**PROJECT REPORT**

on

***MULTI-CONTROLLER IN SOFTWARE DEFINED NETWORKING: A STUDY***

*Submitted in partial fulfillment of requirements for the award of*

**Integrated Master of Science (Int. MSc.)**

In the department of

**Information Technology**

****

*Submitted by*:

**Rajakuddin Ahmed (CS15MI0381)**

*Under the Supervision and Guidance of*

**Mr. N.RANA SINGHA**

ASSISTANT PROFESSOR

**School of Computing Sciences**

**The Assam Kaziranga University, Jorhat, Assam**

**May 2020**

**SCHOOL OF COMPUTING SCIENCES**

**THE ASSAM KAZIRANGA UNIVERSITY**

**JORHAT-785006:: ASSAM :: INDIA**

**CERTIFICATE**

This is to certify that the project report entitled ***“MULTI-CONTROLLER IN SOFTWARE DEFINED NETWORKING”****,* submitted to the School of Computing Sciences (SCS), **THE ASSAM KAZIRANGA UNIVERSITY, JORHAT, ASSAM,** in partialfulfillment for the completion of **Semester – X** of the degree of **Integrated Master of Science (Int. MSc.)** in the department of **Information Technology**, is a record of bona fide work carried out by **MR/Rajakuddin Ahmed**, **Roll No**.CS15MI0381 under my supervision and guidance.

All help received by us from various sources have been duly acknowledged.

No part of this report has been submitted elsewhere for award of any other degree.

----------------------------- -----------------------------

Dean, SCS, KU **Faculty Supervisor**

**Mr N.Rana Singha**

Assistant Professor

Dept. of Information Technology

**ACKNOWLEDGEMENT**

It is great pleasure to present this report on the project named, ”Multi-controller in software defined networking” undertaken by us as a part of our Int. MSc IT(SCS)curriculum.

A project without proper guidance is like ship without a navigator. A successful story is incomplete without paying tribute to those who inspired it. We would like to express our gratitude towards all those people who guided is for preparing this project which was a great learning process for us.

We are heartily indebted to Mr N.Rana Singha who guided us throughout the project and gave up valuable suggestions and encouragement.

We show our gratitude to our Dean (Dr. Tarun Kumar), HOD(Prof. Manoj Kumar Muchahari) and our project co-ordinator(Dr. Purnendu Bikash Acharjee) for providing the facilities and environment to bring out our innovation, talent and spirit of inquiry through this project.

**Full Name Enrollment No. Signature of Student**

**Rajakuddin Ahmed CS15MI0381**

**Declaration**

We declare that all the materials embodied in the project report entitled Multicontroller in software defined networking submitted to the School of Computing Science (SCS), The Assam Kaziranga University, Jorhat, Assam,by us as a X semester project in the partial fulfillment of the Masters of Computer Application in the department of Information Technology(IT) is original and no part of the report submitted to any other institution or organization for the award of any degree or diploma. I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact from any other sources in my project report. For any violation of the above facts I shall remain solely responsible. .

**Signature of Student**

Full Name :Rajakuddin Ahmed

Semester :X

Enrollment No. :CS15MI0381

Department : Information Technology

School : School of Computing Sciences

**LIST OF FIGURES**

**Figure name Figure Number Page No**

SDN architecture 3.1 8

Flat Design 3.2 11

Hierarchical design 3.3 12

Hyper-flow design 3.4 14

Hyper-flow design using Mininet 3.5 15

Background code for Hyper-flow 3.6 16

Updated hyper-flow 3.7 16

Background code foe updated hyper-flow 3.8 17

Hierarchical design using Mininet 3.9 18

**LIST OF TABLES**

**Table name Table Number Page No**

Comparative Table 3.1 19

**CONTENTS**

Certificate

Acknowledgement

Declaration

List of Figures

List of Tables

Abstract

1. Introduction 1

2. Literature Review

2.1 Literature Review 1 2

2.2 Literature Review 2 2-3

2.3 Literature Review 3 3

2.4 Literature Review 4 3-4

2.5 Literature Review 5 4

2.6 Literature Review 6 5

3. Project Over view

3.1 Why SDN? 6

3.2 What is SDN? 6-7

3.3 SDN Architecture 7-10

3.4 Controller 10-12

3.5 Multi-controller challenges 12-13

3.6 Objective 14

3.7 Design and Implementations 15-18

3.8 Comparative table 19

3.9 Software/Hardware requirements 20-24

4. Conclusion 25

5. Future work 26

Bibliography 27-28

**Abstract**

Software Defined networking (SDN) is a novel network paradigm that enables flexible management for networks. It decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The decision making part are on the hands of the controller which is a major advantage for the Software Defined Networking. With the increase of number of users, we may need more than one controller to manage the traffic according to the parameters such as Quality of service and many more. There are already many proposals that had been put in the context of multi-controllers. But none of the above proposal scores well against the scalability parameter. Here, we will make a general survey. It generally aims to rate most of the existing proposals that has been published so far. Also, we will propose a solution that would be better than some of the existing proposal in terms of the mentioned parameter.

**Chapter 1**

**Introduction**

The Internet has been identified as an essential infrastructure that supports social development and technological progress in the past 30 years, and it has profoundly changed the people’s working, studying and living styles. However, traditional network technology has inherent defects of rigid structure and complex configuration and cannot meet the requirement of network innovation. Thus, it is deemed urgent to design and develop a new network architecture that can dynamically and flexibly manage the network . Software-Defined Networking (SDN) is proposed to overcome the aforementioned weaknesses of the traditional network. As a new network paradigm, the SDN revolutionizes network technology by breaking the fundamental idea of traditional networks.

An SDN comprises three layers: data plane, control plane, and application plane. Data plane comprises of network devices (e.g., a router, and switch) and forwards packets according to a decision made by the control plane. Control plane acts as a mediator for the data plane and the application plane and handles the traffic flow in the network. Application plane is on the top of the control plane and achieves customized application logic (e.g., intrusion detection systems , big data analyses

The preliminary design of the control plane only uses one controller for a network. Though the advantages of centralized control in SDN network, SDN faces some problems challenging its nature (i.e., centralized control) due to day-to-day increasing network demands. Further, network operators try their best to strengthen the performance of the network controller, but it is still hard to meet the high demands due to the limited capacity of the single controller. This deficiency presents more obviously in the large-scale network. Moreover, the single point of failure is also the crucial factor in the one controller SDN network. The controller failure will cause disconnections between the controller and the switches. Since the controller software runs on a server and it may suffer from the hardware or software failure, Therefore, the controller failure is common in the network because of hardware or software breakdown. In a word, the above problems triggered by the single controller will hinder the deployment of SDN in actual production networks. To overcome those issues, several works propose using multi-controller working together to achieve the function of the logically centralized controller

In this paper, we focus on the survey of multi-controller research in SDN. We discuss the multi-controller overview in SDN and present the SDN issues of multi-controller: scalability,

**Chapter 2**

**Literature Review**

**2.1 Literature Review #1**

Title : An Overview of Multi-Controller Architecture in Software-Defined Networking

Author : Amir Hossein Moravejosharieh, Michael j watts, Kourosh Ahmadi

Publication : National Digital Switching System System Engineering and Technological Research

**Objective :**

Software-Defined Networking (SDN) is a rapidly growing and widely accepted networking paradigm that provides greater flexibility in network management. This is mainly achieved due to the separation of the control plane and data plane. In an SDN-enabled network, a centralized controller receives the high-level network application requirements, translates them into low-level commands to be instructed on forwarding devices. Such a controller has a limited capacity to respond to flow processing requests issued by forwarding devices. As the size of an SDN-enabled network or the amount of network traffic become larger, a single centralized controller may not be able to handle requests for flow processing which results in controller performance degradation and eventually dysfunctional SDN-enabled network. A practical solution would be the deployment of multi-controller architecture where a group of controllers collaboratively handle a massive amount of network traffic and flow processing requests. The major contribution of this paper is to provide an overview of the multi-controller architecture, its challenges in terms of reliability, scalability, coherence and availability, and also highlighting future research areas in this domain

**2.2 Literature Review #2**

Title : An Overview of SDN Architecture with multiple controllers

Author : Othmane Blial, Mouad Ben Mamoun, Benaini Redouane

Publication : Journal of computer network and communications

**Objective:**

Software-defined networking offers several benefits for networking by separating the control plane from the data plane. However, networks’ scalability, reliability, and availability remain as a big issue. Accordingly, multi-controller architectures are important for SDN-enabled networks. This paper gives a comprehensive overview of SDN multi-controller architectures. It presents SDN and its main instantiation OpenFlow. Then, it explains in detail the differences between multiple types of multi-controller architectures, like the distribution method and the communication system. Furthermore, it provides already implemented and under research examples of multi-controller architectures by describing their design, their communication process, and their performance results.

