Open Flow-Based Monitoring In IXP Manager Using SDN Controller

By

Sayyaf Haider Warraich

00000171365

Supervisor

Dr. Hasnat Khurshid

A thesis submitted in conformity with the requirements for the degree of *Master of Science* in Electrical (Telecommunication) Engineering

Military College Of Signals (MCS)

National University of Sciences and Technology (NUST)

Islamabad, Pakistan

(November 2019)

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by Mr **Sayyaf Haider Warraich**, Registration No. **00000171365** of **Military College of Signals** has been vetted by undersigned, found complete in all respect as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial, fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the student have been also incorporated in the said thesis.

Signature :____________________

Name of Supervisor : Lt Col Hasnat Khurshid, PhD

Date :____________________

Signature (HOD) :____________________

Date :____________________

Signature (Dean) :____________________

Date :____________________

Certificate of Originality

I, *Sayyaf Haider Warraich* declare that this thesis titled "Open Flow-Based Monitoring In IXP Manager Using SDN Controller" and the work presented in it are my own and has been generated by me as a result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a Master of Science degree at NUST
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at NUST or any other institution, this has been clearly stated
3. Where I have consulted the published work of others, this is always clearly attributed
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work
5. I have acknowledged all main sources of help
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself

Sayyaf Haider Warraich,
00000171365

Copyright Notice

This thesis is dedicated to *my beloved parents*

Abstract

Measuring the performance of a network is not a new domain and there has been a lot of research done on how to improve the performance of a network. The performance of a network cannot be improved unless we monitor the network itself. In that context OpenFlow based monitoring in the SDN-IXP manager using the SDN (Software Defined Networking) controller is a novel approach to manage and monitor IXP (Internet Exchange Point). The IXP acts as a local transit point for different ISPs (Internet Service Providers) and content providers. This approach helps in reducing the financial burden by lowering the cost of transit point and network's operation.

The said approach also improves system efficiency with better control, reduces latency by direct connections, and ensures security of the data by keeping traffic in a local network. SDN provides central network monitoring by keeping eye on the whole network, traffic engineering can also be leveraged by using network programmability for applications like centralized device configurations, load balancing, firewall and traffic optimization in the network. The SDN technology has benefits in diverse industries and one of those is IXP.

This thesis proposes a novel Laravel based management framework for redesigned IXP-Manager to monitor flow counters between the SDN controller and OpenFlow Switch through Grafana Platform. The latter is used to show stats after getting data from the database and then a user interface is added to the IXP manager dashboard using redirection. The proposed approach allows SDN IXP Manager to define rules and establish peering between different ISPs in an IXP environment by passing data from SDN-IXP manager to SDN-Controller's YAML file directly.

Acknowledgments

I pay my gratitude to Allah almighty for blessing me a lot and without His guidance I couldn't complete this task. I am thankful to Dr. Hasnat Khurshid for guiding and encouraging me to complete my thesis. His timely and efficient contributions helped me to shape the thesis into its final form and I express my gratefulness for his sincere supervision all the way.

I am also thankful to my GEC members (Dr Imran Rashid, Dr Adil Masood and Dr Bilal Rauf) and Department of Electrical Engineering (MCS) for providing me with an academic base, which enabled me to complete this thesis. I am also very thankful to IGNITE Research and Development fund and Bitsym for providing me a platform to validate or test my scenario. I'd like to thank my parents who have provided me every comfort of life and their utmost affection and support made me reach where I am today.

Contents

List of Figures

List of Tables

List of main Abbreviations

Abbreviations

SDN	Software Defined Networking
IXP	Internet Exchange Point
ISP	Internet Service Provider
DDOS	Distributed Denial of Service
ACL	Access Control List
OSI	Open Systems Interconnection
MAC	Medium Access Control
BGP	Border Gateway Protocol
ONF	Open Networking Foundation
API	Application Programming Interface
ForCES	Forwarding and Control Element Separation
OVS	OpenFlow Virtual Switch
WLAN	Wireless Local Area Network
NCP	Network Control Point
PCE	Path Computation Element
RCP	Routing Control Platform

OSC	Open SDN Controller
ONOS	Open Networking Operating system
VLAN	Virtual Local Area Network
DB	Data Base
GUI	Graphical User Interface
AS	Autonomous System
RIP	Routing Information Protocol
OSPF	Open Shortest Path First

Chapter 1

Introduction

The composition of thesis includes the architecture and implementation of OpenFlow based monitoring in IXP using SDN controller that allows traditional IXP members, to use existing IXP Manager application to monitor stats [1] after converting it into SDN based IXP and allows it to define rules in IXP Manager application for application specific peering or mitigation of DDoS attack [2]. In first section, the Overview, motivation and problem statement of the proposed scheme are discussed, whereas in next section, architecture, design mechanism, observations and future work has been discussed.

1.1 Overview

As the internet is a growing technology worldwide, different techniques and mechanism are adopted to overcome challenges faced by the customer or community in terms of speed, bandwidth, latency, connectivity etc. In the early days of internet, the usage was limited as compared to today, where all the applications need seamless communication and connectivity, social media usage has increased worldwide involving high-quality graphics and HD videos, those are shared over the internet which consumes a lot of bandwidth. This is the bottleneck where a lot of researchers are working on in order to optimize bandwidth and there are different mechanisms adopted for best bandwidth utilization.

Secondly, the major concern worldwide is privacy and security, people and industry are very much concerned about their privacy. Due to evolution of technology the users are now very much dependent on innovative technologies, which requires a lot of information to be processed in order to perform at the optimal levels. At the national level, every country wants to ensure that the privacy of its citizens is preserved. So, in the context of bandwidth utilization and ensuring privacy, the concept of IXP was introduced. Furthermore, there is a need to make IXP optimized in order to convert it into SDN based IXP to follow international best practices.[3]

With the development of SDN enabled IXP, different challenges also emerge, one of them is monitoring and control of traffic that is routed between different ISPs and content providers. In order to overcome the aforementioned challenges, a complete design and mechanism with implementation are discussed in this thesis. In this Section, the Overview, motivation and problem statement of the proposed scheme are discussed. whereas in later this thesis, architecture, design mechanism, observations and future work will be discussed in Chapter 3, 4 and 5.

1.2 Motivation

There are multiple ISPs peering at IXP and to ensure security, programmability optimal performance the traditional IXP is taking steps to convert into SDN enabled IXP. An application for management of SDN enabled IXP with a mechanism to monitor and control traffic is required, since the traditional IXP used IXP manager for that purpose and hence, is not compatible with the SDN enabled IXP. Secondly, there is no mechanism for IXP management team to control the traffic. By defining rules, IXP Management team can control traffic by redirecting traffic, mirroring traffic for machine learning algorithms, optimization and application-specific peering.

There is a need to redesign the IXP manager introducing additional functionalities to make it compatible with SDN technology and integrating the support of the monitoring and control mechanism of traffic. By redesigning IXP manager, IXP management team will have the ability to monitor traffic and apply rules for traffic control, which would help in mitigating the security attacks/threats such as DDoS and to apply redirection on it.

1.3 Problem Statement

Currently, SDN enabled IXPs have no such platform available where customer/ administration portal is available in IXP management tool. Whereas, in traditional IXP there is a well known tool named as IXP Manager, available for management of traffic with an integrated customer/administration portal used by more than 68 IXPs around the world. But the issue with it remains the same, it is not compliant with SDN enabled IXP. However, there are a very few IXPs globally, that have transformed into SDN enabled IXP, so whenever an IXP has to transform into an SDN enabled IXP, it has to develop a dashboard or management portal, initially with minimum features for temporary monitoring. It is not necessary to develop customer administrator portals in the beginning, as it is difficult for any IXP/Operator to handle two different portals at the same time or to migrate to a new portal.

1.4 Proposed approach

As IXP Manager has already been designed to monitor stats of traditional IXP that's why IXP management team manages the IXP manager. So, my proposed scheme is based on Software Defined Networking paradigm an innovative technology in the domain of networking. In this scheme the traditional IXP Manager is converted to Software Defined Networking Based IXP manager to monitor the stats of SDN traffic using gauge controller. Secondly, IXP manager has to be redesigned to define rules from top of the controller to optimize the traffic path or to prevent the security attacks using ACLs and to establish a BGP peering of ISPs. My proposed architecture is complete and described in chapter 4.

Following diagram Figure 1.1 is used to show a proposed architecture.

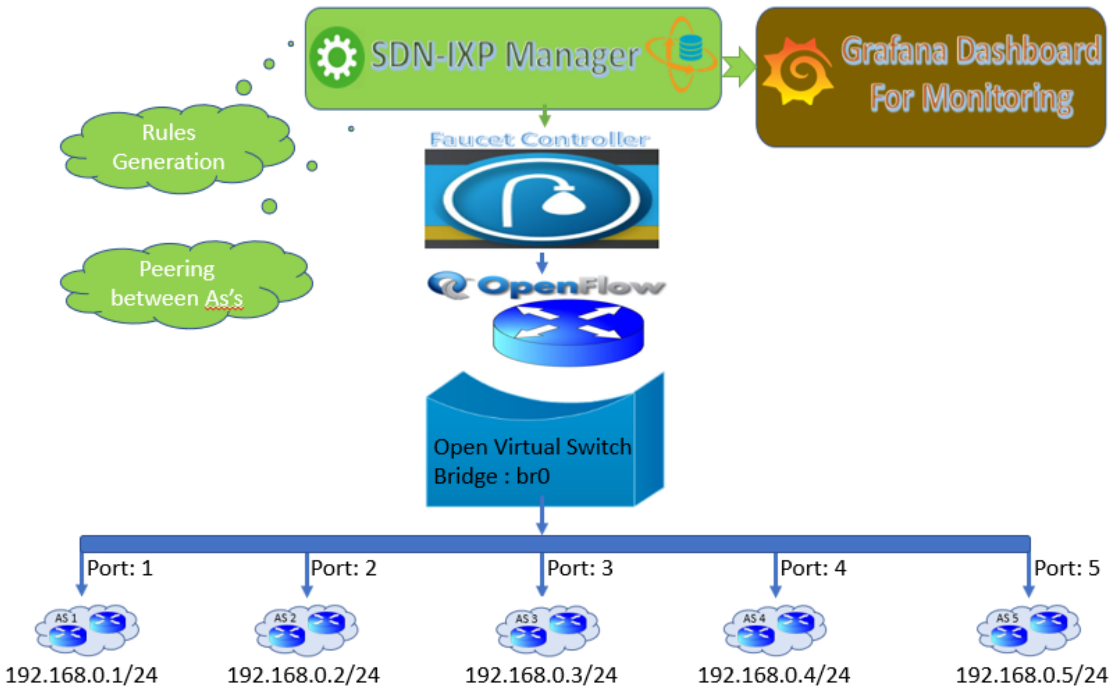


Figure 1.1: Software Define Internet Exchange Point Architecture for Monitoring and rules generation to control or optimize traffic.

1.5 Contributions

In the research work, my main contribution is to design a prototype, which will be used to monitor the traffic of the SDN enabled IXP at one platform, and on the basis of traffic analysis, the rules will be defined and peering will be established between different ISPs. My proposed scheme is divided into further four sub-systems, which are defined below.

- Link SDN-IXP Manager to a dashboard for monitoring SDN traffic.
- Vlan Selection and OpenFlow Switch addition in SDN-IXP Manager.
- Autonomous System addition and peering establish through SDN-IXP Manager.
- Rules definition for traffic control through SDN-IXP Manager.

Chapter 2

Literature Review

This Section of the thesis explains the need, uses and components required in redesigning of SDN-IXP Manager for monitoring of OpenFlow traffic stats in SDN-IXP scenario and establishment of peering between different ISP's.

2.1 Traditional Network

In a traditional network, the network nodes communicate with the other nodes over a network. Following are the sub sections which briefly explain the flow mechanism of traditional network.

2.1.1 Overview of Traditional Network

Currently, the computer networks are based on two common types of communications for switching.

- Packet Switching
- Circuit Switching

The circuit switching is the method in which a dedicated line is provided to both network entities, which is an old method of communication as it causes latency and is costly. In the current era, packet switching is widely used in which there is no need of dedicated line for both network entities while data is divided into small chunks which are further referred to as packets. Major benefit of packet switching is that it chooses the optimal path to reach the destination and it is cost-effective [4].

Now a day's mostly packet switched networks are used which work on the principle of reaching the destination through a series of interconnected network switches and routers. Traditional network follows OSI Model in which switching can work on layer 2 depending upon Mac addresses also called as data link layer and routing can work on layer 3 depending upon IP addresses also called as network layer of OSI model. In traditional network router, a layer 3 device is used for routing which is known to be proprietary and costly. These devices have three planes [5].

- **Management Plane:**

  Management Plane is used to configure the switch as it contains the collection of the command line and GUI tools. This plane interacts with the control plane of the device.

- **Control Plane:**

  Control plane contains the network information and routing algorithms, its basic purpose is to decide the packet routing depending on the network information and it is hard-coded by vendors.

- **Data Plane:**

  The basic purpose of this plane is to move the packets to the desired connected port, depending on the destination.

2.1.2 Architecture of Traditional Network

Figure2.1in the following paragraph clearly explains the architecture of the traditional network, where traditional switches are used for data flow to the end-network-devices, traditional switches are then connected to the router through different links, which will further forward the data to internet cloud [4].

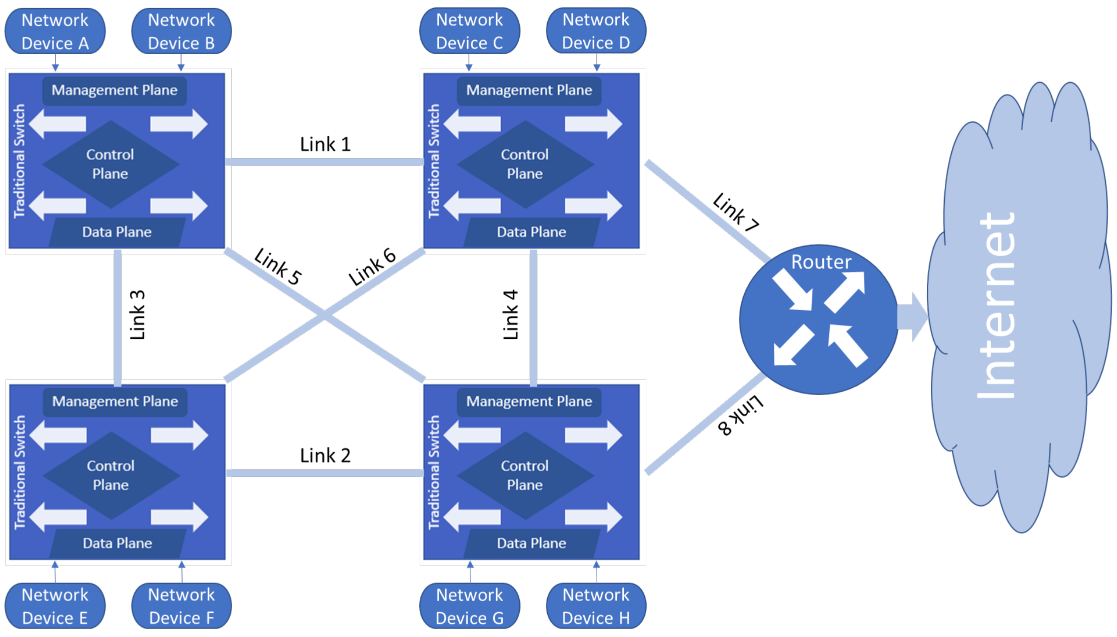


Figure 2.1: Architecture of the Traditional Network

Below mentioned Figure2.2 shows the architectural view of traditional switch where control plane and data plane are both integrated within a switch by vendor and is proprietary.

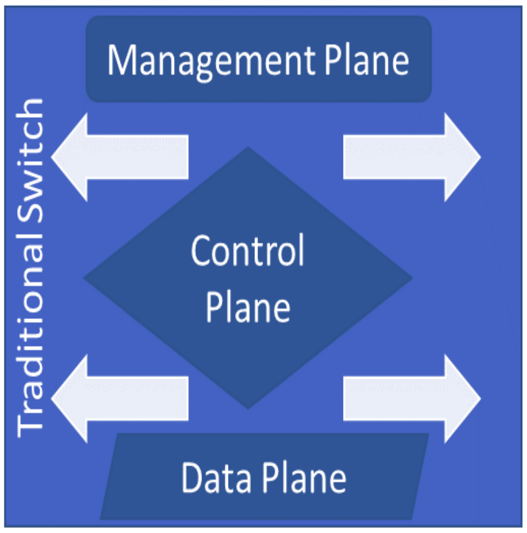


Figure 2.2: Architecture of Traditional Switch

2.2 Internet Exchange Point

Internet Exchange Point is the neutral layer 2 / Layer 3 switching point used to provide a platform to Internet Service providers and content providers of the region to share the traffic locally between them without going over the international link and hence ensuring the security of the data and saving millions in cost. IXP provides multiple benefits, some of which are discussed later on in the following sections.

2.2.1 About IXP

As internet comprises of vast networks with thousands of other individual's network interconnected with one another for traffic sharing. The IXP provides a physical platform where different networks connect with each other to exchange their traffic. To share traffic between different networks, the first step is to establish peering with other networks. The peering is established between two parties based on mutual agreement of both parties that may include sharing their respective details with each other. As a result of establishing peering between Internet service providers and content delivery networks there is no need to set up a connection and share their traffic with rest of the networks. This process saves money in a sense that this will reduce the cost of transit point and network's operation, improves system efficiency with better control, reduce latency by direct connections, and provide security of the data by keeping traffic in a local network [6].

IXP operations are managed by an operator, whose responsibility is to keep the servers up and running, maintenance of the infrastructure without interrupting the ISP's traffic. IXP operations team's major responsibility is to maintain the physical infrastructure of IXP such as cabling, website, servers and switching equipment. They also provide technical assistance on the issues regarding IXP to internet service providers.

Internet Exchange Point is installed at a neutral point to gain maximum benefits of the service. The concept of the IXP is to get rid of the monopoly of different service providers and deliver multiple types of technical support to the Internet service providers.

To promote the knowledge, development and technical assistance of IXP's in the world, multiple associations were formed for this purpose. World is divided into four different regions and the combination of these regional associations named as Internet Exchange

Point Federation (IX-F) [7].

Following are the name of IX-F members.

- African Internet Exchange (AFIX)
- Asia Pacific Internet Exchange (APIX)
- European Internet Exchange (Euro-IX)
- Latin America and Caribbean Internet Exchange (LAC-IX)

2.2.2 Existing Internet Exchange Points in the World

Currently, 956 Internet Exchange Points are installed in 159 countries of the world with 78 IXP's in the African, 201 IXP's in the Asia-Pacific, 333 IXP's in Europe, (120 Latin America + 224 North America) IXP's in the America and Caribbean Regions [8].

There are 868 Internet Exchange Points are currently in working condition in approximately 116 countries of the world. The major challenge for the IXP setup is to convince the local network providers to use the IXP platform for peering and exchange of mutual traffic. IXP requires a very minimum amount of resources to set up the physical infrastructure as maximum part of the hardware is donated by the service provider for their mutual interest.

Following is Figure 2.3 the geographical diagram of the working IXP's in the world.[9]

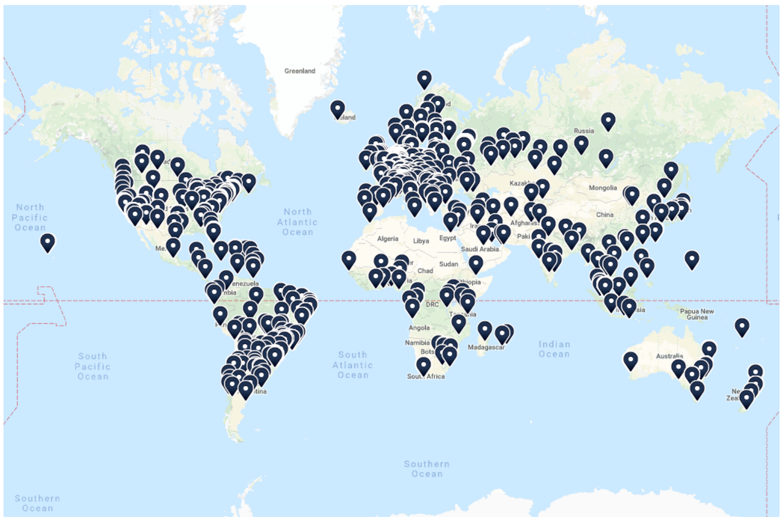


Figure 2.3: Geographical diagram of the working IXP's in the world.

2.2.3 Internet Exchange Point (IXP) Architecture

In traditional internet exchange point minimum 1 switch is required to establish a layer 2 switching fabric, where multiple Internet Service provider and local content delivery providers share their local traffic within the local infrastructure. In below mentioned Figure2.4 service provider need to share their traffic locally, where 1st four internet service provider peered with each other using a traditional switch and form an IXP switching fabric 1, while other 3 internet service provider peered with each other using another switch to form an IXP switching fabric 2. These two different switching fabrics further connected with each other through a link to exchange their local data. [10]

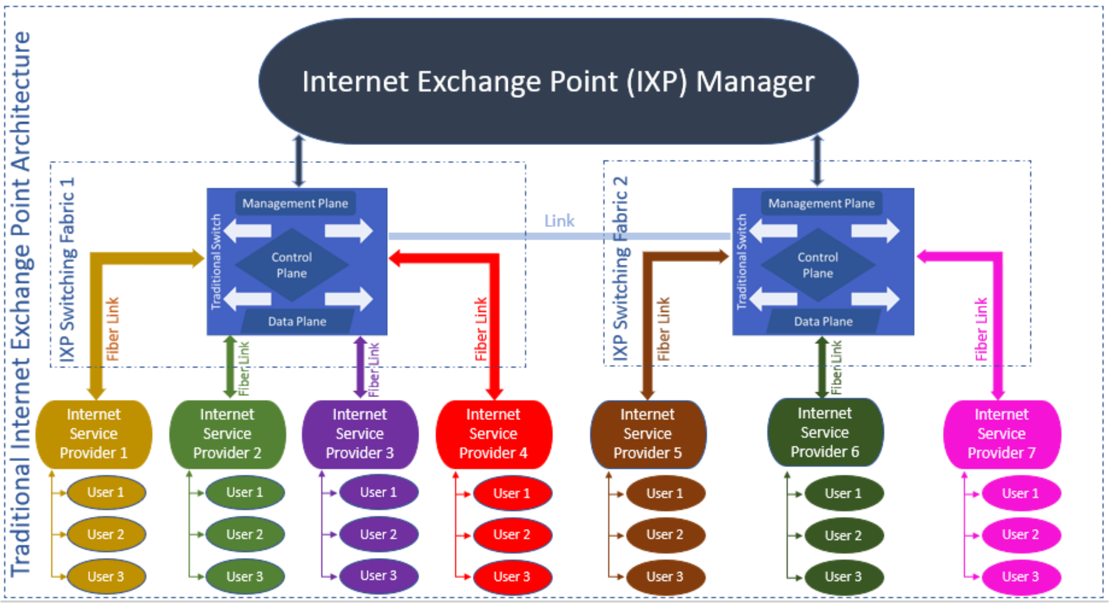

Traditional Internet Exchange Point Architecture
Internet Exchange Point (IXP) Manager
IXP Switching Fabric 1
IXP Switching Fabric 2
Link
Management Plane
Control Plane
Data Plane
Traditional Switch
Fiber Link
Internet Service Provider 1
Internet Service Provider 2
Internet Service Provider 3
Internet Service Provider 4
Internet Service Provider 5
Internet Service Provider 6
Internet Service Provider 7
User 1
User 2
User 3

Figure 2.4: Architecture of Internet Exchange Point

The basic purpose of two different IXP switching fabric creation is to improve latency and reduce operational cost for long distance connectivity. These 2 local switching fabrics interconnected with each other to share their data and work as an Internet Exchange Point (IXP). On top of these IXP Switching fabrics is the Internet Exchange Point (IXP) Manager which exists with a basic aim to manage the peering, graphical representation of traffic and administrative actions. In the IXP manager both admin and customer portal exist with different functionalities according to the requirement.

