Application Aware Data Forwarding in SDN framework

A Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Technology
in
Computer Science & Engineering

by

Shashank Sharma, Sacchidanand Verma and Vidyanand Kumar

to the

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY
ALLAHABAD PRAYAGRAJ

May, 2021

UNDERTAKING

I declare that the work presented in this report titled "*Application Aware Data Forwarding in SDN framework*", submitted to the Computer Science and Engineering Department, Motilal Nehru National Institute of Technology Allahabad, Prayagraj, for the award of the ***Bachelor of Technology*** degree in ***Computer Science & Engineering***, is my original work. I have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, I accept that my degree may be unconditionally withdrawn.

May, 2021
Allahabad

(Shashank Sharma,
Sacchidanand Verma and
Vidyanand Kumar)

CERTIFICATE

Certified that the work contained in the report titled "*Application Aware Data Forwarding in SDN framework*", by *Shashank Sharma, Sacchidanand Verma and Vidyanand Kumar*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Neeraj Tyagi)
Computer Science and Engineering Dept.
M.N.N.I.T, Allahabad

May, 2021

Abstract

Software-Defined Network is completely new networking paradigm which is growing rapidly. To have a good understanding of it is must in order to contribute something from our side. Our aim is to understand the step-by-step execution of device detection(host and switches), topology figuration, finding the optimal path by OpenFlow controller(FLoodLight in our case) for packet-in based on shortest path algorithm(traditional) and so on. Once we are done with it our main focus is to make module corresponding to process packet and find out type of payload(TCP, UDP, ARP, etc.) coming into it, by statistical estimation we fetch the bandwidth of links and then divert the packet(let's say UDP) to path with maximum bandwidth. We call this widest path. The same concept can be extrapolated to different cases we come across for optimal packet forwarding.

Acknowledgements

On the very outset of this report, we would like to extend our heartfelt and sincere obligation towards all the personages who have held us in this endeavour. With- out their active, help, co-operation and encouragement, we would not have made headway in this project. We are especially grateful to our mentor Prof. Neeraj Tyagi, Department of Computer Science and Engineering, M.N.N.I.T. Allahabad for providing us with assistance and necessary facilities in our project. Without his expertise and suggestions, we would never have been able to complete our work.
We are highly grateful to Dr. Mayank Pandey, Department of Computer Science and Engineering, M.N.N.I.T. Allahabad for providing necessary facilities and encouraging for doing the course of work. We are thankful to all technical and non-academic staff of the department for their constant assistance and co-operation.

Contents

Chapter 1

Introduction

Software-defined networking (SDN) is clearly one of the hot items of the tech field at the moment. VMware's purchase of Nicira precipitated a sea change, leading to today's plethora of SDN vendors and array of competing technologies. It reminds me the early noughties—the introduction of virtualization, competing hypervisor technology stacks and Unix/Linux Zones*—followed by the scramble of the incumbents as they claimed performance penalties for virtualized operating systems and platforms, followed by spreading FUD about support status and onerous licensing models. We created a virtual test-bed using Mininet, SDN emulator, using Floodlight SDN controller. The network is set up along with a SDN controller and the transmitted packets and the resulting flows are to be studied.
Our main work was to find the optimal forwarding path for different type of payloads i.e, TCP, UDP, ARP, etc. For now we tried finding widest path(path with maximum Bandwidth) whenever UDP packet was encountered by switch. Once we are done with it, extrapolation becomes way more easier and we can implement other forwarding ideas like path with minimum latency for let's say TCP Packet.

1.1 Objective

The tasks assigned to us were as follows:

- Studying about SDN, Mininet, FloodLight Controller and OpenFlow. Consider a topology of around 40-50 nodes and 7-8 switches. (for a close to real topology you can refer to "topology-zoo.").
- Create a topology with different link properties (i.e., with different bandwidth, different loss-rate and different latency; you can tweak the parameters using Mininet Pyhton script.)
- Topology should have multiple paths from source to destination with varied link characteristics.
- Assume that the end-hosts are running different applications. (Video Streaming, large file transfers using FTP protocol, and simple HTTP based web access).
- Write a new module in the Floodlight controller, which can first analyse the type of traffic or application and then create source to destination flow-entries across switches depending upon the application requirement.(for example: video streaming needs high bandwidth and low latency path, http based web access can be made through a low bandwidth path, etc.)
- Verify and show proof that your controller is able to perform application-aware forwarding.
- Compare the performance of your implementation over the same set of experiments with unmodified controller application.

Chapter 2

Software-Defined Networking

Software-defined networking (SDN) is a category of technologies that make it possible to manage a network via software. SDN technology enables IT administrators to configure their networks using a software application, instead of changing the configuration of physical equipment. SDN software is interoperable, meaning it should be able to work with any router or switch, no matter which vendor made it.
Think of the difference between typing an essay on a typewriter and typing it in a word processing application: Any changes to the typewritten essay mean it has to be retyped, while a word processing document can be revised endlessly. Similarly, SDN technology lets admins reconfigure their networks from a computer, instead of continually plugging and unplugging cables and devices.

2.1 The SDN Architecture

The application layer, not surprisingly, contains the typical network applications or functions organizations use, which can include intrusion detection systems, load balancing or firewalls. Where a traditional network would use a specialized appliance, such as a firewall or load balancer, a software-defined network replaces the appliance with an application that uses the controller to manage data plane behavior. SDN architecture separates the network into three distinguishable layers,

connected through northbound and southbound APIs.
The control layer represents the centralized SDN controller software that acts as the brain of the software-defined network. This controller resides on a server and manages policies and the flow of traffic throughout the network.
The infrastructure layer is made up of the physical switches in the network.
These three layers communicate using respective northbound and southbound application programming interfaces (APIs). For example, applications talk to the controller through its northbound interface, while the controller and switches communicate using southbound interfaces, such as OpenFlow – although other protocols exist.

Chapter 3

Networking Fundamentals Used

3.1 OpenFlow

OpenFlow is a protocol that gives access to the forwarding plane of a network switch. In the OpenFlow architecture is illustrated in the figure. OpenFlow switch is the forwarding device which consists of flow tables as well as an abstraction layer that securely communicates with a controller via OpenFlow protocol.
Flow tables consist of flow entries which determines how packets belonging to a flow will be forwarded. Flow entries consist of three components: matching rules, used to match incoming packets; counters, used to collect statistics for the particular flow, such as number of received packets, number of bytes and duration of the flow; and actions, to be applied for matching packets. When a packet arrives at an OpenFlow switch, packet header fields are matched against the matching fields of the flow table entries. If a matching entry is found, the switch applies the required actions according to the matched flow entry. If there is no match, the switch takes action depending on the instructions defined by the table-miss flow entry, which consists of actions to be performed when no match is found.
OpenFlow protocol handles the communication between controller and switch.
A remote controller can add, update, or delete flow entries from the switch's flow tables using the OpenFlow protocol.

Chapter 4

Experimental Setup

4.1 Tools used

4.1.1 Mininet

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.

4.1.2 FloodLight Controller

Floodlight Controller is an OpenFlow based SDN Controller. Floodlight is not only an OpenFlow controller but also a collection of applications built on top the Flood-light Controller. Floodlight is designed to work with a number of switches, routers, virtual switches, and access points that support the OpenFlow standard. Flood-light Controller offers the ability to easily adapt software and develop applications.
Floodlight Controller can work with both physical and virtual OpenFlow-compatible

Switches.

