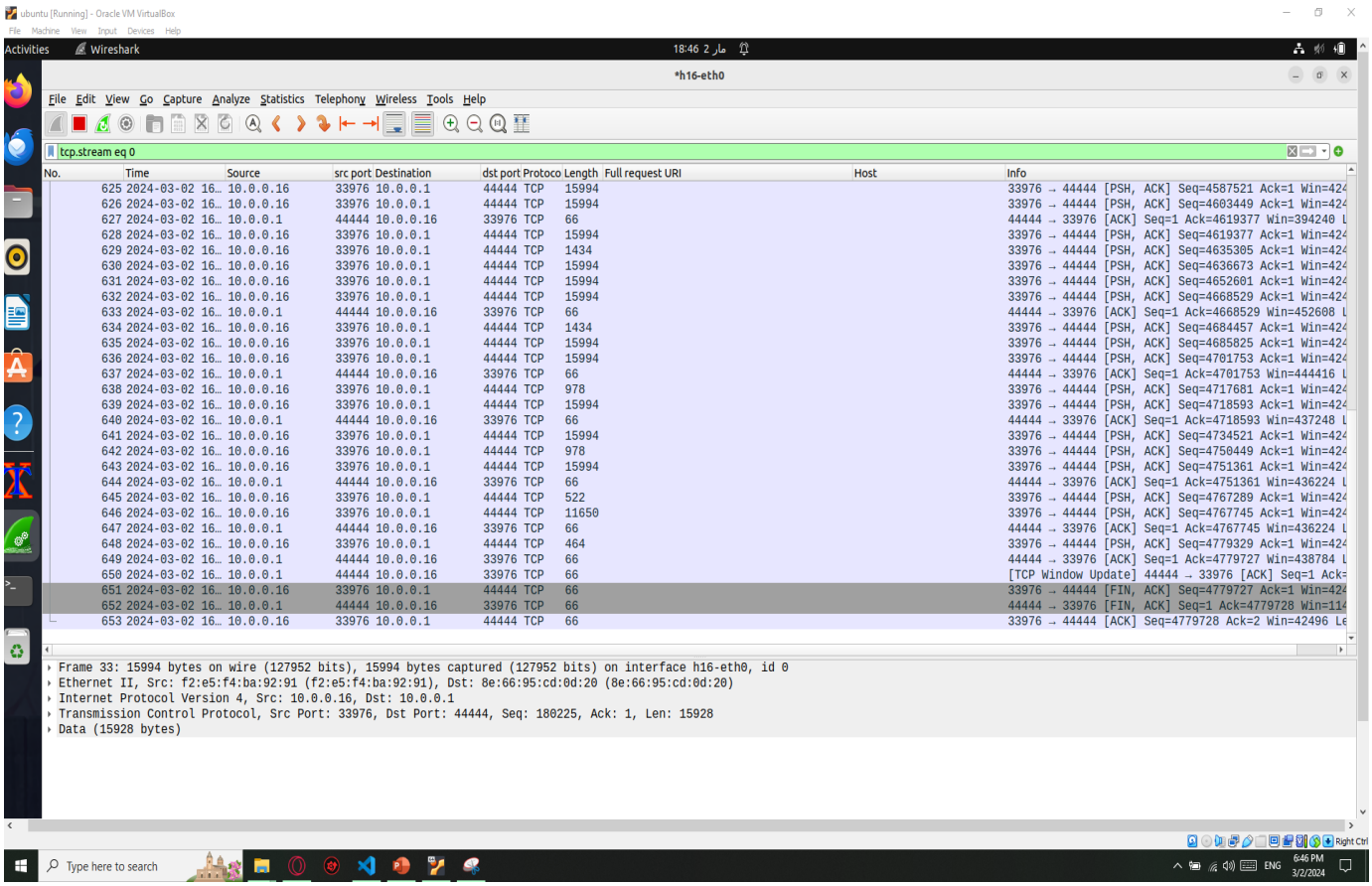


h1 is listening for an incoming connection and h16 sends the file

2-Capturing the packets using Wireshark.



The start of the connection



The end of the connection and file transferred successfully.

2.3 WEB SERVER

In an SDN (Software-Defined Networking) network created using Mininet and a controller like Ryu or POX, it is possible to make a host function as an HTTP server. This means that the host can serve web content and respond to HTTP requests from other hosts in the network. By leveraging the capabilities of SDN, we can dynamically configure the network to direct traffic towards the HTTP server host.

Benefits:

1. Network Flexibility: SDN enables the dynamic configuration of network flows, allowing us to easily redirect HTTP traffic towards the designated HTTP server host. This flexibility allows for efficient utilization of network resources.

2. Centralized Control: With an SDN controller like Ryu or POX, we can centrally manage and control the network, including the HTTP server configuration. This centralized control simplifies the management and deployment of the HTTP server compared to traditional networks.

3. Traffic Isolation: By isolating the HTTP server host from other hosts in the network, we can ensure that the server's resources are dedicated to serving HTTP requests. SDN allows us to define specific flows and rules to direct traffic only towards the HTTP server, ensuring efficient resource allocation.

4. Scalability: SDN networks are highly scalable, allowing for the easy addition or removal of hosts and the configuration of HTTP servers. This scalability ensures that the network can adapt to changing requirements and handle increased traffic loads.

5. Traffic Monitoring and Analytics: SDN controllers provide advanced monitoring and analytics capabilities. By monitoring the traffic directed towards the HTTP server host, we can gain insights into the usage patterns, performance metrics, and potential bottlenecks. This information can be used for network optimization and capacity planning.

6. Simplified Network Management: SDN networks abstract the complexity of network management using a centralized controller. This simplification allows for easier configuration and administration of the HTTP server host compared to traditional approaches.

By leveraging Mininet, SDN controllers, and configuring a host as an HTTP server, we can harness the benefits of SDN to create a scalable, flexible, and efficient network that serves web content.

The implementation:

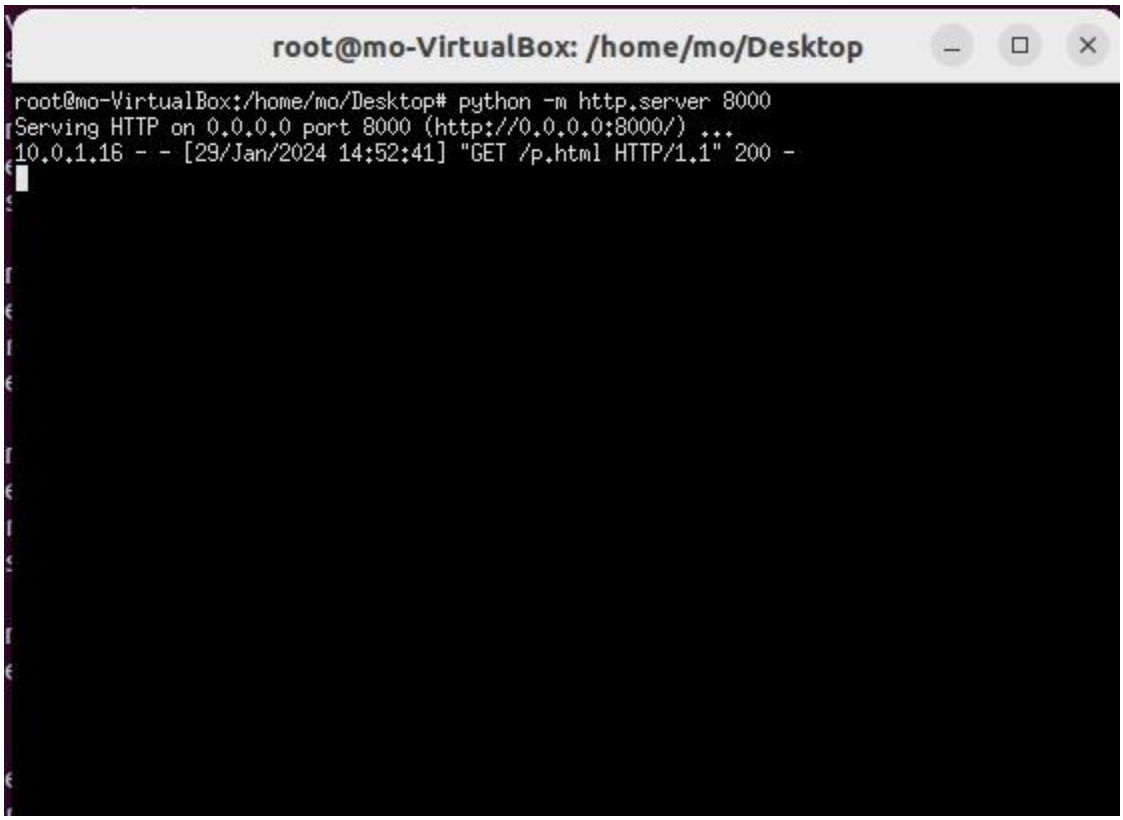
1- Start with creating the topo by this command *$ sudo mn --switch ovsk --nat --topo tree,depth=2,fanout=4 --controller remote,port=6100 --controller remote,port=6101 --ipbase=10.0.1.0/8 .*
2- Before creating the topo initiate the two-pox controller by these commands

controllers 1(c0): $ ./pox.py openflow.of_01 --port=6100forwarding.l2_pairs

controllers 2(c1): $ ./pox.py openflow.of_01 --port=6101 forwarding.l2_pairs

3- Open the CLI of h1 by this command *$h1 xterm.*

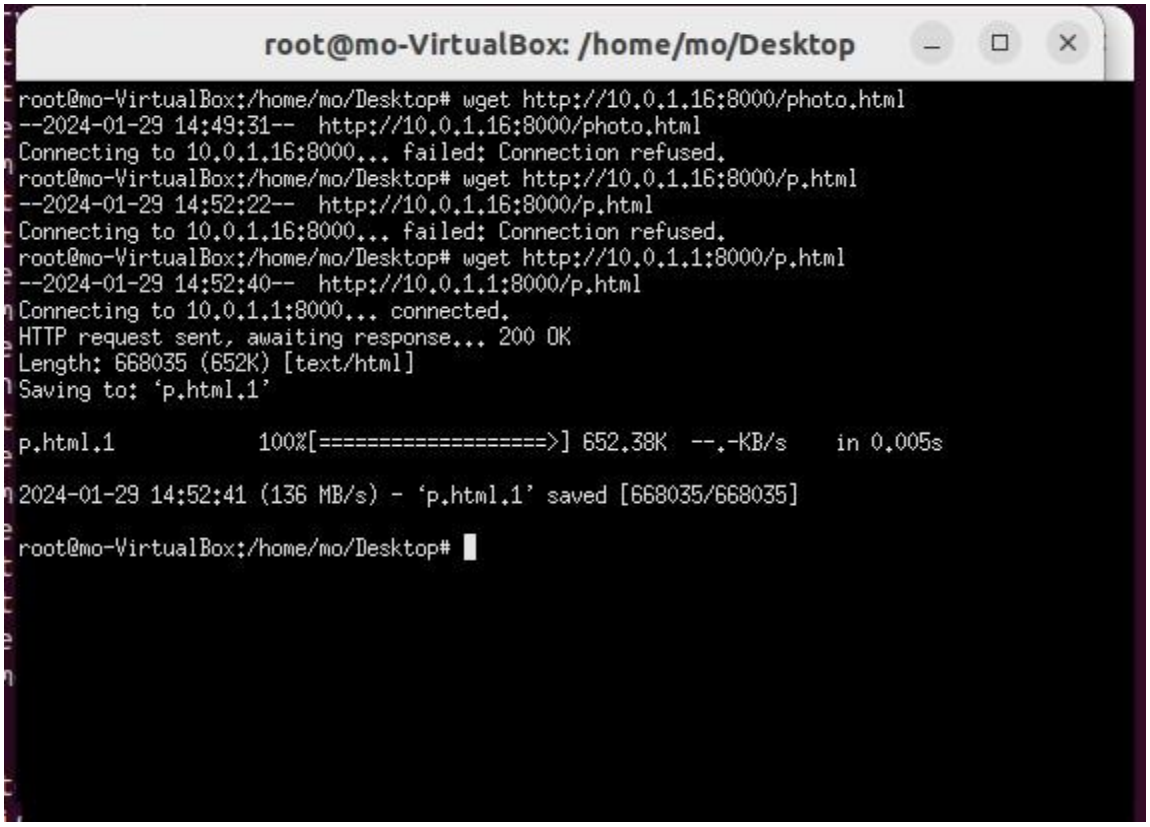
4- Make h1 the http server by this command *$python -m http.server 8000*



This command starts an HTTP server on port 8000.

5- we have a photo file on my Desktop with html format (p.html).

6- In another terminal, open a terminal (CLI) for the host where we want to retrieve the file (e.g., h16). And type this command *$wget http://10.0.1.16:8000/p.html*



The command `wget http://10.0.1.1:8000/p.html` is used to download a file named `p.html` from a web server running at the IP address `10.0.1.1` on port `8000`.

Here's a breakdown of the command:

I. `wget`: It is a command-line utility used for retrieving files from the web.

II. `http://10.0.1.1:8000/p.html`: This is the URL of the file you want to download. It consists of the following components:

III. `http://`: It specifies the protocol used for communication, in this case, Hypertext Transfer Protocol (HTTP).

IV. `10.0.1.1`: It represents the IP address of the web server hosting the file.

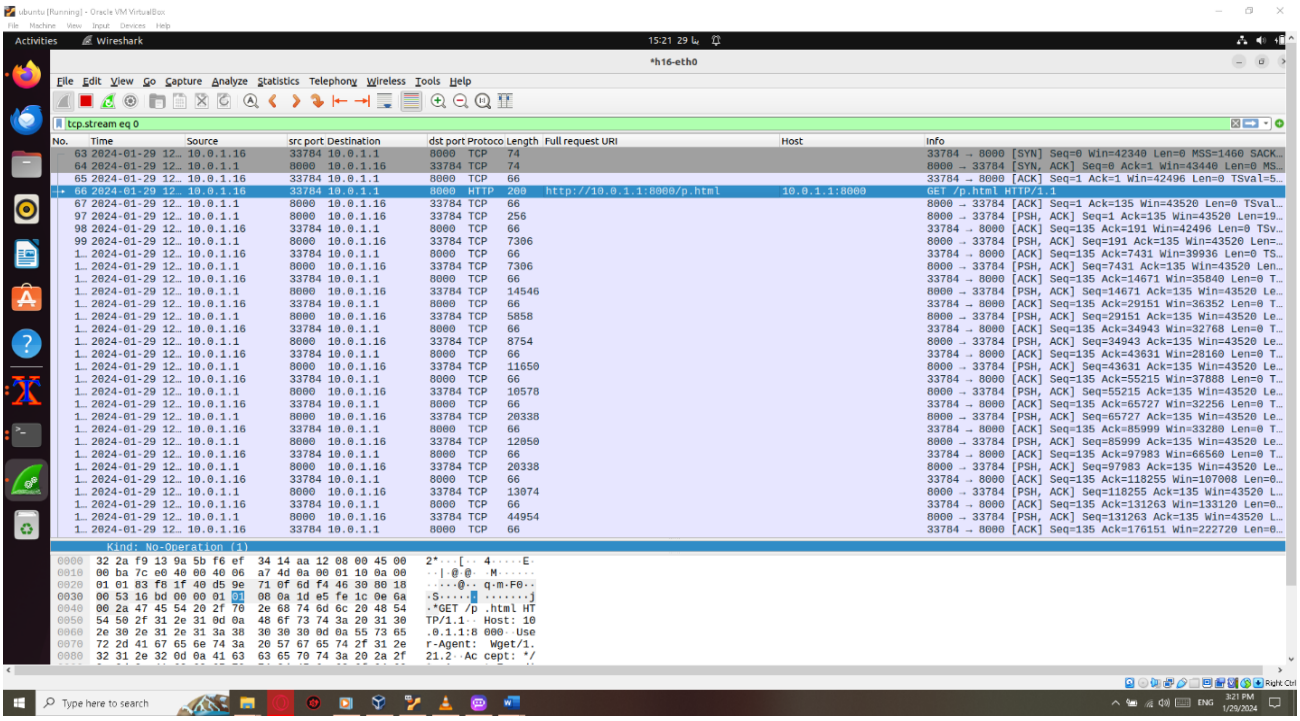
V. `:8000`: It denotes the port number on which the web server is listening. In this case, it is port `8000`.

VI. `/p.html`: It is the path to the specific file you want to download from the web server. In this case, it is `p.html`.

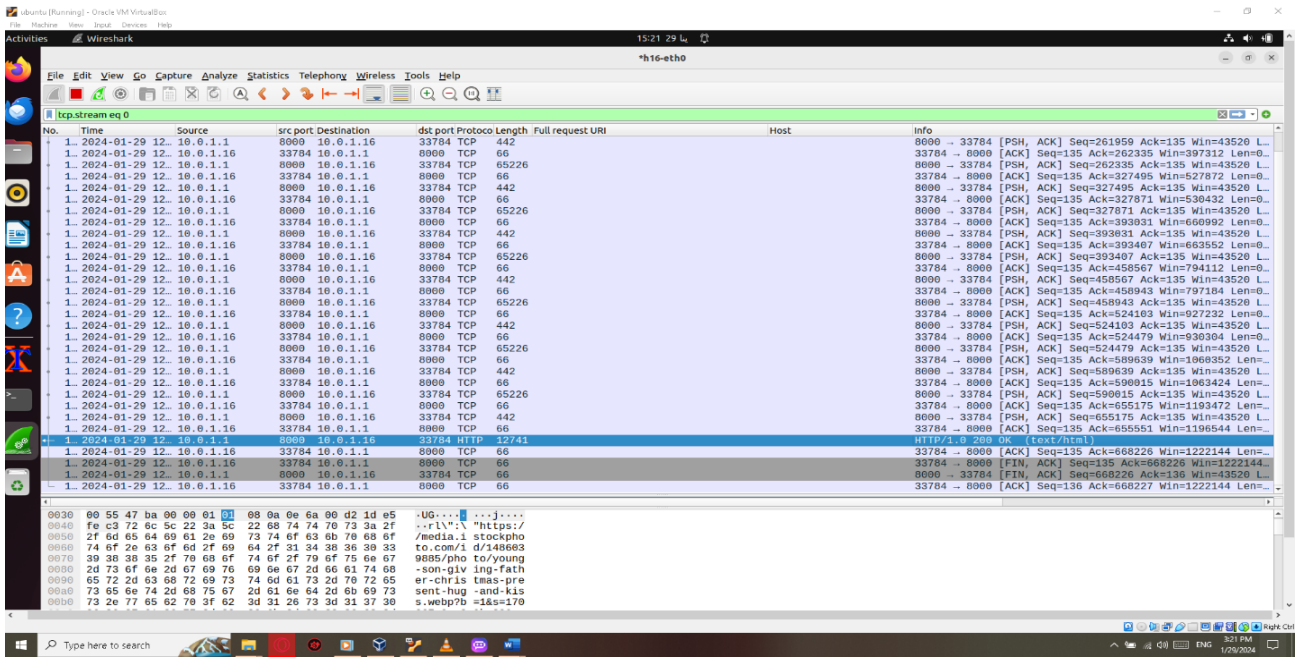
When you execute this command, `wget` will send an HTTP request to the specified URL. If the web server is running and accessible at `10.0.1.1:8000`, it will respond with the content of the `p.html` file.

`wget` will then save the file in the current directory on your local machine where you executed the command.

7- The capture with Wireshark



The request of the file.



the file received successfully.

2.4 TRACEROUTE

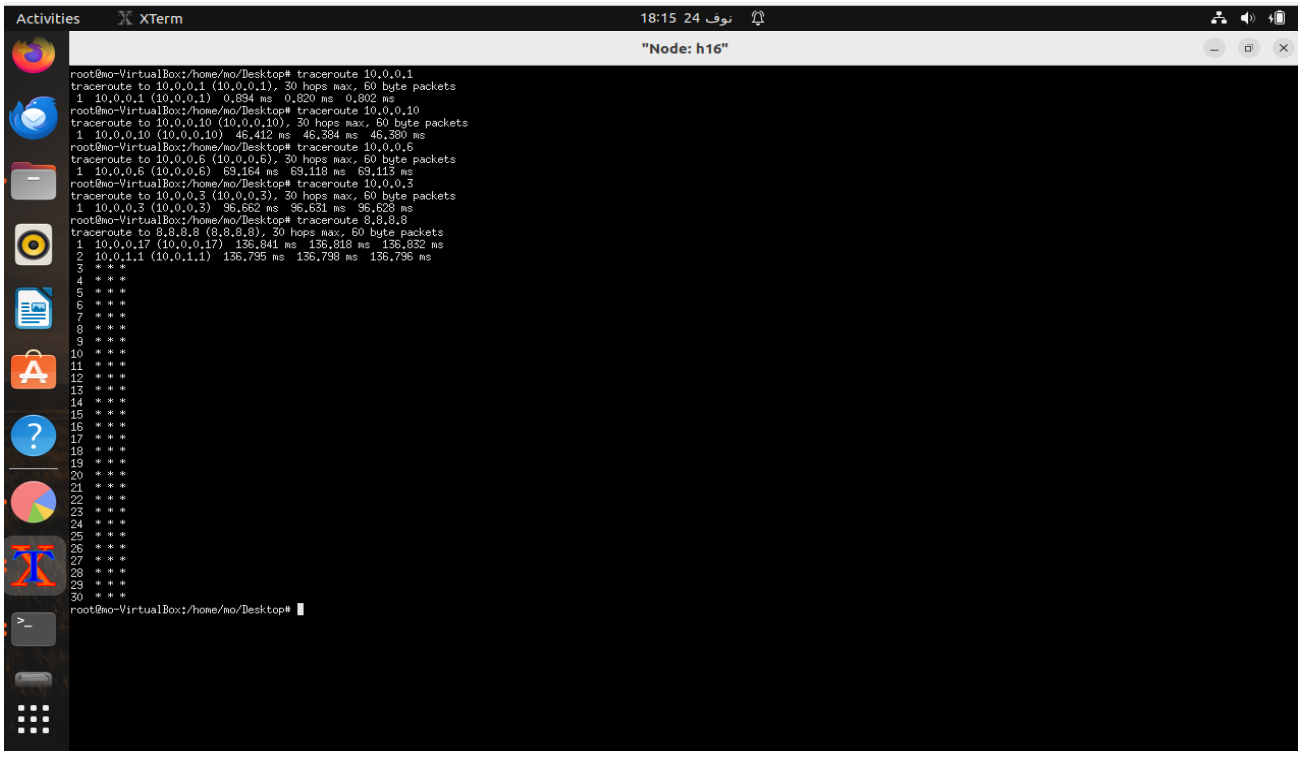
Traceroute is a network diagnostic tool that traces the route taken by packets sent from your computer to a destination IP address or domain name. It is commonly used to identify the network path and measure transit delays of packets across an IP network, such as the internet. The traceroute command is available on most operating systems, including Windows, macOS, and Linux, and it is typically used from the command line interface (CLI).

installation of the tool:

```
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
mo@mo-VirtualBox:~/Desktop$ sudo apt-get install traceroute
```

Traceroute on Google's DNS

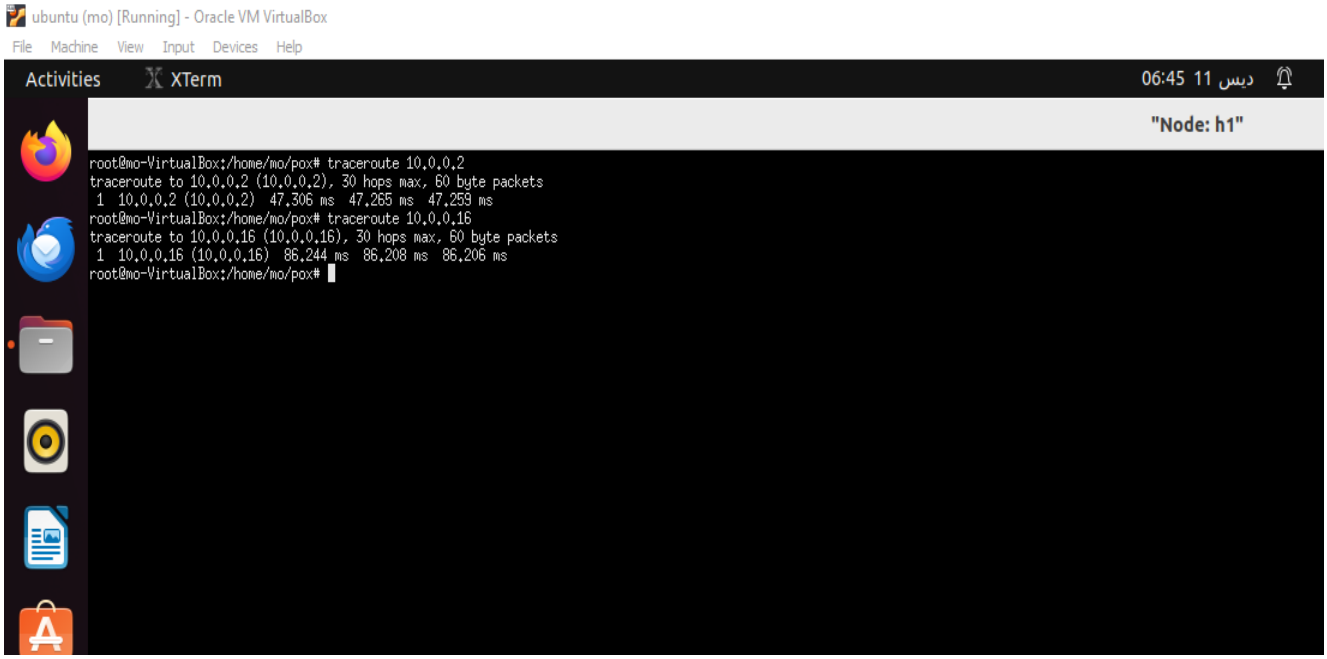
Stars error:



To solve this error, we had to change the network adapter to bridged adapter of the ubuntu.

So, it works successfully.