To perform a layer 3 connectivity between different ISP's, BGP protocol is used to establish peering between two different network entities. For large scale, IXP members management route server is installed, which is further connected to the IXP Manager to support IPv4, IPv6 and AS data. This will develop the trust between different ISP's and create a neutral point for different network entities to exchange their local traffic within the local network.

2.3 OpenFlow Protocol

OpenFlow Protocol was 1st developed by an open source community name Open Networking Foundation (ONF). The basic aim of ONF was to establish a set of rules for the communication between the switch and the controller. After developing the OpenFlow protocol the basic architecture was designed for network devices to communicate and provides vendor neutrality. [11] Due to the need of time, different vendors modified their hardware to make it OpenFlow compliant.

2.3.1 Overview of OpenFlow

As competition grew and the demand for OpenFlow switches increased, multiple vendors started to release OpenFlow compliant switches in the market. Based on the market need the OpenFlow community reviewed the set of rules according to the modified needs of the industry and different versions of OpenFlow were released. The initial OpenFlow version was 1.0.1 released in November 2009 and the latest public version 1.5.1 released in 2015 while immensely popular version 1.1 of OpenFlow was released by ONF in 2011. The multiple versions released by ONF on the market demand lead to new research requirement and hence it became a popular domain of research.[12]

OpenFlow is the baseline of the newly emerging domain of networking namely Software Defined Networking where the control plane is decoupled from the data plane. OpenFlow acts as an API between these two planes.

Similar southbound interface functionalities like communication between control plane and network plane is achieved by another framework namely ForCES (Forwarding and Control Element Separation) which works on the principle of forwarding using separate Logical Function Block (LFB).[13] ForCES works on the master slave model. Where the control elements are master and the forwarding elements act as a slave. A limitation of the said framework is that the control elements do not communicate with another control element. The major benefit of the OpenFlow protocol is that there is a centralized controller that is used to control all network devices in advanced network architecture. [14]

2.3.2 OpenFlow Architecture

In OpenFlow Architecture there are two major components for the said architecture to work, the first one is SDN controller, which is the decision making entity and the second one is OpenFlow switch which acts as a forwarding device and executes the decisions of SDN Controller by forwarding the packets accordingly through a secure channel. OpenFlow is considered a de-facto standard for SDN. There are multiple OpenFlow protocol versions, but in my thesis, I have worked on the OpenFlow version 1.3.

There are three different types of communication used by OpenFlow protocol.

- Controller-to-switch communication
- Asynchronous communication
- Symmetric communication

Controller to switch is the channel used by a controller to control the switch in a manner of configurations, data collection and programming. Asynchronous communication channel is used by OpenFlow switch to inform about the packet's arrival, errors and any change in the state that occurs without a request from the controller. While Symmetric communication channel is used in OpenFlow protocol to check the control channel state whether it is available or not i.e. echo messages. [15]

In a nutshell, OpenFlow Protocol works as an API to get the network information, routing information and rules etc to OpenFlow switch from SDN controller. This protocol runs between the SDN controller and Switch which in geographical language works on the south end. So, it is the southbound API of SDN environment. [16]

Following is the Figure2.5 which gives the overview of the OpenFlow architecture.

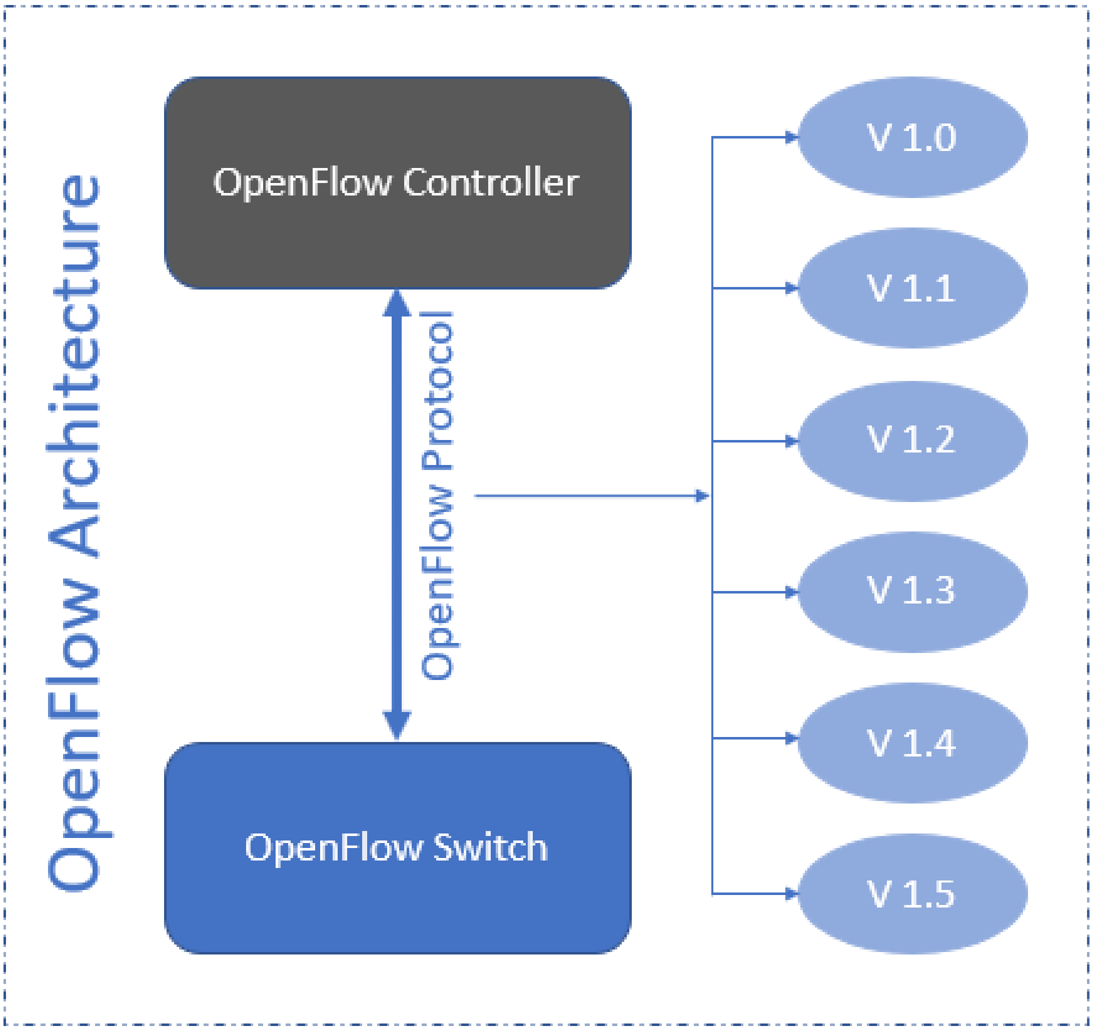


Figure 2.5: OpenFlow Architecture

In OpenFlow protocol, the information is received by the switch in the form of flow table entities. Following are the common fields of the flow table entities. [17]

- **Header Fields**

  In this field of the packet, the information of the destination and source is attached.

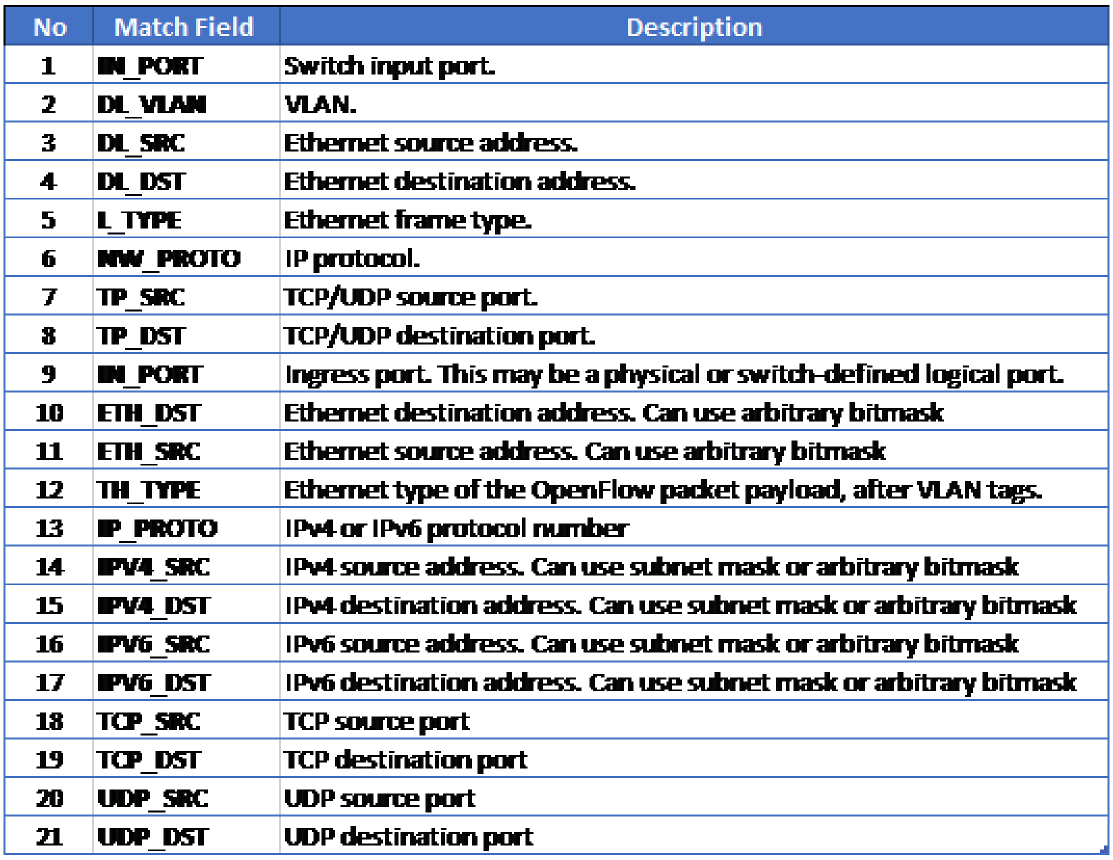
- **Actions**

  As this heading clearly explain the working phenomenon, in this field actions are taken after matching the rules defined.

- **Counters**

  This field of the packet is used for statistics collection as they update the time, size, and amount when every packet arrives.

While in OpenFlow version 1.3 some new flow entities are introduced. Which are discussed later in section 2.4.1. As OpenFlow is the set of protocols so they work on the principle of matching the fields with the packets. Following Figure 2.6 are some common fields that OpenFlow protocol use for matching defined by ONS.][18]



No	Match Field	Description
1	IN_PORT	Switch input port.
2	DL_VLAN	VLAN.
3	DL_SRC	Ethernet source address.
4	DL_DST	Ethernet destination address.
5	L_TYPE	Ethernet frame type.
6	NW_PROTO	IP protocol.
7	TP_SRC	TCP/UDP source port.
8	TP_DST	TCP/UDP destination port.
9	IN_PORT	Ingress port. This may be a physical or switch-defined logical port.
10	ETH_DST	Ethernet destination address. Can use arbitrary bitmask
11	ETH_SRC	Ethernet source address. Can use arbitrary bitmask
12	TH_TYPE	Ethernet type of the OpenFlow packet payload, after VLAN tags.
13	IP_PROTO	IPv4 or IPv6 protocol number
14	IPV4_SRC	IPv4 source address. Can use subnet mask or arbitrary bitmask
15	IPV4_DST	IPv4 destination address. Can use subnet mask or arbitrary bitmask
16	IPV6_SRC	IPv6 source address. Can use subnet mask or arbitrary bitmask
17	IPV6_DST	IPv6 destination address. Can use subnet mask or arbitrary bitmask
18	TCP_SRC	TCP source port
19	TCP_DST	TCP destination port
20	UDP_SRC	UDP source port
21	UDP_DST	UDP destination port

Figure 2.6: Common fields of OpenFlow protocol.

As above mentioned, entities are matched with packets. Decision entity in our scenario SDN controller sends a control message to OpenFlow switch to take action. Following Table 2.1 are the required action entities. [18]

Table 2.1: Required Action Entities

NO	Action Field	Description
1	**Output “port_no”**	This action field is used to forward packets to the specified port.
2	**Group “group_id ”**	This action field is used to process the packets from the group.
3	**Drop**	This action field is used to drop the packet.
4	**Set-Queue “queue_id”**	This action field is used to maintain QOS work as scheduling and forwarding the packet according to the queue.
5	**Push-Tag/Pop-Tag “ethertype”**	This action field is used to add or remove a VLAN or MPLS tag.

2.3.3 OpenFlow Rules

Rules are used in the network to control and manage the traffic. In a traditional network a commonly known word Access Control Lists (ACLs) are used in a similar way as rules. These rules are defined in the network to control the traffic and it is the basic block in OpenFlow protocol with a basic aim to match all the packet fields with the defined rules and then take appropriate action without forwarding packets.

Rules are defined in network to gain multiple advantages such as security and traffic engineering. Following are some rules used by network administrators.

- Allow
- Block
- Redirection
- Mirror

Allow is an action (match field) defined in the form of rules by the network administrator to give permission to a packet to move forward in the network. Allow-all is an exceedingly popular action field used commonly, in which all packets are allowed.

Block is a very important action used by the network administrators in their routine tasks to react in case of any security breach or any decision execution. Applying this rule, the listed packets from a dedicated port are dropped. To apply this field, a wild card of dl_type and ip_proto is required.

Redirection is used in traffic engineering as a baseline in a sense of load balancing, congestion control and port management. Redirection works in the principle of getting data from the port and then move that data to the specific port. Mirror functionality is used to get all the copied data from a specific port and then be used to apply multiple machine learning algorithms and monitoring of traffic. To apply this rule mirror port number is required.

2.4 OpenFlow Switch

OpenFlow switch act as a forwarding device in Software Defined Networking (SDN). The basic aim is to get instruction from the controller about the packet forwarding. OpenFlow switches have tables which contain the detail of flow entities depending upon the OpenFlow protocol version which they support.

2.4.1 OpenFlow Switch Architecture

Following is the Figure 2.7 that contains the architecture entities of OpenFlow switch. [18]

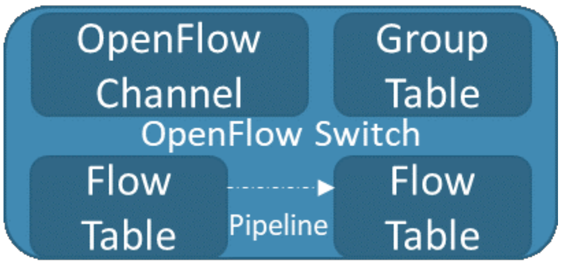


Figure 2.7: OpenFlow switch architecture Flow.

In above mentioned Figure 2.7 OpenFlow channel is a component that is used to establish a communication between OpenFlow switch and SDN controller using a secure channel known as south bound API. Flow table is used to maintain the record of all the entries of the packet, there are multiple OpenFlow tables that exist on a switch where multiple actions are performed. Depending upon the OpenFlow version, it enables every packet which contains multiple components that are further recorded in the flow tables. Following is the Figure 2.8 which shows the fields of flow entries of OpenFlow version1.3.

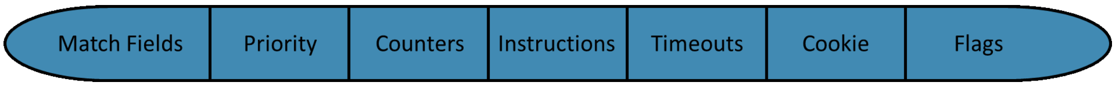


Figure 2.8: Flow entry fields of OpenFlow version 1.3.

2.4.2 OpenFlow Virtual Switch (OVS)

Open Virtual Switch (OVS) is a switching software initially developed by Nicira which was further acquired by VM Ware, is open source and available to be used publicly to perform network switching functionalities in a virtual environment. It is licensed under Apache 2.0. OpenFlow virtual switch was designed to be installed in virtual machine environments on Linux based server. [19]

OVS provides support to enable communication between multiple servers, it is capable to perform, L2 and L3 switching, ACLs and monitoring support. OVS supports multiple monitoring software and control functionalities. [19]

2.4.3 Vendor's OpenFlow supported Switch

As OpenFlow community is growing at a fast pace, there is a need for hardware companies (vendors) to upgrade their equipment to new technologies. Now, almost all vendors released their equipment with OpenFlow support in order to compete with the market. As OpenFlow Networking foundation release different versions of OpenFlow protocol, vendors add OpenFlow version support accordingly.

Following are two different modes available in OpenFlow switch.

- Standalone mode
- Secure Mode

In standalone mode when a connection with SDN controller breaks, switch works in a traditional way to process packets after attempting three times in order to establish connection with controller. While in secure mode when the controller fails to respond then the existing rules keep active rather than to expire in defined time out, while OpenFlow switch keeps on trying to establish a connection with the controller.

Following are some renowned vendor companies who released their OpenFlow switch with multiple OpenFlow versions enabled accordingly.

- Cisco (Nexus 3000 series, Nexus 3100 series, Nexus 3548 series and Nexus 6000 series etc.). [20]
- Juniper (EX4600, EX9200, MX series and QFX5100 switches). [21]
- Hp (FlexFabric 12900, HP 12500, FlexFabric 11900, HP 10500 switches and HP 2920 etc). [22]
- Arista (Arista 7150 series etc.). [23]
- IBM (IBM BNT Rack Switch G8264 etc.). [24]
- Allied Telesis (x930 series, x510 series, DC2552XS/L3, X310 series and x230 series). [25]
- Zodiac (FX OpenFlow). [26]

2.5 Software Defined Networking

Software defined networking introduced a new architecture in the domain of networking and it has opened up a new area of research in networking. Many researches are working in this domain and SDN has evolved very fast in recent past years. Software defined networking framework is used in many industries according to the infrastructure, such as Data center networking where SDN makes a perfect framework for cloud data center networking and provides flexibility and granular control using network virtualization. SDN provides central network monitoring by keeping eye on the whole network, it also provides traffic engineering using network programmability for applications like centralized device configurations, load balancing, firewall and traffic optimization in the network. SDN is the enabler of cloud-based access control and firewalls application which provides security to the application, they also provide SDN based mobility and wireless [27] such as SDN based WLAN network. [28]

2.5.1 About Software Defined Networking

SDN (Software Defined Networking) is an approach to computer networking, that allows network administrators to programmatically initialize, control, change, and manage network behavior via open interfaces and abstraction of lower-level functionality [29]. There were multiple attempts made to separate a control plane from the data plane in switches which did not succeed. In case of SDN architecture, it enables the separation of these two planes and this revolutionary change resulted in the vendor neutrality approach in SDN network.

Around 1980's an idea of physical separation of the control plane from the data plane was introduced by NCP [30], which later on allowed multiple projects to initiate such as ForCES [13] , PCE [31] and RCP [32] . These projects also worked on the same scheme for better management in MPLS, BGP and Ethernet network. McKeow in 2008, introduced the term Software Defined Networking in the research paper. [3]

There are multiple benefits of SDN that have been discussed earlier in the previous sections but the most important as per the current research is to provide a central management system to control all the switches.

2.5.2 Software Defined Networking Architecture

Software defined networking is basically a framework with a basic purpose to separate the control plane from the data plane in traditional switches and this makes it a central control system.

Following are three major components/layers in SDN architecture.

- Application Layer
- Control Layer
- Data Layer

Application layer provides programmability functionality for a programmer to design application on the top of the SDN controller and then north bound API is used as an interface for communication between these two layers. Control Layer is used as the instruction set or decision entity which runs on the top of the OpenFlow switch, in this layer SDN Controller is installed which controls the packet flows and manages the network. Data layer is used to forward the packets and works as a forwarding device where the instructions are taken from the controller using south bound API and then executes the control instructions.

Following is the Figure2.9 which clearly explains the Software Defined Networking architecture.

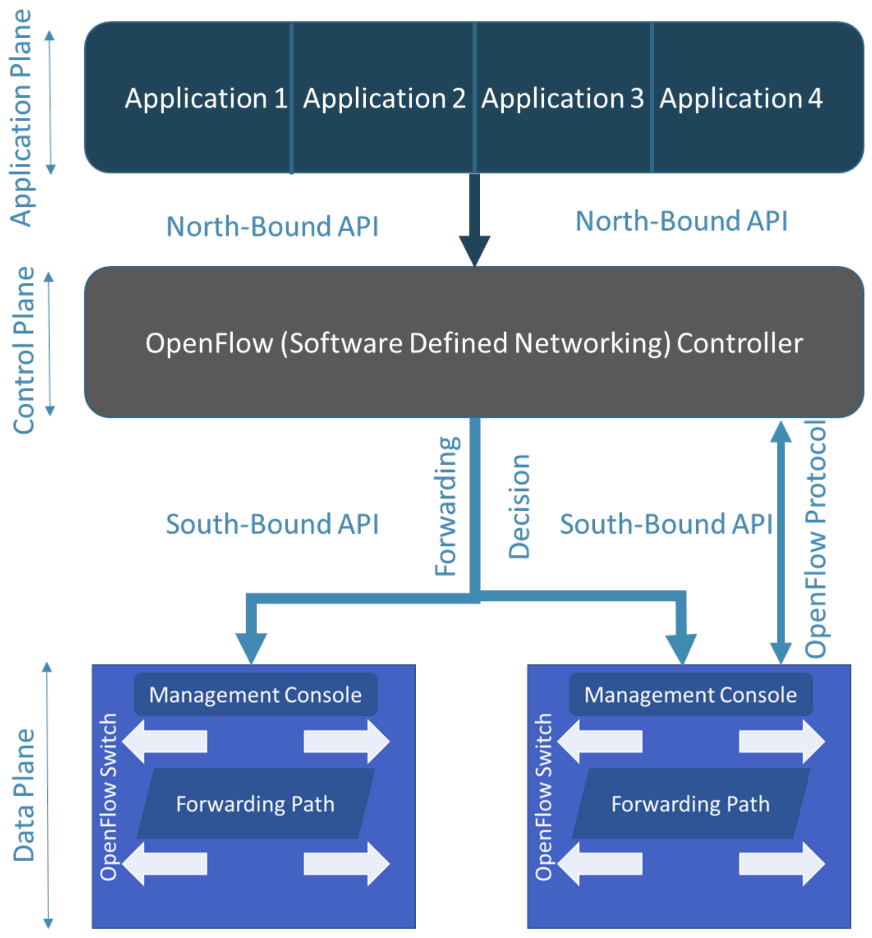


Figure 2.9: Software Defined Networking architecture.

SDN Control Plane

Control plane in SDN network works as decision entity to manage and control the switches interacting with the data plane devices using OpenFlow protocol as a southbound API. In control plane, the controller is a software written in different programming languages depending upon the ease of use, flexibility and to act as a bridge between the application layer and the data layer. Its aim is to take a decision upon the set rules and forward it to the data plane. So, it's the responsibility of the control plane to decide what to do with the network traffic as to select the traffic path, routing protocols etc.