4.1.3 Wireshark

Wireshark is the world's leading network traffic analyzer, and an essential tool for any security professional or systems administrator. This free software lets you analyze network traffic in real time, and is often the best tool for troubleshooting issues on your network.

4.2 Our Approach

4.2.1 Installation and Exploration

The process started with installation of required operating system, software and libraries. The steps followed are listed below.

- Installing Mininet

```
git clone \\*git://github.com/mininet/mininet
cd mininet
git tag  \# list available versions
git checkout -b 2.2.1 2.2.1
\# or whatever version you wish to
install
cd ..
mininet/util/install.sh [options]
```

- Installing Floodlight Controller

```
git clone
git://github.com/floodlight/floodlight.git
cd floodlight
```

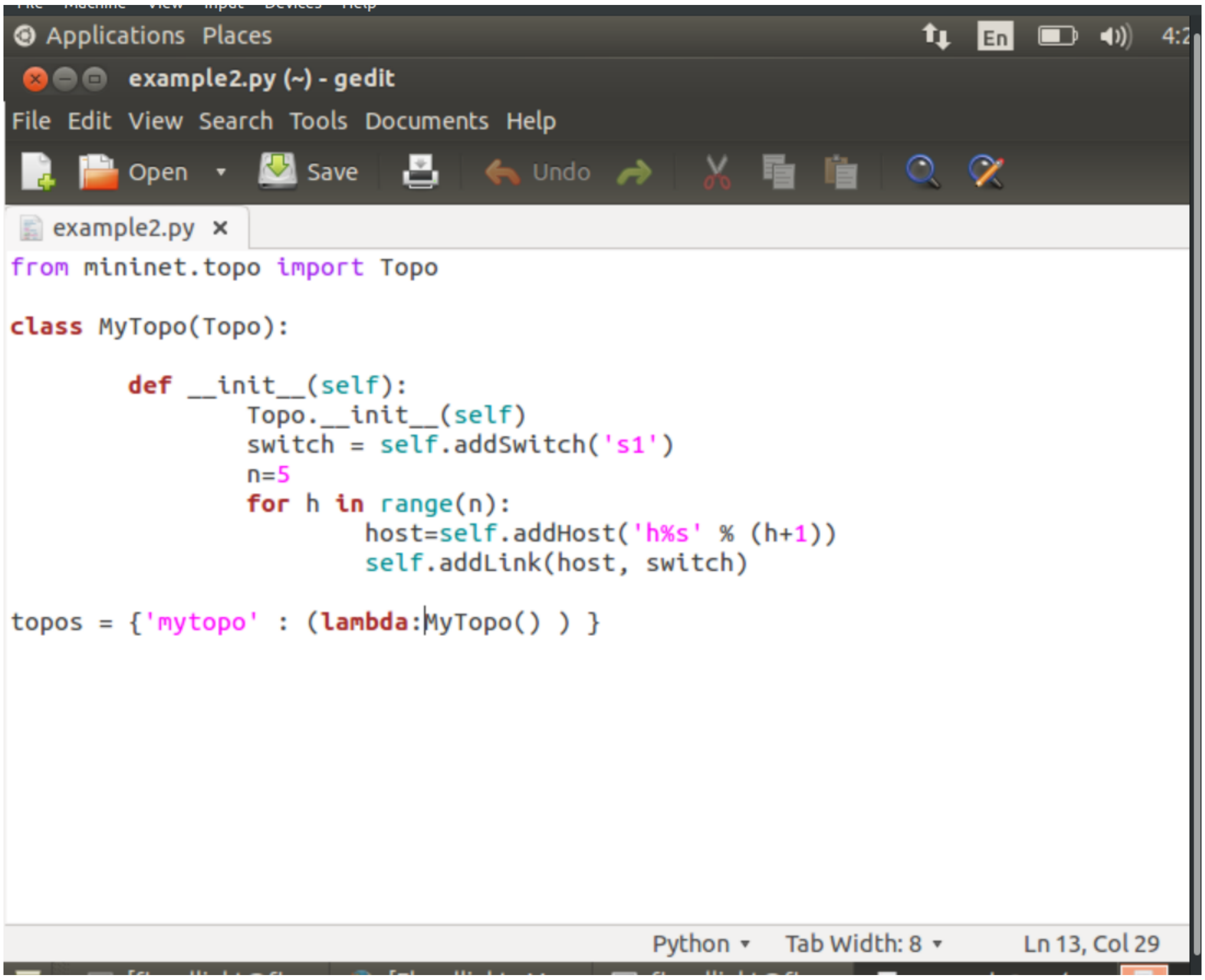
```
git submodule init
git submodule update
ant
sudo mkdir /var/lib/floodlight
sudo chmod 777 /var/lib/floodlight
```

- Installing Wireshark
  sudo apt-get update
  sudo apt install wireshark

4.2.2 Exploring Topology framing

▮ *Writing a Python Script*

We can create topologies in the Mininet-Wi-Fi by writing a python script. The number of host nodes, link layer devices and network layer devices all can be positioned and connected as desired in the topology. In this test, we have created a simple topology. The code to create a topology as mentioned above is given below.


Applications Places
example2.py (~) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
example2.py
Python Tab Width: 8 Ln 13, Col 29

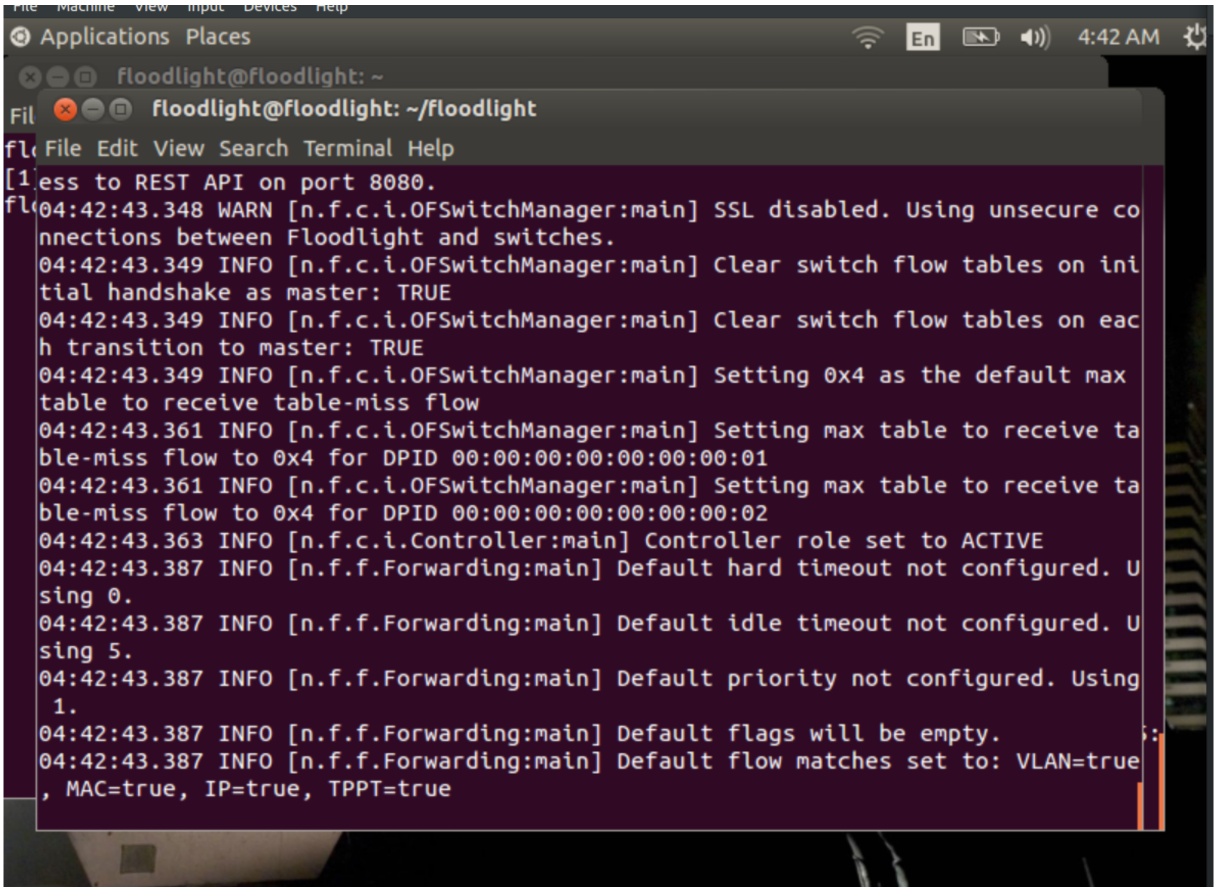
```
from mininet.topo import Topo

class MyTopo(Topo):

        def __init__(self):
                Topo.__init__(self)
                switch = self.addSwitch('s1')
                n=5
                for h in range(n):
                        host=self.addHost('h%s' % (h+1))
                        self.addLink(host, switch)

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

Figure 1: Working of Project

■ *Running FloodLight Controller*

Firstly, the current directory is changed to the directory of Floodlight SDN controller. Then floodlight is executed using the following command.