**2/3 Literature Review #3**

Title : Multi-Controller Based Software-Defined Networking : A Survey

Author : Zehua Guo, Peng Yi, Tao Hu, Julong Lan

Publication : National Digital Switching System System Engineering and Technological Research 12 March 2019

**Objective:**

Software-Defined Networking (SDN) is a novel network paradigm that enables flexible management for networks. As the network size increases, the single centralized controller cannot meet the increasing demand for flow processing. Thus, the promising solution for SDN with large-scale networks is the multi-controller. In this paper, we present a compressive survey for multi-controller research in SDN. First, we introduce the overview of multi-controller, including the origin of multi-controller and its challenges. Then, we classify multi-controller research into four aspects (scalability, consistency, reliability, load balancing) depending on the process of implementing the multi-controller. Finally, we propose some relevant research issues to deal with in the future and conclude the multi-controller research

**2.4 Literature Review #4**

Title : A Survey on Software Defined Networking with Multiple Controllers

Author : Yuan Zhang, Lin Cui, Wei Wang, Yuxiang Zhang

Publication : Journal of Network and Computer Applications

**Objective :**

Ccompared with traditional network, Software Defined Networking (SDN) decouples control plane and data plane, providing programmability to configure the network. In spite of such capability, one of the criticisms of SDN is that the SDN controller is a single point of failure and hence the controller decreases overall network availability. Having multiple controllers improves reliability of the network because the data plane can continue to operate if one controller fails. Furthermore, a single controller of SDN has many limitations on both performance and scalability. Thus, multiple controllers are required and critical for large-scale networks. However, multiple controllers increase network complexity dramatically and impose many new challenges to the management and schedule of SDN. This paper surveys latest researches on multiple controllers of SDN. Benefits and challenges of multiple controllers are discussed after giving an overview of SDN and Openflow in the paper. Afterward, we dwell on the detailed design principles and architectures of SDN with multiple controllers. Following that, current research works on multiple controllers placement and scheduling are carefully summarized and analyzed. Finally, we conclude this survey paper with some future works and suggested open research directions.

**2.5 Literature Review #5**

Title : HyperFlow: A Distributed Control Plane for OpenFlow

Author : Amin Tootoonchian , Yashar Ganjali

Publication : Journal of Network and Computer Applications

**Objective** :

Open-Flow assumes a logically centralized controller, which ideally can be physically distributed. However, current deployments rely on a single controller which has ma-jor drawbacks including lack of scalability. We present Hyper Flow, a distributed event-based control plane for Open-Flow. Hyper-Flow is logically centralized but physically distributed: it provides scalability while keeping the benefits of network control centralization. By passively synchronizing network-wide views of Open-Flow controllers, Hyper-Flow localizes decision making to individual controllers, thus minimizing the control plane response time to data plane requests. Hyper-Flow is resilient to network partitioning and component failures .It also enables interconnecting independently managed Open-Flow networks, an essential feature missing in cur-rent Open-Flow deployments. We have implemented Hyper-Flow as an application for NOX. Our implementation requires minimal changes to NOX, and allows reuse of existing NOX applications with minor modifications. Our preliminary evaluation shows that, assuming sufficient control bandwidth, to bound the window of inconsistency among controllers by a factor of the delay between the farthest controllers, the network changes must occur at a rate lower than 1000 events per second across the network.

**2/6 Literature Review #6**

Title : ElastiCon: An Elastic Distributed SDN Controller

Author : Advait Dixit , Fang Hao , T V Lakshman , Ramana Rao Kompella , Sarit Mukharjee

Publication : Journal of Computer Communication Networks

**Objective:**

Software Defined Networking (SDN) has become a popular paradigm for centralized control in many modern networking scenarios such as data centres and cloud. For large data centres hosting many hundreds of thousands of servers, there are few thousands of switches that need to be managed in a centralized fashion, which cannot be done using a single controller node. Previous works have proposed distributed controller architectures to address scalability issues. A key limitation of these works, however, is that the mapping between a switch and a controller is statically configured, which may result in uneven load distribution among the controllers as traffic conditions change dynamically.

To address this problem, we propose ElastiCon, an elastic distributed controller architecture in which the controller pool is dynamically grown or shrunk according to traffic conditions. To address the load imbalance caused due to spatial and temporal variations in the traffic conditions, ElastiCon automatically balances the load across controllers thus ensuring good performance at all times irrespective of the traffic dynamics. We propose a novel switch migration protocol for enabling such load shifting, which conforms with the Openflow standard. We further design the algorithms for controller load balancing and elasticity. We also build a prototype of ElastiCon and evaluate it extensively to demonstrate the efficacy of ourdesign.

**Chapter 3**

**Project Over view**

In this Paper, we focus on the survey of multi-controller research in SDN. We discuss the overview of SDN, multi-controller overview in SDN and present the SDN issue of multi-controller: scalability.

3.1 **WHY SDN?**

Before moving to SDN first we have to understand about the traditional network architecture. The traditional network architecture is mainly based on network devices such as routers and switches. These devices have integrated software and hardware where the hardware (data plane) responsible for simple packet forwarding and software (control plane) is responsible for setting the forwarding routes to the data plane. We can say that the control plane works like brain for the device. These devices are extremely intelligent because they have their own little brains which is the control plane, and with the help of control plane the devices communicates with other devices and extracts routing information’s from then and creates a table where the best possible route are stored. And then provide the paths to the data plane means they extract the best possible paths and then provide the best path to the data plane, after getting the routes the data plane perform the forwarding function.

Since every device is intelligent so they have to be configured and managed individually. When a network scales its configuration increases, its complexity increases, management also increases .For a huge network having thousands of devices the configuration need to be done individually on per device basis which could be really painful.

Some problems of the traditional networks are;

* Cost is high
* Complexity in configuration and management of the network devices
* Complex traffic patterns
* Vendor specific

To o overcome from the problems, the need for developing and deploying a new network architecture have raised

3.2 **What is SDN?**

Software defined networking is a network framework that involves in the separation of control function with the data forwarding function. Which make the devices, simple packet forwarding device and take brains away from the network devices which mean taking the control plane away from all the devices and put then them together in a centralized manner, so that the network can be operated from a centralized point.

Now the vendors only provide a dumb switch which will only have the forwarding engine. We or the network administrator will decide the control plane by writing custom logic; from this we can say that the cost of the device in SDN is lesser than that of the traditional network. The configuration and management is done from a single point which is the controller so we don’t need to bother about configuring each and every device manually because this can be done from a single point. The controller will have global view of the forwarding devices on the network and then the controller create a simplified or abstract view of the network, the network applications will choose this information to make decisions on how to implement network policies.

3.3 **SDN architecture**

SDN architecture is consisting of three layers and they are data plane, control plane and application plane.

