Configuration Manual

MSc Research Project
Cybersecurity

Sankaran Selvam
Student ID: X21217467

School of Computing
National College of Ireland

Supervisor: Evgeniia Jayasekera

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	SANKARAN SELVAM
Student ID:	X2127467
Programme:	MSCCYB1 **Year:** 2022-2023
Module:	Research Project/Internship
Lecturer:	Evgeniia Jayasekera
Submission Due Date:	14/08/2023
Project Title:	Collaborative approach of Detection of DDOS attack on SDN Networks
Word Count:	1000 **Page Count:** 7

Signature: Sankaran Selvam

Date: 13/8/2023

Configuration Manual

Sankaran Selvam
Student ID: X21217467

1 Introduction

The purpose of this document is to provide a complete description on the project setup and perquisite steps performed to create a test environment. This document contains the system requirement and software used to run the test bed. The detail descriptions such as version and dependencies that need to be installed are mentioned in this document which helps in recreating the proposed setup accurately.

2 System Configuration of the Testbed/ Propose Model

The proposed system requires at least two Virtual machines that is capable of running Snort, Ryu Controller, machine learning model and network simulator like Mininet. Further the system should be able to run all these tools simultaneously and perform other process like dataset creation, data processing for machine learning module. For this purpose, we used the host machine with the below configuration.

Host Machine Specification:

- Performance-oriented CPU: Ryzen7 5800H, stable overclock at 4.2ghz
- 16 GB DDR4 ram.
- External GPU Nvidia RTX 3060 with 6GB RAM.
- 2TB SSD storage

Virtual box is used to create two virtual machines with the below configuration

RYU Controller VM:

- 1 Core Processor
- 4GB ram
- 30GB Storage
- OS: Ubuntu 64-bit

Mininet VM:

- 2 Core Processor
- 4GB ram
- 30GB Storage
- OS: Ubuntu 64-bit

Below are the Software used in our VM:

- 64-bit Windows 11 operating system is running on the Host Machine.
- Python 3.10.5

- Mininet 2.3.1b4
- RYU SDN Controller
- SNORT 2.9.6
- Hping3
- Machine learning libraries like Numpy, Panda and SKlearn were used

3 Implementation Procedures

The setup required two virtual machines to run RYU controller and Mininet for network simulation. We used Virtual box to create VM in our project and both the VM is running on the Ubuntu operating system. The minimum configuration was set to the VM template as mentioned above in the system configuration. For Ryu controller we used minimal installation whereas for the Mininet VM we used Ubuntu with GUI. Below image (Figure 1) represents the actual VM setup on our host machine.

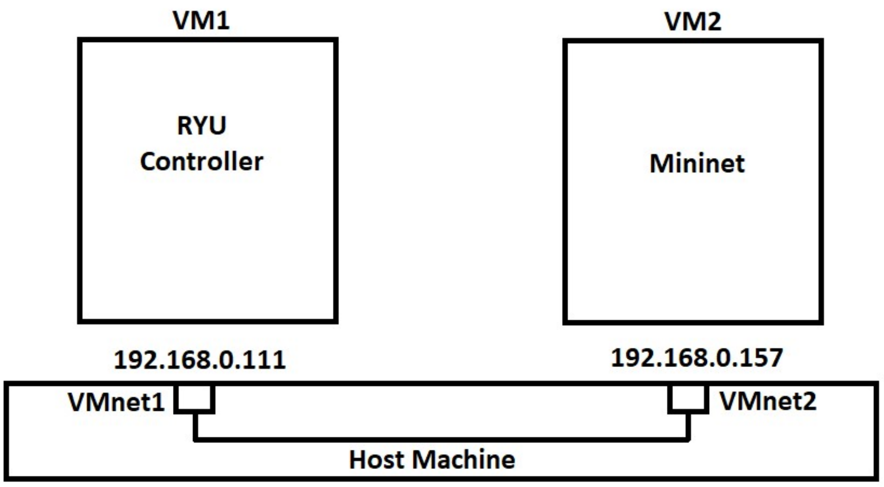
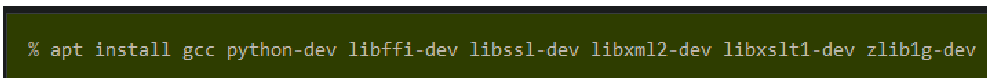


Figure 1 SDN Testbed experimental setup

Once the VMs were created with the help of virtual box we installed the operating system. After successful installation we proceed to install the required dependencies and tools on the VM for creating the proposed system.