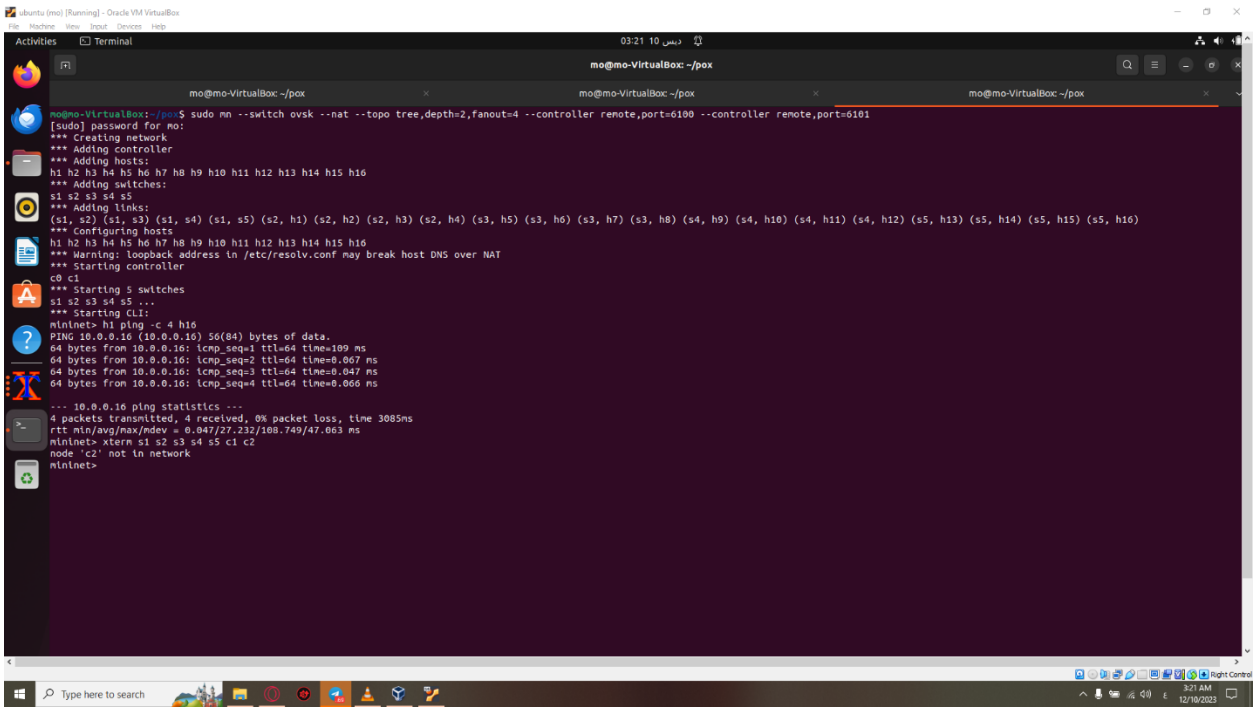
Traceroute on hosts: -



To reach h2 It passes through the s2 then reach the destination h2.

Also to reach h16 it passes through s2 and s1(core) then s5 to reach the destination.

We can also trace the forwarding packets or data by the flow entries as follows:



h1 ping h16

The command "ovs-ofctl dump-flow s1" in Mininet is used to display the flow rules present in the Open vSwitch (OVS) datapath associated with switch "s1". Here's a breakdown of the command

"ovs-ofctl": This is the command-line tool for interacting with OpenFlow switches using the Open vSwitch controller.

dump-flow": This subcommand is used to retrieve and display the existing flow rules in the specified switch.

s1": This refers to the switch named "s1" in the Mininet network topology."

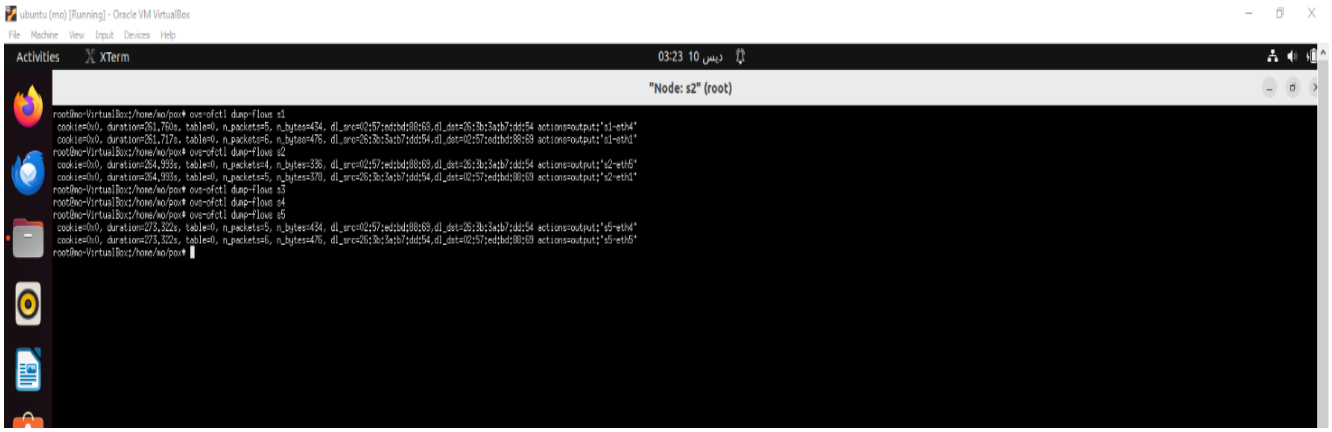
When you run this command, Mininet will communicate with the OVS switch "s1" and retrieve all the flow rules currently installed in that switch. These flow rules define how packets are processed and forwarded within the network, including actions such as matching on specific packet fields and applying forwarding actions.

The output of the command will provide information about each flow rule, including its match fields, actions, priority, and other relevant details. This allows you to inspect and understand the existing flow

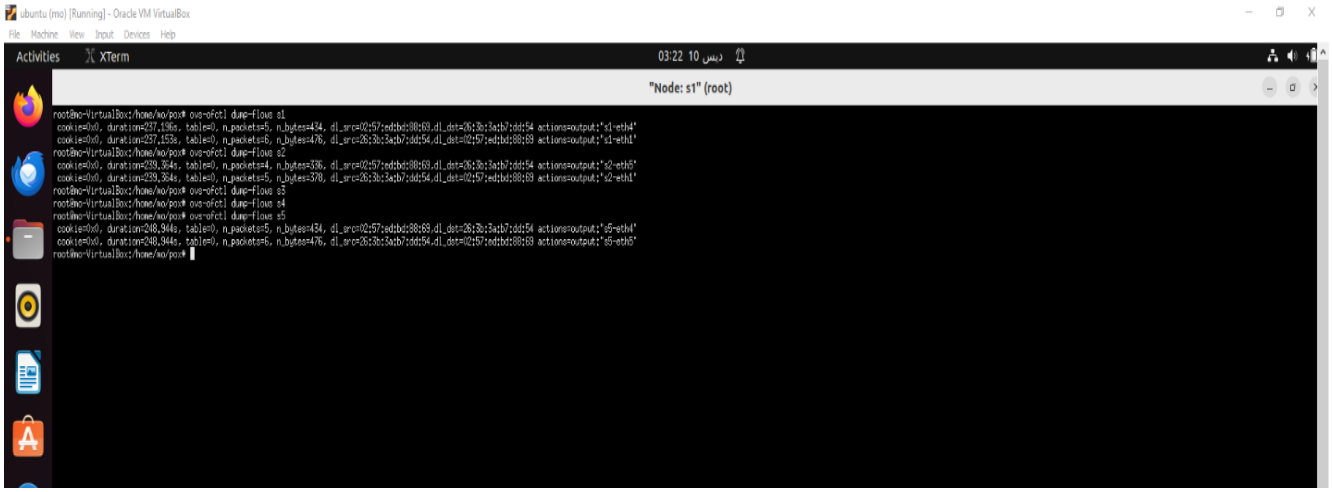
rules in the OpenFlow switch, aiding in troubleshooting, analysis, and verification of network behavior.

h1

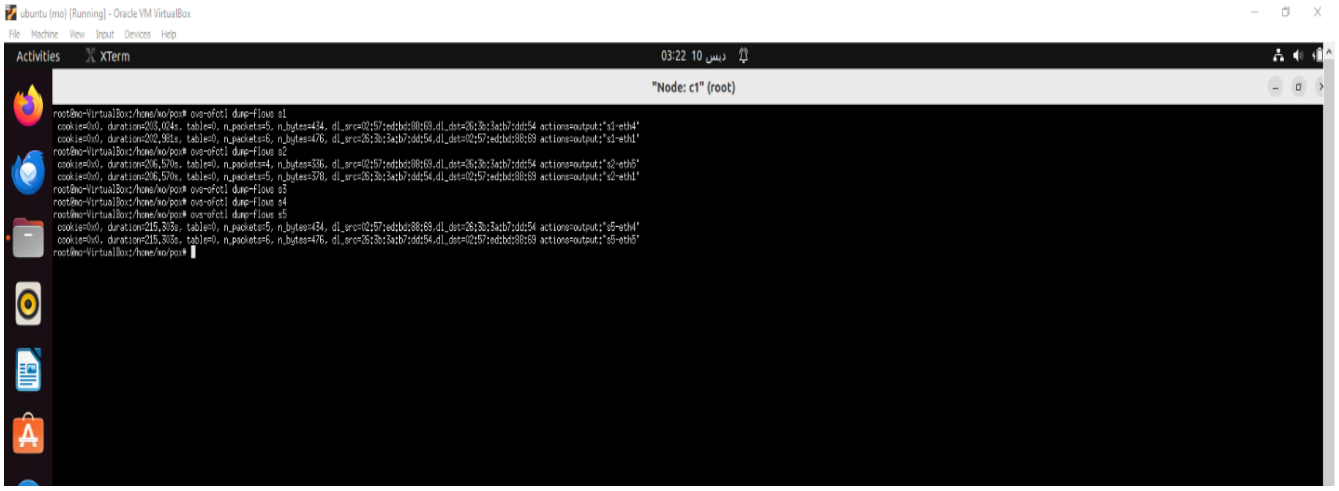
as we see s3 and s4 does not store the flue rules this means that the packets walk through the expected path.



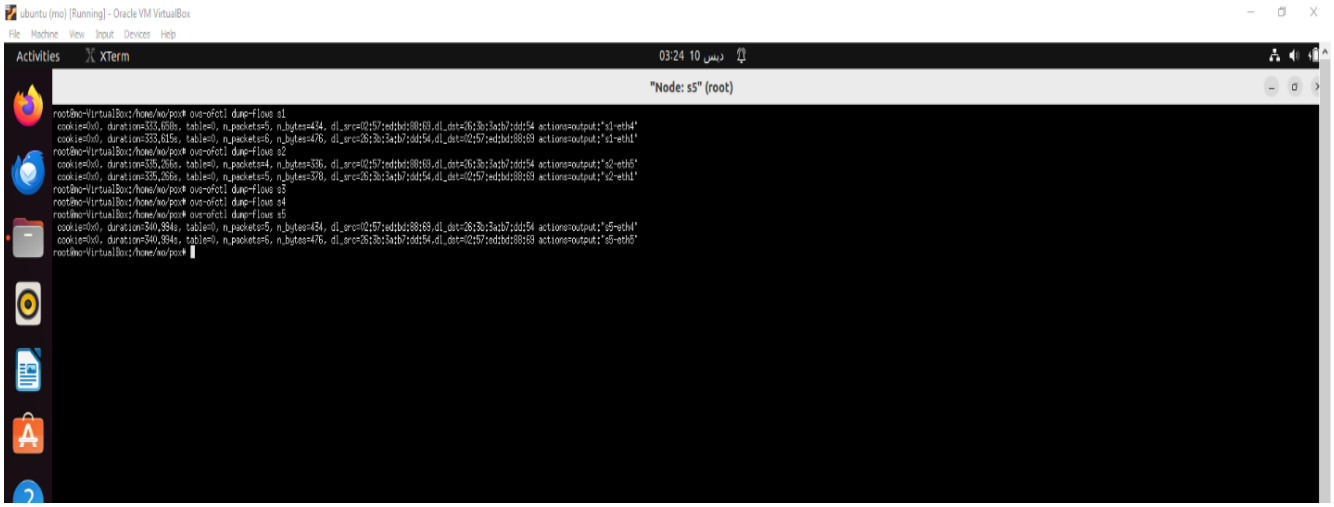
S2



S1


ubuntu (mo) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
XTerm
03:22 10 ديس
"Node: c1" (root)
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s1
cookie=0x0, duration=203.024s, table=0, n_packets=5, n_bytes=434, dl_src=02:57:edfbd:88:69,dl_dst=26:3b:3a:b7:dd:54 actions=output:"s1-eth4"
cookie=0x0, duration=202.981s, table=0, n_packets=6, n_bytes=476, dl_src=26:3b:3a:b7:dd:54,dl_dst=02:57:edfbd:88:69 actions=output:"s1-eth1"
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s2
cookie=0x0, duration=206.570s, table=0, n_packets=4, n_bytes=336, dl_src=02:57:edfbd:88:69,dl_dst=26:3b:3a:b7:dd:54 actions=output:"s2-eth5"
cookie=0x0, duration=206.570s, table=0, n_packets=5, n_bytes=378, dl_src=26:3b:3a:b7:dd:54,dl_dst=02:57:edfbd:88:69 actions=output:"s2-eth1"
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s3
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s4
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s5
cookie=0x0, duration=215.303s, table=0, n_packets=5, n_bytes=434, dl_src=02:57:edfbd:88:69,dl_dst=26:3b:3a:b7:dd:54 actions=output:"s5-eth4"
cookie=0x0, duration=215.303s, table=0, n_packets=6, n_bytes=476, dl_src=26:3b:3a:b7:dd:54,dl_dst=02:57:edfbd:88:69 actions=output:"s5-eth5"
root@mo-VirtualBox:/home/mo/pox#

C1

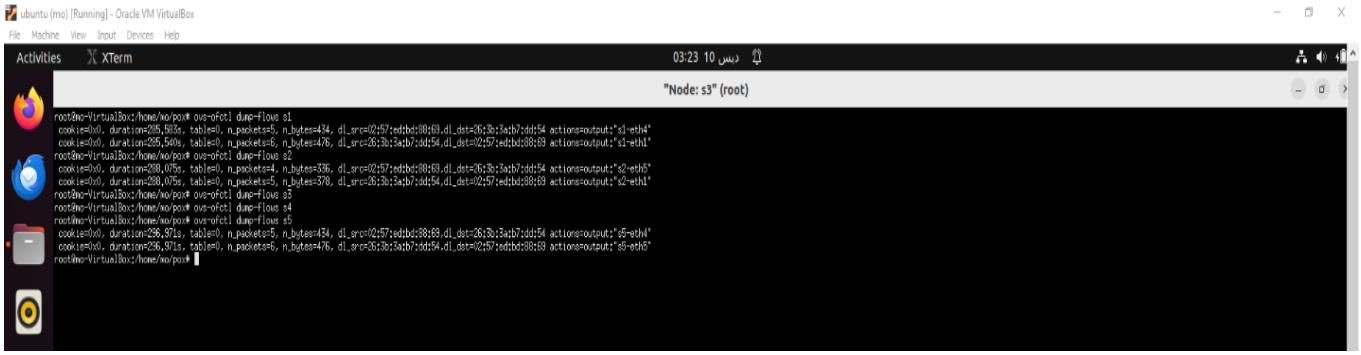

ubuntu (mo) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
XTerm
03:24 10 ديس
"Node: s5" (root)
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s1
cookie=0x0, duration=333.888s, table=0, n_packets=5, n_bytes=434, dl_src=02:57:edfbd:88:69,dl_dst=26:3b:3a:b7:dd:54 actions=output:"s1-eth4"
cookie=0x0, duration=333.815s, table=0, n_packets=6, n_bytes=476, dl_src=26:3b:3a:b7:dd:54,dl_dst=02:57:edfbd:88:69 actions=output:"s1-eth1"
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s2
cookie=0x0, duration=335.266s, table=0, n_packets=4, n_bytes=336, dl_src=02:57:edfbd:88:69,dl_dst=26:3b:3a:b7:dd:54 actions=output:"s2-eth5"
cookie=0x0, duration=335.266s, table=0, n_packets=5, n_bytes=378, dl_src=26:3b:3a:b7:dd:54,dl_dst=02:57:edfbd:88:69 actions=output:"s2-eth1"
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s3
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s4
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s5
cookie=0x0, duration=340.394s, table=0, n_packets=5, n_bytes=434, dl_src=02:57:edfbd:88:69,dl_dst=26:3b:3a:b7:dd:54 actions=output:"s5-eth4"
cookie=0x0, duration=340.394s, table=0, n_packets=6, n_bytes=476, dl_src=26:3b:3a:b7:dd:54,dl_dst=02:57:edfbd:88:69 actions=output:"s5-eth5"
root@mo-VirtualBox:/home/mo/pox#

S5

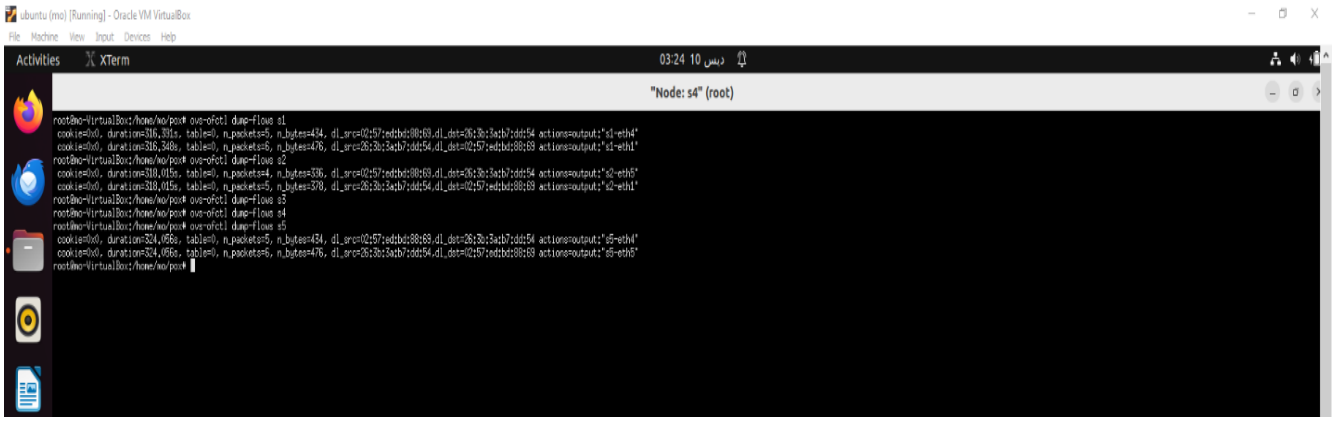

ubuntu (mo) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
XTerm
07:04 11 ديس
"Node: h16"
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s1
cookie=0x0, duration=324.755s, table=0, n_packets=4, n_bytes=336, dl_src=a6:5a:24:17:c2:3f,dl_dst=de:02:c4:68:3b:33 actions=output:"s1-eth4"
cookie=0x0, duration=324.755s, table=0, n_packets=5, n_bytes=378, dl_src=de:02:c4:68:3b:33,dl_dst=a6:5a:24:17:c2:3f actions=output:"s1-eth1"
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s2
cookie=0x0, duration=327.896s, table=0, n_packets=4, n_bytes=336, dl_src=a6:5a:24:17:c2:3f,dl_dst=de:02:c4:68:3b:33 actions=output:"s2-eth5"
cookie=0x0, duration=327.514s, table=0, n_packets=5, n_bytes=434, dl_src=de:02:c4:68:3b:33,dl_dst=a6:5a:24:17:c2:3f actions=output:"s2-eth1"
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s3
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s4
root@mo-VirtualBox:/home/mo/pox# ovs-ofctl dump-flows s5
cookie=0x0, duration=337.028s, table=0, n_packets=5, n_bytes=434, dl_src=a6:5a:24:17:c2:3f,dl_dst=de:02:c4:68:3b:33 actions=output:"s5-eth4"
cookie=0x0, duration=337.028s, table=0, n_packets=6, n_bytes=476, dl_src=de:02:c4:68:3b:33,dl_dst=a6:5a:24:17:c2:3f actions=output:"s5-eth5"
root@mo-VirtualBox:/home/mo/pox#

h16

we found that all switches store the flow entries as follows:



S3



S4

2.5 THE SECOND UPDATED TOPOLOGY

- **Components**: 2 POX controllers for redundancy, 2 core Open vSwitches (OVS), 4 access switches, 16 hosts
- **Configuration**: The network was further updated to include an additional core OVS switch, ensuring even greater redundancy and load balancing. This updated topology included two core OVS switches connected to the POX controllers, and each core switch was connected to two access switches, which in turn connected to 16 hosts. The entire network configuration was implemented using a Python script rather than manual commands in the terminal.

Reasons for the Second Update:

1. **Avoiding Single Point of Failure**: Adding an additional core switch eliminates the single point of failure at the core level, enhancing network resilience.

2. **Improved Load Balancing**: Two core switches distribute the traffic load more evenly, preventing congestion and improving network performance.
3. **Enhanced Redundancy**: Further increases redundancy, ensuring that the network remains operational even if one core switch fails.
4. **Automated Deployment**: Implementing the network with a Python script automates the setup process, reducing human error, saving time, and ensuring consistency in configuration.

By making these updates, the network becomes more robust, scalable, and easier to manage, with enhanced redundancy and performance.

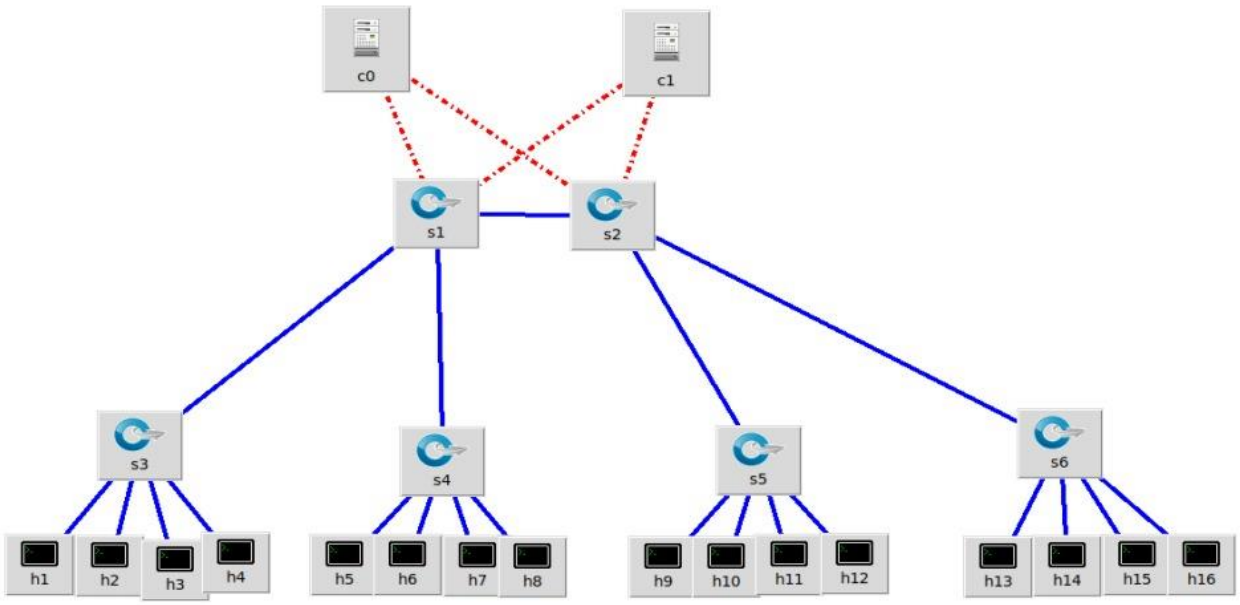


Figure 2.3. THE LAST UPDATED TOPOLOGY

The python script that used to create the topology

```
from mininet.net import Mininet

from mininet.node import Controller, RemoteController, OVSKernelSwitch

from mininet.cli import CLI


def myNetwork():

    net = Mininet(controller=RemoteController, switch=OVSKernelSwitch)
```

```
    # Add controllers
    c0 = net.addController(name='c0', controller=RemoteController, ip='127.0.0.1',
protocol='tcp', port=6102)
    c1 = net.addController(name='c1', controller=RemoteController, ip='127.0.0.1',
protocol='tcp', port=6103)


    # Add core switches
    core1 = net.addSwitch('core1')
    core2 = net.addSwitch('core2')


    # Add access switches
    access1 = net.addSwitch('access1')
    access2 = net.addSwitch('access2')
    access3 = net.addSwitch('access3')
    access4 = net.addSwitch('access4')


    # Connect core switches to access switches
    net.addLink(core1, access1)
    net.addLink(core1, access2)
    net.addLink(core2, access3)
    net.addLink(core2, access4)


    # Add link between core switches
    net.addLink(core1, core2)


    # Add hosts
    hosts = []
    for i in range(1, 17):
        host = net.addHost('h{}'.format(i), defaultRoute='via eth0')
        hosts.append(host)
        if i <= 4:
```

```
            net.addLink(host, access1)
        elif i <= 8:
            net.addLink(host, access2)
        elif i <= 12:
            net.addLink(host, access3)
        else:
            net.addLink(host, access4)

    # Enable NAT
    nat = net.addNAT('nat')
    net.addLink(nat, core1)
    net.addLink(nat, core2)

    # Build and start the network
    net.build()

    core1.start([c0])
    core2.start([c1])

    net.start()
    nat.cmd('sysctl net.ipv4.ip_forward=1')  # Enable IP forwarding on NAT
    nat.cmd('iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE')  # Enable NAT
on NAT interface

    # Set default gateway on each host
    for i in range(1, 17):
        host = net.get('h{}'.format(i))
        host.setDefaultRoute('via {}'.format(host.defaultIntf().IP()))

    CLI(net)
    net.stop()
```

```
if __name__ == '__main__':
    myNetwork()
```

Let's break down this code step-by-step to understand its functionality:

```python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSKernelSwitch
from mininet.cli import CLI
```

- Importing Modules: This section imports necessary classes and functions from Mininet. `Mininet` is the core class for creating and managing networks, `Controller` and `RemoteController` represent the controller nodes, and `OVSKernelSwitch` is the class for creating Open vSwitch instances. `CLI` is used for the command-line interface to interact with the network.

```python
def myNetwork():
net = Mininet(controller=RemoteController, switch=OVSKernelSwitch)
```

- Network Initialization: This line initializes a Mininet instance with `RemoteController` and `OVSKernelSwitch` as default types for controllers and switches.

```python
    # Add controllers
    c0 = net.addController(name='c0', controller=RemoteController, ip='127.0.0.1', protocol='tcp', port=6102)
    c1 = net.addController(name='c1', controller=RemoteController, ip='127.0.0.1', protocol='tcp', port=6103)
```

- Adding Controllers: Two remote controllers (`c0` and `c1`) are added to the network. They are both located at `127.0.0.1` but listen on different TCP ports (6102 and 6103).

```python
    # Add core switches
    core1 = net.addSwitch('core1')
    core2 = net.addSwitch('core2')

    # Add access switches
    access1 = net.addSwitch('access1')
    access2 = net.addSwitch('access2')
    access3 = net.addSwitch('access3')
    access4 = net.addSwitch('access4')
```

- Adding Switches: This section adds two core switches (`core1` and `core2`) and four access switches (`access1`, `access2`, `access3`, and `access4`) to the network.

```python
    # Connect core switches to access switches
    net.addLink(core1, access1)
    net.addLink(core1, access2)
    net.addLink(core2, access3)
    net.addLink(core2, access4)

    # Add link between core switches
    net.addLink(core1, core2)
```

- Connecting Switches: The core switches are connected to the access switches. Additionally, a direct link is created between the two core switches to ensure redundancy and load balancing.

```python
    # Add hosts
    hosts = []
    for i in range(1, 17):
        host = net.addHost('h{}'.format(i), defaultRoute='via eth0')
        hosts.append(host)
        if i <= 4:
            net.addLink(host, access1)
        elif i <= 8:
            net.addLink(host, access2)
        elif i <= 12:
            net.addLink(host, access3)
        else:
            net.addLink(host, access4)
```

- Adding Hosts: Sixteen hosts are added to the network. They are evenly distributed across the four access switches (`access1`, `access2`, `access3`, `access4`). Each host is assigned a default route through its `eth0` interface.

```python
    # Enable NAT
    nat = net.addNAT('nat')
    net.addLink(nat, core1)
    net.addLink(nat, core2)
```

- NAT Configuration: A Network Address Translation (NAT) node is added and linked to both core switches (`core1` and `core2`). This allows the private network to access external networks.

```python
    # Build and start the network
    net.build()


    core1.start([c0])
    core2.start([c1])
```

- Building and Starting the Network: The network is built, and then each core switch is started with its respective controller (`core1` with `c0` and `core2` with `c1`).

```python
    net.start()
    nat.cmd('sysctl net.ipv4.ip_forward=1')  # Enable IP forwarding on NAT
    nat.cmd('iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE')  # Enable NAT
on NAT interface
```

- Starting Network and Configuring NAT: The network is started, and IP forwarding is enabled on the NAT node to allow traffic to be forwarded. The `iptables` command sets up masquerading for outgoing packets, enabling NAT.

```python
  # Set default gateway on each host
    for i in range(1, 17):
        host = net.get('h{}'.format(i))
        host.setDefaultRoute('via {}'.format(host.defaultIntf().IP()))
```

- Setting Default Gateway: Each host's default route is set to its primary interface's IP address, ensuring proper routing of packets.

```python
    CLI(net)

    net.stop()
```

- CLI and Stopping the Network: The CLI is started, allowing interactive control of the network. Once the CLI is exited, the network is stopped.

```python
if __name__ == '__main__':
    myNetwork()
```

- Script Entry Point: This ensures that `myNetwork()` function is called when the script is executed, setting up and running the network.

