OpenFlow based dynamic load balanced switching

Project Report

Presented by,
Gunjan Patel (836122)
Adithi S Athreya (1008502)
Swetha Erukulla (1060536)
Team 5
COEN233 – Fall2013

Table of Contents

Introduction

Objective

Emulate a Software Defined Network (SDN) with emerging technologies like OpenFlow and SDN using tools Mininet and Open Daylight Controller.

What is the problem?

There is need for dynamic management of network resources for high performance and low latency of data transmission in a network.

Why is this project related to this class?

SDN with OpenFlow is an emerging technology in networking, and with this project it is like cherry on the cake, cake being the traditional networking concepts of this class.

Why other approach is not good & why do we think our approach is better?

Traditional network of TCP or UDP, use static switches, i.e. load balancing across paths are based on hash calculations of packets. Issue with this approach is that each packet of such a flow follows the single pre-defined path through the network. In case of discrepancy in the path, like a switch breakdown or physical layer damage, packets tend to drop or the other switches need to be manually configured for choosing a different path. **This becomes a cumbersome task, as the network grows.** Also, disadvantage of hashing is that all links gets the same percentage of hash values or to say, all paths have the same capacity (ECMP - Equal Cost MultiPath). Even if the network is hard coded to work in as multipath network, because of the equal capacity issue, efficient load balancing might not be achieved.

An alternative solution for this issue is Software Defined Networking (SDN). SDN is a concept where a central controller makes the decision for packet traversal and not the switches. Controller dynamically detects the topology by listening to the switches and calculates available path with less load. Controller then directs the switches with forwarding entries needed for the paths thus efficiently balancing the load with every flow. Although SDN is a bit slower than traditional networking, there are added advantages of its own.

- Consideration of end-to-end path
- Congestion control
- Dynamic adaptation to topological changes

Statement of the problem

Is SDN better than traditional networking? If so, how?

Area or scope of investigation

- Initially, a network topology is to be decided and we decided to be fat-tree topology.
- To emulate this topology, we have used mininet, which is an open source network virtualization emulator.
- We have used another open source software called Open Daylight as OpenFlow controller.
- Installation and setup of mininet and Open Daylight, while we understand the software is our initial challenge.
- Connection between mininet and Open Daylight can be established and shown in GUI using eclipse API.
- We have written scripts to emulate the mininet for our requirement.
- Design an algorithm for Open Daylight API and considering to have an end-to-end controlled network.
- To generate traffic in network we have used iperf and by this we measured bandwidth and quality of network link like latency time and performance.
- After traffic generation, our main motto is to monitor the flows, with respect to port statistics and schedule new flows.
- Finally, we would like to compare dynamic load balancing (DLB) with static load balancing (SLB)
- Flows in our project are L2, future scope can be to extend load balancing across routers as well.

Theoretical bases and literature review

Definition of the problem

Data center networks are designed for satisfying the data transmission demand of densely interconnected hosts in the data center. The network topology and switching/routing mechanism can affect the performance and latency significantly. Nowadays, the fat-tree network is one of the most widely used topologies for data center networks. Network engineers also adopt load balancing methods in the design of switching and routing algorithms. However, the requirement of load balancing in fat-tree networks cannot be fully satisfied by traditional approaches. The main reason is the lack of efficient ways to obtain network traffic statistics from each network device. As a solution, the OpenFlow protocol enables monitoring traffic statistics by a centralized controller.

To achieve high performance and low latency, we propose a load balancer for OpenFlow based data center networks. We would like to implement a dynamic traversal algorithm in the load balancer. The task of the algorithm is to distribute traffic of upcoming and incoming network flows and make each alternative path receive equal amounts of traffic load. It can be further applied to large scale networks and schedule data flows dynamically. Our plan for implementation is to use the OpenFlow controller OpenDaylight and network emulator Mininet. We would also like to demonstrate that our dynamic load balancing routing algorithm is superior over the static load balancing algorithm.

Theoretical background of the problem

- **Fat-Tree Network Topology**

  To start with, any network makes sense only if it is efficiently connected, so that, all end nodes can communicate to all other end nodes. In a switched fabric — a network topology that uses switches — the main goal is to connect a large number of endpoints (processors or servers) by using switches that only have a limited number of ports. By cleverly connecting switching elements and forming a topology, a network can interconnect an impressive amount of endpoints.

  Fat-Tree networks were proposed by Charles E. Leiserson in 1985. Such network is a tree, and processors are connected to the bottom layer. The distinctive feature of a fat-tree is that for any switch, the number of links going down to its siblings is equal to the number of links going up to its parent in the upper level. Therefore, the links get “fatter” towards the top of the tree, and switch in the root of the tree has most links compared to any other switch below it as seen in Fig.1.

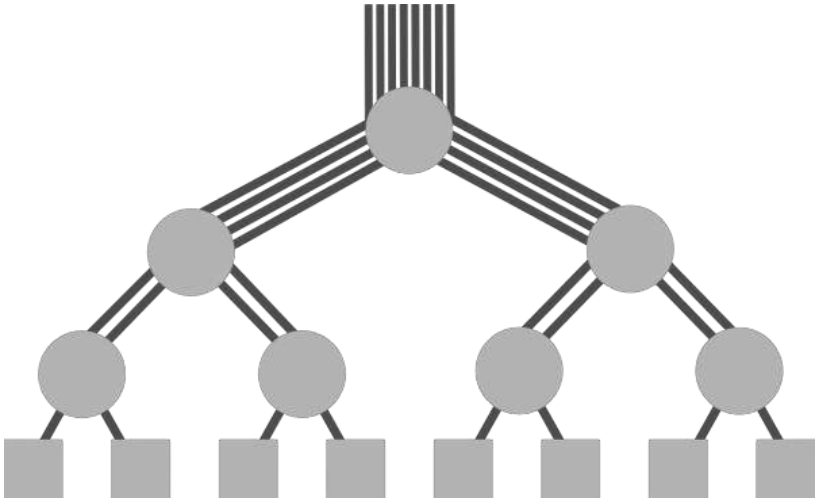


Fig.1 *: Fat-Tree. Circles represent switches, and squares at the bottom are endpoints*

- **Mininet - Network topology emulator**

  Mininet is a network emulator. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. A Mininet host behaves just like a real machine; you can ssh into it (if you start up sshd and bridge the network to your host) and run arbitrary programs. The programs you run can send packets through what seems like a real Ethernet interface, with a given link speed and delay. Packets get processed by what looks like a real Ethernet switch, router, or middlebox, with a given amount of queuing.