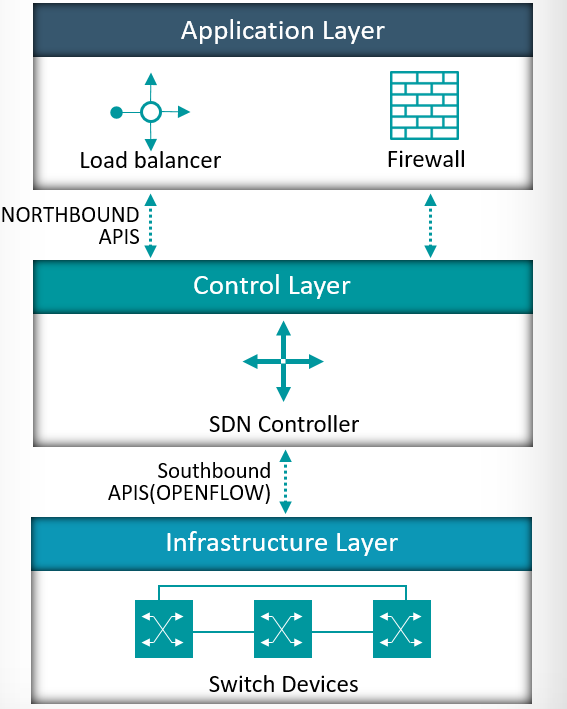
****

Fig 3.1: Software Defined Architecture

**Data plane**

Data plane is often referred to as resource layer or infrastructure layer where the network devices perform the packet transfer and processing according to the decision taken the by the control plane. These devices perform simple packet forwarding function. The data plane communicates with the controller and the controller manages the network via Openflow switch protocol. In addition to simple forwarding of packets the network devices can alter the packet headers before forwarding them, or drop the packets

**Data plane functions**

The principal functions of Data plane network devices are as follows:

* **Control Support function:** Interacts with the SDN control layer to support programmability via resource-control interfaces. The switch communicates with the controller and the controller manages the switch via the OpenFlow switch protocol.
* **Data Forwarding function:** Accepts incoming data flows from other network devices and end systems and forwards them along the data forwarding paths that have been computed and established according to the rules defined by the SDN applications.

These forwarding rules used by the network device are embodied in forwarding tables that indicate for given categories of packets what the next hop in the route should be. In addition to simple forwarding of a packet, the network device can alter the packet header before forwarding, or discard the packet. Arriving packets may be placed in an input queue, awaiting processing by the network device, and forwarded packets are generally placed in an output queue, awaiting transmission.

The network device with three I/O ports: one providing control communication with an SDN controller, and two for the input and output of data packets. The network device may have multiple ports to communicate with multiple SDN controllers, and may have more than two I/O ports for packet flows into and out of the device.

**Control plane**

The SDN control layer maps application layer service requests into specific commands and directives to data plane switches and supplies applications with information about data plane topology and activity. The control layer is implemented as a server or cooperating set of servers known as SDN controllers. Some of the function that provided by the SDN control plane are:

* **Shortest path forwarding**: Uses routing information collected from switches to establish preferred routes.
* **Security mechanisms**: Provides isolation and security enforcement between applications and services.
* **Topology manager**: Builds and maintains switch interconnection topology information.
* **Statistics manager**: Collects data on traffic through the switches.
* **Notification manager**: Receives, processes, and forwards to an application events, such as alarm notifications, security alarms, and state changes.

**Application plane**

Network applications act at application plane, SDN allow the network applications to monitor and manage the network behaviour. The application includes network applications that specifically deal with the control and management of the network. It contains network applications that control, monitor and define network behaviour and resources. The network applications interact with the control plane via application control interfaces. The application plane have a layer called network services abstraction layer, this layer provides an abstract view of the network resources and hides the details about the underlying network devices.

3.4 **Controller**

Controllers are also referred to as the brains in a SDN network. Controllers act as a control point for the networks to manage the flow between the application plane and data plane in order to create a more flexible network.

**Single centralized controller**

The single centralized controller manages the network from a centralized point. As the network size increases, a single centralized controller cannot meet the growing demand for flow processing because of the limited capacity of the controller leads to degradation of the performance of the controller, there is also a single point of failure for a single centralized controller. If a controller fails then the network devices in the network won’t be able to get forwarding routes which may create an inconsistency in the networking. Thus, the promising solution for SDN with large scale organizations is the Multi-controller.

**Multi-controller**

Multi-controller design can divide the entire network into several domains, and each controller manages its own SDN domain. To make sure that packets are transmitted correctly in the network, the controllers must interact the domain information with each other to keep the consistent view. This Multi-controller approach avoids having a SPOF and enables to scale up sharing load among several controllers.

The two basic multi-controller architectures are multi-controller flat design and multi-controller hierarchical design.

* The flat design extends the capability of the control plane, but it also requires complicated controller management and extra control overheads. For example, the controllers must frequently communicate with each other to guarantee the consistent network view. HyperFlow

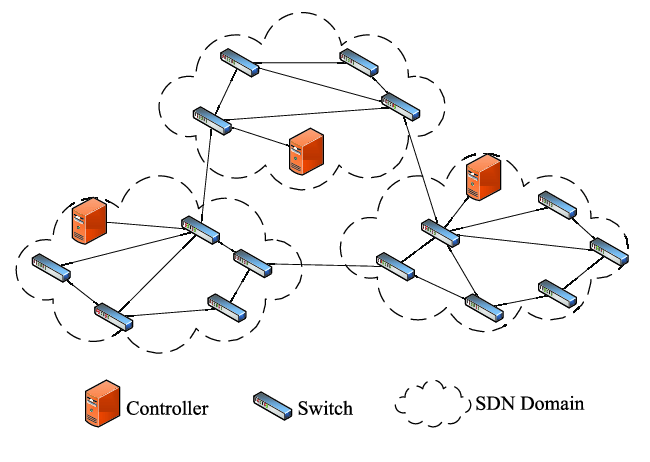


Fig 3.2: Flat multi-controller design

* Hierarchical design usually uses two-layer controllers: domain controller, which manages switches in its local domains and runs local control applications, and root controller, which manages domain controllers and maintains a global network view.

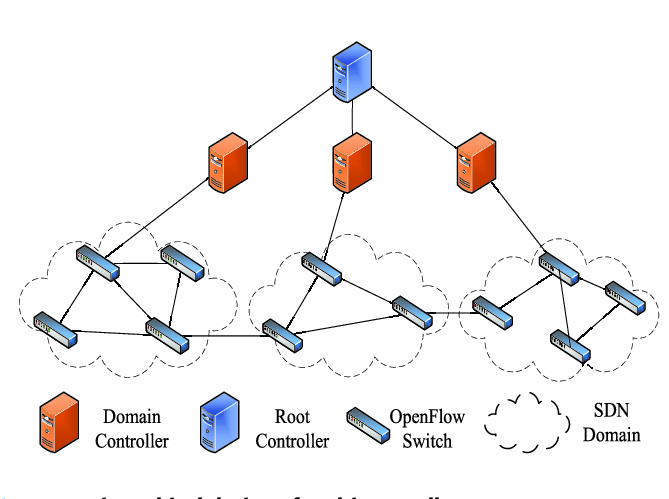


Fig 3.3: Hierarchical multi-controller design

**3.5 Challenges Of Multi-controller**

* **Multi-controller Scalability** : Based on the two basic multi-controller architectures, the proposed for multi-controller is to overcome the shortages of the single controller, such as single controller failure and limited controller capacity. However, multi-controller also raises the challenges of scalability: how to select the controller locations and how to allocate the switches for multi-controller in the network.
* **Multi-controller Consistency** : Multi-controller design can divide the entire network into several domains, and each controller manages its own SDN domain. To make sure that packets are transmitted correctly in the network, the controllers must interact the domain information with each other to keep the consistent view. Therefore, controller consistency also becomes an important issue for the multi-controller SDN.
* **Multi-controller Reliability** : Using multi-controller resolves the single point of failure problem for the controller, but it cannot guarantee the high reliability of the control plane. The connection links among switches and controllers have limited capacity. If these links experience congestion, interruption or failure, controllers and switches cannot normally communicate with each other, leading to the isolation among controllers and switches. Additionally, controllers could be failed or overwhelmed by malicious attacks (e.g., excessive packet-in requests). Thus, the multi-controller reliability is also important for actual deployment of multi-controller.
* **Multi-Controller Load-balancing:** The introduction of multi-controller partitions the network into several SDN domains, while the controllers monitor the local switches in the domain, respectively. However, due to the network traffic variation and the static mapping between switches and controllers, it is likely to produce overloaded controller and under loaded controller in the network. Further, imbalanced load distribution among controllers will seriously degrade the network performance, it is essential to ensure the nice load balancing performance of multi-controller

**3.6 Objective**:

In this paper, our main objective is to solve the multi-controller scalability issue. Initially we will implement the multi-controller model which was given by **“Amin Tootoonchian, Yashar Ganjali”** in their paper **“HyperFlow: A Distributed Control Plane for OpenFlow”** and perform scalability testing by generating traffic on their model and then provide a new multi-controller model which will be more scalable than the HyperFlow model.

