SDN LAB Manual

Software Defined Networks (Anna University)

RECORD NOTE BOOK

Name : …….

Register No : ……

Subject Code/Title :…….....

Year/Semester: ……..…….

KCG College of Technology

Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

[CCS365 SOFTWARE DEFINED NETWORKS LABORATORY]

FIFTH SEMESTER

CONTENTS

EX.NO	DATE	NAME OF THE EXPERIMENTS	PAGE NO	MARKS	FACULTY SIGNATURE
1		Setup your own virtual SDN lab i)VirtualBox/Mininet Environment for SDN - http://mininet.org ii) https://www.kathara.org iii) GNS3			
2		Create a simple mininet topology with SDN controller and use Wireshark to capture and visualize the OpenFlow messages such as OpenFlow FLOW MOD, PACKET IN, PACKET OUT etc.			
3		Create a SDN application that uses the Northbound API to program flow table rules on the switch for various use cases like L2 learning switch, Traffic Engineering, Firewall etc			
4		Create a simple end-to-end network service with two VNFs using vim-emu https://github.com/containernet/vim-emu 5) Install OSM and onboard and orchestrate network service.			

EX.NO: 1 DATE:	Setup your own virtual SDN lab **i) Virtualbox/Mininet Environment for SDN - http://mininet.org** **ii) https://www.kathara.org** **iii) GNS3**

AIM:

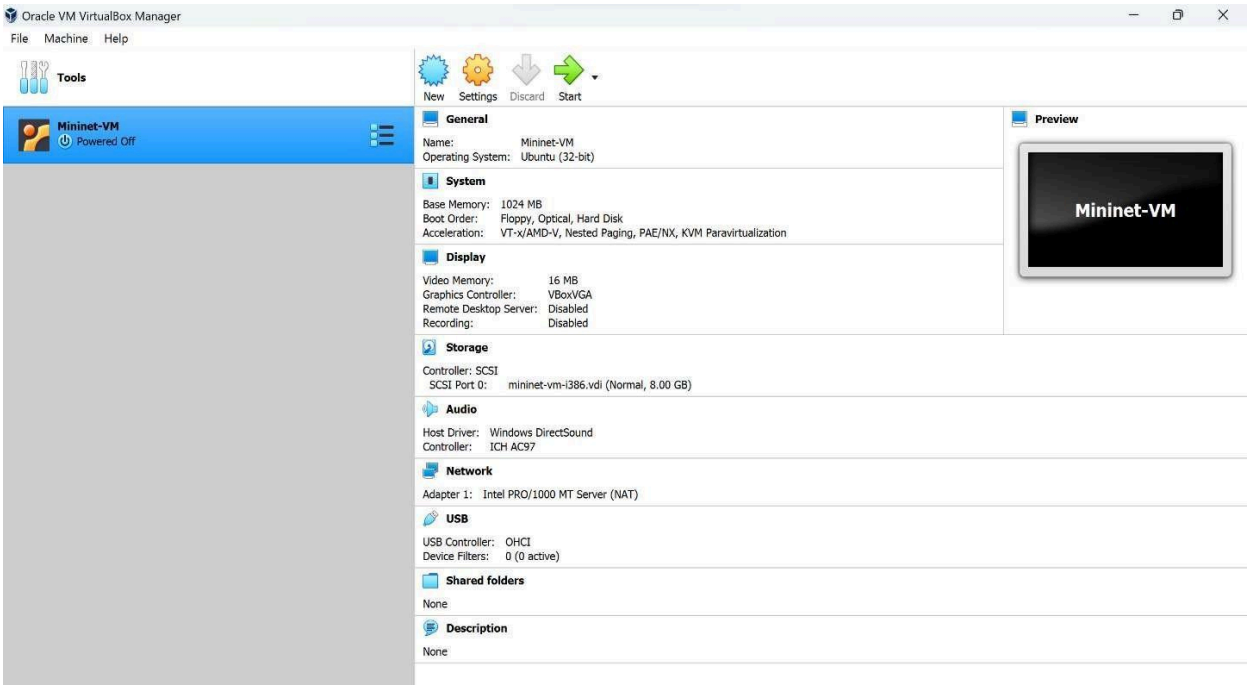
To Setup your own virtual SDN lab using one of the three options: Virtualbox/Mininet, Kathara, or GNS3