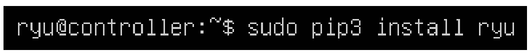
3.1 RYU Installation:

On both the machine the python was installed for scripting and Ryu framwork was installed using python pip module (SDN, 2021). Further, all the machine learning packages were installed, imported and programmed using python. The below command line (Figure 3) was used to install Ryu controller and the below mentioned (Figure 2) dependencies were installed prior to the installation of the Ryu controller.



```
% apt install gcc python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev zlib1g-dev
```

Figure 2 Dependencies

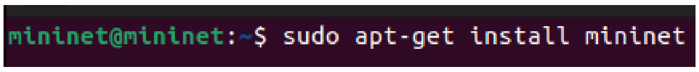


```
ryu@controller:~$ sudo pip3 install ryu
```

Figure 3 Ryu Installation

3.2 Mininet Installation:

Mininet is a tool used for network simulation. It helps in creating different network topology for software defined network (mininet, 2019). It is easy to create a topology and connect to the internal or external controller using the controller IP address to manage the traffic in the simulated network. The below command line (Figure 4) was used to install Mininet.

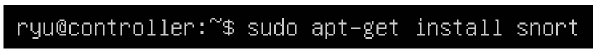


```
mininet@mininet:~$ sudo apt-get install mininet
```

Figure 4 Mininet Installation

3.3 Snort Installation:

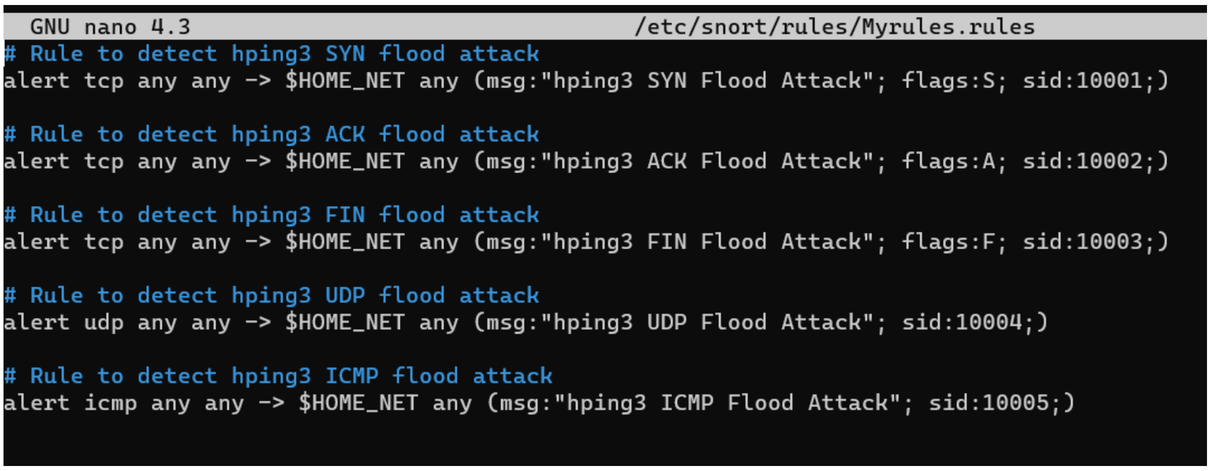
Snort is a powerful opensource network intrusion detection tool. Snort monitors the network in real time for any malicious activities and it generates alerts based on the predefined rules (Snort, 2019). It also allows user to create custom rules to generate alerts which helps in taking action and preventive measures. We installed snort using the below cmd (Figure 5) in to Ryu machine and modified the controller code to integrate it with snort.



```
ryu@controller:~$ sudo apt-get install snort
```