```
cd floodlight
java {jar target/floodlight.jar
```

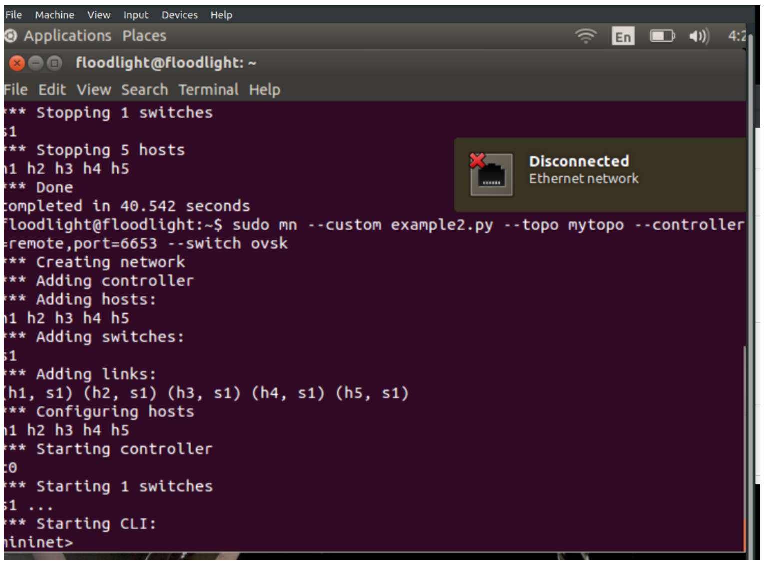

Applications Places
4:42 AM
floodlight@floodlight: ~
floodlight@floodlight: ~/floodlight
File Edit View Search Terminal Help

```
ess to REST API on port 8080.
04:42:43.348 WARN [n.f.c.i.OFSwitchManager:main] SSL disabled. Using unsecure co
nnections between Floodlight and switches.
04:42:43.349 INFO [n.f.c.i.OFSwitchManager:main] Clear switch flow tables on ini
tial handshake as master: TRUE
04:42:43.349 INFO [n.f.c.i.OFSwitchManager:main] Clear switch flow tables on eac
h transition to master: TRUE
04:42:43.349 INFO [n.f.c.i.OFSwitchManager:main] Setting 0x4 as the default max
table to receive table-miss flow
04:42:43.361 INFO [n.f.c.i.OFSwitchManager:main] Setting max table to receive ta
ble-miss flow to 0x4 for DPID 00:00:00:00:00:00:00:01
04:42:43.361 INFO [n.f.c.i.OFSwitchManager:main] Setting max table to receive ta
ble-miss flow to 0x4 for DPID 00:00:00:00:00:00:00:02
04:42:43.363 INFO [n.f.c.i.Controller:main] Controller role set to ACTIVE
04:42:43.387 INFO [n.f.f.Forwarding:main] Default hard timeout not configured. U
sing 0.
04:42:43.387 INFO [n.f.f.Forwarding:main] Default idle timeout not configured. U
sing 5.
04:42:43.387 INFO [n.f.f.Forwarding:main] Default priority not configured. Using
 1.
04:42:43.387 INFO [n.f.f.Forwarding:main] Default flags will be empty.
04:42:43.387 INFO [n.f.f.Forwarding:main] Default flow matches set to: VLAN=true
, MAC=true, IP=true, TPPT=true
```

Figure 2: Working of Project

- *Launching Mininet*

```
sudo mn {custom example2.py --topo mytopo {controller=remote, port=6653 {switch ovsk
```



