Layers Matching of Open Flow Entries in Open vSwitch

Zin May Aye
Professor, Faculty of Computer Hardware and Technologies,
University of Computer Studies, Yangon
zinmayaye@ucsy.edu.mm

Abstract

This paper proposed the usages of Mininet in various adoptions and methods. The emulator is used in sdn related controllers, switches and hosts in implementing the instant virtual network. Usage of Mininet adoption in varies with different purposes. The researchers apply it for SDN projects and testbed, and the students and teachers adopt it for labs and projects and IT services apply it for many sdn applications. By using Mininet, user can deploy host, switch and controllers run on VM (virtual machine). The virtual mode strategy of Mininet has been conducted for the purpose of prototyping and emulating for the network traffic analysis. In this paper Layer 2,3 and 4 flow entries are matched in the OVS flow entries and tested with QoS applied in the modification of the packets. The wireshark is used for the analyzing of the packet traffic in the ethernet port of the OVS switch.
Keywords : *OVS,SDN,Mininet*

1. Introduction

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. It emulates the network hosts links, and switches on a single machine. Mininet uses process-based virtualization to run many hosts and switches on a single OS kernel and also provides an easy way to get correct system *behavior* and also support miniedit.py for the GUI interface to experiment with topologies. In this paper, all of the network paradigms are tested on the command line interface mode. The first is a simple topology for testing and check the flows of traffic between the devices. Priority traffic flows are run and tested for the flow actions. Python based network programming is also essential in creating topology design in mininet and a handy Python API for creating varying sizes of networks and topologies can be applied in the emulator.

2. Related Works

Bob Lantz, Brian O'Connor [1] explained that Mininet already has the capability of creating containers as well as the network from a single, simple Python API. In his paper the author explained the applied of Mininet and the voids the need for installing, configuring of the required systems. To support configurations that exceed the resources of a single server, The experimental cluster of Mininet support may easily be used to easily and distribute the virtual testbed across multiple physical servers. In the paper of [1], the author evaluated on the tool of SDN in emulation process and explained the Mininet's performance and the evaluation . In that paper was specified the tests were conducted to study on the emulator limitations that are related to the environment of the emulation, the capabilities of the resources. The author also evaluated, the Mininet scalability on different topologies with applied varying number of nodes and scenarios. The results of that paper described that the simulation environment has a remarkable effect on the required time for building a topology.

3. The Basic of OVS and mininet

Open vSwitch, abbreviated as OVS, is an open-source implementation and the main purpose of is to provide a switching stack for hardware virtualization environments, and providing protocols and standards used in computer networks. It is licensed under the open source Apache 2 license. It is well-suited for an open-source project that allows hypervisors to virtualize the networking layer.

Mininet is a kind of a *network emulator that suppor in designing and creating* virtual hosts, switches, controllers, and links. The host on the Mininet run on the Linux software and its switch supports OpenFlow for flexible traditional switching processes. 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.[3]

Mininet command lines use various commands for implementing the desired network. The most of the applied commands need to create the virtual testing network topology used a command-line launcher (mn) to instantiate networks as in the followings:

```
mn --topo single,4
mn --topo tree,depth=2,fanout=3
```

3.1 Creating Simple Topology in CLI

In OVS it is need to learn and understand the flow entries on OVS, ovs-ofctl command, flow entries on an OpenFlow capable switch control behavior of packets, typically installed dynamically with an SDN controller

Figure 1 shows the simple network topology for the switch and host connections. The switch is the open flow switch. The Open flow is installed in Mininet. The Switch is Open virtual Switch (OVS) and need to add the flows manual from network admin to access the connection with the hosts. Many hosts can applied up to (4096) in Mininet based on the version. By applying the command under the root with *minineditt.py*, the required network infrastructure of virtual switch and three hosts topology can be applied and e designed as shown in figure 1.

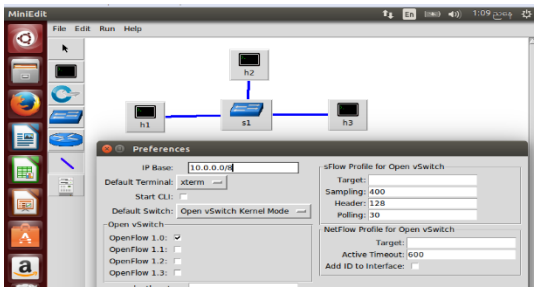

Figure 1. An OVS and hosts connection

In command line interface, it is needed to write for the formation of that topology.