As we know when a network size increases its become difficult for a single controller to manage the network. So they have proposed Hyper-flow which is a FLAT multi-controller architecture.



Fig 3.4: Multi-Controller Hyper-flow Design

This is the multi-controller model which was proposed in order to eliminate the single controller scalability issue. Here the whole network is divided into three domains and each domain is controlled by a controller. Each controller must communicate with each other to keep the consistent view of the network.

**3.7 Design and Implementation**:

We have implemented the given HyperFlow model using Mininet to perform the scalability testing. We will perform the testing by increasing the no of forwarding devices and hosts in order to increase the traffic.

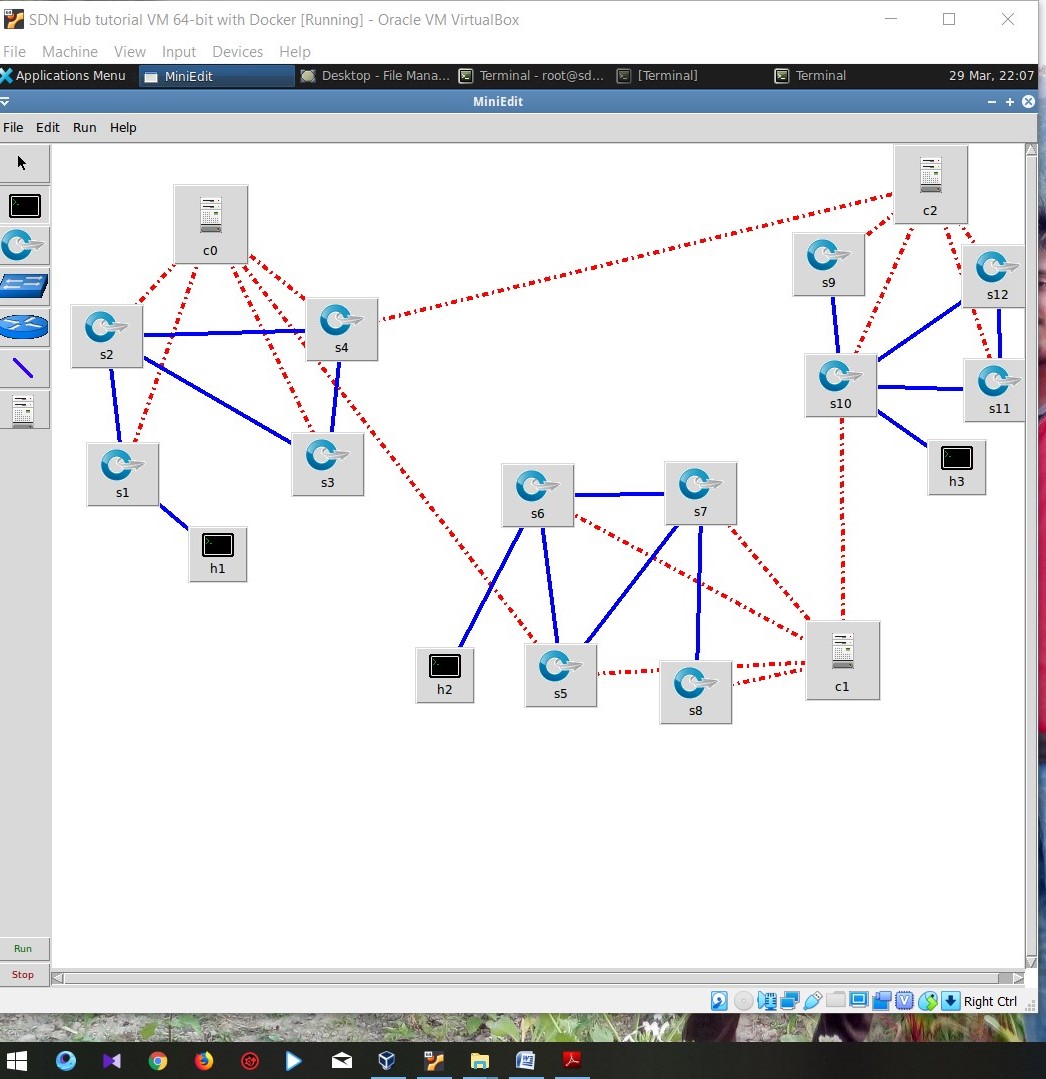


Fig 3.5: Hyper-flow design using Mininet

In Fig we can see that, I have implemented the HyperFlow model using Mininet. Here we can see that there is no direct link between the controllers but we know that they are all logically connected to each other. Here we can also see that initially I am using one host for each domain.

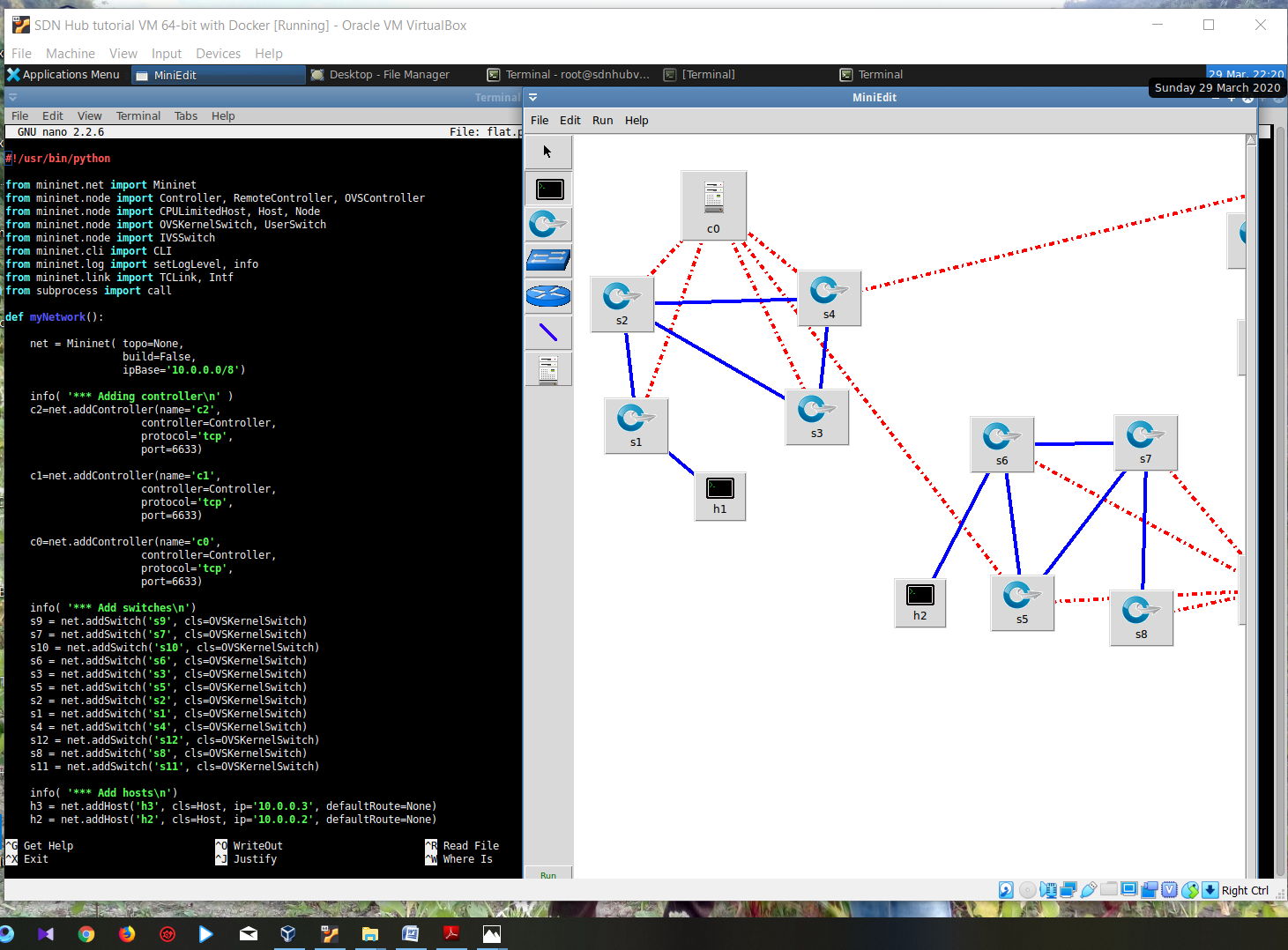


Fig 3.6: Background code for hyper-flow

In Fig3.6 we can see the background code for the hyper-flow model I have implemented using Mininet. Where we can see how the switches or forwarding devices are added, how the controllers are added and the links between them are added. In the controllers, we are using RYU as an operating system.

