Contents

1 Introduction **3**

1.1 OpenFlow . . . 3

1.2 Floodlight . . . 4

1.3 Multipath Routing . . . 5

2 Literature Survey **6**

2.1 Equal Cost Multipath routing . . . 6

2.2 Mininet . . . 6

2.3 Floodlight - Openflow Controller . . . 7

2.3.1 DeviceManagerImpl . . . 7

2.3.2 FloodlightProvider . . . 7

2.3.3 Forwarding . . . 7

2.3.4 LinkDiscoveryManager . . . 7

2.3.5 MemoryStorageSource . . . 8

2.3.6 TopologyService . . . 8

3 Problem Definition and Objectives **9**

3.1 Problem Definition . . . 9

3.2 Objectives . . . 9

4 Work-done **10**

4.1 Simulating OpenFlow network . . . 10

4.1.1 Installation . . . 10

4.1.2 OpenFlow network without controller . . . 10

4.1.3 OpenFlow network with Floodlight controller . . . 12

4.1.4 Floodlight - Shortest path algorithm . . . 15

5 Results and Analysis **16**

6 Conclusion and Futurework **18**

A Custom topology code - Shortest path **20**

List of Figures

1 Dedicated OpenFlow Switch 4
2 Architecture of Floodlight Controller 4
3 SSH into Mininet VM 10
4 Sample OpenFlow topology 11
5 Sample OpenFlow topology - simulation 12
7 Running Floodlight controller 12
6 Manually adding flow entries 13
8 Floodlight controller sending LLDP packets 13
9 Ping success in sample topology 14
10 Ping success in custom topology 14
11 Topology for Shortest path algorithm 15
12 packets captured by wireshark 16
13 Flow tables added by controller 17

1 Introduction

As IP networks are becoming larger and more complex, the operators of these networks gain more and more interest in traffic engineering. Traffic engineering encompasses performance evaluation and performance optimisation of operational IP networks. An important goal with traffic engineering is to use the available network resources more efficiently for different types of load patterns in order to provide a better and more reliable service to customers.

Current routing protocols in the Internet calculate the shortest path to a destination in some metric without knowing anything about the traffic demand, link reliability etc. Multipath routing aims to exploit the resources of the underlying physical network by providing multiple paths between source-destination pairs. Multipath routing has a potential to aggregate bandwidth on various paths, allowing a network to support data transfer rates higher than what is possible with any one path.

1.1 OpenFlow

OpenFlow [1] is an open switching protocol that is based on the concept of Software Defined Networks (SDN). The typical architecture of a generic switch comprises of a control panel and a forwarding plane. The control plane is where the decision to forward a set of packets along a certain port is taken while the forwarding plane does the forwarding of data based on the above decision. Traditionally, switches combine both the planes in the same device and this leads to non-scalable and closed switching solution. OpenFlow divides the control and forwarding plane into separate entities and running them on separate devices. A network administrator can partition traffic into production and research flows. In this way, researchers can try new protocols, security models etc. An OpenFlow Switch consists at least three parts:

1. A Flow Table: tells the switch how to process the flow.
2. A Secure Channel: connects switch and remote controller.
3. The OpenFlow Protocol: provides standards for controller to communicate with a switch.

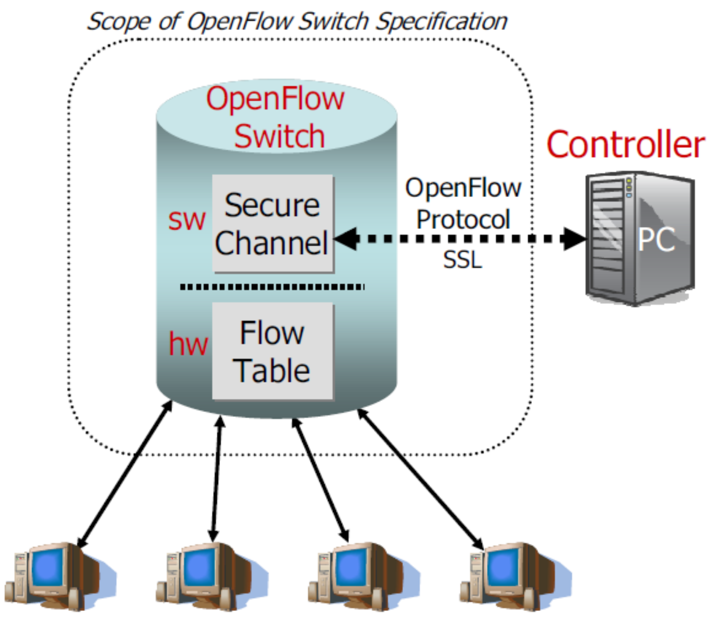


Figure 1: Dedicated OpenFlow Switch

1.2 Floodlight

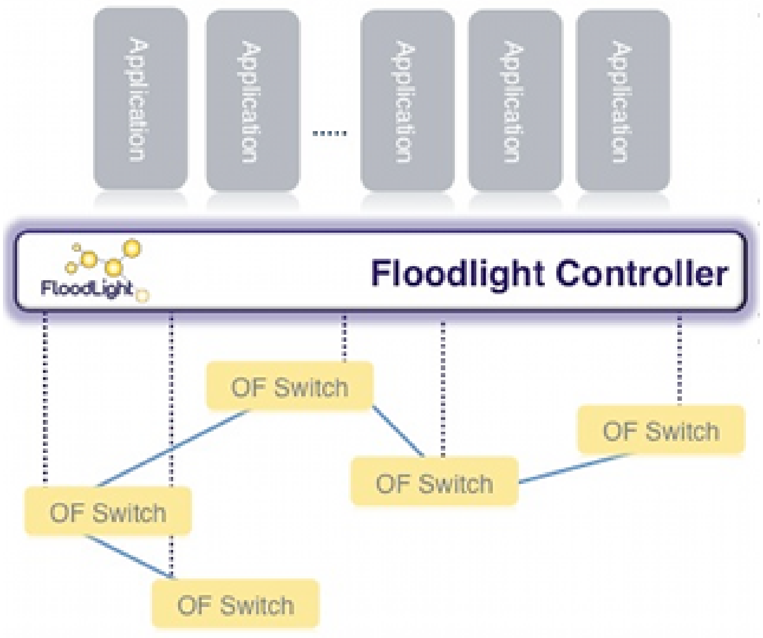


Figure 2: Architecture of Floodlight Controller

The Floodlight controller [2] is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller supported by community of developers.

- It offers a module system that helps developers to extend and enhance.
- It supports a broad range of virtual and physical OpenFlow switches
- It is easy to build and setup with minimal dependencies.

- It is designed to be high performance

1.3 Multipath Routing

Most routing protocol in current use such as RIP, EIGP and OSPF make very inefficient use of bandwidth usage. Multipath routing [3] is an answer for those defects. Two factors to be considered for efficient routing graph construction are loop freedom and connectivity [7].

2 Literature Survey

2.1 Equal Cost Multipath routing

Equal-cost multi-path (ECMP) [4] is a routing technique for routing packets along multiple paths of equal cost. The forwarding engine identifies paths by next-hop. When forwarding a packet the router must decide which next-hop (path) to use.

2.2 Mininet

Mininet [5] creates scalable software defined networks on a single PC by using Linux processes in network namespaces. It allows quickly creating, interacting with, customizing and a share software defined prototype, and provides a smooth path to migrating onto physical hardware.

- provides a simple and inexpensive network testbed for developing OpenFlow applications
- enables multiple concurrent developers to work independently on the same topology
- supports system-level regression tests, which are repeatable and easily packaged
- 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
- is usable out of the box without programming
- also provides a straightforward and extensible Python API for network creation and experimentation

How it Works

Nearly every operating system virtualizes computing resources using a process abstraction. Mininet uses process-based virtualization to run many (we've successfully booted up to 4096) hosts and switches on a single OS kernel. Since version 2.2.26, Linux has supported network namespaces, a lightweight virtualization feature that provides individual processes with separate network interfaces, routing tables, and ARP tables. The full Linux container architecture adds chroot() jails, process and user namespaces, and CPU and memory limits to provide full OS-level virtualization, but Mininet does not require these additional features. Mininet can create kernel or user-space OpenFlow switches, controllers to control the switches, and hosts to communicate over the simulated network. Mininet connects switches and hosts using virtual ethernet (veth) pairs. While Mininet currently depends on the Linux kernel, in the future it may support other operating systems with process-based virtualization, such Solaris containers or FreeBSD jails.

Mininet's code is almost entirely Python, except for a short C utility.

2.3 Floodlight - Openflow Controller

It [6] offers a module system that makes it simple to extend and enhance. Descriptions of modules are given below:

2.3.1 DeviceManagerImpl

DeviceManagerImpl tracks devices as they move around a network and defines the destination device for a new flow.