$ sudo mn –topo=single,3 –controller= none, --mac

After running the topology the network can be seen by applying net command, and it is shown in figure 2 and link output in figure 3. The dump command will show the process id and the ip address of the nodes in the topology.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0
mininet>
```

Figure 2. Network nodes for the topology

```
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
h3-eth0<->s1-eth3 (OK OK)
```

Figure 3. Network links in the topology

3.2. Openflow Vswitch and Flow Entries

To apply the flows in openflow it needs to apply ovs-ofctl command to manually add flow entries. Flow entries on an Openflow capable switch control the behavior of packets[4]. These flows can be deleted as required. Without the flow entries and adding the required flows, the host and switch can never be connected and the host will be failure in ping test for the network reachability. Flows can be added and deleted by applying the following commands :

$sh ovs-ofctl add-flow s1 action=normal

$sh ovs-ofctl del-flows s1

Apply pingall can see the host can respond icmp echo in the connection. While deleting the flow entry all the host connections cutoff and no reachability to each other as in figure 4.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
```

Figure 4. Test reachabilty between hosts

3.3. Layer 2 matching

Layer 2 matching is based on the mac address of the hosts in the virtual network. It can be called the /MAC Based Flow Entry matching. The flow entries including of mac addresses are added and input/output port is specified. This can be defined as Layer 2 matching in openFlow switch. By adding two flows for the purpose of bidirection communication, the switch can know the packet to transfer from which port that it enters. In the mac-based flow, openflow virtual switch accepts the flow with mac address for the host with both source and the destination and apply mac matching in flow entries for the connections. The first and foremost is remove and delete the flows added in manually in OpenVswitch. The bi-directional flow is provided for the specified source and destination mac addresses. In the mininet prompt use the command [sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01, dl_dst= 00:00 :00 : 00 :00:02,actions=output:2] and [sh ovs-ofctl add-flow s1 dl_src=00:00:00: 00:00: 02,dl_dst=00: 00 : 00:00:00:01,actions=output:1]. Ths flow rule is not assisting of L2 broadcasting in the OpenVSwitch. The broadcast mechanism is added for the flooding and allowing ARP request for layer 2 matching. In the topology with protocol number of 1 is added to provide icmp and applying the command [sh ovs-ofctl add-flow s1dl_type= 0x806,nw_ proto=1,actions=flood]. This action means outputting packets on all physical ports other than the port on which it was received and any ports on which flooding is disabled . the dl_type of 0x806 refers to arp protocol and packets must match the lower 8 bits of the ARP opcodes.The *flood* action class which sends packets to all existing ports except the ingress port All hosts can have ping access with layer2 matching in the network topology. Figure 6 shows the adding the flow entry based on the source and destination mac addresses.

```
mininet>
mininet>
mininet> sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,actions=o
utput:2
mininet> sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,actions=o
utput:1
mininet> sh ovs-ofctl add-flow s1 dl_type=0x806,nw_proto=1,actions=flood
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 X
h3 ->
```