Figure 5 Snort Installation

After successful installation of Snort IDS, we have created a new alerts rules as shown below (Figure 6) and configured in the snort.conf file to generate alerts based on the rules we have created.



```
  GNU nano 4.3                                    /etc/snort/rules/Myrules.rules
# Rule to detect hping3 SYN flood attack
alert tcp any any -> $HOME_NET any (msg:"hping3 SYN Flood Attack"; flags:S; sid:10001;)

# Rule to detect hping3 ACK flood attack
alert tcp any any -> $HOME_NET any (msg:"hping3 ACK Flood Attack"; flags:A; sid:10002;)

# Rule to detect hping3 FIN flood attack
alert tcp any any -> $HOME_NET any (msg:"hping3 FIN Flood Attack"; flags:F; sid:10003;)

# Rule to detect hping3 UDP flood attack
alert udp any any -> $HOME_NET any (msg:"hping3 UDP Flood Attack"; sid:10004;)

# Rule to detect hping3 ICMP flood attack
alert icmp any any -> $HOME_NET any (msg:"hping3 ICMP Flood Attack"; sid:10005;)
```

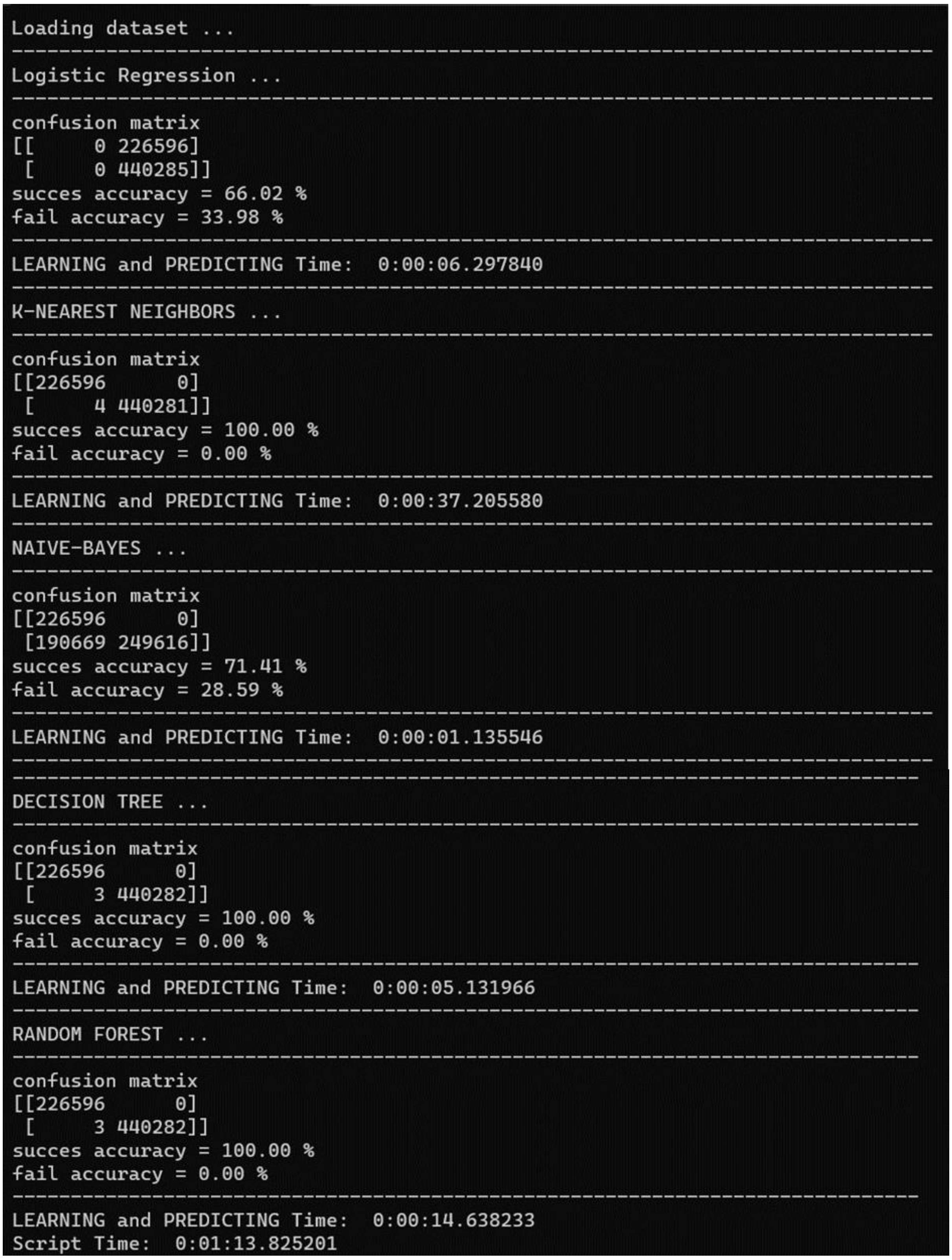
Figure 6 Snort Rules

3.4 Dataset creation:

To ensure the accurate of the model we created our own data set by generating a legitimate traffic and DDos traffic using the python script and recorded the data in to the CSV file. The Synthetic dataset was created for our testing purpose.

3.5 ML Model Selection for integration:

We have used synthetic dataset we generated to training the ML model. To find the which algorithm returns better accuracy results we used the algorithms shown below in Figure 7. The image showcases the accuracy of each model, we noted that Random forest and Decision Tree algorithm showed 100% accuracy rate with the generated dataset.