The Device Manager learns about devices through PacketIn requests. It takes information from the PacketIn and classifies the device according to how the entity classifier is setup. By default the entity classifier uses MAC address and VLAN to identify a device. These two properties will define what is unique as a device. The Device Manager will learn about other properties such as IP addresses as well. One important piece of information is the device attachment points. If a PacketIn is received on a switch an attachment point will be created for that device. A device can have as many as one attachment point per OpenFlow island, where an island is defined as a strongly connected set of OpenFlow switches talking to the same Floodlight controller. The Device Manager will also age out attachment points, IPs, and devices themselves. Last seen timestamps are used to keep control of the aging process.

2.3.2 FloodlightProvider

The FloodlightProvider provides two main pieces of functionality. It handles the connections to switches and turns OpenFlow messages into events that other modules can listen for. The second big function that it provides is decides the order in which specific OpenFlow messages (i.e. PacketIn, FlowRemoved, PortStatus, etc) are dispatched to the modules that listen for the messages. Modules can then decide to allow the processing of the message to go onto the next listener or to stop processing the message.

2.3.3 Forwarding

Forwarding will forward packets between two devices. The source and destination devices will be classified by the IDeviceService.

Since Floodlight is designed to work in networks that contain both OpenFlow and non-OpenFlow switches Forwarding has to take this into account. The algorithm will find all OpenFlow islands that have device attachment points for both the source and destination devices. FlowMods will then be installed along the shortest path for the flow. If a PacketIn is received is received on an island and there is no attachment point for the device on that island the packet will be flooded.

2.3.4 LinkDiscoveryManager

The link discovery service is responsible for discovering and maintaining the status of links in the OpenFlow network.

The link discovery services uses both LLDPs and broadcast packets (aka BDDPs) to detect links. The LLDP destination MAC is 01:80:c2:00:00:0e and the is ff:ff:ff:ff:ff:ff (broadcast address). The ether type for both LLDPs and BDDPs is LLDP (0x88cc). There are two assumptions made in order for the topology to be learned properly.

- Any switch (including OpenFlow switches) will consume a link-local packet (LLDP).
- Honors layer 2 broadcasts.

Links can either be "direct" or "broadcast". A direct link will be established if an LLDP is sent out one port and the same LLDP is received on another port. This implies that the ports are directly connected. A broadcast link is created if a BDDP is sent out a port and received on another. This implies that there is another layer 2 switch not under the control of the controller between these two ports.

2.3.5 MemoryStorageSource

The MemoryStorageSource is an in memory NoSQL style storage source. Notifications for changes in the database are also supported.

Other Floodlight modules that depend upon the IStorageSourceService interface can create/delete/modify data in the memory storage source. All data is shared and there is no enforcement. Modules can also register for changes to data in specific tables and rows. Any module that wants to do this should implement the IStorageSourceListener interface.

2.3.6 TopologyService

The Topology Service computes topologies based on link information it learns from the ILinkDiscoveryService. An important concept that the TopologyService keeps is the idea of an OpenFlow 'island'. An island is defined as a group of strongly connected OpenFlow switches under the same instance of Floodlight. Islands can be interconnected using non-OpenFlow switches on the same layer 2 domain.

All the information about the current topology is stored in an immutable data structure called the topology instance. If there is any change in the topology, a new instance is created and the topology changed notification message is called. If other modules want to listen for changes in topology they can implement the ITopologyListener interface.

3 Problem Definition and Objectives

3.1 Problem Definition

Currently the Floodlight Controller uses Shortest First Path algorithm to forward the packets which uses network resources inefficiently.Multipath routing can be effectively used for maximum utilization of network resources. It gives the node a choice of next hops for the same destination.

3.2 Objectives

The objective of this project is to add a module to the Floodlight Controller which does forwarding with efficient multipath routing algorithm.

Tasks:

1. Understand OpenFlow/SDN architecture.

2. Get to know about the tools (Mininet, Wireshark and Floodlight Controller) required for implementing the project.

3. Analysing the existing multipath routing protocols and coming up with optimized multipath routing algorithm for Floodlight controller.

4. Integrating the above algorithm to Floodlight controller and doing performance evaluation.

4 Work-done

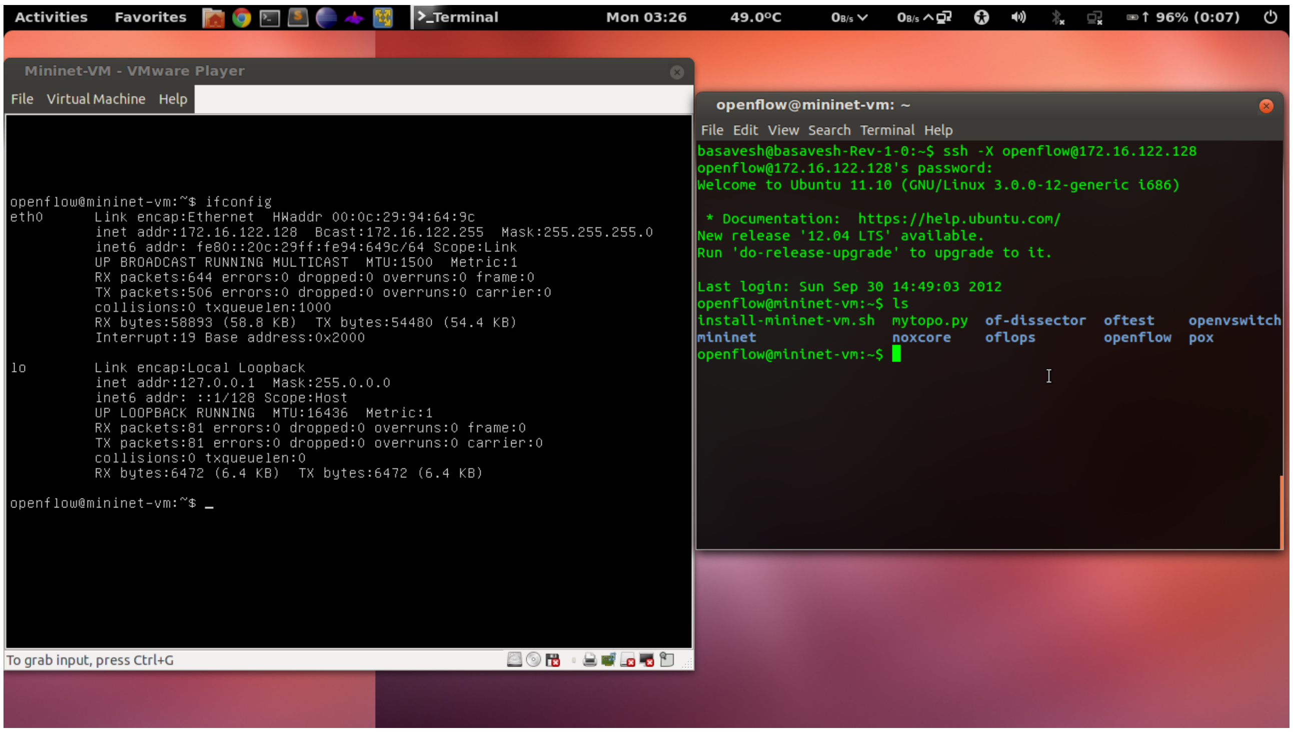
4.1 Simulating OpenFlow network

4.1.1 Installation

1. Mininet VM image - Download from `https://github.com/downloads/mininet/mininet/mininet-vm-ubuntu11.10-052312.vmware.zip`
2. Download and install virtualization system, VMware Player for linux.
3. Download Floodlight Controller from Floodlight Github.