Control plane also maintains the communication set up between the control plane and data plane by sending live messages and to get the network information from the connected devices. [33]

SDN Data Plane

Data plane functionality in software defined networking is usually performed by the switch with open flow protocol enabled. The core concept of Data Plane is to work as a packet forwarding entity without controlling functionality. The latter function is performed only by control plane of the SDN controller. But with multiple kinds of researches and optimization to improve the performance and accuracy of the system, some functionalities such as flow statistics collection make it somehow intelligence from a dump device. This approach reduces overload from the controller and improves accuracy of the controller. [34]

In Data plane every “Packet IN ”is buffered according to the rules defined in OpenFlow protocol, until the decision to forward that packet is taken by the SDN controller. When a decision is taken by the controller, as per the defined rules the switch differentiates the packet type and forward accordingly. In SDN framework, OpenFlow switch act as a data forwarding device performing data plane functionalities only.

Data Plane and control plane communication are established by a secure channel. Data plane act as a forwarding device so that the forwarding table is maintained in it. There are three major fields in the table depending upon the OpenFlow version enabled namely, matching, actions and counters, while the fields of tables increased on the later version of OpenFlow. In SDN, matching is performed on all fields of Packet header like source and destination Ip addresses, Mac addresses, Vlan ID's and ports. Action is to forward the packet to the next hop, forward packet to the controller, drop the packet, in queue and modify request. While counter is to count all the incoming packet and the then forward to get statistical results. [35]

In SDN, there are two types of network rules, one is related to data plane where mostly layer 2 forwarding is done and the second one is to send to the controller for broadcasting from the application layer, in the current scenario application layer is SDN-IXP Manager.

2.6 SDN Controller

The evolution of networking technology in the past few years gave birth to SDN, where the control plane separated by data plane. A new term of SDN Controller has been introduced recently, with the basics purpose to control the control plane functionalities and to do so, multiple APIs have been written to establish a connection between SDN applications and the SDN controller. The OpenFlow protocol is also enabled to set up communication between the SDN controller and the data forwarding devices.

2.6.1 Overview

The OpenFlow protocol was introduced in 2008 as a supporting protocol for Software Defined Networking and opened the domain for SDN controller. The Software Defined Networking provides vendors neutrality which made available an open platform for everyone to write their controller according to their recruitment. Multiple open source controllers have been written with multiple features.

As OpenFlow community is growing at a rapid pace, there is a need that existing vendors upgrade their firmware or add support of OpenFlow. Now approximately every vendor has introduced its OpenFlow hardware in addition to the SDN controllers.

As first SDN controller was designed by Nicira by the name NOX, which was later acquired by VMware. Open Daylight launched its controller back in 2013 with support for vendor switches, such as cisco. Now hundreds of controllers are written with multiple functionalities on the same principles. [36]

2.6.2 Existing SDN Controllers

The controller is the main component in Software Defined Networking. Therefore, multiple vendor companies design their own controllers to compete in the market and fulfill customer requirements. There are multiple OpenFlow controllers designed by OpenFlow communities. Following are some vendors who designed their controllers.

- Cisco [Open SDN Controller (OSC)]. [37]
- Juniper networks [Open Contrail]. [38]
- Hewlett Packard Enterprise [HPE VAN SDN Controller]. [39]
- Big Switch Networks [Big switch controller]. [40]

To meet the market requirement, multiple vendors acquired opensource products and then commercialized those after modification. Cisco is the leading market player in computer networking technologies, so they designed their own controller namely “open SDN controller ”(OSC) and then improved their functionalities in collaboration with Open Source community support such as “Open Day light ”community. Juniper also designed their own controller name “Open Contrail ”after acquiring contrail an open-source product in December 2012. [38]

Hewlett Packard is a renowned international company with a lot of products, made its own “HPE VAN SDN controller ”for its new innovative networking devices. A could and data center networking company namely “Big Switch Networks ”also developed a controller namely “Big Switch Controller ”. Similarly, new companies working on innovation and emerging technologies like pica 8, Nauga network, cumulus networks and Pluribus Networks also contributed to support OpenFlow products. [41] [42] [43]

Following are some SDN Controller developed by open source community.

- Open Daylight
- Flood Light
- Ryu Controller
- ONOS

- Beacon
- Faucet Controller

A well-known SDN controller “Open Daylight ”has many contributions in uplifting of the SDN technologies, many renowned vendor companies use their support for their open source products, such as cisco. It is also being used as an industry standard for many new OpenFlow controllers and is considered to be as most advanced SDN controller. Until now, 10 releases of Open Daylight are available as open source. [44]

Floodlight controller also has contributed in the community with multiple supports and up gradation. Currently, they have announced their new version v1.2 with new features and support. This controller is also supported by vendor companies such as “Big Switch Networks ”. [45]

Ryu Controller is also a widely used controller with a lot of application's support and very developer friendly, many developers use this as a baseline to develop their own controller and to merge their work with a controller. Their code is easily available at GitHub and is licensed under “Apache 2.0 ”. [46]

“Open Networking Operating system ”(ONOS) is also an open source SDN Controller develop by open source community major aims to support NFV services and applications support of performance monitoring for operators in the telecom sector. There are a lot of similarities of ONOS with Open Daylight controller which is one of the most renowned controllers in the domain of SDN. ONOS is developed very recently in late 2014. [47]

Beacon SDN controller is a relatively old controller with the basic aim to focus on providing support to research and academia. Many programmers use their logical base concept to develop the most powerful controller with multiple features like in Floodlight SDN controller. Beacon controller is well recognized because it is considered to be a pioneer in SDN Controller. [48]

Faucet SDN controller is considered to be emerging in the domain of Software Defined Networking with the advantage of being deployed in sizeable network topologies. Faucet SDN controller's 1st version was released in back 2015. The further details of faucet SDN controller are discussed in the next section. [49]

2.7 Faucet SDN Controller

Faucet SDN controller is considered to be emerging in the domain of Software Defined Networking with the advantage of being deployed in sizeable network topologies. Faucet controller was designed to work in an efficient way, the controller code is just 1000 lines and written in python language which is much user friendly as compared with other coding languages. The working phenomenon of faucet is much comfortable for the network administrator as it works in the same way as server clustering does. The major benefit of SDN controllers is that it is independent of vendor functionalities, faucet work perfectly in this concept.

2.7.1 About

Faucet was developed by Josh bailey (from Google), Kit Lorier (From REANNZ Networks) and Brad Cowie (From Waikato University). The 1st version of faucet was created in 2015 in New Zealand and due to its open source availability at GitHub, SDN community has so much contribution and support for the development and growth of faucet. Later in Sep, 2016 faucet launched its v1.1 and v1.2 in Oct 2016 with multiple releases, updates support and the current faucet has v1.9.9.

Faucet controller has an advantage in the practical deployments in most of the renowned places and it is evolving day by day. The faucet controller has also been deployed in Toulouse, France IXP and became a pioneer in Europe for converting existing IXP from traditional to SDN enabled. Currently, faucet production deployment is in many renowned places such as Open Networking Foundation, REANNZ, GEANT and AARNET etc. [50]

2.7.2 Introduction

Faucet is a Software Defined Controller with the basic aim to provide control functionalities of the SDN environment. Faucet is 1.3 Version OpenFlow hardware compliant. In networking, the monitoring flows are the major load on a network that affects the efficiency of the overall system. To overcome this challenge faucet has been technically divided into two sub-controllers.

- Gauge Controller
- Faucet Controller

Gauge controller is specifically assigned for monitoring and collection of flow counters and monitor the ports statistics through an OpenFlow link between switch and controller. Then its task is to export it into the database (Prometheus/influx DB) without affecting the state of the OpenFlow switch. [51]

Faucet Controller has performed control functionalities such as forwarding decisions, Rules and establishment of peering between different networking devices etc. It was tested on different OpenFlow compliant switches.

As Faucet works on the principle of vendor neutrality of OpenFlow protocol, faucet performs its control functionalities through an independent server-based software. Faucet has an edge that it works without an application on the application layer as it is controlled by YAML file where peering information and rules are fed into controller. [51]

2.7.3 Architecture

Following is the Figure2.10 that completely explain the architectural behavior of the faucet controller.

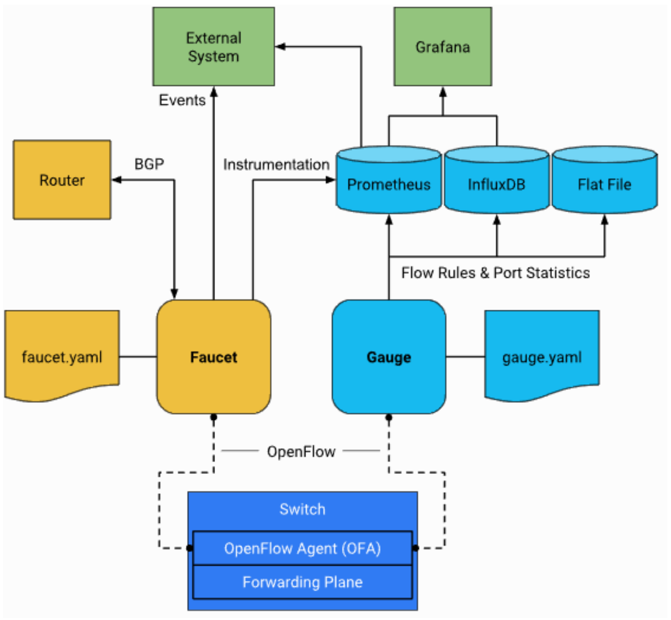


Figure 2.10: Faucet SDN Controller Architecture.

In the above Figure 2.10 [52] both faucet and gauge controllers are directly connected to the OpenFlow compliant switch, while port statistics of the gauge controller are saved in the time series database from where the statistical data is shown in the graphical dashboard (Grafana). Whereas, the faucet controller gets peering information and rules from the YAML file to control and manage the network behavior. [52]

2.7.4 Pipeline Faucet

Faucet controller works on the principle of OpenFlow protocol, where a controller controls the OpenFlow compliant switch through a specific set of instructions commonly known as pipeline. There are ten tables in facet pipeline where some tables are eliminated depending upon the topology of the network.

Following is the Figure2.11 which clearly explain the faucet pipeline. [52]

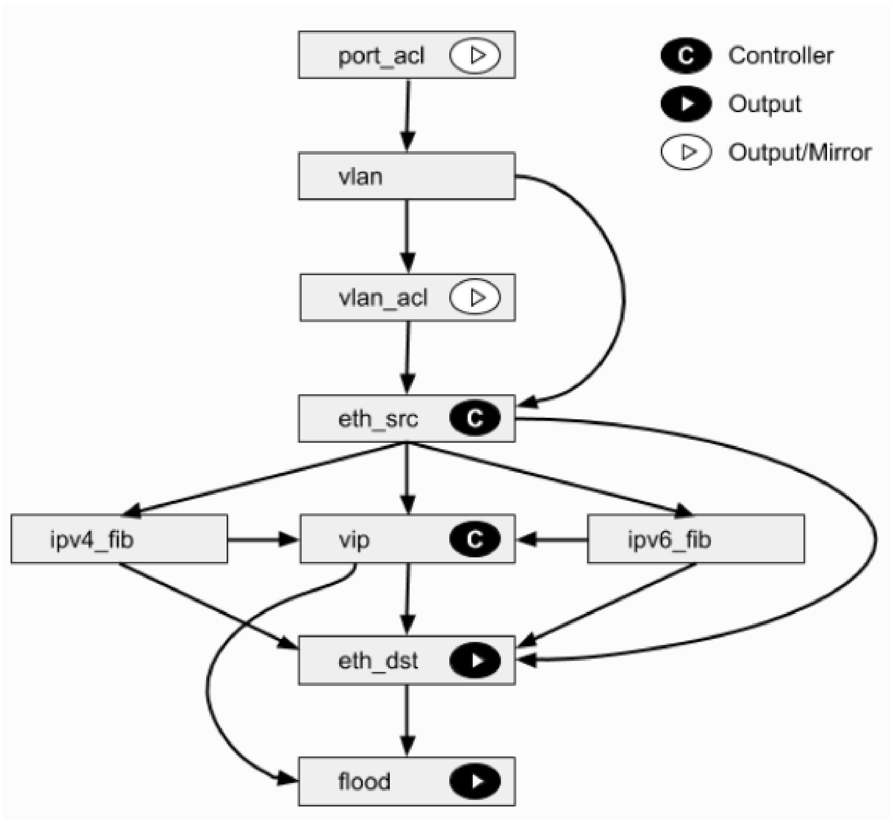


Figure 2.11: Faucet pipeline

In above Figured 2.11 “Port_Acl”table is the 1st table in faucet pipeline used for the implementation of rules defined at controller (such as block, allow and mirror etc.) for OpenFlow switch. 2nd table in faucet pipeline is “VLAN”table which performs matching operations for both tagged and untagged ports. “eth_src”and “eth _dst”table in faucet pipeline uses to match the source and destination field to move the packets to correct destinations. 5th field of the faucet pipeline “vip”is used for matching operations of the ipv4 and ipv6. Last table of the pipeline “flood”is used for broadcasting and multi casting.

In Figure 2.11 above, there are 9 tables in the pipeline whereas, there is one additional table named “Eth_Dst _HAIRPIN”is used, where a communication between different vlans is required while the port is same in case of routing between hosts. [52]

2.8 TouSix Manager

TouSix manager is a web-based platform used for controlling and monitoring of the SDN enabled Internet Exchange Point. The main aim for designing TouSIX manager was to manage the Toluse SDN-IXP control functionalities.

2.8.1 About

TouSix Manager is a combination of two words “Toulouse”came out from the name of the city of France and the second one is “SIX”short form of SDN internet exchange point. TouSix manager code is written in python.

TouSix manager has statistical collector to show the traffic of the SDN enabled internet exchange point and there is a direct communication with the OpenFlow controller (Ryu controller) to pass down the OpenFlow information.

Design of the TouSIX manager is specifically to monitor or maintain the Toulouse SDN-Internet Exchange point and it also provided the ease by graphical web interface to integrate new services.

2.8.2 Introduction

The TouSix Manager is the pioneer in the domain of SDN-IXP management software. Before the development SDN IXP Manager the traditional IXP manager software was designed to monitor and provide services to traditional IXPs. TouSix manager was developed on the top of the SDN controller, especially for the RYU controller to provide GUI based control and monitoring of Toulouse Internet Exchange Point. [53]

In TouSiX manager there are following major functionalities introduced to monitor and maintain the SDN Internet Exchange Point.

- They provide an OpenFlow operations using graphical-user-interface based instructions.
- Monitor the SDN traffic of IXP.
- BGP Configuration generator.
- Rules generator.

2.8.3 Architecture

Following is the Figure2.12 which shows the architecture of TouSix Manager.

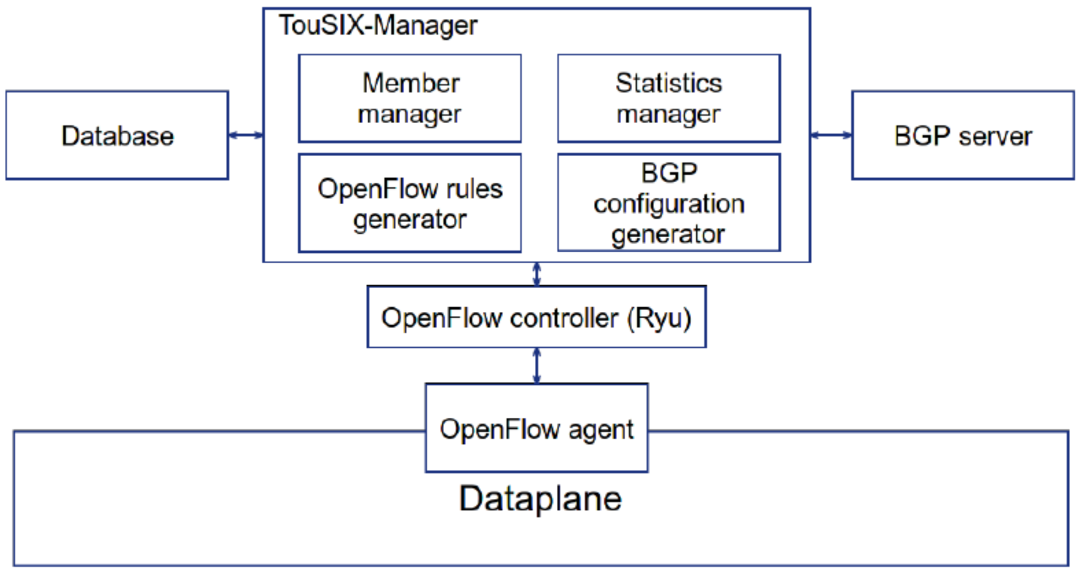


Figure 2.12: TouSix Manager Architecture

In above Figure2.12 TouSix manager has 4 major components namely.

- Member Management
- Statistical Manager
- OpenFlow Rules Generator
- BGP Configuration Generator

There is a separate database for storing rules of OpenFlow and a database for website management data.

The 1ST component of the TouSix manager is used to manage the administration's end functionalities and management. The 2nd component of TouSix manager is used for statistical data of the IXP to be monitored where different ISPs have peered together and their traffic is exchanged at that point. The 3rd component of TouSix manager is to generate OpenFlow rules and the 4th component of the TouSix manager is used for establishing a communication between different ISPs as they need a BGP protocol to work and configure. [54]

2.8.4 Limitations

Following are some major limitation in TouSix Manager Verses IXP Manager.

- TouSix Manager is a python-based application and there is an integration of Django framework to perform some functions, while IXP manager is a full stack management system for ISP's monitoring.
- TouSix manager includes only an administration portal for ISP's management while IXP includes both administration and customer portal for ISP's management.
- TouSix manager is designed to control the Toulouse IXP, while IXP manager is designed in general as an IXP management tool. Whereas, 76 IXPs around the world are using this technology and have strong customer support.
- TouSix manager is limited tool for the ease of management operations while IXP Manager has a wide range of supporting tools and APIs.

Chapter 3

Proposed Architecture

In this chapter, the proposed architecture/schema of the complete system has been discussed.

3.1 About

The proposed scheme in general, will be used in telecom sector and specifically at the Internet Exchange Point (IXP), where different ISPs peer with each other and share local traffic/content at the IXP without routing it through the international gateway. The international gateway traffic is not free of cost, by peering at an IXP the ISPs save money and address privacy concerns of the local/national traffic by routing it within the country only. The question arises that how the proposed scheme can be helpful in Internet exchange point (IXP) scenario.

The traditional Internet Exchange Point lacks support of advanced features, as discussed above. By transforming, traditional IXP into SDN enabled IXP the feature of monitoring and control has been introduced in it. That is why the proposed schema is especially important to provide monitoring and control functionality to SDN IXP, which will help SDN IXP users to monitor their traffic stats and control the traffic by defining rules to redirect the path or block the traffic in case of security breach at top of the SDN controller.

Similarly, the proposed scheme also provides an ease of use and flexibility to IXP management team to view stats of the traffic flowing through the IXP. It also helps to define rules to control traffic at single platform that is graphical user interface (user friendly) without going down to the controller's complex yaml file. Furthermore, proposed scheme also allows writing the rules manually one by one, which once cause different errors such as indentation, syntax, mismatch etc.

The proposed architecture consists of the following five components.

- Statistical Dashboard Linkage
- Vlan Selection
- OpenFlow Switch Addition
- Interface addition
- Rules Generation

The first component of the proposed architecture is the Statistical Dashboard Linkage at SDN-IXP Manager for the purpose of monitoring and statistical view of traffic. The Second component is a Vlan selection that is used to select Vlan for establishing communication between peered Autonomous Systems (ASs). The Third component of the proposed scheme is OpenFlow Virtual Switch (OVS) addition, whereas OVS is used to provide a switching mechanism with the support of computer communication protocols. The Fourth component of the proposed schema is Interface addition, which is used to establish peering between ISPs on top of the OpenFlow switch at SDN controller through the proposed schema. Fifth component of the proposed scheme is to generate rules to provide security at SDN IXP manager by blocking or allowing specific traffic. The security provision can also be applied to servers in case of security breach. This component is also used for static load balancing, i.e. to divide load between ports by redirection in case there is a traffic overload. It can also be used to mirror traffic for running data processing algorithms, such as machine learning or artificial intelligence. In this proposed scheme there is no need of a separate database for storing rules as in case of TouSix manager, SDN-IXP manager push rules to the Yaml file of the OpenFlow controller (Faucet) by avoiding controller framework API dependency.

The proposed architecture of the system is given in Figure 3.1.

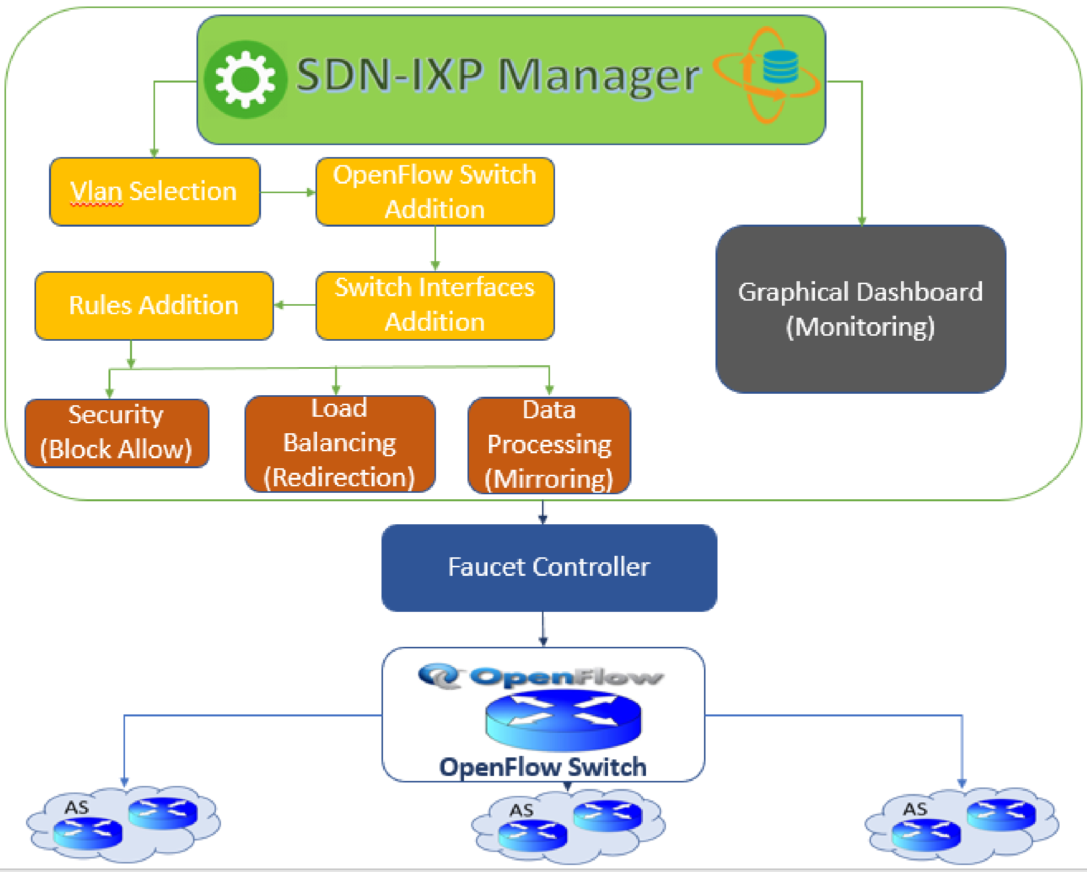


Figure 3.1: Proposed Architecture of Monitoring and control of SDN IXP Manager

3.2 Used Tools

The research and development of proposed architecture has been performed in PHP Laravel Framework 5.5. Whereas, early IXP manager used PHP Zend Framework components. In other words, it is a blend of both the frameworks. For database related operations, Laravel Doctrine Framework has been used. The implementation of the project required use of multiple Open source libraries and modules. A few of the important components used in this project are listed below.