  Why MININET?

- **Custom topologies**: a single switch, huge Internet-like topologies, the Stanford backbone, a data center, or anything else can be create
- **Can run real programs**: anything that runs on Linux is available to run, from web servers, to TCP window monitoring tools, to Wireshark.
- **Customize packet forwarding:** Mininet switches are programmable using the OpenFlow protocol
- **Can share and replicate results**: anyone with a computer can run any other code once they packaged it up.
- **You can use it easily:** you can create and run Mininet experiments by writing simple (or complex if necessary) Python scripts.

Compared to simulators, Mininet runs real, unmodified code including application code, OS kernel code, and control plane code (both OpenFlow controller code and Open vSwitch code) and easily connects to real networks.

Limitations of mininet:
Mininet-based networks cannot (currently) exceed the CPU or bandwidth available on a single server. It cannot run non-Linux-compatible OpenFlow switches or applications; this has not been a major issue in practice.

- **Software Defined networking(SDN)**

The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices, is what called a Software Defined networking (SDN).

SDN is an approach to computer networking which evolved from work done at UC Berkeley and Stanford University around 2008. It 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 system that makes decisions about where traffic is sent (the control plane) from the underlying systems that forwards traffic to the selected destination (the data plane), thus enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow protocol is a foundational element for building SDN solutions. A typical SDN configuration can be thought of as shown in the Fig.2.

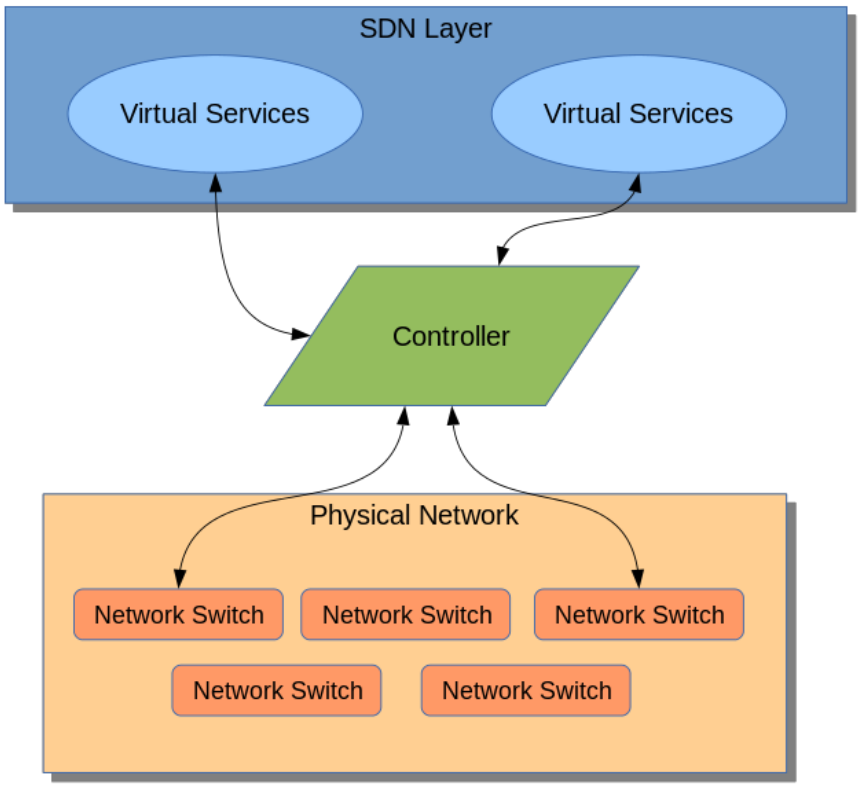


***Fig.2** : The controller acts as an interface between the physical network and the SDN layer*

Advantages of SDN

- **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.

Even though SDN seems to the perfect solution to today's huge data networking problems, it has **disadvantages** of its own. As we are moving from traditional distributed networking to centralized approach, there is a lot of load on the controller itself. Any packet losses between the controller and switches have bigger consequences, unlike in traditional approach, where one single packet loss doesn't affect the performance so much. Making the whole SDN architecture modular and scalable is a network design challenge when there are dense networks. Changing network architecture from the current distributed model to the SDN model, while feasible within a datacenter is not possible beyond organizational boundaries.

- **OpenFlow - A type of SDN mechanism**

An SDN architecture has two distinct networking APIs: northbound and southbound. OpenFlow is a southbound API, which allows the path of network packets through the network of switches to be determined by software running on multiple routers.

How does Open Flow work?

In a classical router or switch, the fast packet forwarding (data path) and the high level routing decisions (control path) occur on the same device. An OpenFlow Switch separates these two functions. The data path portion still resides on the switch, while high-level routing decisions are moved to a separate controller, typically a standard server. The OpenFlow Switch and Controller communicate via the OpenFlow protocol, which defines messages, such as packet-received, send-packet-out, modify-forwarding-table, and get-stats.
The data path of an OpenFlow Switch presents a clean flow table abstraction; each flow table entry contains a set of packet fields to match, and an action (such as send-out-port, modify-field, or drop). When an OpenFlow Switch receives a packet it has never seen before, for which it has no matching flow entries, it sends this packet to the controller. The controller then makes a decision on how to handle this packet. It can drop the packet, or it can add a flow entry directing the switch on how to forward similar packets in the future. Fig.3 shown below is an idealized open flow switch. Flow table is controlled by a remote controller via secure channel.