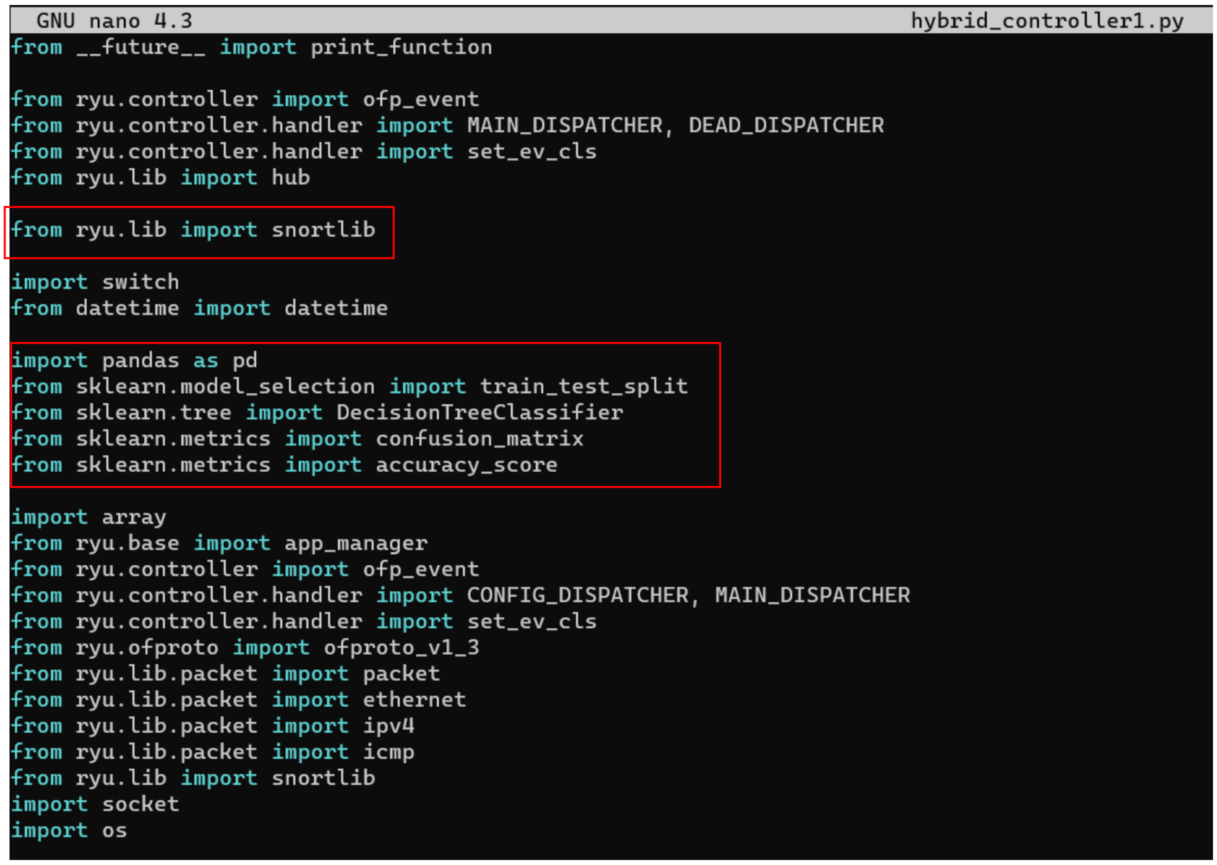


```
Loading dataset ...
--------------------------------------------------------------------------------
Logistic Regression ...
--------------------------------------------------------------------------------
confusion matrix
[[     0 226596]
 [     0 440285]]
succes accuracy = 66.02 %
fail accuracy = 33.98 %
--------------------------------------------------------------------------------
LEARNING and PREDICTING Time:  0:00:06.297840
--------------------------------------------------------------------------------
K-NEAREST NEIGHBORS ...
--------------------------------------------------------------------------------
confusion matrix
[[226596      0]
 [     4 440281]]
succes accuracy = 100.00 %
fail accuracy = 0.00 %
--------------------------------------------------------------------------------
LEARNING and PREDICTING Time:  0:00:37.205580
--------------------------------------------------------------------------------
NAIVE-BAYES ...
--------------------------------------------------------------------------------
confusion matrix
[[226596      0]
 [190669 249616]]
succes accuracy = 71.41 %
fail accuracy = 28.59 %
--------------------------------------------------------------------------------
LEARNING and PREDICTING Time:  0:00:01.135546
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
DECISION TREE ...
--------------------------------------------------------------------------------
confusion matrix
[[226596      0]
 [     3 440282]]
succes accuracy = 100.00 %
fail accuracy = 0.00 %
--------------------------------------------------------------------------------
LEARNING and PREDICTING Time:  0:00:05.131966
--------------------------------------------------------------------------------
RANDOM FOREST ...
--------------------------------------------------------------------------------
confusion matrix
[[226596      0]
 [     3 440282]]
succes accuracy = 100.00 %
fail accuracy = 0.00 %
--------------------------------------------------------------------------------
LEARNING and PREDICTING Time:  0:00:14.638233
Script Time:  0:01:13.825201
```