```
** Stopping 1 switches
1
** Stopping 5 hosts
1 h2 h3 h4 h5
** Done
ompleted in 40.542 seconds
loodlight@floodlight:~$ sudo mn --custom example2.py --topo mytopo --controller
remote,port=6653 --switch ovsk
** Creating network
** Adding controller
** Adding hosts:
1 h2 h3 h4 h5
** Adding switches:
1
** Adding links:
h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1)
** Configuring hosts
1 h2 h3 h4 h5
** Starting controller
0
** Starting 1 switches
1 ...
** Starting CLI:
ininet>
```

Figure 3: Working of Project

- *Topology WebView*

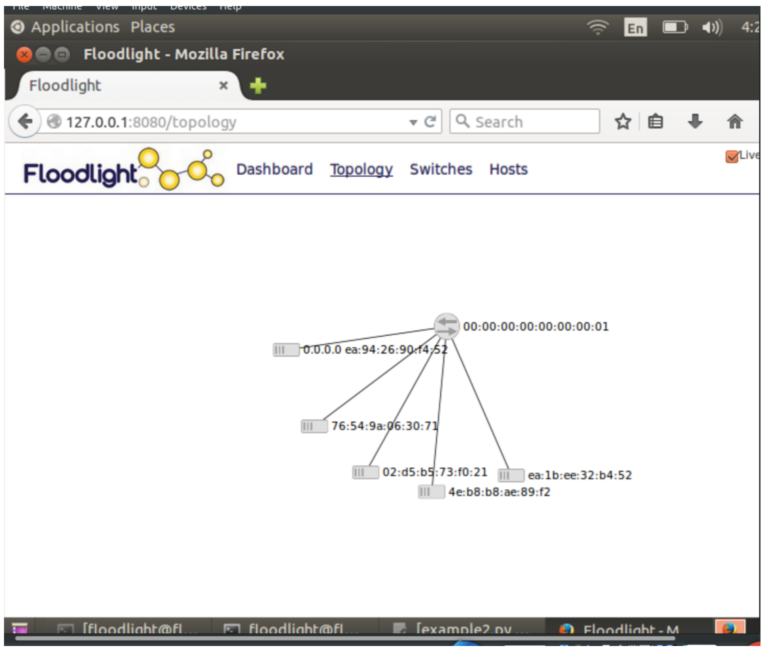
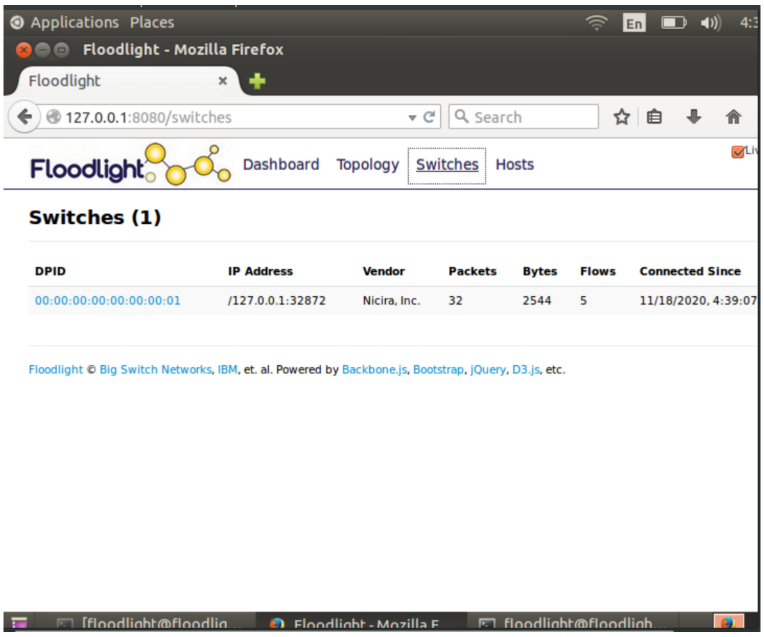


Figure 4: Working of Project

- *Switches in Network*



Switches (1)

DPID	IP Address	Vendor	Packets	Bytes	Flows	Connected Since
00:00:00:00:00:00:00:01	/127.0.0.1:32872	Nicira, Inc.	32	2544	5	11/18/2020, 4:39:07

Floodlight © Big Switch Networks, IBM, et. al. Powered by Backbone.js, Bootstrap, jQuery, D3.js, etc.

Figure 5: Working of Project

- *Hosts*

- *Pinging from one host to another*

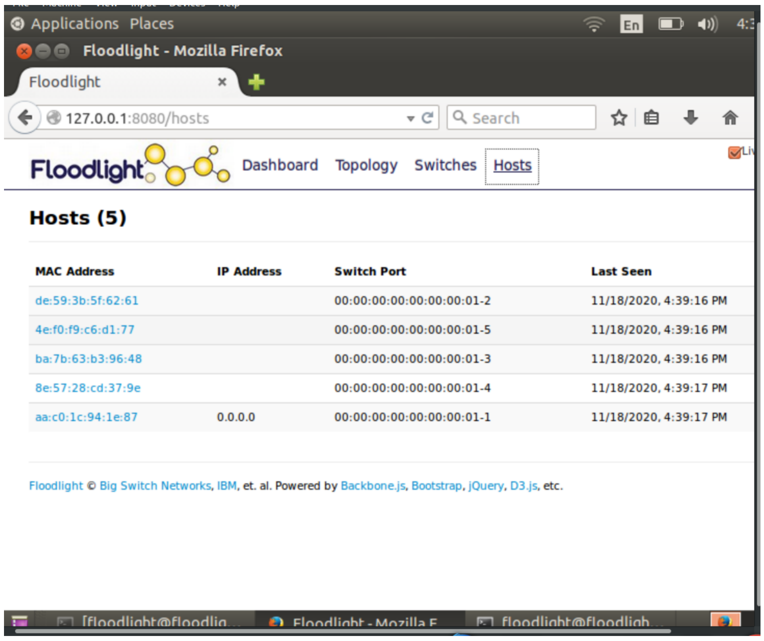
A terminal of end node named sta5 is opened. xterm h1 In the terminal of sta5, type the command to ping to access point sta1. ping 10.0.0.1

- *Capturing Packets Through Wireshark*

Wireshark can be started so as to view wireless control traffic.

```
sudo wireshark
```

Firstly, Wireshark is run on the terminal of the controller machine. LLDP packets sent between the controller and Mininet-WiFi is observed.
Through Wireshark, the ICMP packets encapsulated in wireless frames during the pinging between nodes are also observed. This can be done by running Wireshark in the terminals of the nodes pinging to each other.

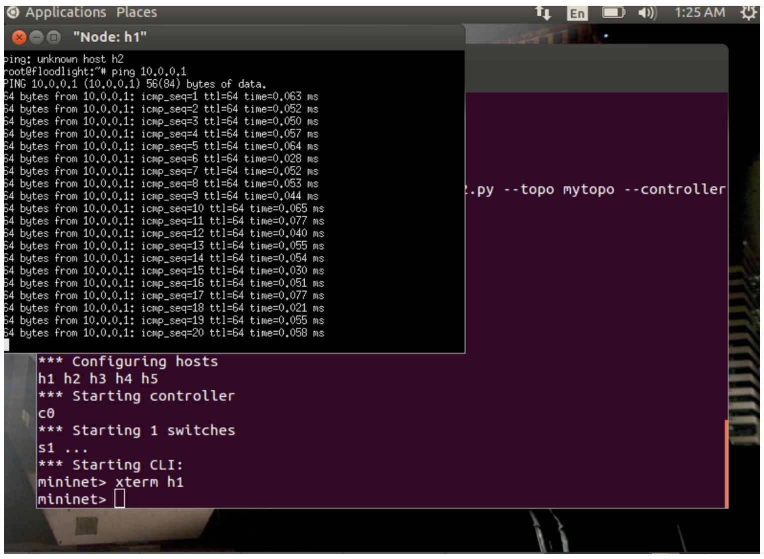


Hosts (5)

MAC Address	IP Address	Switch Port	Last Seen
de:59:3b:5f:62:61		00:00:00:00:00:00:00:01-2	11/18/2020, 4:39:16 PM
4e:f0:f9:c6:d1:77		00:00:00:00:00:00:00:01-5	11/18/2020, 4:39:16 PM
ba:7b:63:b3:96:48		00:00:00:00:00:00:00:01-3	11/18/2020, 4:39:16 PM
8e:57:28:cd:37:9e		00:00:00:00:00:00:00:01-4	11/18/2020, 4:39:17 PM
aa:c0:1c:94:1e:87	0.0.0.0	00:00:00:00:00:00:00:01-1	11/18/2020, 4:39:17 PM

Floodlight © Big Switch Networks, IBM, et. al. Powered by Backbone.js, Bootstrap, jQuery, D3.js, etc.

Figure 6: Working of Project



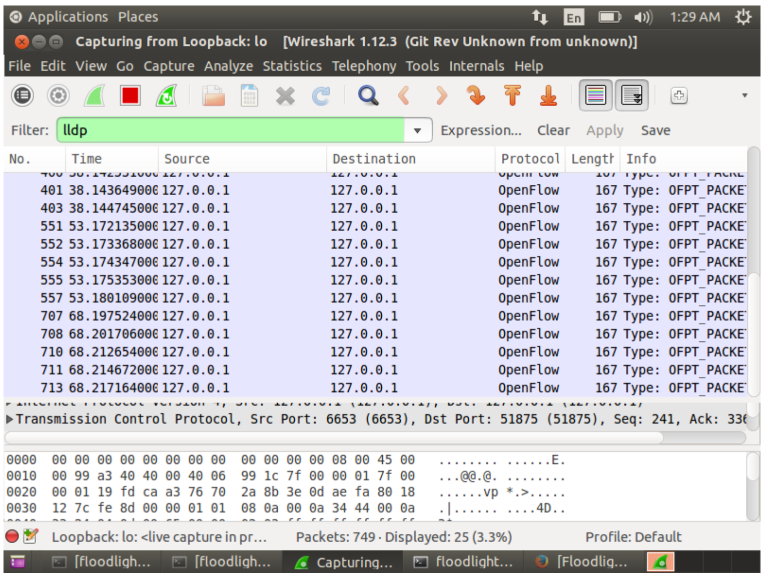
```
ping: unknown host h2
root@floodlight:~# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.063 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.052 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.050 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.057 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.064 ms
64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=0.028 ms
64 bytes from 10.0.0.1: icmp_seq=7 ttl=64 time=0.052 ms
64 bytes from 10.0.0.1: icmp_seq=8 ttl=64 time=0.053 ms
64 bytes from 10.0.0.1: icmp_seq=9 ttl=64 time=0.044 ms
64 bytes from 10.0.0.1: icmp_seq=10 ttl=64 time=0.065 ms
64 bytes from 10.0.0.1: icmp_seq=11 ttl=64 time=0.077 ms
64 bytes from 10.0.0.1: icmp_seq=12 ttl=64 time=0.040 ms
64 bytes from 10.0.0.1: icmp_seq=13 ttl=64 time=0.055 ms
64 bytes from 10.0.0.1: icmp_seq=14 ttl=64 time=0.054 ms
64 bytes from 10.0.0.1: icmp_seq=15 ttl=64 time=0.030 ms
64 bytes from 10.0.0.1: icmp_seq=16 ttl=64 time=0.051 ms
64 bytes from 10.0.0.1: icmp_seq=17 ttl=64 time=0.077 ms
64 bytes from 10.0.0.1: icmp_seq=18 ttl=64 time=0.021 ms
64 bytes from 10.0.0.1: icmp_seq=19 ttl=64 time=0.055 ms
64 bytes from 10.0.0.1: icmp_seq=20 ttl=64 time=0.058 ms
```