OpenFlow message structure:
OpenFlow has a predefined message structure. These messages are called OF messages that are passed between the controller and the OpenFlow switch. Every switch will have it's own Flow Table as opposite to MAC address table and routing table in the traditional switches. Every new incoming frame/packet is matched against the existing Flow table on the switch and take the necessary action specified in the 'Action' field of the Flow table. You can see the OF message structure in Fig.4 below.

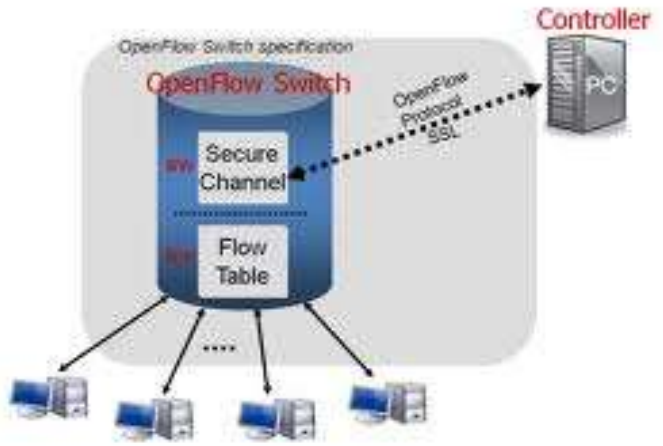


***Fig.3** : idealized open flow switch*

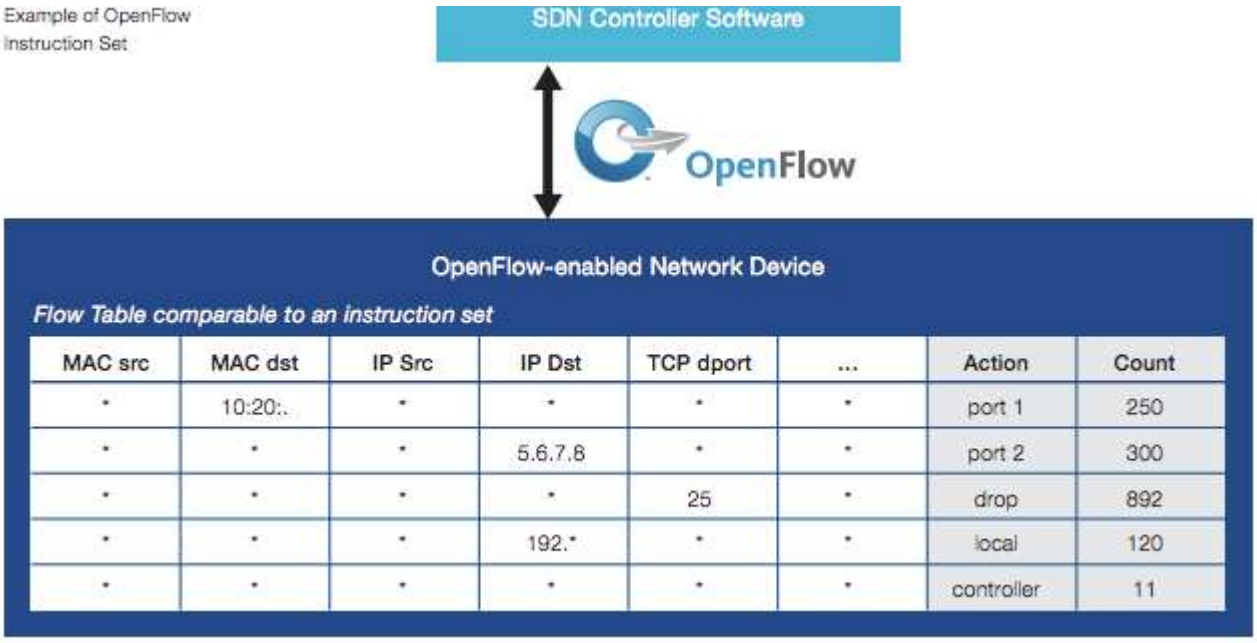


***Fig.4** : OF message structure*

- **OpenDaylight Controller**

  OpenDaylight is an open source project with a modular, pluggable, and flexible controller platform at its core. This OpenFlow controller is implemented strictly in software and is contained within its own Java Virtual Machine (JVM). As such, it can be deployed on any hardware and operating system platform that supports Java.

  OpenDaylight is a community to promote and/or propose standardization of SDN northbound APIs, so that services that use an open flow controller can be written quickly and effectively. This controller is based on OSGi (Open Serices Gateway initiative) framework and it exposes REST (REpresentational State Transfer - a web based) API.

  The controller platform itself contains a collection of dynamically pluggable modules to perform needed network tasks. There are a series of base network services for such tasks as understanding what devices are contained within the network and the capabilities of each, statistics gathering, etc. In addition, platform oriented services and other extensions can also be inserted into the controller platform for enhanced SDN functionality.
  The southbound interface is capable of supporting multiple protocols (as separate plugins), e.g. OpenFlow 1.0, OpenFlow 1.3, BGP-LS, etc. These modules are dynamically linked into a Service Abstraction Layer (SAL). The SAL exposes device services to which the modules north of it are written. The SAL determines how to fulfill the requested service irrespective of the underlying protocol used between the controller and the network devices.

