MINT 709 Capstone Project Report

Router Scripting

Narinder Singh Lehndra
lehndra@ualberta.ca

Supervisor: Arsh Saini

Submitted to: Prof. Mike McGregor

Department of Computing Science
University of Alberta

March 7, 2015

Abstract

Networks are getting more complex day by day. Network traffic gets generated using various routing protocols and it originates from various networks, regions and geolocations. So it becomes a difficult task to monitor and analyze this large and complex network information especially if one considers logging into the intended routers and running commands to get the required information.

There is a profound need to make a user friendly interface which can be used to store network information on a regular basis and provide a front end to present router and network data as per required parameters.

The main goal of this project is to create a router script to get the route distribution on the network from various routing protocols and VRF information coming from various locations and then, provide a front end and database to present and store this information.

$\sim$

Contents

List of Figures

List of Tables

List of Acronyms

MPLS Multiprotocol Label Switching

VRF Virtual Routing Forwarding

LSR Label Switch Router

PE Router Provider Edge Router

P Router Provider Router

CE Router Customer Edge Router

VPN Virtual Private Network

LSP Label Swiched Path

FEC Forwarding Equivalence Class

LDP Label Distribution Protocol

IGP Interior Gateway Protocol

IP Internet Protocol

IPv4 Internet Protocol version 4

LIB Label Information Base

LFIB Label Forwaring Information Base

CEF Cisco Express Forwarding

RD Route Distinguisher

RT Route Target

BGP Border Gateway Protocol

iBGP Internal Border Gateway Protocol

IS-IS Intermediate System to Intermediate System

OSPF Open Shortest Path First

SSH Secure Shell

URL Uniform Resource Locator

SQL Structured Query Language

IOS Internet Operating System

AS Autonomous System

API Application Programming Interface

Chapter 1

Introduction

With organizations growing day by day, they operate from not just a single location but multiple locations and regions. Multiprotocol Label Switching (MPLS) helps to achieve the connectivity among these different locations by using the service provider routers. But these networks are getting more complex day by day with network traffic getting generated using various routing protocols and originating from various networks, regions and geolocations.

To sustain the reliabiliy and performance of the these networks, the network traffic needs to be monitored on a regular basis. It becomes a tedious task to monitor and analyze this large and complex network information especially if one considers logging into the intended routers and running commands to get the required information.

So, there is a growing need to make a user friendly web interface which can be used to get the network information passing through one location, which includes route distribution on the network from various routing protocols and VRF information coming from different locations, store that network information in a database and provide a front end to present router and network data as per required parameters.

The goal of this project is to create a database and provide a web based front end to display the route distribution from various routing protocols on the network, VRF coming from different regions, networks and geolocations. The required information will be retrieved using a custom script that will reach the routers. The retrieved information will be modulated as per needs and inserted into the database. The intent is to provide a user friendly platform to present network

data so that it can be easily analysed by the network engineer and necessary changes can be made in the network if need arises.

~

Chapter 2

Literature Review

In order to simulate the proposed network of multiple customers, we will create an MPLS network connecting different customer locations. After that a python script will be coded to make a secure shell login into the routers and get the required network information. Django, a Python based web framework will be used to store the retrieved information into Mysql database and display the VRF information on a web page.

In the following section we will try to provide background description on the all the technologies which we will be using to simulate the network and retrieve the information from the network and then store and display the VRF information.

2.1 Multiprotocol Label Switching (MPLS)

In MPLS, a label-to-label mapping is built between routers in order to advertise MPLS labels betwen routers. These MPLS labels are attached to the IP packets. This enables the routers to forward the traffic according to the label and not according to the destination IP address. So, label switching is used to forward the packets, not the IP swiching.

In Label switching, the packets switched are no longer IPv4 packets, IPv6 packets, or even Layer 2 frames when they are switched, but they are labeled. The label is the most important item to MPLS. In the following sections of this chapter we will have look on the functionality of the label and how it is distributed in a network.

2.1.1 MPLS Labels

An MPLS label can be defined as the information attached to every packet which tells the intermediate router that the packet must be forwarded to which edge router.

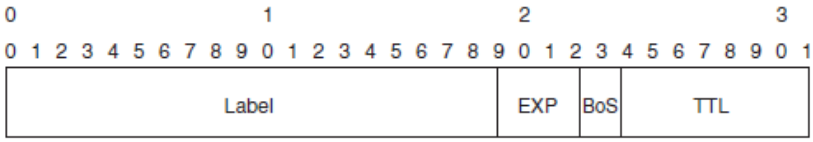

Figure 2.1: MPLS Label

Label: 20 bits field , value can be between 0 and 2^{20}-1 (1048575).
EXP: experimental bits used for QoS (Quality of Service).
BoS: Bottom of Stack (BoS) indicates that if the label is the last label in the stack or not.
0 - not a last label in stack
1 - last label in stack
TTL: Time to Live , decreased by one at each hop.

In order to route the packet through the MPLS network, more than one label might be needed on top of the packet. This is accomplished by packing the labels into a stack. We call the first label in the stack as the top label, and the last label as the bottom label with any number of labels between them.This label stack resides before the header of the transported protocol, but after the Layer 2 header.

MPLS does not fit in the OSI layering too well because it is neither a Layer 2 protocol nor a Layer 3 protocol. So, in terms of OSI model, MPLS is generally viewed as 2.5 layer.

2.1.2 Label Switch Router

Label Switch Router (LSR) is a MPLS supporting router that understands MPLS labels and can recieve and transmit MPLS labeled packets over a network. There are three kinds of LSRs:

- **Ingress LSR**: This LSR receives an unlabeled packet and inserts a label in front of the packet before sending it on a data link.

- **Egress LSR**: This LSR receives a labeled packet and removes the label from the packet before sending it on a data link. Ingress and Egress LSRs are edge LSRs, also called as Provider Edge (PE) routers in case of MPLS VPN.
- **Intermediate LSR**: This LSR receives an incoming labeled packet, performs some operation on the packet, switches the packets before sending it on a correct data link. These routers are also called as Provider routers in case of MPLS VPN.

There are three operations that an LSR can do:

- **Pop**: It can remove one or more labels before sending the packet.
- **Push**: It can add one or more labels before sending the packet.
- **Intermediate LSR**: It can be used to swap the top label on the packet with a new label before the packet is sent on the data link.

2.1.3 Label Switched Path

Label Swiched Path (LSP) is the path through the MPLS network created by the sequence of LSRs. The first router in an LSP is the ingress LSR and the last router in an LSP is the egress LSR.

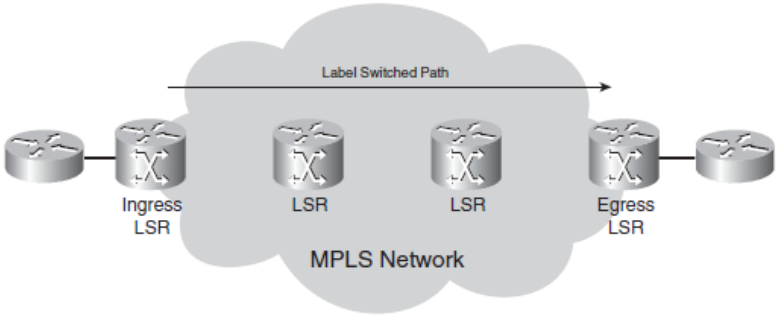

Figure 2.2: Label Switched Path

2.1.4 Forwarding Equivalence Class

A Forwarding Equivalence Class (FEC) is a term used to describe a group of packets in an MPLS network that are forwarded along the same path and are met with the same forwarding treatment.

2.1.5 Label Distribution with LDP

Label Distribution Protocol (LDP) is the protocol used for distributing labels for IGP prefixes. Each LSR creates a local binding to bind a label to the IPv4 for every IGP IP prefix in its IP routing table. This binding is then distributed to all its LDP neighbours by the LSR. These received bindings called remote bindings and other local bindings are stored in a special table, the label information base (LIB) by the neighbours.

The LSR needs to pick only one out of all the remote bindings for one prefix and use it to determine the outgoing label for that IP prefix. The next hop of the IP prefix is determined by the routing table. The LSR chooses the remote binding received from the downstream LSR, the next hop for that perefix in the routing table. This information is used to set up its label forwarding information base (LFIB) where the incoming label is the label from the local binding and the outgoing label is the label from the remote binding chosen by the routing table. So, the LSR is now capable of swapping the incoming label it assigned to the received packet with the outgoing label assigned by the adjacent next-hop LSR.

2.1.6 Label Forwarding Instance Base

The table used to forward labeled packets is called Label Forwarding Instance Base (LFIB). The label from the local binding on the particular LSR is the incoming label and the label from the remote binding chosen by the LSR from all possible remote bindings is the outgoing label. Only one of the possible outgoing labels from all the possible remote bindings in the LIB is chosen by the LFIB and it gets installed in the LFIB. The best path found in the routing table determines the remote label chosen.

2.1.7 Cisco Express Forwarding

Cisco Express Forwarding (CEF) is the default forwarding method used by Cisco IOS used to forward or switch the packets. Labeled packets that enter the router get swiched according to the LFIB on the router and IP packets according to the CEF table on the router. The outgoing packet can be a labeled packet or an IP packet regardless of whether the packet is switched according to the LFIB or the CEF table. In Cisco IOS, CEF is the only switching method that can label an incoming IP packet. CEF consists of two key components: The Forwarding Information Base and adjacencies.