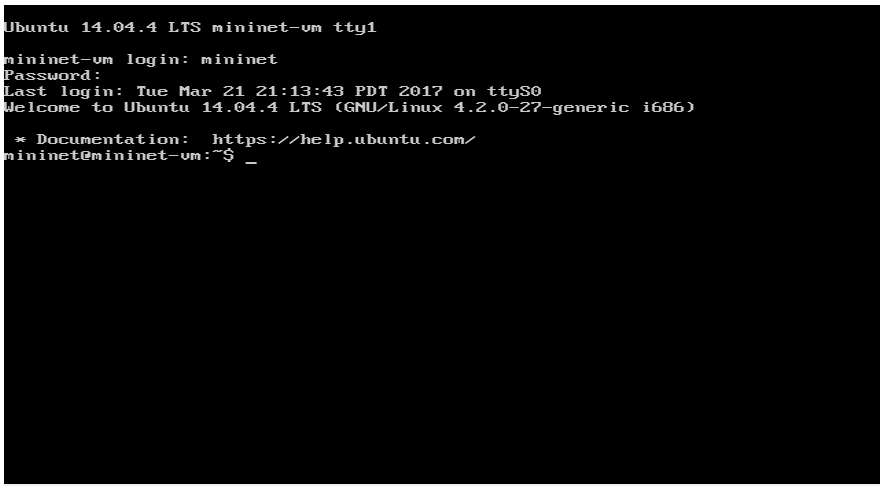
PROCEDURE:

i) Virtualbox/Mininet Environment for SDN - http://mininet.org

1. Download and install Virtualbox from its [official website].
2. Download and install Mininet from its [official website]. You can choose to install Mininet as a virtual machine image, a native installation, or a source code installation.
3. Launch Mininet and create a virtual network using the command line interface or the graphical user interface. You can also use predefined network topologies or custom scripts to create your network.
4. Connect your virtual network to an SDN controller, such as OpenDaylight, using the OpenFlow protocol. You can download and install OpenDaylight from its [official website].
5. Test and run different SDN applications and scenarios on your virtual network using the Mininet commands or the SDN controller interface.

OUTPUT:





RESULT:

Thus, the setup of VirtualBox/Mininet Environment for SDN was installed successfully.

EX.NO:2 DATE:		**Create a simple mininet topology with SDN controller and use Wireshark to capture and visualize the OpenFlow messages such as OpenFlow FLOW MOD, PACKET IN, PACKET OUT etc,.**

AIM:

1. Create a simple Mininet network with an SDN controller (Ryu).
2. Capture OpenFlow messages, including FLOW_MOD, PACKET_IN, and PACKET_OUT, using Wireshark.
3. Visualize the captured OpenFlow messages for analysis.

PROCEDURE:

1. **Install Mininet:**

Ensure you have Mininet installed on your system. You can use a Linux distribution for this task.

2. **Install Ryu SDN Controller:**

Install Ryu, a popular SDN controller, using pip:

pip install ryu

3.Create the Mininet Topology:

Create a Python script (e.g., `mininet_topology.py`) to define your Mininet network topology and start Mininet with the Ryu controller. Here's an example topology with a single switch and two hosts:

```python
from mininet.net import Mininet
from mininet.topo import SingleSwitchTopo
from mininet.node import RemoteController
# Create a Mininet instance
net = Mininet(topo=SingleSwitchTopo(2), controller=RemoteController)
# Start Mininet
net.start()
```

4. Start the Ryu Controller:

In a separate terminal, start the Ryu controller:

```
ryu-manager
```

5. Capture OpenFlow Messages with Wireshark:

- Start Wireshark and select your network interface (e.g., `eth0`).
- Apply a display filter to capture only OpenFlow messages. Use the filter expression: `of`.
- Begin capturing packets by clicking the "Start" button in Wireshark.

6. Generate OpenFlow Messages:

In the Mininet terminal, you can use the Mininet CLI to generate OpenFlow messages. For example, you can add a flow rule (FLOW_MOD) or generate traffic (PACKET_OUT).

To add a flow rule (FLOW_MOD):

```
mininet> h1 ovs-ofctl add-flow s1 in_port=1,actions=output:2
```

To generate traffic (PACKET_OUT):

```
mininet> h1 ping -c 1 h2
```

7. Stop Wireshark Capture:

Stop capturing packets in Wireshark when you've generated enough OpenFlow messages.

OUTPUT:

- The Wireshark capture should display OpenFlow messages exchanged between the Ryu controller and the Mininet switch. You will see messages like FLOW_MOD, PACKET_IN, and PACKET_OUT, along with their details.

- Use Wireshark's visualization tools to analyze and inspect the captured OpenFlow messages. You can filter, sort, and drill down into specific messages to understand the communication between the controller and switches.

RESULT:

This setup allows you to observe and analyze the OpenFlow messaging in a simple SDN network. You can further customize the Mininet topology and generate more complex OpenFlow scenarios for testing and analysis.