- Virtual Machine Virtual-Box
- Faucet Controller
- Gauge Controller
- Grafana Server
- Prometheus Data Base

- IXP Manager
- Quagga Software Routing Suite
- Open Virtual Switch
- Iperf Network Performance Measurement

3.2.1 Virtual Machine Virtual Box

A Virtual Machine is an emulation software that is used to emulate a computer system. The functionalities are dependent on the concept of computer architecture. VM requires resources of a real machine to setup an operating system and multiple of these operating systems can be created on a single physical machine, isolated from each other, which provides flexibility and reduces the cost of new hardware. By using tools like virtual machine, different Autonomous systems, SDN controller (at the virtual server) and web-based application (IXP) server have been created on a single machine.

3.2.2 Faucet SDN Controller

Faucet is an Open source controller, it was basically designed to control the SDN control plane functionalities without the vendor dependencies, as an independent software. Faucet controller is used to control the routing, rules, and then pushing down the instructions to the switch using OpenFlow protocol 1.3.

3.2.3 Gauge SDN Controller

Gauge Controller is fundamentally a part of the faucet open source controller but work as a separate controller than faucet controller. The faucet controller is used to control the SDN control plane functionality whereas, the gauge controller is only used to collect the statistics of traffic data without compromising the faucet controller to balance the load between control and monitoring.

3.2.4 Grafana Server

Grafana dashboard is a graphical dashboard that is used to display statistics of traffic for monitoring and analysis of the data. It is a separate window that does not affect control functionalities.

3.2.5 Prometheus Database

Prometheus is an efficient time series database that is used to store statistics of the collected data for further analysis. Prometheus is further linked to Grafana dashboard for a graphical view of the data.

3.2.6 IXP Manager

IXP Manager is an internationally used management system used at Internet Exchange Point to monitor, control and manage the traffic. It has a customer portal for customer side view, administration portal for administration side view and provides end to end provisioning. It is a web-based application for monitoring and management.

3.2.7 Quagga Routing Suite

A Quagga is a software that acts as a routing suite and basic purpose of its development is to implement routing algorithms like BGP, OSPF and RIP etc in Linux operating systems. There are multiple components in Quagga architecture, but in the proposed architecture, there are two main components used namely, Zebra and Bgpd. The reason quagga has been used in the project, is to simulate physical AS communication through physical OpenFlow switch.

3.2.8 Open Virtual Switch

Open Virtual switch is software-based virtual switch that is installed on a system to provide functionalities of multilayer switch virtually without having a physical switch. It supports all computer network protocols and standards for network automation.

3.2.9 Iperf Network Performance Measurement

Iperf is a tool that is used by a network administrators or researchers to test the performance of the network. It has primarily been used to test in the said scenario to check the maximum amount of bandwidth that can be achieved during the communication.

Chapter 4

Design and Implementation

This chapter briefly explains the design and implementation scheme of all the components, which are used in my architecture.

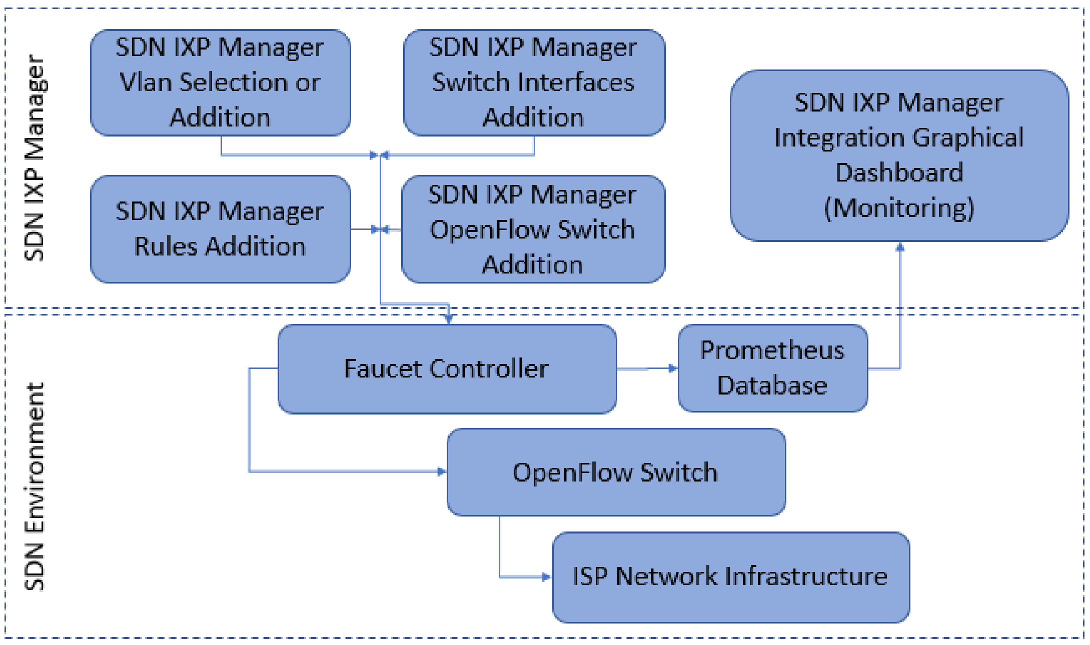

SDN IXP Manager
SDN IXP Manager Vlan Selection or Addition
SDN IXP Manager Switch Interfaces Addition
SDN IXP Manager Integration Graphical Dashboard (Monitoring)
SDN IXP Manager Rules Addition
SDN IXP Manager OpenFlow Switch Addition
SDN Environment
Faucet Controller
Prometheus Database
OpenFlow Switch
ISP Network Infrastructure

Figure 4.1: Components Descriptions for SDN IXP Manager Monitoring and Control.

The above mentioned Figure 4.1 illustrates different blocks integrated in a scheme to exhibit a mechanism of a) monitoring the traffic in OpenFlow scenario using SDN controller b) establish a peering between different ISPs c) Rules addition d) OpenFlow switch addition e) VLAN addition in SDN-IXP scenario using the redesigned SDN-IXP Manager.

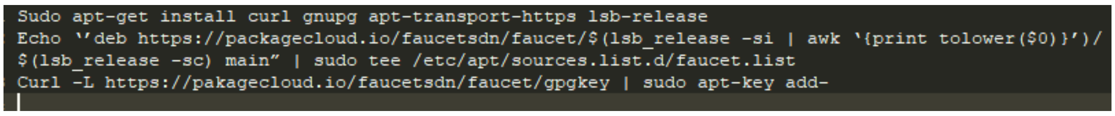
Following is the description of the above-mentioned block with configuration, integration and coding. [55]

4.1 Setup and Configure SDN environment server

This portion aims to setup and configure the SDN OpenFlow controller (Faucet and Gauge), install the required tools and then configure all the tools like OpenFlow Virtual Switch, Prometheus database and IXP Manager Software, in order to ensure that SDN environment has been established. Furthermore, it will be modified according to the proposed scheme.

4.1.1 Setup and configure SDN OpenFlow Controller (Faucet and Gauge)

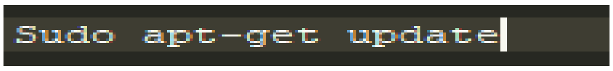
Initially, the faucet repository will be added with all the components linked to the server from which the research work will be conducted. The following Figure 4.2 show the commands that are used to fetch the target repository.



```
Sudo apt-get install curl gnupg apt-transport-https lsb-release
Echo ''deb https://packagecloud.io/faucetsdn/faucet/$(lsb_release -si | awk '{print tolower($0)}')/
$(lsb_release -sc) main" | sudo tee /etc/apt/sources.list.d/faucet.list
Curl -L https://pakagecloud.io/faucetsdn/faucet/gpgkey | sudo apt-key add-
```

Figure 4.2: Commands used to fetch the target repository.

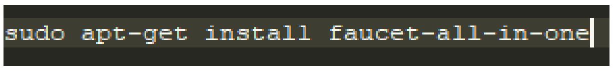
Following Command shown in Figure 4.3 use to Update the system where faucet repository added.



```
Sudo apt-get update
```

Figure 4.3: Command used to Update.

In order to get the benefit and rectify all the dependencies of the faucet and its component, the following command shown in Figure 4.4 is used to install the faucet package with all its components.



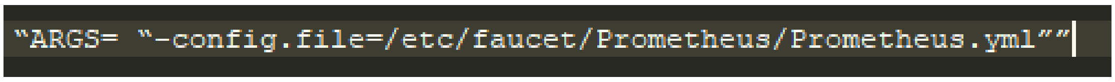
```
sudo apt-get install faucet-all-in-one
```

Figure 4.4: Command used to install faucet package.

The above-mentioned command shown in Figure 4.4 is also used to setup faucet and gauge controller with its components. Faucet and gauge initially come with ***faucet.yml*** and ***gauge.yml*** containing default configuration to setup the controller. As far as gauge controller is concerned there is no need to change the configuration but faucet YAML file change is required according to the topology via IXP-Manager.

4.1.2 Setup and configure Time Series Data Base (Prometheus)

The Faucet and gauge controllers need a time series database to store metric data. While adding faucet repository, it also gets the sample template of Prometheus with it. The sample template requires modifications as per the user requirement. It is important to set the receiving path shown in Figure 4.5 whereas the other configuration needs no change.



```
"ARGS= "-config.file=/etc/faucet/Prometheus/Prometheus.yml""
```

Figure 4.5: Setup receiving path.

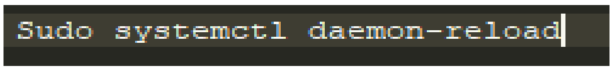
After setting up the above-mentioned argument in Prometheus database, the file needs to be restarted in order to ensure that the changes have been successfully applied.

4.1.3 Setup and configure Statistical Dashboard (Grafana)

Grafana is an open source platform to view statistical results with integrated support for SDN controller. In order to enable the support for SDN Controller there is a need to configure Grafana. As stated earlier, the installation of faucet controller repository downloads all the components with itself. The only need at this stage is to enable the Grafana server, re-load the daemon and start the Grafana server using ubuntu command line.

Following are the commands shown in Figure 4.6, 4.7 and 4.8 used to setup the Grafana statistical dashboard.

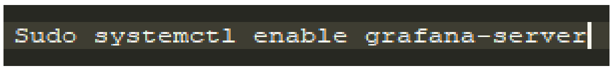
To start the services, first re-load the Daemon using following command shown in Figure 4.6.



```
Sudo systemctl daemon-reload
```

Figure 4.6: Commands used to reload Daemon.

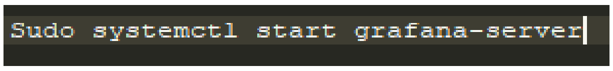
To Enable the Grafana Server use the following command shown in Figure 4.7.



```
Sudo systemctl enable grafana-server
```

Figure 4.7: Commands used to enable Grafana server.

To start the Grafana Server use following command shown in Figure 4.8.



```
Sudo systemctl start grafana-server
```

Figure 4.8: Commands used to start Grafana server.

Using above mentioned commands grafana server is set up, it is accessible from the web interface local host with **port number 3000**.

Secondly, the most important thing is to set grafana web application as it gets the data from the time series database (Prometheus). For the same, set the name, type and url of Prometheus in grafana to view the stats, the Prometheus database is accessible from **localhost:9090**.

For SDN controller, there are a few dashboards developed by open source community in which grafana can be integrated to view the results in form of ports stats, inventory detail and instrument information with a separate dashboard for each.

After successful set up of the Grafana dashboard, it takes the shape as following shown in Figure 4.9

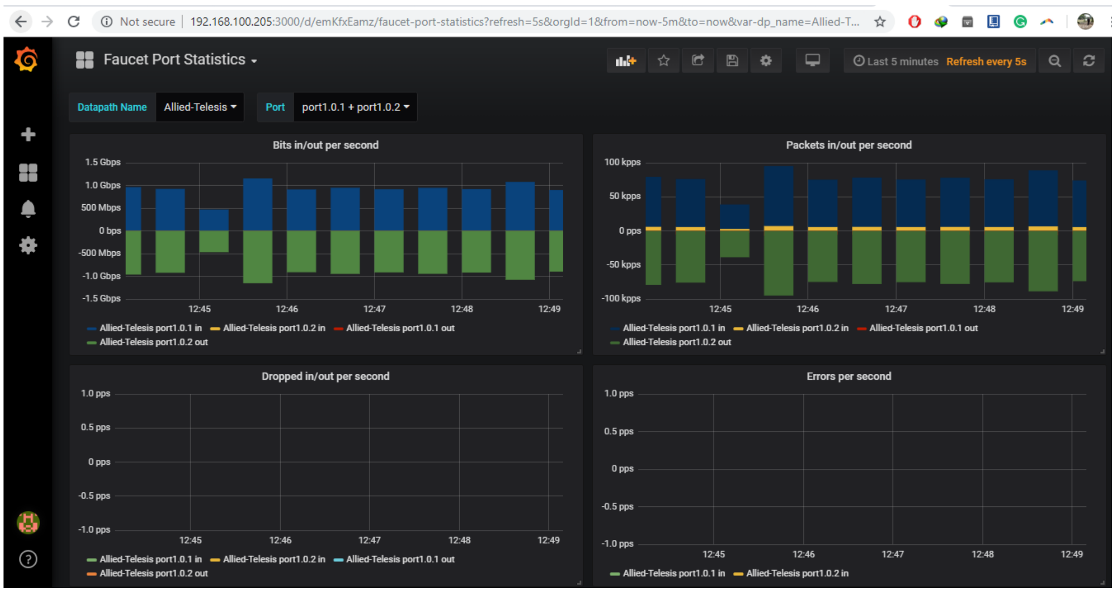

Not secure | 192.168.100.205:3000/d/emKfxEamz/faucet-port-statistics?refresh=5s&orgId=1&from=now-5m&to=now&var-dp_name=Allied-T...
Faucet Port Statistics
Last 5 minutes Refresh every 5s
Datapath Name
Allied-Telesis
Port
port1.0.1 + port1.0.2
Bits in/out per second
Packets in/out per second
Dropped in/out per second
Errors per second
12:45
12:46
12:47
12:48
12:49
Allied-Telesis port1.0.1 in
Allied-Telesis port1.0.2 in
Allied-Telesis port1.0.1 out
Allied-Telesis port1.0.2 out

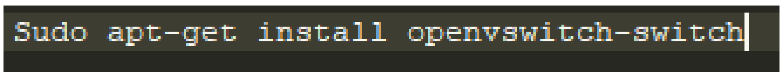
Figure 4.9: Statistical view of Grafana Dashboard

4.1.4 Setup and Configure OpenFlow Virtual Switch (OVS)

In order to perform testing of SDN controller with SDN_IXP manager the OpenFlow enabled switch is required for switching purpose and to act as a dump switch only, used for data forwarding and matching without taking decisions. For the same reason, an OpenFlow Virtual Switch is required to be configured and setup on the server to perform the above-mentioned activities in a virtual environment.

Following are the steps involved to setup an OpenFlow switch environment on a hosted server.

- To install OpenFlow switch on a desired server, the following command is used as shown in Figure 4.10.



```
Sudo apt-get install openvswitch-switch
```

Figure 4.10: Command to install Open Virtual Switch

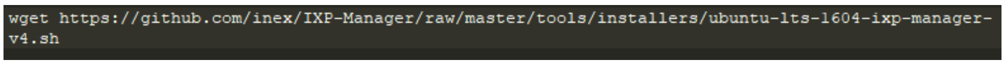
- Configure the OpenFlow Virtual switch. The configuration of an OVS includes adding a bridge with ports integration, setting up the Data Path ID, setting up the internal configuration (fail mode enable etc.) of OpenFlow switch, setting up the interfaces of the added port(s) and controller listening ports in order to integrate OpenFlow Virtual switch to the SDN controller.

4.1.5 Configuration of IXP Management Software

To setup the internet exchange point, at first there is a need to setup the IXP management software, which is used by 70 IXP's globally. As mentioned earlier the existing IXP only offers traditional network control. The existing networks evolved with the advancement in technology and the need for redesigning of the architecture surfaced. With the redesign, the existing users will also have to migrate to a newer technology in order to avail all the benefits of the new technology.

Following are the steps required to install IXP Manager for further configuration which will download the script and install it.

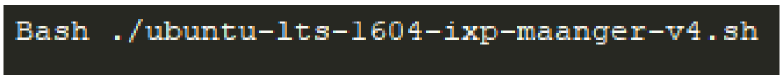
- To download the IXP Manager, use the following command as shown in Figure 4.11.



```
wget https://github.com/inex/IXP-Manager/raw/master/tools/installers/ubuntu-lts-1604-ixp-manager-
v4.sh
```

Figure 4.11: Command to download IXP-Manager.

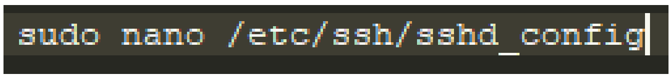
- To execute the installation shell of IXP manager, use the following command as shown in Figure 4.12



```
Bash ./ubuntu-lts-1604-ixp-maanger-v4.sh
```

Figure 4.12: Command to execute the installation shell.

- After executing this command IXP Manager will install all the tools (i.e. MySQL, Laravel etc.) that are required to run the IXP Manager. During Installation username and password should be set while a random root password is generated which will be saved and used as required.
- For the implementation of IXP Manager SSH following command as shown in Figure 4.13 is used to edit the configuration file.



```
sudo nano /etc/ssh/sshd_config
```

Figure 4.13: Command to edit the configuration shell.

- To enable the SSH open the editable file and set the permit root login and strict mode to “yes”.

4.2 Mechanism to link Statistical Dashboard

The major outline of this research is monitoring of network/IXP administration and network traffic i.e. data (packet and bits) inflow and outflow from each port of the switch, packet drop rate, errors per second, controller information (CPU usage, memory, cold restart etc.) and Controller inventory information. To view all the afore-mentioned information, separate statistical collectors are already built by the OpenFlow community. As mentioned earlier, the basic goal of this research is to develop a dashboard for SDN-IXP Manager so that a user can view the stats. For that purpose, a separate statistical collector is required to be set up and integrate it with the IXP Manager and redirect it to the configured collectors for visualization of the stats for SDN-IXP Manager user.

4.2.1 Mechanism for redirection in SDN IXP Manager

In order to access grafana via SDN-IXP Manager, the front end of the existing IXP Manager is needed to be redesigned by adding a new button linked to grafana dashboard. The admin or customer can navigate to the statistical portal to monitor the traffic and controller information by clicking the recently added grafana button in the IXP Manager. For the said purpose, following effort is required in the existing IXP Manager:

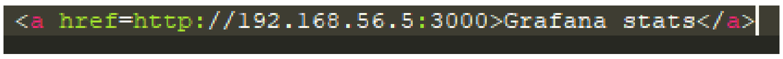
To add the button, identify the file from which front end modification will be performed. Following is the location of the files in which modifications are required.

“***Serv/ixpmanager/resources/views***”

Following is the file name in which button for grafana redirection is added.

“***menu.foil.php***”

To redirect to grafana from server where the SDN scenario is running, following link is used as shown in Figue 4.14, which contains a tag name with properties added.



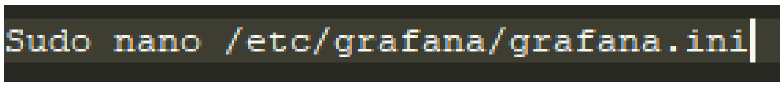
```
<a href=http://192.168.56.5:3000>Grafana stats</a>
```

Figure 4.14: Redirect Link.

After executing the above command the link is redirected to grafana, but there is a need to remove authentication from it. For the said purpose there is some modification required in grafana file. Following is the location where the modification is performed.

"***/etc/grafana/grafana.ini***"

To edit the grafana.ini file use the following command as shown in Figure 4.15



```
Sudo nano /etc/grafana/grafana.ini
```

Figure 4.15: Command used to edit the Grafana file.

Some modifications required to remove authentication are:

- Disable the login form by changing the field from false to true.
- Enable anonymous access by changing the field from false to true.
- Assigning name to the field for which authorization is required by entering the "org_name"and "org_role".

4.3 Mechanism to Establish Peering and Rules Generation

The proposed architecture of IXP Manager requires establishing peering and rules generation. In order to push the peering information and rules to the controller, the front end of SDN-IXP Manager has been redesigned. To make it possible, a YAML file is created from SDN-IXP Manager with rule and peering information, which will be further used by the faucet controller to implement the desired actions. In this way of redesigning of IXP Manager's front end, an initial page containing blocks is created. The code is written in the file named **generateyaml.foil.php**.

Following blocks are generated through SDN-IXP Manager.

- Add Vlan information or Select the existing Vlan.
- Add OpenFlow Switch.
- Add or select the interfaces of the switch.
- Add or select the rules.

Above mentioned blocks are implemented in an order such that 1st step is to select vlans, 2nd step is to add OpenFLow switch details, 3rd step is to add OpenFlow switch interfaces and 4th step is the rules generation (namely ACLs). The details of aforementioned steps, coding and configuration are discussed below.

4.3.1 Mechanism to add VLan Addition

Following information is required from the user end to add Vlan information through SDN-IXP Manager.

- Vlan Name
- Vlan ID (vid)
- Vlan Description

To get above mentioned data in SDN-IXP Manager, code modifications are done in “**generateyaml.foil.php** ”file.

4.3.2 Mechanism to Add OpenFlow switch

Following information is required from user end to add OpenFlow switch information through SDN-IXP Manager.

- OpenFlow Switch Name.
- OpenFlow Datapath ID (dp_id).
- OpenFlow Hardware Name (OpenFlow Switch Name by manufacturer).

To get above mentioned data in SDN-IXP Manager platform, following are the code modifications done in “**generateyaml.foil.php** ”file of SDN-IXP Manager.

4.3.3 Mechanism to add OpenFlow switch interfaces information

In order to establish peering between different ISPs, the prerequisite is to add OpenFlow switch interfaces information in SDN controller, which in this case is through SDN-IXP Manager. To make necessary changes in it the following information is required from user end.

- Interface Name
- Interface Description
- Native Vlan (native_vlan)

Above-mentioned modifications are done in “**generateyaml.foil.php** ”file of SDN-IXP Manager.

4.3.4 Mechanism to add Rules Generation

The rules definition in a network are the most important task for a Network Engineer, as these rules are required to prevent security breach, multiple cyber-attacks, and redirection of data etc. Following are the common rules that are traditionally used by network engineers and the same are used in this research to be integrated in SDN-IXP Manager.

- Block Ports
- Mirror Ports
- Redirection
- Allow All

Following parameters are required from user end to add Rules or Access Control List (ACL) in interface block through SDN-IXP Manager.

- Destination Mac (Match Field=dl_type)
- Ethernet Type Protocol (Match Field=ip_proto)
- Action (Match Field= allow (True/ False) / output)
- Mirror Port Number
- Destination Ethernet mac address of port from whom packet is monitored (Match Field=dl_dst)
- Destination Ethernet mac address of packet (Match Field=dl_dst)

Above-mentioned modifications are done in “**generateyaml.foil.php** ”to generate rules in SDN-IXP Manager.

Chapter 5

Results and Discussion

This chapter is used to discuss results and observation of the proposed architecture.

5.1 SDN-IXP Testing Equipment

For testing the proposed schema, Following are the hardware, software and tools used.