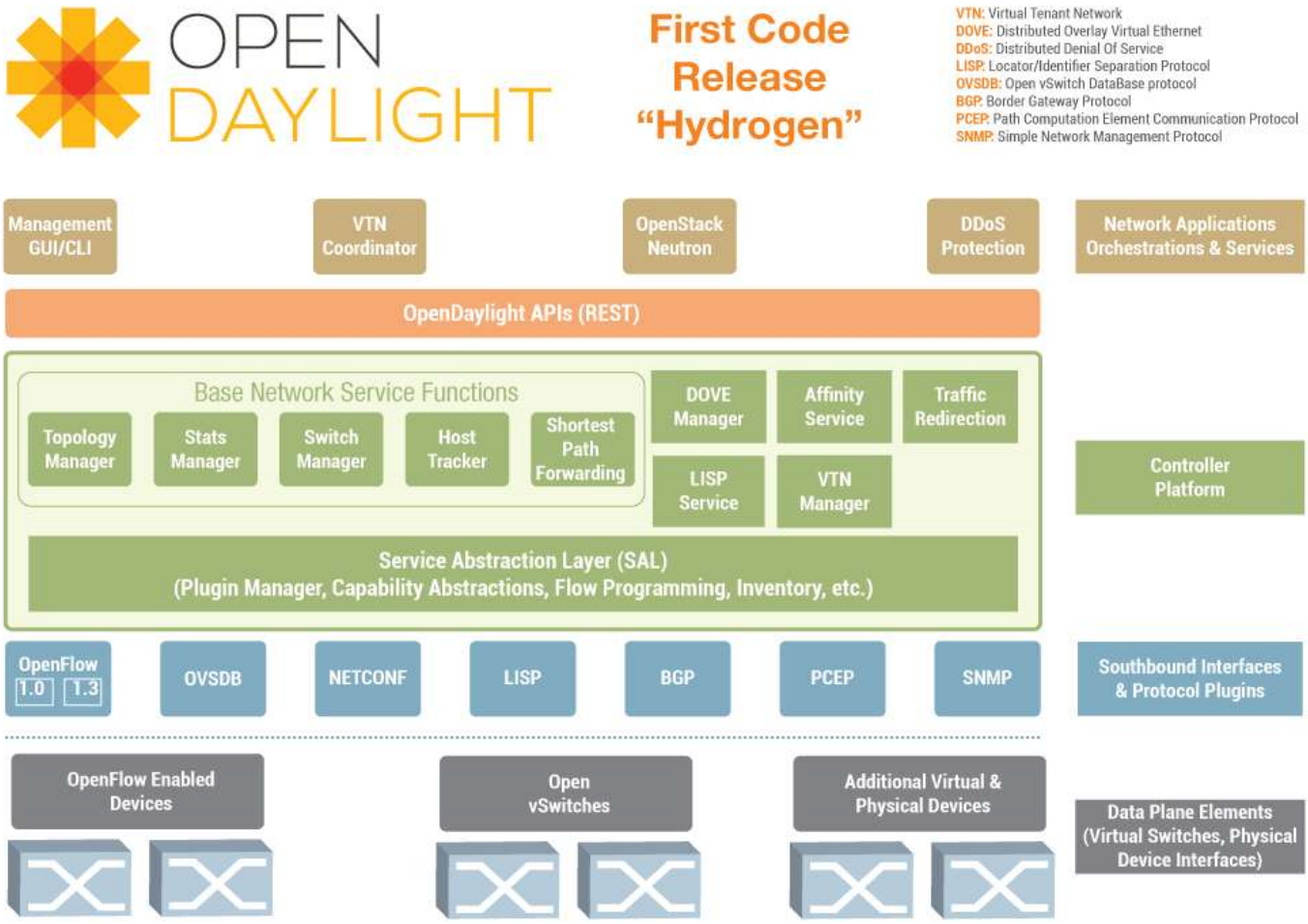


Fig.5 *: Layers in SDN with OpenDaylight*

- **IPERF :**
  - Iperf is a commonly used network testing tool that can create TCP and UDP data streams and measure the throughput of a network that is carrying them.
  - It can measure the bandwidth and the quality of a network link.
  - Iperf has a client and server functionality, and can measure the throughput between the two ends, either unidirectionally or bidirectionally as shown in Fig 6. DUT represents Device Under Test.
  - It is open source software and runs on various platforms including Linux, UNIX and Windows.

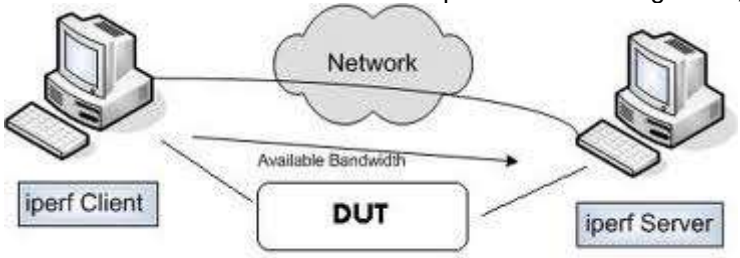


Fig.6 *: bandwidth measurement by iperf*

Related research and Our solution to solve the problem:

Our main inspiration for this project was from a paper “OpenFlow based Load Balancing for Fat-Tree Networks with Multipath Support” by Yu Li and Deng Pan. But this paper had some issues like not handling 2-way traffic control (upcoming and incoming). So, our approach to it with this issue taken into consideration, lead us in understanding the importance of controllers. Our research then moved to different types of controllers like Beacon (developed by Stanford), OpenDaylight (Open source controller) and FloodLight (developed by BigSwitch). Upon comparing these controllers [13], we came to a conclusion of using OpenDaylight, as it is more advanced and newer than others and is the most vastly used one. Also, OpenDaylight, which is backed by industry giants such as Cisco, Juniper, Brocade, IBM, etc, and follows a controller model that, in addition to OpenFlow, alternative south-bound protocols can be introduced.

Advantage/Disadvantage of those research

As explained in the previous section, major advantage of this project is that it is SDN. Centralized monitoring and controlling, quicker adaptation to changes in topology, visibility of end-to-end path for any packet, are the primal factors that lead our research. OpenFlow mechanism remains the fundamental element for SDN. Mininet is still a good open source emulator. Beacon was an older, less used type of controller.