Now we will increase the no of hosts in the network in order to increase the traffic and test the scalability of the model.

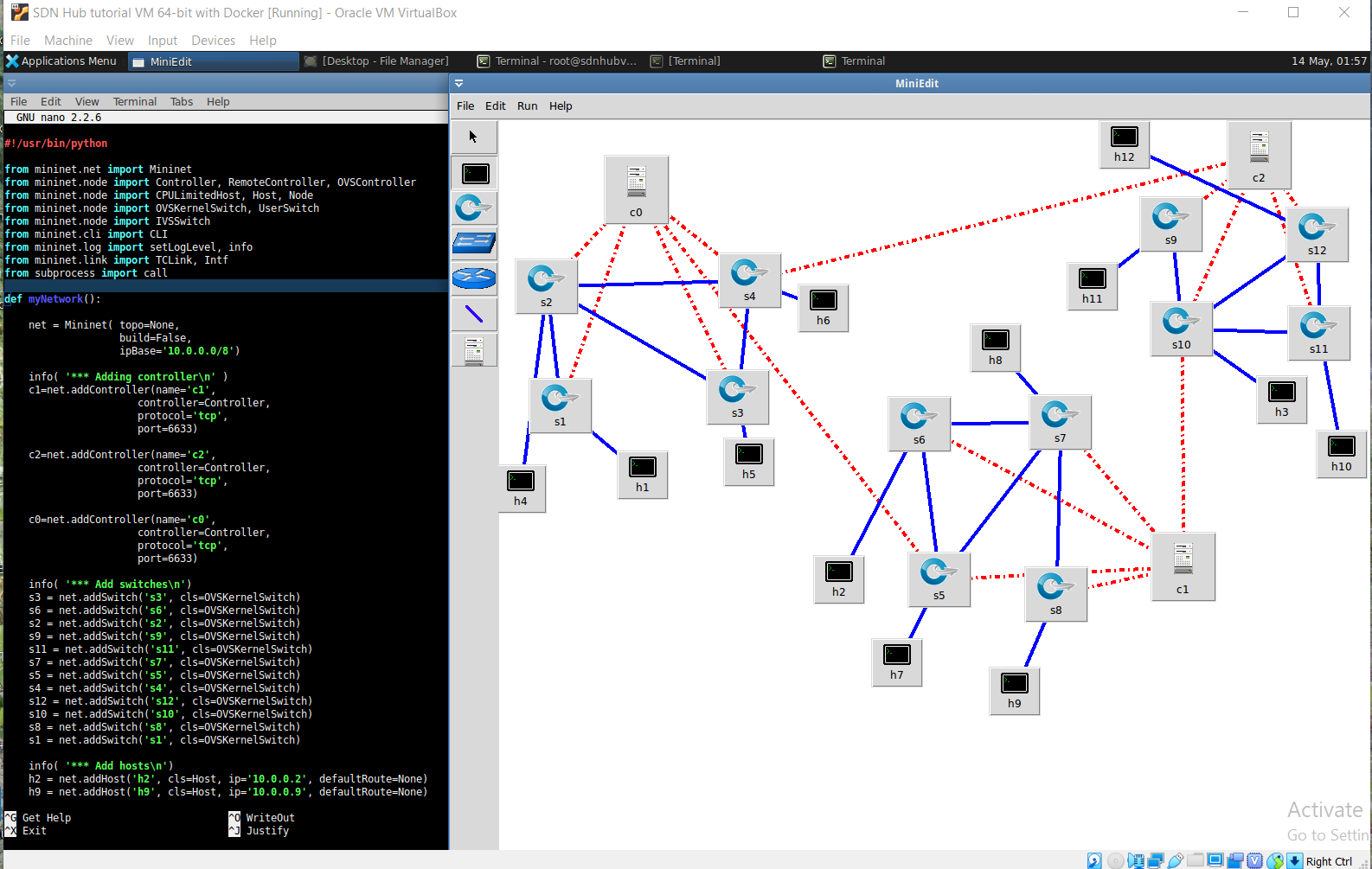


Fig 3.7: hyper-flow model after adding hosts

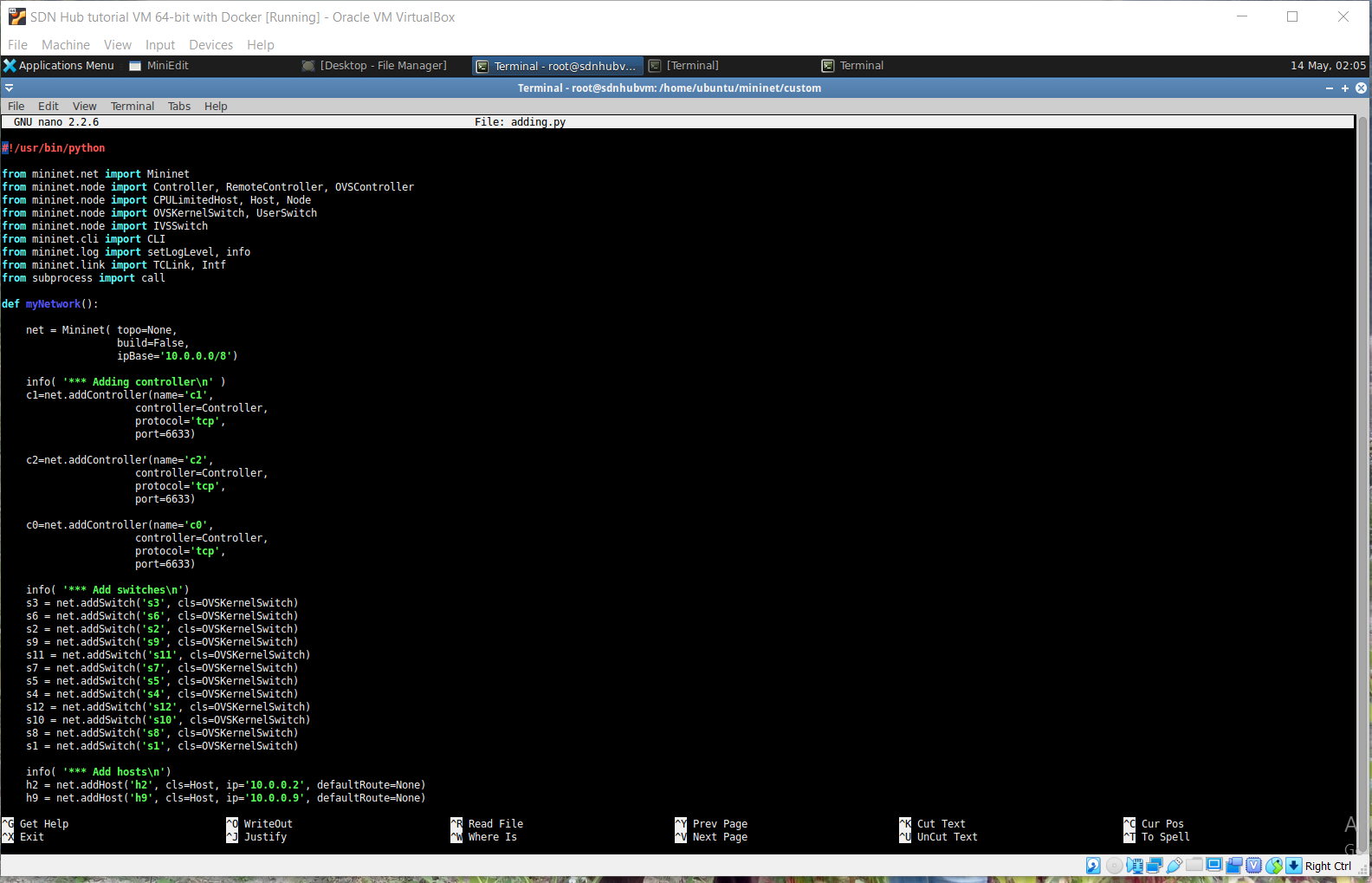
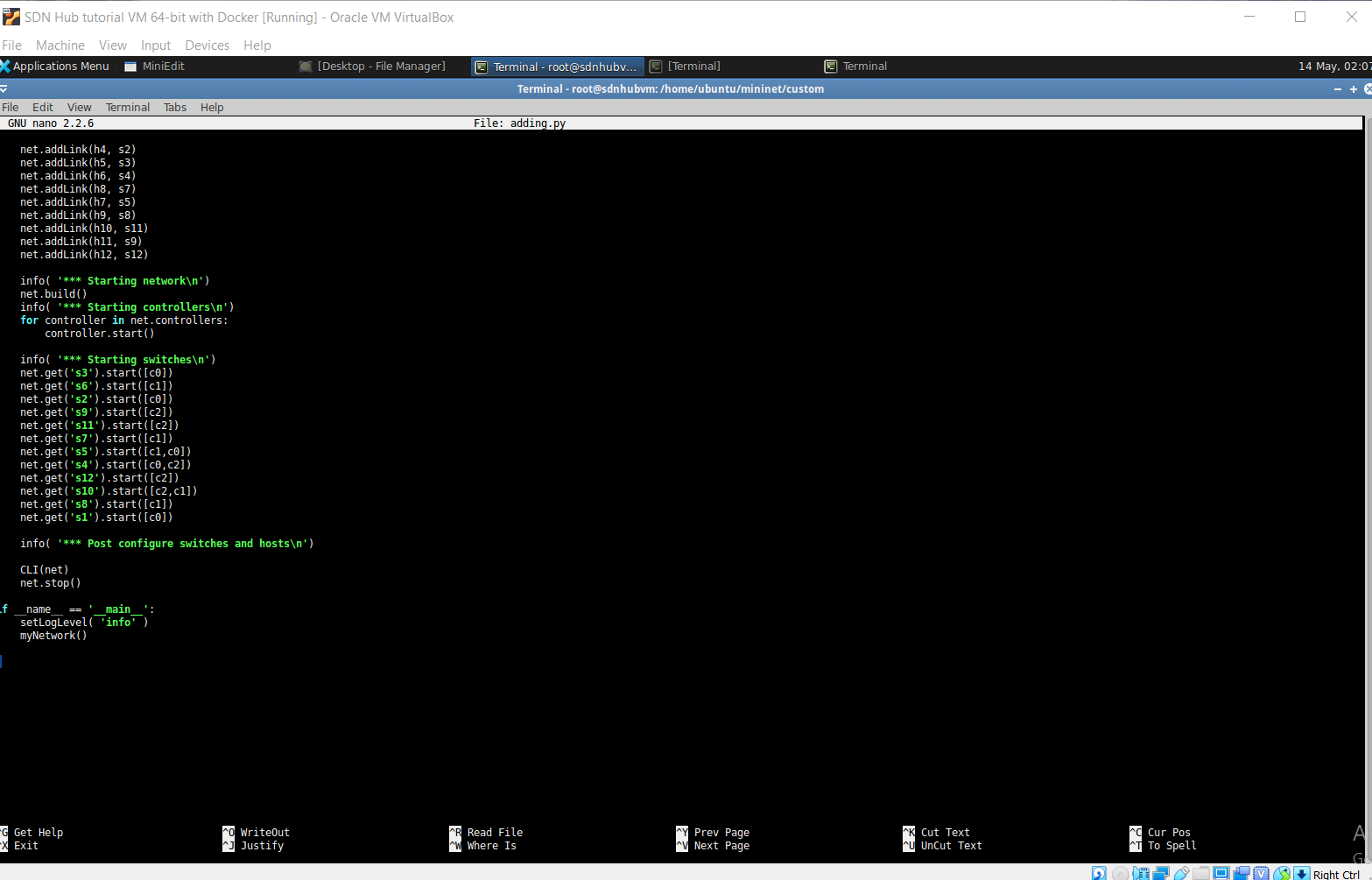
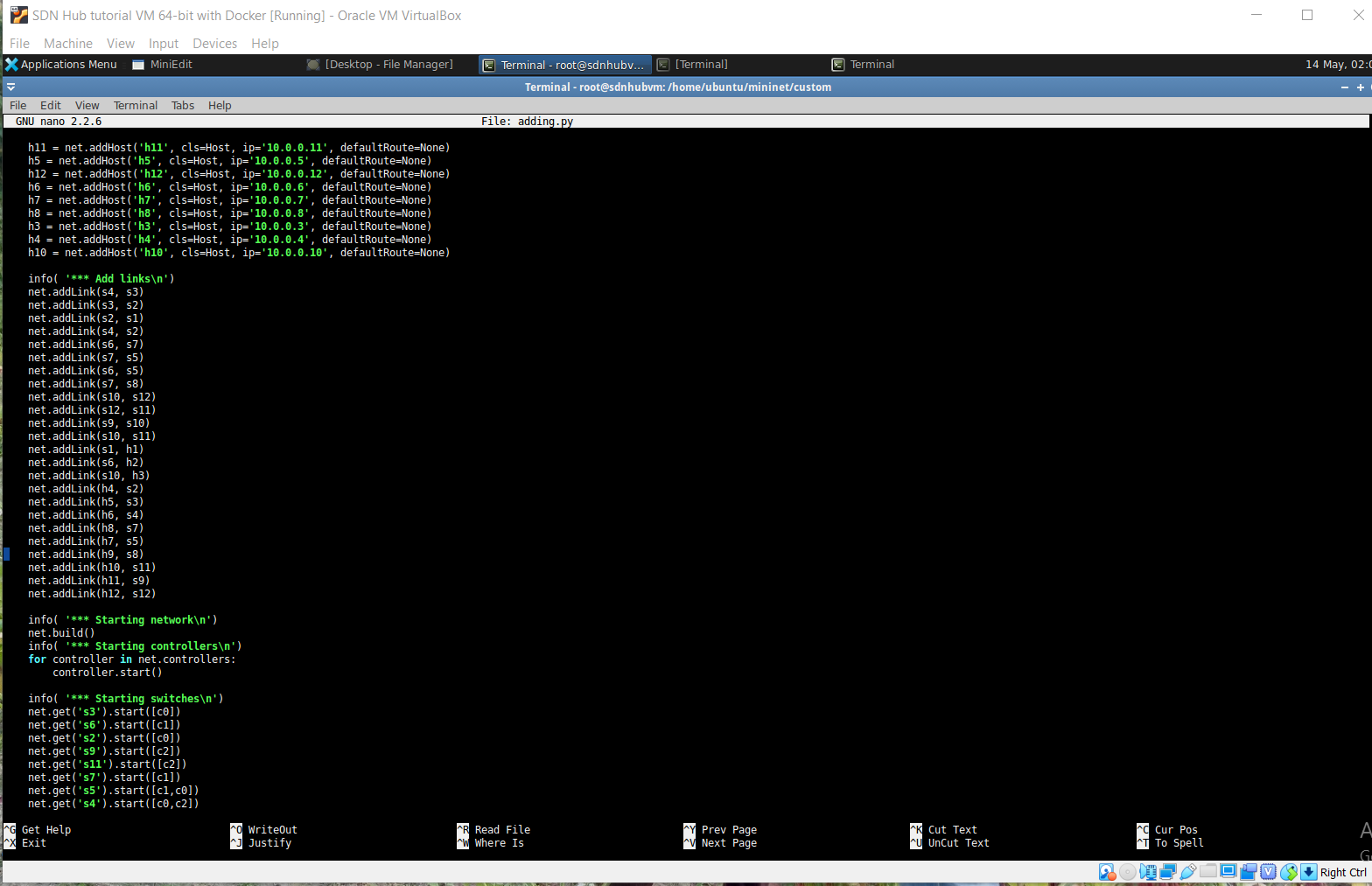
** **

Fig 3.8: Background code after adding hosts

The Hyper Flow model is an example of Multi-controller Flat architecture. After studying the Multi-controller architectures we came to know that using Flat architecture we can make the network scalable. In flat architecture the whole network is divided into several domains and each domain is operates by a controller. The controllers must communicate with each other to keep the consistent view of the network but it still has some limitations. If the communicating link between the controllers were broken then they won’t be able to communicate with each other and can’t keep the consistent view of the network. To solve this issue we should use a Hierarchical model instead of a Flat model. In a hierarchical model each domain is controlled by a controller (child controller) and the entire child controller were controlled by a Root controller. The child controllers only communicate with the root controller they don’t communicate with each other.