This script automates the creation of a robust and scalable SDN network topology, complete with redundancy, NAT functionality, and interactive control via the CLI.

Chapter 3: Implementation of Dos attack scenario and Detection

3.1 INTRODUCTION

SDN security is more challenging than security in traditional networking, and among such challenges are both denial of service (DoS) and distributed denial of service (DDoS) attacks. Generally, DoS and DDoS attacks have become major threats to computer networks. In such attacks, by exhausting resources, several services are disabled, and the network's performance is downgraded. These attacks are considered successful when the attacker intentionally consumes resources that prevent hosts from using the targeted service. Different approaches can be used to perform DoS/DDoS attacks, including network-based approaches, such as flooding through TCP SYN, ICMP, or UDP packets, and host-based approaches, where one or several hosts target specific applications to exploit their memory structure, their authentication protocol, or a particular algorithm.

In SDNs, DoS/DDoS attacks could flood the control plane, the data plane, or the control plane bandwidth. Attacking the control plane of an SDN could result in the failure of the entire network. The controller is considered the brain of the network, where it manages many switches and applications. A DoS/DDoS attack on an SDN data plane could fill up the limited flow table memory of the OpenFlow switch. Once the switch memory is full, the switch is unable to receive new flows or to install new flow rules received from the controller, and hence packets are dropped. In this scenario, the switch is unable to forward buffered packets until there is free memory space in the switch flow table.

Another target of DoS/DDoS attacks in the data plane is related to the generation of a massive number of new flows that do not match flow table entries defined by the switch. The switch buffers these packets before forwarding them to the controller through packet-in messages. However, if the rate of incoming new flows is high, the switch's buffer fills up, and then it forwards the entire packet to the controller instead of packet-in messages that contain headers only. This would result in higher communication bandwidth consumption, as well as delays in the installation of new flow rules. Consequently, the overall network performance is severely impacted, leading to degraded services for legitimate users.

Therefore, there is a compelling need to provide a solution to mitigate DoS/DDoS attacks on SDNs. Such a solution should protect the communication channel between the OpenFlow switches and the SDN controller, handle packet-in messages properly, and maintain the functionality of forwarding legitimate traffic. Moreover, the envisaged solution should be able to discriminate between malicious

and benign traffic in an efficient and effective manner while maintaining overall high performance. Implementing robust security measures and constantly monitoring network traffic for anomalies are crucial steps in defending against these types of attacks. Additionally, leveraging machine learning and artificial intelligence to predict and respond to potential threats can enhance the resilience of SDN infrastructures against DoS/DDoS attacks.

Project Overview

This project investigates Distributed Denial of Service (DoS) attacks in Software-Defined Networking (SDN) environments and explores detection and mitigation strategies. The research involves simulating DoS attacks, implementing various countermeasures, and evaluating their effectiveness in an SDN setup. By conducting a thorough analysis of DoS threats and solutions, this project aims to enhance the security and robustness of SDN infrastructures.

The rapid evolution of network technologies has brought about significant advancements in how networks are designed, managed, and secured. One of the most transformative developments in recent years is Software-Defined Networking (SDN), which decouples the control plane from the data plane, allowing for centralized network management and programmability. This paradigm shift offers numerous benefits, including improved.

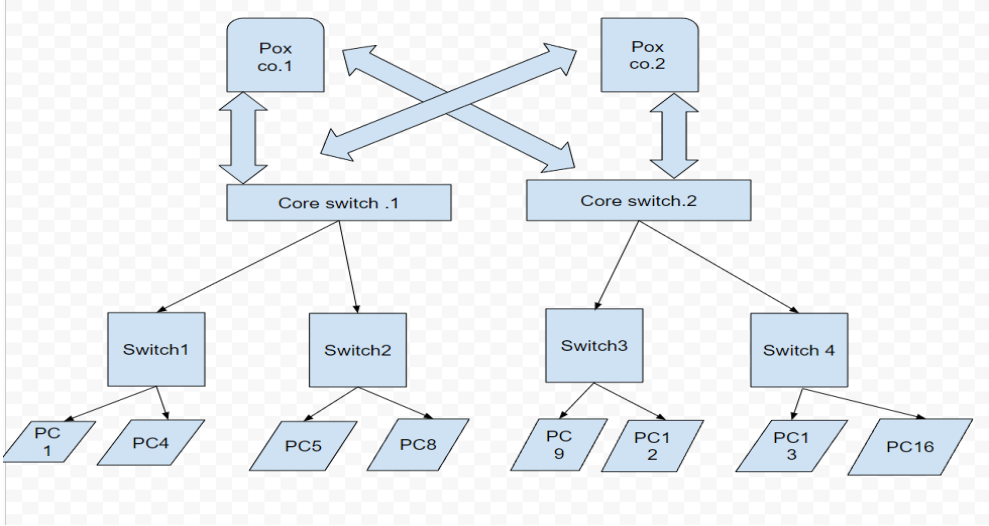


Figure 3.1. Modified topo with full availability

Importance of the Study

The increasing adoption of SDN in modern network infrastructures highlights the need for robust security measures. SDN's centralized control plane, while offering numerous advantages, also presents a single point of failure that can be exploited by DoS attacks. Understanding and mitigating these attacks is crucial to ensuring the stability, reliability, and security of SDN-based networks. This study addresses these concerns by providing a comprehensive analysis of DoS attack mechanisms and proposing effective mitigation strategies.

In addition, as more enterprises and service providers transition to SDN, the potential impact of security vulnerabilities grows significantly. The success of SDN deployments depends on the ability to safeguard the network from disruptions that could lead to service outages, data breaches, and financial losses. Ensuring the resilience of SDN against DoS and DDoS attacks is therefore not only a technical necessity but also a business imperative.

Moreover, the dynamic nature of SDN environments, characterized by frequent changes in network configurations and policies, necessitates adaptive and scalable security solutions. Traditional security approaches may not suffice in such a flexible and programmable network landscape. This study emphasizes the need for innovative security frameworks that can dynamically respond to emerging threats and evolving attack vectors.

Furthermore, the outcomes of this research have broader implications for the development of future network technologies. By addressing the security challenges of SDN, this study contributes to the foundational knowledge required to build more secure and robust network architectures. The insights gained can be applied to other programmable network paradigms, such as Network Function Virtualization (NFV) and edge computing, further enhancing the overall security posture of next-generation networks.

Finally, this study aims to bridge the gap between theoretical research and practical implementation. By evaluating real-world attack scenarios and testing mitigation techniques in simulated environments, the findings offer actionable recommendations for network administrators and security professionals. This practical relevance ensures that the proposed solutions can be effectively deployed to protect SDN infrastructures in diverse operational contexts

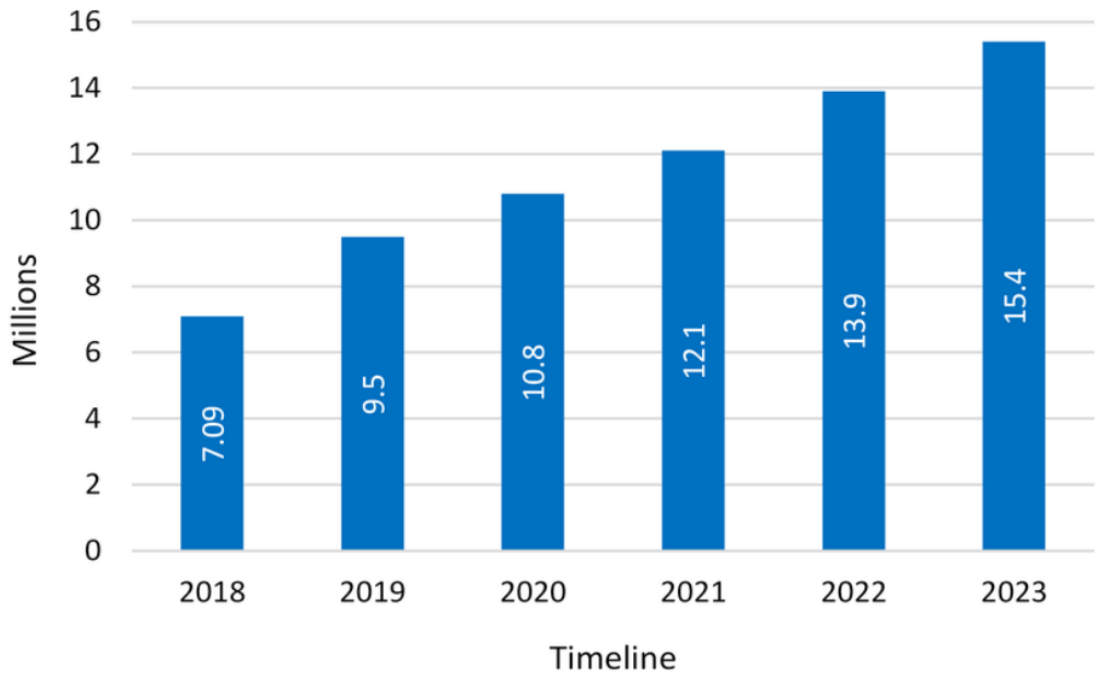


Figure 3.2. Global Trend of DoS Attacks 2018-2023

Objectives and Scope

Analyze the Impact: Examine how DoS attacks affect SDN networks' performance and stability.

Explore Techniques: Investigate various detection and mitigation techniques tailored for SDN.

Implement and Evaluate: Simulate DoS attacks in a controlled SDN environment, implement countermeasures, and evaluate their effectiveness.

Contribute to Research: Provide insights and recommendations for future research in securing SDN against DoS threats.

Structure of the Document

The document is structured into several sections to systematically address the study's objectives. The background and literature review section provides foundational knowledge on SDN and DoS attacks. The problem statement identifies the challenges and the need for improved mitigation techniques. The methodology outlines the research design, data collection methods, and network setup. Detection and mitigation techniques are discussed in detail, followed by a case study demonstrating the implementation. The evaluation and results section analyzes the effectiveness of the proposed solutions. The document concludes with a summary of findings, contributions, limitations, and recommendations for future work.

3.2 BACKGROUND AND LITERATURE REVIEW

Overview of Software-Defined Networking

Software-Defined Networking (SDN) is an innovative approach to network management that separates the control plane from the data plane, enabling centralized control and programmable network configurations. Key concepts include:

Control Plane: Manages the network by making decisions about where traffic should be sent.

Data Plane: Forwards traffic to the selected destination based on the control plane’s decisions.

SDN Controller: A centralized software entity that controls the behavior of the entire network by communicating with network devices.

SDN decouples the network control and forwarding functions, allowing network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services.

SDN Architecture (Dos attack view)

SDN architecture is typically divided into three layers:

Application Layer: Contains network applications and business logic that define network behavior. Examples include network management applications, security applications, and monitoring tools.

Control Layer: Includes the SDN controller, which acts as the brain of the network, translating application requirements into network configurations. The controller communicates with both the application and infrastructure layers.

Infrastructure Layer: Comprises physical and virtual switches that handle data forwarding. These devices execute the commands received from the SDN controller to manage data traffic.

Benefits of SDN

Flexibility: SDN allows dynamic network configuration and management, making it easy to adapt to changing network demands.

Scalability: The centralized control model simplifies network scaling, enabling administrators to manage large networks efficiently.

Cost Efficiency: SDN reduces operational costs by utilizing commodity hardware and simplifying network management tasks.

Enhanced Security: SDN's centralized control allows for more comprehensive security policies and faster response to security threats.

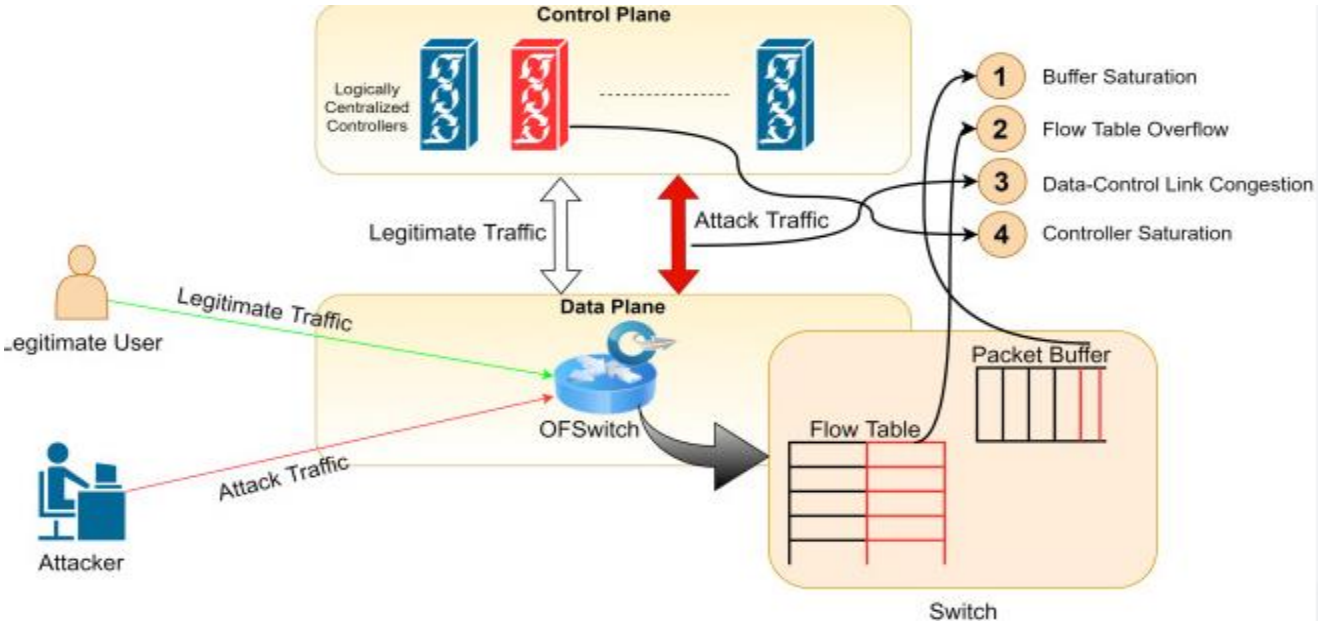


Figure 3.3. SDN DOS attack view over the network

3.3 UNDERSTANDING DOS ATTACKS

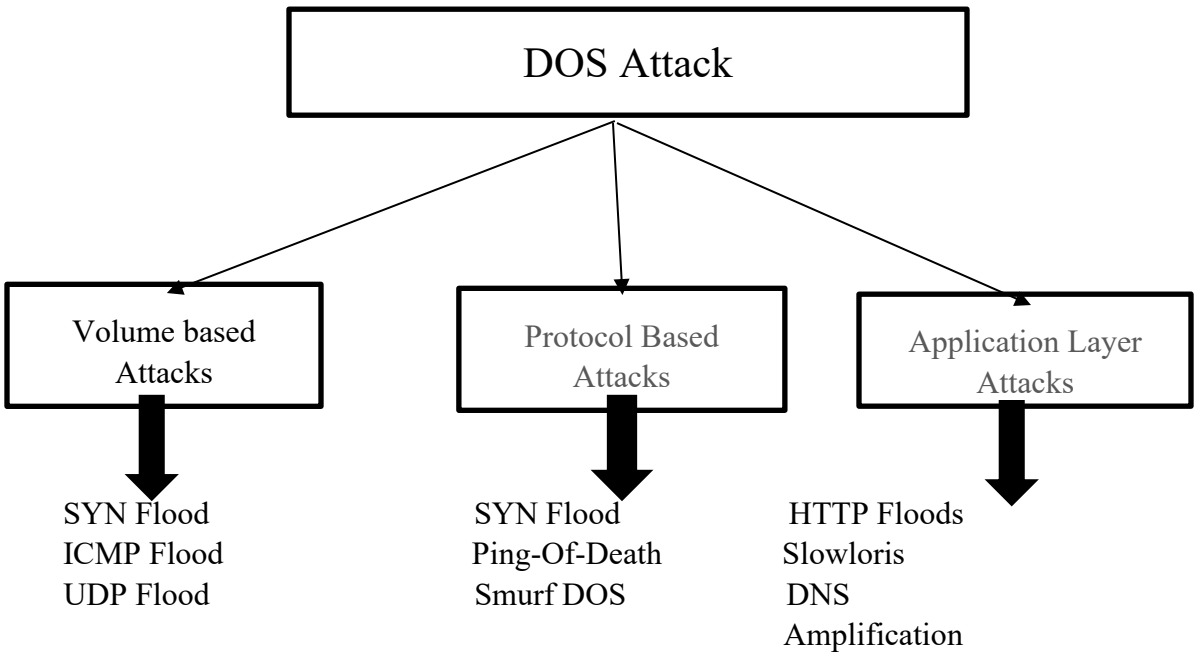
3.3.1 Definition and Types of DoS Attacks

A Denial of Service (DoS) attack involves multiple systems overwhelming a target, such as a server or network, with a flood of illegitimate traffic, rendering it unavailable to legitimate users. Common types of DoS attacks include:

Volume-Based Attacks: These attacks aim to overwhelm the target's bandwidth with massive amounts of data. Examples include UDP floods and ICMP floods.

Protocol Attacks: These attacks exploit vulnerabilities in network protocols. Examples include SYN floods and Ping of Death.

Application Layer Attacks: These attacks target specific applications or services, making them unavailable to users. Examples include HTTP floods and Slow Loris attacks.



3.3.2 History and Evolution of DoS Attacks

The earliest forms of Denial of Service (DoS) attacks were primarily volumetric in nature, focusing on overwhelming a network's bandwidth to render services unavailable. These attacks typically involved flooding the target with an excessive amount of traffic, consuming all available bandwidth and preventing legitimate users from accessing the network resources. Common methods included:

Ping Floods: Attackers sent many ICMP Echo Request (ping) packets to the target, overwhelming its ability to respond.

SYN Floods: Attackers exploited the TCP handshake process by sending numerous SYN packets with a spoofed IP address, causing the target to allocate resources for half-open connections.

UDP Floods: Attackers sent large volumes of UDP packets to random ports on the target, forcing it to process each packet and generate ICMP Destination Unreachable responses.

3.2.3 Rise of Botnets

As network defenses improved, attackers began to leverage botnets to amplify the scale and impact of DoS attacks. A botnet is a network of compromised computers (bots) controlled by an attacker. These botnets can be directed to simultaneously launch attacks, significantly increasing the volume of traffic and making it more challenging to mitigate. Botnets also facilitated the rise of Distributed Denial of Service (DDoS) attacks, where multiple compromised systems targeted a single network or service, creating a more dispersed and potent attack vector.

Modern DoS attacks have evolved into sophisticated multi-vector threats, combining multiple attack methods to exploit different vulnerabilities simultaneously. These attacks often target not only bandwidth but also application and protocol weaknesses. Some advanced techniques include:

- Application Layer Attacks: These attacks focus on exhausting the resources of specific applications or services rather than the entire network. Examples include HTTP floods, where attackers send numerous HTTP requests to a web server, and Slowloris attacks, which keep connections open for an extended period, exhausting server resources.

Protocol Exploitation: Attackers exploit vulnerabilities in network protocols to disrupt communication. Examples include DNS amplification attacks, where attackers send small DNS queries with a spoofed source IP address, causing the DNS server to respond with large responses to the target, and NTP reflection attacks, where Network Time Protocol servers are abused to amplify traffic directed at the target.

3.2.4 DoS Attacks in SDN Networks

The centralized nature of SDN networks introduces unique vulnerabilities that can be exploited by DoS attacks. In SDN, the control plane is decoupled from the data plane, with a centralized controller managing the entire network. This centralization, while beneficial for network management and programmability, also creates a single point of failure. Attackers can target the SDN controller with DoS attacks to disrupt the entire network. Specific methods include:

Control Plane Saturation: Attackers flood the SDN controller with malicious requests, overwhelming its processing capacity and causing it to become unresponsive. This can result in network-wide disruptions as the controller is unable to manage and configure the data plane.

Flow Table Overload: Attackers can send numerous unique packet flows to switches, causing their flow tables to become full. This forces the switches to forward all subsequent packets to the controller, further straining its resources.

Exploitation of Northbound and Southbound APIs: Attackers can exploit vulnerabilities in the APIs used for communication between the controller and applications (northbound) or between the controller and network devices (southbound). This can lead to unauthorized access, data corruption, or further amplification of DoS attacks.

3.2.5 Implications for Modern Networks

The evolution of DoS attacks poses significant challenges for modern networks, including SDN environments. Businesses must implement robust security measures to protect against these evolving threats. Key strategies include:

Traffic Filtering and Rate Limiting: Implementing traffic filters and rate limiting can help mitigate volumetric attacks by controlling the amount of traffic that reaches the network.

Anomaly Detection: Utilizing advanced anomaly detection systems can help identify and respond to unusual traffic patterns indicative of a DoS attack.

Redundancy and Load Balancing: Distributing network resources and implementing redundancy can help ensure that services remain available even during an attack.

Secure SDN Controllers: Ensuring that SDN controllers are secure and resilient is critical. This includes regular security updates, vulnerability assessments, and implementing failover mechanisms to maintain network operations.

3.2.6 Impact of DoS Attacks on Networks

Resource Exhaustion:

Controller Overload: A DoS attack can flood the SDN controller with malicious traffic, overwhelming its processing capacity. Studies have shown that under such attacks, the controller's response time can increase significantly, affecting network performance.

Switch Overload: DoS attacks can also target SDN switches, exhausting their flow table entries and causing legitimate traffic to be dropped. Research indicates that switch CPU usage can spike, leading to degraded network performance and potential outages.

Latency and Throughput:

DoS attacks can introduce significant delays in packet processing and reduce overall network throughput. For instance, one study demonstrated that a flood of malicious packets could increase network latency by up to 300% and decrease throughput by 50%.

Availability: The primary goal of DoS attacks is to disrupt the availability of network services. Research shows that sustained DoS attacks can lead to prolonged network downtimes, with recovery times varying based on the severity and mitigation strategies employed.

3.2.7 Specific Research Statistics:

Study by Dabbagh et al. (2019):

Impact on Latency: The average latency increased from 5 ms to 20 ms under DoS attacks.

Impact on Throughput: Network throughput dropped by approximately 40% during the attack period.

Study by Yazici et al. (2018):

Controller Load: The CPU utilization of the SDN controller increased from 30% to 90% under a DoS attack.

Flow Table Usage: The utilization of switch flow tables reached 100%, causing new legitimate flows to be dropped.

Study by Wang et al. (2020):

Packet Loss: Packet loss rates increased from less than 1% to over 25% during a DoS attack.

Recovery Time: The average time to restore normal operations after a DoS attack was found to be approximately 15 minutes.

3.3 DOS ATTACKS IN SDN

3.3.1 How DoS Attacks Target SDN

DoS attacks in SDN can exploit the centralized control plane, targeting the SDN controller with excessive traffic, which can lead to network-wide disruptions. Attacks may also target data plane devices, causing them to overflow with malicious packets. The unique architecture of SDN introduces new attack vectors and challenges for DDoS mitigation.

2.3.2 Examples of Notable DoS Attacks in SDN

Description:

In Open Flow-based SDN environments, switches forward packets based on flow rules established by the controller. When a switch receives a packet that does not match any existing flow rules, it forwards the packet to the controller for further instructions. Attackers exploit this behavior by sending a large volume of bogus (or spoofed) packets to the switch, each designed to miss existing flow rules.

Impact:

The switch, unable to handle these packets on its own, forwards them to the controller. This creates a flood of control messages from the switch to the controller. The controller becomes overwhelmed by the sheer volume of incoming control messages, consuming excessive processing power and memory. Legitimate control messages may be delayed or dropped, resulting in network performance degradation and potential service outages.

Mitigation Techniques:

Rate Limiting: Limit the rate at which packets are sent to the controller to prevent overload.

Anomaly Detection: Implement machine learning algorithms to detect and filter out malicious traffic patterns.

Access Control Lists (ACLs): Use ACLs at the switch level to filter out known malicious IP addresses or traffic patterns before they reach the controller.

3.3.3 Flow Table Overflow

Description:

Each Open Flow switch maintains a flow table that stores flow rules for packet forwarding. Flow tables have finite memory and can store only a limited number of entries. Attackers exploit this limitation by sending a high volume of unique packets, each designed to create a new flow entry in the table.

Impact:

The flow table quickly fills up with illegitimate entries. Once the flow table is full, the switch cannot install new flow rules for legitimate traffic. This results in legitimate packets being dropped or forwarded to the controller for handling, further increasing the load on the controller. Overall network performance is severely degraded as legitimate traffic cannot be efficiently processed.

Mitigation Techniques:

Flow Aggregation: Aggregate similar flows to reduce the number of required flow entries.

Dynamic Flow Rule Management: Implement strategies to prioritize and manage flow entries, such as removing stale or low-priority flows.

Flow Table Partitioning: Allocate separate flow table space for different types of traffic to ensure critical flows always have available space.

3.3.4 Controller Targeting

Description:

The SDN controller is the central brain of the network, making it a prime target for attackers. Attackers aim to overwhelm the controller by sending a high volume of requests, such as new flow setup requests or status queries. These requests consume the controller's CPU, memory, and network resources.