5.1.1 Server (Host)

Following are the specification of the server.

Name: HP AMD A10-5745M APU with Radeon™HD Graphics 2.10GHZ

Processor: AMD Quad-Core 2.1 GHz, up to 2.9 GHz [4Core(s), 4 logical Processors]

RAM: 8 GB DDR3L SDRAM (1333 MHz)

Video Graphics: AMD Radeon R7 M260 (Video Graphics Memory: 6GB)

Physical Memory: 1 TB

Operating System: Ubuntu Linux 16.04 LTS

5.1.2 KVM

An opensource virtualization tool to create multiple virtual Operating systems.

5.1.3 Faucet Controller

SDN controller used to control the OpenFlow switch.

5.1.4 OpenFlow Switch

OpenFlow virtual switch (OVS) /OpenFlow compliant switch (i.e: **Allied Telesis X-930** etc).

5.1.5 Quagga

A routing suite used to establish a communication between different autonomous systems having BGP enabled.

5.1.6 SDN-IXP Manager

Used to manage SDN enable IXP's traffic.

5.1.7 Iperf

A network performance checking tool used to generate dummy traffic.

5.2 Network Topology

There are two network topologies created and tested. One is using physical switch having OpenFlow enabled and second is using virtual switch.

Following is the network topology as shown in Figure 5.1 used in our scenario to create IXP network scenario using physical switch. In this network topology, two separate autonomous systems are created, and to establish a connection between these two autonomous systems, an OpenFlow compatible switch is used, which is further connected to the SDN controller and SDN-IXP manager.

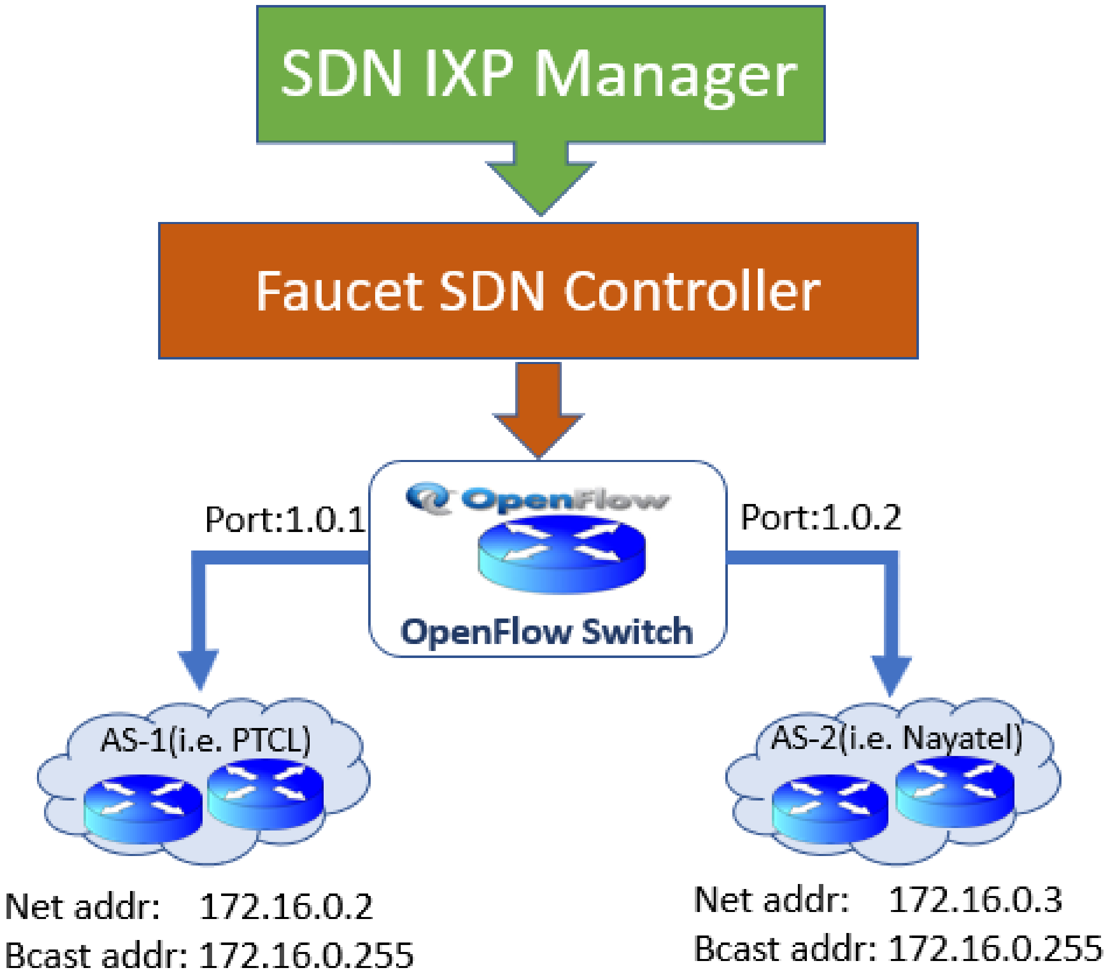


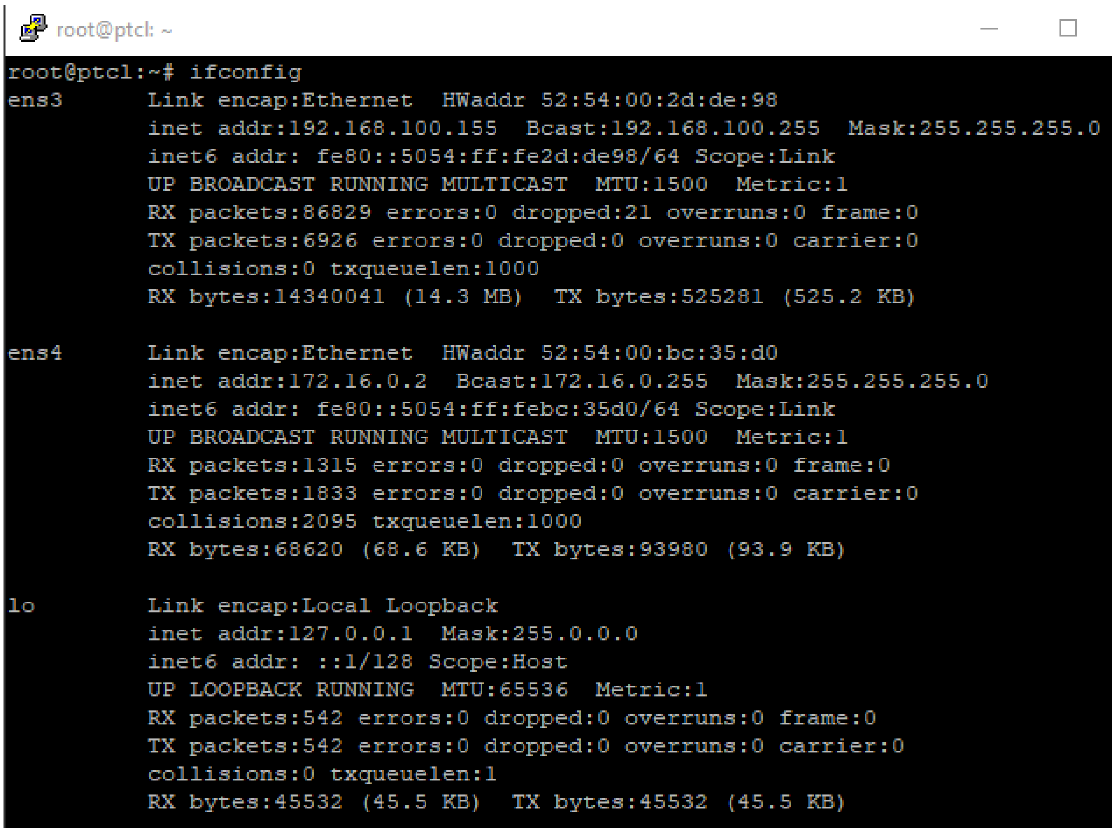
Figure 5.1: Network Topology for Physical OpenFlow Switch.

5.2.1 Network Configurations

Network configurations are made on both virtual servers to act as a separate autonomous system and bgp peering between these systems are established for communication. The purpose of these two virtual servers creations is testing. For ease their names are defined on the names of Pakistan largest ISPS, Ptcl and Nayatel.

AS-1 (PTCL) network Configuration

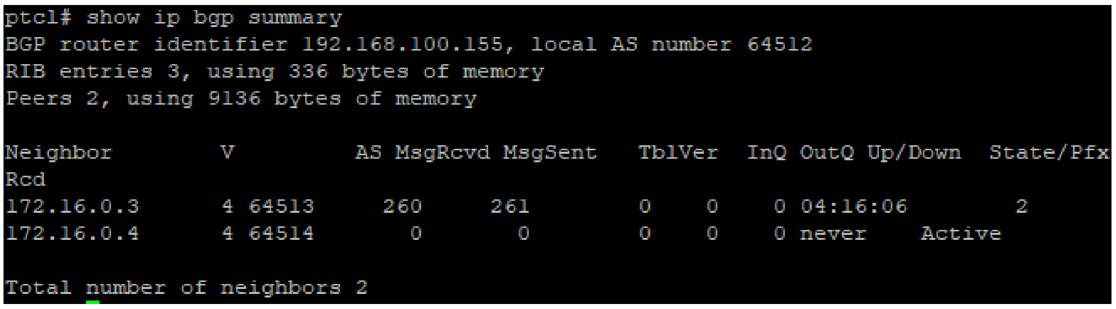
Following Figure 5.2 show the network configuration made on virtual AS-1 (PTCL) network server.



```
root@ptcl: ~
root@ptcl:~# ifconfig
ens3      Link encap:Ethernet  HWaddr 52:54:00:2d:de:98
          inet addr:192.168.100.155  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fe2d:de98/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:86829 errors:0 dropped:21 overruns:0 frame:0
          TX packets:6926 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14340041 (14.3 MB)  TX bytes:525281 (525.2 KB)

ens4      Link encap:Ethernet  HWaddr 52:54:00:bc:35:d0
          inet addr:172.16.0.2  Bcast:172.16.0.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:febc:35d0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1315 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1833 errors:0 dropped:0 overruns:0 carrier:0
          collisions:2095 txqueuelen:1000
          RX bytes:68620 (68.6 KB)  TX bytes:93980 (93.9 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:542 errors:0 dropped:0 overruns:0 frame:0
          TX packets:542 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:45532 (45.5 KB)  TX bytes:45532 (45.5 KB)
```

Figure 5.2: Network configuration made on virtual AS-1 (PTCL) network server.

Following Figure 5.3 are the BGP configurations made on AS-1(PTCL) server.

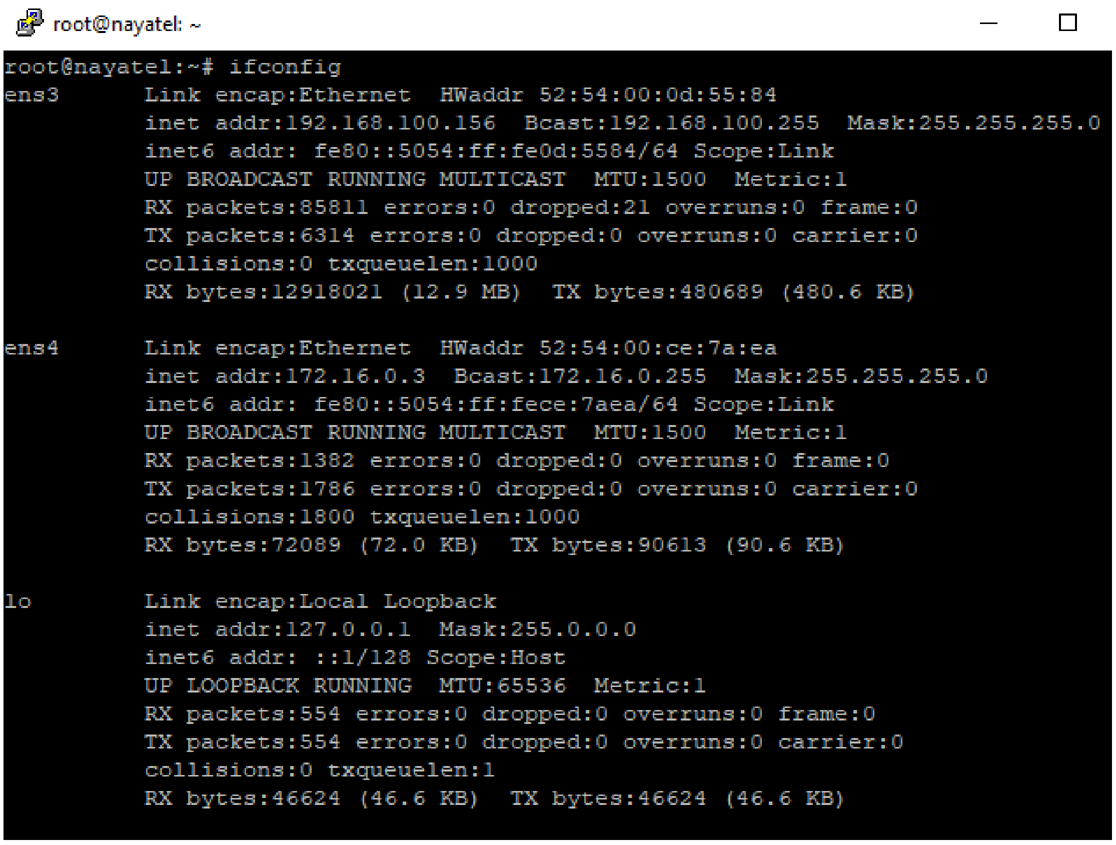


```
ptcl# show ip bgp summary
BGP router identifier 192.168.100.155, local AS number 64512
RIB entries 3, using 336 bytes of memory
Peers 2, using 9136 bytes of memory

Neighbor        V         AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/Pfx
Rcd
172.16.0.3      4 64513     260     261        0    0    0 04:16:06        2
172.16.0.4      4 64514       0       0        0    0    0 never    Active

Total number of neighbors 2
```

Figure 5.3: BGP configurations made on AS-1(PTCL) server.

AS-2 (Nayatel) network Configuration

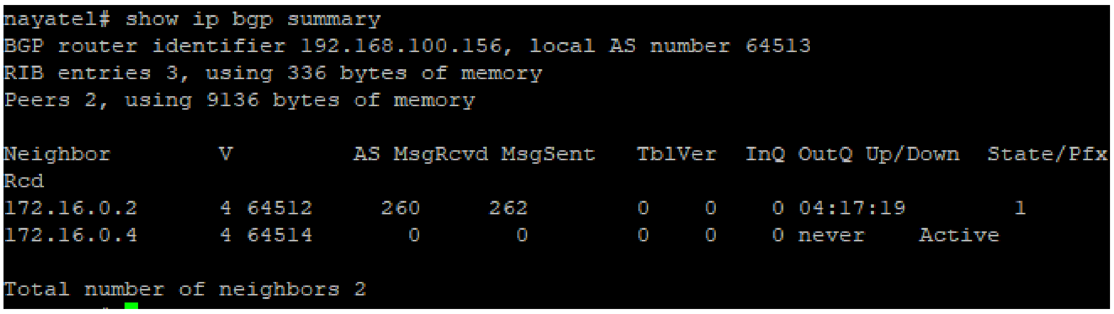
Following Figure 5.4 shows the network configuration made on virtual AS-2 (Nayatel) network server.



```
root@nayatel: ~
root@nayatel:~# ifconfig
ens3      Link encap:Ethernet  HWaddr 52:54:00:0d:55:84
          inet addr:192.168.100.156  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fe0d:5584/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:85811 errors:0 dropped:21 overruns:0 frame:0
          TX packets:6314 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:12918021 (12.9 MB)  TX bytes:480689 (480.6 KB)

ens4      Link encap:Ethernet  HWaddr 52:54:00:ce:7a:ea
          inet addr:172.16.0.3  Bcast:172.16.0.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fece:7aea/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1382 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1786 errors:0 dropped:0 overruns:0 carrier:0
          collisions:1800 txqueuelen:1000
          RX bytes:72089 (72.0 KB)  TX bytes:90613 (90.6 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:554 errors:0 dropped:0 overruns:0 frame:0
          TX packets:554 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:46624 (46.6 KB)  TX bytes:46624 (46.6 KB)
```

Figure 5.4: Network configuration made on virtual AS-2 (Nayatel) network server.

Following are the Figure 5.5 BGP configurations made on AS-2 (Nayatel) server.



```
nayatel# show ip bgp summary
BGP router identifier 192.168.100.156, local AS number 64513
RIB entries 3, using 336 bytes of memory
Peers 2, using 9136 bytes of memory

Neighbor        V         AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/Pfx
Rcd
172.16.0.2      4 64512     260     262        0    0    0 04:17:19        1
172.16.0.4      4 64514       0       0        0    0    0 never    Active

Total number of neighbors 2
```

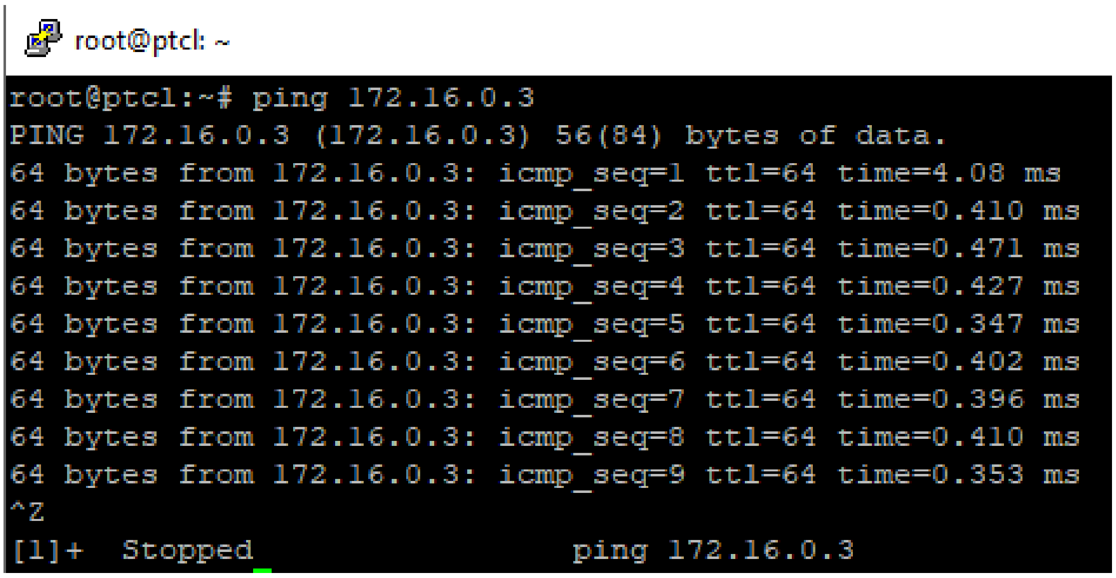
Figure 5.5: BGP configurations made on AS-2 (Nayatel) server.

5.3 Network Testing

To test the network performance and connectivity, we use ping to check the connectivity and Iperf tool to check the network performance. Following are the results obtained after testing AS-1 (i.e. PTCL) and AS-2 (i.e. Nayatel server).

5.3.1 AS-1 PTCL server

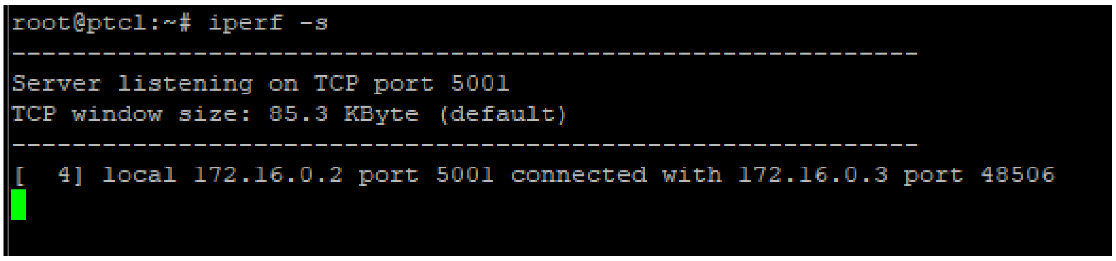
Following are results as shown in Figure 5.6 obtained after pinging the AS-2 (Nayatel) server (172.16.0.3).



```
root@ptcl: ~
root@ptcl:~# ping 172.16.0.3
PING 172.16.0.3 (172.16.0.3) 56(84) bytes of data.
64 bytes from 172.16.0.3: icmp_seq=1 ttl=64 time=4.08 ms
64 bytes from 172.16.0.3: icmp_seq=2 ttl=64 time=0.410 ms
64 bytes from 172.16.0.3: icmp_seq=3 ttl=64 time=0.471 ms
64 bytes from 172.16.0.3: icmp_seq=4 ttl=64 time=0.427 ms
64 bytes from 172.16.0.3: icmp_seq=5 ttl=64 time=0.347 ms
64 bytes from 172.16.0.3: icmp_seq=6 ttl=64 time=0.402 ms
64 bytes from 172.16.0.3: icmp_seq=7 ttl=64 time=0.396 ms
64 bytes from 172.16.0.3: icmp_seq=8 ttl=64 time=0.410 ms
64 bytes from 172.16.0.3: icmp_seq=9 ttl=64 time=0.353 ms
^Z
[1]+  Stopped                 ping 172.16.0.3
```

Figure 5.6: AS-1 PTCL server Ping results.

The below Figure 5.7 illustrates assigning AS-1 (PTCL) as a server while testing the Iperf tool.

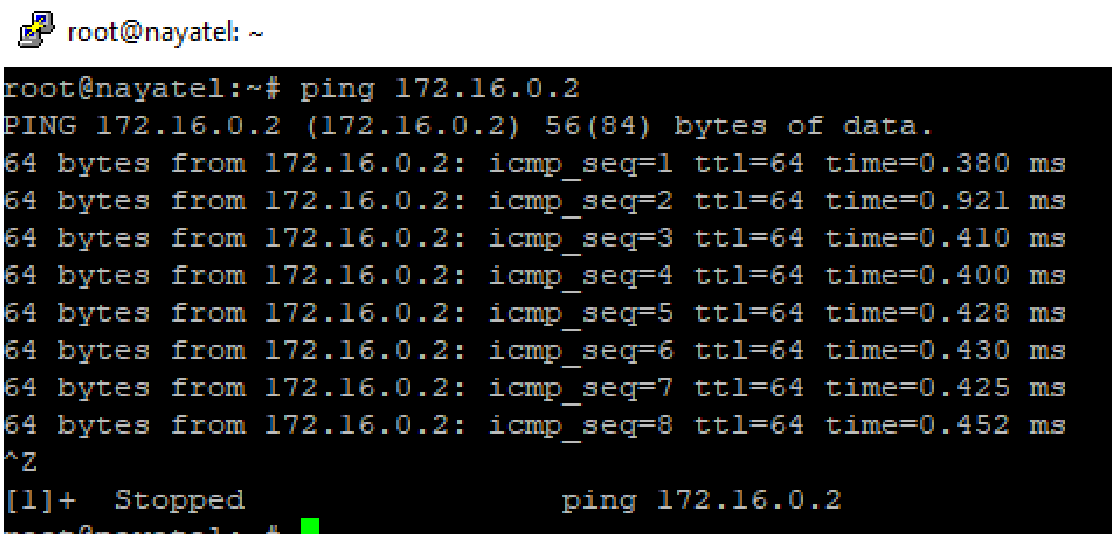


```
root@ptcl:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 172.16.0.2 port 5001 connected with 172.16.0.3 port 48506
```

Figure 5.7: AS-1 PTCL server Iperf results.

