CO390 Seminar
Report

Software Defined Network and Emulation using mininet

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

Bachelor of Technology
in
Computer Science and Engineering

Submitted by

Roll No	Name of Student
15CO112	Aswanth P P

Under the guidance of
Ms. C Aishwarya

Department of Computer Science and Engineering
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
Surathkal, Karnataka, India – 575 025

Even Semester 2017-18

Department of Computer Science and Engineering

National Institute of Technology Karnataka

Certificate

This is to certify that this is a bonafide record of the project presented by the students whose names are given below during the Even Semester 2017-18 in partial fulfilment of the requirements of the degree of Bachelor of Technology in Computer Science and Engineering.

Roll No	Name of Student
15CO112	Aswanth P P

Ms. C Aishwarya
(Course Instructor)

Date:

Abstract

Software defined networking (SDN) is an approach to using open protocols, such as OpenFlow, to apply globally aware software control at the edges of the network to access network switches and routers that typically would use closed and proprietary firmware.It offers numerous benefits including on-demand provisioning, automated load balancing, streamlined physical infrastructure and the ability to scale network resources in lockstep with application and data needs.

Originally, SDN focused solely on separation of the control plane of the network, which makes decisions about how packets should flow through the network from the data plane of the network, which actually moves packets from place to place. When a packet arrives at a switch in the network, rules built into the switch's proprietary firmware tell the switch where to forward the packet. The switch sends every packet going to the same destination along the same path, and treats all the packets the exact same way. In a classic SDN scenario, rules for packet handling are sent to the switch from a controller, an application running on a server somewhere, and switches (also known as data plane devices) query the controller for guidance as needed, and provide it with information about traffic they are handling. Controllers and switches communicate through a controller's south bound interface, usually OpenFlow, although other protocols exist.

Where a traditional network would use a specialized appliance such as a firewall or link-load balancer, an SDN deploys an application that uses the controller to manage data plane behavior. Applications talk to the controller though its north-bound interface. As of the end of 2014, there is no formal standard for the application interface of the controller to match OpenFlow as a general south-bound interface. It is likely that the OpenDaylight controller's northbound application program interface (API) may emerge as a defacto standard over time, given its broad vendor support.

Contents

Chapter 1

Introduction

Software-defined networking (SDN) technology is a novel approach to cloud computing that facilitates network management and enables programmatically efficient network configuration in order to improve network performance and monitoring. SDN is meant to address the fact that the static architecture of traditional networks is decentralized and complex while current networks require more flexibility and easy troubleshooting. SDN suggests to centralize network intelligence in one network component by disassociating the forwarding process of network packets (Data Plane) from the routing process (Control plane). The control plane consists of one or more controllers which are considered as the brain of SDN network where the whole intelligence is incorporated.

The main issue that facing in software define networking fields are how to experiment in SDN without procuring the hardware.There are lot of network simulators are available.The main one which has all the functionality of SDN is Mininet a software emulator for prototyping a large network on a single machine. Mininet can be used to quickly create a realistic virtual network running actual kernel, switch and software application code on a personal computer. Mininet allows the user to quickly create, interact with, customize and share a software-defined network (SDN) prototype to simulate a network topology that uses Openow switches.

Chapter 2

Technologies Used

2.1 mininet

Mininet is a network emulator, or perhaps more precisely a network emulation orchestration system. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. A Mininet host behaves just like a real machine; you can ssh into it (if you start up sshd and bridge the network to your host) and run arbitrary programs (including anything that is installed on the underlying Linux system.) The programs you run can send packets through what seems like a real Ethernet interface, with a given link speed and delay. Packets get processed by what looks like a real Ethernet switch, router, or middlebox, with a given amount of queueing.

In short, Mininet's virtual hosts, switches, links, and controllers are the real thing they are just created using software rather than hardware and for the most part their behavior is similar to discrete hardware elements. It is usually possible to create a Mininet network that resembles a hardware network, or a hardware network that resembles a Mininet network, and to run the same binary code and applications on either platform.

2.2 pox controller

POX is an open source development platform for Python-based software-defined networking (SDN) control applications, such as OpenFlow SDN controllers. POX, which enables rapid development and prototyping, is becoming more commonly used than NOX, a sister project.

Chapter 3

Characteristics of SDN

3.1 Traditional Networks vs SDN

The traditional approach to networking is characterized by two main factors:

- Network functionality is mainly implemented in a dedicated appliance. In this case, dedicated appliance refers to one or multiple switches,routers and/or application delivery controllers.
- Most functionality within this appliance is implemented in dedicated hardware.An Application Specific Integrated Circuit (or: ASIC) is often used for this purpose.