4.1.2 OpenFlow network without controller

1. Run Mininet VM in VMware Player and login to VM.
2. SSH into VM - ssh -X openflow@openflow [mininet VM local ip address]



Activities
Favorites
Terminal
Mon 03:26
49.0°C
96% (0:07)
Mininet-VM - VMware Player
File Virtual Machine Help
To grab input, press Ctrl+G
openflow@mininet-vm: ~
File Edit View Search Terminal Help

```
openflow@mininet-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:94:64:9c
          inet addr:172.16.122.128  Bcast:172.16.122.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe94:649c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:644 errors:0 dropped:0 overruns:0 frame:0
          TX packets:506 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:58893 (58.8 KB)  TX bytes:54480 (54.4 KB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:81 errors:0 dropped:0 overruns:0 frame:0
          TX packets:81 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:6472 (6.4 KB)  TX bytes:6472 (6.4 KB)

openflow@mininet-vm:~$ _
```

```
basavesh@basavesh-Rev-1-0:~$ ssh -X openflow@172.16.122.128
openflow@172.16.122.128's password:
Welcome to Ubuntu 11.10 (GNU/Linux 3.0.0-12-generic i686)

 * Documentation:  https://help.ubuntu.com/
New release '12.04 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Sep 30 14:49:03 2012
openflow@mininet-vm:~$ ls
install-mininet-vm.sh  mytopo.py  of-dissector  oftest    openvswitch
mininet                noxcore    oflops        openflow  pox
openflow@mininet-vm:~$
```

Figure 3: SSH into Mininet VM

Short Descriptions

- **OpenFlow Controller:** sits above the OpenFlow interface. The OpenFlow reference distribution includes a controller that acts as an Ethernet learning switch in combination with an OpenFlow switch.
- **OpenFlow Switch:** sits below the OpenFlow interface. The OpenFlow reference distribution includes a user-space software switch. Open vSwitch is another software but kernel-based switch, while there is a number of hardware switches available from Broadcom (Stanford Indigo release), HP, NEC, and others.

- **dpctl:** command-line utility that sends quick OpenFlow messages, useful for viewing switch port and flow stats, plus manually inserting flow entries.
- **Wireshark:** general (non-OF-specific) graphical utility for viewing packets. The OpenFlow reference distribution includes a Wireshark dissector, which parses OpenFlow messages sent to the OpenFlow default port (6633) in a conveniently readable way.
- **iperf:** general command-line utility for testing the speed of a single TCP connection.
- **cbench:** utility for testing the flow setup rate of OpenFlow controllers.

3. Start a sample network. Figure given below depicts the topology.

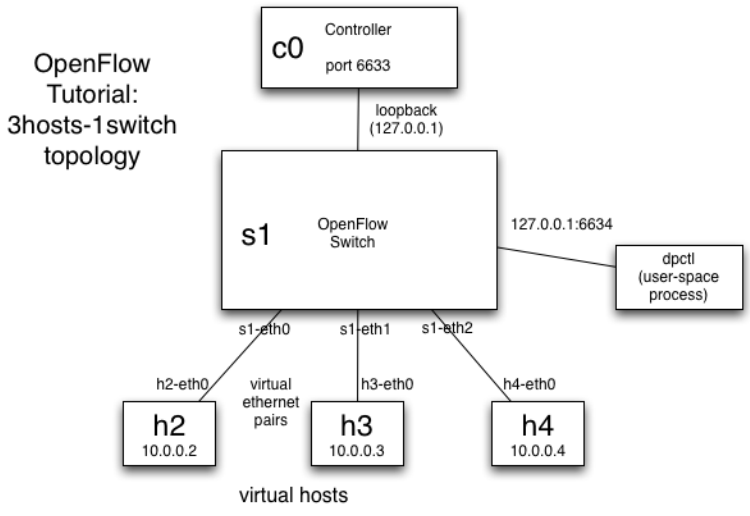
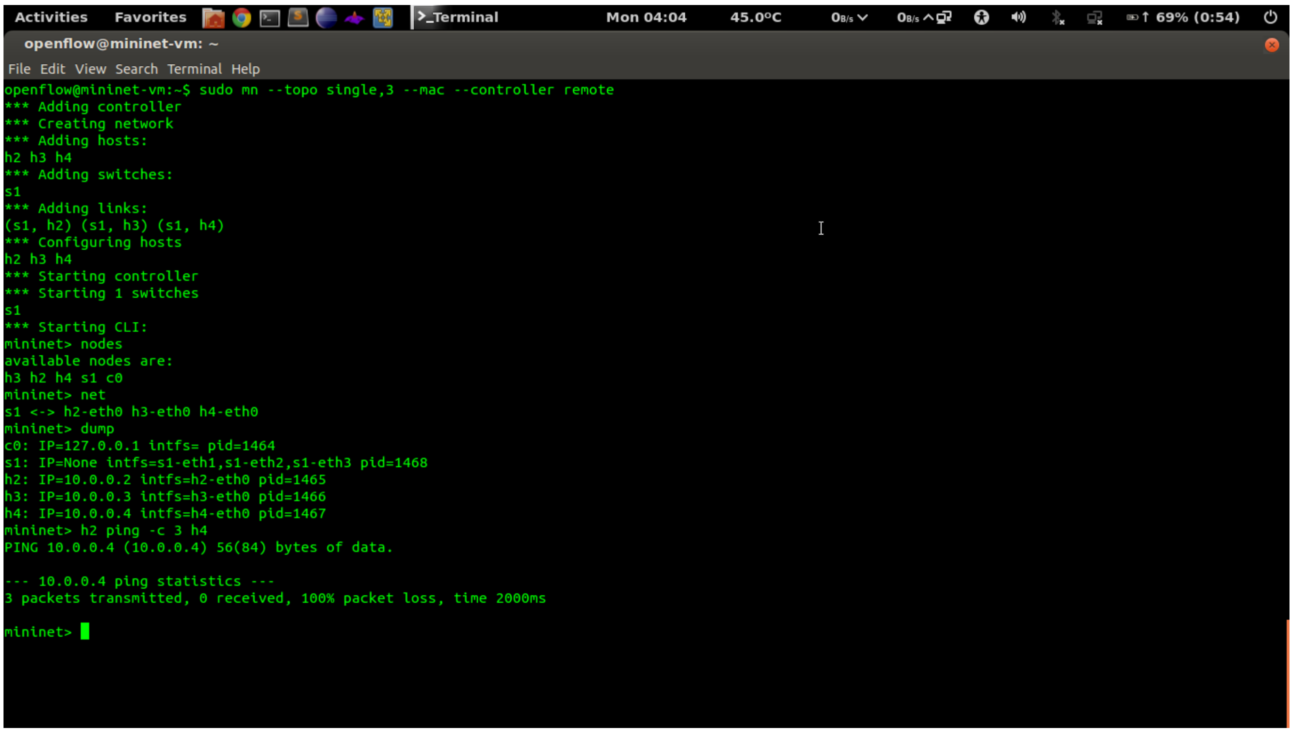


Figure 4: Sample OpenFlow topology

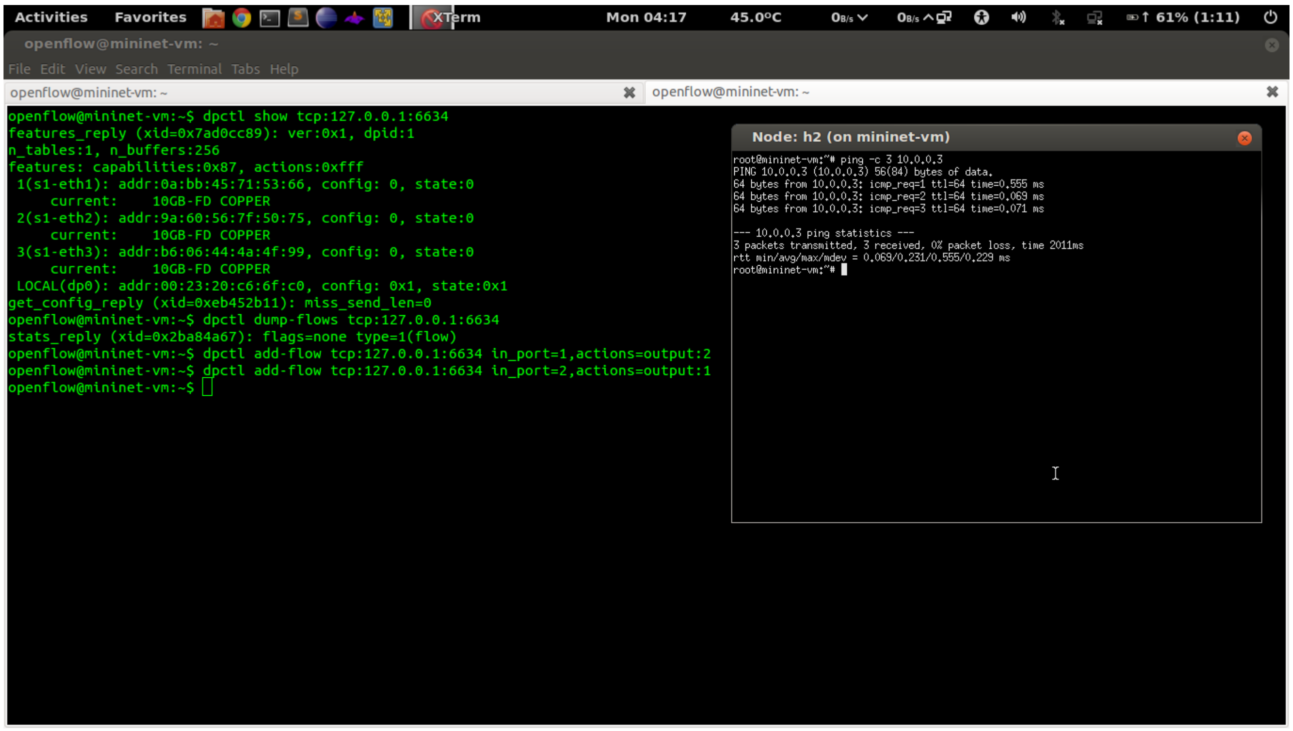


```
openflow@mininet-vm:~$ sudo mn --topo single,3 --mac --controller remote
*** Adding controller
*** Creating network
*** Adding hosts:
h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> nodes
available nodes are:
h3 h2 h4 s1 c0
mininet> net
s1 <-> h2-eth0 h3-eth0 h4-eth0
mininet> dump
c0: IP=127.0.0.1 intfs= pid=1464
s1: IP=None intfs=s1-eth1,s1-eth2,s1-eth3 pid=1468
h2: IP=10.0.0.2 intfs=h2-eth0 pid=1465
h3: IP=10.0.0.3 intfs=h3-eth0 pid=1466
h4: IP=10.0.0.4 intfs=h4-eth0 pid=1467
mininet> h2 ping -c 3 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2000ms

mininet>
```