2.1.8 MPLS VPN

MPLS Virtual Private Network is the most widespread implementation of the MPLS technology. It is seen as the next step by the large enterprise companies in their network design as MPLS VPN can provide scalability and divide the network into separate smaller networks making it easier to isolate common IT infrastructure to individual departments.

In order to get familiar with MPLS VPN terminology, look at the following figure for an overview of the MPLS VPN model.

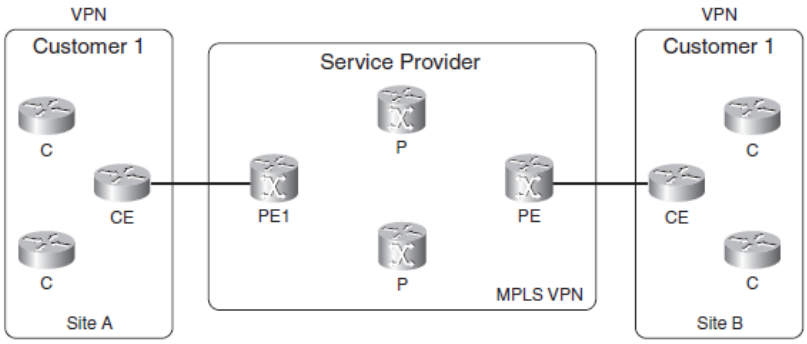

Figure 2.3: MPLS VPN Overview

The provider edge (PE) router has a direct connection with the customer edge router at Layer 3. A provider (P) router does not have any direct connections with the customer (C) routers. Both PE and P routers run MPLS in the MPLS VPN implementation. So, they must be able to distribute labels between them and forward labeled packets. As the CE and PE routers communicate at Layer 3, they must run a routing protocol between them. The CE router has only one peer as PE router outside of its own site and if the CE router is multihomed, it can have multiple PE routers. The CE router cannot peer with any of the CE routers from other sites across the service provider network.

2.1.8.1 Architectural Overview:

Some basic building blocks on the PE routers are needed to achieve MPLS VPN:

Virtaul Routing Forwarding Virtual Routing/Forwarding is a VPN routing and Forwarding instance. It is a combination of the VPN routing table, the VRF CEF table and the associated IP

routing protocols on the PE router. For each attached VPN, a PE router has a VRF instance. The following figure shows that a PE router holds the global IP routing table besides a VRF routing table per VPN connected to the PE.

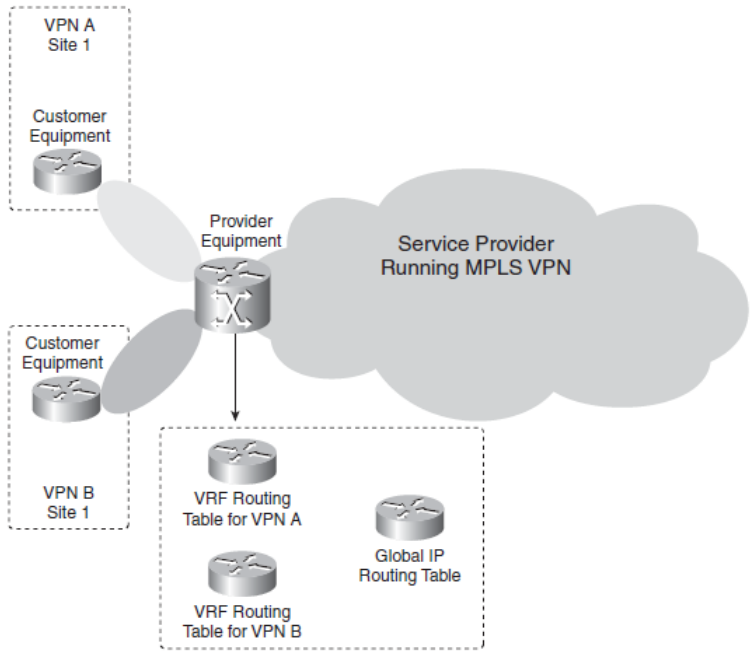

Figure 2.4: VRFs on a PE Router

As each VPN should have its own routing table, a PE router has a private routing table called the VRF routing table. The VRF on the PE router can be created with the **ip vrf** command. The command **ip vrf forwarding** is used to assign PE-CE interfaces on the PE router to a VRF.

RD A VPNv4 (or VPN-IPv4) route comprises of 8-byte Route-Distinguisher (RD) and 4-byte IPv4 address. When a PE router receives an IPv4 prefix, it translates it into VPNv4 prefixes. So if the same address space is used in different VPNs, it is possible for BGP to carry completely different routes to that address, one for each VPN. An RD is simply a number; it does not provide any information. It is only used to translate an IPv4 prefix into VPNv4 prefix, making same IPv4 prefix a completely different VPNv4 prefix, allowing BGP to distribute these VPNv4 prefixes. An RD is 64 bits in length comprising three fields: type (two bytes), administrator, and value.

RTs Route Target (RT) is an MPLS feature used to control the communication between different sites. An RT is a BGP extended community. It indicates which routes should be imported

from MP-BGP into the VRF. When we export an RT it means that the exported vpnv4 route receives an additional BGP extended community (RT) as configured under *ip vrf* on the PE router, when the route is redistributed from the VRF routing table into MP-BGP. When we import an RT it means that the received vpnv4 route from MP-BGP is checked for a matching extended community (RT) with the ones in the configuration. The prefix is put into the VRF routing table as an IPv4 route if the result is a match. The prefix is rejected if the match does not occur. The command to configure RTs for a VRF is **route-target {import | export | both}** *route-target-extcommunity*. The keyword **both** indicates both import and export.

2.1.8.2 BGP:

Border Gateway Protocol (BGP) is the standard protocol for interdomain routing. As BGP allows flexible and extended policies to be implemented, it is ideal to carry MPLS VPN routes and we will be using it to implement MPLS VPN in our proposed network. The VPNv4 prefix made up by the combination of the RD with the IPv4 prefix needs to be carried by iBGP between the PE routers.

The BGP routing process has the concept of address families to support the Multiprotocol behavior of BGP in Cisco IOS. The address family vpnv4 under the router bgp process is used to configure the vpnv4 BGP sessions and parameters, which the PE routers need. The address family ipv4 vrf *vrf-name* under the router bgp process on the PE routers is used to configure the BGP sessions and parameters toward the CE routers, across the VRF interfaces.

For the address family vpnv4 configuration, the BGP neighbor needs to be defined in the global part of the BGP configuration. Then the BGP neighbor needs to be enabled in the address family vpnv4 by specifying the *activate* keyword.

2.1.8.3 IS-IS:

Intermediate System to Intermediate System (IS-IS) is a link state routing protocol like OSPF. It runs directly over Layer2, not over IP. It can be configured with the command **router isis**. To configure a particular interface to be able to use IS-IS, the command **ip router isis** is used on that interface. We will be using IS-IS as the routing protocol to create the routing link between PE

and P routers in our proposed network.

2.1.8.4 PE-CE Routing Protocols:

PE and CE routers need to have routing between them. There are many routing protocols that can be used for this purpose but we will be using Open Shortest Path First (OSPF) protocol.

OSPF is redistributed into iBGP and vice versa on the PE routers to propagate the customer routes from PE to PE. OSPF process command is configured with the VRF keyword to run OSPF for a VRF. The syntax is **router ospf** *process-id* **vrf** *vrf-name*. Also, the OSPF VRF process needs to be redistributed into BGP and vice versa. All the regular OSPF commands can be configured for the OSPF VRF process.

2.2 Python

We will be using Python as the router scripting language for getting the VRF data from the routers and parsing the data retrieved to get the desired information to display on a web page. Python is a powerful high level programming language. It has efficient high-level data structures and an effective approach to object-oriented programming. It is an ideal language for scripting and rapid application development in many areas on most platforms because of its elegant syntax and dynamic typing, together with its interpreted nature.

The Python interpreter and the extensive standard library can be freely downloaded in source or binary form for all major platforms from the Python Web site, `https://www.python.org`, and may be freely distributed. You can also find distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation from the same site.

The version of Python that we will be using is Python 2.7.6.

2.2.1 Paramiko

In order to connect to a router from an external device, we need to enable secure shell (SSH) on the router and make an SSH connection into the router. SSH is a cryptographic network protocol

for making a secure data connection. In a client-server architecture, it establishes a secure channel over an insecure network by connecting an SSH client application with an SSH server.

We will use **Paramiko** module of Python to make an SSH connection into the router from our web framework. Paramiko is a Python implementation of the SSHv2 protocol and it provides both client and server functionality. Paramiko is a pure Python interface around SSH networking concepts besides leveraging a Python C extension for low level cryptography. The high-level client API starts with creation of an **SSHClient** object. As a client, you are responsible for authenticating using a password or private key, and checking the server's host key. Paramiko can be installed on a Linux machine by using **pip**:
$ pip install paramiko

2.3 Django

We will use Django web-based framework to provide a web based front end to our application displaying the router VRF information. Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. It takes care of much of the hassles of web development, so that the developer can focus on writing the application without needing to start Python programming from scratch. It is free and open source.

For our project, we will use and install Django version 1.7.4. In the following sections, we will have a look at the key terminologies, modules and knowledge areas pertaining to developing a web application using Django.

2.3.1 Django Project

The first step in developing a Django application is creating a project. A project is a collection of settings for an instance of Django, including database configuration, Django-specific options and application-specific settings. It can be created by using the command **django-admin**.**py start-project** *project-name*.

In order to start the Django development server, we need to navigate to the container directory

and run the command **python manage.py runserver**.