5.3.2 AS-2 (Nayatel) server

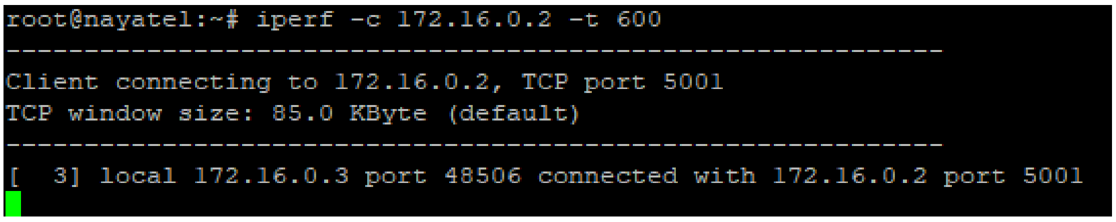
Following are the Results as shown in Figure 5.8 obtained after pinging the AS-1 (PTCL) server (172.16.0.2).



```
root@nayatel: ~
root@nayatel:~# ping 172.16.0.2
PING 172.16.0.2 (172.16.0.2) 56(84) bytes of data.
64 bytes from 172.16.0.2: icmp_seq=1 ttl=64 time=0.380 ms
64 bytes from 172.16.0.2: icmp_seq=2 ttl=64 time=0.921 ms
64 bytes from 172.16.0.2: icmp_seq=3 ttl=64 time=0.410 ms
64 bytes from 172.16.0.2: icmp_seq=4 ttl=64 time=0.400 ms
64 bytes from 172.16.0.2: icmp_seq=5 ttl=64 time=0.428 ms
64 bytes from 172.16.0.2: icmp_seq=6 ttl=64 time=0.430 ms
64 bytes from 172.16.0.2: icmp_seq=7 ttl=64 time=0.425 ms
64 bytes from 172.16.0.2: icmp_seq=8 ttl=64 time=0.452 ms
^Z
[1]+  Stopped                 ping 172.16.0.2
```

Figure 5.8: AS-2 Nayatel server Ping results.

Figure 5.9 of assigning AS-2 (Nayatel) as a client while testing Iperf tool is given by following.

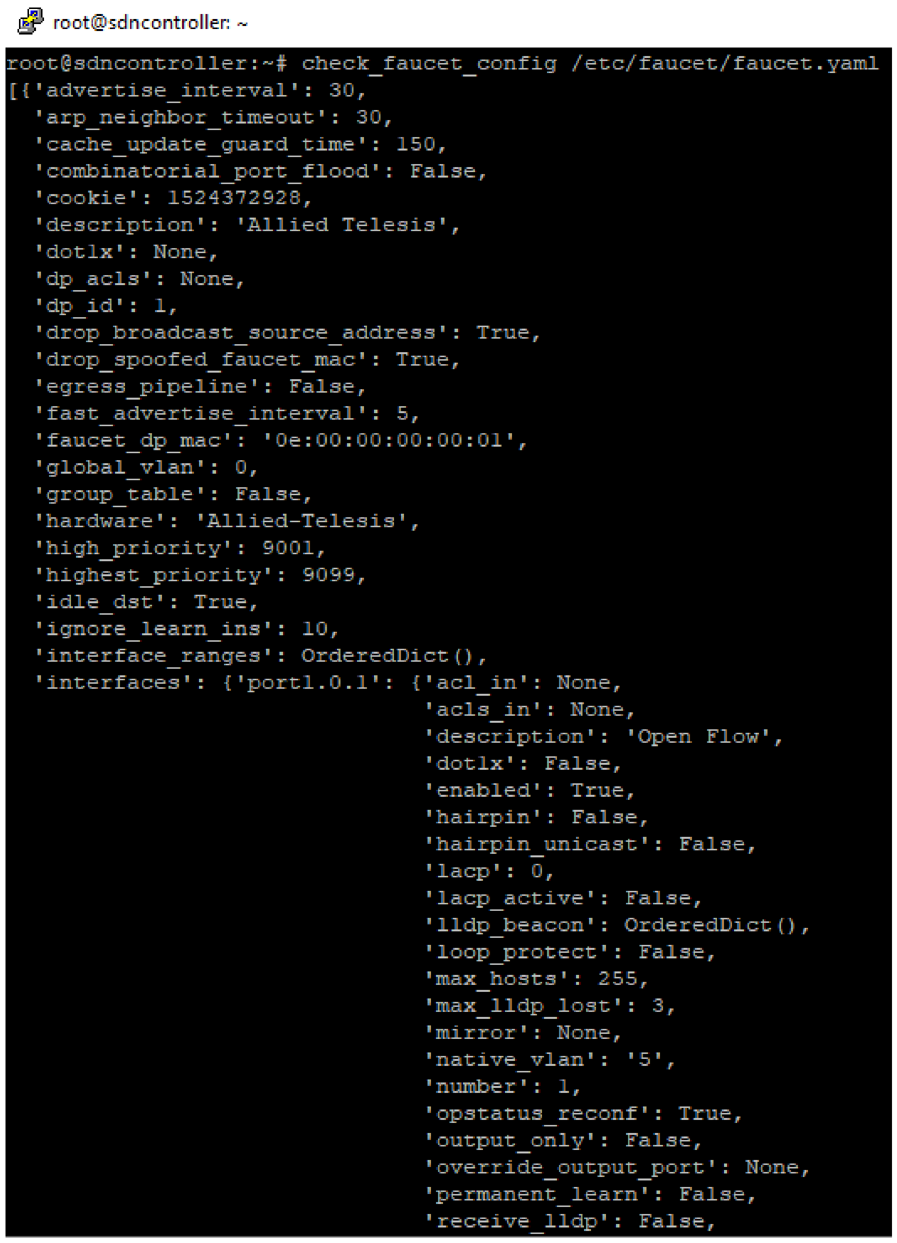


```
root@nayatel:~# iperf -c 172.16.0.2 -t 600
------------------------------------------------------------
Client connecting to 172.16.0.2, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 172.16.0.3 port 48506 connected with 172.16.0.2 port 5001
```

Figure 5.9: AS-2 Nayatel server Iperf results.

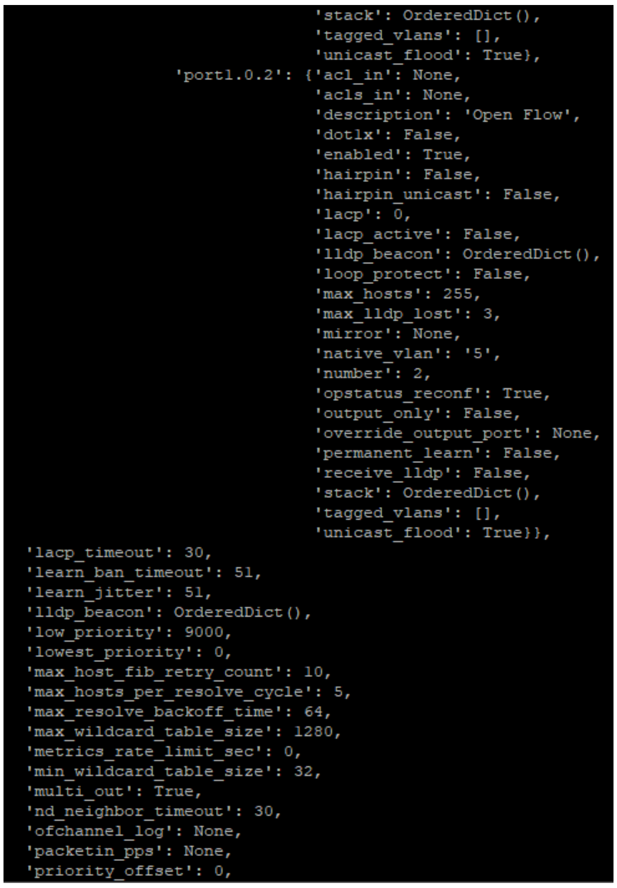
5.4 SDN Controller Configuration

Faucet (SDN controller) is the major component in SDN-IXP environment. Following Figure 5.10 , 5.11 AND 5.12are the configurations made on the controller.


root@sdncontroller: ~

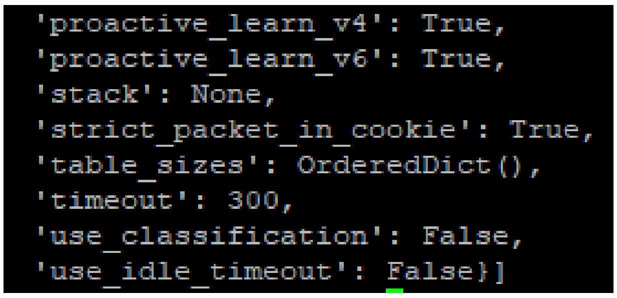
```
root@sdncontroller:~# check_faucet_config /etc/faucet/faucet.yaml
[{'advertise_interval': 30,
  'arp_neighbor_timeout': 30,
  'cache_update_guard_time': 150,
  'combinatorial_port_flood': False,
  'cookie': 1524372928,
  'description': 'Allied Telesis',
  'dot1x': None,
  'dp_acls': None,
  'dp_id': 1,
  'drop_broadcast_source_address': True,
  'drop_spoofed_faucet_mac': True,
  'egress_pipeline': False,
  'fast_advertise_interval': 5,
  'faucet_dp_mac': '0e:00:00:00:00:01',
  'global_vlan': 0,
  'group_table': False,
  'hardware': 'Allied-Telesis',
  'high_priority': 9001,
  'highest_priority': 9099,
  'idle_dst': True,
  'ignore_learn_ins': 10,
  'interface_ranges': OrderedDict(),
  'interfaces': {'port1.0.1': {'acl_in': None,
                               'acls_in': None,
                               'description': 'Open Flow',
                               'dot1x': False,
                               'enabled': True,
                               'hairpin': False,
                               'hairpin_unicast': False,
                               'lacp': 0,
                               'lacp_active': False,
                               'lldp_beacon': OrderedDict(),
                               'loop_protect': False,
                               'max_hosts': 255,
                               'max_lldp_lost': 3,
                               'mirror': None,
                               'native_vlan': '5',
                               'number': 1,
                               'opstatus_reconf': True,
                               'output_only': False,
                               'override_output_port': None,
                               'permanent_learn': False,
                               'receive_lldp': False,
```

Figure 5.10: Faucet (SDN controller) configuration (a).



```
                             'stack': OrderedDict(),
                             'tagged_vlans': [],
                             'unicast_flood': True},
               'port1.0.2': {'acl_in': None,
                             'acls_in': None,
                             'description': 'Open Flow',
                             'dot1x': False,
                             'enabled': True,
                             'hairpin': False,
                             'hairpin_unicast': False,
                             'lacp': 0,
                             'lacp_active': False,
                             'lldp_beacon': OrderedDict(),
                             'loop_protect': False,
                             'max_hosts': 255,
                             'max_lldp_lost': 3,
                             'mirror': None,
                             'native_vlan': '5',
                             'number': 2,
                             'opstatus_reconf': True,
                             'output_only': False,
                             'override_output_port': None,
                             'permanent_learn': False,
                             'receive_lldp': False,
                             'stack': OrderedDict(),
                             'tagged_vlans': [],
                             'unicast_flood': True}},
'lacp_timeout': 30,
'learn_ban_timeout': 51,
'learn_jitter': 51,
'lldp_beacon': OrderedDict(),
'low_priority': 9000,
'lowest_priority': 0,
'max_host_fib_retry_count': 10,
'max_hosts_per_resolve_cycle': 5,
'max_resolve_backoff_time': 64,
'max_wildcard_table_size': 1280,
'metrics_rate_limit_sec': 0,
'min_wildcard_table_size': 32,
'multi_out': True,
'nd_neighbor_timeout': 30,
'ofchannel_log': None,
'packetin_pps': None,
'priority_offset': 0,
```

Figure 5.11: Faucet (SDN controller) configuration (b).

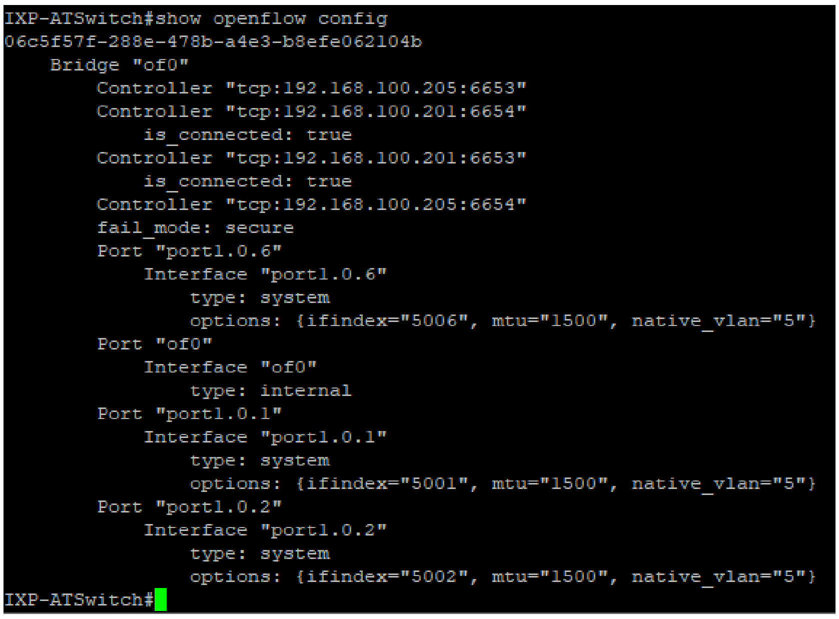


```
'proactive_learn_v4': True,
'proactive_learn_v6': True,
'stack': None,
'strict_packet_in_cookie': True,
'table_sizes': OrderedDict(),
'timeout': 300,
'use_classification': False,
'use_idle_timeout': False}]
```

Figure 5.12: Faucet (SDN controller) configuration (c).

5.5 OpenFlow switch Configuration

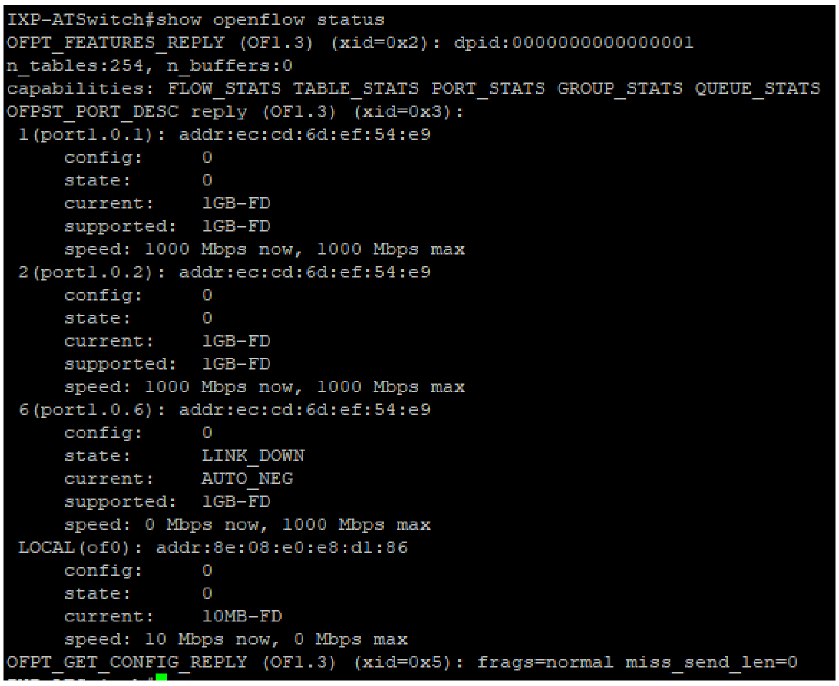
As OpenFlow switch is the bridge between two autonomous systems to establish connectivity. Following Figure 5.13 are the configurations made on the OpenFlow compliant switch.



```
IXP-ATSwitch#show openflow config
06c5f57f-288e-478b-a4e3-b8efe062104b
    Bridge "of0"
        Controller "tcp:192.168.100.205:6653"
        Controller "tcp:192.168.100.201:6654"
            is_connected: true
        Controller "tcp:192.168.100.201:6653"
            is_connected: true
        Controller "tcp:192.168.100.205:6654"
        fail_mode: secure
        Port "port1.0.6"
            Interface "port1.0.6"
                type: system
                options: {ifindex="5006", mtu="1500", native_vlan="5"}
        Port "of0"
            Interface "of0"
                type: internal
        Port "port1.0.1"
            Interface "port1.0.1"
                type: system
                options: {ifindex="5001", mtu="1500", native_vlan="5"}
        Port "port1.0.2"
            Interface "port1.0.2"
                type: system
                options: {ifindex="5002", mtu="1500", native_vlan="5"}
IXP-ATSwitch#
```

Figure 5.13: OpenFlow switch configuration.

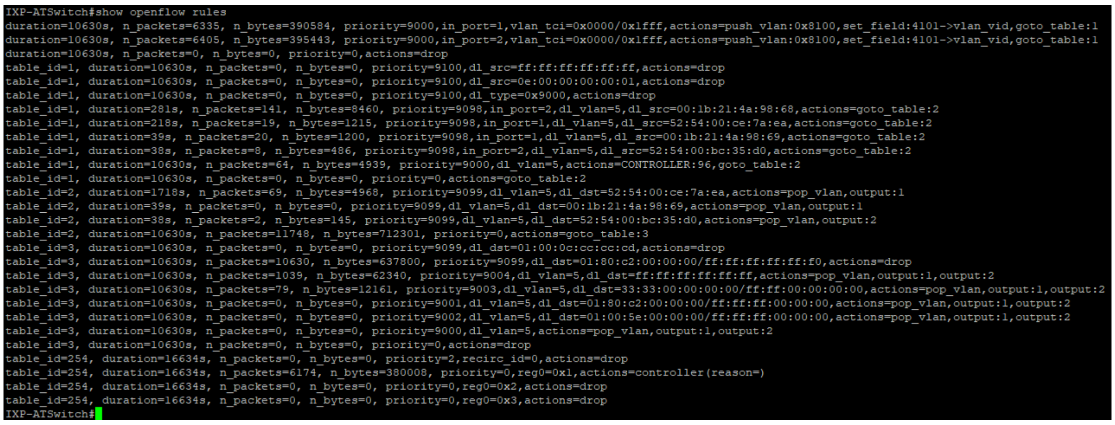
The below Figure 5.14 clearly explains the information of each of the ports of the switch.



```
IXP-ATSwitch#show openflow status
OFPT_FEATURES_REPLY (OF1.3) (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS GROUP_STATS QUEUE_STATS
OFPST_PORT_DESC reply (OF1.3) (xid=0x3):
 1(port1.0.1): addr:ec:cd:6d:ef:54:e9
     config:     0
     state:      0
     current:    1GB-FD
     supported:  1GB-FD
     speed: 1000 Mbps now, 1000 Mbps max
 2(port1.0.2): addr:ec:cd:6d:ef:54:e9
     config:     0
     state:      0
     current:    1GB-FD
     supported:  1GB-FD
     speed: 1000 Mbps now, 1000 Mbps max
 6(port1.0.6): addr:ec:cd:6d:ef:54:e9
     config:     0
     state:      LINK_DOWN
     current:    AUTO_NEG
     supported:  1GB-FD
     speed: 0 Mbps now, 1000 Mbps max
 LOCAL(of0): addr:8e:08:e0:e8:d1:86
     config:     0
     state:      0
     current:    10MB-FD
     speed: 10 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (OF1.3) (xid=0x5): frags=normal miss_send_len=0
```

Figure 5.14: OpenFlow switch status.

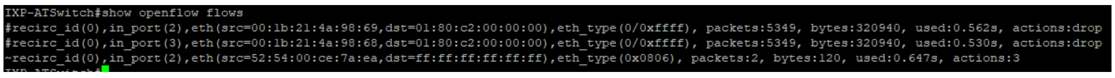
The defined rules of OpenFlow Switch are shown in Figure 5.15.



```
IXP-ATSwitch#show openflow rules
duration=10630s, n_packets=6335, n_bytes=390584, priority=9000,in_port=1,vlan_tci=0x0000/0x1fff,actions=push_vlan:0x8100,set_field:4101->vlan_vid,goto_table:1
duration=10630s, n_packets=6405, n_bytes=395443, priority=9000,in_port=2,vlan_tci=0x0000/0x1fff,actions=push_vlan:0x8100,set_field:4101->vlan_vid,goto_table:1
duration=10630s, n_packets=0, n_bytes=0, priority=0,actions=drop
table_id=1, duration=10630s, n_packets=0, n_bytes=0, priority=9100,dl_src=ff:ff:ff:ff:ff:ff,actions=drop
table_id=1, duration=10630s, n_packets=0, n_bytes=0, priority=9100,dl_src=0e:00:00:00:00:01,actions=drop
table_id=1, duration=10630s, n_packets=0, n_bytes=0, priority=9100,dl_type=0x9000,actions=drop
table_id=1, duration=281s, n_packets=141, n_bytes=8460, priority=9098,in_port=2,dl_vlan=5,dl_src=00:1b:21:4a:98:68,actions=goto_table:2
table_id=1, duration=218s, n_packets=19, n_bytes=1215, priority=9098,in_port=1,dl_vlan=5,dl_src=52:54:00:ce:7a:ea,actions=goto_table:2
table_id=1, duration=39s, n_packets=20, n_bytes=1200, priority=9098,in_port=1,dl_vlan=5,dl_src=00:1b:21:4a:98:69,actions=goto_table:2
table_id=1, duration=38s, n_packets=8, n_bytes=486, priority=9098,in_port=2,dl_vlan=5,dl_src=52:54:00:bc:35:d0,actions=goto_table:2
table_id=1, duration=10630s, n_packets=64, n_bytes=4939, priority=9000,dl_vlan=5,actions=CONTROLLER:96,goto_table:2
table_id=1, duration=10630s, n_packets=0, n_bytes=0, priority=0,actions=goto_table:2
table_id=2, duration=1718s, n_packets=69, n_bytes=4968, priority=9099,dl_vlan=5,dl_dst=52:54:00:ce:7a:ea,actions=pop_vlan,output:1
table_id=2, duration=39s, n_packets=0, n_bytes=0, priority=9099,dl_vlan=5,dl_dst=00:1b:21:4a:98:69,actions=pop_vlan,output:1
table_id=2, duration=38s, n_packets=2, n_bytes=145, priority=9099,dl_vlan=5,dl_dst=52:54:00:bc:35:d0,actions=pop_vlan,output:2
table_id=2, duration=10630s, n_packets=11748, n_bytes=712301, priority=0,actions=goto_table:3
table_id=3, duration=10630s, n_packets=0, n_bytes=0, priority=9099,dl_dst=01:00:0c:cc:cc:cd,actions=drop
table_id=3, duration=10630s, n_packets=10630, n_bytes=637800, priority=9099,dl_dst=01:80:c2:00:00:00/ff:ff:ff:ff:ff:f0,actions=drop
table_id=3, duration=10630s, n_packets=1039, n_bytes=62340, priority=9004,dl_vlan=5,dl_dst=ff:ff:ff:ff:ff:ff,actions=pop_vlan,output:1,output:2
table_id=3, duration=10630s, n_packets=79, n_bytes=12161, priority=9003,dl_vlan=5,dl_dst=33:33:00:00:00:00/ff:ff:00:00:00:00,actions=pop_vlan,output:1,output:2
table_id=3, duration=10630s, n_packets=0, n_bytes=0, priority=9001,dl_vlan=5,dl_dst=01:80:c2:00:00:00/ff:ff:ff:00:00:00,actions=pop_vlan,output:1,output:2
table_id=3, duration=10630s, n_packets=0, n_bytes=0, priority=9002,dl_vlan=5,dl_dst=01:00:5e:00:00:00/ff:ff:ff:00:00:00,actions=pop_vlan,output:1,output:2
table_id=3, duration=10630s, n_packets=0, n_bytes=0, priority=9000,dl_vlan=5,actions=pop_vlan,output:1,output:2
table_id=3, duration=10630s, n_packets=0, n_bytes=0, priority=0,actions=drop
table_id=254, duration=16634s, n_packets=0, n_bytes=0, priority=2,recirc_id=0,actions=drop
table_id=254, duration=16634s, n_packets=6174, n_bytes=380008, priority=0,reg0=0x1,actions=controller(reason=)
table_id=254, duration=16634s, n_packets=0, n_bytes=0, priority=0,reg0=0x2,actions=drop
table_id=254, duration=16634s, n_packets=0, n_bytes=0, priority=0,reg0=0x3,actions=drop
IXP-ATSwitch#
```

Figure 5.15: Defined OpenFlow rules.