Figure 5: Sample OpenFlow topology - simulation

4. Initially the ping from h2 to h3 or h3 to h4 fails, because the switch is not connected to controller and doesn't know where to send the incoming packets/traffic.

5. Manually add the flow entries with the help of dpctl commands.



```
openflow@mininet-vm:~$ dpctl show tcp:127.0.0.1:6634
features_reply (xid=0x7ad0cc89): ver:0x1, dpid:1
n_tables:1, n_buffers:256
features: capabilities:0x87, actions:0xfff
 1(s1-eth1): addr:0a:bb:45:71:53:66, config: 0, state:0
     current:    10GB-FD COPPER
 2(s1-eth2): addr:9a:60:56:7f:50:75, config: 0, state:0
     current:    10GB-FD COPPER
 3(s1-eth3): addr:b6:06:44:4a:4f:99, config: 0, state:0
     current:    10GB-FD COPPER
 LOCAL(dp0): addr:00:23:20:c6:6f:c0, config: 0x1, state:0x1
get_config_reply (xid=0xeb452b11): miss_send_len=0
openflow@mininet-vm:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x2ba84a67): flags=none type=1(flow)
openflow@mininet-vm:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2
openflow@mininet-vm:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=2,actions=output:1
openflow@mininet-vm:~$ 
```

```
root@mininet-vm:~# ping -c 3 10.0.0.3
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=0.555 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.069 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=0.071 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2011ms
rtt min/avg/max/mdev = 0.069/0.231/0.555/0.229 ms
root@mininet-vm:~# 
```

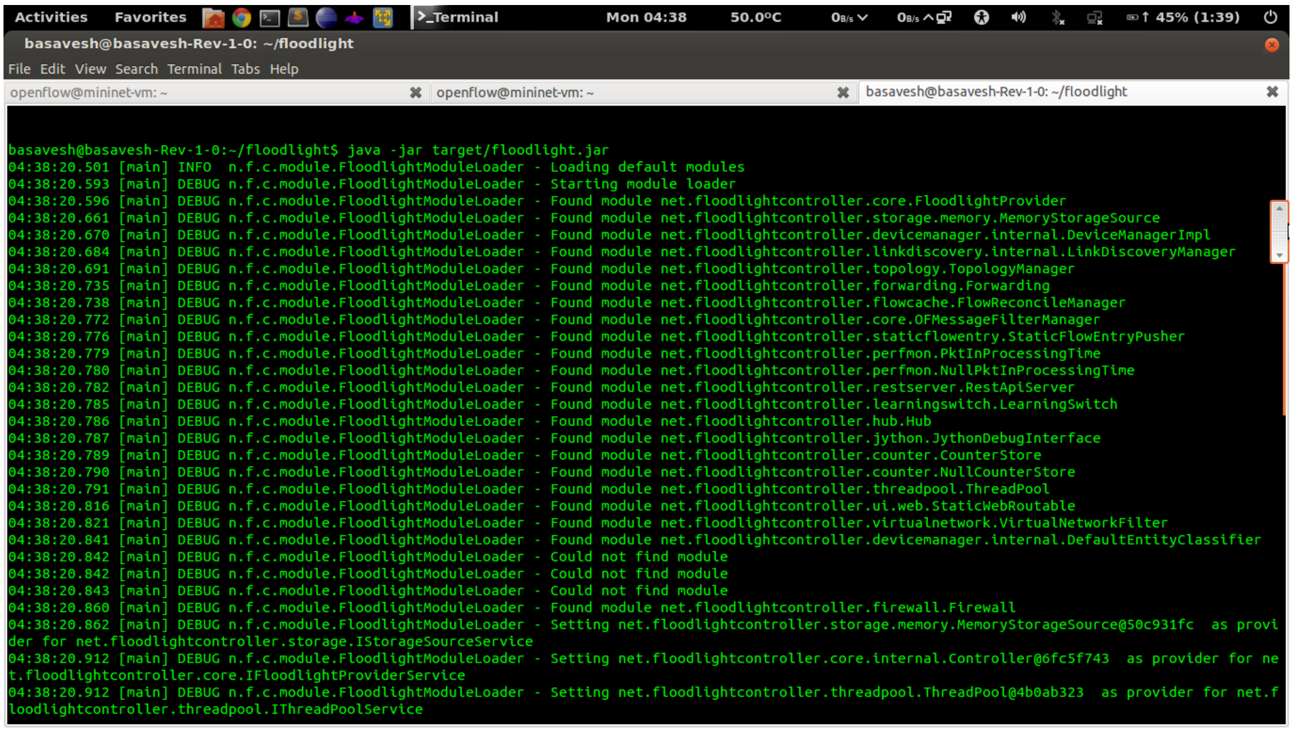
Figure 6: Manually adding flow entries

6. Now pinging from h2 to h3 will be SUCCESS!!! as there is a flow entry for that traffic.

4.1.3 OpenFlow network with Floodlight controller

1. Login in to Mininet VM.

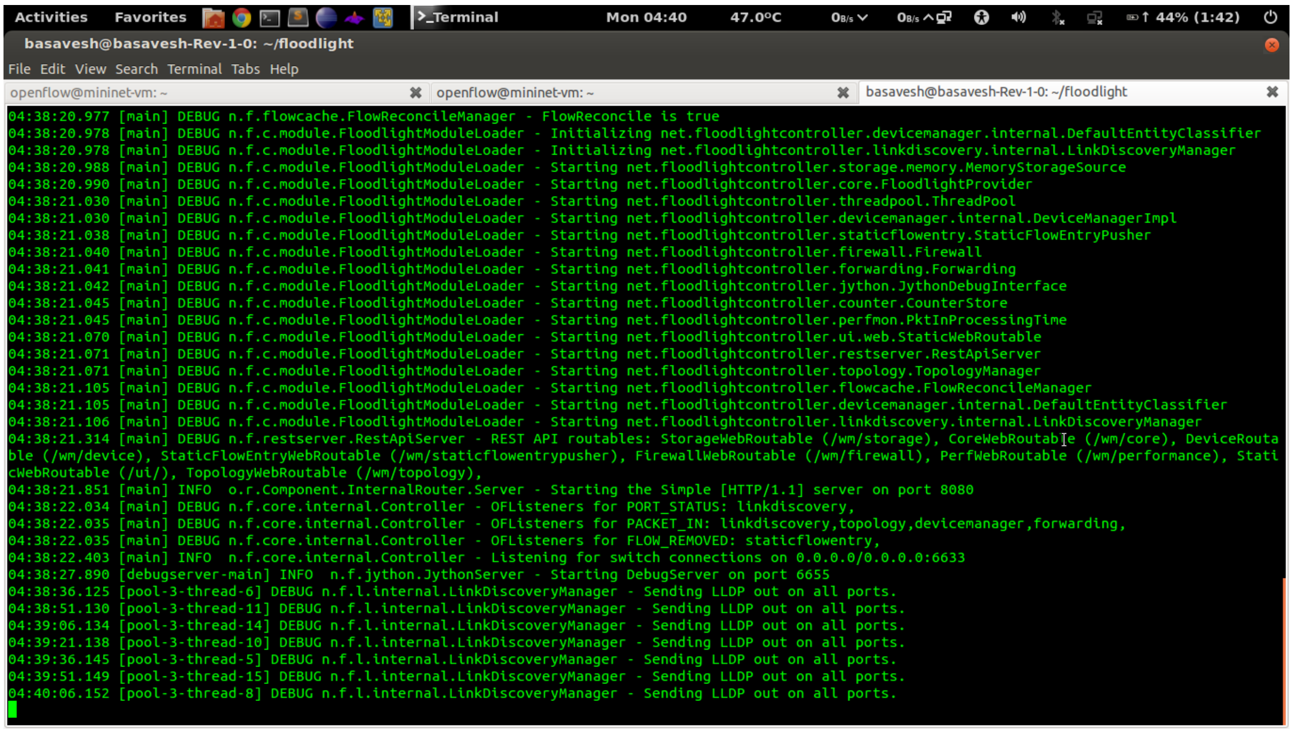
2. Download Floodlight controller and run in the Host machine.