Figure 7 ML Models

3.6 Hybrid Controller:

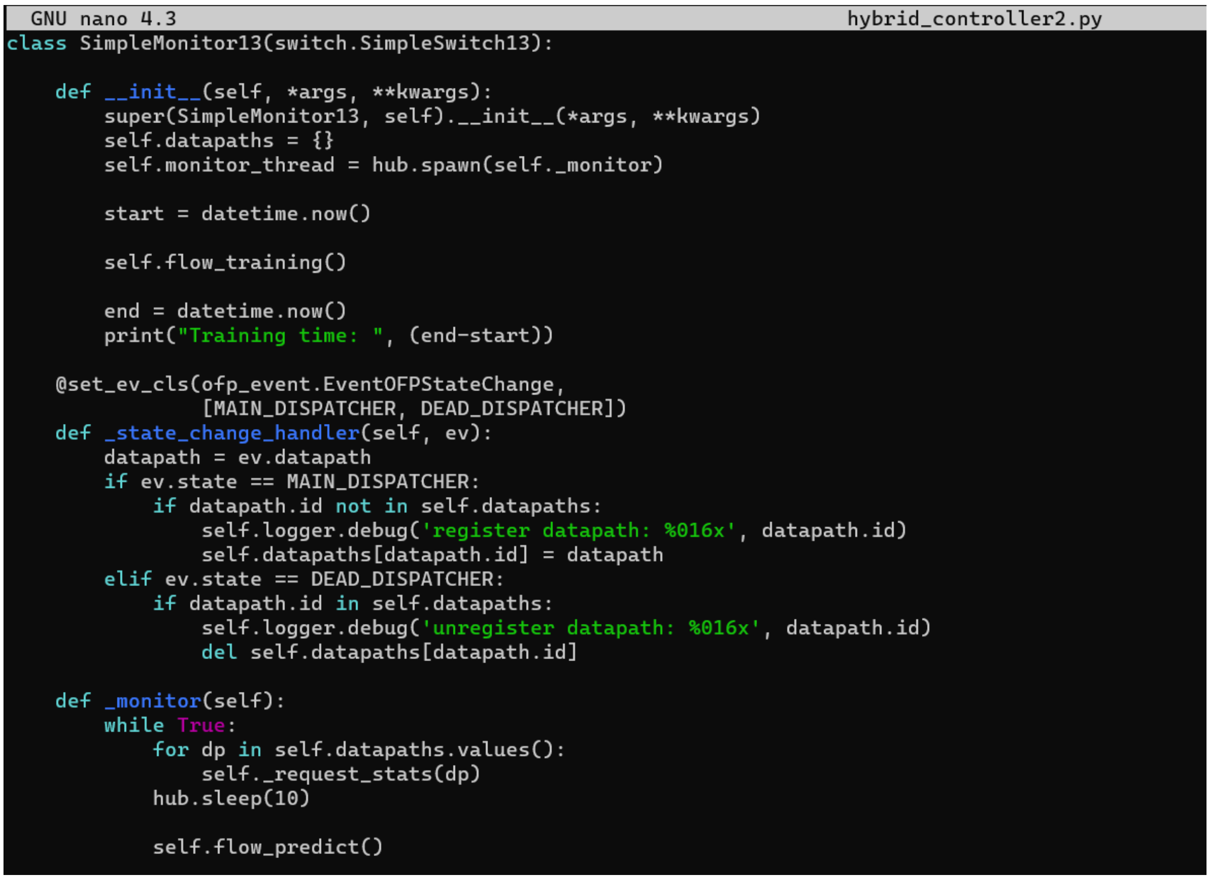
The proposed system is a modified controller, it is a combination of two different classes that are SimpleMonitor13 and SimpleSwitchSnort. The below image (Figure 8) shows the modules imported and used in the Ryu controller for integrating snort and ML module



```
  GNU nano 4.3                                                    hybrid_controller1.py
from __future__ import print_function

from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub

from ryu.lib import snortlib

import switch
from datetime import datetime

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

import array
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ipv4
from ryu.lib.packet import icmp
from ryu.lib import snortlib
import socket
import os
```