Django has been referred to as an MTV framework. In the MTV development pattern:

- M stands for Model, the data access layer. This layer defines everything about the data: its access, validation, its behavior and the relationships between the data.
- T stands for Template, the presentation layer. This layer defines presentation-related information that covers how to display data on a web page.
- V stands for View, the business logic layer. This layer acts as the bridge between models and templates by defining the logic to access the model and presenting the appropriate templates.

We will discuss each of these in the coming sections.

2.3.2 Views and URLconfs

In Django, the contents of the page are produced by a view function, and the URL is specified in a URLconf.

Within the directory created after starting the project, an empty file called *views.py* is created. The contents of the page are generated with coding in this module. We will retrieve and parse the VRF information from the router after making SSH connection in this module.

On the other hand, a URLconf is like a table of contents for your Django-powered Web site. It is a mapping between URLs and the view functions that should be called for those URLs. Another file *urls.py* gets created after you start the project. A mapping is added between a URL pattern and the view function in order to add a URL and view to the URLconf.

2.3.3 Templates

A Django template is defined as a string of text that is intended to separate the presentation of a document from its data. A template defines placeholders and various bits of basic logic (template

tags) that regulate how the document should be displayed. In general, templates are used for producing HTML, but Django templates can generate any text-based format. We will use HTML templates in Django to display the contents of *views.py*.

In order to load the templates, we need to tell the framework where we store our templates. To do that, we need to change *settings.py* (a file generated after starting the project) and add the template directory to TEMPLATE_DIRS tuple.

2.3.4 Models

A Django model is a representation of the description of the data in our database by using Python code. It is the layout of the data−the equivalent of SQL *CREATE TABLE* statement but it is in Python instead of SQL. It defines more than just database column definitions. Django uses a model to execute SQL code behind the scenes and return convenient Python data structures representing the rows in your database tables.

We need to configure our database by changing the *settings.py* and adding the database details (MySQL in our case) in the DATABASES tuple. In order to create a model, we first need to create an *app* in Django project. An app is a portable set of Django functionality, usually including models and views, that lives together in a single Python package. We can create an app with the command **python manage.py startapp** *app-name*. This command creates an *app-name* directory with files naming *__init__.py*, *models.py*, *tests.py* and *views.py*. We can create the models for our application by modifying *models.py* file. The following commands are regularly used to validate, migrate and update the models with the backend database:

- **python manage.py validate** to validate the models after making any changes to them.
- **python manage.py migrate** to apply migrations as well as unapplying and listing their status.
- **python manage.py makemigrations** to create new migrations based on the changes you have made to your models.

2.3.5 Django Admin Site

Django provides an automatic admin interface that enables the adding, editing and deletion of site content. The feature reads metadata in the model to provide a powerful and production-ready interface that site administrators can use. We will be using this admin interface to view stored VRF information of routers if the user wants to view it in non real-time mode. We need to make sure that the admin interface is activated before adding our model to the admin site. In order to do that we need to modify *admin.py* file in the *app-name* directory by adding the following lines:
from *project-name*.*app-name*.**models import** *model-name*
admin.**site**.**register(***model-name***)**

This tells the Django admin site to offer an interface for this model. To view the admin site, run the development server and visit `http://127.0.0.1:8000/admin/` in your Web browser. A *model-name* section can be seen beside the other links for site administration.

2.4 MySQL Database

We will be using MySQL Database as the backend for Django models to store the router VRF information. The MySQL software delivers a very fast, multi-threaded, multi-user, and robust SQL (Structured Query Language) database server. It is widely used in web applications. We will use MySQL server version 14.14 distribution 5.5.41 for our application. After making sure that the MySQL server is running in our system, we need to login into the server and create our database by using the command **CREATE DATABASE** *database-name*. After the database is created, it can be configured in Django framework to be used with Django models.

$\sim$

Chapter 3

Network Model and Implementation

This chapter describes simulation setup for the proposed network and the environment to create the web based application to display the router information. Introduction about framework that is used to support the proposed network model and different components that are being used in it.

3.1 Simulation Framework

MPLS VPN network will be created in GNS3, a virtual network development environment to build networks. The PE routers in the MPLS VPN will accessed from the GNS3 from a Ubuntu Virtual Machine. The web application will be set up and used in this Ubuntu Virtual Machine. A brief description about these in given in the coming sections.

3.1.1 GNS3

GNS3 is a software tool which acts as an alternative to using real computer labs for computer network engineers, administrators and network engineering students. It can also be used to experiment new features or to check configurations that need to be deployed later on real devices. Other features of GNS3 include connection of the virtual network to real ones or packet captures using Wireshark. GNS3 provides a graphical user interface to design and configure virtual networks. It can be used on multiple operating systems, including Windows, Linux, and Mac OS X and it runs on traditional PC hardware.

GNS3 uses the following emulators to run the very same operating systems as in real networks in order to provide complete and accurate simulations:

- **Dynamips** to emulate Cisco IOS.
- **VirtualBox** to provide virtual environment to run desktop and server operating systems.
- **Qemu** a generic open source machine emulator.

To simulate our network, we will use GNS3 version 1.2.3 under GPL v3 license. All the routers used in the network will be Cisco 2691 with *c2691-adventerprisek9-mz.124-5a* image.

3.1.2 Ubuntu Virtual Machine

Ubuntu is a Debian based Linux Operating Sytem and its default desktop environment is Unity. We will be using Ubuntu Virtual Machine 14.04 LTS in VirtualBox, a virtualization software package, in order to install and use Django framework for web application deployment. Ubuntu 14.04 LTS comes with python 2.7.6 and we need to install Paramiko module of Python. We also need to install Django and MySQL server.

$\sim$

Chapter 4

Simulation Setup

GNS3 and Ubuntu 14.04 LTS are used to implement the MPLS VPN network and the proposed web application.

4.1 Network Topology

In order to implement the proposed project, an MPLS VPN network will be created according to the following network diagram:

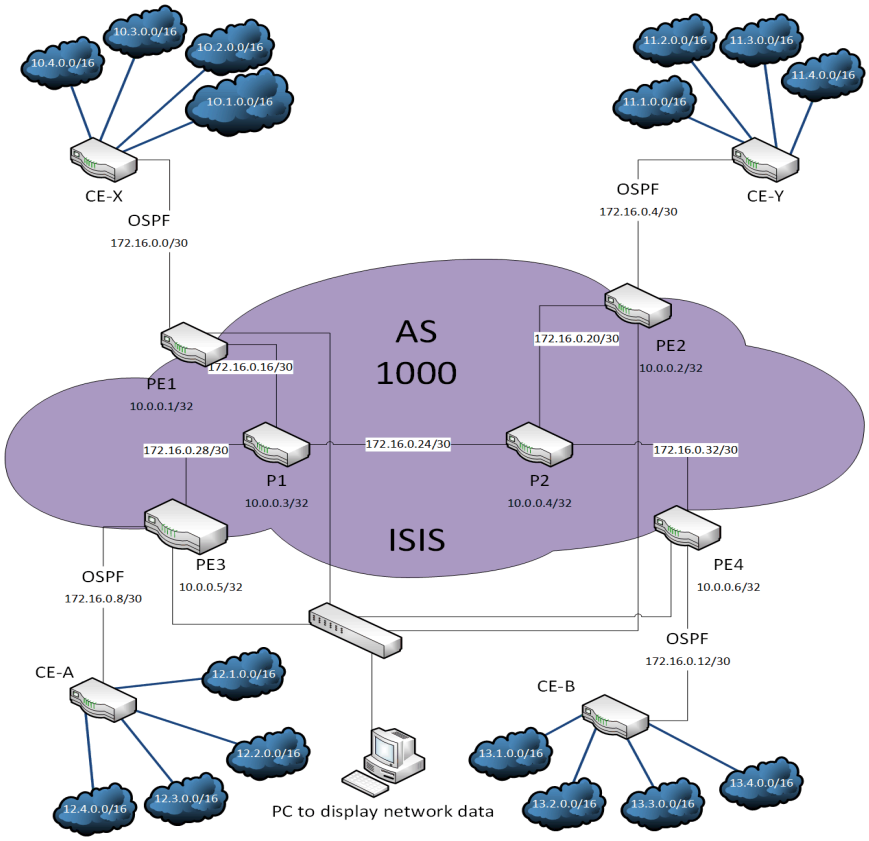

Figure 4.1: Network Topology

Customer Locations: There will be four customer networks - X, Y, A and B which will be connected to the MPLS network through their respective customer edge routers - CE-X, CE-Y, CE-A and CE-B. The networks connected to respective customers will be:

Customer	Networks
X	10.1.0.0/16, 10.2.0.0/16, 10.3.0.0/16, 10.4.0.0/16
Y	11.1.0.0/16, 11.2.0.0/16, 11.3.0.0/16, 11.4.0.0/16
A	12.1.0.0/16, 12.2.0.0/16, 12.3.0.0/16, 12.4.0.0/16
B	13.1.0.0/16, 13.2.0.0/16, 13.3.0.0/16, 13.4.0.0/16

Table 4.1: Customer Networks

MPLS Network: We will be using BGP (AS 1000) to implement MPLS VPN in PE routers in our proposed network. All the provider routers will be connected using IS-IS protocol to enable routing among PE and P routers. Four provider edge routers - PE1, PE2, PE3 and PE4 will connect the MPLS network to the customer networks. MPLS LDP must be enabled in all the PE and P routers. OSPF will be used as PE-CE routing protocol and it will also help to redistribute customer networks into the BGP running in MPLS network. The following VRFs with RDs need to be defined on the PE routers according to the customer locations they are connected to:

PE router	VRF name	Route Distinguisher
PE1	customerX_1	1000:1
PE2	customerY_1	1000:2
PE3	customerA_1	1000:3
PE4	customerB_1	1000:4

Table 4.2: VRFs on PE routers

4.2 Network Scenario

In this section, we discuss the network scenario we will use to run the MPLS network. On each PE router, we will export selective local VRF routes as well as import selective VRF routes from other customer locations. This is done according to the parameters laid out in the following table:

PE router	Customer name	Import	Export
PE1	Customer X	Customer Y, Customer A, Customer B	10.1.0.0/16, 10.3.0.0/16
PE2	Customer Y	Customer A, Customer B	11.2.0.0/16
PE3	Customer A	Customer X, Customer B	12.2.0.0/16
PE4	Customer B	Customer Y, Customer A	13.2.0.0/16

Table 4.3: VRFs to import and export

After configuriing the VRF export and import routes according to the above scenario, the VRF routes on PE routers will look like this:

PE1 router:

Routing Table: customerA_1

12.0.0.0/32 is subnetted, 1 subnets

B 12.2.0.1 [200/11] via 10.0.0.5, 1d19h

Routing Table: customerB_1

13.0.0.0/32 is subnetted, 1 subnets

B 13.2.0.1 [200/11] via 10.0.0.6, 1d00h

Routing Table: customerX_1

172.16.0.0/30 is subnetted, 1 subnets

C 172.16.0.0 is directly connected, FastEthernet0/0

10.0.0.0/32 is subnetted, 4 subnets

O 10.3.0.1 [110/11] via 172.16.0.1, 1d21h, FastEthernet0/0

O 10.2.0.1 [110/11] via 172.16.0.1, 1d21h, FastEthernet0/0

O 10.1.0.1 [110/11] via 172.16.0.1, 1d21h, FastEthernet0/0

O 10.4.0.1 [110/11] via 172.16.0.1, 1d21h, FastEthernet0/0

Routing Table: customerY_1

11.0.0.0/32 is subnetted, 1 subnets

B 11.2.0.1 [200/11] via 10.0.0.2, 1d20h

PE2 router:

Routing Table: customerA_1

12.0.0.0/32 is subnetted, 1 subnets

B 12.2.0.1 [200/11] via 10.0.0.5, 1d19h

Routing Table: customerB_1

13.0.0.0/32 is subnetted, 1 subnets

B 13.2.0.1 [200/11] via 10.0.0.6, 1d00h

Routing Table: customerX_1

10.0.0.0/32 is subnetted, 2 subnets

B 10.3.0.1 [200/11] via 10.0.0.1, 1d20h

B 10.1.0.1 [200/11] via 10.0.0.1, 1d20h

Routing Table: customerY_1

172.16.0.0/30 is subnetted, 1 subnets

C 172.16.0.4 is directly connected, FastEthernet0/0

11.0.0.0/32 is subnetted, 4 subnets

O 11.2.0.1 [110/11] via 172.16.0.5, 1d21h, FastEthernet0/0

O 11.3.0.1 [110/11] via 172.16.0.5, 1d21h, FastEthernet0/0

O 11.1.0.1 [110/11] via 172.16.0.5, 1d21h, FastEthernet0/0

O 11.4.0.1 [110/11] via 172.16.0.5, 1d21h, FastEthernet0/0

PE3 router:

Routing Table: customerA_1

172.16.0.0/30 is subnetted, 1 subnets

C 172.16.0.8 is directly connected, FastEthernet0/0

12.0.0.0/32 is subnetted, 4 subnets

O 12.4.0.1 [110/11] via 172.16.0.10, 1d21h, FastEthernet0/0

O 12.1.0.1 [110/11] via 172.16.0.10, 1d21h, FastEthernet0/0

O 12.3.0.1 [110/11] via 172.16.0.10, 1d21h, FastEthernet0/0

O 12.2.0.1 [110/11] via 172.16.0.10, 1d21h, FastEthernet0/0

Routing Table: customerB_1

13.0.0.0/32 is subnetted, 1 subnets

B 13.2.0.1 [200/11] via 10.0.0.6, 1d00h

Routing Table: customerX_1

10.0.0.0/32 is subnetted, 2 subnets

B 10.3.0.1 [200/11] via 10.0.0.1, 1d19h

B 10.1.0.1 [200/11] via 10.0.0.1, 1d19h

PE4 router:

Routing Table: customerA_1
12.0.0.0/32 is subnetted, 1 subnets
B 12.2.0.1 [200/11] via 10.0.0.5, 1d00h

Routing Table: customerB_1
13.0.0.0/32 is subnetted, 1 subnets
B 13.2.0.1 [200/11] via 10.0.0.6, 1d00h

Routing Table: customerX_1
172.16.0.0/30 is subnetted, 1 subnets
C 172.16.0.12 is directly connected, FastEthernet0/0
13.0.0.0/32 is subnetted, 4 subnets
O 13.4.0.1 [110/11] via 172.16.0.14, 1d21h, FastEthernet0/0
O 13.1.0.1 [110/11] via 172.16.0.14, 1d21h, FastEthernet0/0
O 13.2.0.1 [110/11] via 172.16.0.14, 1d21h, FastEthernet0/0
O 13.3.0.1 [110/11] via 172.16.0.14, 1d21h, FastEthernet0/0

Routing Table: customerY_1
11.0.0.0/32 is subnetted, 1 subnets
B 11.2.0.1 [200/11] via 10.0.0.2, 1d00h

Now that we have the desired VRF routes on the PE routers, we need to get this information and display it using our Django web interface. For that we need to connect these PE routers to the Ubuntu machine using a switch as shown in the network topology. The network topology of our MPLS VPN network as viewed in GNS3:

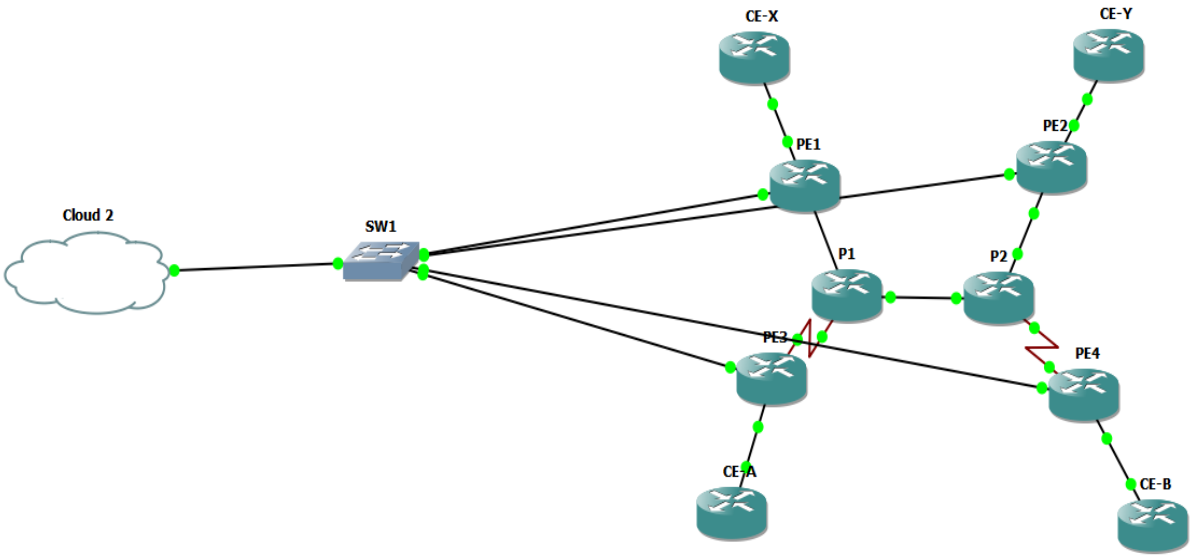

Figure 4.2: Network Topology in GNS3

The details of how to create the web application to display VRF information is explained in the next section.

4.3 Django Web Application

This section covers the implementation of Django Web-based framework to provide a web based front end to our application displaying the router VRF information. We start by creating a Django project named *MPLS_Network*. In that project, we create an app named *vrf* that will create the MTV framework for our project and enable us to build and modify according to our project requirements. We will discuss the Django Views, Models, Templates and Admin site configurations specific to our project in the coming sections.