Activities
Favorites
>_Terminal
Mon 04:38
50.0°C
0B/s
0B/s
45% (1:39)
basavesh@basavesh-Rev-1-0: ~/floodlight
File Edit View Search Terminal Tabs Help
openflow@mininet-vm: ~
openflow@mininet-vm: ~
basavesh@basavesh-Rev-1-0: ~/floodlight

```
basavesh@basavesh-Rev-1-0:~/floodlight$ java -jar target/floodlight.jar
04:38:20.501 [main] INFO  n.f.c.module.FloodlightModuleLoader - Loading default modules
04:38:20.593 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting module loader
04:38:20.596 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.core.FloodlightProvider
04:38:20.661 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.storage.memory.MemoryStorageSource
04:38:20.670 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.devicemanager.internal.DeviceManagerImpl
04:38:20.684 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager
04:38:20.691 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.topology.TopologyManager
04:38:20.735 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.forwarding.Forwarding
04:38:20.738 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.flowcache.FlowReconcileManager
04:38:20.772 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.core.OFMessageFilterManager
04:38:20.776 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.staticflowentry.StaticFlowEntryPusher
04:38:20.779 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.perfmon.PktInProcessingTime
04:38:20.780 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.perfmon.NullPktInProcessingTime
04:38:20.782 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.restserver.RestApiServer
04:38:20.785 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.learningswitch.LearningSwitch
04:38:20.786 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.hub.Hub
04:38:20.787 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.jython.JythonDebugInterface
04:38:20.789 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.counter.CounterStore
04:38:20.790 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.counter.NullCounterStore
04:38:20.791 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.threadpool.ThreadPool
04:38:20.816 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.ui.web.StaticWebRoutable
04:38:20.821 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.virtualnetwork.VirtualNetworkFilter
04:38:20.841 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.devicemanager.internal.DefaultEntityClassifier
04:38:20.842 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Could not find module
04:38:20.842 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Could not find module
04:38:20.843 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Could not find module
04:38:20.860 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Found module net.floodlightcontroller.firewall.Firewall
04:38:20.862 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Setting net.floodlightcontroller.storage.memory.MemoryStorageSource@50c931fc  as provi
der for net.floodlightcontroller.storage.IStorageSourceService
04:38:20.912 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Setting net.floodlightcontroller.core.internal.Controller@6fc5f743  as provider for ne
t.floodlightcontroller.core.IFloodlightProviderService
04:38:20.912 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Setting net.floodlightcontroller.threadpool.ThreadPool@4b0ab323  as provider for net.f
loodlightcontroller.threadpool.IThreadPoolService
```

Figure 7: Running Floodlight controller

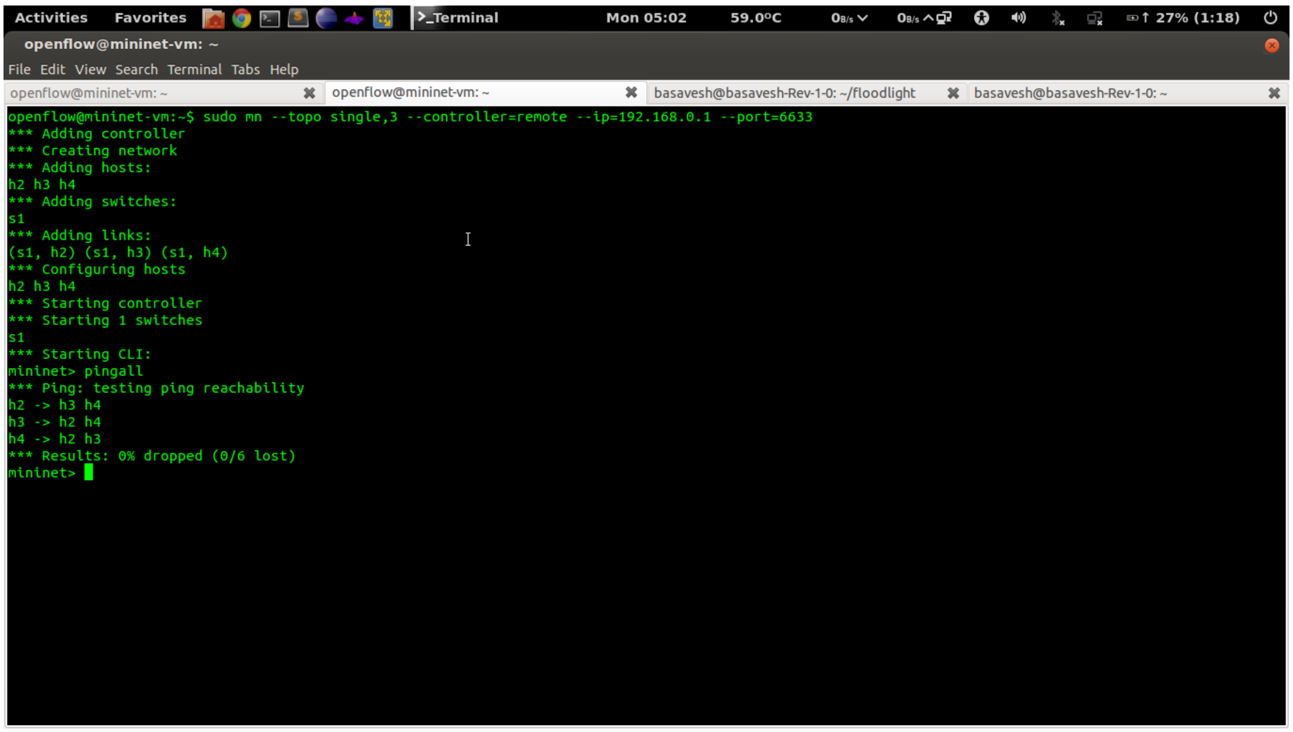
3. Floodlight controller loads all the modules and starts sending LLDP packets on all ports to discover the links.


Activities
Favorites
>_Terminal
Mon 04:40
47.0°C
44% (1:42)
basavesh@basavesh-Rev-1-0: ~/floodlight
File Edit View Search Terminal Tabs Help
openflow@mininet-vm: ~
openflow@mininet-vm: ~
basavesh@basavesh-Rev-1-0: ~/floodlight

```
04:38:20.977 [main] DEBUG n.f.flowcache.FlowReconcileManager - FlowReconcile is true
04:38:20.978 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Initializing net.floodlightcontroller.devicemanager.internal.DefaultEntityClassifier
04:38:20.978 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Initializing net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager
04:38:20.988 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.storage.memory.MemoryStorageSource
04:38:20.990 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.core.FloodlightProvider
04:38:21.030 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.threadpool.ThreadPool
04:38:21.030 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.devicemanager.internal.DeviceManagerImpl
04:38:21.038 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.staticflowentry.StaticFlowEntryPusher
04:38:21.040 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.firewall.Firewall
04:38:21.041 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.forwarding.Forwarding
04:38:21.042 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.jython.JythonDebugInterface
04:38:21.045 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.counter.CounterStore
04:38:21.045 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.perfmon.PktInProcessingTime
04:38:21.070 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.ui.web.StaticWebRoutable
04:38:21.071 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.restserver.RestApiServer
04:38:21.071 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.topology.TopologyManager
04:38:21.105 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.flowcache.FlowReconcileManager
04:38:21.105 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.devicemanager.internal.DefaultEntityClassifier
04:38:21.106 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager
04:38:21.314 [main] DEBUG n.f.restserver.RestApiServer - REST API routables: StorageWebRoutable (/wm/storage), CoreWebRoutable (/wm/core), DeviceRouta
ble (/wm/device), StaticFlowEntryWebRoutable (/wm/staticflowentrypusher), FirewallWebRoutable (/wm/firewall), PerfWebRoutable (/wm/performance), Stati
cWebRoutable (/ui/), TopologyWebRoutable (/wm/topology),
04:38:21.851 [main] INFO  o.r.Component.InternalRouter.Server - Starting the Simple [HTTP/1.1] server on port 8080
04:38:22.034 [main] DEBUG n.f.core.internal.Controller - OFListeners for PORT_STATUS: linkdiscovery,
04:38:22.035 [main] DEBUG n.f.core.internal.Controller - OFListeners for PACKET_IN: linkdiscovery,topology,devicemanager,forwarding,
04:38:22.035 [main] DEBUG n.f.core.internal.Controller - OFListeners for FLOW_REMOVED: staticflowentry,
04:38:22.403 [main] INFO  n.f.core.internal.Controller - Listening for switch connections on 0.0.0.0/0.0.0.0:6633
04:38:27.890 [debugserver-main] INFO  n.f.jython.JythonServer - Starting DebugServer on port 6655
04:38:36.125 [pool-3-thread-6] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
04:38:51.130 [pool-3-thread-11] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
04:39:06.134 [pool-3-thread-14] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
04:39:21.138 [pool-3-thread-10] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
04:39:36.145 [pool-3-thread-5] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
04:39:51.149 [pool-3-thread-15] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
04:40:06.152 [pool-3-thread-8] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
```

