Protocal Implementation in Mininet

Yulai Bi 5110309514
School of Electronic Engineering,
Shanghai Jiao Tong University

June 21, 2014

Abstract

Mininet provides a simple and inexpensive network testbed for developing SDN network, which decouples the network control and forwarding functions, enabling the network control to become directly programmable for applications and network services. In this article, I provide a set of basic operation of Mininet, some algorithm of the controller and the verification of the behavior of the controller.

1 Introduction

1.1 SDN

Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications.

This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services.

The SDN architecture is:

- Directly programmable: Network control is directly programmable because it is decoupled from forwarding functions.
- Agile: Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs.
- Centrally managed: Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.
- Programmatically configured: SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.
  Open standards-based and vendor-neutral: When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.

1.2 OpenFlow

OpenFlow is an open interface for remotely controlling the forwarding tables in network switches, routers, and access points. Upon this low-level primitive, researchers can build networks with new high-level properties. It is a communication protocol that gives access to the forwarding plane of a network switch or router over the network. The OpenFlow protocol is a foundational element for building SDN solutions.

The working procedure of SDN network based on OpenFlow is shown in Figure1. As we can see in Figure1, The application layer, control layer and infrastructure layer is separated, making it easily to controller the action of the network.

1.3 Mininet

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. It provides a simple and inexpensive network testbed for developing OpenFlow applications. Mininet networks run real code including standard Linux network applications as well as the real Linux kernel and network stack. Because of this, the code you develop and test on Mininet, for an OpenFlow controller, modified switch, or host, can move to a real system with minimal changes, for real-world testing, performance evaluation, and deployment. Importantly this means that a design that works in Mininet can usually move directly to hardware switches for line-rate packet forwarding.

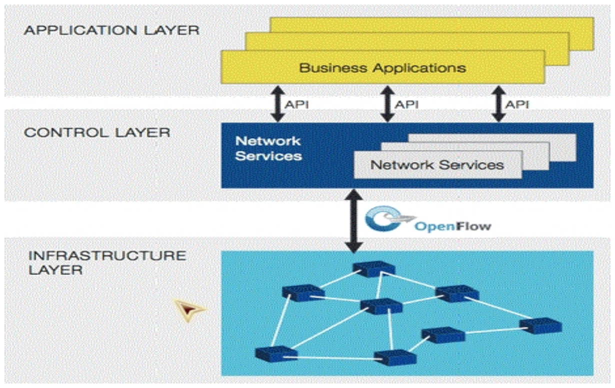


Figure 1: The Working Procedure of SDN Network Based on OpenFlow

1.4 Beacon

Beacon is a java-based SDN controller platform for research and education.

2 Basic operation of Mininet

2.1 Start Network

To create a 1switch-3hosts network like Figure2 in the VM, we need to enter the following command in the terminal.

```
terminal> sudo mn –topo single,3 –mac –switch ovsk –controller remote
```

In which, the *topo single,3* means setting a network with 1 switch and 3 hosts, the *mac* means setting the switch MAC host MAC and IP address to small, unique and easy to read IDs, the *switch ovsk* means setting a switch with a state of Open Vswitch, the *controller remote* means setting a remote controller which could be in the VM, outside the VM and on your local machine, or anywhere in the world.

But the 1switch-3hosts network shown in Figure2 cant satisfy us if we are going to extend the controller from a single-switch network to a multiple-switch multiple-host network.

Fortunately, we can set up the topology of our own. Custom topologies can be easily defined as well, using a simple Python API, part of the code is provided below. The code is easy to understand, first we add three hosts and two switches to the network, and then, connects two switches directly, connect two hosts to switch1, and connect the other host to switch2. The related topology is shown in Figure3.

```
# Add hosts and switches
left1Host = self.addHost( 'h1' )
left2Host = self.addHost( 'h2' )
rightHost = self.addHost( 'h3' )
leftSwitch = self.addSwitch( 's1' )
rightSwitch = self.addSwitch( 's2' )
# Add links
self.addLink( left1Host, leftSwitch )
self.addLink( left2Host, leftSwitch )
self.addLink( leftSwitch, rightSwitch )
self.addLink( rightSwitch, rightHost )
```