4.3.1 Models

We will define our data structures in the *models.py* file. The following code will create a model **vrfs** for our VRF routes and also create a function **create** to be called to create VRF records:

```
from django.db import models

class vrfs(models.Model):
        router_id = models.CharField(max_length = 60)
        customer_name = models.CharField(max_length = 50)
        network = models.CharField(max_length = 100)
        protocol = models.CharField(max_length = 30)
        nexthop = models.CharField(max_length = 100)
        timestamp = models.DateTimeField()
        @classmethod
        def create(cls, router_id, customer_name,network,protocol,nexthop, timestamp):
                vrf_record=cls(router_id=router_id,customer_name=customer_name,network=
                    network,protocol=protocol,nexthop=nexthop, timestamp=timestamp)
                return vrf_record
```

After that, a database named **mpls_network** needs to be created in MySQL database. Then the DATABASES tuple in *settings.py* needs to be modified to configure **mpls_network** database with Django framework before we migrate and synchronise Django model **vrfs** with the database:

```
DATABASES = {
        'default': {
                'ENGINE': 'django.db.backends.mysql',
                'NAME': 'mpls_network',
                'USER': 'root',
                'PASSWORD': 'root',
                 }
         }
```

4.3.2 Views

The *views.py* file needs to modified to get the VRF routes from the router using Paramiko SSH, save that information in the database and pass the same information to the templates in order to display the information at the front end. The *views.py* needs to get the login information from the html template **login**.**html** - the login html page. After making the SSH authentication and login into the intended router, the VRF routes are retrieved from the router. The following code does the same:

```
import paramiko


ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(router_id, username=username, password=password)
stdin, stdout, stderr = ssh.exec_command("show ip route vrf * | e -")
routes=(stdout.readlines())
ssh.close()
```

In the above code, **router_id** is the ip address of the router interface that is used for secure shell, **username** is the router username and **password** is the router password. The data in the **routes** variable needs to be parsed to get the routes information. The information about the routes that

need to be retrieved by parsing **routes** variable is:

- **Customer Name**
- **Customer Network**
- **Protocol**
- **Nexthop**

This information needs to be saved in the database by calling the *vrfs* (as explained in the previous section) class function *create* in *models.py* to create a record of VRF routes.

```
from vrf.models import vrfs

new_vrf=vrfs.create(router_id,customer_name, customer_network, protocol, nexthop, timestamp)
new_vrf.save()
```

After the above information is saved, it is passed to the html template **loggedin**.**html** which is the html web page that displays all the VRF routes information.

4.3.3 Templates

We need to create HTML templates **login**.**html** to get the login information using web page and **loggedin**.**html** to display the VRF routes on the web page. In order to add the mapping between the URLs of these templates and the corresponding functions in *views.py* that provide data to these templates, modifications are made in *urls.py* file. The templates directory where we place all our templates is configured in *settings.py* file:

```
TEMPLATE_DIRS = ("vrf/templates",)
```

4.3.4 Django Admin Site

Django admin interface will be used to view stored VRF information of routers if the user wants to view it in non real-time mode. In order to activate it and add **vrfs** model in the admin site, we

need to modify *admin.py* file in the *vrf* directory by adding the following lines:

```
from django.contrib import admin
from vrf.models import vrfs


admin.site.register(vrfs)
```

The stored VRF routes can now be viewed by visiting `http://127.0.0.1:8000/admin/` in the web browser.

~

Chapter 5

Working

This chapter covers the working of the web based interface we implemented to present the VRF routes generated in the MPLS VPN network.

5.1 Running Django Server

The Django server needs to be running before we can view any web page related to the web application. Migrate all the models to the database and validate the server. After that, run the server by using the command **python manage.py runserver** in a terminal.

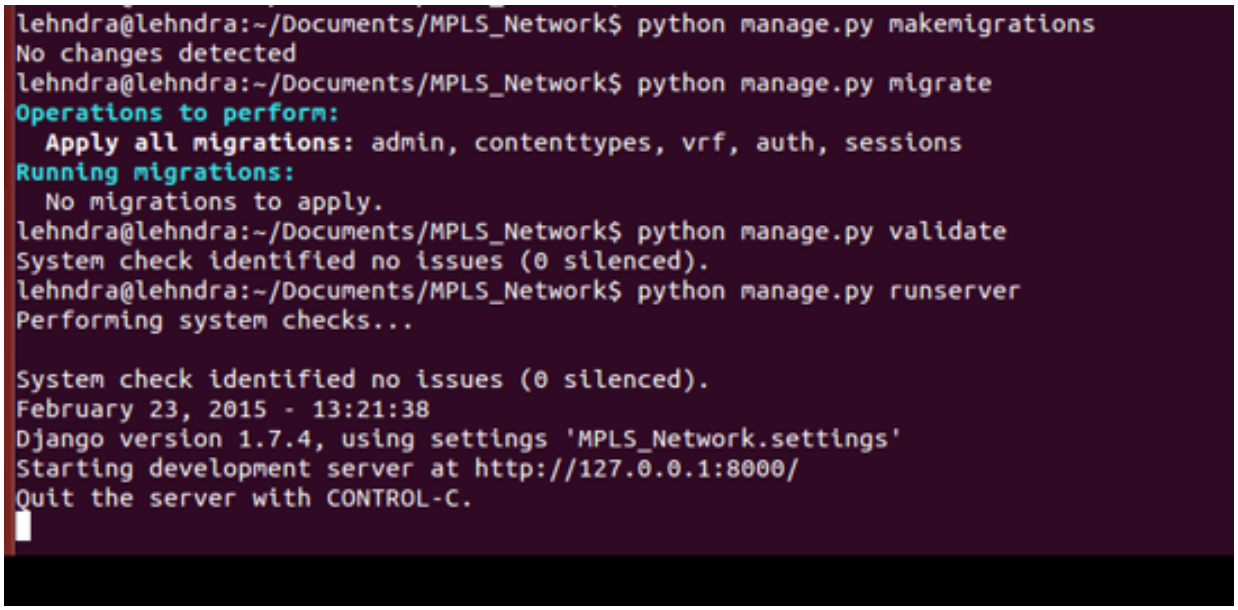

Figure 5.1: Running Django Server

As seen in the above figure, server is running at `http://127.0.0.1:8000`.

5.2 Web Interface

After the Django server is up and running, we can open a web browser and view `http://127.0.0.1:8000/login` page to login to any PE routers.

PE routers have the following IP address configured on their respective interfaces connected to the Ubuntu machine on which the Django server is running:

- **PE1** - 192.168.1.2
- **PE2** - 192.168.1.4
- **PE3** - 192.168.1.5
- **PE4** - 192.168.1.6

We can use these IP addresses to login to these routers using our web interface. We can start by PE1 that has 192.168.1.2 IP address:

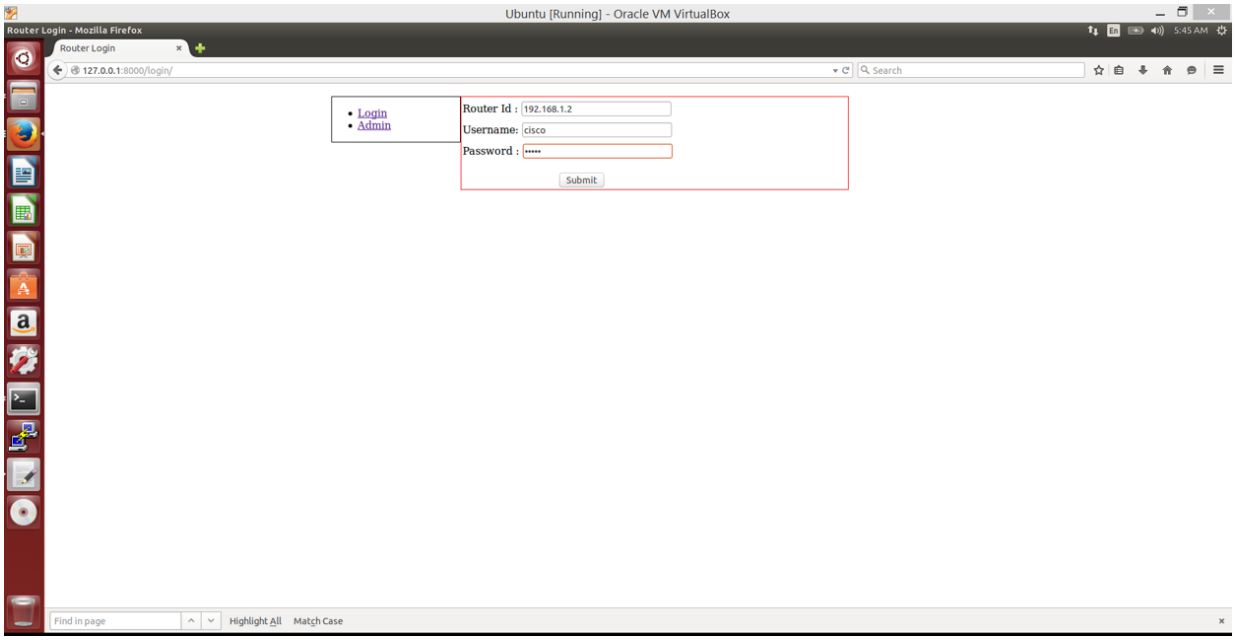

Figure 5.2: Router Login Page

After successful login, we can view the web page that displays the VRF routes on PE1 router including local VRF routes and also the VRF routes coming from other locations showing the next hop, protocol used and customer location from where the VRF routes are originating.