Where our solution differs from others & why our solution is better?

As we propose to emulate OpenDaylight controller with mininet for a 2-way (uplink and downlink) control, our response will be definitely different from traditional approach but also would be better with respect to only upward control approach as in the main reference paper [1].

Hypothesis (or goals)

We would like to propose the following hypotheses for this project.

- We should get better overall throughput with dynamic switching which adapts to any network topology change in real-time without any user intervention
- Performance should be better and down time should be minimal as compared to traditional approach.
- The evaluation results should demonstrate that dynamic load balancing routing algorithm is better in data rate over static load balancing algorithm.

Methodology

How to solve the problem

Algorithm design

- For OpenDaylight with mininet

- Listens to every switch and link set-up in mininet, and establishes flow-paths between every hosts.
- Switches links are individually analyzed for current utilization
- Calculates total path utilization for all the possible paths between the source and destination host
- Centralized controller then directs the switches and links with forwarding entries, based on the calculated results from the algorithm

Control Loop *{*

Floyd Warshall

Adjust links

Run SPF for affected s-d pairs

}

Stats monitoring loop *{*

Read statistics on all nodes.

Monitor link utilization as a metric of total bandwidth of the link (very easy to do)

Trigger Control Loop if link utilization of any one link is above threshold

}

Tools and languages used

In this project, we propose to use the fat-tree topology, which contains multiple paths among hosts so it can provide higher available bandwidth than a single-path tree with the same number of nodes. It is typically a 3-layer hierarchical tree that consists of switches on the core, aggregation and top-of-rack (ToR) layers. The hosts connect to the switches on the ToR layer. The multipath feature of fat-tree networks enables chances to distribute data traffic on different network components. It is a practical task to achieve load balancing to help schedule traffic in fat-tree networks.

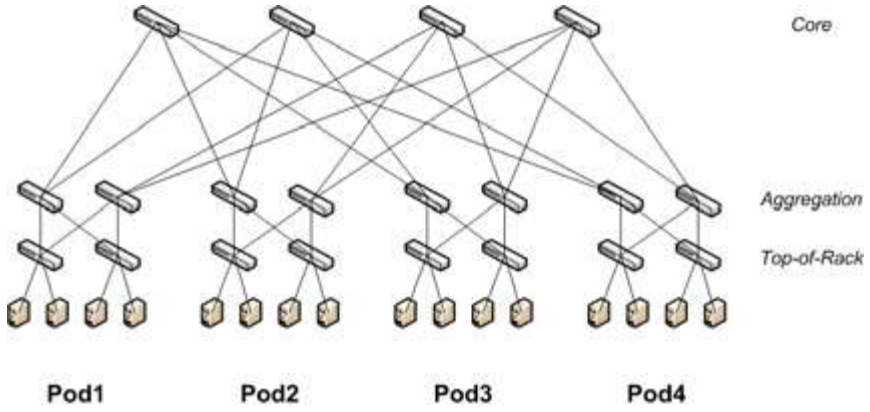


Fig.7 *: A k=4 3-layer fat-tree network*

Fat-tree topology is emulated using a tool called Mininet. Mininet runs on Linux kernel and we have used **PYTHON** in mininet to define our topology.

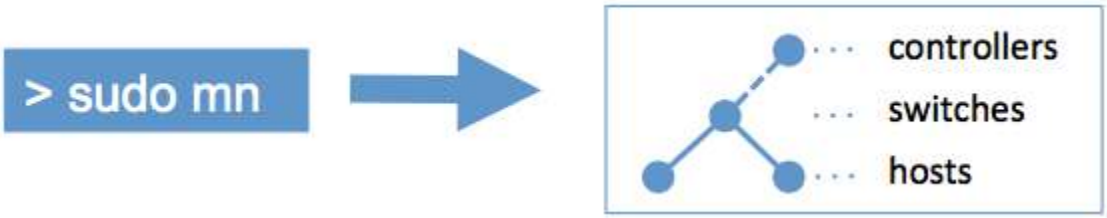


***Fig.8** : Depiction of emulation in mininet*

Then, we have used OpenDaylight controller, to connect, control and monitor network set-up in mininet. **Java API**s are used in OpenDaylight to configure the controller.