Software-defined networking, or SDN, allows network engineers and admins to respond quickly to changing business environments. When it comes to SDN vs. traditional networking, the latter is often viewed separately from SDN. While a typical data center may not seem appropriate for SDN, software-defined technologies can be used effectively and efficiently in a harmonious relationship with more traditional networking. With the right plan, an IT team can create a single, unified infrastructure.

3.2 Architecture

The switches in SDN have separate control and data plane.Hence There will have centralized control on each flow.The controller in SDN is programmable, hence can bring dynamic changes to network flow.
Southbound API : Controller to network element-OpenFlow.That is information ragarding flow table entries.
Northbound API : Controller to application.This will be the programming

policies such as blocking IP or discarding certain flows.That is this API provides information regarding firwall blocking or NAT box functions

3.2.1 Sample Topology

The sample topology for Software Defined Network architecture

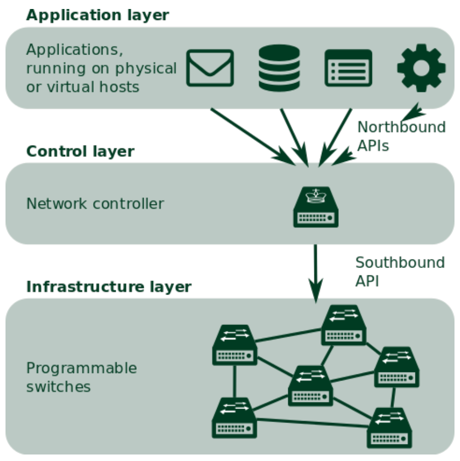

Figure 3.1: SDN sample architecture along with other network application

- Application here may be either firewall,NAT and other network function utilities
- Controller can be programmed by the applications to give priorities to a flow

3.2.2 Packet Forwarding Rule

Switches will contact controller only if there is no flow table entry for incoming packets [O,D] pair.

- Forward rule is installed on all switches along the path
- First packet of flow contact controller
- Later packets of flow forwarded by switch without controllers interventio

n

Chapter 4

Implementation

As explained software defined networking need controllers and dump switches which differs a lot from traditional switch where these intelligent and data plane are coupled together .Hence how to experiment the protocols and without procuring the hardware.
Mininet is one of a kind that is An Instant Virtual Network on your Laptop (or other PC) where it support SDN features where other simulator does not provide. Also have the provision to add user defined functions and protocols to test it in virtual environment before the actual implementation.

4.1 Installation

Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine .Also Mininet is actively developed and supported, and is released under a permissive BSD Open Source license.Source code is available on github can be installed free of cost. Steps to install mininet

- Clone the github repository :

  ```
   git clone git://github.com/mininet/mininet
  ```

- Change directory to cloned repository :

  ```
   cd mininet
  ./util/install.sh -a
  ```

- Verify the installation

  ```
  sudo mn --test pingall
  ```

4.2 Execution

After installation we can use mininet to emulate the experiments using comand line (terminal)
command to launch default topology

```
sudo mn
```

A default topology will be loaded which has two hosts that are connected to a switch(OVSSwitch) and a default controller(ovs-controller)

Sample commands available in mininet
mininet: nodes : To display all nodes
mininet: net : To display all links
mininet: dump : To display information about all nodes
Use help to get more CLI commands.

4.3 Designing custom topology

Mininet provides some inbuilt typologies like linear,single,minimal,reversed,torus and tree.
Example:

```
        sudo mn --topo linear,4
```

creates a topology of 4 nodes, each connected with a separate switch

```
        sudo mn --topo single,4
```

creates a topology 4 nodes, each connected with a single switch

```
        sudo mn --link tc,bw=10,delay=10ms
```

set bandwidth to 10Mbps and delay to 10ms for all the links in network

4.3.1 Creating Topology using mininet GUI tool

Redirect terminal to mininet directory and run this command

```
        sudo python ./examples/miniedit.py
```

This will open window where user can drag and drop nodes,switches and other network elements to create topology