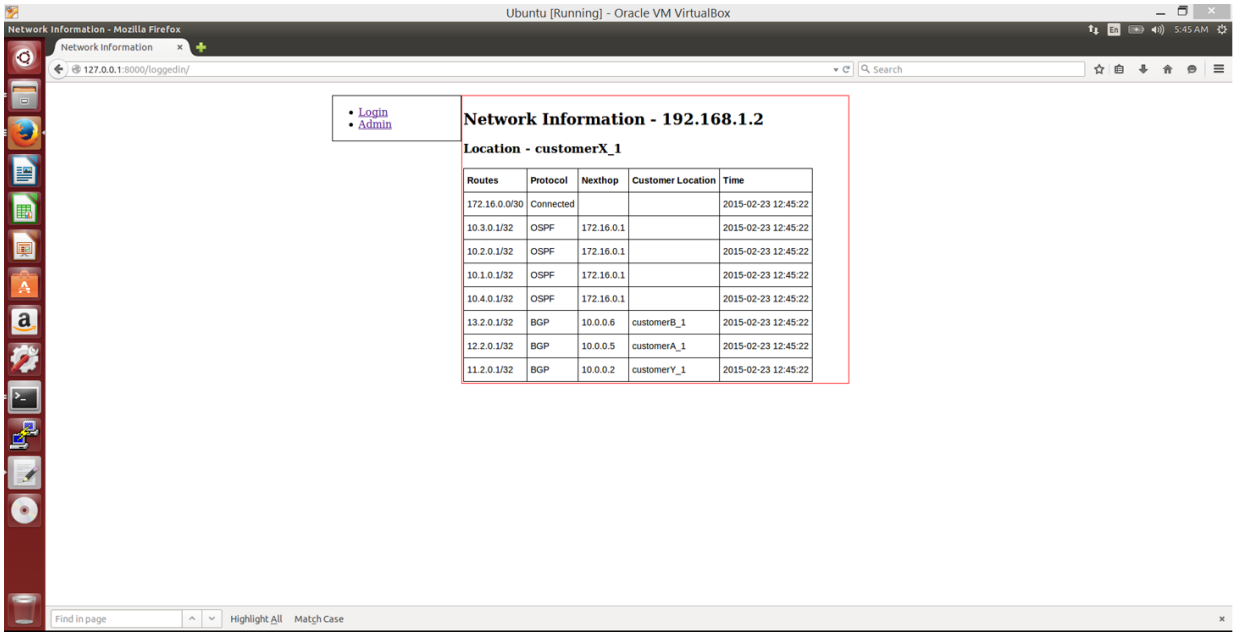

Figure 5.3: VRF routes at Customer X

Similarly, we can view VRF routes at locations of Customer Y, A and B by logging into the PE2, PE3 and PE4 routers.

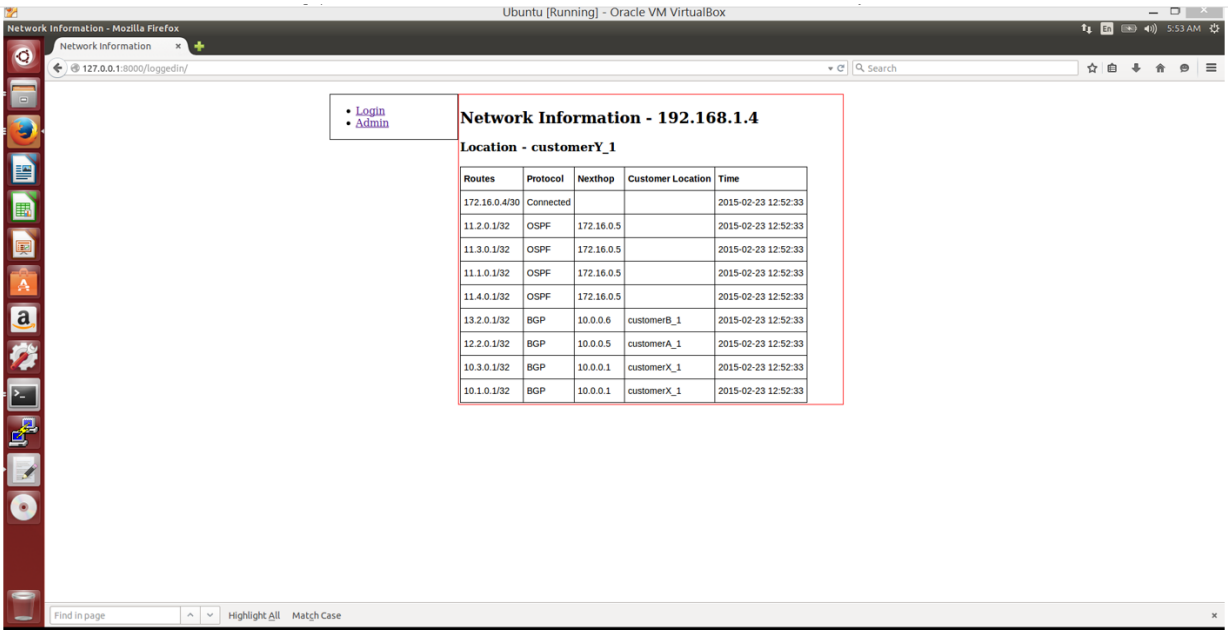

Figure 5.4: VRF routes at Customer Y

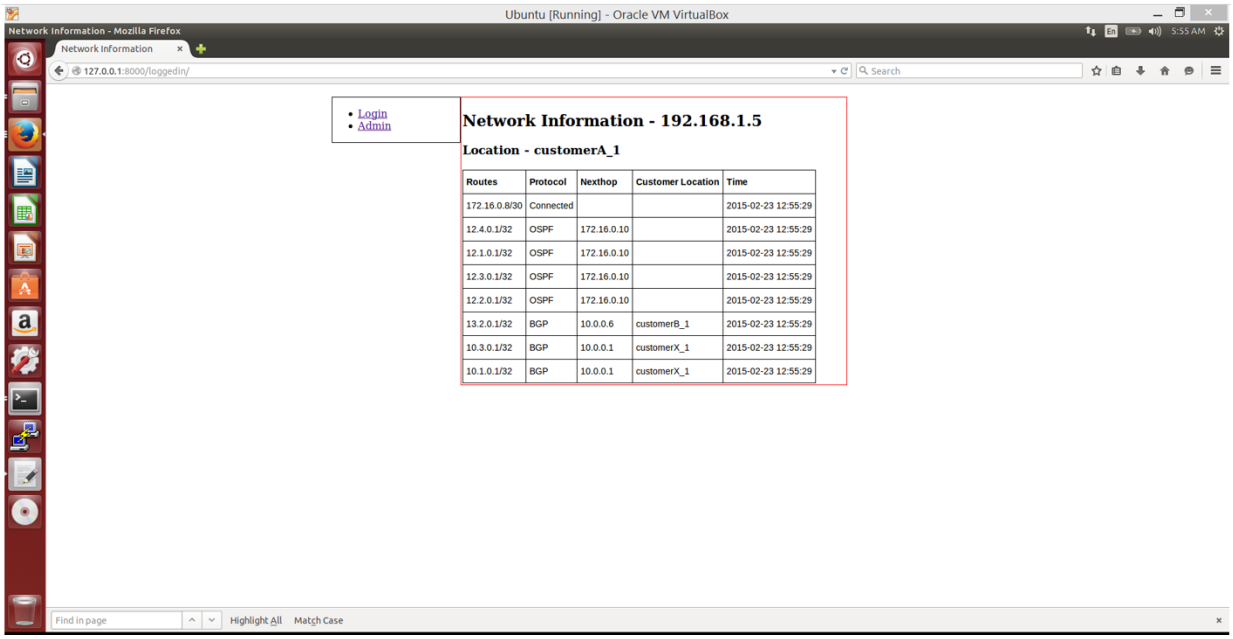

Figure 5.5: VRF routes at Customer A

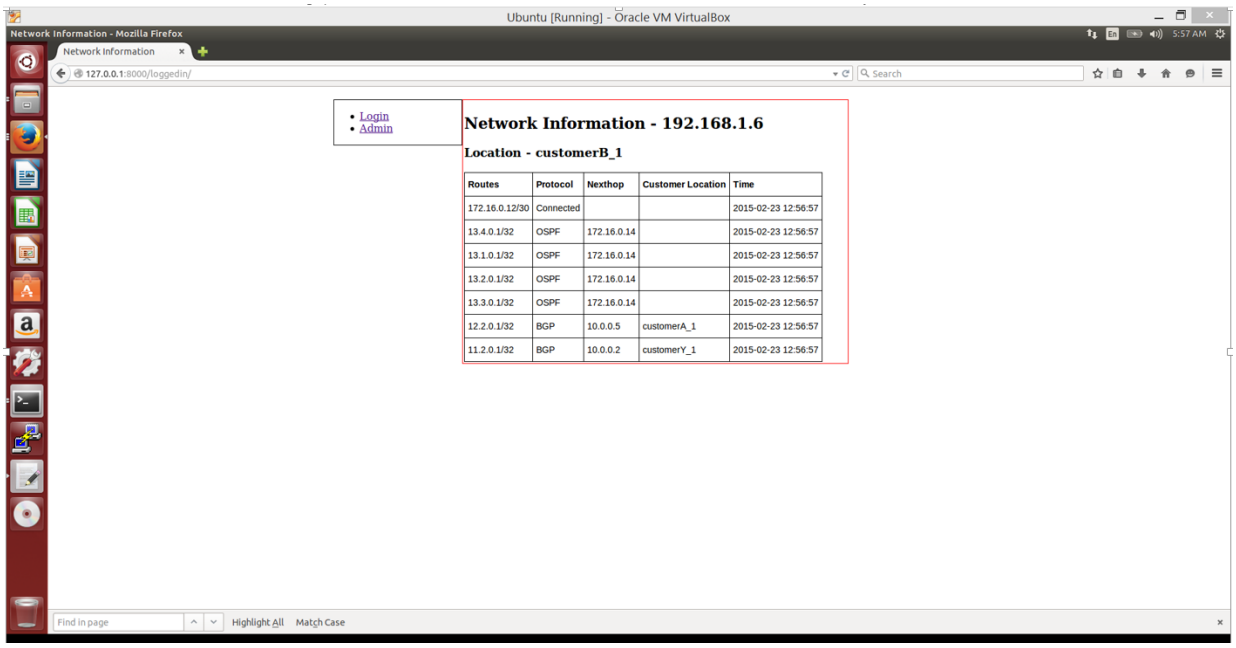

Figure 5.6: VRF routes at Customer B

In the next sections, we will have a look at the VRF routes stored in the MySQL database.