Figure 8: Floodlight controller sending LLDP packets

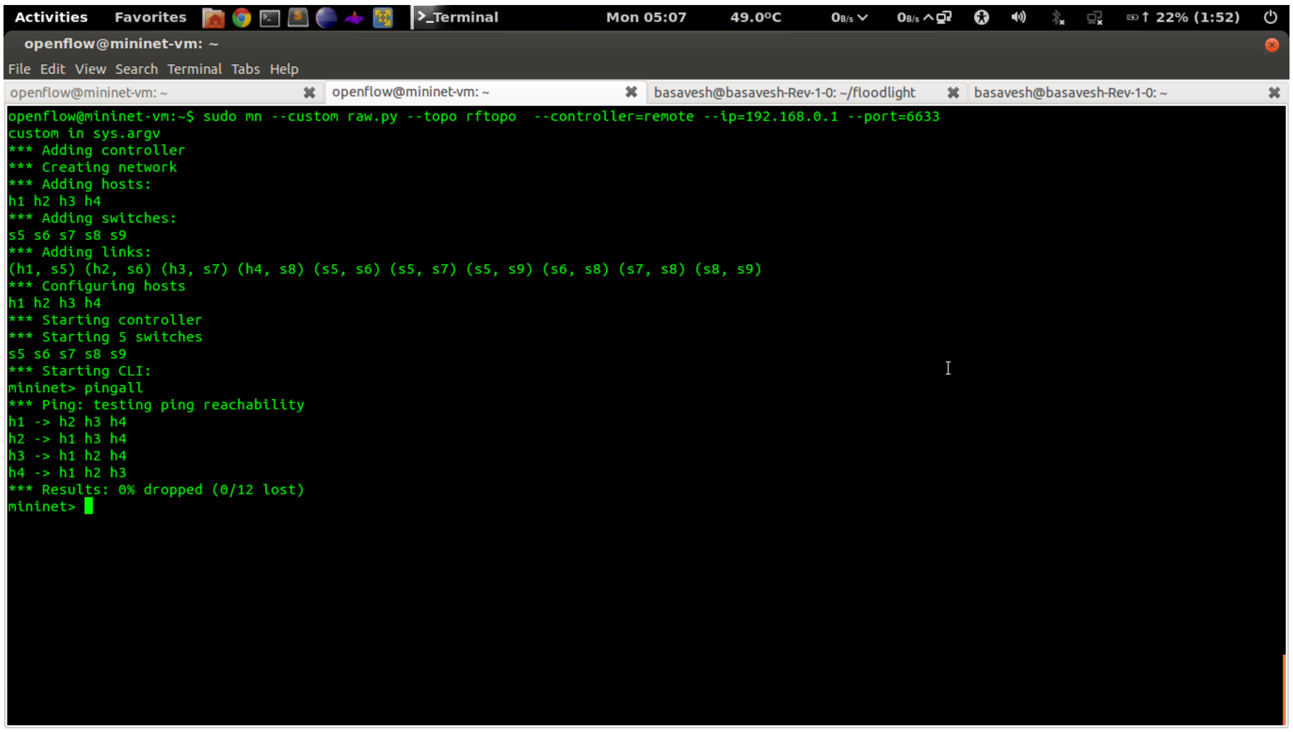
4. Create sample topology and attach remote controller(on port 6633) and ping all hosts.



```
openflow@mininet-vm:~$ sudo mn --topo single,3 --controller=remote --ip=192.168.0.1 --port=6633
*** Adding controller
*** Creating network
*** Adding hosts:
h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h2 -> h3 h4
h3 -> h2 h4
h4 -> h2 h3
*** Results: 0% dropped (0/6 lost)
mininet>
```

Figure 9: Ping success in sample topology

5. Create custom topology and attach remote controller(on port 6633) and ping all hosts.



```
openflow@mininet-vm:~$ sudo mn --custom raw.py --topo rftopo  --controller=remote --ip=192.168.0.1 --port=6633
custom in sys.argv
*** Adding controller
*** Creating network
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s5 s6 s7 s8 s9
*** Adding links:
(h1, s5) (h2, s6) (h3, s7) (h4, s8) (s5, s6) (s5, s7) (s5, s9) (s6, s8) (s7, s8) (s8, s9)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 5 switches
s5 s6 s7 s8 s9
*** Starting CLI:
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)
mininet> 
```

Figure 10: Ping success in custom topology

4.1.4 Floodlight - Shortest path algorithm

Currently Floodlight controller (topology aware) uses Shortest path algorithm (Djikstra).

Custom topology

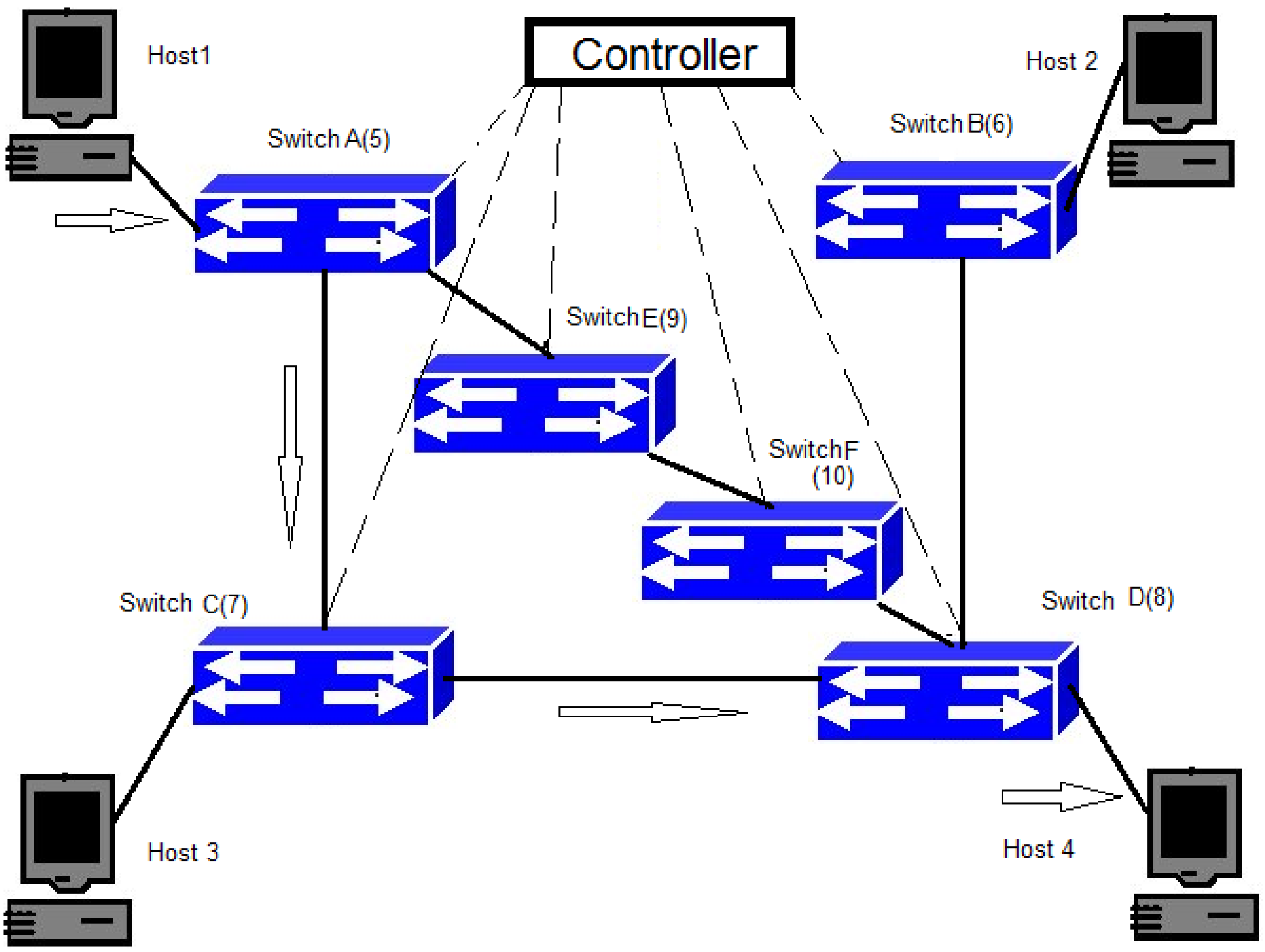


Figure 11: Topology for Shortest path algorithm

Steps involved in finding path

- DeviceManagerImpl module will know the devices registered in the topology.
- LinkDiscoveryManager module will know the links/edges between nodes
- TopologyManager module will use the Djikstra algorithm and forwards the packets using Forwarding module.