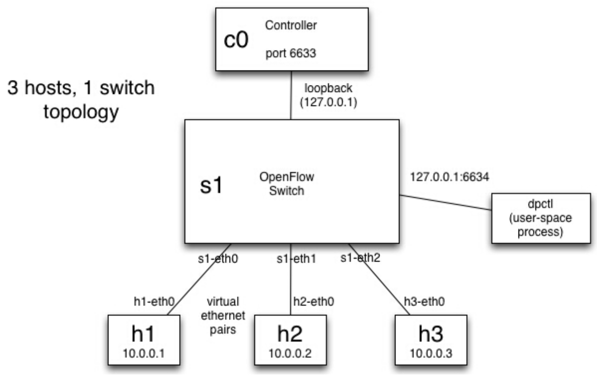


Figure 2: A 1switch-3hosts Topology

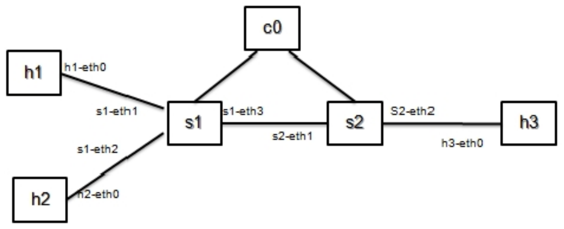


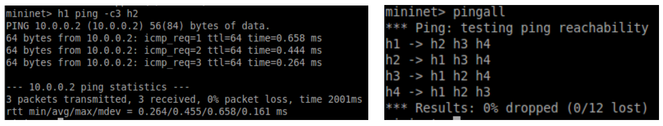
Figure 3: A 2switches-3hosts Topology

2.2 Test Connectivity

Ping test is used to test the connectivity of the network in doing mininet experiment. The pingall test entered in the mininet console is used to test the connection between all the hosts. The other one is used to test the connection between two specific hosts. The response of the two ping test is shown in Figure4.

mininet> pingall

mininet> h1 ping c3 h2



```
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=0.658 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.444 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.264 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.264/0.455/0.658/0.161 ms
```

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (0/12 lost)
```

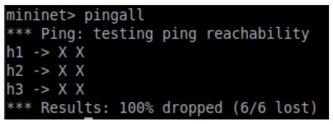
Figure 4: The Response to Ping Test

3 Remote Controller

The core of the research is using a remote controller, if we simply start a network like Figure2 using a remote controller and test the connectivity of the hosts, we just get the result shown in Figure5 that the hosts cant connect with each other.

Since, there is no controller connected to the switch and therefore the switch doesn't know what to do with incoming traffic, leading to ping failure. There are a handful of controllers available: such as POX, NOX, Beacon or Floodlight and I choose to use Beacon mentioned before as my remote controller.

In the following sections, I will provide some algorithm of the controller and the verification of



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

Figure 5: Ping Failure Without Remote Controller

the behavior of them.

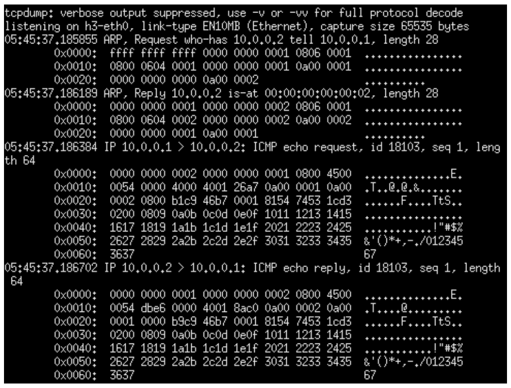
3.1 Hub

A network hub, which works at physical layer, is an unsophisticated device in comparison with, for example, a switch. A hub does not examine or manage any of the traffic that comes through it, which means that any packet entering any port is rebroadcast on all other ports. Part of the code is provided below, where *OFPP_FLOOD* means output all openflow ports except the input port.

OFAction action = new OFActionOutput(OFPort.OFPP_FLOOD.getValue());

Then we can verify hub behavior with tcpdump, which is a utility to print the packets seen by a host. To do this, we'll create xterms for each host and view the traffic in each. In the Mininet console, start up four xterms, and run tcpdump so that the packets can be seen by the host. Then, ping h2 from h1, we can see from the Figure6 the identical ARP and ICMP packets corresponding to the ping appear in all of the four xterms running tcpdump. This is how a hub works; it sends all packets to every port on the network.

```
mininet > xterm h1 h2 h3
h1 > tcpdump -XX -n -i h1-eth0
h2 > tcpdump -XX -n -i h2-eth0
h3 > tcpdump -XX -n -i h3-eth0
mininet > h1 ping -c1 h2
```