```
.py --topo mytopo --controller
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> xterm h1
mininet>
```

Figure 7: Working of Project

Since these packets will be forwarded by the associated access points out a port other than the port on which the packets were received, the access points will operate like normal OpenFlow-enabled switches. Each access point will forward the first ping packet it receives in each direction to the Mininet reference controller. The controller will set up flows on the access points to establish a connection between the stations h1 and h2.

The OF (OpenFlow) packets, which carry information about the designated flows, are captured through Wireshark in the controller machine.


Applications Places
En
1:29 AM
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Clear
Apply
Save

```
Capturing from Loopback: lo   [Wireshark 1.12.3 (Git Rev Unknown from unknown)]
Filter: lldp

No.  Time          Source      Destination  Protocol  Length Info
 401 38.14364900€ 127.0.0.1   127.0.0.1    OpenFlow    167 Type: OFPT_PACKE
 403 38.14474500€ 127.0.0.1   127.0.0.1    OpenFlow    167 Type: OFPT_PACKE
 551 53.17213500€ 127.0.0.1   127.0.0.1    OpenFlow    167 Type: OFPT_PACKE
 552 53.17336800€ 127.0.0.1   127.0.0.1    OpenFlow    167 Type: OFPT_PACKE
 554 53.17434700€ 127.0.0.1   127.0.0.1    OpenFlow    167 Type: OFPT_PACKE
 555 53.17535300€ 127.0.0.1   127.0.0.1    OpenFlow    167 Type: OFPT_PACKE
 557 53.18010900€ 127.0.0.1   127.0.0.1    OpenFlow    167 Type: OFPT_PACKE
 707 68.19752400€ 127.0.0.1   127.0.0.1    OpenFlow    167 Type: OFPT_PACKE
 708 68.20170600€ 127.0.0.1   127.0.0.1    OpenFlow    167 Type: OFPT_PACKE
 710 68.21265400€ 127.0.0.1   127.0.0.1    OpenFlow    167 Type: OFPT_PACKE
 711 68.21467200€ 127.0.0.1   127.0.0.1    OpenFlow    167 Type: OFPT_PACKE
 713 68.21716400€ 127.0.0.1   127.0.0.1    OpenFlow    167 Type: OFPT_PACKE

▶Transmission Control Protocol, Src Port: 6653 (6653), Dst Port: 51875 (51875), Seq: 241, Ack: 33€

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 99 a3 40 40 00 40 06  99 1c 7f 00 00 01 7f 00   ...@@.@. ........
0020  00 01 19 fd ca a3 76 70  2a 8b 3e 0d ae fa 80 18   ......vp *.>.....
0030  12 7c fe 8d 00 00 01 01  08 0a 00 0a 34 44 00 0a   .|...... ....4D..

Loopback: lo: <live capture in pr...   Packets: 749 · Displayed: 25 (3.3%)   Profile: Default
[floodligh...  [floodligh...  Capturing...  floodlight...  [Floodlig...
```

Figure 8: Working of Project

4.2.3 Final Topology

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
```

```
class MyTopo(Topo):

def __init__(self):
Topo.__init__(self)
s1=self.addSwitch('s1')
s2=self.addSwitch('s2')
s3=self.addSwitch('s3')
s4=self.addSwitch('s4')

n=8
for h in range(n):
host=self.addHost('h%s' % (h+1), cpu=.8/n)
if(h<3):
self.addLink(host, s1, bw=5, delay='5ms',  max_queue_size=1000)
elif(h<5):
self.addLink(host, s2, bw=5, delay='5ms',  max_queue_size=1000)
elif(h<7):
self.addLink(host, s3, bw=5, delay='5ms',  max_queue_size=1000)
elif(h<9):
self.addLink(host, s4, bw=5, delay='5ms',  max_queue_size=1000)
self.addLink(s1, s2, bw=10, delay='5ms', max_queue_size=1000)
self.addLink(s1, s3, bw=20, delay='5ms', max_queue_size=1000)
self.addLink(s4, s2, bw=15, delay='5ms', max_queue_size=1000)
self.addLink(s3, s4, bw=30, delay='5ms', max_queue_size=1000)


def perfTest():
topo=MyTopo()
net=Mininet(topo=topo, host=CPULimitedHost, link=TCLink)
net.start()
net.pingAll()
h1, h4=net.get('h1', 'h4')
net.iperf( (h1,h4) )
net.stop()


if __name__=='__main__':
setLogLevel('info')
perfTest
```

