Proceedings of the 2nd International Conference on Computer and Information Science and Technology (CIST'16)
Ottawa, Canada – May 11 – 12, 2016
Paper No. 133

Modeling Software Defined Networks using Mininet

Casimer DeCusatis[1], PhD, Aparicio Carranza[2], PhD and Jean Delgado-Caceres[2]
[1]Marist College, Poughkeepsie, NY USA
[2]New York City College of Technology of The City University of New York (CUNY)
Brooklyn, New York USA

Abstract - A software defined network (SDN) implements separation of the network data and control planes, combined with centralized management. This is becoming an important feature of cloud computing environments, enterprise data centers, and telecommunication service providers, all of whom are implementing software defined data centers. In this paper, we provide a tutorial on the use of network modeling tools such as Mininet to investigate the behavioral characteristics of SDN. Beginning with a default Mininet installation in a virtual machine (VM), we describe how to instantiate SDN features such as the open source network controller, Open Daylight. Commands to spawn multiple instances of servers and network nodes will be discussed, along with recommendations on network latency and scale. The resulting models have applications to network education and training as well as providing a way to evaluate SDN configurations prior to production deployment.

1. Introduction

In recent years, the design of software defined data centers has received increased attention as a means of increasing the deployment speed for new applications [1]. Since both server and storage virtualization are relatively mature, a key enabler for this approach is the creation of software-defined networks (SDN). Some of the distinguishing features of an SDN network include separation of the data and management/control planes; a shift towards centralized management; and abstraction of key network attributes through an application programming interface (API), which enables the creation of virtual, automated network flows [1, 2].

The abstractions provided by SDN can potentially reduce management complexity and standardize traffic routing mechanisms, which contribute to better network reliability, availability, and serviceability. Further, introducing a centralized SDN network controller enables end-to-end network visibility and faster reconfiguration in response to disasters. For example, a fully automated system could be programmed to invoke disaster recovery protocols without the presence of a human administrator. SDN can reduce network recovery times, and make it easier to schedule and automate periodic testing of a business continuity plan. Prior to the introduction of SDN, re-provisioning a data center network could take days or weeks, while the network between multiple data centers might require weeks or months [3]. SDN makes it possible to build the physical network infrastructure once, then reconfigure it using only software. The resulting in location agnostic networks reduce reconfiguration time to a few minutes, the same order of magnitude as provisioning new virtual machines (VMs) on a server.

In this paper, we consider the use of Mininet to model the behavior of SDN networks. We provide a brief tutorial on the installation and use of Mininet in a virtual machine (VM) environment, and discuss the requirements for installing the Open Daylight open source SDN controller. Scalability is discussed, both in terms of host server limitations and virtual network response times. We provide a framework for using this approach to create and simulate SDN networks for both educational and research purposes.

2. Mininet Installation and Configuration

Mininet is a popular open source network emulator capable of creating networks of virtual hosts, switches, and SDN controllers [4]. Mininet was created in Python and provides a Python API for user customization (*though it does rely on some C programming utilities*). It can run in a VM based on Linux Ubuntu server version 14.04, and using process-based virtualization it can create hundreds or thousands of virtual host and network instances on a single operating system. The Mininet hosts run standard Linux operating systems, and Mininet switches running Linux can support OpenFlow (note that currently Mininet does not support non-Linux compatible SDN controllers or OpenFlow switches). Mininet provides a way

to emulate system behavior (*subject to any limitations of the simulation hardware*), and since it runs a real Linux kernel and network stack (including compatible kernel extensions), any code developed using Mininet can be transferred to production-ready hardware with minimal changes. Mininet includes an OpenFlow-aware command line interface, which supports a basic set of network topologies and allows highly flexible custom topologies.

We experimented with Mininet 2.2.1 running under Ubuntu Linux on a Toshiba laptop using an Intel Core i7-4700MQ CPU processor at a 2.40GHz x 8, 5.7 GB of Memory and natively running Ubuntu 14.04 from the CLI by retrieving the code from its online repository (*which also checks automatically to insure that the latest version of Mininet is being used*) using the command

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

The basic Mininet install can then be completed using the command

$ mininet/util/install.sh

A simple default virtual network consisting of two hosts, represented as h1 and h2, a switch, s1 and a controller, c0 (as shown in Figure 1) can be created using the command

$ sudo mn

Similarly, a Mininet instance can be created in a VM, for example using VirtualBox which is available from the Mininet web site [5]). Note that since the Mininet VM is based on Ubuntu Server, it may not use predictable network interface names like *enp0s3*, so the CLI will display interface names such as *eth0*. This has been corrected for Ubuntu 15.10 and higher. Once the installation is completed, the ping command can be applied to verify that connectivity exists within hosts.