5.3 Connecting to MySQL database

We can login and connect to MySQL database **mpls_network** in order to directly view the stored VRF routes in the database.

```
lehndra@lehndra:~$ sudo mysql -u root -p
[sudo] password for lehndra:
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 181
Server version: 5.5.41-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> connect mpls_network
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Connection id:    182
Current database: mpls_network
```

Figure 5.7: Connecting to MySQL database

As the model **vrfs** that stores all the VRF information was created in **vrf** app, we can view the stored rows at the backend using the SQL query **select * from vrf_vrfs;**.

```
mysql> select * from vrf_vrfs;
+-----+---------------+----------------+-----------+-------------+---------------------+-------------+
| id  | customer_name | network        | protocol  | nexthop     | timestamp           | router_id   |
+-----+---------------+----------------+-----------+-------------+---------------------+-------------+
| 182 | customerB_1   | 13.2.0.1/32    | BGP       | 10.0.0.6    | 2015-02-23 12:45:22 | 192.168.1.2 |
| 183 | customerY_1   | 11.2.0.1/32    | BGP       | 10.0.0.2    | 2015-02-23 12:45:22 | 192.168.1.2 |
| 184 | customerA_1   | 12.2.0.1/32    | BGP       | 10.0.0.5    | 2015-02-23 12:45:22 | 192.168.1.2 |
| 185 | customerX_1   | 172.16.0.0/30  | Connected |             | 2015-02-23 12:45:22 | 192.168.1.2 |
| 186 | customerX_1   | 10.3.0.1/32    | OSPF      | 172.16.0.1  | 2015-02-23 12:45:22 | 192.168.1.2 |
| 187 | customerX_1   | 10.2.0.1/32    | OSPF      | 172.16.0.1  | 2015-02-23 12:45:22 | 192.168.1.2 |
| 188 | customerX_1   | 10.1.0.1/32    | OSPF      | 172.16.0.1  | 2015-02-23 12:45:22 | 192.168.1.2 |
| 189 | customerX_1   | 10.4.0.1/32    | OSPF      | 172.16.0.1  | 2015-02-23 12:45:22 | 192.168.1.2 |
| 190 | customerB_1   | 13.2.0.1/32    | BGP       | 10.0.0.6    | 2015-02-23 12:52:33 | 192.168.1.4 |
| 191 | customerY_1   | 172.16.0.4/30  | Connected |             | 2015-02-23 12:52:33 | 192.168.1.4 |
| 192 | customerY_1   | 11.2.0.1/32    | OSPF      | 172.16.0.5  | 2015-02-23 12:52:33 | 192.168.1.4 |
| 193 | customerY_1   | 11.3.0.1/32    | OSPF      | 172.16.0.5  | 2015-02-23 12:52:33 | 192.168.1.4 |
| 194 | customerY_1   | 11.1.0.1/32    | OSPF      | 172.16.0.5  | 2015-02-23 12:52:33 | 192.168.1.4 |
| 195 | customerY_1   | 11.4.0.1/32    | OSPF      | 172.16.0.5  | 2015-02-23 12:52:33 | 192.168.1.4 |
| 196 | customerA_1   | 12.2.0.1/32    | BGP       | 10.0.0.5    | 2015-02-23 12:52:33 | 192.168.1.4 |
| 197 | customerX_1   | 10.3.0.1/32    | BGP       | 10.0.0.1    | 2015-02-23 12:52:33 | 192.168.1.4 |
| 198 | customerX_1   | 10.1.0.1/32    | BGP       | 10.0.0.1    | 2015-02-23 12:52:33 | 192.168.1.4 |
| 199 | customerB_1   | 13.2.0.1/32    | BGP       | 10.0.0.6    | 2015-02-23 12:55:29 | 192.168.1.5 |
| 200 | customerA_1   | 172.16.0.8/30  | Connected |             | 2015-02-23 12:55:29 | 192.168.1.5 |
| 201 | customerA_1   | 12.4.0.1/32    | OSPF      | 172.16.0.10 | 2015-02-23 12:55:29 | 192.168.1.5 |
| 202 | customerA_1   | 12.1.0.1/32    | OSPF      | 172.16.0.10 | 2015-02-23 12:55:29 | 192.168.1.5 |
| 203 | customerA_1   | 12.3.0.1/32    | OSPF      | 172.16.0.10 | 2015-02-23 12:55:29 | 192.168.1.5 |
| 204 | customerA_1   | 12.2.0.1/32    | OSPF      | 172.16.0.10 | 2015-02-23 12:55:29 | 192.168.1.5 |
| 205 | customerX_1   | 10.3.0.1/32    | BGP       | 10.0.0.1    | 2015-02-23 12:55:29 | 192.168.1.5 |
| 206 | customerX_1   | 10.1.0.1/32    | BGP       | 10.0.0.1    | 2015-02-23 12:55:29 | 192.168.1.5 |
| 207 | customerB_1   | 172.16.0.12/30 | Connected |             | 2015-02-23 12:56:57 | 192.168.1.6 |
| 208 | customerB_1   | 13.4.0.1/32    | OSPF      | 172.16.0.14 | 2015-02-23 12:56:57 | 192.168.1.6 |
| 209 | customerB_1   | 13.1.0.1/32    | OSPF      | 172.16.0.14 | 2015-02-23 12:56:57 | 192.168.1.6 |
| 210 | customerB_1   | 13.2.0.1/32    | OSPF      | 172.16.0.14 | 2015-02-23 12:56:57 | 192.168.1.6 |
| 211 | customerB_1   | 13.3.0.1/32    | OSPF      | 172.16.0.14 | 2015-02-23 12:56:57 | 192.168.1.6 |
| 212 | customerY_1   | 11.2.0.1/32    | BGP       | 10.0.0.2    | 2015-02-23 12:56:57 | 192.168.1.6 |
| 213 | customerA_1   | 12.2.0.1/32    | BGP       | 10.0.0.5    | 2015-02-23 12:56:57 | 192.168.1.6 |
+-----+---------------+----------------+-----------+-------------+---------------------+-------------+
32 rows in set (0.00 sec)

mysql>
```

Figure 5.8: Stored VRFs in database

5.4 Django Admin Site

We can also view the stored VRF routes using Django admin site. Open the page `http://127.0.0.1:8000/admin` and login using the login credentials of the Ubuntu machine.