```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
05:45:37.185855 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
05:45:37.186189 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0001 0a00 0001                 ..........
05:45:37.186384 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 18103, seq 1, leng
th 64
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 0000 4000 4001 26a7 0a00 0001 0a00  .T..@.@.&.......
        0x0020:  0002 0800 b1c9 46b7 0001 8154 7453 1cd3  ......F....TtS..
        0x0030:  0200 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
05:45:37.186702 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 18103, seq 1, length
 64
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 dbe6 0000 4001 8ac0 0a00 0002 0a00  .T....@.........
        0x0020:  0001 0000 b9c9 46b7 0001 8154 7453 1cd3  ......F....TtS..
        0x0030:  0200 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
```

Figure 6: The Verification of the Behavior of a Hub

3.2 L2 learning Switch

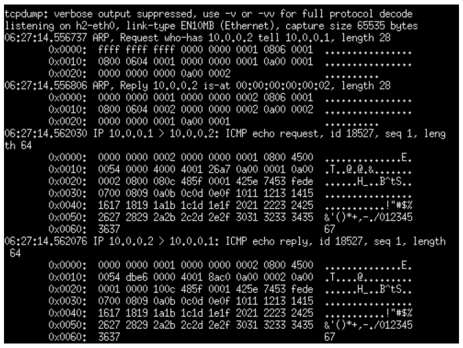
When we see a packet, we'd like to output it on a port which will eventually lead to the destination. To accomplish this, I build a table that maps addresses to ports. The table is populated by observing traffic. When we see a packet from some source coming from some port, then we know that source is out that port. When we want to forward traffic, we look up the destination in the table. If we know the port, we simply send the massage out. If we don't know the port, we just send the message out all ports except the one it came in on.

The algorithm of l2 learning switch is provided as follows, for each packet from the switch:

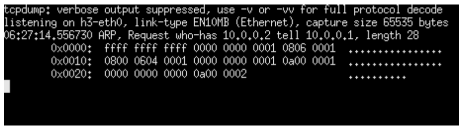
- Use source address and switch port to update address/port table.
- Drop the LLDP packet.
- If the destination is multicast, just flood the packet to all the ports except the one it came on.
- Drop the packet whose output port is the same as input port.
- If we know the port for destination address, just install flow table entry in the switch and send the packet out appropriate port.
- If the port for destination address is not in our address/port table, just act as a hub to flood the packet to all the ports except the one it came on.

Then we use the tcpdump method mentioned before to verify the behavior of the l2 learning switch. In the Mininet console, start up the xterms of h2 and h3, and then ping h2 from h1 for the first time. As we can see in Figure7, we can see that an ARP request appear in both xterm, which means that the switch dont know the Port for destination address and flood the ARP packet out. Then h2 replied and let the switch know where the port is and install flow table. After that, the switch know the port for destination and just send the subsequent packet out appropriate port. Then ping h2 from h1 for the second time, as is shown in Figure8, since the switch have already known the port of destination, no packet is shown in the xterm of h3.

mininet > xterm h2 h3
h2 > tcpdump -XX -n -i h2-eth0
h3 > tcpdump -XX -n -i h3-eth0
mininet > h1 ping -c1 h2

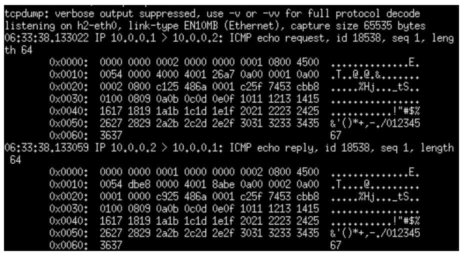


```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
06:27:14.556737 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
06:27:14.556806 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0001 0a00 0001                 ..........
06:27:14.562030 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 18527, seq 1, leng
th 64
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 0000 4000 4001 26a7 0a00 0001 0a00  .T..@.@.&.......
        0x0020:  0002 0800 080c 485f 0001 425e 7453 fede  ......H_..B^tS..
        0x0030:  0700 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
06:27:14.562076 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 18527, seq 1, length
 64
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 dbe6 0000 4001 8ac0 0a00 0002 0a00  .T....@.........
        0x0020:  0001 0000 100c 485f 0001 425e 7453 fede  ......H_..B^tS..
        0x0030:  0700 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
```

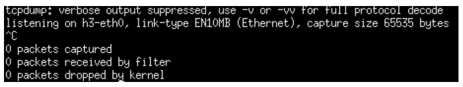


```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
06:27:14.556730 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
```

Figure 7: The Verification of the Behavior of a L2 Learning Switch



```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
06:33:38.133022 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 18538, seq 1, leng
th 64
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 0000 4000 4001 26a7 0a00 0001 0a00  .T..@.@.&.......
        0x0020:  0002 0800 c125 486a 0001 c25f 7453 cbb8  .....%Hj..._tS..
        0x0030:  0100 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
06:33:38.133059 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 18538, seq 1, length
 64
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 dbe8 0000 4001 8abe 0a00 0002 0a00  .T....@.........
        0x0020:  0001 0000 c925 486a 0001 c25f 7453 cbb8  .....%Hj..._tS..
        0x0030:  0100 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
```



```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
```

Figure 8: The Verification of the Behavior of a L2 Learning Switch

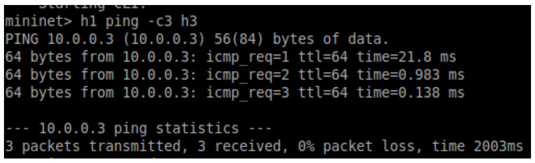
3.3 Multiple Learning Switch

The difference between single learning switch controller and multiple learning switch controller is that we need to create a address/port table for each switch rather than a single table for the whole network. The definition of the macTables, which is indexed by the switch, is provided below.

protected Map<IOFSwitch, Map<Long,Short>> macTables = new HashMap<IOFSwitch, Map<Long,Short>>();

I create a 2switches-3hosts topology shown in Figure3 to verify the behavior of multiple learning switch. Just ping h3 from h1, the result shown in Figure9 tell that the host1 is connected to host3, thus the controller works correctly.

terminal > sudo mn –custom mytopo.py –topo mytopo mac controller remote



```
mininet> h1 ping -c3 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=21.8 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.983 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=0.138 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
```

Figure 9: The Verification of the Behavior of a Multiple Learning Switch

4 Future Work

I have been learning some theory about multicast, and willing to implement some releted and complicated protocol using mininet.