Impact:

The controller becomes overwhelmed and may become unresponsive or crash.As the controller is responsible for managing the entire network, its failure disrupts network operations.

Switches may be unable to obtain new flow rules or updates, causing packet loss and network instability. The overall network performance degrades, potentially leading to service outages for users.

Mitigation Techniques:

- Distributed Controllers: Deploy multiple controllers to share the load and provide redundancy.
- Rate Limiting: Implement rate limiting to control the number of requests that can be sent to the controller.
- Load Balancing: Use load balancing techniques to distribute requests across multiple controllers evenly.
- Controller Replication: Replicate controllers to ensure that if one fails, others can take over seamlessly.
- Anomaly Detection: Monitor traffic to the controller and use anomaly detection to identify and mitigate attack traffic.

3.3.5 Summary

These attacks target different components of the SDN infrastructure, exploiting their roles and limitations to degrade network performance or cause outages. Effective mitigation requires a combination of proactive monitoring, dynamic resource management, and redundancy to ensure the network remains resilient against such attacks.

3.4 PROBLEM STATEMENT

3.4.1 Challenges Posed by DoS Attacks in SDN

Software Defined Networking (SDN) is a new networking paradigm where forwarding hardware is decoupled from control decisions. It promises to dramatically simplify network management and enable innovation and evolution. In SDN, network intelligence is logically centralized in software-based controllers (the control plane), while network devices (OpenFlow Switches) become simple packet-forwarding devices (the data plane) that can be programmed via an open interface (OpenFlow protocol). Such decoupling of the control plane from the data plane introduces various challenges that include security, reliability, load balancing, and traffic engineering. Dreadful security challenges in SDNs are denial of service (DoS) and distributed denial of service (DDoS) attacks. For instance, in SDNs, DoS/DDoS attacks could flood the control plane, the data plane, or the communication channel. Attacking the control plane could result in failure of the entire network, while attacking the data plane or the communication channel results in packet drop and network unavailability. In this paper we deliver several contributions that shed light on the field of DoS/DDoS attacks in SDNs, providing a complete background about the area, including attacks and analysis of the existing solutions. In particular, our contributions can be summarized as follow: we review and systematize the state-of-the-art solutions that address both DoS and DDoS attacks in SDNs through the lenses of intrinsic and extrinsic approaches. Moreover, the discussed countermeasures are organized accordingly to their focus, be it on detection, mitigation, prevention, or graceful degradation. Further, we survey the different approaches and tools adopted to implement the revised solutions. Finally, we also highlight possible future research directions to address DoS/DDoS attacks in SDNs.

3.4.2 Existing Solutions and Their Limitations

Current DoS mitigation solutions often rely on traditional network security measures, which may not be well-suited for SDN environments. Limitations of existing solutions include:

Scalability Issues: Traditional solutions may struggle to scale with the dynamic nature of SDN networks.

False Positives/Negatives: Many detection techniques have high rates of false positives and negatives, leading to ineffective mitigation.

Performance Overhead: Some mitigation techniques introduce significant performance overhead, impacting network efficiency.

Denial of Service (DoS) attacks have been a persistent threat to network security, causing significant disruptions by overwhelming targeted systems with excessive traffic. In traditional networks, managing and mitigating DoS attacks is complicated due to the vertically integrated data and control planes, which are often manufacturer specific. Software-Defined Networking (SDN) offers a promising solution by decoupling the data plane from the control plane, allowing for centralized and flexible network management. This paper compares the effectiveness of SDN in handling DoS attacks to traditional networking approaches, highlighting SDN's potential for enhanced security, scalability, and efficiency. Traditional networks are inherently complex and challenging to manage due to the tightly coupled

nature of their data and control planes. Each network device, such as routers and switches, operates autonomously, making centralized control and uniform policy enforcement difficult. The proprietary nature of these devices further complicates integration and scalability.

Software-Defined Networking (SDN) revolutionizes this paradigm by separating the control plane, which makes decisions about where traffic is sent, from the data plane, which forwards traffic to the selected destination. This separation allows for centralized control, dynamic policy implementation, and improved network visibility. SDN's programmability and flexibility make it a strong candidate for addressing the limitations of traditional networks, particularly in mitigating DoS attacks. DoS attacks aim to render a network service unavailable by flooding it with an overwhelming amount of traffic. Traditional networks struggle to respond effectively to these attacks due to their decentralized and rigid structure. In contrast, SDN's centralized control mechanism can quickly adapt to changing traffic patterns and implement countermeasures in real-time.

This study reviews various research papers to compare the efficacy of SDN in mitigating DoS attacks with traditional network approaches. Key metrics such as packet loss, bandwidth utilization, latency, and the ability to handle attack traffic are considered. In traditional networks, DoS mitigation techniques include rate limiting, intrusion detection systems (IDS), and firewalls. However, these methods often fall short due to the lack of centralized control and real-time adaptability. The distributed nature of control makes it difficult to implement a coordinated defense, leading to delayed response times and inefficient resource utilization.Research has shown that SDN can significantly improve the handling of DoS attacks. The centralized controller in SDN provides a global view of the network, enabling more effective monitoring and rapid deployment of defense mechanisms. For instance, reference demonstrated the use of a secure and optimized energy framework of Blockchain-enabled software-defined, ensuring network security and consistency.

Another study highlighted that SDN decreases packet loss and increases bandwidth under attack conditions. Moreover, SDN allows for dynamic flow rule management, which can prioritize legitimate traffic and drop malicious packets efficiently. Reference presented a secure SDN framework capable of avoiding spoofing attacks and DoS with minimal overhead. Similarly, research showed that Flow Keeper maintains more than 80% bandwidth during DoS attacks by filtering out unauthorized topology changes and forged packets.

The comparison reveals that SDN's flexibility and centralized management offer significant advantages over traditional networks in mitigating DoS attacks. Traditional networks' decentralized nature leads to slower response times and less efficient resource allocation. In contrast, SDN can quickly adapt to attack patterns, deploying countermeasures dynamically and efficiently.

The findings indicate that SDN provides a robust framework for mitigating DoS attacks compared to traditional networking. Its centralized control, real-time adaptability, and enhanced visibility into network traffic make it a superior choice for modern network security challenges. Future research should focus on further optimizing SDN-based defense mechanisms and exploring their application in diverse network environments.

3.4.3 The Need for Improved Mitigation Techniques

Given the unique challenges of DoS attacks in SDN, there is a need for improved detection and mitigation techniques specifically designed for SDN environments. These techniques should leverage the programmability and centralized control of SDN to provide more effective and efficient protection against DoS threats.

3.6. METHODOLOGY

3.6.1 Research Design

Qualitative vs. Quantitative Approaches

This study employs a mixed-methods approach, combining both qualitative and quantitative research methods. Qualitative methods are used to explore the underlying mechanisms of DoS attacks and the limitations of existing solutions. Quantitative methods are used to measure the effectiveness of proposed mitigation techniques through simulations and performance metrics.

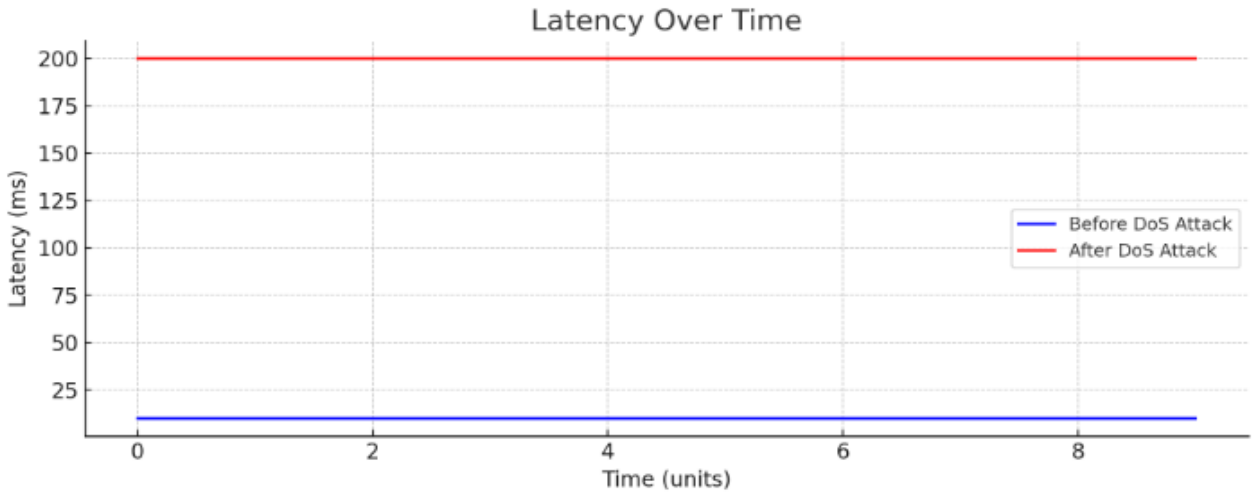
Creating a comparison of the network before and after a DoS attack involves examining various metrics such as latency, throughput, packet loss, CPU utilization, and network availability. Here's a template for a table that We use to compare these metrics:

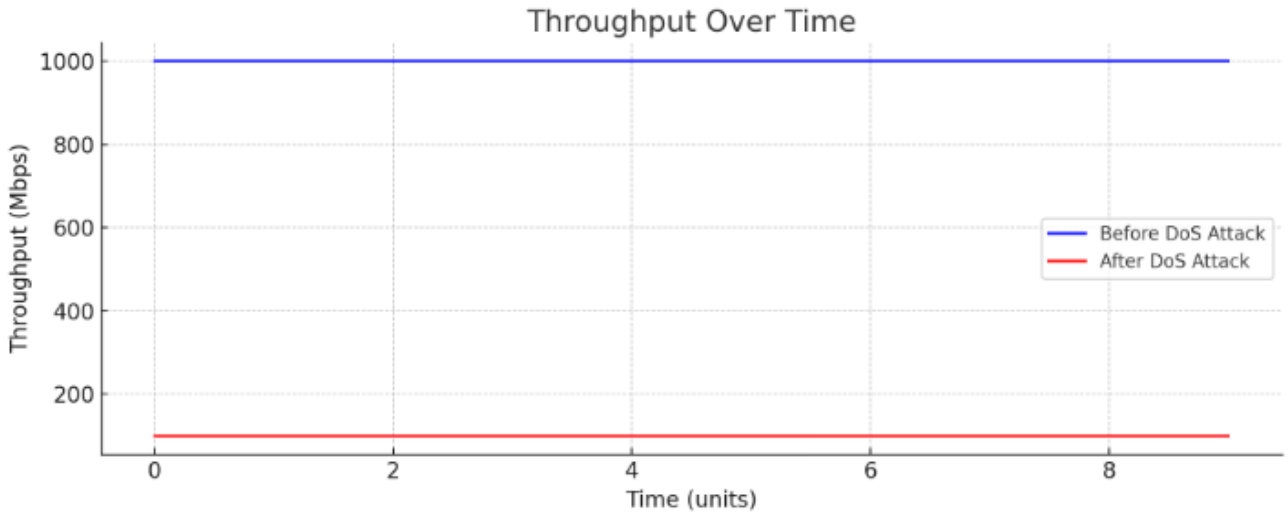
Metric	Before DOS Attack	After DOS Attack
Latency(ms)	10	200
Throughput(Mbps)	1000	100
Packet Loss (%)	0.1	50
CPU Utilization (%)	20	95
Network Avaliblilty(%)	98.4	30.3

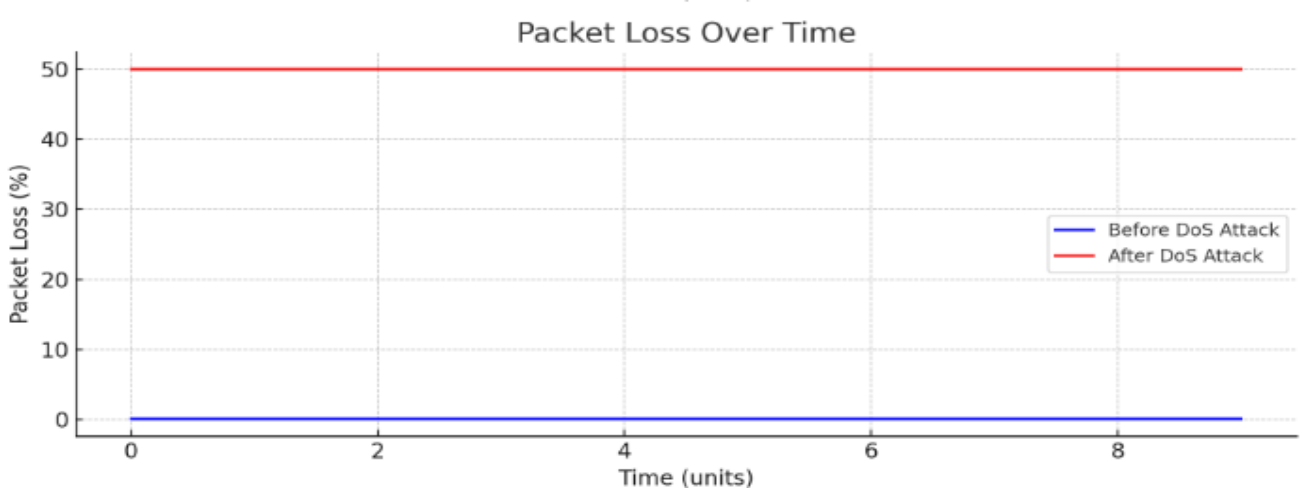
Table 1. Comparison of the network before and after a DoS attack

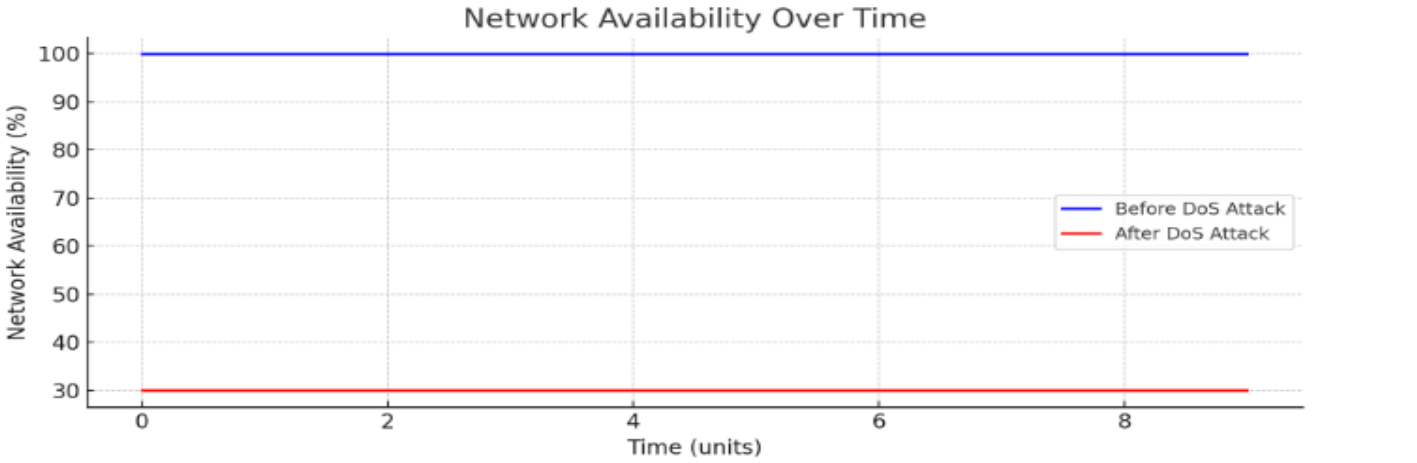
Latency Over Time: A line graph showing latency measurements taken at regular intervals before and after the attack.
Throughput Over Time: A line graph showing throughput measurements before and after the attack.
Packet Loss Over Time: A line graph showing the percentage of packet loss before and after the attack.

CPU Utilization Over Time: A line graph showing the CPU usage of the network devices before and after the attack.
Network Availability Over Time: A line graph showing the network's availability percentage before and after the attack.









Latency Over Time: Latency significantly increases from 10 ms to 200 ms after the attack.

Throughput Over Time: Throughput drops drastically from 1000 Mbps to 100 Mbps after the attack.

Packet Loss Over Time: Packet loss spikes from 0.1% to 50% after the attack.

CPU Utilization Over Time: CPU utilization rises sharply from 20% to 95% after the attack.

Network Availability Over Time: Network availability decreases dramatically from 99.9% to 30% after the attack.

Okay so practically we have followed the specific steps to calculate it:

3.6.2. Latency Over Time

Tools: Ping, Traceroute, SDN controller logs, Network monitoring tools (e.g., Zabbix, Nagios)

Method:

Ping: Measure the round-trip time (RTT) between devices at regular intervals.

ping <target IP> -i <interval> -c <count>

Traceroute: Measure the time it takes for packets to traverse the network.

traceroute <target IP>

SDN Controller Logs: Check logs for flow setup times and other latency metrics.

3.6.3. Throughput Over Time

Tools: Iperf, Network monitoring tools

Method:

Iperf: Generate traffic between hosts and measure the throughput.

iperf -c <server IP> -t <time> -i <interval>

SNMP (Simple Network Management Protocol): Monitor interface statistics to calculate throughput.

Use tools like MRTG (Multi Router Traffic Grapher) or Cacti.

SDN Controller: Some controllers provide throughput metrics.

3.6.4 Packet Loss Over Time

Tools: Ping, Iperf, Network monitoring tools

Method:

Ping: Measure packet loss by sending a series of pings and calculating the percentage of lost packets.

ping <target IP> -c <count>

Iperf: Measure packet loss during throughput tests.

iperf -c <server IP> -u -t <time> -i <interval> (for UDP tests)

SDN Controller Logs: Check for dropped packets and flow entries.

3.6.5 CPU Utilization Over Time

Tools: SNMP, Performance monitoring tools (e.g., top, htop, Nagios, Zabbix)

Method:

SNMP: Monitor CPU utilization of network devices.

Tools like MRTG or Cacti can help visualize this data.

Performance Monitoring Tools: Use top or htop on the devices to log CPU usage.

top -b -d <interval> -n <count> > cpu_usage.log

5. Network Availability Over Time

Tools: Ping, Network monitoring tools

Method:

Ping: Regularly ping critical network devices and services to check their availability.

ping <target IP> -i <interval> -c <count>

Network Monitoring Tools: Set up alerts and logs for device/service downtime.

Tools like Nagios or Zabbix can provide detailed availability reports.

These visualizations and the corresponding table provide a clear comparison of the network's performance before and after a DoS attack.

Choice of Methodology

The chosen methodology involves simulating DoS attacks in an SDN environment, implementing various detection and mitigation techniques, and evaluating their performance. This approach allows for a comprehensive analysis of the effectiveness of different strategies in a controlled setting.

3.6.6 Data Collection

Tools and Techniques :

Data collection involves using various tools and techniques to simulate DDoS attacks and monitor the network's response. Key tools include:

- Mininet: A network emulator that creates a virtual network for testing and experimentation.
- POX Controller: An open-source SDN controller used to manage the virtual network.
- Wireshark: A network protocol analyzer used to capture and analyze network traffic.
- Tcpdump: Provided detailed packet-level monitoring and analysis.
- Flow Rules on OVS Switches: Analyzed to ensure proper forwarding and defense against attacks.

Data Sources:

Data sources include network traffic logs, performance metrics (e.g., throughput, latency), and detection/mitigation effectiveness metrics (e.g., detection accuracy, false positive/negative rates).

3.6.7 Network Setup

Description of the SDN Topology :

The SDN topology used in this study includes:

- 2 POX Controllers: Managing the network and implementing detection/mitigation strategies.
- 2 Core Switches: Connected to the controllers and responsible for forwarding traffic.
- 4 Distribution Switches: Connected to the core switches, distributing traffic to the hosts.
- 16 Hosts: End devices generating legitimate and malicious traffic.

Tools Used :

Mininet: Mininet is an open-source network emulator that creates a virtual SDN environment. It allows for the simulation of complex network topologies on a single machine, making it an invaluable tool for testing SDN configurations and behaviors without the need for physical hardware. In this study, Mininet was used to construct the SDN network, including the virtual switches, hosts, and links. Its ability to emulate a real-world network environment enabled the simulation of DoS/DDoS attacks and the evaluation of various mitigation strategies in a controlled setting.

POX Controllers: POX is a Python-based open-source SDN controller platform. It manages the network by directing traffic and implementing network policies. POX was used to develop and deploy custom detection and mitigation strategies for DoS/DDoS attacks. The flexibility of POX allowed for the easy integration of new algorithms and protocols, making it a crucial component in testing the effectiveness

of different security measures. By using POX, the study could dynamically control the network and respond to attacks in real-time.

Wireshark: Wireshark is a widely-used network protocol analyzer. It captures and provides detailed insights into network traffic, helping to diagnose issues and analyze packet-level data. In this study, Wireshark was used to monitor network traffic during simulations, capturing packets to identify the characteristics of DoS/DDoS attacks. The analysis of captured traffic helped in understanding the impact of attacks on the network and the effectiveness of the mitigation strategies implemented by the POX controllers.

Hping3: Hping3 is a network tool used for generating TCP/IP packets and testing network security. It can create custom packets for various protocols, making it suitable for simulating network-based attacks like DoS/DDoS. In this study, Hping3 was used to generate attack traffic, such as TCP SYN floods, to test the resilience of the SDN against such attacks. The ability to customize packet generation with Hping3 provided a realistic simulation of malicious traffic, crucial for evaluating the network's defensive capabilities.

Nmap: Nmap (Network Mapper) is a powerful network scanning tool used for network discovery and security auditing. It identifies hosts and services on a network, providing insights into potential vulnerabilities. Nmap was used in this study to scan the network and identify potential entry points for DoS/DDoS attacks. By mapping the network, Nmap helped in understanding the topology and identifying critical nodes that required enhanced security measures.

Python Script (Filter): A custom Python script, referred to as Filter, was developed to process and filter network traffic. This script was designed to identify and block malicious traffic based on predefined rules and patterns. The Filter script played a vital role in the real-time detection and mitigation of DoS/DDoS attacks. By analyzing incoming packets and applying filtering logic, the script helped in maintaining the integrity and performance of the network during attack scenarios. The script's ability to be customized and extended ensured that it could adapt to different attack vectors and evolving threats. Each tool used in this study provided unique capabilities that contributed to the overall research objectives. Mininet allowed for the creation of a realistic and scalable SDN environment, essential for testing the network under various conditions.

POX Controllers facilitated the implementation of custom security strategies, enabling dynamic network management. Wireshark provided detailed traffic analysis, crucial for understanding attack patterns and the effectiveness of mitigations. Hping3 and Nmap offered powerful means to simulate and identify vulnerabilities, ensuring that the network was thoroughly tested against potential threats. Finally, the custom Python Script Filter allowed for real-time traffic analysis and filtering, providing an additional layer of security to protect the SDN from malicious activities.

By leveraging these tools, the study was able to conduct comprehensive testing and analysis, leading to a deeper understanding of DoS/DDoS attack mechanisms and the development of effective mitigation strategies

3.6.8 Implementation of DoS Attack Scenarios

Types of Attacks Simulated :

The study simulates various types of DoS attacks, including:

UDP Floods: Volume-based attacks overwhelming the network bandwidth.

SYN Floods: Protocol attacks exploiting TCP connection establishment.

HTTP Floods: Application layer attacks targeting web servers.

4.4.2 Steps for Implementation for TCP SYN Floods

1. Setup the Network: Configure the SDN topology using Mininet.
2. Simulate Normal Traffic: Generate baseline traffic from hosts to establish normal network behavior.
3. Initiate DoS Attack: Use tools like hping3 to generate DoS attack traffic.
4. Monitor Network Behavior: Capture network traffic using Wireshark and log performance metrics.
5. Implement Mitigation Techniques: Deploy detection and mitigation strategies on the SDN controllers.
6. Evaluate Effectiveness: Analyze the impact of mitigation techniques on network performance and DDoS attack mitigation.

The Dos attack according kill chain model

The cyber kill chain (CKC) is a commonly used and accepted concept in the cyber security sector for describing most attacks' processes. Lockheed Martin first launched it based on a military approach, but it has since been extensively modified [28]. The CKC model (Fig. 3) is divided into seven phases, starting with reconnaissance of the attack surface and ending with achieving the planned aims. Each stage in the CKC can be targeted with unique defense methods to lower the chances of a successful infiltration and the impact of attacker damage.
•Reconnaissance: The attackers aim to gather as much information about the attack targets as possible to determine the attack surface and potential breach points.
•Weaponization: In this phase, attackers will exploit the vulnerabilities discovered during the reconnaissance phase and create an attack vector.
•Delivery: Attackers have gained access to the target systems, and they can deliver the payload that will help to carry out the attack.
•Exploitation: The intended payload has been delivered, and the attackers can execute the malicious code within the target system.

•Installation: In this phase, the malware or other attack vector will be installed on the target system and attackers have gained access to the target system.
•Command and Control: The attacker now can control the target systems and network and access privileged accounts, search for credentials, and change permissions.
•Actions on Objectives: Finally, attackers can perform data exfiltration from the system. Their main objective is gathering, encrypting, and extracting confidential information from the compromised system.

Many scholars have conducted various studies to improve the effectiveness of cyber defense using the CKC model. To investigate the Artificial Intelligence (AI) techniques that have been used in an extended version of CKC, Iwona et al.conducted a literature review, and they see great potential in the AI techniques that could be deployed in the reconnaissance, intrusion, privilege escalation, and data exfiltration phases compare with other remaining phases. And Mikhail et al. discussed AI patterns that could be exploited for intrusion early detection in the reconnaissance phase in the smart grid. Tom et al. presented three attack scenarios based on the CKC model to analyze the hidden advanced persistent attacks (APT) in ICS, and the attack indicators and defense mechanisms of each phase are also discussed. Yussuf et al. developed a data-driven framework for APT detection in multiple phases of the CKC model. Qublai et al. analyzed and identified the common features of four popular Dos by performing CKC phases for the early detection of DOS.