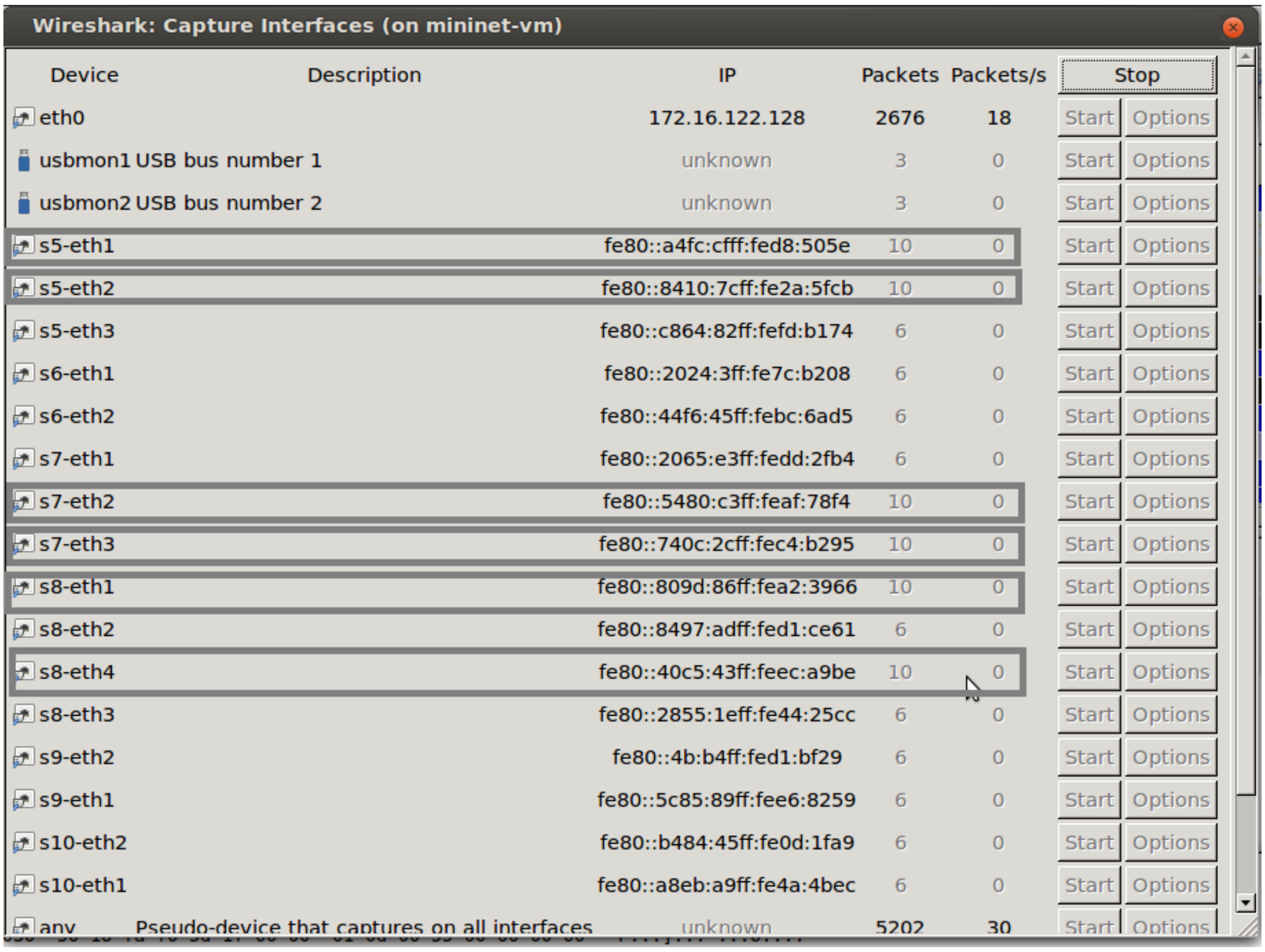
For this topology, when we ping from h1 to h4. The packets will be forwarded from H1–> S5, S5–> S7, S7–>S8, S8–> H4, which is the shortest path.

5 Results and Analysis

For this custom topology show in Figure 4.1.4 on page 15, when we ping from h1 to h4. The packets will be forwarded from H1–> S5, S5–> S7, S7–>S8, S8–> H4, which is the shortest path.

This is verified by two methods:

1. Packets caputred by Wireshark



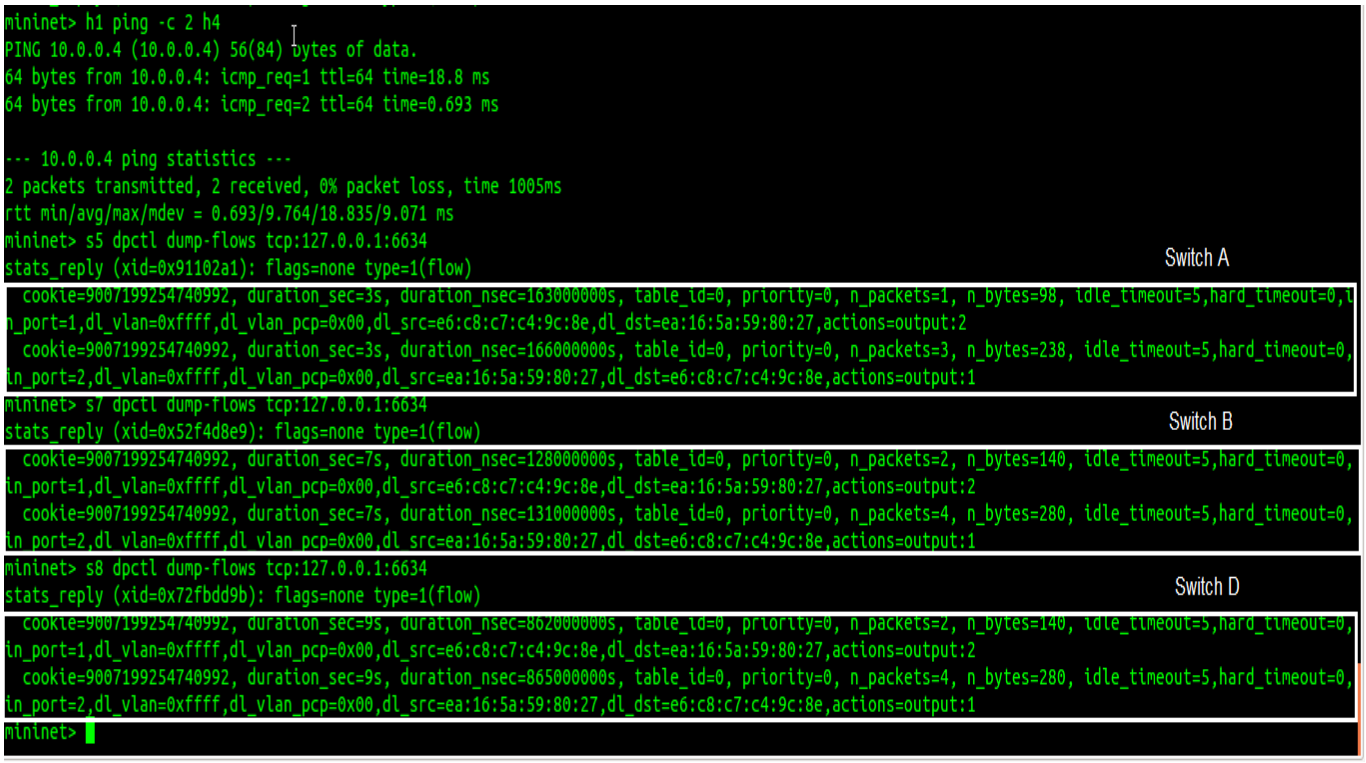
Wireshark: Capture Interfaces (on mininet-vm)