$ sudo mn --test pingall

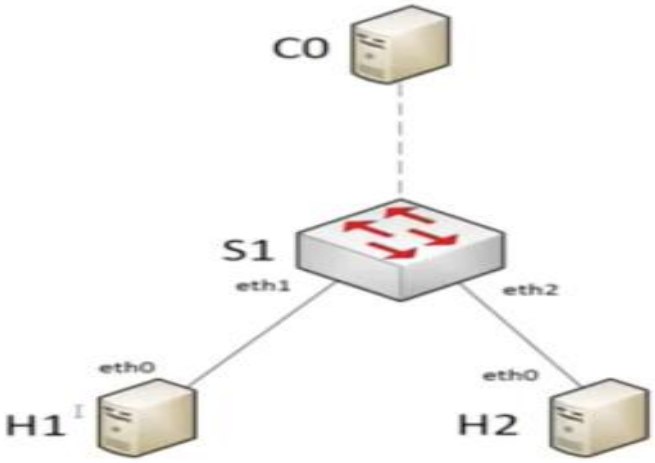

Fig. 1: Graphical representation of network created by command sudo mn.

Many other network configurations can be created in a similar manner. For example, the following command spawns a single switch with three attached hosts, each of which is assigned a MAC address and static IP address:

$ sudo mn --arp --topo single,3 --mac --switch ovsk --controller remote

where the parameters specified in this command include the following:

–mac: Auto set MAC addresses
–arp: Populate static ARP entries of each host in each other
–switch: ovsk refers to kernel mode OVS
–controller: remote controller can take IP address and port number as options

As another example, the following command spawns two switches connected by an inter-switch link (ISL) with one host attached to each switch:

$ sudo mn --topo linear --switch ovsk --controller remote

Some other commonly used CLI management commands include the following:

h1 ping h2 This is used to ping a host and test reachability
nodes To view the list of nodes available
dpctl To control and edit flow tables.
iperf To perform a TCP bandwidth test between hosts (see Figure 2)

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h2
.*** Results: ['51.7 Gbits/sec', '51.8 Gbits/sec']
```

Fig. 2: TCP Speed Test on Mininet.

We can also open a terminal window and SSH into the Mininet VM with X Forwarding enabled (if using a Windows System Server, this can be done using Xming with Putty as an SSH client). It is also possible to display OpenFlow messages passing between the switch and controller, since the Mininet VM comes with the WireShark application pre-installed, and a custom version of the OpenFlow dissector. This can be invoked with the command:

[1] $ sudo wireshark

It should be noted that starting Wireshark with root privileges is a security risk, although for research environments the risk is likely minimal. We can create a display filter for OpenFlow messages by entering the text of in the Filter window of WireShark and selecting *Apply*.

3. Virtual Networks in Mininet

We accessed Mininet's directory and created a text file in Python for our network, as illustrated in Figure 3.

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's1' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( rightHost, leftSwitch )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Fig. 3: Text file for virtual network written in Python.

In this case, components are added using the script:

```
leftHost = self.addHost( ‘h1’ )
leftSwitch = self.addSwitch( ‘s1’ )
```

And connected to each other by a link using the script:

```
self.addLink( leftHost, leftSwitch )
```

This text file is then saved and runs using the command:

$ sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --topo my topo --test pingall

To demonstrate the scalability of this approach, we modified the Mininet network to create 64 hosts and 9 switches [6, 7]. As expected, the time to ping resources increases as the network grows larger; results from our network are shown in Table 1.

Table 1. Ping Test Performance.

Hosts	**Switches**	**Links**	**Ping Test (seconds)**
2	1	2	5.244
3	2	4	5.45
10	3	30	19.515
64	9	72	87.30

4. Open Daylight Configuration

The ping response times between the switch and controller are significant because each OpenFlow-enabled switch needs to connect with a centralized SDN controller on a regular basis (the controller holds the routing tables, for example, so whenever a packet with a new IP address arrives at the ingress of an OpenFlow switch the switch must ask the controller how to proceed). We can also create an Open Daylight instance connected to the Mininet network.

Open Daylight can be installed in a second VM, and it’s convenient to connect both VMs to a host-only network so they can communicate with each other and with SSH. To build the OpenDaylight version 1.3 virtual machine, we can download and install a server ISO disk image [6]. The minimum configuration to support Open Daylight should include about 2 GB RAM and 2 CPUs. Each VM should have two network adapters. By default, the first VM network adapter is attached to the VirtualBox NAT interface and is already configured when the VM boots. The second network adapter needs to be configured to a host-only interface as noted previously. The VirtualBox DHCP server will assign an IP address to the host-only network; we can edit the “network interfaces” file to insure that devices will remain connected after the VM restarts. We can either SSH into the Open Daylight VM using the VirtualBox interface or by using a separate terminal emulator.

Since Open Daylight is a Java program, we need to install and configure the Java runtime environment, using the commands

$ sudo apt-get update
$ sudo apt-get install default-jre-headless

Instructions on how to set the Java_Home environment variable are provided elsewhere [4, 6, 7]. The Open Daylight software can then be installed from the open source ODL website *(www.openflow.org*). On a Linux host by using the *wget* command to obtain the .tar file:

$ wget https://nexus.opendaylight.org/content/groups/public/org/opendaylight/integration/distribution-karaf/0.4.0-Beryllium/distribution-karaf-0.4.0-Beryllium.tar.gz

Extracting the .tar file creates a folder named distribution-karaf-0.4.0-Beryllium which contains the OpenDaylight software and plugins. OpenDaylight is packaged in a Karaf container that includes all the software and optional plugins in a single distribution folder. To run OpenDaylight, run the karaf command inside the package distribution folder:

$ cd distribution-karaf-0.4.0-Beryllium
$ ./bin/karaf

[2] Once Open Daylight is running, the following command lists all the optional features:

$ opendaylight-user@root> feature:list

These features may include the following:

dl-restconf:	Allows access to RESTCONF API
dl-l2switch-switch:	Provides network functionality similar to an Ethernet switch
dl-mdsal-apidocs:	Allows access to Yang API (not implemented in our testing)
dl-dlux-all:	OpenDaylight graphical user interface

The Open Daylight GUI can be accessed from a browser on the host system by entering the URL of the OpenDaylight User Interface (DLUX UI), which is running on the Open Daylight VM (and thus has the same IP address as this VM) under default port 8181. The default username and password are both *admin*. The GUI will display the topology of the Mininet network, and can be used to request detailed information about the nodes and host connections.

As an example, on a Mininet VM, we created a network topology using 4 switches in a linear topology, each connected to one host, with the Open Daylight controller VM having an IPIP address of 192.168.50.1 on port 6033 (we can auto-configure the MAC address for each attached host). This is done using the following Mininet command:

$ sudo mn --topo linear,4 --mac --controller=remote,ip=192.168.50.1,port=6033 --switch ovs,protocols=OpenFlow13

We can then test that the OpenDaylight network is working by pinging all nodes and verifying that every host is reachable.

5. Conclusions

The growing popularity of SDN within cloud computing environments and related applications has led to an interest in modeling the behavior of SDN network configurations. We have demonstrated the use of Mininet to simulate an SDN network using an Open Daylight controller. We discussed how to install Mininet and the controller in two virtual machines, provided some common useful commands, and discussed how to capture OpenFlow message exchanges on the network. Future work may include implementing the Yang data modeling structures and REST API in this environment.

References

[1] C. DeCusatis, "Reference Architecture for Multi-Layer Software Defined Optical Data Center Networks," *Electronics*, vol. 4, no. 3, pp. 633-650, 2015.

[2] K. Barabesh, R. Cohen, D. Hadas, V. Jain, R. Recio, and B. Rochwerger, "A case for overlays in DCN virtualization," in *Proc. 2011 IEEE DC CAVES workshop, collocated with the 22nd International Tele-traffic Congress (ITC 22).*

[3] J. Manville, "The power of a programmable cloud," in *OFC 2012 Annual Meeting,* Anaheim, CA, 2013.

[4] F. Keti and S. Askar. "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments," in *6th International Conference on Intelligent Systems, Modelling and Simulation*, 2015.

[5] SDN Hub. (2016, April 20). Useful Mininet setup. [Online]. Available: http://sdnhub.org/resources/useful-mininet-setups/

[6] Linkletter, (2016, April 20). Using the Open Daylight SDN controller with the Mininet network emulator [Online].

Available: http://www.brianlinkletter.com/using-the-opendaylight-sdn-controller-with-the-mininet-network-emulator/#more-4419
[7] M. Casado, N. Foster, and A. Guha. “Abstractions for Software-Defined Networks,” vol. 57, pp. 86-95.
[8] Communications of the ACM. (2014). See also The OpenFlow switch. [Online]. Available: http://www.openflowswitch.org.