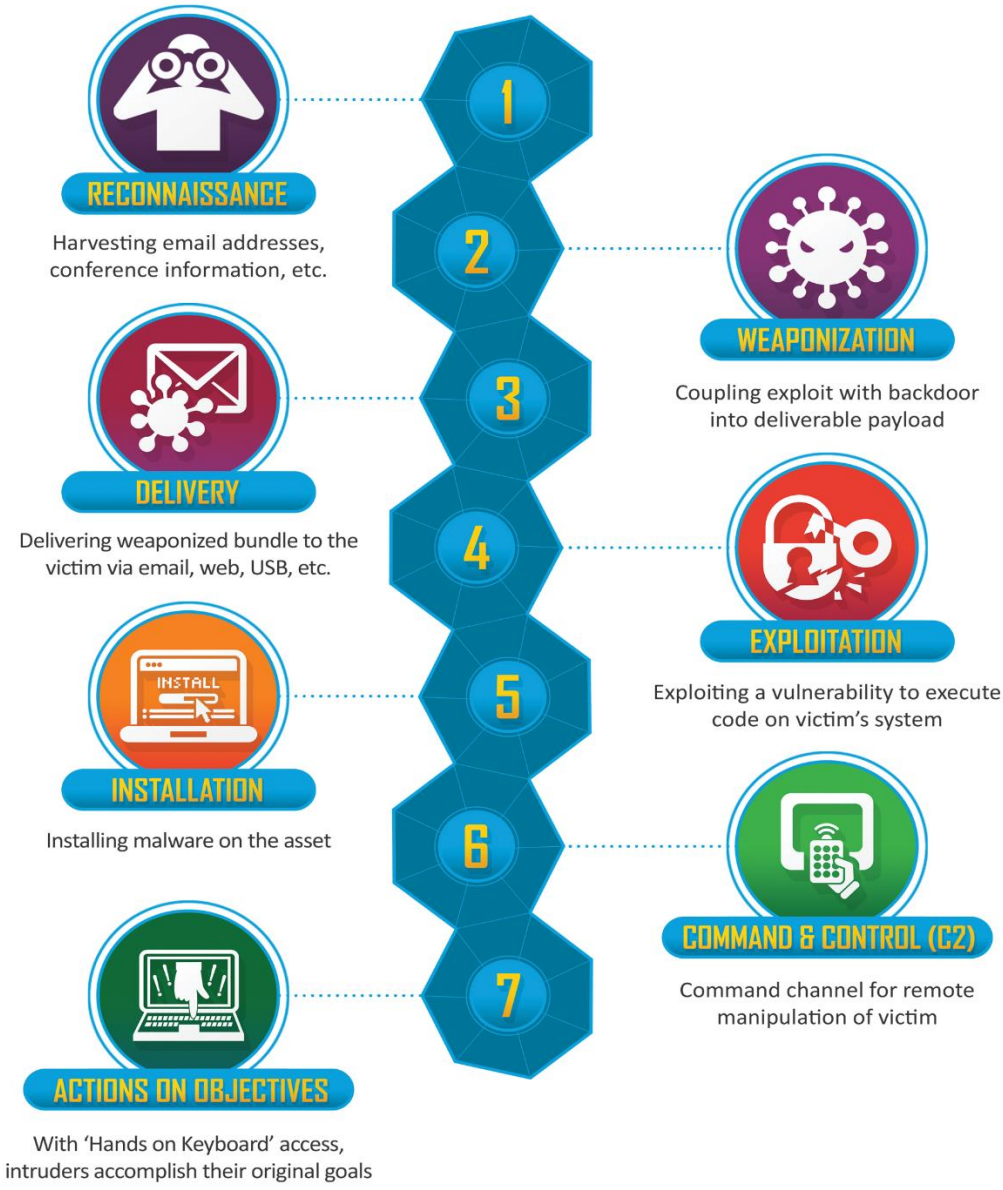


Figure 3.4. Cyber kill chain model

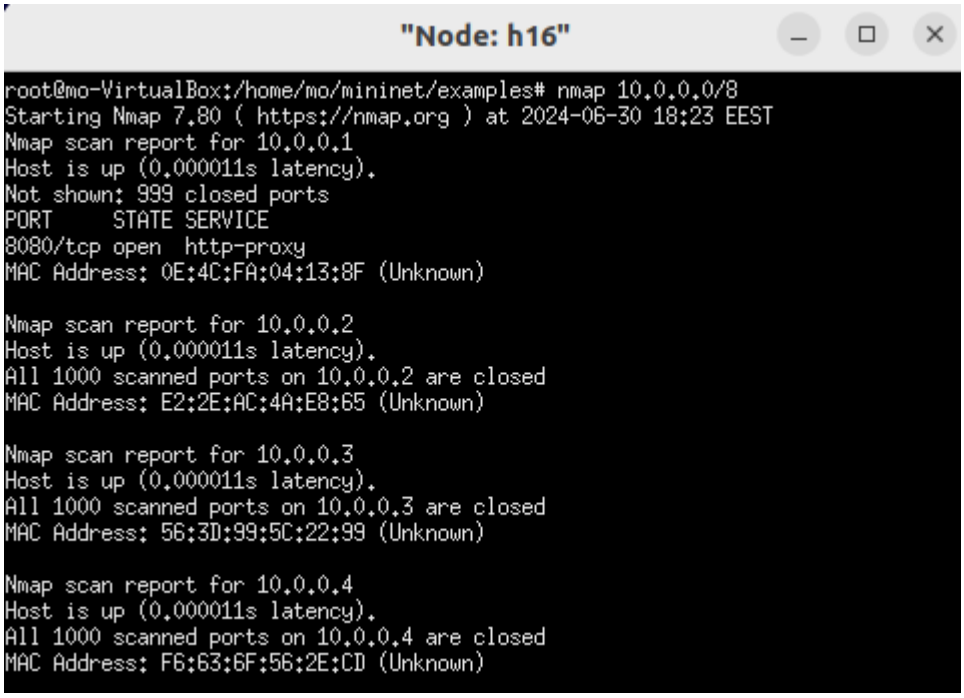
Here's a step-by-step breakdown of how our DoS attack scenario fits into the kill chain model:

1. Reconnaissance

Goal: Gather information about the target.

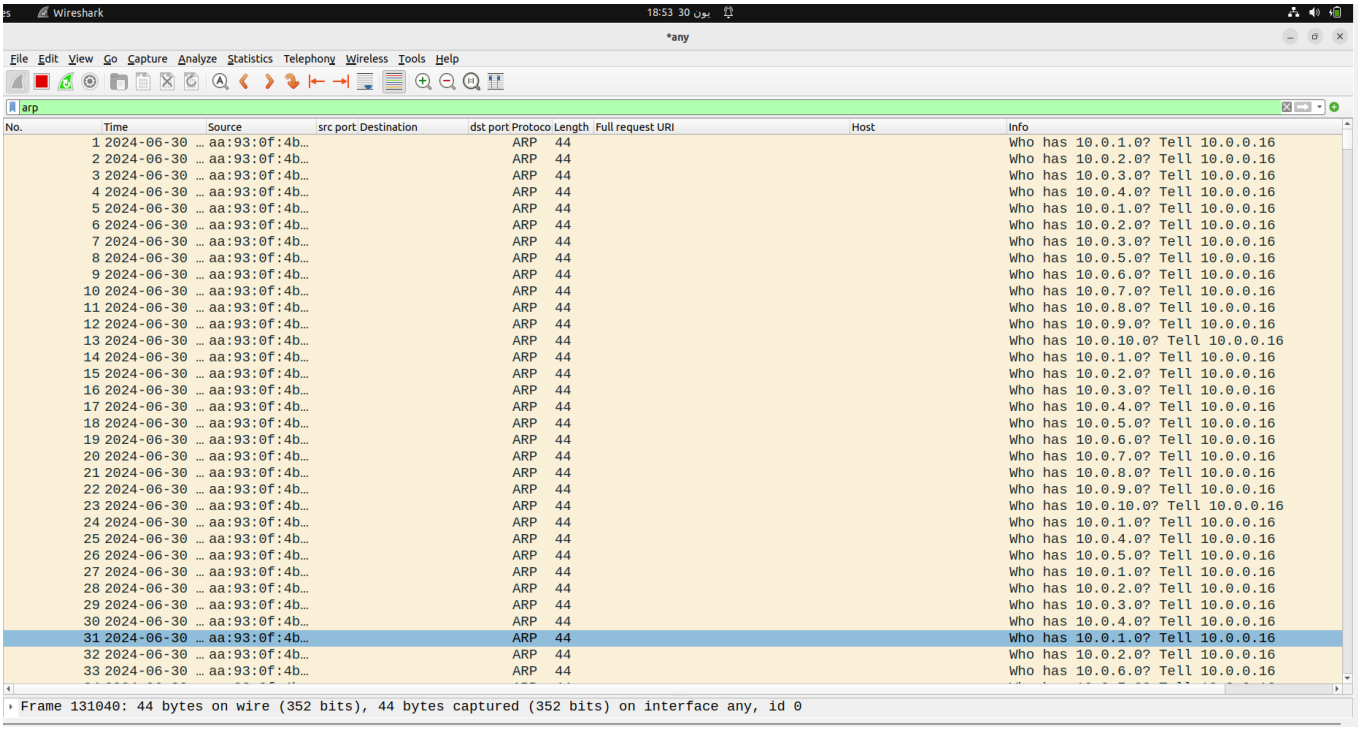
- **Action:** Perform network scanning to identify open ports and services running on h1.
    - **Details:** Using nmap on h1 to discover open port 8080, which indicates the HTTP server is running.

- **How:**
  I. We let the attacker be h16 and he does the nmap on the hole network to discover the server because he does, t know which ip the server is working with, and which port is open to attack on this port with his ip address.



  II. As we see that nmap gives us the ip and the mac address of the http server, also it gives us the open port that the server works on (8080). The scan also gives us the remaining ip and mac addresses and the status of the ports of the devices on the network.

  III. When the scan was running on the network, we captured the traffic on the server (h1) to see what happens as follows:



Understanding ARP Packets

1. **ARP Requests and Replies**:
   - **ARP Request**: Sent by a host to determine the MAC address of another host with a known IP address.
   - **ARP Reply**: Sent in response to an ARP request, providing the MAC address associated with the requested IP address.
2. **Common Occurrences**:
   - ARP packets are common in network environments as they are used for IP-to-MAC address mapping.
   - Before communication can occur, a device needs to know the MAC address of the device it wants to communicate with.

Why You're Seeing ARP Packets

1. **Initial Communication**:
   - When you initiate network scanning with nmap or any other tool, your host (h16) will need to resolve the IP addresses of other hosts (like h1) to their MAC addresses. This triggers ARP requests and replies.
2. **Network Discovery**:

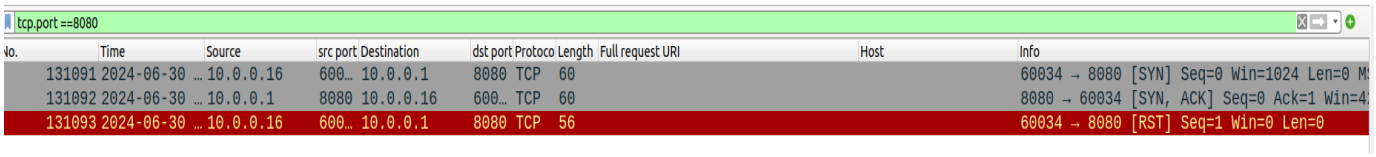
- During reconnaissance, devices frequently communicate to discover each other's presence and details, leading to an increase in ARP traffic.

How to Proceed

To focus on capturing the relevant packets for your reconnaissance phase, you may want to filter out ARP packets and focus on TCP/IP packets. Here's how you can do this:

1. **Wireshark Filters**:
   - Use Wireshark filters to exclude ARP packets and focus on other protocols.
   - Filter to exclude ARP: not arp
   - To see only TCP packets: tcp
2. **Capturing Network Scanning**:
   - Run nmap again and use the appropriate filters in Wireshark to capture TCP/IP packets.
   - Example nmap command: nmap -p 8080 h1
   - Wireshark filter: tcp.port == 8080
3. **Analyzing Captured Data**:
   - Look for SYN packets (part of the TCP handshake) indicating attempts to connect to port 8080.
   - Identify any responses from h1, confirming that port 8080 is open.

IV. Here is the filtered traffic on port 8080



Understanding the Sequence

Here's what happens during this sequence in your context:

1. **SYN Packet from h16 to h1**:
   - h16 initiates a connection by sending a SYN packet to port 8080 on h1.
2. **SYN-ACK Packet from h1 to h16**:
   - h1 responds to the SYN packet with a SYN-ACK packet, indicating that it is ready to establish a connection on port 8080.
3. **RST Packet from h16 to h1**:

- Instead of completing the handshake with an ACK packet, h16 sends an RST packet. This aborts the connection setup process.

V. This detailed sequence helps to demonstrate the reconnaissance phase in our kill chain model, showing how an attacker might identify open ports while minimizing their footprint on the network.

VI. Now the attacker (h16) knows the ip and the open port of the server so he now can apply the next step of the kill chain model which is Weaponization.

2. Weaponization

Goal: Create a deliverable payload for the attack.

- **Action:** Prepare the DoS attack payload using hping3.
- **Command:**

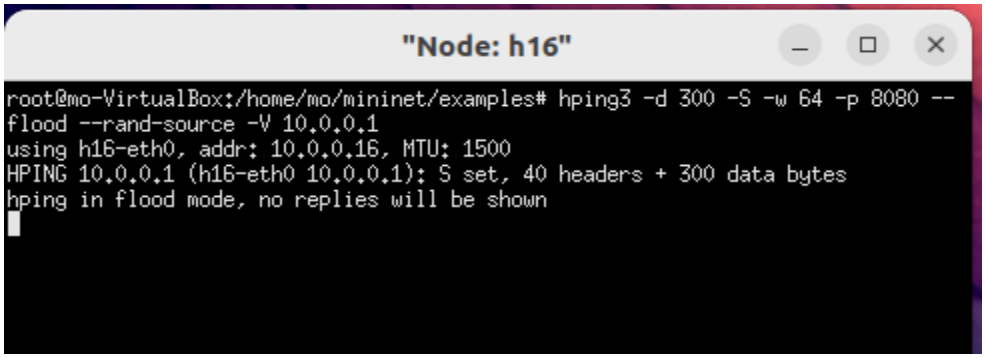
$hping3 -d 300 -S -w 64 -p 8080 --flood --rand-source -V 10.0.0.1

- **Explanation:**

- Payload: The SYN flood attack is crafted to overwhelm the target (h1) by sending a high volume of SYN packets to port 8080.
- Flood Mode: Ensures packets are sent as quickly as possible to maximize the impact.
- Random Source IPs: Helps evade detection and filtering by the target, making it difficult to mitigate the attack.
- Data Size: Each packet includes 300 bytes of data, adding to the load on the target.
- Verbose Output: Provides detailed feedback during the attack, useful for monitoring and analysis.

- **How:**

I. h16 opens the configuration window (prompt) using xterm tool to add the command of the attack as follows:



Here's an explanation of each component of this command:

1. **hping3**:
   - This is the tool being used. hping3 is a network packet crafting tool, useful for security testing and attacks such as DoS.

2. **-d 300**:
    - **Data size**: Sets the size of the data field in the packet to 300 bytes.
3. **-S**:
    - **SYN flag**: Sets the SYN flag in TCP packets. This is used to initiate a TCP connection.
4. **-w 64**:
    - **Window size**: Sets the TCP window size to 64. This parameter controls the amount of data that can be sent before an acknowledgment is required.
5. **-p 8080**:
    - **Port**: Specifies the destination port number, which in this case is 8080 (where the HTTP server is running).
6. **--flood**:
    - **Flood mode**: Sends packets as fast as possible, generating a high volume of traffic to overwhelm the target.
7. **--rand-source**:
    - **Random source**: Randomizes the source IP addresses of the packets. This helps in avoiding detection and makes it difficult for the target to filter out the attack packets.
8. **-V**:
    - **Verbose mode**: Enables verbose output, providing detailed information about what hping3 is doing.
9. **10.0.0.1**:
    - **Target IP**: The IP address of the target host, which in this case is 10.0.0.1 (h1).

II. After running the command of the attack on h16 we goes throw the next step of the kill chain model which is Delivery.

3. Delivery

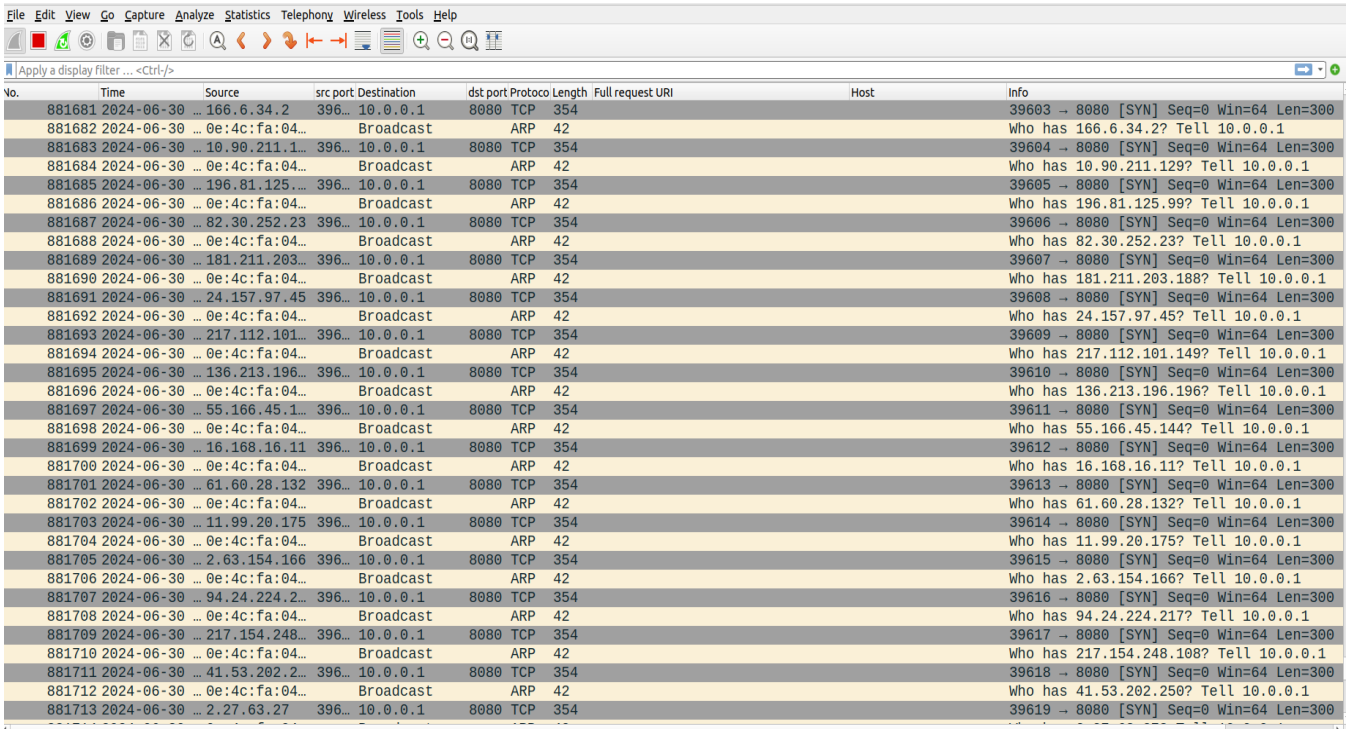
Goal: Deliver the weapon to the target environment.

- **Action:** Initiate the DoS attack from h16.
    - **Details:** Use hping3 to send a high volume of TCP packets to port 8080 on h1.
- **How:**

I. When running the command of the attack on h16 ,the server flooded with tcp packets from random ip addresses.

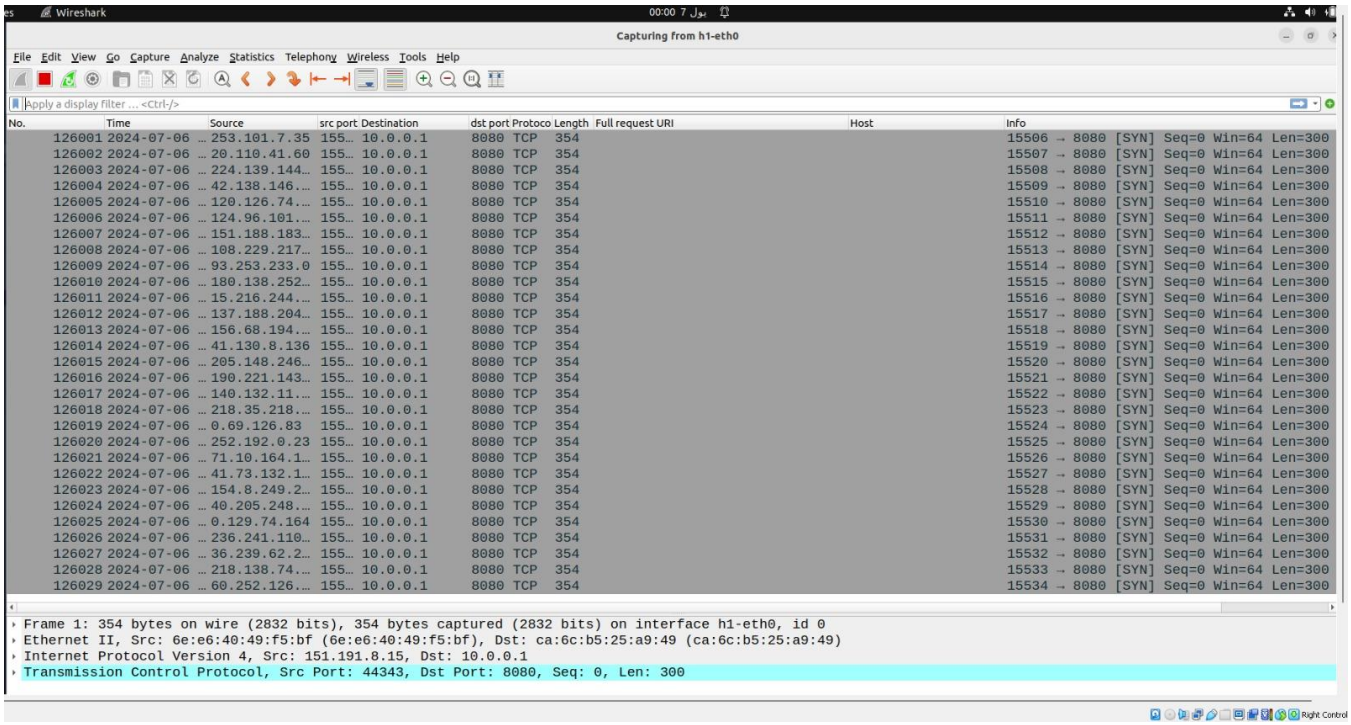
II. Here are several ways for detection when deliver the attack.

a. Using Wireshark



No.	Time	Source	src port	Destination	dst port	Protocol	Length	Full request URI	Host	Info
881681	2024-06-30 …	166.6.34.2	396…	10.0.0.1	8080	TCP	354			39603 → 8080 [SYN] Seq=0 Win=64 Len=300
881682	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 166.6.34.2? Tell 10.0.0.1
881683	2024-06-30 …	10.90.211.1…	396…	10.0.0.1	8080	TCP	354			39604 → 8080 [SYN] Seq=0 Win=64 Len=300
881684	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 10.90.211.129? Tell 10.0.0.1
881685	2024-06-30 …	196.81.125.…	396…	10.0.0.1	8080	TCP	354			39605 → 8080 [SYN] Seq=0 Win=64 Len=300
881686	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 196.81.125.99? Tell 10.0.0.1
881687	2024-06-30 …	82.30.252.23	396…	10.0.0.1	8080	TCP	354			39606 → 8080 [SYN] Seq=0 Win=64 Len=300
881688	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 82.30.252.23? Tell 10.0.0.1
881689	2024-06-30 …	181.211.203…	396…	10.0.0.1	8080	TCP	354			39607 → 8080 [SYN] Seq=0 Win=64 Len=300
881690	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 181.211.203.188? Tell 10.0.0.1
881691	2024-06-30 …	24.157.97.45	396…	10.0.0.1	8080	TCP	354			39608 → 8080 [SYN] Seq=0 Win=64 Len=300
881692	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 24.157.97.45? Tell 10.0.0.1
881693	2024-06-30 …	217.112.101…	396…	10.0.0.1	8080	TCP	354			39609 → 8080 [SYN] Seq=0 Win=64 Len=300
881694	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 217.112.101.149? Tell 10.0.0.1
881695	2024-06-30 …	136.213.196…	396…	10.0.0.1	8080	TCP	354			39610 → 8080 [SYN] Seq=0 Win=64 Len=300
881696	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 136.213.196.196? Tell 10.0.0.1
881697	2024-06-30 …	55.166.45.1…	396…	10.0.0.1	8080	TCP	354			39611 → 8080 [SYN] Seq=0 Win=64 Len=300
881698	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 55.166.45.144? Tell 10.0.0.1
881699	2024-06-30 …	16.168.16.11	396…	10.0.0.1	8080	TCP	354			39612 → 8080 [SYN] Seq=0 Win=64 Len=300
881700	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 16.168.16.11? Tell 10.0.0.1
881701	2024-06-30 …	61.60.28.132	396…	10.0.0.1	8080	TCP	354			39613 → 8080 [SYN] Seq=0 Win=64 Len=300
881702	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 61.60.28.132? Tell 10.0.0.1
881703	2024-06-30 …	11.99.20.175	396…	10.0.0.1	8080	TCP	354			39614 → 8080 [SYN] Seq=0 Win=64 Len=300
881704	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 11.99.20.175? Tell 10.0.0.1
881705	2024-06-30 …	2.63.154.166	396…	10.0.0.1	8080	TCP	354			39615 → 8080 [SYN] Seq=0 Win=64 Len=300
881706	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 2.63.154.166? Tell 10.0.0.1
881707	2024-06-30 …	94.24.224.2…	396…	10.0.0.1	8080	TCP	354			39616 → 8080 [SYN] Seq=0 Win=64 Len=300
881708	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 94.24.224.217? Tell 10.0.0.1
881709	2024-06-30 …	217.154.248…	396…	10.0.0.1	8080	TCP	354			39617 → 8080 [SYN] Seq=0 Win=64 Len=300
881710	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 217.154.248.108? Tell 10.0.0.1
881711	2024-06-30 …	41.53.202.2…	396…	10.0.0.1	8080	TCP	354			39618 → 8080 [SYN] Seq=0 Win=64 Len=300
881712	2024-06-30 …	0e:4c:fa:04…		Broadcast		ARP	42			Who has 41.53.202.250? Tell 10.0.0.1
881713	2024-06-30 …	2.27.63.27	396…	10.0.0.1	8080	TCP	354			39619 → 8080 [SYN] Seq=0 Win=64 Len=300