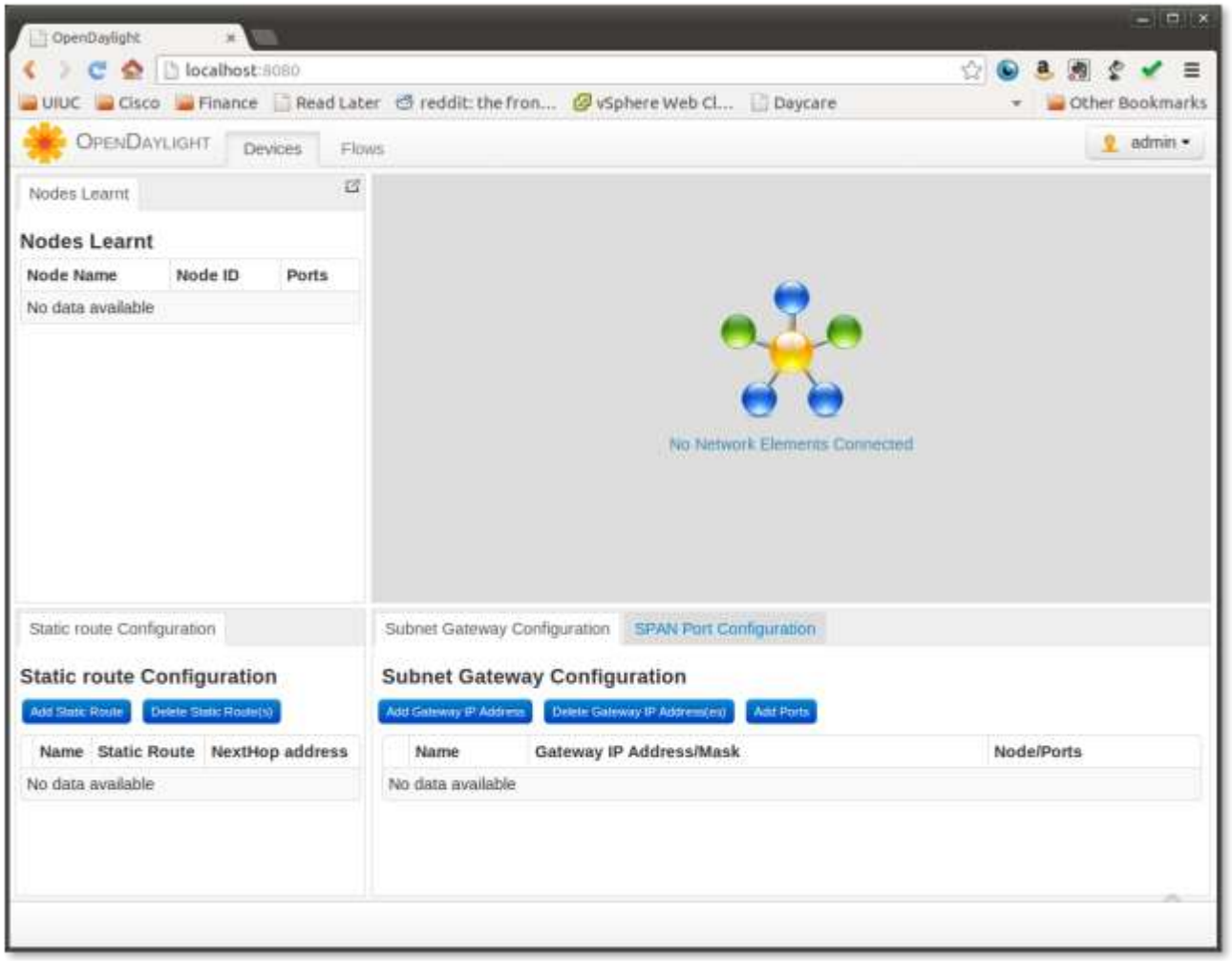


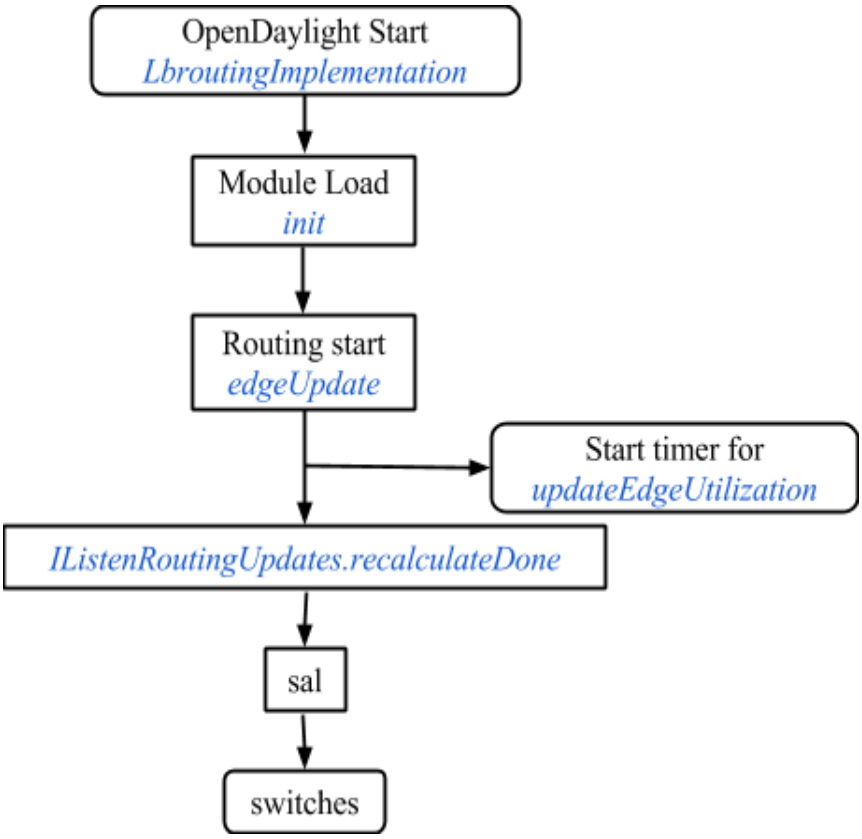
***Fig.8** : Java API for OpenDaylight*

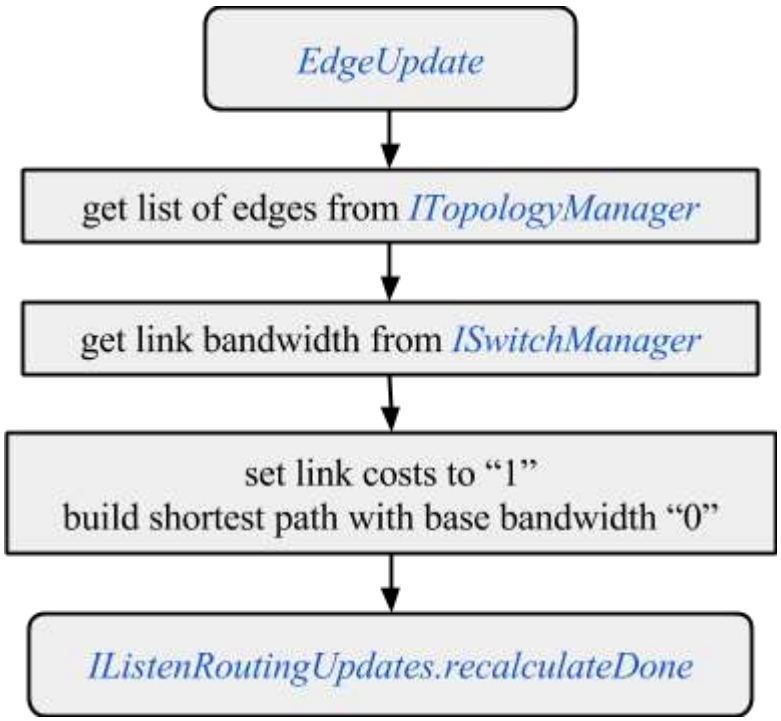
How to generate/collect input and output data