Figure 8 Hybrid controller Python Module

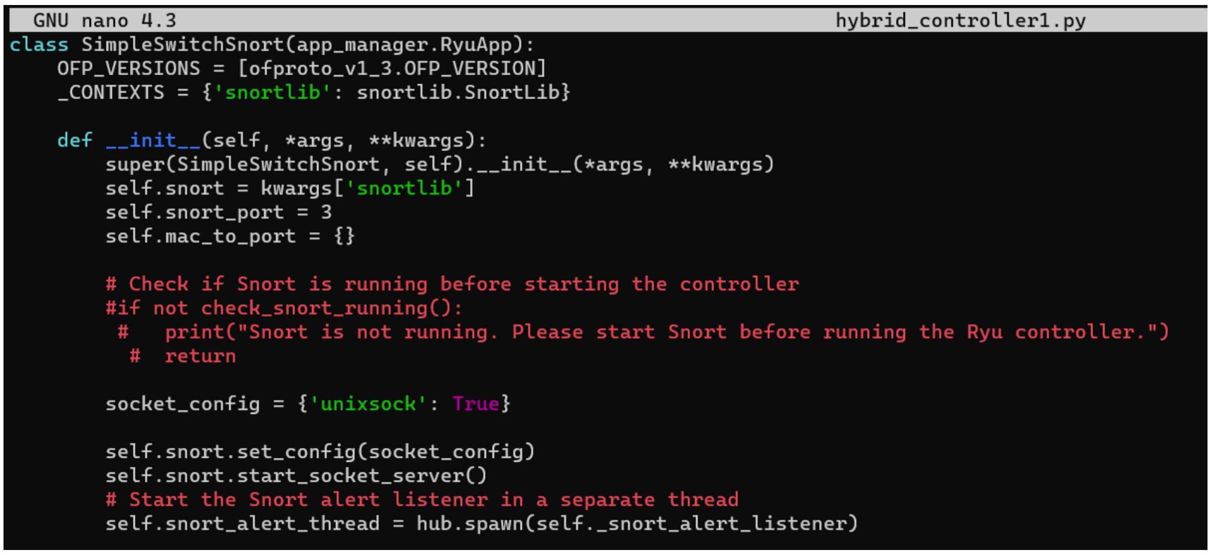
The Below figure (Figure 9) shows SimpleMonitor13 Class is responsible for monitoring the network flows and applying machine learning based traffic analysis using Decision Tree model. The monitor() function periodically requests flow statistics, and the flow_predict() function predicts the traffic type using the trained machine learning model with the synthetic datasets generated.



```
  GNU nano 4.3                                                    hybrid_controller2.py
class SimpleMonitor13(switch.SimpleSwitch13):

    def __init__(self, *args, **kwargs):
        super(SimpleMonitor13, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)

        start = datetime.now()

        self.flow_training()

        end = datetime.now()
        print("Training time: ", (end-start))

    @set_ev_cls(ofp_event.EventOFPStateChange,
                [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if datapath.id not in self.datapaths:
                self.logger.debug('register datapath: %016x', datapath.id)
                self.datapaths[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.datapaths:
                self.logger.debug('unregister datapath: %016x', datapath.id)
                del self.datapaths[datapath.id]

    def _monitor(self):
        while True:
            for dp in self.datapaths.values():
                self._request_stats(dp)
            hub.sleep(10)

            self.flow_predict()
```

Figure 9 Ryu ML code

The Below figure (Figure 9) of SimpleSwitchSnort Class is an extension of the Ryu framework's RyuApp class which his responsible for handling OpenFlow switch features and integrating with Snort alerts (sdn, 2015). The _dump_alert function processes Snort alerts and triggers machine learning-based actions based on the alerts.



```
  GNU nano 4.3                                                   hybrid_controller1.py
class SimpleSwitchSnort(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'snortlib': snortlib.SnortLib}

    def __init__(self, *args, **kwargs):
        super(SimpleSwitchSnort, self).__init__(*args, **kwargs)
        self.snort = kwargs['snortlib']
        self.snort_port = 3
        self.mac_to_port = {}

        # Check if Snort is running before starting the controller
        #if not check_snort_running():
         #    print("Snort is not running. Please start Snort before running the Ryu controller.")
          #   return

        socket_config = {'unixsock': True}

        self.snort.set_config(socket_config)
        self.snort.start_socket_server()
        # Start the Snort alert listener in a separate thread
        self.snort_alert_thread = hub.spawn(self._snort_alert_listener)
```

Figure 10 Ryu Snort Integration

This is the complete description of our test environment setup; we have covered all the tool and dependencies that we have installed and used on our test environment. Also, we specified the system configurations, OS and software versions that we used in order for the reader to replicated the testbed and use it for future enhancements.

References

mininet (2019) ‘mininet/mininet’. GitHub. Available at: https://github.com/mininet/mininet.
sdn, faucet (2015) ‘snort_integrate.rst’. GitHub. Available at: https://github.com/faucetsdn/ryu/blob/master/doc/source/snort_integrate.rst (Accessed: 10 August 2023).
SDN, F. (2021) ‘faucetsdn/ryu’. GitHub. Available at: https://github.com/faucetsdn/ryu.
Snort (2019) ‘Snort - Network Intrusion Detection & Prevention System’. Snort.org. Available at: https://www.snort.org/.