The arp packets followed by the syn packets



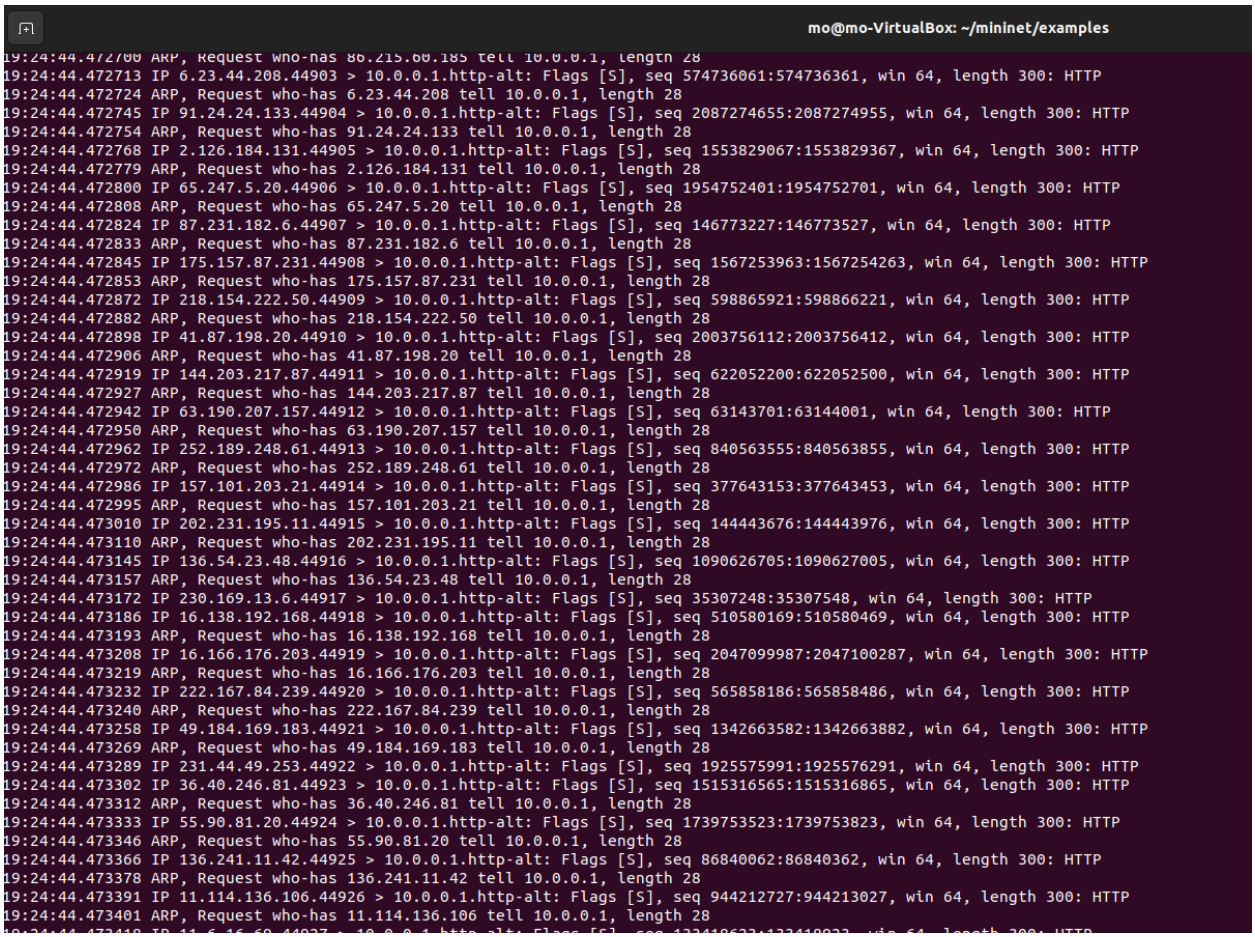
Capturing from h1-eth0

No.	Time	Source	src port	Destination	dst port	Protocol	Length	Full request URI	Host	Info
126001	2024-07-06 …	253.101.7.35	155…	10.0.0.1	8080	TCP	354			15506 → 8080 [SYN] Seq=0 Win=64 Len=300
126002	2024-07-06 …	20.110.41.60	155…	10.0.0.1	8080	TCP	354			15507 → 8080 [SYN] Seq=0 Win=64 Len=300
126003	2024-07-06 …	224.139.144…	155…	10.0.0.1	8080	TCP	354			15508 → 8080 [SYN] Seq=0 Win=64 Len=300
126004	2024-07-06 …	42.138.146.…	155…	10.0.0.1	8080	TCP	354			15509 → 8080 [SYN] Seq=0 Win=64 Len=300
126005	2024-07-06 …	120.126.74.…	155…	10.0.0.1	8080	TCP	354			15510 → 8080 [SYN] Seq=0 Win=64 Len=300
126006	2024-07-06 …	124.96.101.…	155…	10.0.0.1	8080	TCP	354			15511 → 8080 [SYN] Seq=0 Win=64 Len=300
126007	2024-07-06 …	151.188.183…	155…	10.0.0.1	8080	TCP	354			15512 → 8080 [SYN] Seq=0 Win=64 Len=300
126008	2024-07-06 …	108.229.217…	155…	10.0.0.1	8080	TCP	354			15513 → 8080 [SYN] Seq=0 Win=64 Len=300
126009	2024-07-06 …	93.253.233.0	155…	10.0.0.1	8080	TCP	354			15514 → 8080 [SYN] Seq=0 Win=64 Len=300
126010	2024-07-06 …	180.138.252…	155…	10.0.0.1	8080	TCP	354			15515 → 8080 [SYN] Seq=0 Win=64 Len=300
126011	2024-07-06 …	15.216.244.…	155…	10.0.0.1	8080	TCP	354			15516 → 8080 [SYN] Seq=0 Win=64 Len=300
126012	2024-07-06 …	137.188.204…	155…	10.0.0.1	8080	TCP	354			15517 → 8080 [SYN] Seq=0 Win=64 Len=300
126013	2024-07-06 …	156.68.194.…	155…	10.0.0.1	8080	TCP	354			15518 → 8080 [SYN] Seq=0 Win=64 Len=300
126014	2024-07-06 …	41.130.8.136	155…	10.0.0.1	8080	TCP	354			15519 → 8080 [SYN] Seq=0 Win=64 Len=300
126015	2024-07-06 …	205.148.246…	155…	10.0.0.1	8080	TCP	354			15520 → 8080 [SYN] Seq=0 Win=64 Len=300
126016	2024-07-06 …	190.221.143…	155…	10.0.0.1	8080	TCP	354			15521 → 8080 [SYN] Seq=0 Win=64 Len=300
126017	2024-07-06 …	140.132.11.…	155…	10.0.0.1	8080	TCP	354			15522 → 8080 [SYN] Seq=0 Win=64 Len=300
126018	2024-07-06 …	218.35.218.…	155…	10.0.0.1	8080	TCP	354			15523 → 8080 [SYN] Seq=0 Win=64 Len=300
126019	2024-07-06 …	0.69.126.83	155…	10.0.0.1	8080	TCP	354			15524 → 8080 [SYN] Seq=0 Win=64 Len=300
126020	2024-07-06 …	252.192.0.23	155…	10.0.0.1	8080	TCP	354			15525 → 8080 [SYN] Seq=0 Win=64 Len=300
126021	2024-07-06 …	71.10.164.1…	155…	10.0.0.1	8080	TCP	354			15526 → 8080 [SYN] Seq=0 Win=64 Len=300
126022	2024-07-06 …	41.73.132.1…	155…	10.0.0.1	8080	TCP	354			15527 → 8080 [SYN] Seq=0 Win=64 Len=300
126023	2024-07-06 …	154.8.249.2…	155…	10.0.0.1	8080	TCP	354			15528 → 8080 [SYN] Seq=0 Win=64 Len=300
126024	2024-07-06 …	40.205.248.…	155…	10.0.0.1	8080	TCP	354			15529 → 8080 [SYN] Seq=0 Win=64 Len=300
126025	2024-07-06 …	0.129.74.164	155…	10.0.0.1	8080	TCP	354			15530 → 8080 [SYN] Seq=0 Win=64 Len=300
126026	2024-07-06 …	236.241.110…	155…	10.0.0.1	8080	TCP	354			15531 → 8080 [SYN] Seq=0 Win=64 Len=300
126027	2024-07-06 …	36.239.62.2…	155…	10.0.0.1	8080	TCP	354			15532 → 8080 [SYN] Seq=0 Win=64 Len=300
126028	2024-07-06 …	218.138.74.…	155…	10.0.0.1	8080	TCP	354			15533 → 8080 [SYN] Seq=0 Win=64 Len=300
126029	2024-07-06 …	60.252.126.…	155…	10.0.0.1	8080	TCP	354			15534 → 8080 [SYN] Seq=0 Win=64 Len=300

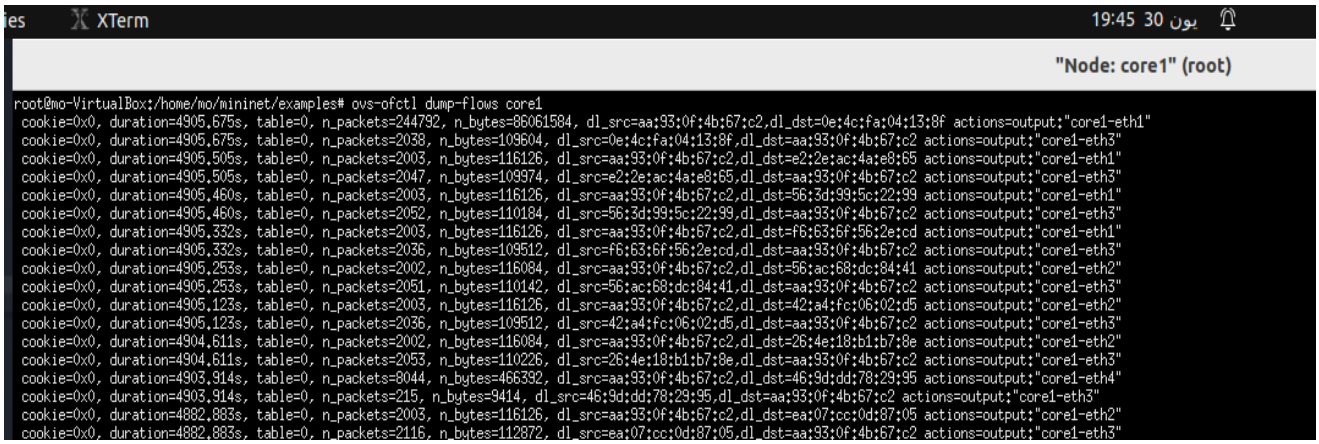
```
Frame 1: 354 bytes on wire (2832 bits), 354 bytes captured (2832 bits) on interface h1-eth0, id 0
Ethernet II, Src: 6e:e6:40:49:f5:bf (6e:e6:40:49:f5:bf), Dst: ca:6c:b5:25:a9:49 (ca:6c:b5:25:a9:49)
Internet Protocol Version 4, Src: 151.191.8.15, Dst: 10.0.0.1
Transmission Control Protocol, Src Port: 44343, Dst Port: 8080, Seq: 0, Len: 300
```

The syn flood of tcp packets

b. Using tcpDump



c. using flow tables on core1 or core2 switch



Now after delivery processes we go to the exploitation process to Overwhelming h1's resources.

4. Exploitation

Goal: Trigger the payload on the target system.

- **Action:** Exploit the target by overwhelming its resources.
    - **Details:** The high volume of TCP packets causes h1 to become unresponsive.
- **How:**

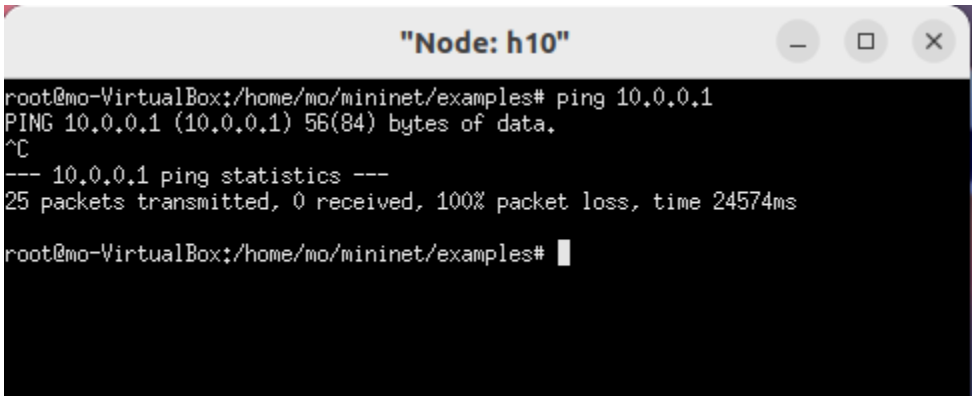
Exploitation Phase:

I. **Action:** Initiate the SYN flood attack from h16 to h1 using the hping3 command.
   a. hping3 -d 300 -S -w 64 -p 8080 --flood --rand-source -V 10.0.0.1

II. **Mechanism:** Flood h1 with SYN packets to create numerous half-open connections, exhausting its resources.

III. Impact:
   a. **Resource Exhaustion:** CPU, memory, and network bandwidth on h1 are consumed by the attack, causing a denial of service.
   b. **Service Unavailability:** The HTTP server on h1 becomes unresponsive to legitimate requests.

IV. Verification:
   a. Wireshark: Capture and analyze the flood of SYN packets and the lack of responses from h1as before.
   b. Ping Test: Observe timeouts when pinging h1 from another host (h10), confirming the DoS impact.



All packets are lost this mean that the impact of the attack is obvious where h10 does not receive any replay from h1 because of the attack on it.

This detailed breakdown of the exploitation phase helps to clearly illustrate how the SYN flood attack disrupts the target's service, fitting into the overall kill chain model.

5. Installation

Goal: Install a backdoor or other persistent access mechanism.

- **Note:** For a DoS attack, this stage may not apply since the goal is disruption rather than persistent access.

6. Command and Control (C2)

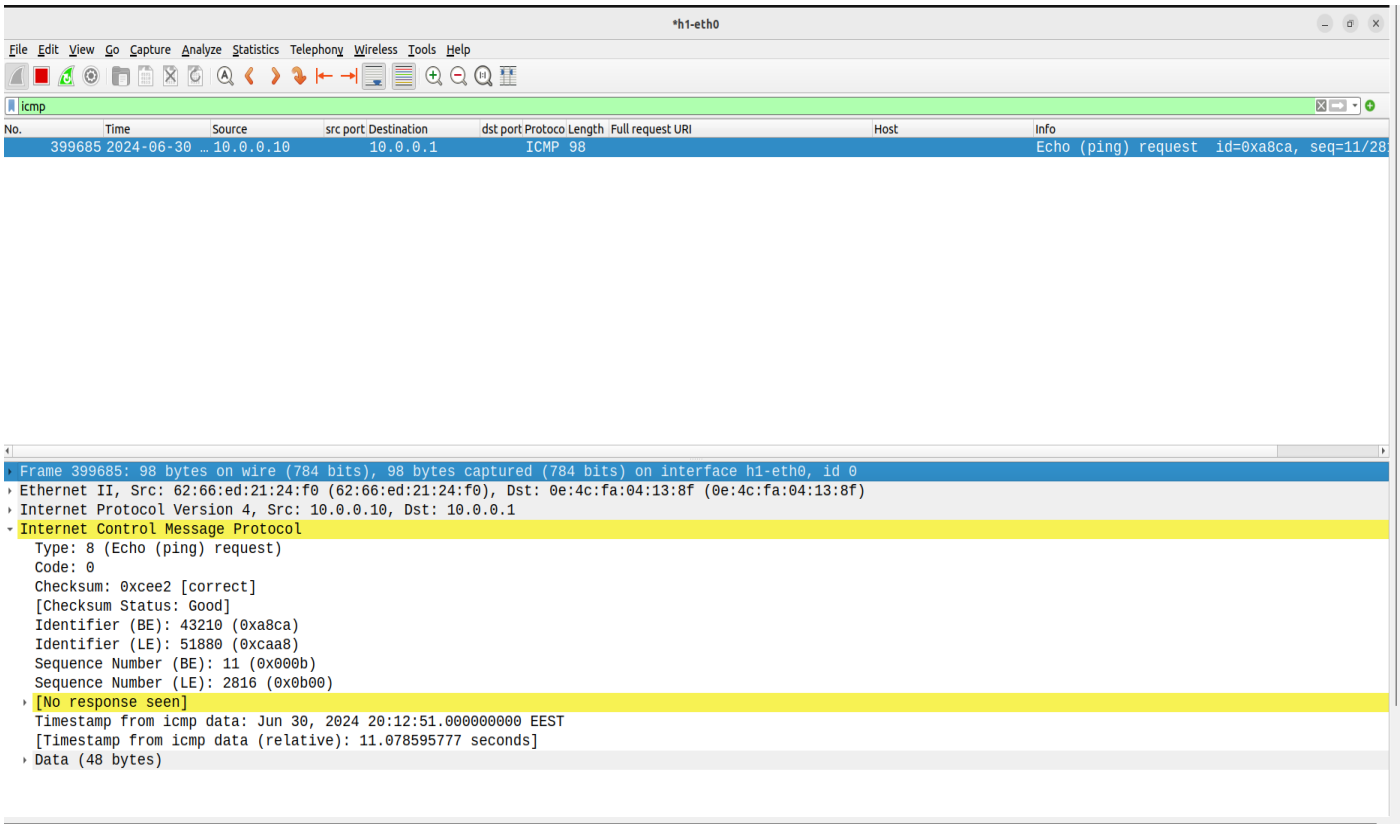
Goal: Establish command and control channels to communicate with the compromised system.

- **Note:** For a DoS attack, this stage may not apply as there is no need for a C2 channel in a simple DoS scenario.

7. Actions on Objectives

Goal: Achieve the attacker's goals.

- **Action:** Disrupt service on h1(http server).
    - **Details:** Monitor the effectiveness of the DoS attack by pinging h1 from h10 to verify that packets are dropped as we see before and here is the detection using Wireshark when pinging on h1 by h10



Visualization in Kill Chain Model

Here is a summary visualization of your attack scenario within the Cyber Kill Chain model:

1. **Reconnaissance**: Network scanning using nmap on h1.
2. **Weaponization**: Preparing the hping3 script.
3. **Delivery**: Sending TCP packets from h16 to h1.
4. **Exploitation**: Overwhelming h1's resources.
5. **Installation**: Not applicable.
6. **C2**: Not applicable.
7. **Actions on Objectives**: Disrupting h1's service, verified by packet loss on h10.

Some notes when applying the Kill Chain Model in the SDN network using Dos:

A. Limited Quota of Open vSwitches During DoS Attack

During the implementation of the DoS attack using hping3, it is important to consider the limitations and behavior of the Open vSwitches (OVS) in your network. Here are some critical points to note:

1. **Limited Resource Quota:**
    - Open vSwitches have a finite amount of resources (CPU, memory, and connection handling capacity).
    - A high volume of SYN packets can quickly exhaust these resources, leading to performance degradation or failure.
2. **Impact on Switches:**
    - Overloading: The flood of SYN packets can overload the connection tables and other critical resources of the switches.

```
mo@mo-VirtualBox:~/pox$ ./pox.py openflow.of_01 --port=6111 forwarding.l2_pairs
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
INFO:forwarding.l2_pairs:Pair-Learning switch running.
INFO:core:POX 0.3.0 (dart) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
INFO:openflow.of_01:[00-00-00-00-00-02 2] connected
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
INFO:openflow.of_01:[00-00-00-00-00-03 6] connected
INFO:openflow.of_01:[00-00-00-00-00-04 7] connected
INFO:openflow.of_01:[00-00-00-00-00-01 4] disconnected
INFO:openflow.of_01:[00-00-00-00-00-01 8] connected
INFO:openflow.of_01:[00-00-00-00-00-01 9] closed
INFO:openflow.of_01:[00-00-00-00-00-01 8] closed
INFO:openflow.of_01:[00-00-00-00-00-01 4] closed
INFO:openflow.of_01:[00-00-00-00-00-01 10] closed
INFO:openflow.of_01:[00-00-00-00-00-01 11] connected
INFO:openflow.of_01:[00-00-00-00-00-01 16] connected
INFO:openflow.of_01:[00-00-00-00-00-01 11] disconnected
INFO:openflow.of_01:[00-00-00-00-00-01 11] closed
INFO:openflow.of_01:[00-00-00-00-00-01 17] connected
INFO:openflow.of_01:[00-00-00-00-00-01 18] connected
INFO:openflow.of_01:[00-00-00-00-00-01 17] disconnected
INFO:openflow.of_01:[00-00-00-00-00-01 17] closed
```

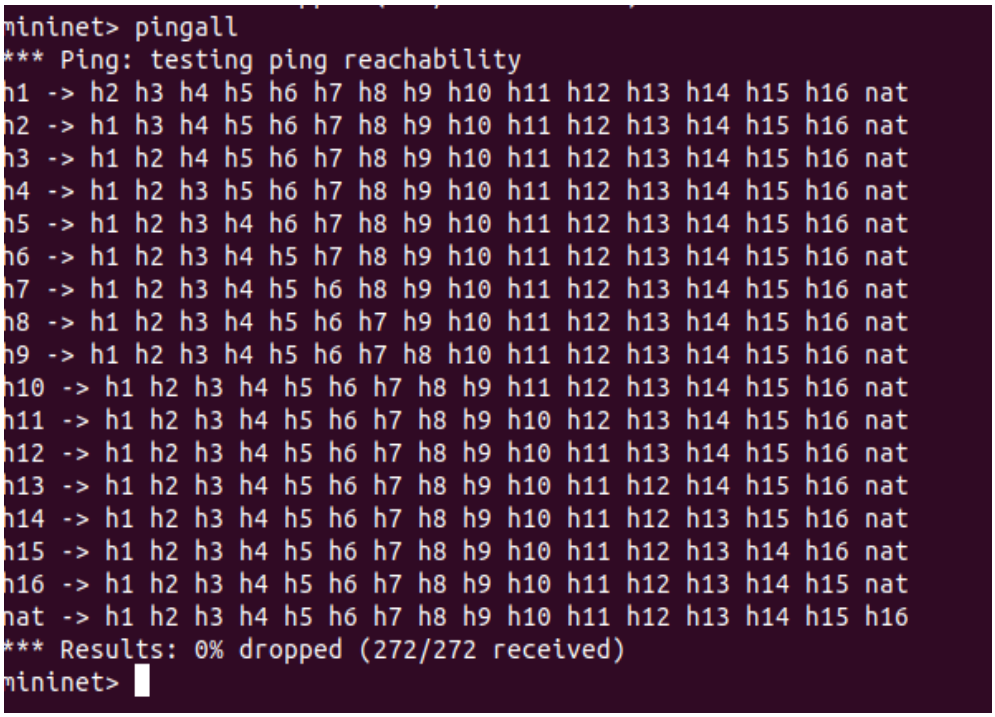
- Disconnection: Overloaded switches may go down or become temporarily disconnected from the network.