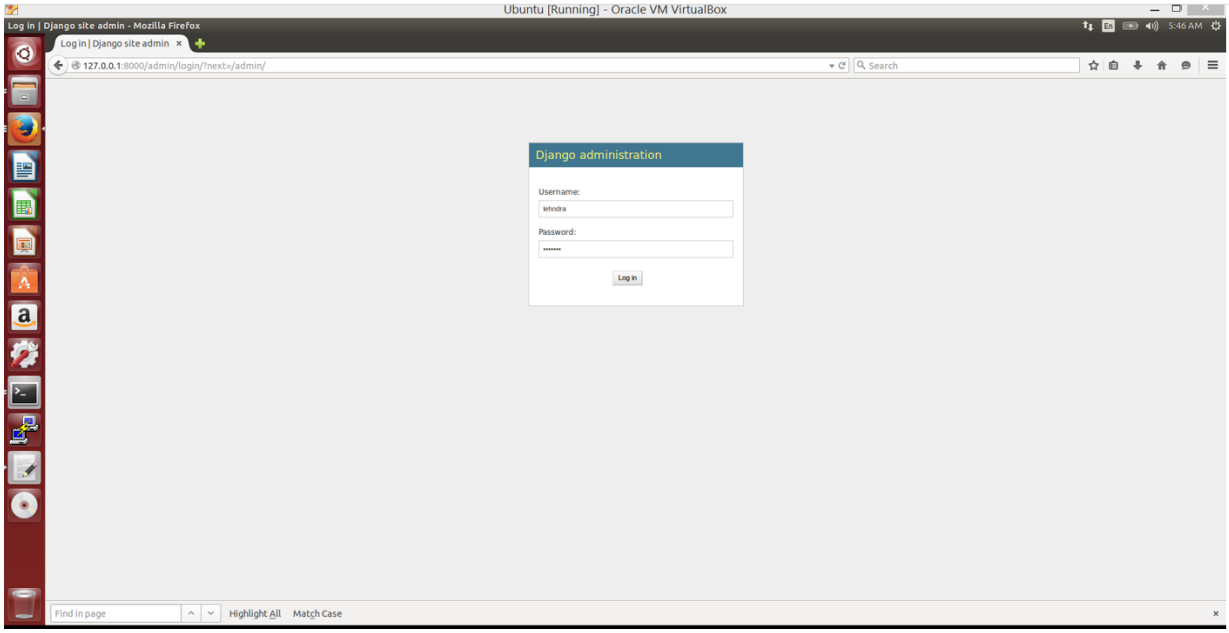

Figure 5.9: Admin Login Page

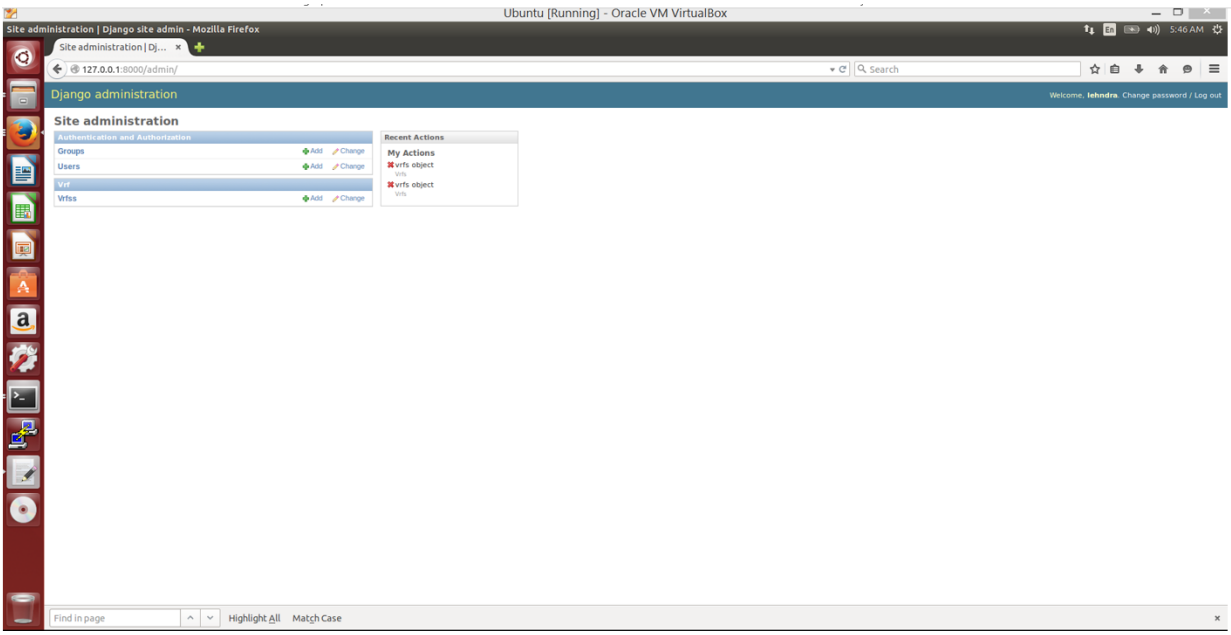

Figure 5.10: Admin Logged In Page

After logging in as admin, we can go the link named **Vrfs** to view the stored VRF objects.

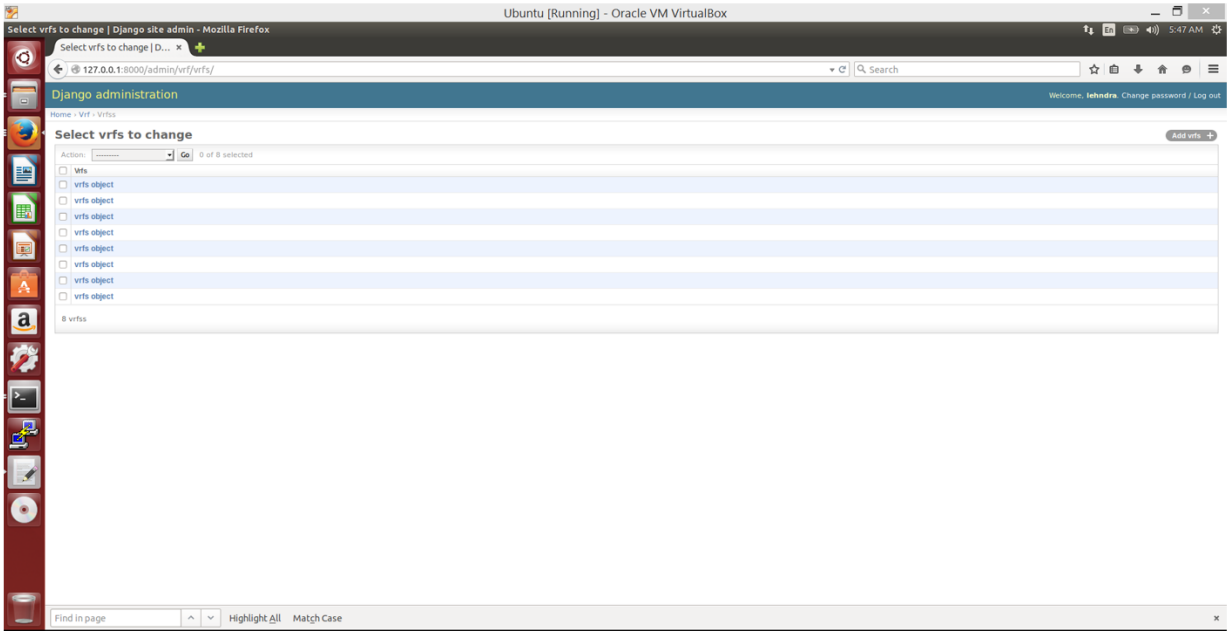

Figure 5.11: VRF objects

Click any VRF object to view stored information of VRF routes.

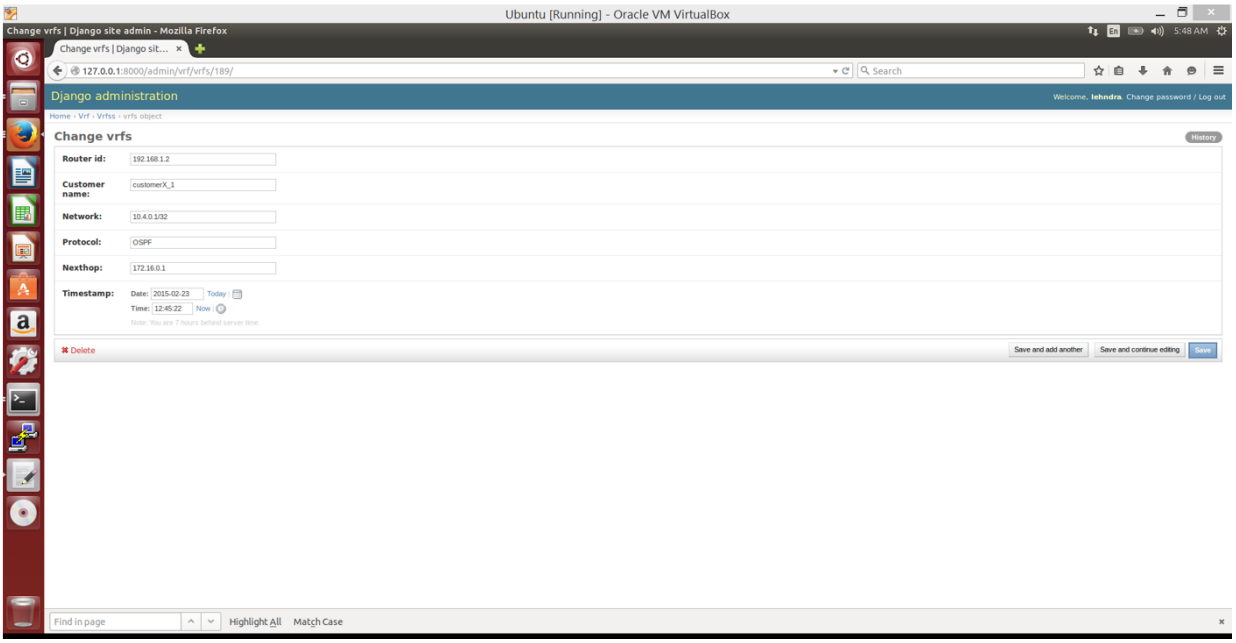

Figure 5.12: View VRF object

~

Chapter 6

Conclusion and Future Work

In this project, we were able to create a database and provide a web based front end to display the route distribution from various routing protocols on the network, VRF coming from different regions, networks and geolocations, which was the goal of this project. The required information was retrieved using a custom Python script, modulated as per needs and inserted into the database.

A user friendly interface has been created to present network data so that it can be easily analysed by the network engineer and necessary changes can be made in the network if need arises. It can be used to monitor and study the network traffic passing through one location, which includes route distribution on the network from various routing protocols and VRF information coming from different locations.

In order to study more realistic networks, the concepts like MPLS Traffic Engineering, metric costs on routes and adding more routes in the network, can be involved in future. The web based interface implemented in this project can also be improved and if need be, replaced with a full fledged Graphical User Interface.

~

References

[1] MPLS Fundamentals by Luc De Ghein. Cisco Press. ISBN: 1587051974

[2] [Online] https://sites.google.com/site/amitsciscozone/home/important-tips/mpls-wiki/route-distinguisher-its-types

[3] [Online] http://packetlife.net/blog/2013/jun/10/route-distinguishers-and-route-targets/

[4] [Online] https://docs.python.org/3/tutorial/index.html

[5] [Online] http://en.wikipedia.org/wiki/Secure_Shell

[6] [Online] http://docs.paramiko.org/en/1.15

[7] [Online] http://www.paramiko.org/

[8] [Online] http://docs.paramiko.org/en/1.15/api/client.html#paramiko.client.SSHClient.connect

[9] [Online] http://www.djangobook.com/

[10] [Online] https://www.djangoproject.com

[11] [Online] https://docs.djangoproject.com

[12] [Online] http://code.tutsplus.com/articles/python-from-scratch-create-a-dynamic-website-

[13] [Online] https://www.youtube.com/watch?v=CFypO_LNmcc

[14] [Online] http://dev.mysql.com/doc/refman/5.5/en/introduction.html

[15] [Online] http://en.wikipedia.org/wiki/GNS3

[16] [Online] https://community.gns3.com/docs

[17] [Online] http://en.wikipedia.org/wiki/Ubuntu_(operating_system)