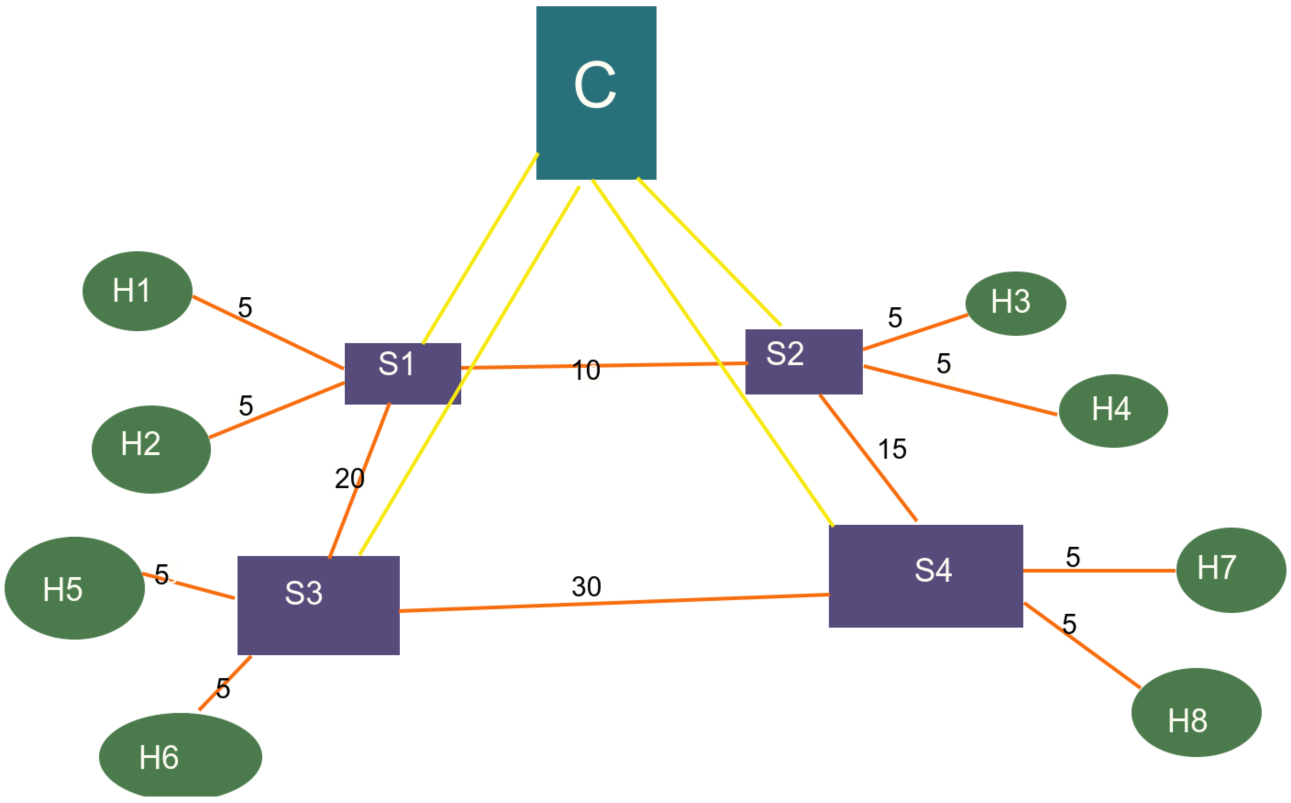


Figure 9: Topology used for study

There are 4 switches and 8 hosts being monitored by the OpenFlow Controller(C). The numerals mentioned are bandwidth, rest of the things we have not considered for our study. Again it is easy to extrapolate once we are done with this aspect.

4.2.4 A Detailed look at path discovery

After going through each module of huge FloodLight source code and knowing the work of each one we came across pushRoute() method in routing/forwardingBase.java. we started to search from where was the method being called. From now onwards we will sort of backtrack the method calls of our interest and know the actual procedure how route is decided by OpenFlow Controller.
• pushRoute() was called by a method doforwardflow() having its implementation written in forwarding/forwarding.java. some of its related parameters:
IOFSwitch sw - Input OpenFlow Switch
OFPacketIn pi - OpenFlow Packet that came in to switch sw from some host attached to it.
The method in itself checks various terms like: if source and destination are on same island, and many more.
it calls routingEngineService.getRoute()
• getRoute() of our interest has its roots in topology/topologyManager.java
It calls for getCurrentInstance(tunnel Enabled)
• we found definition to getCurrentInstance(tunnel Enabled) in same file topology/topologyManager.java.
It returns currentInstance variable.
• value returned from getCurrentInstance(tunnel Enabled) is assigned to nt which is an object of type TopologyInstance. nt calls its method compute().
• As discussed compute() is method of class TopologyInstance written in topology/topologyInstance.java.
The definition of compute() is the most important.
There are five steps and each time a method is called as follows:
Step 1-Compute clusters ignoring domain links. Create nodes for clusters in higher level topology.
identifyOpenDomain()
Step 2- Add links to clusters addLinksToOpenflowDomains()
Step 3- Compute shortest path trees in each clusters for unicast routing. The tree

are rooted at destination. Cost for tunnel links and direct links are the same. calculateBroadcastNodePortInClusters()
step 4- compute broadcast tree in each cluster. calculateBroadcastNodePortsInClusters()
step 5- print topology printTopology()

4.2.5 Collecting stats from network devices

Controller receives the information from all switches in the network and based on the received information, a controller can build the network topology. Then switches receive flow table from the controller to operate properly based on the flow table. If a switch receives a packet from a flow which does not have a matching entry in the flow table, then the switch transmits the packet to the controller. After receiving it, based on the packet information, the controller generates a forwarding rule and then sends the forwarding rule to the switch. A controller can add, update, and delete flow entries in flow tables, both reactively and proactively. we examine the traffic statistics collection features of OpenFlow protocol. Using the provided features we calculate the actual link utilizations and available link capacities in different time granularities in a test environment.
The two types of messages that are provided by OpenFlow are:
• STATISTICS REQUEST: Message sent from the controller to a switch requesting its current set of statistics for flows, ports, etc.
• STATISTICS REPLY: Message sent from a switch to the controller, in reply to a request message.
Often network measurements are performed on different layers. Measurements on the application layer are preferred to accurately measure application performance. Network layer measurements use infrastructure components (such as routers and switches) to obtain statistics. This approach is not considered sufficient for some as the measurement granularity is often limited to port based counters. It lacks the ability to differ between different applications and traffic flows. The Simple Network Management Protocol (SNMP) is one of the most used protocols to monitor network status.

our goal to get actual network usage information that can be used in different parts of the network management layer to provision the network online. Thus our aim is to get raw link usage information from the network devices independent of the application layer details. For each link on the network, we collect actual link usage in terms of number of bytes. We use this information to calculate near real-time available bandwidth on each link. This information will help us with deploying custom routing protocols.

4.2.6 Collecting stats using REST APIs

• The above described approach is to design a module of our own and then harness the services but floodlight has provided us with a number of REST APIs which can approached to get result in JSON format. From these key value pair it is very easy to find what all we need. Following are certain REST APIs which will serve us.
• Statistics API

```
/wm/statistics/bandwidth/<switchId>/<portId>/json
```

function: Fetch RX/TX bandwidth consumption
switchId: Valid Switch DPID as colon-delimited hex string or integer. Use "all" for all switches.
portId: Valid switch port number

4.2.7 Processing PACKET-IN and Detection of its type

The following steps are involved:
• Creating a module PacketTracker.java
• Running a simple topology example2.py on mininet

```
sudo mn --custom example2.py --topo mytopo --controller=remote,
port=6653 --switch ovsk
```