Device	Description	IP	Packets	Packets/s	Stop
eth0		172.16.122.128	2676	18	Start Options
usbmon1	USB bus number 1	unknown	3	0	Start Options
usbmon2	USB bus number 2	unknown	3	0	Start Options
s5-eth1		fe80::a4fc:cfff:fed8:505e	10	0	Start Options
s5-eth2		fe80::8410:7cff:fe2a:5fcb	10	0	Start Options
s5-eth3		fe80::c864:82ff:fefd:b174	6	0	Start Options
s6-eth1		fe80::2024:3ff:fe7c:b208	6	0	Start Options
s6-eth2		fe80::44f6:45ff:febc:6ad5	6	0	Start Options
s7-eth1		fe80::2065:e3ff:fedd:2fb4	6	0	Start Options
s7-eth2		fe80::5480:c3ff:feaf:78f4	10	0	Start Options
s7-eth3		fe80::740c:2cff:fec4:b295	10	0	Start Options
s8-eth1		fe80::809d:86ff:fea2:3966	10	0	Start Options
s8-eth2		fe80::8497:adff:fed1:ce61	6	0	Start Options
s8-eth4		fe80::40c5:43ff:feec:a9be	10	0	Start Options
s8-eth3		fe80::2855:1eff:fe44:25cc	6	0	Start Options
s9-eth2		fe80::4b:b4ff:fed1:bf29	6	0	Start Options
s9-eth1		fe80::5c85:89ff:fee6:8259	6	0	Start Options
s10-eth2		fe80::b484:45ff:fe0d:1fa9	6	0	Start Options
s10-eth1		fe80::a8eb:a9ff:fe4a:4bec	6	0	Start Options
any	Pseudo-device that captures on all interfaces	unknown	5202	30	Start Options

Figure 12: packets captured by wireshark

Except the LLDP packets broadcasted by the controller. These switch shows the flow of additional packets generated by the ping (ICMP).

2. Flow entries added by the controller on Switches (A,C,D).



```
mininet> h1 ping -c 2 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_req=1 ttl=64 time=18.8 ms
64 bytes from 10.0.0.4: icmp_req=2 ttl=64 time=0.693 ms

--- 10.0.0.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1005ms
rtt min/avg/max/mdev = 0.693/9.764/18.835/9.071 ms
mininet> s5 dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x91102a1): flags=none type=1(flow)
  cookie=9007199254740992, duration_sec=3s, duration_nsec=163000000s, table_id=0, priority=0, n_packets=1, n_bytes=98, idle_timeout=5,hard_timeout=0,i
n_port=1,dl_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=e6:c8:c7:c4:9c:8e,dl_dst=ea:16:5a:59:80:27,actions=output:2
  cookie=9007199254740992, duration_sec=3s, duration_nsec=166000000s, table_id=0, priority=0, n_packets=3, n_bytes=238, idle_timeout=5,hard_timeout=0,
in_port=2,dl_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=ea:16:5a:59:80:27,dl_dst=e6:c8:c7:c4:9c:8e,actions=output:1
mininet> s7 dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x52f4d8e9): flags=none type=1(flow)
  cookie=9007199254740992, duration_sec=7s, duration_nsec=128000000s, table_id=0, priority=0, n_packets=2, n_bytes=140, idle_timeout=5,hard_timeout=0,
in_port=1,dl_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=e6:c8:c7:c4:9c:8e,dl_dst=ea:16:5a:59:80:27,actions=output:2
  cookie=9007199254740992, duration_sec=7s, duration_nsec=131000000s, table_id=0, priority=0, n_packets=4, n_bytes=280, idle_timeout=5,hard_timeout=0,
in_port=2,dl_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=ea:16:5a:59:80:27,dl_dst=e6:c8:c7:c4:9c:8e,actions=output:1
mininet> s8 dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x72fbdd9b): flags=none type=1(flow)
  cookie=9007199254740992, duration_sec=9s, duration_nsec=862000000s, table_id=0, priority=0, n_packets=2, n_bytes=140, idle_timeout=5,hard_timeout=0,
in_port=1,dl_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=e6:c8:c7:c4:9c:8e,dl_dst=ea:16:5a:59:80:27,actions=output:2
  cookie=9007199254740992, duration_sec=9s, duration_nsec=865000000s, table_id=0, priority=0, n_packets=4, n_bytes=280, idle_timeout=5,hard_timeout=0,
in_port=2,dl_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=ea:16:5a:59:80:27,dl_dst=e6:c8:c7:c4:9c:8e,actions=output:1
mininet>
```

Figure 13: Flow tables added by controller

This shows the flow table entries of Switch A, C and D.

Switch A:

The important entries of this Switch flow table are:

Entry 1

src= e6:c8:c7:c4:9c:8e

dst= ea:16:5a:59:80:27

input-port=1

action = output:2

Entry 2

src= ea:16:5a:59:80:27

dst= e6:c8:c7:c4:9c:8e

input-port=2

action = output:1

Similar entries are added for switch C and switch D

6 Conclusion and Futurework

From our simulations, the working of the Mininet simulator and Floodlight Controller was clearly understood.The role of each module in control decision is identified.

To forward a incoming packets the switch needs to have flow entries for that particular source and destination path.If the entry is not present the switch will contact the controller for forwarding packets.We can add the flow entries manually or automatically with the help of controller.

It was found and verified that Floodlight controller uses Djikstra's Shortest path algorithm to forward the packets from source to destination in layer 2.Once the shortest path is identified by the controller, it adds necessary flow entries in switches accordingly. With the help of these entries the packets are forwareded from source to destination.

In future stages of work, we will be simulating a topology with multipath from source to destination. With the help of load generator we will demonstrate that congestion occurs in network when we use single path (Shortest path algorithm).

We will implement Equal cost multipath routing and demonstrate that we can achieve more performance by distributing the load among different paths.

References

[1] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan Turner , *OpenFlow: Enabling Innovation in Campus Networks.*ACM SIGCOMM, Volume 38 Issue 2, April, 2008.

[2] Floodlight Controller, *http: // floodlight. openflowhub. org/*

[3] Multipath Routing, *https: // en. wikipedia. org/ wiki/ Multipath_ routing*

[4] C Hopps, *Analysis of an Equal-Cost Multi-Path Algorithm*, https://ebook.tools.ietf.org/html/rfc2992,2000

[5] Mininet Documentation, *http: // yuba. stanford. edu/ foswiki/ bin/ view/ OpenFlow/ MininetDocumentation*

[6] Floodlight modules, *http: // www. openflowhub. org/ display/ floodlightcontroller/ Floodlight+ Controller+ Wiki*

[7] Jiayue He and Jennifer Rexford, *Toward Internet-Wide Multipath Routing* ,Network-IEEE,Vol:22,Issue:2, March-April 2008

Appendix

A Custom topology code - Shortest path

```
from mininet.topo import Topo, Node

class RFTopo( Topo ):
    "RouteFlow Demo Setup"

    def __init__( self, enable_all = True ):
        "Create custom topo."

        # Add default members to class.
        super( RFTopo, self ).__init__()

        # Set Node IDs for hosts and switches
        h1 = 1
        h2 = 2
        h3 = 3
        h4 = 4
        sA = 5
        sB = 6
        sC = 7
        sD = 8
        sE = 9
        sF = 10

        # Add nodes
        self.add_node( h1, Node( is_switch=False ) )
        self.add_node( h2, Node( is_switch=False ) )
        self.add_node( h3, Node( is_switch=False ) )
        self.add_node( h4, Node( is_switch=False ) )
        self.add_node( sA, Node( is_switch=True ) )
        self.add_node( sB, Node( is_switch=True ) )
        self.add_node( sC, Node( is_switch=True ) )
        self.add_node( sD, Node( is_switch=True ) )
        self.add_node( sE, Node( is_switch=True) )
        self.add_node( sF, Node( is_switch=True) )

        # Add edges
        self.add_edge( h1, sA )
        self.add_edge( h2, sB )
        self.add_edge( h3, sC )
        self.add_edge( h4, sD )
        self.add_edge( sE, sF )
        self.add_edge( sB, sD )
        self.add_edge( sD, sF )
        self.add_edge( sC, sA )
        self.add_edge( sA, sE )
        self.add_edge( sC, sD )
#       self.add_edge( sE, sD )
#       self.add_edge( sA, sB )



        # Consider all switches and hosts 'on'
        self.enable_all()


topos = { 'rftopo': ( lambda: RFTopo() ) }
```