Figure 5. The commands for layer2 mac address entries

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=363.677s, table=0, n_packets=3, n_bytes=238, idle_age=63, d
l_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=322.119s, table=0, n_packets=4, n_bytes=280, idle_age=63, d
l_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=88.952s, table=0, n_packets=13, n_bytes=546, idle_age=48, a
rp,arp_op=1 actions=FLOOD
```

Figure 6. Flow entry with mac address

3.4. Layer 3 matching

It is the critical in use of fields in OpenFlow to determine whether particular field values agree with a

set of constraints called a match. At layer 3, the IPv4 or IPv6, source and destination. Layer 4 matching use port (tcp/udp) in the fields. Other data fields are computed Ethernet contains an Ethertype and IPv4 contains IP protocol type. Some types of matching consist of exact match, that specified exact nw_src and nw_dst ip addresses. The other is wildcard and bitwise matches. Bitwise match, as in nw_src= nw/nw_mask can be applied. In Layer3 matching, it is required dl_type that represents for ether type of ipv4 and if it is not included, the matching of wildcard for network source and destination will be ignored in the openflow. The keyword (ip) can be also used in for dl_type. For the action, it is for broadcasting exclude ingress port. The ether type of 0x808 is used in the flow. The configuration format is as follows:

ovs-ofctl add-flow <bridge> dl_type=<ethernet type>,nw_src=ip[/netmask],actions=<action>

ovs-ofctl add-flow <bridge> dl_type=<ethernet type>,nw_dst=ip[/netmask],actions=<action>

3.5 Priority field in the Flows in Layer3 Matching

In the flow of the packet the sensible behavior is sometimes expected as more specific flows taking precedence over less specific flows. A higher value will match before a lower one. For mod-flows without --strict, priority is only significant if the command creates a new flow, that is, non-strict mod-flows does not match on priority and will not change the priority of existing flows. Other commands do not allow priority to be specified. priority=value. The priority at which a wildcarded entry will match in comparison to others. value is a number between 0 and 65535, inclusive. An exact-match entry will always have priority value of 65535. When adding a flow, if the field is not specified, the flow's priority will default to 32768. In the following two flow rules, one is 500 and the other is 800.

Testing of Host3 to Host1 is success and the flow of in port 1 to output 2 is denied. The command of [sh-ovs-ofctl add-flow s1 priority=500, dl_type=0x800, nw_src= 10.0.0.0/24 ,nw_dst= 10.0.0.0./24,action=normal] and [sh-ovs-ofctl add-flow s1 priority=800,ip,nw_src=10.0.0.3, actions =normal] are applied for two different priorities. This action is in regard to using single-flow table only. The analyzing of these flows in OVS with command dump-flows is show in figure 9.

When two flow entries with wildcards are matched, the entry with the higher priority will match before the lower one. By setting priority 32768 that is default, and the action is drop, the matching of priority is only one with the default value and the packet will drop. There is no communication access between hosts and dump-flow shows the new action drop rule entry. H1 and H2 cannot have access and the ping results are failed as in figure 7.

The dump flow result shows the drop action for the flow rules in figure 13. The value of priority is between 0 and 65535. Priority is the critical role in flow entry as the high priority is only applied in action and other lower priority flows are ignored just like in router's ACL rules. The default priority is 32768. Add drop rule in S1 for the priority value greater than the default and select hard timeout the flow will be deleted and connectivity is resume. First all the flows are checked for successful connectivity with mac-based flow entries. The flow of priority value greater than default is added and then analyze the connection. As in a rule like [sh ovs-ofctl add-flow s1 priority =40000, hard_ timeout=30, actions=drop].

Within the hard_timeout, host has stopped the reachability and then resume after hard_timeout value.

```
mininet> h1 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 999ms
pipe 2
```

Figure 7. Ping failed due to priority action

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=745.868s, table=0, n_packets=4, n_bytes=280, idle_age=516,
 priority=500,in_port=1 actions=output:2
 cookie=0x0, duration=529.127s, table=0, n_packets=4, n_bytes=280, idle_age=516,
 priority=500,in_port=2 actions=output:1
 cookie=0x0, duration=184.161s, table=0, n_packets=8, n_bytes=448, idle_age=45,
actions=drop
```