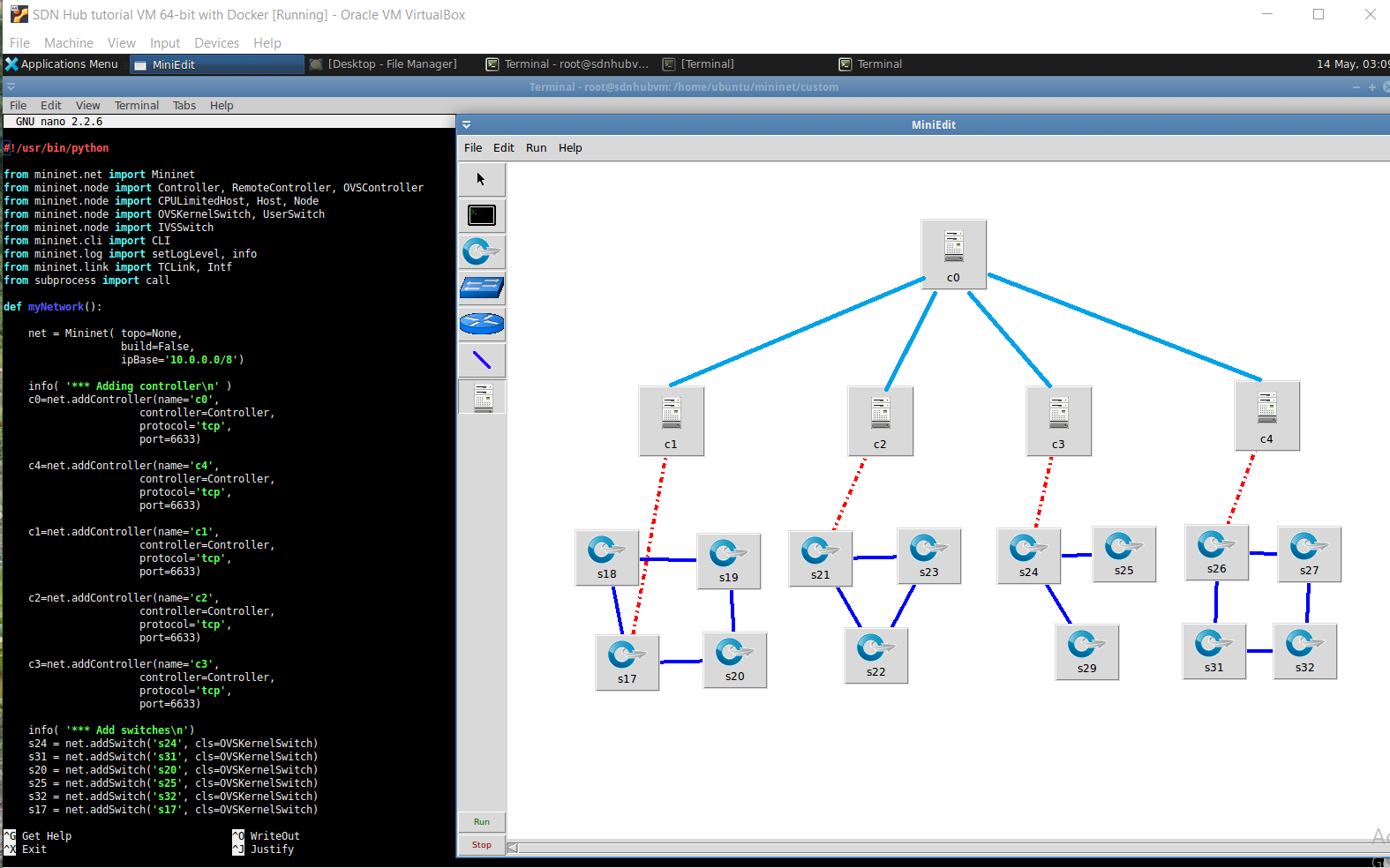


Fig 3.9: Multi-controller hierarchical model

**3.8 Comparative table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| References | Addressed Issue | | | | Proposed objectives and solutions |
| Storage | Security | Performance Evaluation | Implementing applications using SDN |
| A Survey of SDN past, present and future of programmable networks | yes | yes | No | no | In this paper, spurred by the vision that future online worlds will include infrastructure–based and infrastructure–less systems, the creators investigate the utilization of the Software–Defined Networking (SDN) worldview in these so–called "heterogeneous" arranged situations. |
| The Open SDN Architecture | no | no | No | yes | This white paper provides a study about Big Switch Networks Open SDN Suite architecture that provides unmatched network agility, choice in network hardware, and optimized network operations. |
| Wireless mesh software defined networks (wmSDN) | no | no | Yes | no | The authors proposed a solution to integrate SDN functionality in a Wireless Mesh, trying to face the reliability concerns related to this environment. The proposed wmSDN approach integrates “ready-to-market” technologies. |
| Network Innovation using Openflow A Survey, | no | yes | Yes | yes | A study has been given about Openflow and difficulties confronting the expansive scale arrangement of Openflow-based systems and no usage gave. |
| Realizing the power of SDN with HP virtual application networks | no | no | Yes | no | The design, implementation, and evaluation of B4, a private WAN connecting Google’s data centres across the planet, have been presented. The objective function of the proposed system is to deliver max-min fair allocation to applications. |

Table 3.1: Comparative table

**3.9 Software/hardware requirements**

* Mininet
* Miniedit
* Java
* Python

**3.9.1 Mininet**

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking.

Mininet supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network on a laptop or other PC.

Mininet:

* Provides a simple and inexpensive **network testbed** for developing OpenFlow applications
* Enables **multiple concurrent developers** to work independently on the same topology
* Supports **system-level regression tests**, which are repeatable and easily packaged
* Enables **complex topology testing,** without the need to wire up a physical network
* Includes a **CLI**that is topology-aware and OpenFlow-aware, for debugging or running network-wide tests
* Supports **arbitrary custom topologies**, and includes a basic set of **parametrized topologies**
* is **usable out of the box** without programming, but
* also Provides a straightforward and extensible **Python API** for network creation and experimentation

Mininet provides an easy way to get correct system behaviour(and, to the extent supported by your hardware, performance) and to experiment with topologies.

Mininet networks run real code including standard Unix/Linux network applications as well as the real Linux kernel and network stack (including any kernel extensions which you may have available, as long as they are compatible with network namespaces.)

Because of this, the code you develop and test on Mininet, for an OpenFlow controller, modified switch, or host, can move to a real system with minimal changes, for real-world testing, performance evaluation, and deployment. Importantly this means that a design that works in Mininet can usually move directly to hardware switches for line-rate packet forwarding.