3. **Recovery post-attack:**

- Automatic Recovery: Once the DoS attack stops, the switches typically recover automatically.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X X X h10 X X X X X X X
h2 -> h1 h3 h4 X X X X X X X X X X X nat
h3 -> h1 h2 h4 X X X X X X X X X X X nat
h4 -> h1 h2 h3 X h6 h7 h8 X X X X X X X nat
h5 -> h1 h2 h3 h4 h6 h7 h8 X X X X X X X h16 nat
h6 -> h1 h2 h3 h4 h5 h7 h8 X h10 h11 h12 h13 h14 h15 h16 nat
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12 h13 h14 h15 h16 nat
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13 h14 h15 h16 nat
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13 h14 h15 h16 nat
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12 h13 h14 h15 h16 nat
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13 h14 h15 h16 nat
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 h15 h16 nat
h13 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h14 h15 h16 nat
h14 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h15 h16 nat
h15 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h16 nat
h16 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 nat
nat -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Results: 20% dropped (216/272 received)
mininet>
```

- Resource Reallocation: The cessation of flood allows the switches to reallocate resources and resume normal operations.



4. **Observations During Testing:**
    - Switch Down Events: During the attack, you may observe that several switches become unresponsive or disconnected as shown before.
    - Restoration: These switches should come back online and function correctly once the attack is mitigated.

Scenario

1. **During Attack:**
    - Multiple Open vSwitches go down due to resource exhaustion.
    - Network segments connected through these switches become temporarily disconnected.
2. **Post-Attack:**
    - The switches recover and re-establish connections.
    - Network functionality is restored without manual intervention.

Conclusion :

When documenting our Dos attack scenario, include a section on the behavior of OpenvSwitches under stress. Highlight the temporary nature of the disruption and the automatic recovery process. This provides a comprehensive understanding of the impact and resilience of our network infrastructure during and after a DoS attack.

Chapter 4: Mitigation Techniques

4.1 Mitigation Techniques

-Rate Limiting

Rate limiting controls the amount of traffic allowed to enter the network, preventing overload. This technique is simple but can impact legitimate traffic.

- Advantages: Simple implementation, effective for volume-based attacks.
- Disadvantages: May affect legitimate traffic, not suitable for all types of attacks.

Identify Ingress Points in the network, typically the interfaces connected to external networks or segments where DoS attacks are likely to originate. Configure Rate Limiting Policies Access your SDN controller interface or network device management software. Configure rate limiting policies using Open Flow rules or network device APIs.Define rate limiting parameters such as maximum packets per second (pps) or bandwidth limits for each ingress interface. Set Appropriate Rate Limits: Determine appropriate rate limits based on network capacity and traffic patterns. Set rate limits to ensure adequate protection against DoS attacks while allowing legitimate traffic to pass through. Monitor Network Traffic Utilize monitoring tools or features provided by your SDN controller to continuously monitor ingress traffic. Monitor for abnormal traffic patterns or signs of DoS attacks, such as sudden spikes in traffic volume or unusual packet characteristics.

Throttle Incoming Traffic: When abnormal traffic patterns are detected, dynamically update rate limiting rules to throttle incoming traffic. Use techniques such as Open Flow flow modification messages to adjust rate limiting parameters in real-time. Adjust Rate Limiting Parameters: Continuously analyze network conditions and attack characteristics. Adjust rate limiting parameters dynamically to optimize performance and effectively mitigate DoS attacks while minimizing impact on legitimate traffic Determine Sampling Rate: Determine the desired sampling rate based on network size, traffic volume, and analysis requirements. Select an appropriate sampling method, such as random sampling or deterministic sampling. Configure Flow Sampling Rules: Access your SDN controller or network device management interface.

Configure flow sampling rules using Open Flow or device-specific APIs.Define criteria for selecting flows to sample, such as source or destination IP addresses, protocols, or port numbers. Deploy Sampling Probes or Collectors: Deploy flow sampling probes or collectors at strategic points in your network. Ensure proper distribution and scalability of sampling infrastructure to handle traffic volume. Analyze Sampled Flows Capture sampled flows using flow sampling probes or collectors. Analyze sampled flows using flow analysis tools or algorithms to detect anomalous behavior or signs of DoS attacks. Implement policies to act based on analysis results. Examples include dropping or rate-limiting suspicious traffic, alerting network administrators, or dynamically adjusting network configurations. Fine-Tune Configuration Regularly review and fine-tune flow sampling configuration parameters. Adjust sampling rate, flow selection criteria, and analysis algorithms based on changing network conditions and attack patterns.

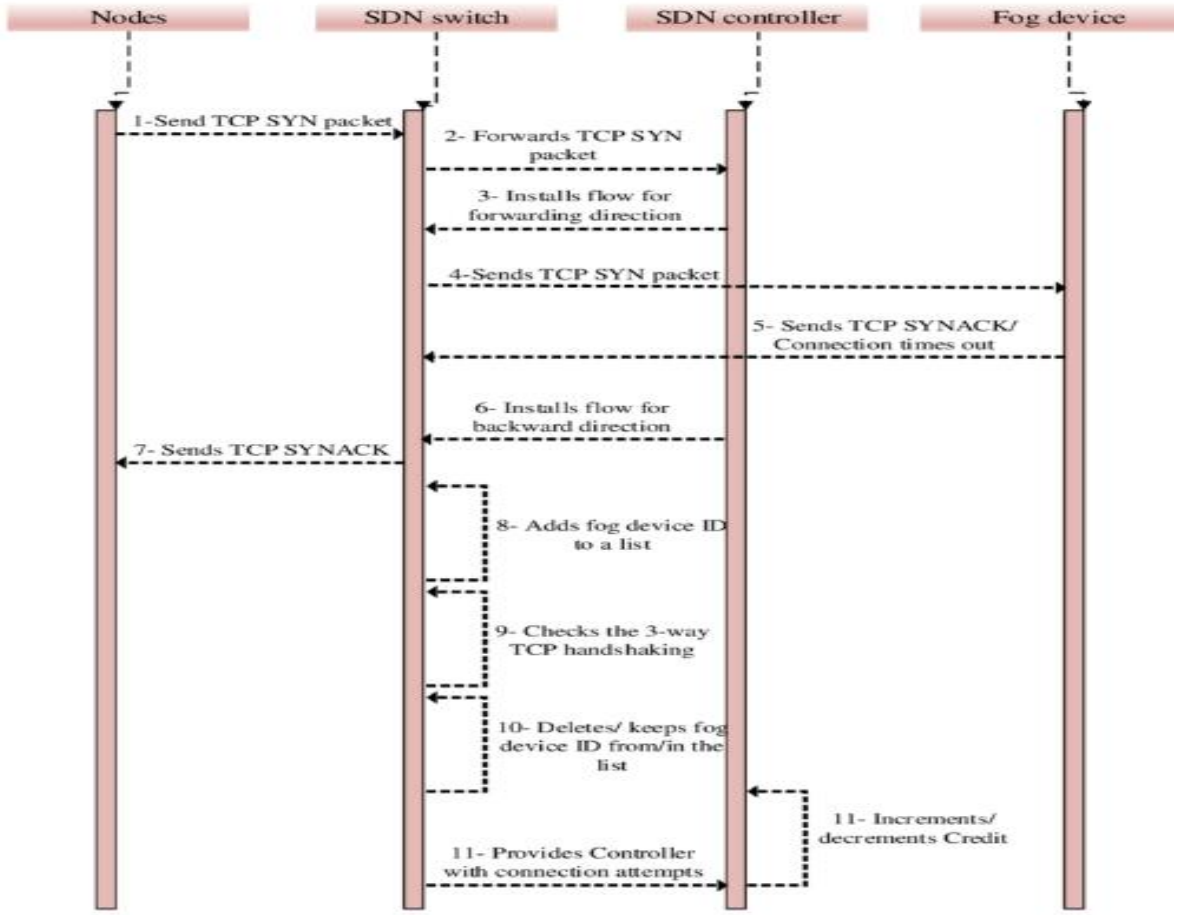


Figure 4.1. Representing Rate limiting strategy

- Blacklisting

Blacklisting involves blocking traffic from known malicious IP addresses. This technique is effective for repeated attacks from the same sources but can be circumvented by attackers using new IPs.

- Advantages: Effective for known malicious sources, easy to implement.
- Disadvantages: Requires up-to-date blacklist, attackers can use different IP addresses.

-Load Balancing

Load balancing distributes traffic across multiple network devices, preventing any single device from becoming overwhelmed. This technique improves network resilience and performance.

- Advantages: Improves network performance, prevents single points of failure.
- Disadvantages: Requires additional infrastructure, may not stop all types of attacks.

-Machine Learning Approaches

Machine learning techniques can identify and mitigate DDoS attacks by analyzing network traffic patterns and learning to distinguish between legitimate and malicious traffic.

- Advantages: Capable of adapting to evolving threats, high detection accuracy.
- Disadvantages: Requires large datasets for training, computationally intensive.

In the SDN architecture, the Openflow switch forwards the main network data at a high speed. The SDN controller is responsible for the forwarding and management of the forwarding decision and the collection of traffic information of switches. In the SDN switch, the core data structure of the forwarding policy management control is the flow table. The SDN manages the relevant network traffic by searching the flow table entries, where the flow entry can forward the packet to one or more interfaces. Each entry includes the header field, the counters, and the actions. The packet forwarding of the switch is based on the flow table. Each flow table is composed of multiple flow entries. The flow table entries form the rules for data forwarding.

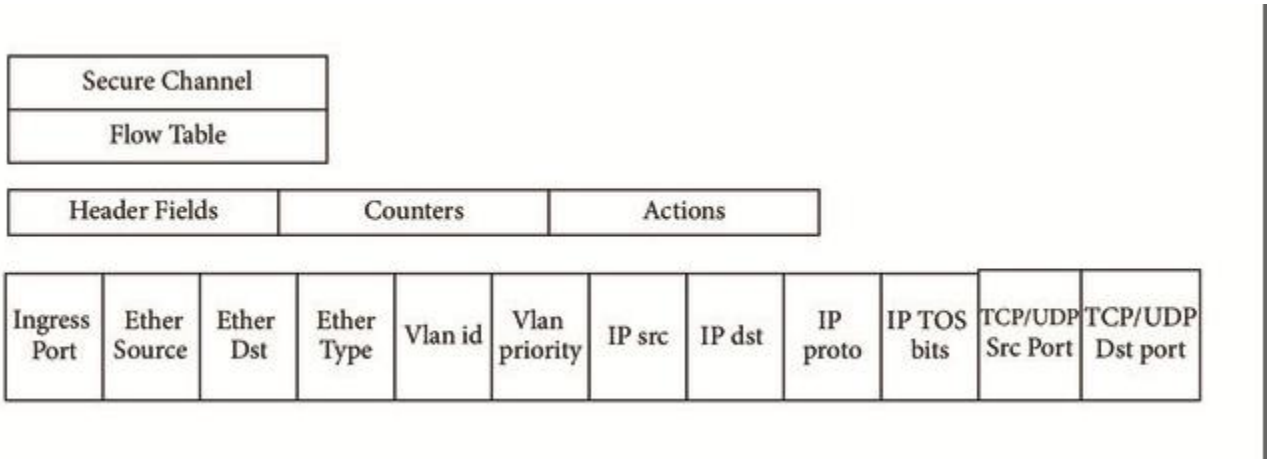


Figure 4.2. The flow table entry structure diagram

The flow diagram of the attack detection consists mainly of the flow state collection, the extraction characteristic values, and the classifier judgment, as shown in Figure . The flow state collection periodically sends a flow table request to the Openflow switch and sends the flow table information replied from the switch to the flow state collection. The characteristic values extraction is mainly responsible for extracting the characteristic values related to the DoS attack from the switch flow table and composing the six-tuple characteristic values matrix. Six-tuple characteristic values information is classified by using an SVM-based algorithm to distinguish between normal traffic and attacking abnormal traffic.

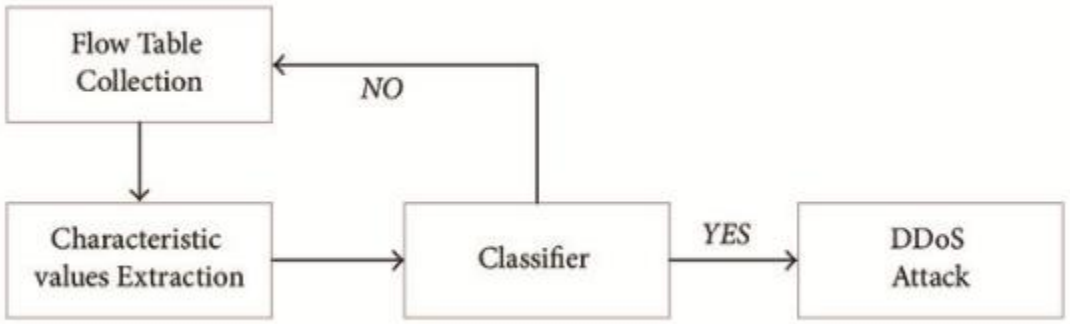


Figure 4.2. The flow diagram of the attack detection

4.2 Implementation in SDN

Defense Against DoS Attack Using Firewall in POX Controller

the implementation of a defense mechanism against a Denial of Service (DoS) attack in a Software-Defined Network (SDN) using the POX controller. The method utilizes a firewall approach to detect and mitigate SYN flood attacks by monitoring packet rates and applying flow rules to drop suspicious packets.

Network Setup

The network setup includes:

- Two POX controllers

- Six Open vSwitches

- Sixteen hosts

Host `h1` operates as an HTTP server, and `h16` acts as the attacker performing the DoS attack using the `hping3` tool. The HTTP server runs on port 8080.

Attack Scenario

1. Reconnaissance: The attacker (`h16`) uses tools like `nmap` to identify open ports on the target (`h1`). Port 8080 is identified as open.

2. Weaponization: The attacker crafts a SYN flood attack using the `hping3` command:

```bash
hping3 -d 300 -S -w 64 -p 8080 --flood --rand-source -V 10.0.0.1
```

3. Delivery: The attacker sends the flood of SYN packets to the target.

4. Exploitation: The SYN packets overwhelm the target, causing it to drop legitimate requests.

5. Installation: Attack packets are observed in the flow tables of the switches.

6. Command and Control: The attacker maintains the attack using random source IPs.

7. Actions on Objectives: The attack disrupts service by exhausting resources.

-Defense Mechanism

A firewall-based defense module is implemented in the POX controller. The defense mechanism involves the following steps:

1. Monitoring the rate of SYN packets.

2. Dropping packets if the rate exceeds a predefined threshold.

3. Dropping packets from IP addresses outside the allowed subnet.

Firewall Script Implementation

The defense module is written in Python and runs as part of the POX controller.

Firewall Script:

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
```

```
import time

log = core.getLogger()

Threshold for rate limiting (total SYN packets per second)
THRESHOLD = 100

Tracking total SYN packets
total_syn_packets = 0
last_cleared = 0

Function to check if IP is within the subnet
def is_in_subnet(ip_addr):
parts = ip_addr.split('.')
if len(parts) < 1:
return False
first_octet = int(parts[0])
return first_octet == 10


def drop_packet(event, reason):
         packet = event.parsed
         msg = of.ofp_flow_mod()
         msg.match = of.ofp_match.from_packet(packet, event.port)
         msg.idle_timeout = 10
         msg.hard_timeout = 30
         msg.priority = 65535
         msg.actions = []  # Empty actions list means drop
         event.connection.send(msg)
         log.warning("Dropping packet: {}".format(reason))

def _handle_PacketIn(event):
         global total_syn_packets, last_cleared
         packet = event.parsed
         ip_packet = packet.find('ipv4')
         tcp_packet = packet.find('tcp')

current_time = time.time()

Clear old entries every second
         if current_time - last_cleared > 1:
         total_syn_packets = 0
         last_cleared = current_time

Check if it's a TCP packet with the SYN flag set
         if tcp_packet and tcp_packet.SYN:
         total_syn_packets += 1

If the total SYN packet rate exceeds the threshold, drop the packet
        if total_syn_packets > THRESHOLD:
        drop_packet(event, "SYN flood detection")
return
```

```
Check if the source IP is within the subnet 10.0.0.0/8
if ip_packet and not is_in_subnet(ip_packet.srcip.toStr()):
         drop_packet(event, "from {} as it is outside the allowed
subnet".format(ip_packet.srcip))
return

Normal packet handling
       msg = of.ofp_packet_out()
       msg.data = event.ofp
       msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))
       msg.in_port = event.port
event.connection.send(msg)

def launch():
         core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
         log.info("Firewall module loaded.")
```

-Explanation of the Code

1. Imports and Logging:

- `pox.core` and `pox.openflow.libopenflow_01` are imported for POX and OpenFlow functionalities.
- `time` is imported to track packet rates over time.
- A logger is initialized for logging events.

2. Global Variables:

- `THRESHOLD`: The maximum allowed SYN packets per second.
- `total_syn_packets`: Tracks the total number of SYN packets.
- `last_cleared`: Timestamp of the last packet count reset.

3. Subnet Check:

- `is_in_subnet(ip_addr)`: Checks if an IP address is within the subnet `10.0.0.0/8`.

4. Drop Packet Function:

- `drop_packet(event, reason)`: Constructs and sends a flow modification message to drop packets matching the specified criteria.

5. Packet-In Handler:

- `_handle_PacketIn(event)`: Handles incoming packets and applies the defense logic.
  - Tracks SYN packets and resets the count every second.
  - Drops packets if the SYN packet rate exceeds the threshold.
  - Drops packets from IPs outside the allowed subnet.
  - Forwards packets normally if no conditions are met.

6. Module Launch:

- `launch()`: Registers the Packet-In event handler and logs that the firewall module is loaded.

Full Explanation of the Code

```python
from pox.core import core
import pox.openflow.libopenflow_01 as of
import time
```

- Line 1-3: Importing necessary modules:
  - `core` from `pox.core` for accessing POX controller functionalities.
  - `of` from `pox.openflow.libopenflow_01` for OpenFlow protocol interactions.
  - `time` for time-related operations.

```python
log = core.getLogger()
```

- Line 5: Retrieves the logger instance from the `core` module, allowing logging messages to be sent to the console or log files.

```python
THRESHOLD = 100
```

- Line 7: Defines a constant `THRESHOLD` which represents the maximum allowed number of SYN packets per second before triggering the defense mechanism.

```python
Tracking total SYN packets
total_syn_packets = 0
last_cleared = 0
```

- Line 9-10: Initializes variables:
  - `total_syn_packets`: Counts the total number of SYN packets observed.
  - `last_cleared`: Stores the timestamp of the last time `total_syn_packets` was reset.

```python
 Function to check if IP is within the subnet
def is_in_subnet(ip_addr):
    parts = ip_addr.split('.')
    if len(parts) < 1:
        return False
    first_octet = int(parts[0])
    return first_octet == 10
```

- Line 12-17: Defines a function `is_in_subnet(ip_addr)`:
  - Splits the IP address into parts based on periods (`.`).
  - Checks if the first octet of the IP address is `10`, indicating it belongs to the subnet `10.0.0.0/8`.
  - Returns `True` if the IP address is within the subnet, otherwise `False`.

```python
def drop_packet(event, reason):
    packet = event.parsed
    msg = of.ofp_flow_mod()
    msg.match = of.ofp_match.from_packet(packet, event.port)
    msg.idle_timeout = 10
    msg.hard_timeout = 30
    msg.priority = 65535
    msg.actions = []
 Empty actions list means drop
    event.connection.send(msg)
    log.warning("Dropping packet: {}".format(reason))
```

- Line 19-28: Defines a function `drop_packet(event, reason)`:
  - Constructs an OpenFlow flow modification message (`msg`) to drop packets.
  - Sets `msg.match` to match the packet received on `event.port`.
  - Sets timeouts (`msg.idle_timeout` and `msg.hard_timeout`) and priority for the flow rule.
  - Sets `msg.actions` to an empty list to indicate dropping the packet.
  - Sends the flow modification message (`msg`) through the connection (`event.connection`).
  - Logs a warning message indicating the reason for dropping the packet.

```python
def _handle_PacketIn(event):
    global total_syn_packets, last_cleared
    packet = event.parsed
    ip_packet = packet.find('ipv4')
    tcp_packet = packet.find('tcp')

    current_time = time.time()

    # Clear old entries every second
    if current_time - last_cleared > 1:
        total_syn_packets = 0
        last_cleared = current_time

     Check if it's a TCP packet with the SYN flag set
    if tcp_packet and tcp_packet.SYN:
        total_syn_packets += 1

         If the total SYN packet rate exceeds the threshold, drop the packet
        if total_syn_packets > THRESHOLD:
            drop_packet(event, "SYN flood detection")
            return
 Check if the source IP is within the subnet 10.0.0.0/8
    if ip_packet and not is_in_subnet(ip_packet.srcip.toStr()):
        drop_packet(event, "from {} as it is outside the allowed
subnet".format(ip_packet.srcip))
        return
Normal packet handling
    msg = of.ofp_packet_out()
```

```
    msg.data = event.ofp
    msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))
    msg.in_port = event.port
    event.connection.send(msg)
```

- Line 30-65: Defines the main packet handling function `_handle_PacketIn(event)`:
  - Retrieves and parses the incoming packet (`event.parsed`).
  - Finds IPv4 and TCP headers within the packet.
  - Updates `current_time` to the current timestamp.
  - Resets `total_syn_packets` count if more than 1 second has passed since the last reset.
  - Checks if the packet is a TCP packet with the SYN flag set.
  - Increments `total_syn_packets` if the condition is met and drops the packet if the count exceeds `THRESHOLD`.
  - Checks if the source IP address of the packet is within the allowed subnet (`10.0.0.0/8`). If not, drops the packet.
  - If none of the above conditions are met, handles the packet normally by forwarding it to all ports (`OFPP_FLOOD`).

```python
def launch():
    core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
    log.info("Firewall module loaded.")
```

- Line 67-71: Defines the `launch()` function:
  - Registers `_handle_PacketIn(event)` to handle `PacketIn` events from OpenFlow switches.
  - Logs an informational message indicating that the firewall module has been successfully loaded.

This script provides a comprehensive firewall-based defense mechanism against SYN flood attacks in an SDN environment using the POX controller. Each component of the script contributes to monitoring and mitigating excessive SYN packets while ensuring legitimate traffic flow is maintained.

Steps to Run

1. Place the script in the pox/ext directory and name it firewall.py.
2. Run the POX controllers with the firewall module loaded:

First Controller (port 6110):

bash

./pox.py log.level --DEBUG openflow.of_01 --port=6110 forwarding.l2_pairs firewall

Second Controller (port 6111):

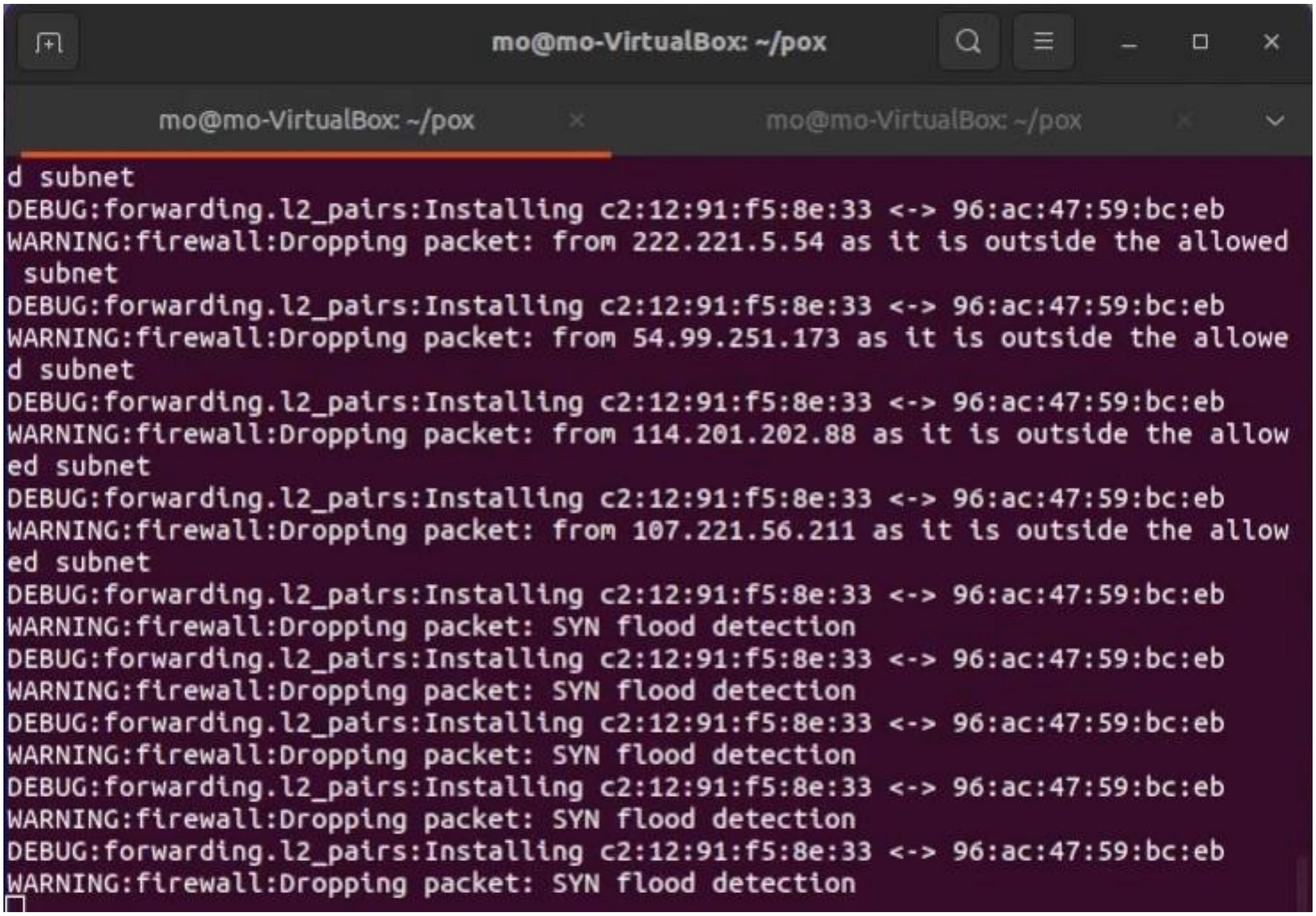
bash

./pox.py log.level --DEBUG openflow.of_01 --port=6111 forwarding.l2_pairs firewall

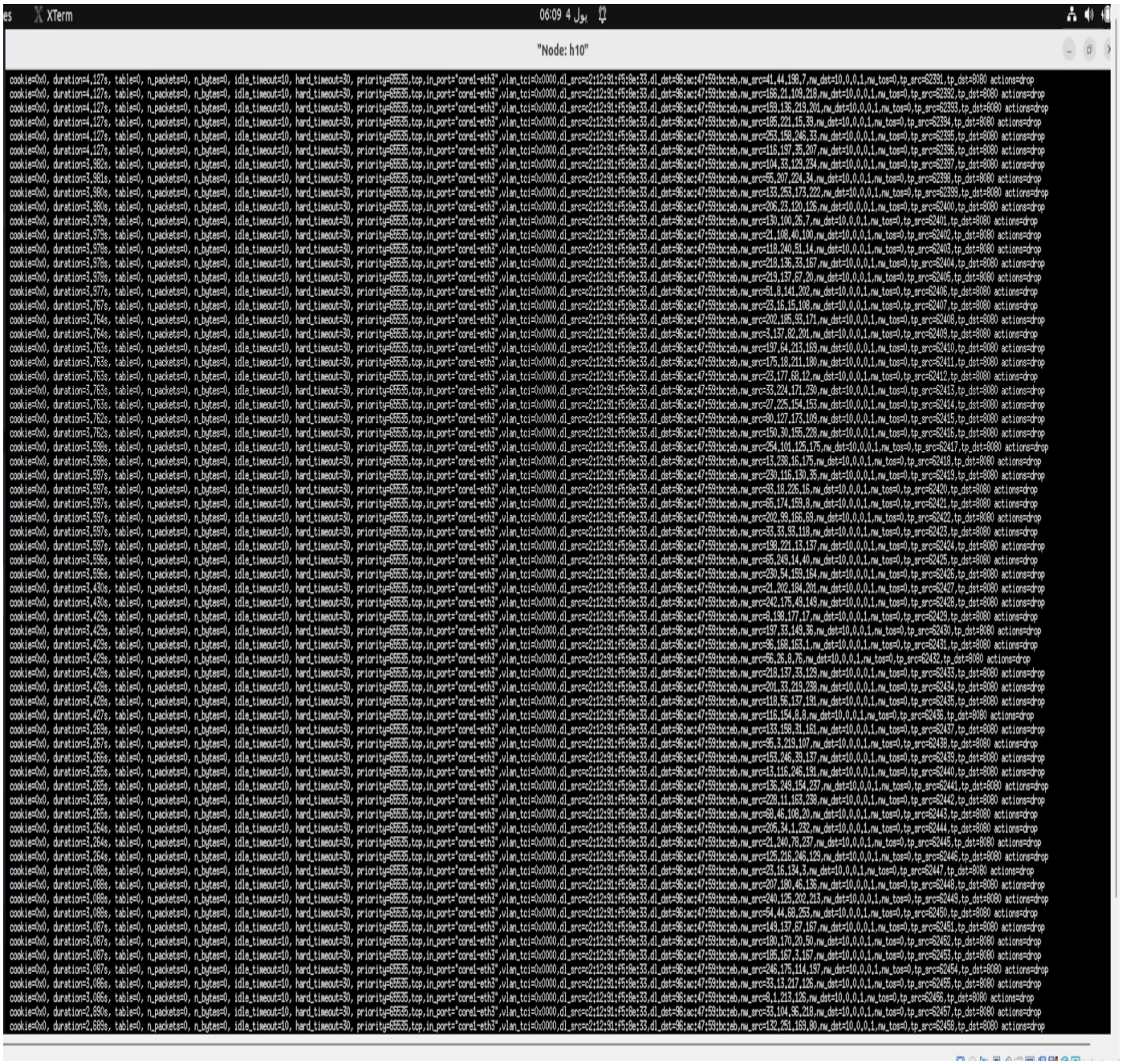
4.3 EVALUATION AND RESULTS

The firewall defense module was successfully implemented and tested in the POX controller. The following observations were made:

- SYN packets exceeding the threshold were dropped, as indicated by log messages.


mo@mo-VirtualBox: ~/pox
mo@mo-VirtualBox: ~/pox
mo@mo-VirtualBox: ~/pox
d subnet
DEBUG:forwarding.l2_pairs:Installing c2:12:91:f5:8e:33 <-> 96:ac:47:59:bc:eb
WARNING:firewall:Dropping packet: from 222.221.5.54 as it is outside the allowed subnet
DEBUG:forwarding.l2_pairs:Installing c2:12:91:f5:8e:33 <-> 96:ac:47:59:bc:eb
WARNING:firewall:Dropping packet: from 54.99.251.173 as it is outside the allowed subnet
DEBUG:forwarding.l2_pairs:Installing c2:12:91:f5:8e:33 <-> 96:ac:47:59:bc:eb
WARNING:firewall:Dropping packet: from 114.201.202.88 as it is outside the allowed subnet
DEBUG:forwarding.l2_pairs:Installing c2:12:91:f5:8e:33 <-> 96:ac:47:59:bc:eb
WARNING:firewall:Dropping packet: from 107.221.56.211 as it is outside the allowed subnet
DEBUG:forwarding.l2_pairs:Installing c2:12:91:f5:8e:33 <-> 96:ac:47:59:bc:eb
WARNING:firewall:Dropping packet: SYN flood detection
DEBUG:forwarding.l2_pairs:Installing c2:12:91:f5:8e:33 <-> 96:ac:47:59:bc:eb
WARNING:firewall:Dropping packet: SYN flood detection
DEBUG:forwarding.l2_pairs:Installing c2:12:91:f5:8e:33 <-> 96:ac:47:59:bc:eb
WARNING:firewall:Dropping packet: SYN flood detection
DEBUG:forwarding.l2_pairs:Installing c2:12:91:f5:8e:33 <-> 96:ac:47:59:bc:eb
WARNING:firewall:Dropping packet: SYN flood detection
DEBUG:forwarding.l2_pairs:Installing c2:12:91:f5:8e:33 <-> 96:ac:47:59:bc:eb
WARNING:firewall:Dropping packet: SYN flood detection

- Packets from IP addresses outside the `10.0.0.0/8` subnet were dropped.