And the below Figure 5.16 shows the flows in the OpenFlow Switch.



```
IXP-ATSwitch#show openflow flows
#recirc_id(0),in_port(2),eth(src=00:1b:21:4a:98:69,dst=01:80:c2:00:00:00),eth_type(0/0xffff), packets:5349, bytes:320940, used:0.562s, actions:drop
#recirc_id(0),in_port(3),eth(src=00:1b:21:4a:98:68,dst=01:80:c2:00:00:00),eth_type(0/0xffff), packets:5349, bytes:320940, used:0.530s, actions:drop
~recirc_id(0),in_port(2),eth(src=52:54:00:ce:7a:ea,dst=ff:ff:ff:ff:ff:ff),eth_type(0x0806), packets:2, bytes:120, used:0.647s, actions:3
IXP-ATSwitch#
```

Figure 5.16: Flows in OpenFlow swotch.

5.6 Grafana Statistical Dashboard

Grafana is a statistical dashboard used to show the statistical results and statistical information in the form of visualizations and graphs.

5.6.1 SDN Controller Status

Following is the Figure 5.17 which shows the configuration of cold restart of the controller, as cold restart is used when there is a need for the controller to relearn completely about the topology.

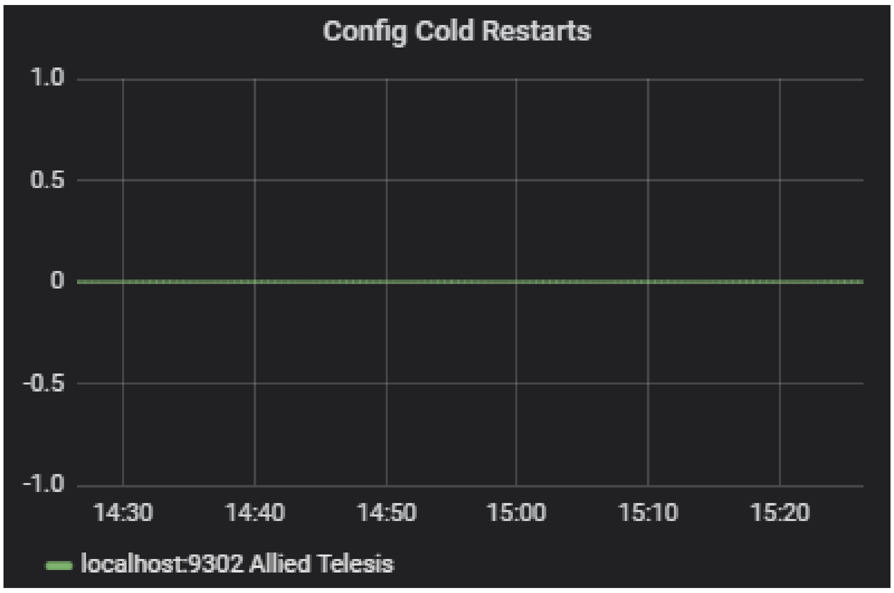


Figure 5.17: SDN Controller cold restart status.

The following Figure 5.18 illustrates the configuration reload requests of the controller. As it is a very important parameter used in faucet controller to countdown the reload requests.

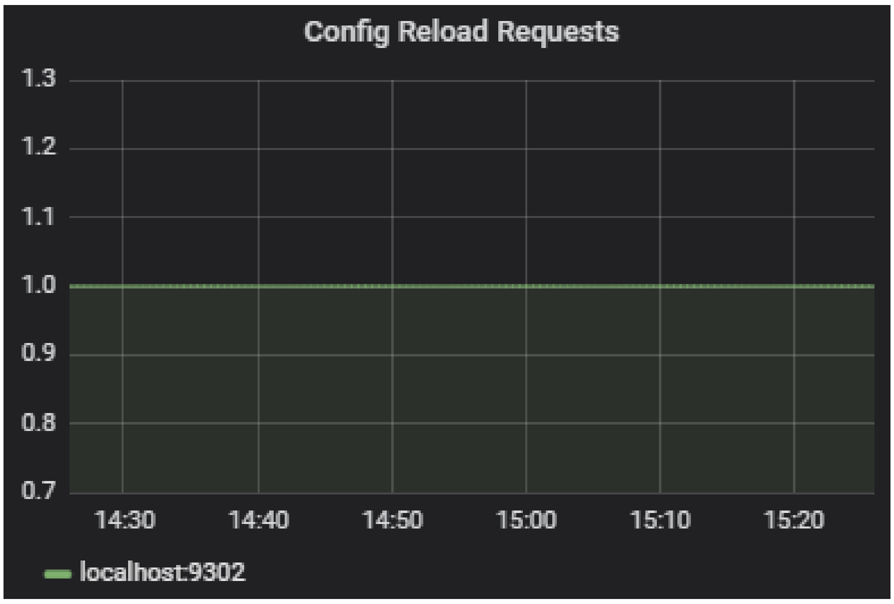


Figure 5.18: SDN Controller reload requests status.

Warm reload is used when there is a need to inform the controller about the changes being made in the controller. Following is the Figure 5.19 which shows the configuration of warm reloads of the controller.

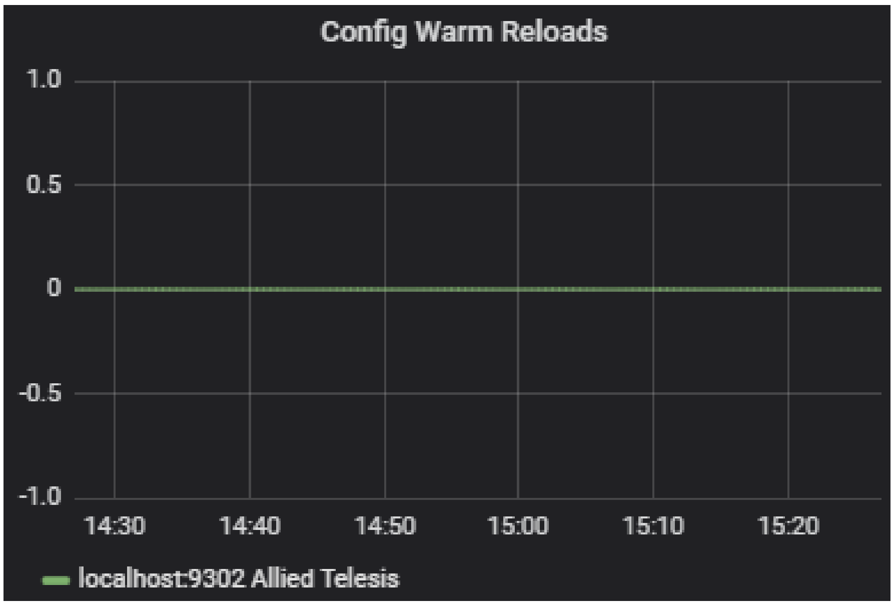
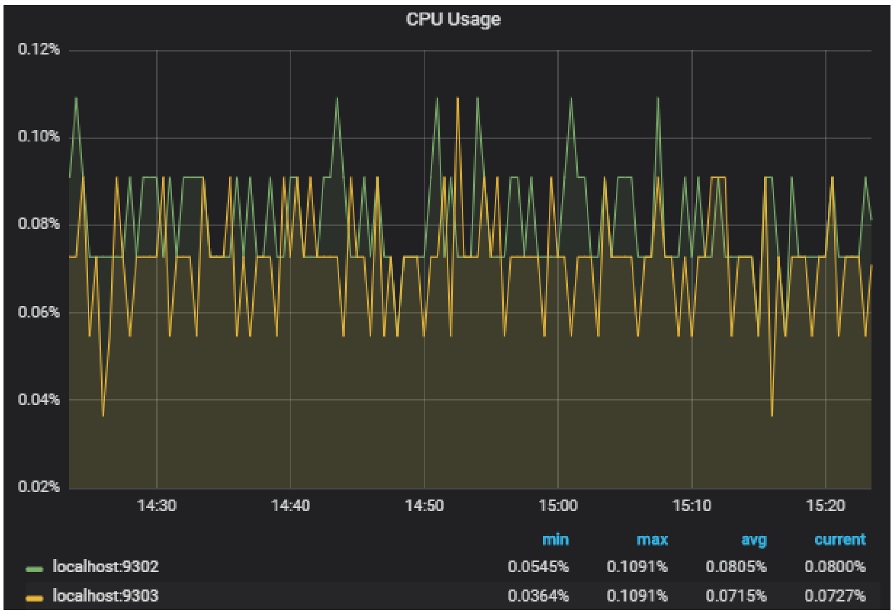


Figure 5.19: SDN Controller warm reload status.

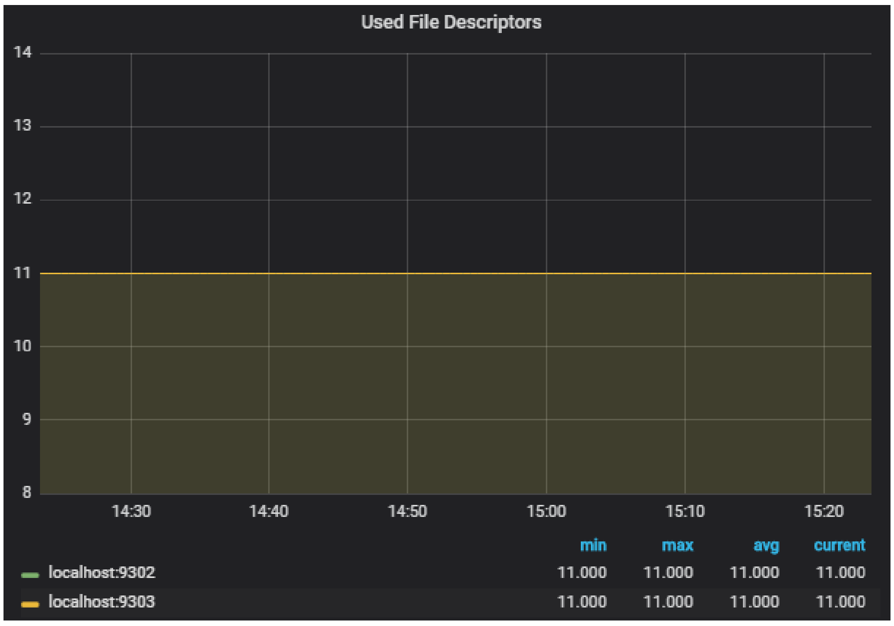
The below Figure 5.20 shows the CPU usage, in terms of memory usage, when controller is running on the server.



	min	max	avg	current
🟩 localhost:9302	0.0545%	0.1091%	0.0805%	0.0800%
🟨 localhost:9303	0.0364%	0.1091%	0.0715%	0.0727%

Figure 5.20: SDN Controller CPU usage status.

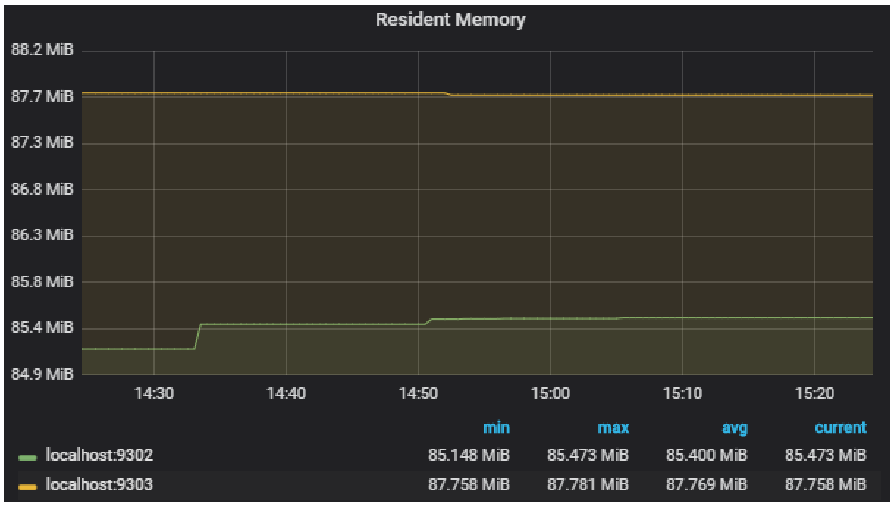
Following is the Figure 5.21 which shows the numbers of files open at a time in a server called **File Descriptor**.



	min	max	avg	current
🟩 localhost:9302	11.000	11.000	11.000	11.000
🟨 localhost:9303	11.000	11.000	11.000	11.000

Figure 5.21: SDN Controller file descriptors status.

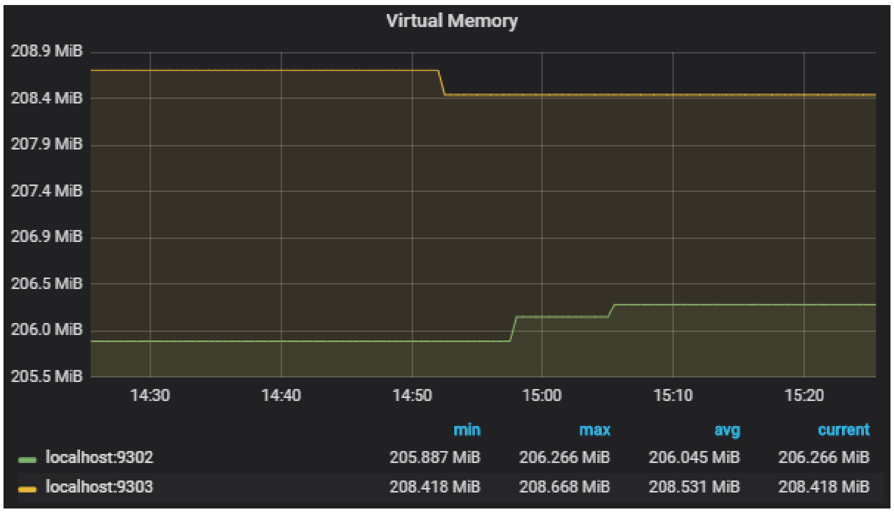
Following is the Figure 5.22 which shows the resident memory usage when controller is running on the server.



	min	max	avg	current
🟩 localhost:9302	85.148 MiB	85.473 MiB	85.400 MiB	85.473 MiB
🟨 localhost:9303	87.758 MiB	87.781 MiB	87.769 MiB	87.758 MiB

Figure 5.22: SDN Controller resident memory status.

Following is the Figure 5.23 which shows the virtual memory usage when controller is running on the server.



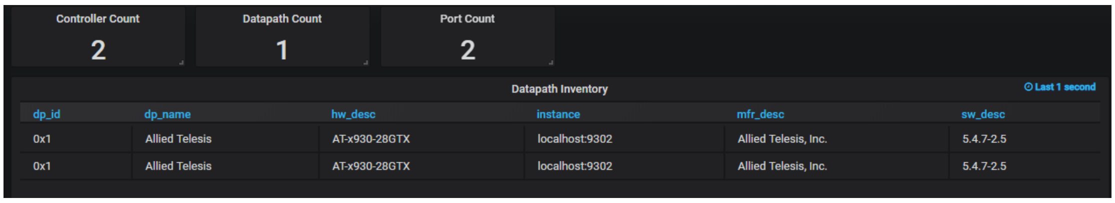
	min	max	avg	current
localhost:9302	205.887 MiB	206.266 MiB	206.045 MiB	206.266 MiB
localhost:9303	208.418 MiB	208.668 MiB	208.531 MiB	208.418 MiB

Figure 5.23: SDN Controller virtual memory status.

5.6.2 SDN Controller Inventory

Controller is the major point in SDN scenario, which contains the information of all the components connected to them. For the purpose, I use Grafana as a statistical dashboard where the information of the connected inventory is displayed.

Following is the Figure 5.24 which contains the inventory information of the controller.


Controller Count 2

Datapath Count 1

Port Count 2

Datapath Inventory

Last 1 second

dp_id	dp_name	hw_desc	instance	mfr_desc	sw_desc
0x1	Allied Telesis	AT-x930-28GTX	localhost:9302	Allied Telesis, Inc.	5.4.7-2.5
0x1	Allied Telesis	AT-x930-28GTX	localhost:9302	Allied Telesis, Inc.	5.4.7-2.5

Figure 5.24: SDN Controller Inventory.

5.6.3 Switch Port Statistics Results

While sending or receiving traffic through OpenFlow compliant switch, A major concern of ISPS is regarding the port switches, as they need to know the port in and out for traffic statistics, In our scenario Grafana is used as a statistical dashboard to display the statistics of each of the ports of the switch. It contains the information of the "in and out"**bits/sec, packets/sec, packets drops/sec** and **errors**.

The Figure 5.25 below contains the results of "in and out"ports of allied Telesis switch in **bits/sec**.

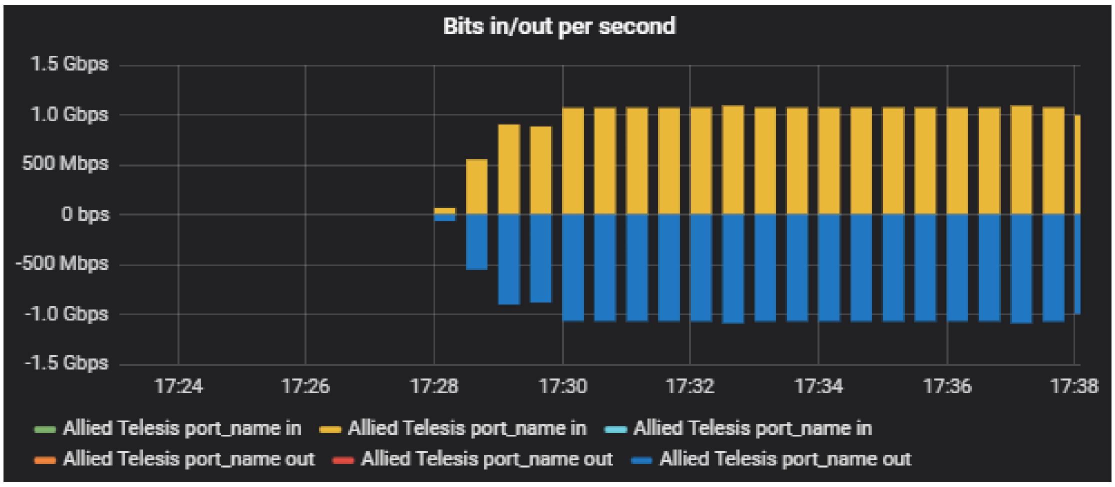


Figure 5.25: Switch ports in/out **bits/sec**.

The below Figure 5.26 contains the results of "in and out"ports of allied Telesis switch in **packets/sec**.

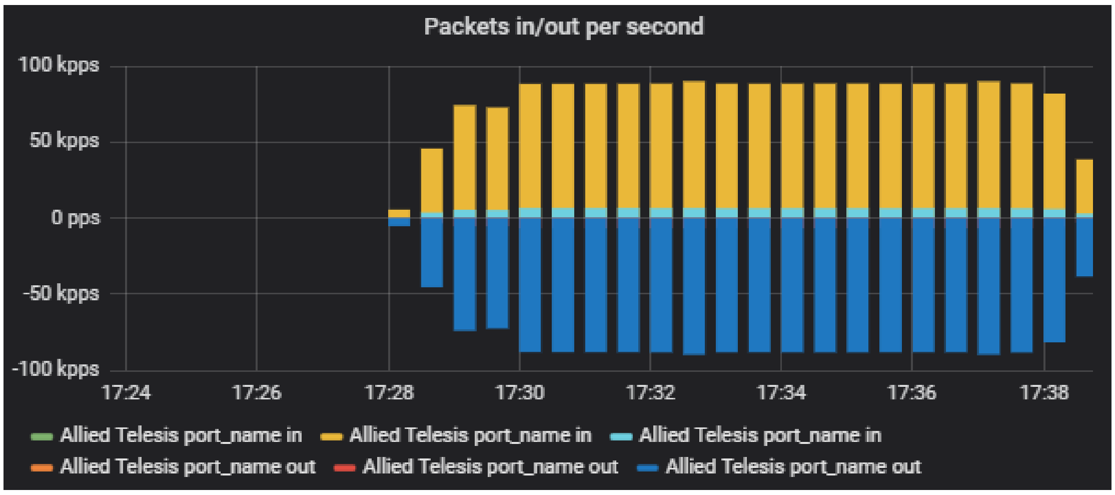


Figure 5.26: Switch ports in/out **packets/sec**.

The Figure 5.27 below contains the results of errors in "in and out"ports of allied Telesis switch in **errors/sec**.

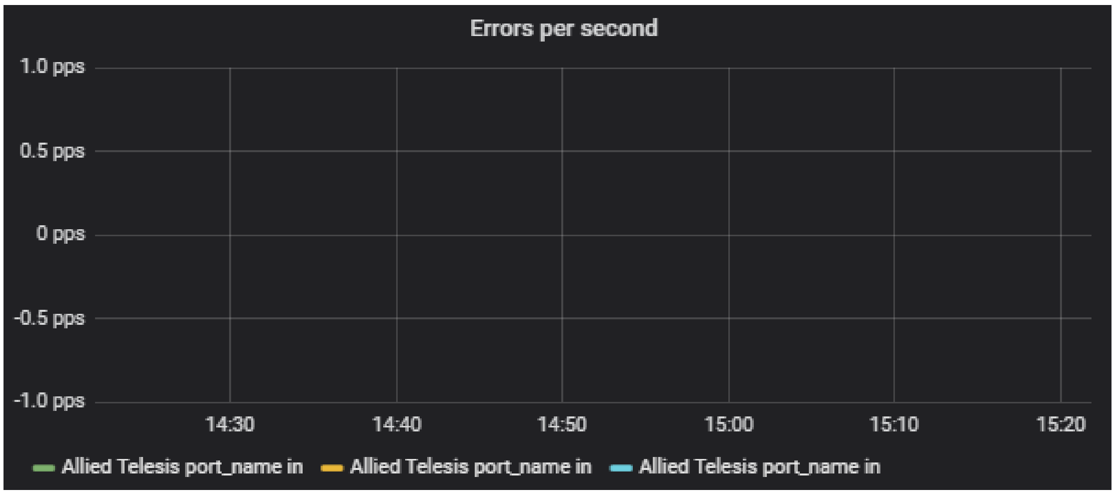


Figure 5.27: Switch ports in/out **errors/sec**.

The below Figure 5.28 contains the results of packets dropped "in and out"ports of allied Telesis switch in **packets/sec**.