• Creating seperate xterm on mininet CLI

```
xterm h1 h2
```

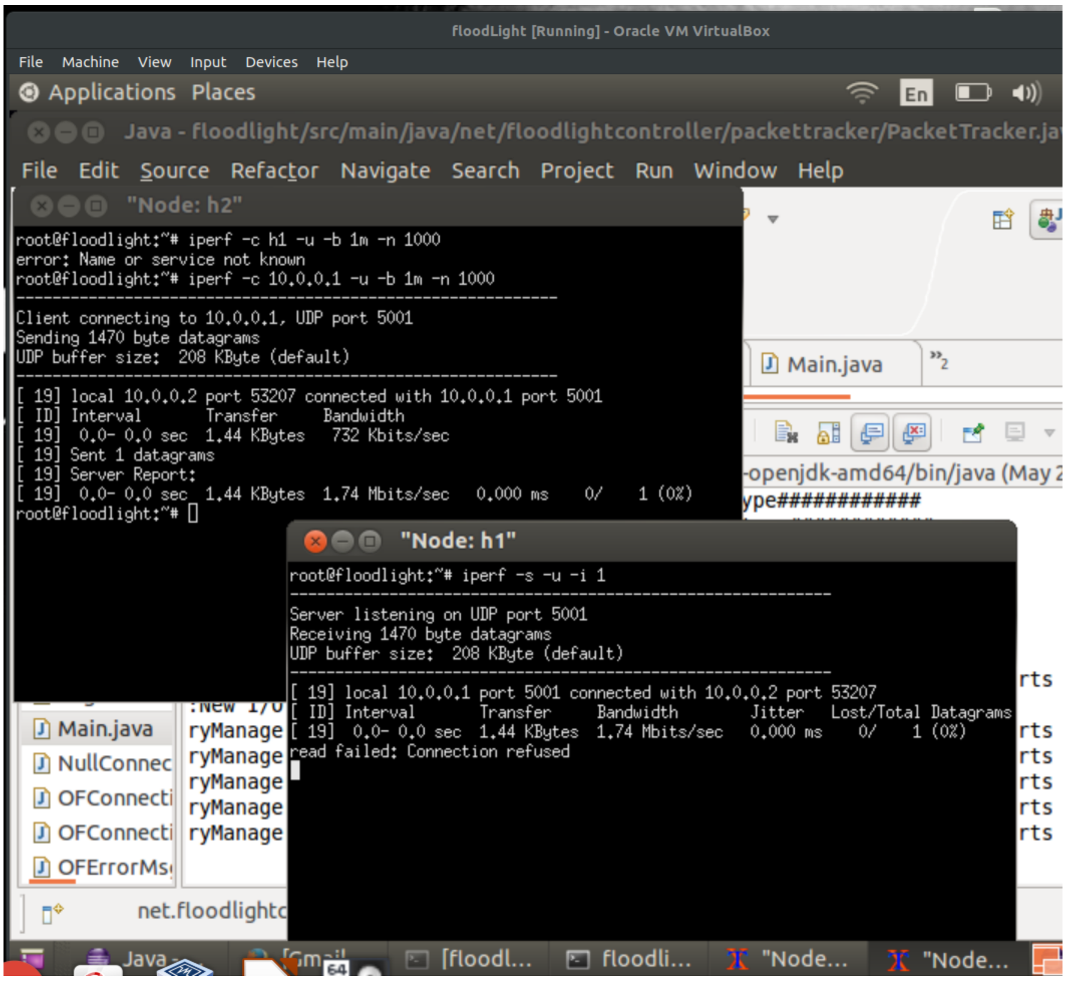
- Creating h1 as UDP server which sends packet at interval of 1 second

```
iperf -s -u -i 1
```

- Then in h2 run below command, which creates a udp client that connects to h1 at address 10.0.0.1, bandwidth = 1M, number of bytes to transport = 1000

```
iperf -c 10.0.0.1 -u -b 1m -n 1000
```

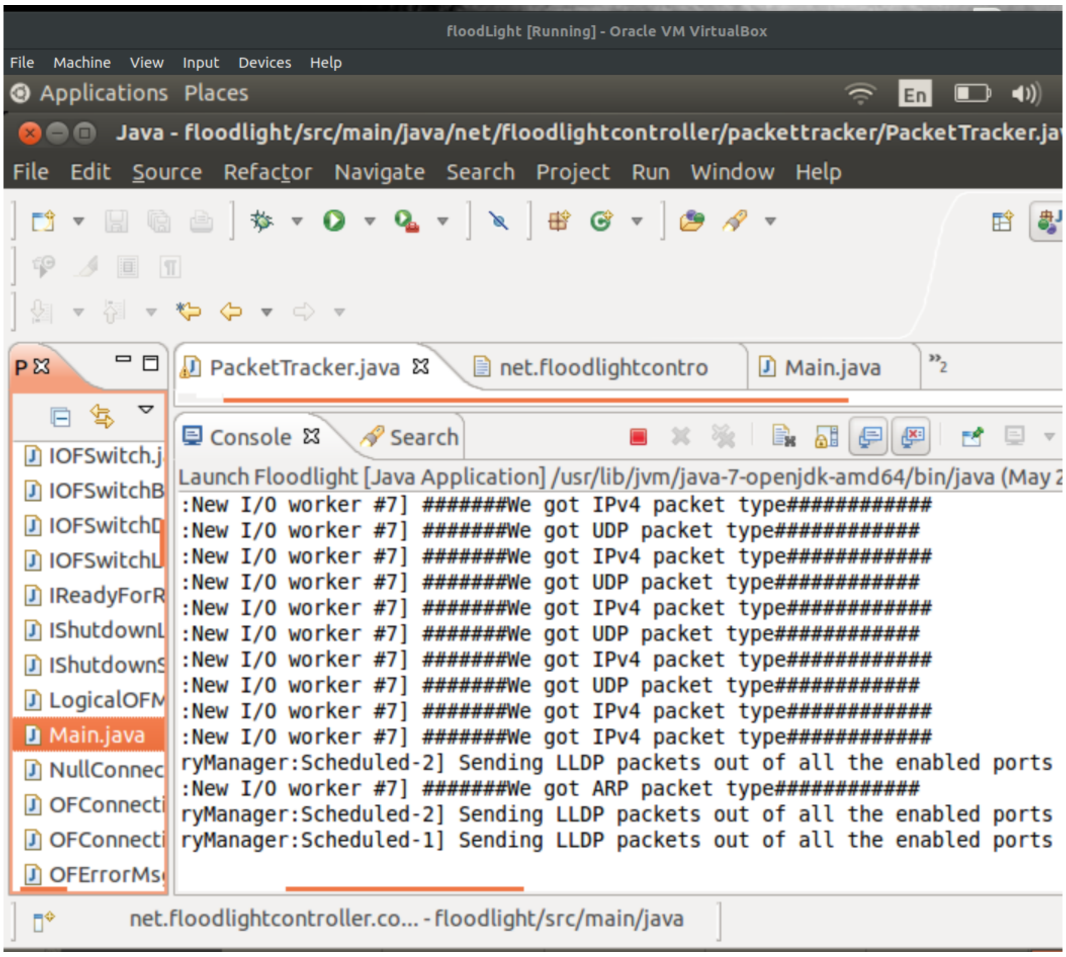
- Results we got


FloodLight [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places
Java - floodlight/src/main/java/net/floodlightcontroller/packettracker/PacketTracker.ja
File Edit Source Refactor Navigate Search Project Run Window Help
"Node: h2"
"Node: h1"
Main.java
NullConnec
OFConnecti
OFErrorMs
net.floodlightc

```
root@floodlight:~# iperf -c h1 -u -b 1m -n 1000
error: Name or service not known
root@floodlight:~# iperf -c 10.0.0.1 -u -b 1m -n 1000
------------------------------------------------------------
Client connecting to 10.0.0.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 19] local 10.0.0.2 port 53207 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 19]  0.0- 0.0 sec  1.44 KBytes   732 Kbits/sec
[ 19] Sent 1 datagrams
[ 19] Server Report:
[ 19]  0.0- 0.0 sec  1.44 KBytes  1.74 Mbits/sec   0.000 ms    0/    1 (0%)
root@floodlight:~#
```

```
root@floodlight:~# iperf -s -u -i 1
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 19] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 53207
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 19]  0.0- 0.0 sec  1.44 KBytes  1.74 Mbits/sec   0.000 ms    0/    1 (0%)
read failed: Connection refused
```