## How it Works

Nearly every operating system virtualizes computing resources using a process abstraction. Mininet uses process-based virtualizationto run many (we’ve successfully booted up to 4096) hosts and switches on a single OS kernel. Since version 2.2.26, Linux has supported network namespaces, a lightweight virtualization feature that provides individual processes with separate network interfaces, routing tables, and ARP tables. The full Linux container architecture adds chroot**()**jails, process and user namespaces, and CPU and memory limits to provide full OS-level virtualization, but Mininet does not require these additional features. Mininet can create kernel or user-space OpenFlow switches, controllers to control the switches, and hosts to communicate over the simulated network. Mininet connects switches and hosts using virtual Ethernet (veth) pairs. While Mininet currently depends on the Linux kernel, in the future it may support other operating systems with process-based virtualization, such Solaris containers or! FreeBSD jails. Mininet’s code is almost entirely Python, except for a short C utility.

## Why it’s better

Mininet combines many of the best features of emulators, hardware testbeds, and simulators. Compared to full system virtualization based approaches, Mininet:

* **Boots faster**: seconds instead of minutes
* **Scales larger**: hundreds of hosts and switches vs. single digits
* **Provides more bandwidth**: typically 2Gbps total bandwidth on modest hardware
* **Installs easily**: a prepackaged VM is available that runs on VMware or Virtual Box for Mac/Win/Linux with OpenFlow v1.0 tools already installed.
* is **inexpensive** and **always available** (even before conference deadlines)
* is **quickly reconfigurable and restartable**
* easily **connects to real networks**
* offers **interactive performance** - you can type at it

## Limitations

Mininet-based networks cannot (currently) exceed the CPU or bandwidth available on a single server.

Mininet cannot (currently) run non-Linux-compatible OpenFlow switches or applications; this has not been a major issue in practice.

**Mininet's API**

Mininet's API is built at three primary levels:

* Low-level API: The low-level API consists of the base node and link classes (such as Host, Switch, and Link and their subclasses) which can actually be instantiated individually and used to create a network, but it is a bit unwieldy.
* Mid-level API: The mid-level API adds the Mininet object which serves as a container for nodes and links. It provides a number of methods (such as addHost(), addSwitch(), and addLink()) for adding nodes and links to a network, as well as network configuration, startup and shutdown (notably start() and stop().)
* High-level API: The high-level API adds a topology template abstraction, the Topo class, which provides the ability to create reusable, parametrized topology templates. These templates can be passed to the mn command (via the --custom option) and used from the command line.

**3.9.2 Miniedit**

The miniedit is a simple GUI editor included in mininet. It helps to understand how mininet can be extended and how the topology looks like. MiniEdit has a simple user interface that presents a canvas with a row of tool icons on the left side of the window, and a menu bar along the top of the window. The user interface of miniedit GUI looks as in the image below:

**3.9.3 Python**

Python is an interpreted high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3.

The Python 2 language, i.e. Python 2.7.x, was officially discontinued on January 1, 2020 (first planned for 2015) after which security patches and other improvements will not be released for it. With Python 2's end-of-life, only Python 3.5.x and later are supported.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source[ reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development

Features of python

* Easy to learn and use
* Expressive language
* Interpreted language
* Free and open source
* Object Oriented language
* Extensible and Integrated
* GUI programming support
* Large standard library

**3.9.4 Java**

Java is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible. It is intended to let application developers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but it has fewer low-level facilities than either of them. As of 2019, Java was one of the most popular programming languages in use according to GitHub, particularly for client-server web applications, with a reported 9 million developers.

Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle) and released in 1995 as a core component of Sun Microsystems' Java platform. The original and reference implementation Java compilers, virtual machines, and class libraries were originally released by Sun under proprietary licenses. As of May 2007, in compliance with the specifications of the Java Community Process, Sun had relicensed most of its Java technologies under the GNU General Public License. Meanwhile, others have developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java (bytecode compiler), GNU Classpath (standard libraries), and IcedTea-Web (browser plugin for applets).

The latest versions are Java 13, released in September 2019, and Java 11, a currently supported long-term support (LTS) version, released on September 25, 2018; Oracle released for the legacy Java 8 LTS the last free public update in January 2019 for commercial use, while it will otherwise still support Java 8 with public updates for personal use up to at least December 2020. Oracle (and others) highly recommend uninstalling older versions of Java because of serious risks due to unresolved security issues. Since Java 9 (and 10 and 12) is no longer supported, Oracle advises its users to immediately transition to the latest version (currently Java 13) or an LTS release

There were five primary goals in the creation of the Java language

* It must be simple, object-oriented, and familiar.
* It must be robust and secure.
* It must be architecture-neutral and portable.
* It must execute with high performance.
* It must be interpreted, threaded, and dynamic.

**Chapter 4**

**Conclusion**

The design and performance of the control plane are the critical part of SDN. In order to achieve the large-scale application of SDN, the control plane has evolved from the single centralized controller to multiple controllers. In this paper, based on the existing literature, we first provide an overview of multi-controller, including the origin of multi-controller. Then, we summarize the main research challenge of multi-controller: scalability. Meanwhile, we also consider the corresponding solution for the scalability issue

**Chapter 5**

**Future Work**

In future we need to perform some scalability test over the Hyper-flow model, after that we also needed to perform some test over the given new hierarchical model to check for the scalability of the new model.

**Bibliography**

[1] H. c. Wang and H. s. Doong, “Validation in Internet Survey Research: Reviews and Future Suggestions,” System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on, Waikoloa, HI, 2007

[2] A. M. Ahmed, T. Qiu, F. Xia, B. Jedari and S. Abolfazli, “EventBased Mobile Social Networks: Services, Technologies, and Applications,” in IEEE Access, 2014, vol. 2

[3] T. Benson, A. Akella, and D. Maltz, “Unraveling the complexity of network management,” in Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, ser. NSDI’09, Berkeley, CA, USA, 2009

[4] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, “Intelligent design enables architectural evolution,” in Proceedings of the 10th ACM Workshop on Hot Topics in Networks, ser. HotNets-X. New York, NY, USA: ACM, 2011.

[5] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, “Software-defined internet architecture: Decoupling architecture from infrastructure,” in Proceedings of the 11th ACM Workshop on Hot Topics in Networks, ser. HotNets-XI. New York, NY, USA: ACM, 2012

[6] FEAMSTER N, REXFORD J, ZEGURA E. “The road to SDN: an intellectual history of programmable networks,” ACM Sigcomm Computer Communication Review, vol. 44, no. 2

[7] NADEAU T D, GRAY K. “SDN: Software Defined Networks,” O’Reilly Media, Inc, 2013.

[8] V. López, O. González de Dios, B. Fuentes, M. Yannuzzi, J. P. Fernández-Palacios and D. López, “Towards a network operating system,” OFC 2014, San Francisco, CA, 2014

[9] Ryu. [Online]. Available: http://Osrg.github.com/ryu/

[10] Zehua Guo, Ruoyan Liu, Yang Xu, Andrey Gushchin, Anwar Walid and H. Jonathan Chao, “STAR: Preventing flow-table overflow in software-defined networks,” Computer Networks, 2017.

[11] L. Sidki, Y. Ben-Shimol and A. Sadovski, “Fault tolerant mechanisms for SDN controllers,” 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, 2016

[12] Oktian Y E, Lee S G, Lee H J, et al. “Distributed SDN controller system: A survey on design choice,” Computer Networks, 2017, vol. 121,

[13] Y. Jia, N. Hua, Y. Yu, Y. Li and X. Zheng, “Experimenting with multi-controller collaboration for large-scale intra-data center networks,” 2017 Optical Fiber Communications Conference and Exhibition (OFC), Los Angeles, CA, 2017,

[14] Technical White Paper, “Realizing the power of SDN with HP virtual application networks,” © Copyright 2012 Hewlett-Packard Development Company, L.P. “The Open SDN Architecture,” Copyright 2013 Big Switch Networks, Inc. All rights reserved.

[15] D. F. Macedo, D. Guedes, L. F. M. Vieira, M. A. M. Vieira, and M. Nogueira, “Programmable Networks from Software-Defined Radio to Software-Defined Networking,” IEEE Communication Surveys & Tutorials, vol. 17, no.2, second quarter 2015.

[16] A. Lara, A. Kolasani, and B. Ramamurthy, “Network Innovation using OpenFlow A Survey,” IEEE Communications Surveys & Tutorials, vol. 16, no. 1, first quarter 2014.