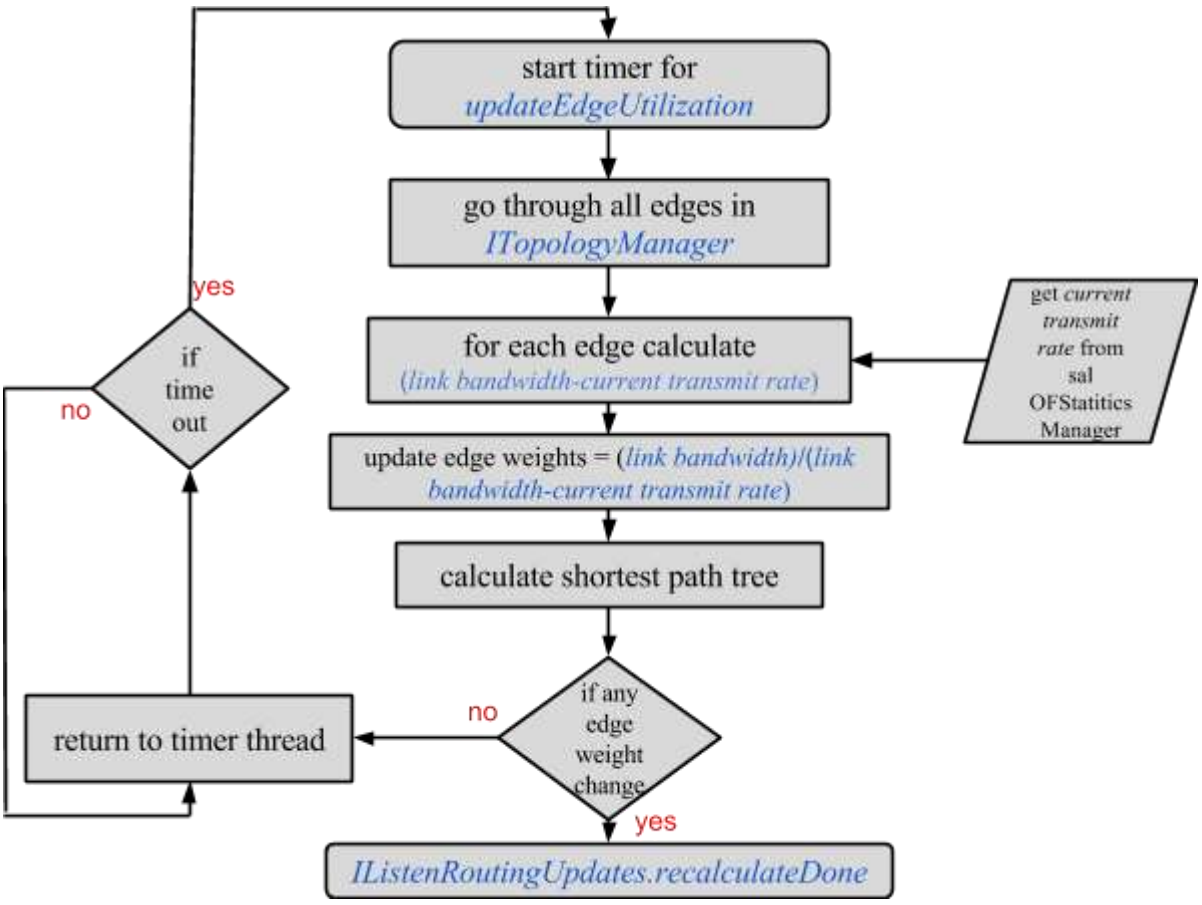
- Iperf generates random traffic i.e., a host sends packets to any other host in the network with uniform probability, in virtual networks.
- Then Iperf clients transmit data flows to Iperf servers.
- The performance can be measured and collected by Open Daylight controller.