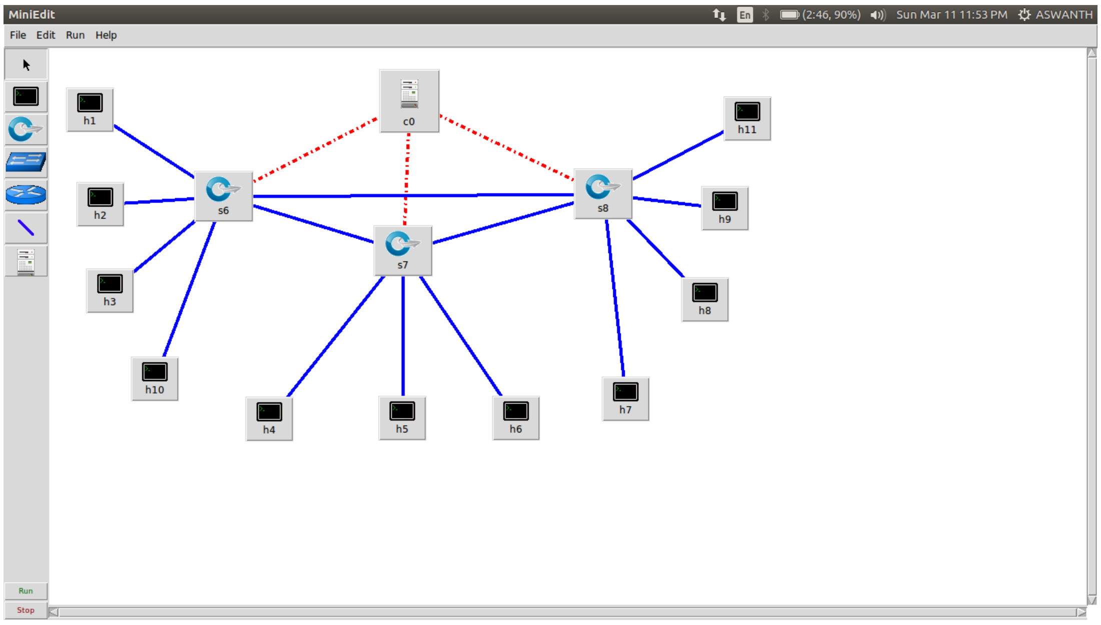

Figure 4.1: GUI in mininet to design custom topology

4.3.2 Launch custom topology in mininet

Since we are only created topology we are not deployed it over mininet environment.For that we need to convert it to python script.

Topology created by GUI tool can exported to python script by selecting **File-¿Export Level 2 Script** option from the above GUI and save it in custom folder of mininet

Two ways to launch the topology :

```
1.  sudo mn --custom ./custom/topo-2sw-2host.py --topo mytopo
```

Here topology name is mytopo and saved in custom folder.That is toplogy is hardcoded in python without using GUI tool

```
2.  sudo python ./custom/mycustom.py
```

Above command is useful only if mycustom.py is exported as level 2 script from GUI tool

4.4 Installing Different Controllers

An SDN controller is an application in software-defined networking (SDN) that manages flow control to enable intelligent networking. SDN controllers are based on protocols, such as OpenFlow, that allow servers to tell switches where to send packets.

4.4.1 Usage of remote controller in mininet

mininet supports the usage of remote controller . steps to execute remote controller

- Run remote controller(which is downloaded need to be used instead of default on) in a terminal
- Execute mininet topology specifying controller as remote controller

  ```
  sudo mn --topo linear,4--controller=remote,ip=127.0.0.1,port=6633
  ```

  This will create mininet emulated environment which uses remote controller running parallel

4.4.2 Different Controllers

There are many open source controllers are available SDN ,few among them are :

- **NOx Controller** : It is an open source development platform for C++-based software-defined networking (SDN) control applications
  Link : https://github.com/noxrepo/nox
- **POX Controller** : It is an open source development platform for Python-based software-defined networking (SDN) control applications, such as OpenFlow SDN controllers. POX, which enables rapid development and prototyping, is becoming more commonly used than NOX, a sister project.
  Link : https://github.com/noxrepo/pox
- **Floodlight Open SDN Controller** : It is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. It is supported by a community of developers including a number of engineers from Big Switch Networks. Link : https://github.com/floodlight/floodlight