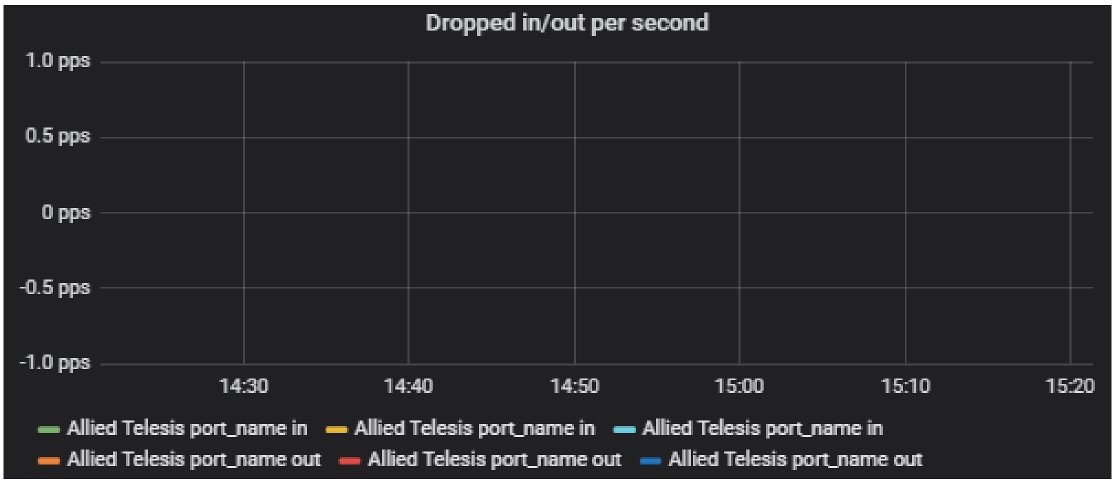


Figure 5.28: Switch ports in/out **dropped packets/sec**.

5.6.4 SDN-IXP Manager

As my methodology's major focus is to provide ease by integrating SDN IXP traffic monitoring, peering and rules defined through one platform, therefore so, following is the Figure 5.29 which shows IXP-Monitoring and Control link in the IXP Manager.

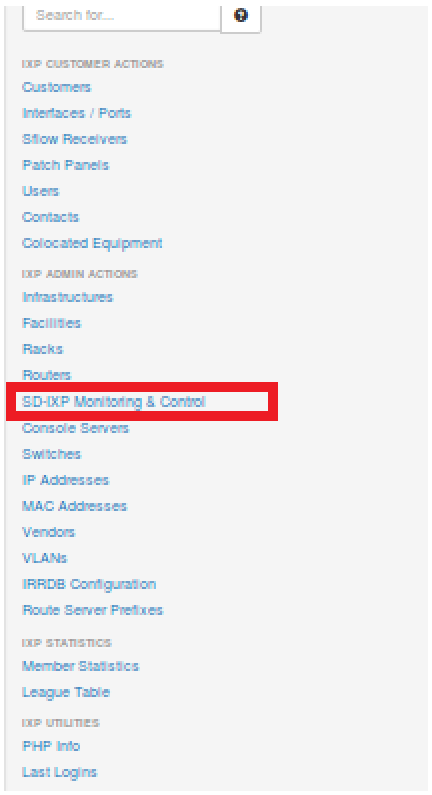
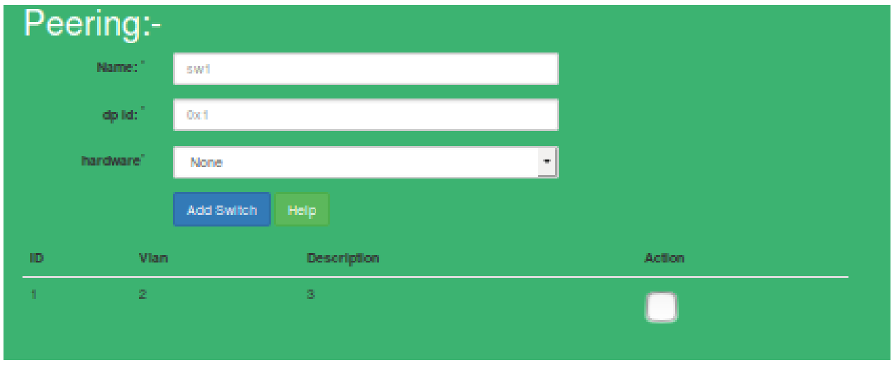


Figure 5.29: IXP-Monitoring and Control link in IXP Manager.

As monitoring is the crux of networking, attached Figure 5.30 shows the link to redirect IXP management team to view the internet traffic.

Figure 5.30: Monitoring redirect button.

The below Figure 5.31 shows Peering information i.e. how to gather information from networking team for establishing a peering between different **ISP's**.



Peering:-

Name:* sw1

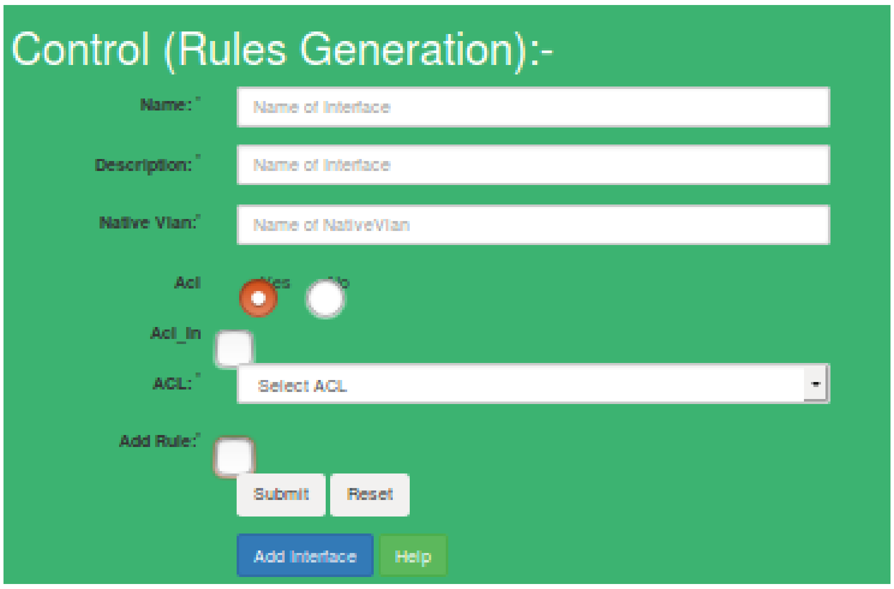
dp id:* 0x1

hardware* None

ID	Vlan	Description	Action
1	2	3	☐

Figure 5.31: Peering Form.

Following Figure 5.32 shows the form used by the IXP management team to generate rules for controlling OpenFlow compliant switches.



Control (Rules Generation):-

Name:* Name of Interface

Description:* Name of Interface

Native Vlan:* Name of NativeVlan

Acl [✓ Yes] No

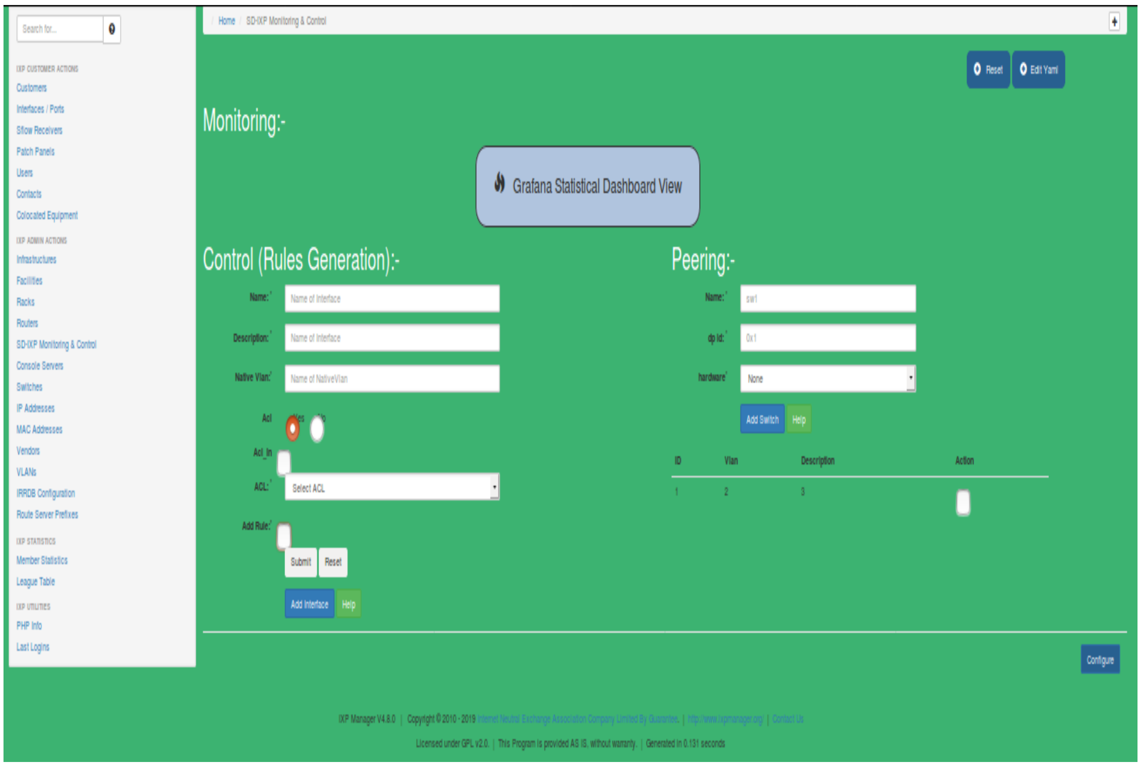
Acl_In ☐

ACL:* Select ACL

Add Rule:* ☐

Figure 5.32: Rules Generation Form.

Following is the Figure 5.33 which shows the complete page where SDN-IXP management team configures the IXP and passes it on to the controller for further processing.



Monitoring:-

Control (Rules Generation):-

Name:* Name of Interface

Description:* Name of Interface

Native Vlan:* Name of NativeVlan

Acl [✓ Yes] No

Acl_In

ACL:* Select ACL

Add Rule:*

Peering:-

Name:* sw1

dp id:* 0x1

hardware* None

ID	Vlan	Description	Action
1	2	3	☐

IXP Manager V4.8.0 | Copyright © 2010 - 2019 Internet Neutral Exchange Association Company Limited By Guarantee. | http://www.ixpmanager.org/ | Contact Us

Licensed under GPL v2.0. | This Program is provided AS IS, without warranty. | Generated in 0.131 seconds

Figure 5.33: Monitoring and Control Page.

Chapter 6

Conclusion

To cater for the rapid growth of internal network traffic within the country, IXPs are designed as a solution to many problems as they reduce latency, as well as bandwidth loss on gateway fiber and improve the security of data etc. With all these technical benefits, they provide a neutral point for all ISPs to share their traffic without getting involved between different ISPs. This results in an increase of the speed of data transfer between ISPs.

With the passage of time and technology evolvement opens a new domain of research namely SDN. As it comes with multiple benefits in terms of programmability of the controller without vendor dependency, separate the control portion of network device from data plane enable centralized management system, reduce the cost and ease for network management team by one click configuration etc, SDN provides multiple benefits to multiple enterprises according to their need as IXP is one of its beneficiaries.

Traditional IXPs used IXP Manager for their traffic management, peering and control but there is a limitation of technology while using traditional network i.e. Packet forwarding and rules are defined only on the destination address field of the packet, while integrating SDN technology with the IXP network enables the decision on every field of the packet. SDN provides benefits to IXP community in terms of broader view which helps IXP to engineer internet traffic, security and load balancing. So, after transitioning into a newer technology, there come some challenges in the form of monitoring and control of IXP traffic. This is the point where the proposed scheme helps IXP management team to overcome this challenge as it is the most effective and convenient way.

The proposed scheme of architecture provides an ease for the control and monitoring of IXP peering. The proposed architecture is used as one platform for the network administrator to monitor the SDN network traffic and define rules from the top of the controller. This scheme has a GUI based framework which can get all the important information from the user and establish a peering between two different ISPs and is also used to control the traffic of the network by defining rules in the form of an access control list (ACL). That is further embedded into the controller that is used to establish an effective communication between different ISPs. The second major concern for network management team is to monitor the internet traffic, which was rectified by redirecting it to the separate dashboard, namely grafana, which is used to show the statistical data of the switch after getting it from the SDN controller. This schema enables a door of technology that creates ease for the existing IXP users who used the IXP just for traditional network, to use SDN enabled IXP network monitoring and control simply by continuing the use of existing IXP management software.

Chapter 7

Future Work and Open Sourcing

To cater for the rapid growth of internet traffic, SDN IXP opens up a new research domain for network engineers globally to come up with new and improved management techniques. Previously, it was not possible through traditional networking as the static nature of traditional hardware devices limit the agility and also doesn't allow rapid changes to network functionality to support new and dynamic use cases. ISPs can easily manage their networking infrastructure operations and security through SDN-IXP Manager as it provides the graphical interface for a network engineer to define rules and monitoring of internet traffic. While converting traditional IXP Manager to SDN IXP Manager there are two more improvements that are needed to be added in future. First is to add multiple SDN controllers in the advance SDN-IXP Manager and the second is to add multiple OpenFlow rules. These improvements are discussed in detail below.

7.1 Multiple SDN Controller addition

As the technology is growing, multiple SDN controllers are being designed by multiple researchers, vendors and organizations. These controllers come with multiple features to cater for the networking demands, these controllers are designed according to the needs of the target customers. SDN provides benefits to the multiple industries that use networking devices. Currently, my proposed architecture is designed using a faucet controller because of its practical adaptability and use cases in SDN IXP deployment. While there is always an area open for other SDN controllers to be integrated into SDN

IXP manager according to the need and usability of the organization.

7.2 New OpenFlow Protocol Rules addition

OpenFlow protocol is the major component in SDN because this protocol is used as a northbound API between the SDN controller and OpenFlow compliant switch. As of now, multiple versions of the OpenFlow protocol have been released with a new and advanced set of standards. While upgrading these versions, new rules are included for effective control. So, in my proposed architecture there is an open opportunity to integrate these rules while redesigning the graphical interface. As it only requires to define the rule type and add its parameters in the existing infrastructure.

7.3 Open Sourcing the work

My research work can be beneficial for the ISPs to manage their networks more effectively and it will also be helpful to the researchers and open source internet community. I believe in community-driven innovation and there is a lot of work that has been done by the open source community in this regard. I therefore want to make my research available for the open source community to benefit from. The research along with results will be made available on GitHub repo for the open source community to make further contribution and improvement in SDN IXP. The said work will be made available after the submission of my publication.

References

[1] Remy Lapeyrade, Marc Bruyere, and Philippe Owezarski. Openflow-based migration and management of the touix ixp. *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 2016. doi: 10.1109/noms.2016.7502975.

[2] Lennart van Gijtenbeek. Research project ii ddos defense mechanisms for ixp infrastructures july 13 , 2018. 2018.

[3] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38 (2):69–74, March 2008. ISSN 0146-4833. doi: 10.1145/1355734.1355746. URL http://doi.acm.org/10.1145/1355734.1355746.

[4] L. G. Roberts. The evolution of packet switching. *Proceedings of the IEEE*, 66(11): 1307–1313, Nov 1978. doi: 10.1109/PROC.1978.11141.

[5] Y. Jarraya, T. Madi, and M. Debbabi. A survey and a layered taxonomy of software-defined networking. *IEEE Communications Surveys Tutorials*, 16(4):1955–1980, Fourthquarter 2014. doi: 10.1109/COMST.2014.2320094.

[6] C. Labovitz, G. R. Malan, and F. Jahanian. Internet routing instability. *IEEE/ACM Transactions on Networking*, 6(5):515–528, Oct 1998. doi: 10.1109/90.731185.

[7] ixp models_2019, 2019. URL http://www.ix-f.net/ixp-models.html.

[8] Internet exchange directory pch-2019, 2019. URL https://www.pch.net/ixp/dir#!mt-sort=ctry2Casc.

[9] internet exchange map_2019, 2019. URL https://www.internetexchangemap.com/.

[10] S. Tomic and A. Jukan. Performance analysis of infrastructure service provision with gmpls-based traffic engineering. *IEEE Journal on Selected Areas in Communications*, 25(5):881–894, June 2007. doi: 10.1109/JSAC.2007.070603.

[11] Open networking foundation is an operator led consortium leveraging sdn, nfv and cloud technologies to transform operator networks and business models_2019, 2019. URL https://www.opennetworking.org/?utm_referrer=https://www.google.com/.

[12] Sdn technical specifications | open networking foundation_2019, 2019. URL https://www.opennetworking.org/software-defined-standards/specifications/.

[13] Rfc 5810 - forwarding and control element separation (forces) protocol specification_2019, 2019. URL https://tools.ietf.org/html/rfc5810.

[14] Draft-wang-forces-compare-openflow-forces-01 - analysis of comparisons between openflow and forces_2019, 2019. URL https://tools.ietf.org/html/draft-wang-forces-compare-openflow-forces-01.

[15] Wolfgang Braun and Michael Menth. Software-defined networking using openflow: Protocols, applications and architectural design choices. *Future Internet*, 6:302–336, 05 2014. doi: 10.3390/fi6020302.

[16] J. Tourrilhes, P. Sharma, S. Banerjee, and J. Pettit. Sdn and openflow evolution: A standards perspective. *Computer*, 47(11):22–29, Nov 2014. doi: 10.1109/MC.2014.326.

[17] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. Are we ready for sdn? implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7):36–43, July 2013. doi: 10.1109/MCOM.2013.6553676.

[18] openflow-spec-v1.3.0_2012, 2012. URL https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf.

[19] Open vswitch documentation, open vswitch 2.12.90 documentation_2019, 2019. URL https://docs.openvswitch.org/en/latest/.

[20] Product Support. Switches - support and downloads, 2019. URL https://www.cisco.com/c/en/us/support/switches/index.html.

[21] Openflow support on juniper networks devices - techlibrary - juniper networks_2019, 2019. URL https://www.juniper.net/documentation/en_US/release-independent/junos/topics/reference/general/junos-sdn-openflow-supported-platforms.html.

[22] Sdn products, portfolio and solutions | hp_2019, 2019. URL https://techlibrary.hpe.com/si/en/networking/solutions/technology/sdn/portfolio.aspx#.XbA8N-gzbDd.

[23] 7150 series_2019, 2019. URL https://www.arista.com/en/products/7150-series.

[24] Ibm bnt rackswitch g8264 iscli command reference guide (6.4)_2019, 2019. URL https://www.ibm.com/support/pages/ibm-bnt-rackswitch-g8264-iscli-command-reference-guide-64.

[25] x930 series_2019, 2019. URL https://www.alliedtelesis.com/products/switches/x930-series.

[26] Maen Artimy. Zodiac-fx switch openflow throughput, 2019. URL http://adhocnode.com/zodiac-fx-switch-openflow-throughput/.

[27] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, and Teresa Vazao. Towards programmable enterprise wlans with odin. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN ’12, pages 115–120, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1477-0. doi: 10.1145/2342441.2342465. URL http://doi.acm.org/10.1145/2342441.2342465.

[28] D. Kreutz, F. M. V. Ramos, P. E. VerÃŋssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015. doi: 10.1109/JPROC.2014.2371999.

[29] Introduction to faucet, python documentation_2019, 2019. URL https://docs.faucet.nz/en/latest/intro.html#.

[30] D. Sheinbein and R. P. Weber. Stored program controlled network: 800 service using spc network capability. *The Bell System Technical Journal*, 61(7):1737–1744, Sep. 1982. doi: 10.1002/j.1538-7305.1982.tb04370.x.

[31] Francesco Paolucci, Filippo Cugini, Alessio Giorgetti, Nicola Sambo, and Piero Castoldi. A survey on the path computation element (pce) architecture. *Communications Surveys Tutorials, IEEE*, 15:1819–1841, 01 2013. doi: 10.1109/SURV.2013.011413.00087.

[32] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. Design and implementation of a routing control platform. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 15–28, Berkeley, CA, USA, 2005. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1251203.1251205.

[33] Understanding the sdn architecture and sdn control plane_2019, 2019. URL https://www.sdxcentral.com/networking/sdn/definitions/inside-sdn-architecture/.

[34] Murat Karakus and Arjan Durresi. A survey. *Comput. Netw.*, 112(C):279–293, January 2017. ISSN 1389-1286. doi: 10.1016/j.comnet.2016.11.017. URL https://doi.org/10.1016/j.comnet.2016.11.017.

[35] Arash Shaghaghi, Mohamed Ali Kaafar, Rajkumar Buyya, and Sanjay Jha. Software-defined network (sdn) data plane security: Issues, solutions and future directions, 2019. URL https://arxiv.org/abs/1804.00262.

[36] Sdn controllers (or sdn controller platforms)_2019, 2019. URL https://www.sdxcentral.com/networking/sdn/definitions/sdn-controllers/.

[37] Product Support and Cloud Management. Cloud and systems management - cisco open sdn controller, 2019. URL https://www.cisco.com/c/en/us/support/cloud-systems-management/open-sdn-controller/tsd-products-support-series-home.html.

[38] Contrail products - juniper networks_2019, 2019. URL https://www.juniper.net/us/en/products-services/sdn/contrail/.

[39] Introduction to the hpe van sdn controller_2019, 2019. URL https://techhub.hpe.com/eginfolib/networking/docs/sdn/sdnc2_7/5200-0910prog/content/c_sdnc-pg-intro.html.

[40] Sdn controller_2019, 2019. URL https://www.bigswitch.com/tags/sdn-controller.

[41] Openflow switches (open flow) in enterprise networks - pica8_2019, 2019. URL https://www.pica8.com/openflow-switch/.

[42] The status of sdn today | cumulus networks_2019, 2019. URL https://cumulusnetworks.com/lp/sdn-dead-long-live-sdn/.

[43] Software-defined networking (sdn) solutions pluribus_2019, 2019. URL https://www.pluribusnetworks.com/solutions/.

[44] Opendaylight documentation opendaylight documentation fluorine documentation_2019, 2019. URL https://docs.opendaylight.org/en/stable-fluorine/.

[45] Project floodlight - openflow news and projects_2019, 2019. URL http://www.projectfloodlight.org/.

[46] Ryu controller tutorial sdn hub_2019, 2019. URL http://sdnhub.org/tutorials/ryu/.

[47] About onos (open network operating system), 2019. URL onos.

[48] David Erickson. The beacon openflow controller. pages 13–18, 08 2013. doi: 10.1145/2491185.2491189.

[49] Openflow-based faucet provides high-speed sdn controller virtualization review_2019, 2019. URL https://virtualizationreview.com/articles/2019/01/22/faucet.aspx.

[50] Stephen Stuart Josh Bailey. Faucet: Deploying sdn in the enterprise communications for acm 2016 12, 08 2019. URL https://queue.acm.org/detail.cfm?id=3015763.

[51] Faucet documentation, faucet documentation_2019, 2019. URL https://docs.faucet.nz/en/latest/.

[52] Faucet architecture - faucet documentation_2019, 2019. URL https://docs.faucet.nz/en/latest/architecture.html.

[53] Philippe Owezarski Remy Lapeyrade, Marc Bruyere. Openflow-based migration and management of the touix ixp. page 7, 04 2016. doi: 10.1109/NOMS.2016.7502975f.

[54] Tousix project | tousix internet exchange_2019, 2019. URL http://www.touix.net/en/content/tousix-project.

[55] Installation faucet python documentation_2019, 2019. URL https://docs.faucet.nz/en/latest/installation.html.