EX.NO: 3 DATE:		**Create a SDN application that uses the Northbound API to program flow table rules on the switch for various use cases like L2 learning switch, Traffic Engineering, Firewall etc**

AIM:

Develop an SDN application that uses the Northbound API to program flow table rules on SDN switches for different use cases: L2 learning switch, Traffic Engineering, and Firewall.

PROCEDURE:

1. Set Up the Development Environment:

Install the Ryu SDN controller and any necessary Python libraries.

2. Create the Ryu SDN Application:

Create a Python script for your Ryu SDN application, which will implement the Northbound API to program flow rules.

3. L2 Learning Switch Use Case:

In your Ryu application, use the Northbound API to program flow rules for basic L2 learning. For example, when a packet arrives, add a flow entry to the switch's flow table based on the source MAC address and port.

4. Traffic Engineering Use Case:

Implement traffic engineering rules using the Northbound API. For instance, you can define flow rules to optimize traffic paths or prioritize specific traffic based on application requirements.

5. Firewall Use Case:

Implement firewall rules using the Northbound API to drop or allow specific traffic based on criteria such as source/destination IP addresses, ports, or protocols.

6. Run the SDN Application:

Start your Ryu SDN application with the Ryu manager using the following command:

```
ryu-manager your_application.py
```

7. Test the SDN Application:

Create a virtual network or use a Mininet topology for testing your SDN application. Generate traffic to see how the application programs the flow table rules.

OUTPUT:

- For the L2 learning switch use case, the application should program flow rules that enable the switch to learn and forward traffic based on MAC addresses.

- In the traffic engineering use case, the application should optimize traffic paths or prioritize certain types of traffic as per your defined rules.

- In the firewall use case, the application should allow or block traffic based on your defined criteria.

- The application's output should include log messages or other forms of feedback, showing that it is functioning correctly.

RESULT:

The result is a working SDN application that leverages the Northbound API to program flow table rules on SDN switches for different use cases. The application should effectively control network traffic based on the specified policies and rules.

EX.NO: 4 DATE:		**Create a simple end-to-end network service with two VNFs using vim-emu https://github.com/containernet/vim-emu 5) Install OSM and onboard and orchestrate network service.**

AIM:

- Create a simple end-to-end network service with two VNFs using vim-emu.
- Install OSM and onboard and orchestrate the network service.

PROCEDURE:

1. Set Up the Environment:

Make sure you have a Linux system. You can set up a virtual machine or a dedicated system.

- Install the required dependencies, including Docker and Docker Compose.

2. Install vim-emu:

Follow the instructions in the vim-emu GitHub repository (https://github.com/containernet/vim-emu) to install vim-emu.

3. Create Network Topology:

Define a network topology using vim-emu. You can create a Python script that specifies the network, VNFs, and their interconnections. For example, create a file named **`network_topology.py`:**

```python
from mininet.net import Containernet
from mininet.node import Docker
from mininet.link import TCLink
from mininet.cli import CLI

net = Containernet()

vnf1 = net.addDocker('vnf1', dimage="vnf1_image")
vnf2 = net.addDocker('vnf2', dimage="vnf2_image")
net.addLink(vnf1, vnf2, cls=TCLink)

net.start()

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

4. Create VNF Docker Images:

Build Docker images for your VNFs. Create a `Dockerfile` for each VNF, specifying its requirements and configurations. Build the images using Docker commands.

5. Launch the Network Topology:
Run your network topology with vim-emu:

```
sudo python network_topology.py
```

This will start the network and deploy the VNFs.

6. **Install OSM:**
Follow the instructions on the OSM GitHub repository (https://osm.etsi.org) to install OSM, which includes installing the OSM client and server components.

7. Onboard the Network Service:
Use the OSM client to onboard your network service. You'll need to provide a descriptor (e.g., a TOSCA YAML file) for your service.

```
osm ns-create <your_network_service_descriptor_file>
```

8. Instantiate the Network Service:
Instantiate the network service using OSM:

```
osm ns-instantiate <ns-instance-name> <your_network_service_name>
```

Replace `<ns-instance-name>` with a suitable name for your instantiated service.

OUTPUT:

When running the network topology script with vim-emu, you should see the network and VNFs being deployed. You can test connectivity and traffic between VNFs.

After installing OSM and onboarding the network service descriptor, you should see your network service listed within OSM.

When instantiating the network service, OSM will orchestrate the deployment of the VNFs and connect them according to your descriptor.

RESULT:

The result is a fully orchestrated end-to-end network service with two VNFs, established using vim-emu and OSM. The network service is now ready to accept and process traffic as specified in your descriptor.