Chapter 5

Source Code

5.1 Topology creation

Python code to create topology consists of three SDN switches and one controller ,such that each switch forms a start toplogy and each switch have 3 nodes each

```
import sys
import getopt
import time
from os import popen
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import sendp, IP, UDP, Ether, TCP
from random import randrange

def generateSourceIP():
    #not valid for first octet of IP address
    not_valid = [10, 127, 254, 1, 2, 169, 172, 192]

    #selects a random number in the range [1,256)
    first = randrange(1, 256)

    while first in not_valid:
        first = randrange(1, 256)

    #eg, ip = "100.200.10.1"
    ip = ".".join([str(first), str(randrange(1,256)),
    str(randrange(1,256)), str(randrange(1,256))])
    return ip

#start, end: given as command line arguments. eg, python traffic.py -s 2 -e 65
def generateDestinationIP(start, end):
    first = 10
    second = 0;
    third = 0;

    #eg, ip = "10.0.0.64"
    ip = ".".join([str(first), str(second), str(third), str(randrange(start,end)
    return ip

def main(argv):
    try:
        opts, args = getopt.getopt(sys.argv[1:], 's:e:', ['start=','end='])
    except getopt.GetoptError:
        sys.exit(2)
    for opt, arg in opts:
        if opt =='-s':
            start = int(arg)
        elif opt =='-e':
            end = int(arg)
    #open interface eth0 to send packets
    interface = popen('ifconfig | awk \'/eth0/ {print $1}\'').read()

    for i in xrange(1000):
        packets = Ether() / IP(dst = generateDestinationIP (start, end), src = g
        print(repr(packets))

    #rstrip() strips whitespace characters from the end of interface
        sendp(packets, iface = interface.rstrip(), inter = 0.1)

if __name__ == '__main__':
  main(sys.argv)
```

5.2 Generate traffic

The source code to generate normal traffic among emulated nodes

```
import sys
import getopt
import time
from os import popen
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import sendp, IP, UDP, Ether, TCP
from random import randrange

def generateSourceIP():
    #not valid for first octet of IP address
    not_valid = [10, 127, 254, 1, 2, 169, 172, 192]

    #selects a random number in the range [1,256)
    first = randrange(1, 256)

    while first in not_valid:
        first = randrange(1, 256)

    #eg, ip = "100.200.10.1"
    ip = ".".join([str(first), str(randrange(1,256)), str(randrange(1,256)), str
    return ip

#start, end: given as command line arguments. eg, python traffic.py -s 2 -e 65
def generateDestinationIP(start, end):
    first = 10
    second = 0;
    third = 0;

    #eg, ip = "10.0.0.64"
    ip = ".".join([str(first), str(second), str(third), str(randrange(start,end)
    return ip

def main(argv):
    try:
        opts, args = getopt.getopt(sys.argv[1:], ’s:e:’, [’start=’,’end=’])
    except getopt.GetoptError:
        sys.exit(2)
    for opt, arg in opts:
        if opt ==’-s’:
            start = int(arg)
        elif opt ==’-e’:
            end = int(arg)
        if start == ’’:
            sys.exit()
        if end == ’’:
            sys.exit()
    #open interface eth0 to send packets
    interface = popen(’ifconfig | awk \’/eth0/ {print $1}\’’).read()

    for i in xrange(1000):
        packets = Ether() / IP(dst = generateDestinationIP (start, end), src = g
        print(repr(packets))

    #rstrip() strips whitespace characters from the end of interface
        sendp(packets, iface = interface.rstrip(), inter = 0.1)

if __name__ == ’__main__’:
  main(sys.argv)
```

Chapter 6

Conclusion

Traditional networks are complex and hard to manage. One of the reasons is that the control and data planes are vertically integrated and vendor specific. Another, concurring reason, is that typical networking devices are also tightly tied to line products and versions.

SDN created an opportunity for solving these longstanding problems. Some of the key ideas of SDN are the introduction of dynamic programmability in forwarding devices through open southbound interfaces, the decoupling of the control and data plane, and the global view of the network by logical centralization of the network brain. While data plane elements became dumb, but highly efficient and programmable packet forwarding devices, the control plane elements are now represented by a single entity, the controller or NOS.

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.

- Provides a simple and inexpensive network testbed for developing OpenFlow applications
- 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

References

[1] SDN concepts : https://github.com/sdnds-tw/awesome-sdn

[2] Why mininet?: http://openvswitch.org/support/ovscon2015/16/1305-lantz.pdf

[3] Mininet installation: http://mininet.org/download/

[4] Mininet commands (topology, link variation, etc): http://mininet.org/walkthrough/

[5] Miniedit: http://www.brianlinkletter.com/how-to-use-miniedit-mininets-graphical-user-interface/

[6] Mininet with POX controller: http://www.brianlinkletter.com/using-the-pox-sdn-controller/