Figure 10: Xterm of both Hosts



floodLight [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places
Java - floodlight/src/main/java/net/floodlightcontroller/packettracker/PacketTracker.ja
File Edit Source Refactor Navigate Search Project Run Window Help
PacketTracker.java
net.floodlightcontro
Main.java
Console
Search
IOFSwitch.j
IOFSwitchB
IOFSwitchD
IOFSwitchL
IReadyForR
IShutdownL
IShutdownS
LogicalOFM
Main.java
NullConnec
OFConnecti
OFConnecti
OFErrorMs
net.floodlightcontroller.co... - floodlight/src/main/java

```
Launch Floodlight [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (May 2
:New I/O worker #7] #######We got IPv4 packet type############
:New I/O worker #7] #######We got UDP packet type############
:New I/O worker #7] #######We got IPv4 packet type############
:New I/O worker #7] #######We got UDP packet type############
:New I/O worker #7] #######We got IPv4 packet type############
:New I/O worker #7] #######We got UDP packet type############
:New I/O worker #7] #######We got IPv4 packet type############
:New I/O worker #7] #######We got UDP packet type############
:New I/O worker #7] #######We got IPv4 packet type############
:New I/O worker #7] #######We got IPv4 packet type############
ryManager:Scheduled-2] Sending LLDP packets out of all the enabled ports
:New I/O worker #7] #######We got ARP packet type############
ryManager:Scheduled-2] Sending LLDP packets out of all the enabled ports
ryManager:Scheduled-1] Sending LLDP packets out of all the enabled ports
```

Figure 11: UDP Packet Detected

4.2.8 Ordering above module before routing/ForwardingBase.java

• ForwardingBase.java is responsible for calculating the route everytime.
We have interface IOFMessageListener's method isCallbackOrderingPrereq(OFType type, String name)
which is responsible for telling which module will run before it.The code below mentioned enables this.

```
@Override
public boolean isCallbackOrderingPrereq(OFType type, String name) {
    return (type.equals(OFType.PACKET_IN)
        && (name.equals("topology") || name.equals("devicemanager")));
}
```

4.2.9 Creating PacketMataData class

• we create PacketMetaData.java class having static attribute PacketTypeIs to keep track of current packet type so that it can acessed anywhere.

4.2.10 Analysis of Flow Tables

• LLDP packet keeps on finding path every 15 second so as to keep updated if any changes are incurred in topology.
• It is better to keep this to 300 second, so that we can experiment properly in the meantime.
• set LLDP-TO-ALL-INTERVAL=3000 second in LinkDiscoveryManager.java.

4.2.11 Clearing Flow Table entries

• Since selective insertion of Flow Table Entries seems tough at this point of time, we go for another approach. Whenever a packet lets say UDP is received by our PacketTracker.java module we will clear all the Table entries.
• There are two ways to clear the entries:

1. Delete static entries from switch by using REST API manually on browser just before we are going to experiment.

```
http://127.0.0.1:8080/wm/staticflowpusher/clear/all/json
```

2.Clear Table Entries from all switches by somehow calling clearAllTables() from OFSwitchHandshakeHandler.java.

4.2.12 Topology will be recalculated

• Control goes to compute() method of TopologyManager.java.
• Everything is okay till we reach calculateShortestPathTreeInClusters() and Dijkstra is called.
• Conventionally we are using equal edge weights for calculating shortest path, But here we will modify edge weight according to Bandwidth statistics.

4.2.13 Bandwidth stats from REST API

• We can actually get Bandwidth usage between ports of two switches.

```
/wm/statistics/bandwidth/<switchId>/<portId>/json
```

function: Fetch RX/TX bandwidth consumption
switchId: Valid Switch DPID as colon-delimited hex string or integer. Use "all" for all switches.
portId: Valid switch port number
• We can make a module which calls such REST APIs and provide us with all data we need.

4.2.14 Algorithm for widest path instead of shortest path

Psudo Code

```
Initialize bandwidth as an array of size | V | | V | to be INT_MAX
Initialize next as an array of size | V | | V | to be null
function FloydWarshall_AllPairs_WidestPathProblem (G(V, E))
```

```
for each vertex v in V do
    bandwidth[v][v] ← 0
end for
for each edge (u, v) in E do
    bandwidth[u][v] ← w(u, v)
end for
for k from 1 to | V | do
    for i from 1 to | V | do
        for j from 1 to | V | do
            if min(bandwidth[i][k], bandwidth[k][j]) > bandwidth[i][j] then
                bandwidth[i][j] ← min(bandwidth[i][k], bandwidth[k][j])
                next[i][j] ← k
            end if
        end for
    end for
end for
end function
function Trace Path(i, j)
    if bandwidth[i][j] == INT_MAX then
        return \There does not exists any path from i to j"
    end if
    k ← next[i][j]
    if k == null then
        return \ "
    else
        return Trace Path(i, j) + k + Trace Path(k, j)
    end if
end function
```

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Software Defined Network is the topic under limelight now a days. It was important to first go through openFlow protocol on which FloodLight controller is based. Since it is tough and expensive to study everything through hardware implementation, so Mininet Emulator was important to be understood. The LLDP packets were traced through wireshark to give insight to functioning of FloodLight controller routing mechanism.
Path Optimization is something which can reduce Network cost for sure.
It will also make optimal utilisation of all links we have.
The packet loss due to queue overflow is also reduced.
Choosing Widest path specially for UDP Packet is just an example, we can extend this idea to lets say finding a path which experience minimum delay or say packet loss.

5.2 Future Work

FloodLight controller is open-source project we are expected to contribute and hence become part of community. Regarding this project, we can implement the proposed part like clearing the flow tables. We can also find some alternative algorithm for modification of path as discussed in conclusion part.

References

[1]How to work with fast failover *https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/*
Failover+OpenFlow+Groups

[2]How to create a packet out message *https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/*

[3]How to write a module *https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/13435*

[4]Mininet Miniedit GUI *http://www.brianlinkletter.com/how-to-use-miniedit-*
mininets-graphical-user-interface/

[5]Which packet does ping command use *https://www.google.com/search?q=which+packet+do+ping+c*
8

[6]Python API in mininet to send UDP Packet *https://stackoverflow.com/questions/35869439/how-*
can-i-use-mininet-python-api-to-send-udp-packets-from-one-host-to-another

[7]Source Code Analysis of every module of floodlight *https://programmersought.com/article/60914041*

[8]REST API in floodlight *https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/13435*

[9]Software Defined Networks Reactive Flow Programming and Load Balance
switching *https://dione.lib.unipi.gr/xmlui/bitstream/handle/unipi/10680/Kallianiotis*$_N$*ikolaos.pdf?seq*
1$isAllowed = y$[10]$Experimenting with scalability of floodlight https://www.researchgate.net/publicatio$