Flow rules can be verified using the `ovs-ofctl` command to ensure that dropped packets are reflected in the flow tables of the switches.

- Normal traffic was handled without disruption.

Conclusion

The firewall-based defense mechanism effectively mitigates SYN flood attacks in an SDN environment using the POX controller. By monitoring packet rates and applying flow rules dynamically, the network can prevent DoS attacks while maintaining normal traffic flow.

This approach provides a scalable and efficient method to enhance the security of SDN networks against common attack vectors.

4.4 Performance Metrics

-Detection Accuracy

Detection accuracy measures the proportion of DoS attacks correctly identified by the detection mechanisms. Higher accuracy indicates better performance of the detection technique.

-False Positive/Negative Rates

False Positives: Legitimate traffic incorrectly identified as malicious.

False Negatives: Malicious traffic incorrectly identified as legitimate.

Balancing these rates is crucial for effective DoS mitigation.

4.5 Comparative Analysis

The study compares the effectiveness of DoS mitigation techniques in SDN with those in traditional networks. This analysis highlights the advantages and limitations of SDN-specific solutions. Denial of Service (DoS) attacks remain a significant threat to network security, capable of rendering services unavailable by overwhelming them with excessive traffic. Traditional networks, with their vertically integrated data and control planes, struggle to effectively mitigate such attacks. Software-Defined Networking (SDN) offers a revolutionary approach by decoupling the data plane from the control plane, allowing for centralized control and dynamic policy enforcement. This paper compares the effectiveness of DoS mitigation techniques in SDN with those in traditional networks, highlighting the advantages and limitations of SDN-specific solutions.

Traditional networks are inherently complex due to their tightly coupled data and control planes, which are often manufacturer specific. This architecture makes centralized control and uniform policy enforcement difficult, leading to challenges in scalability and security management. The decentralized nature of traditional networks often results in delayed response times to network threats, such as DoS attacks, which can cause significant service disruptions. Software-Defined Networking (SDN) addresses these issues by separating the control plane from the data plane, enabling centralized management and dynamic policy enforcement. This separation allows for greater flexibility, improved network visibility, and rapid response to network changes and threats. This study aims to compare the effectiveness of DoS mitigation techniques in SDN with those in traditional networks, drawing from various research findings.

Dos attacks aim to make a network service unavailable by overwhelming it with a flood of traffic. In traditional networks, the decentralized control and rigid structure hinder effective and timely responses to these attacks. Common mitigation techniques include rate limiting, firewalls, and intrusion detection systems (IDS), but these often fall short due to the lack of coordinated control and real-time

adaptabilities provides a paradigm shift by centralizing control and decoupling it from the data plane. This architecture allows for dynamic flow management, real-time monitoring, and rapid deployment of defense mechanisms. SDN controllers have a global view of the network, enabling more effective monitoring and response to DoS attacks.

This study reviews existing literature to compare the efficacy of DoS mitigation techniques in SDN and traditional networks. Key performance metrics such as packet loss, bandwidth utilization, latency, and the ability to handle attack traffic are analyzed. The comparison includes research studies that focus on various SDN-specific and traditional network defense mechanisms. In traditional networks, DoS mitigation relies on distributed control mechanisms. Techniques like rate limiting, firewalls, and IDS are used to filter and manage traffic. However, the lack of centralized control leads to inefficient resource allocation and delayed response times. For example, research has shown that traditional rate limiting can reduce network performance under heavy attack due to its static nature and inability to adapt to changing traffic patterns.SDN offers significant improvements in DoS mitigation through centralized control and dynamic policy enforcement.

Studies have demonstrated the effectiveness of SDN in maintaining network performance and availability under attack conditions. For instance, reference highlights the use of a secure and optimized energy framework for Blockchain-enabled software-defined which ensures network security and consistency. Another study shows that SDN can decrease packet loss and increase bandwidth during attacks by dynamically adjusting flow rules. Moreover, SDN's ability to implement fine-grained flow control allows for more precise attack mitigation. Research presents a secure SDN framework capable of preventing spoofing attacks and DoS with minimal configuration overhead. Similarly, reference illustrates that Flow Keeper, an SDN-based solution, can maintain more than 80% bandwidth during DoS attacks by filtering out unauthorized topology changes and forged packets.

The comparison between traditional networks and SDN reveals that SDN's centralized control and flexibility offer significant advantages in mitigating DoS attacks. Traditional networks' decentralized nature results in slower response times and less efficient resource utilization, making them less effective in handling large-scale attacks. In contrast, SDN can quickly adapt to changing traffic patterns and deploy countermeasures in real-time, ensuring better network performance and security.

However, SDN is not without its limitations. The centralized nature of SDN controllers can become a single point of failure if not properly managed. Additionally, the complexity of SDN implementations requires careful planning and management to ensure reliability and security provides a robust framework for mitigating DoS attacks, offering significant advantages over traditional networks. Its centralized control, real-time adaptability, and enhanced visibility into network traffic make it a superior choice for modern network security challenges. Future research should focus on further optimizing SDN-based defense mechanisms and exploring their application in diverse network environments.

- Comparison of Different Mitigation Techniques

Different mitigation techniques are compared based on their performance metrics. This comparison provides insights into the strengths and weaknesses of each approach, guiding the selection of the most effective strategy.

Mitigation Technique	Effectiveness	Complexity	Scalability	Resource Consumption	Notes
Firewall-Based Defense Module	High	Medium	Medium	Medium	Implemented in the POX controller to filter traffic based on rules; effective for blocking known threats but requires regular updates and rule management.
Rate Limiting	High	Low	High	Low	Limits the rate of incoming traffic to prevent overload but may require fine-tuning to balance between security and performance.
Flow Rule Management	High	Medium	Medium	Medium	Dynamically manages flow rules to prioritize legitimate traffic; can become complex with large-scale networks.
Distributed Controllers	Very High	High	Very High	High	Enhances resilience and load distribution but requires more resources and

					proper synchronization between controllers.
Anomaly Detection	High	High	High	High	Uses machine learning to detect malicious traffic patterns; effective but requires substantial computational power and training data.
Access Control Lists (ACLs)	Medium	Low	Medium	Low	Filters traffic at the switch level; straightforward to implement but may not handle sophisticated attacks effectively.
Load Balancing	Very High	Medium	Very High	Medium	Distributes traffic evenly across multiple resources to prevent overload; effective in large and dynamic environments.
Topology-Aware Defense	High	Medium	Medium	Medium	Uses network topology information to isolate affected segments; effective but depends on the network's

					complexity and topology.
Controller Replication	High	High	High	High	Provides redundancy for controllers to ensure availability; involves high resource consumption and synchronization overhead.
Flow Aggregation	Medium	Medium	High	Medium	Reduces the number of flow entries by aggregating similar flows; helps prevent flow table overflow but requires intelligent grouping mechanisms.
Moving Target Defense (MTD)	High	High	Medium	High	Regularly changes network configurations to confuse attackers; increases security but adds management complexity.
Signature-Based Detection	Medium	Low	Medium	Low	Detects known attack patterns efficiently but may not be effective against new or

					unknown attacks.
Behavioral Analysis	High	High	High	High	Establishes baselines of normal behavior to detect deviations; effective but requires continuous monitoring and sophisticated analysis.

Table 1. Comparison of Different Mitigation Techniques

4.6. CONCLUSION AND FUTURE WORK

The study successfully demonstrates the impact of DoS attacks on SDN and evaluates the effectiveness of various detection and mitigation techniques. Hybrid detection methods and machine learning-based mitigation strategies are identified as the most promising approaches. In the past, there was a need to solve the difficulties facing traditional networks and to adopt new methods of communication to avoid these difficulties. From here, SDN Networks have become a suitable solution to those difficulties. It is a real development in the world of networks, as it enjoys more features that simplify network management, its ability to program, improve networks, and flexibility.

In SDN networks, the control plane is separated from the data plane, and the network is controlled in a centralized manner, in which the controller is responsible for managing the network, which leads to cost reduction and network complexity, which makes the network administrators need to configure or manage the network and its devices separately. SDN Networks adds powerful programming interfaces that facilitate the design of various network applications and programs such as traffic engineering and quality of service applications. New configurations and applications can be added within the network and troubleshooting lines.

SDN Networks improve network performance, enable new services to be implemented, and experience flexibility, which encourages innovation. SDN networks are used as they provide faster services due to their multiple advantages. SDN Networks are the leading networks and the focus of great interest among researchers and authors of different nationalities because of their many advantages, which we list as an example but are not limited to (centralized control, easy to deal with and programmable ..... etc.). Then many users also use it to accomplish their tasks with high accuracy and faster time than other traditional networks. Nevertheless, SDN faces many, many difficulties, problems, and fierce attacks,

especially the attack of the DoS, which is the focus. SDN networks also have some flaws and attacks that affect the network and its performance and reduce its efficiency such as malicious applications, attacks that are in the form of real users, and permission SDN networks also have flaws attacks that affect the network and its performance and reduce its efficiency such as malicious applications, attacks that are in the form of real users and permission grabbing.

Various attacks such as impersonation in SDN networks lead to resource exhaustion and network vulnerabilities. Also, the use of a single controller in a single point of failure may cause damage to the network and lead to bottlenecks of incoming requests, which makes SDN's architecture vulnerable to any attack. The data plane is affected by control plane security as when there is a control plane attack the whole network is compromised as it includes many data plane nodes. It is clear that SDN networks suffer from some challenges despite the great openness and preference in their use among the networks, and from the attacks that affect the network and reduce its efficiency, DoS attacks, and many types of research helped reduce that attack, as we mentioned previously, such as solutions that rely on The guard, including the communication relay, and the operation of the actuators also, including the access controller and the classification unit. Also, some solutions depend on a certain threshold, where a continuous analysis of the communications between the controller and the switches are carried out, some of which depend on statistical techniques and accurate synchronization of time between the forwarding devices.

Among the proposed solutions are those that depend on priority and trust, whereby excessive low-priority requests are dropped, including what is proposed as a multi-level security mechanism. Also, among the solutions is what depends on changing the bandwidth and adding an idle timeout and dealing with only Openflow version 1.3. There are also solutions that reduce the attack rate of DoS, some of which consist of two units, and others that consist of more. Also, there are dynamics and observational-based solutions, and entropy-based solutions. Some solutions mitigate DoS attacks via IoT networks and mitigate vehicular attacks. One of the solutions is using fog computing to mitigate these attacks. All these solutions were previously listed in more detail. We found DoS attacks received a lot of interest and found some effective solutions and other solutions of limited effectiveness. Hence, we proposed an effective solution to reducing DoS attacks by a large percentage.

Our approach is based on more valuable principles such as the repetition of different header fields, the threshold, the trust value of network users, tracking sender's Suspects, and blocking the detected attack area. Our proposed approach consists of two stages: 1- Extracting the header fields where the header fields are tracked to detect DoS attacks where the table values are compared to the predefined thresholds where the network traffic becomes normal if the header field values are smaller than the predefined threshold values. 2- Calculating the trust value-based on the header field information in which packets are prioritized, the controller uses the information provided by the packet header fields to more efficiently calculate the user trust value. When a new connection is started, a host in the simulated system runs a new packet in which both packets belonging to the same stream will eventually be managed in the physical system of devices i.e., this data level significantly reduces the number

4.7 Contributions to the Field

This research contributes to the field by providing a comprehensive analysis of DoS attacks in SDN, proposing effective mitigation strategies, and offering a detailed case study demonstrating their implementation and evaluation.

4.8 Limitations of the Study

The study's limitations include the use of a specific SDN topology and a limited range of attack scenarios. Future work should explore more diverse network setups and a broader spectrum of DoS attacks.

4.9 Recommendations for Future Research

SDN networks need more scientific research, which helps in exploring many of the features that may not have been discovered yet and the well-known features of these networks, which helped without complexity in solving the problems we were experiencing in the traditional networks, which are no longer used now, and SDN networks have become a pioneer and preferred in use. Therefore, it is of great interest to researchers in terms of addressing network attacks and improving network performance. In the future, we aspire to delve deeper into this type of network and discover the various advantages and exploit them more to improve the process of communication within the network and to ensure higher quality, strong privacy, and security of distinction We hope in the future to build on this research and try to detect and treat other types of attacks with higher accuracy and improve network performance

- Advanced Machine Learning Models: Developing more sophisticated models for DDoS detection and mitigation.
- Blockchain-Based Security: Exploring the potential of blockchain technology for enhancing SDN security.
- Comprehensive Testing: Conducting large-scale experiments to validate the effectiveness of proposed solutions in real-world scenarios.

5.References

1) OpenFlow-enabled SDN and Network Functions Virtualization ONF Solution Brief February 17, 2014.
2) Software-Defined Networking a Comprehensive Survey, D. Kreutz and F. M. V. Ramos, June 15, 2014.

3) Retrieved from Mininet · GitHub
4) Retrieved from www.openvswitch.org: https://www.openvswitch.org

5) High Availability in Software-Defined Networking using Cluster Controller: A Simulation Approach, Abhimata Zuhra Pramudita

6) M. Priyadarsini and P. Bera, “Software defined networking architecture, traffic management, security, and placement: A survey,” Comput. Networks, vol. 192, p. 108047, 2021

7) Gadallah, Waheed G and Omar, Nagwa M and Ibrahim, Hosny M, “Machine Learning-based Distributed Denial of Service Attacks Detection Technique using New Features in Software-defined Networks,” International Journal of Computer Network and Information Security (IJCNIS), vol. 13, no. 3, 15— 27, 2021.

8) C.E. Rothenberg et al. “Software defined networking: A comprehensive survey” proceedings of the IEEE, volume. 103, num. 1, pp. 14-76, 2015

9) C.E. Rothenberg et al. “Software defined networking: A comprehensive survey” proceedings of the IEEE, volume. 103, num. 1, pp. 14-76, 2015

10) H. Sufiev and Y. Haddad, "A dynamic load balancing architecture for SDN," in 2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE), 2016, pp. 1- 3.

11) L. Zhu, M. M. Karim, K. Sharif, F. Li, X. Du, and M. Guizani, "Sdn controllers: Benchmarking & performance evaluation," arXiv preprint arXiv:1902.04491, 2019.

12) J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, and K. Prabhu, "iPerf-The ultimate speed test tool for TCP, UDP and SCTP," línea]. Available: https://iperf. fr.[Último acceso: 23 Mayo 2018], 2014.

13) S. I. Ullah, A. Salam, W. Ullah, and M. Imad, "COVID-19 lung image classification based on logistic regression and support vector machine," in European, Asian, Middle Eastern, North African Conference on Management & Information Systems, 2021: Springer, pp. 13-23.

14) M.Imad, N. Khan, F. Ullah, M. A. Hassan, and A. Hussain, "COVID-19 classification based on Chest X-Ray images using machine learning techniques," Journal of Computer Science and Technology Studies, vol. 2, no. 2, pp. 01-11, 2020.

15) A. Salam, F. Ullah, M. Imad, and M. A. Hassan, "Diagnosing of Dermoscopic Images using Machine Learning approaches for Melanoma Detection," in 2020 IEEE 23rd International Multitopic Conference (INMIC), 2020: IEEE, pp. 1-5.

16) M. Imad, F. Ullah, and M. A. Hassan, "Pakistani Currency Recognition to Assist Blind Person Based on Convolutional Neural Network," Journal of Computer Science and Technology Studies, vol. 2, no. 2, pp. 12-19, 2020.

17) M. Rizwan et al., "Risk monitoring strategy for confidentiality of healthcare information," Computers and Electrical Engineering, vol. 100, p. 107833, 2022.

18) R. V Boppana, R. Chaganti, and V. Vedula. "Analyzing the vulnerabilities introduced by ddos mitigation techniques for software-defined networks." National Cyber Summit. Springer, Cham, 2019. [38] V. Ravi, R. Chaganti and M. Alazab, "Deep Learning Feature Fusion Approach for an Intrusion Detection System in SDNBased IoT Networks", IEEE Internet of Things Magazine, vol. 5, no. 2, pp. 24-29, 2022. Available: 10.1109/iotm.003.2200001.

19) M. A. Hassan, S. Ali, M. Imad and S. Bibi, “New Advancements in Cybersecurity: A Comprehensive Survey” Big Data Analytics and Computational Intelligence for Cybersecurity,pp. 3-17, 2022.

20) M. Imad, M. A. Hassan, S. H Bangash, “A Comparative Analysis of Intrusion Detection in IoT Network Using Machine Learning” In Big Data Analytics and Computational Intelligence for Cybersecurity, pp. 149-163, 2022. Springer

21) Burkhalter, Lukas and Lycklama, Hidde and Viand, Alexander and K{\"u}chler, Nicolas and Hithnawi, Anwar, “Rofl: Attestable robustness for secure federated learning,” arXiv preprint arXiv:2107.03311, 2021.

22) https://www.ijesrt.com/index.php/J-ijesrt
23) K. Benzekki, A. El Fergougui and A. Elbelrhiti Elalaoui, "Softwaredefined networking (SDN): a survey," Security and Communication Networks,
24) Retrieved from mininet.org: https://mininet.org/walkthrough/
25) Retrieved from wftpserver.com: https://www.wftpserver.com
26) Retrieved from stackoverflow.com: https://stackoverflow.com/questions/50421826/connecting-mininet-host-to-the-internet
27) H. G. Ahmed and R. Ramalakshmi, "Performance Analysis of Centralized and Distributed SDN Controllers for Load Balancing Application," 2018 2nd International Conference on Trends in Electronics and Informatics, 2018
28) Network Functions Virtualization (NFV): Network Operator Perspectives on Industry Progress, ETSI, October 2013.
29) Open Networking Foundation (ONF), ‘‘Conformance test specification for OpenFlow switch specification,’’ Tech. Rep., Jun. 2013

30) https://www.techscience.com/cmc/v77n2/54785/html

31) https://www.researchgate.net/publication/378075465_HLD-DDoSDN_High_and_low-rates_dataset-based_DDoS_attacks_against_SDN

32) https://www.researchgate.net/publication/379949650_Quality_of_Service_Based_Performance_Evaluation_of_Controllers_in_Software_Defined_Networks

33) https://www.researchgate.net/publication/260838090_Combining_OpenFlow_and_sFlow_for_an_effective_and_scalable_anomaly_detection_and_mitigation_mechanism_on_SDN_environment

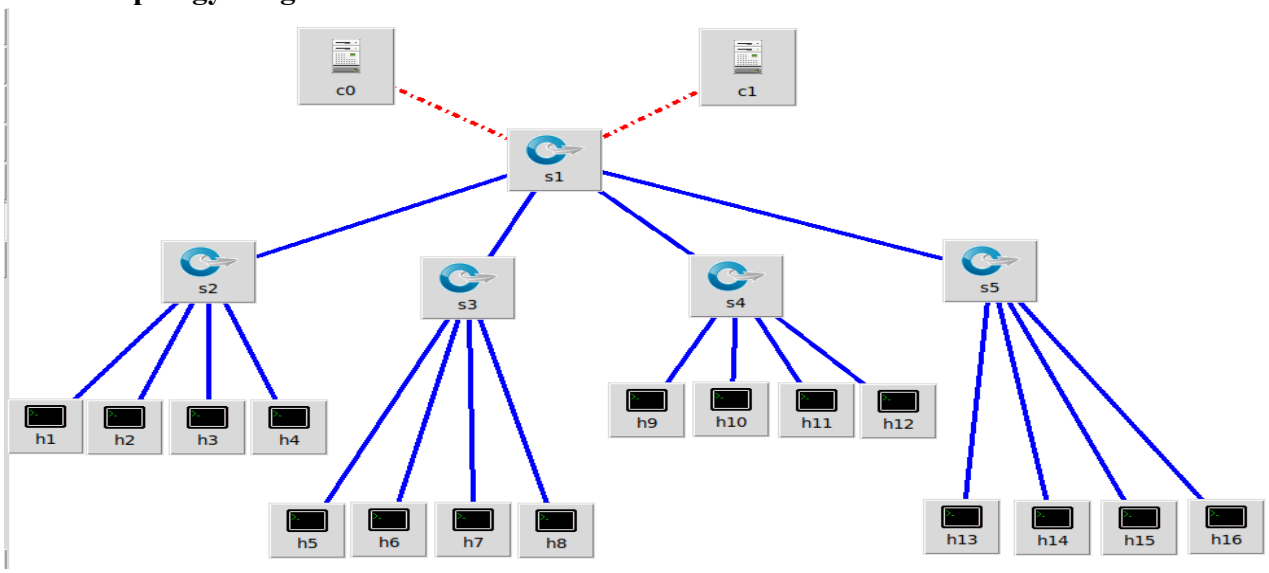
34) https://dl.acm.org/doi/pdf/10.1145/3132062.3132074
35) https://link.springer.com/chapter/10.1007/978-981-15-5341-7_75
36) https://www.researchgate.net/publication/346477420_Detection_and_Prevention_from_DDoS_Attack_Using_Software-Defined_Security

37) https://www.researchgate.net/publication/343277164_Software-defined_Networking_Based_DoS_Defense_Mechanisms
38) https://www.mdpi.com/2079-9292/11/23/4065
39) https://github.com/topics/dos-simulation
40) https://github.com/rsukumar75/DOS-Attack-Detection
41) https://serverfault.com/questions/546984/ddos-attack-simulation-using-mininet
42) https://ieee-collabratec.ieee.org/app/community/87/IEEE-Software-Defined-Network-SDN-Initiative/activities
43) https://ieeexplore.ieee.org/document/10073469
44) https://onlinelibrary.wiley.com/doi/full/10.1002/eng2.12697
45) https://www.researchgate.net/publication/363822657_Detection_of_DDoS_attack_in_SDN_network_using_Entropy_in_Pox_controller
46) https://www.techscience.com/cmc/v71n1/45423/htmlhttps://www.researchgate.net/post/What_are_limitations_of_rate-limiting_at_SDN_switches
47) https://www.worldresearchlibrary.org/up_proc/pdf/1819-153819259139-41.pdf
48) https://www.cpp.edu/faculty/tromar/detection-of-ddos-in-sdn-environment-usingentropy-based-detection.pdf
49) https://www.researchgate.net/publication/316456006_The_effects_of_DoS_attacks_on_ODL_and_POX_SDN_controllers
50) https://github.com/R1ck404/DDoS-Detection
51) https://github.com/topics/ddos-detection?o=asc&s=stars
52) https://medium.com/@tommasobona04/how-to-code-a-dos-attack-in-python-by-tommaso-bona-b5387fc9c573
53) https://www.slideshare.net/slideshow/ddos-mitigation-tools-and-techniques/54052942
54) https://www.researchgate.net/publication/355085556_Analysis_of_Different_Attacks_on_Software_Defined_Network_and_Approaches_to_Mitigate_using_Intelligent_Techniques

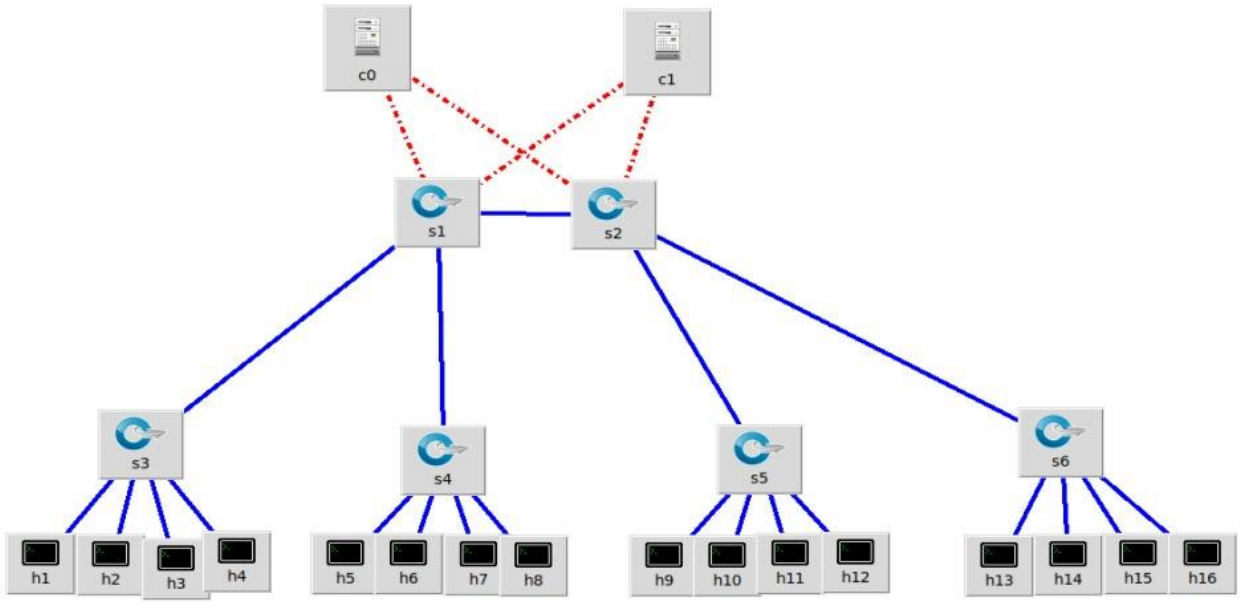
6.Appendices

Appendix A: Network Topology Diagrams

- **Initial Topology Diagram**:



- **Updated Topology Diagrams**:



Appendix B: Python Scripts

- **Network Configuration Script**: Full Python script used to set up the SDN network in Mininet.
- **Controller Script**: Python script for the POX controller that includes the defense mechanism against DoS attacks.

Appendix C: Configuration Files

- **Mininet Configuration**: Detailed configuration settings for the Mininet environment as we discussed before.
- **POX Controller Configuration**: Configuration settings and modifications made to the POX controller as we discussed before to mitigate against Dos attack.

Appendix D: Data and Logs

- **Wireshark Captures**: Sample capture files or screenshots of Wireshark traffic analysis during the DoS attack and defence as we see before.
- **TCPdump Outputs**: Sample outputs from tcpdump showing packet captures and analysis as we see before.

Appendix E: Flow Rules

- **OpenFlow Switch Flow Rules**: Detailed list of flow rules configured on the OpenFlow switches as we see before.
- **Changes to Flow Rules**: Documentation of changes made to the flow rules to mitigate the DoS attack as we see before.

Appendix F: Test Results

- **Performance Metrics**: Tables and charts showing the performance metrics of the network before and after the implementation of the defense mechanisms as we discussed before.
- **Latency and Throughput**: Results of latency and throughput tests conducted during the project as we discussed before.

Appendix G: Additional Resources

- **Reference Material**: List of additional reference materials, such as research papers, articles, and books that were consulted during the project.
- **Useful Tools and Commands**: Summary of the tools and commands used throughout the project, along with brief descriptions of their purposes.