Figure 8. Dump flows in flow table

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=393.354s, table=0, n_packets=4, n_bytes=280, idle_age=163,
 priority=500,in_port=1 actions=output:2
 cookie=0x0, duration=176.613s, table=0, n_packets=4, n_bytes=280, idle_age=163,
 priority=500,in_port=2 actions=output:1
```

Figure 9. Dump-flows of layer3 flows in OVS

3.6. QoS Marking in flow

In the consideration of quality of sevice affects in flow entries, the flows are tested with the manually added priority values for each of the unidirectional flow and added the action for the packet in these flows. In this system, Differentiated Services (DiffServ) is treated by intermediate systems with relative priorities based on the type of services (ToS) field. It is inserted in the flow to test the Layer 2 fram applied the dscp code in the traffic. DS field of EF PHB is applied to test the open flow traffic and capture with wireshark as show in figure 9. Code point 0x2e is work with EF forwarding in openflow virtual switch and the flow is actioned with the priority value of the flow table entry as shown in Figure 10. The traffic flow of dscp is analyzed in the wireshark tool. In the flow entry of the dscp code 46 turn into 8bits and get the value of 184. The aim is not to pass traffic with QoS of dscp value 46 to other hosts. The adding of QoS flow and the host connectivity flows to the flow table of virtual swich is as follows and also in the figure 10.

[Sh-ovs-ofctl add-flow s1 priority=800,ip,nw_src=10.0.0.3,actions=mod_nw_tos:184,normal .]

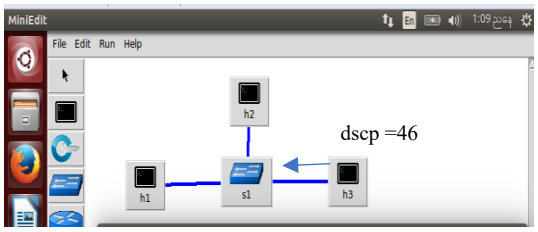

Figure 9. QoS marking for the OVS flow

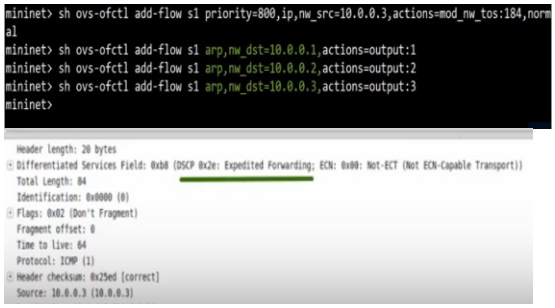

Figure 10. Dump flows of QoS flow in table entry and wireshark capture code

3.7. Verifying ARP Protocol

The flows of communication between three hosts are manually added and analyze the ARP protocol messages in the transactions.

The flow rules are added as follows:

Mininet>sh ovs-ofctl add-flow s1
arp,nw_dst=10.0.0.1,actions=output:1
Mininet>sh ovs-ofctl add-flow s1
arp,nw_dst=10.0.0.2,actions=output:2
Mininet>sh ovs-ofctl add-flow s1
arp,nw_dst=10.0.0.3,actions=output:3

The command *pingall* and the *dump-flows* output show the required output for ARP between the hosts and switch. Figure 10 shows the ping test between hosts and figure 11 shows the dump-flows with arp responses.

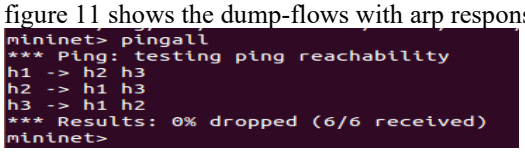

Figure 10. Ping test between hosts

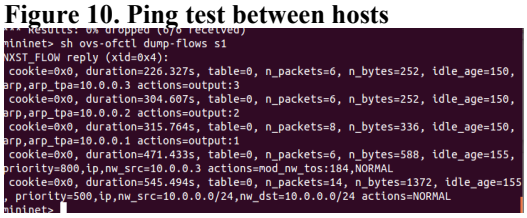

Figure 11. Arp dump-flows in flows monitoring

3.8 Applied Wireshark in Mininet

Wireshark is applied as the network protocol analyzer in the networking fiels. By applying wireshark, the use can analyze and see what's happening in the network at details and , it also has the features of deep inspection of hundreds of protocols, Live capture and offline analysis and so on. In Mininet with the virtual network topology, wireshark can be applied and analyze the traffic in details. The followings are applied in Mininet to view the wireshark protocol analyzer in virtual network.

mininet> h1 wireshark &
mininet> h2 wireshark &
mininet> sh wireshark &

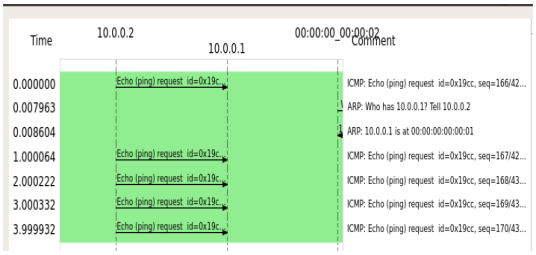

Figure 12. ARP traffic in wireshark

3.9. Layer 4 Matching

For the layer 4 matching, it is needed to apply application protocol in the flow. The design for http server is shown in figure 13.

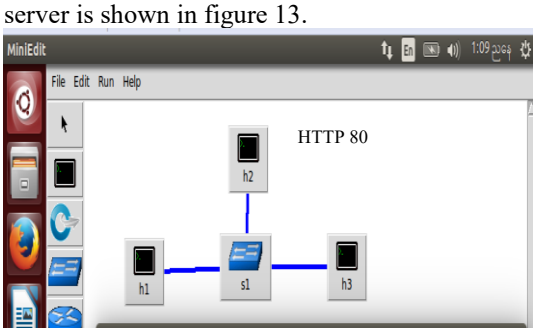

Figure 13. HTTP server in Host 3 for layer 4 matching

The process is tested with H3 as the http server rand the port 80 is allowed in the flow entry to access from the other hosts. The curl command figure 15 is applied to access the web sever in H3 and match tp_src or tp_dst and need to specify dl_type and nw_proto in the flow. The port of tcp UDP is specified as required for the application protocol. Layers 4 matching is start with assigning Host 3 to Http server and the flow entry is configured for other host with application layer protocol of tcp port 80 and output ports are assigned to port 3 of OVS. If the http ports are matched in the traffic flows, the h1 host can access to webserver as shown in figure 14. One can also apply xterm for host H3 to apply http sever as in figure 14 and h1 and h2 clients access to server as in figure 15. The flows of http traffic with tcp sequences are shown in figure 16.

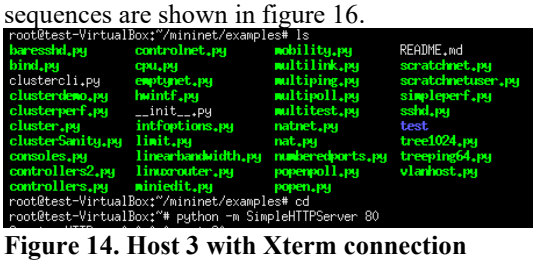

Figure 14. Host 3 with Xterm connection

The command for layer4 matching in the hosts in mininet are as follows:

mininet> h3 python -m SimpleHTTPServer 80 &
mininet> sh ovs-ofctl add-flow s1 arp, action=normal

mininet> sh ovs-ofctl add-flow s1,dl_type=0x800,nw_proto=6,tp_dst=80,actions=output:3
mininet> sh ovs-ofctl add-flow s1,ip,nw_src =10.0.0.3, actions=normal

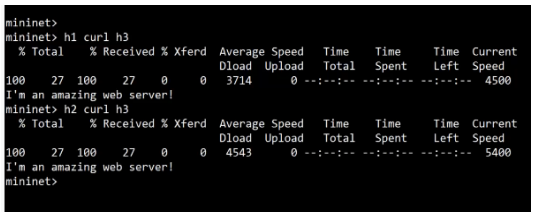

Figure 15. Use curl command to access http server

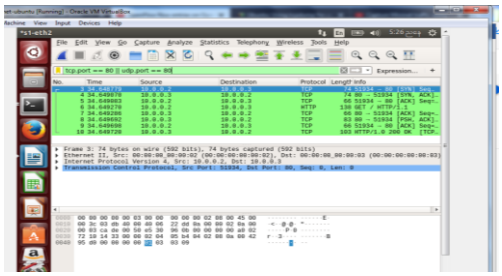

Figure 16. Wireshark traffic with http

4. Results and Discussions

For the OpenFlow OVS, the flows are applied, and the corresponding command lines are processed for the layer matching. In the layer matching Layer 2 matching applied mac addresses for the source and destinations hosts and analyze the flow output in Mininet. For the Layer 3, the ip addresses are used in the flow entries and the protocol are inserted in generating the flow rules. The priority values are also used for the action rule to work properly or not in the flow entries. In the Layer 4 matching, the http server in one of the host is accesses from the other host with the respective flow entries that specified application port in entries. The layers matchings are also analyzed by applying wireshark tool for monitoring traffics. For further analysis, the controller is added and the applied rules can be analyzed with multiple flow tables. This paper is based on the single flow table of the OVS switch and its open flow entries with Mininet.

5. Conclusion

For the testing with OpenFlow switch, the VirtualBox graphical interface 6.1.6 and the Guest OS of ubuntu 16.04 are applied. The Mininet version of 2.2.170321 is installed with the openflow controller package for the testing OVS switch application. In this paper the simple virtual network topology without the controller is applied. The testing of the network is emphasized on the layer application command and the parameter usages performance of mininet tool. The network traffic is captured in wireshark. The mininet is a great and convenient tool and has scalability to be studied. Mininet runs on a single system and resources need to be shared among virtual hosts and switches.[5] OVS switch only is applied in the testing and flow rules entries for the switch are specified and analyzed in the system. For further application, the remote controller, POX or Ryu controllers and others can be applied to the network to control the flows and the traffic can be analyzed in the environment and applied with the OVS applications.

Acknowledgements

I wish to express my sincere appreciation to my Rector to do the and guide for the research paper. The physical and technical contribution of the referenced paper and the authors to support the paper to be considered and tested. Without the persistent help and guidelines the goal of this paper would not have been realized."

References

[1]. Bob Lantz, Brian O'Connor, A Mininet-based Virtual Testbed for Distributed SDN Development, August 18-20, 2015, London, UK.
[2]. Faris Keti, Shavan Askar , Electrical and Computer Engineering Department, Environments Emulation of Software Defined Networks Using Mininet in Different Simulation, 2015, 6th International Conference on Intelligent Systems, Modelling and Simulation
[3]. The Openflow Switch, openflowswitch.org.
[4]. Mininet Project. Mininet. http://mininet.org/.
[5].Mininet.An Instant Virtual Network on your Laptop.2014, Sept. 2014, http://mininet.org.