Implementation

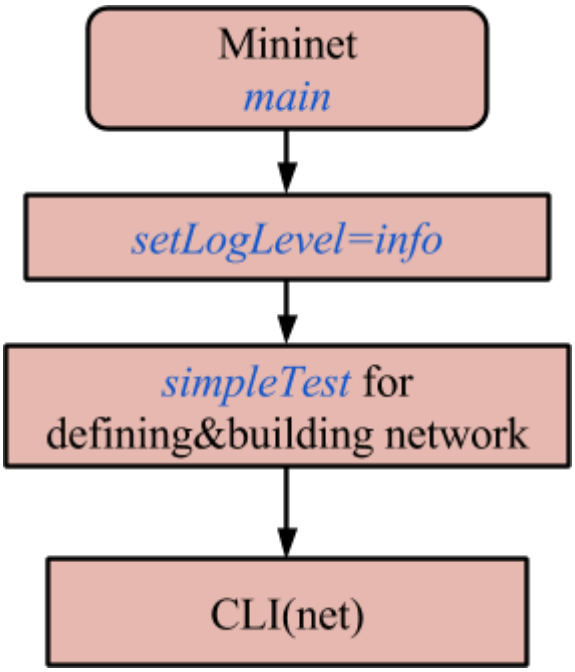
Flow chart for ODL

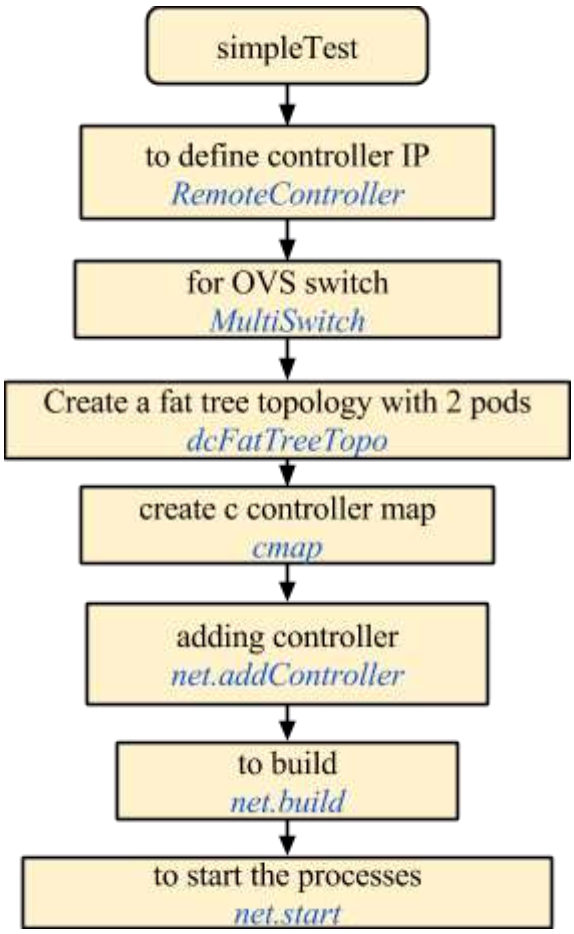


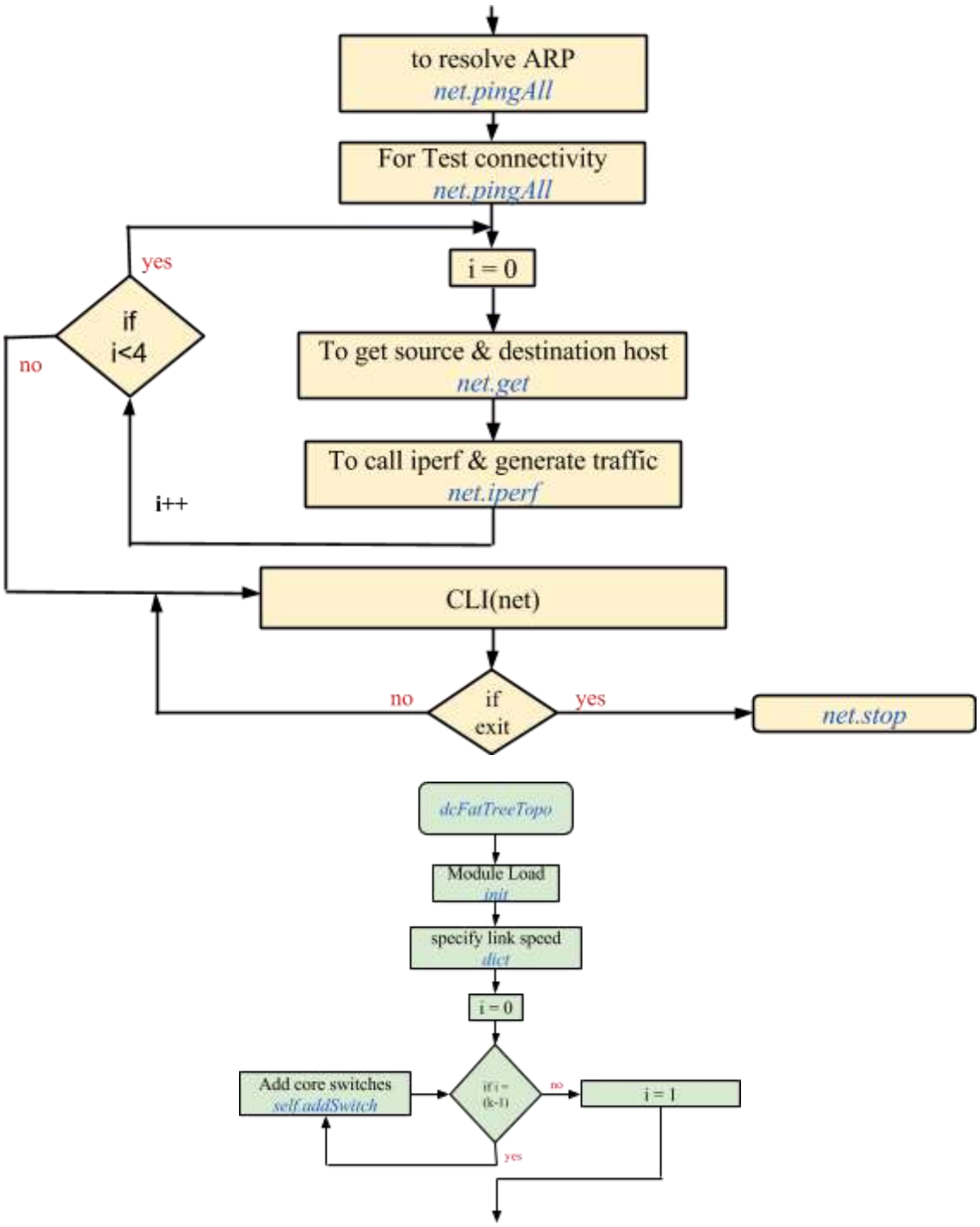

EdgeUpdate
get list of edges from ITopologyManager
get link bandwidth from ISwitchManager
set link costs to "1"
build shortest path with base bandwidth "0"
IListenRoutingUpdates.recalculateDone

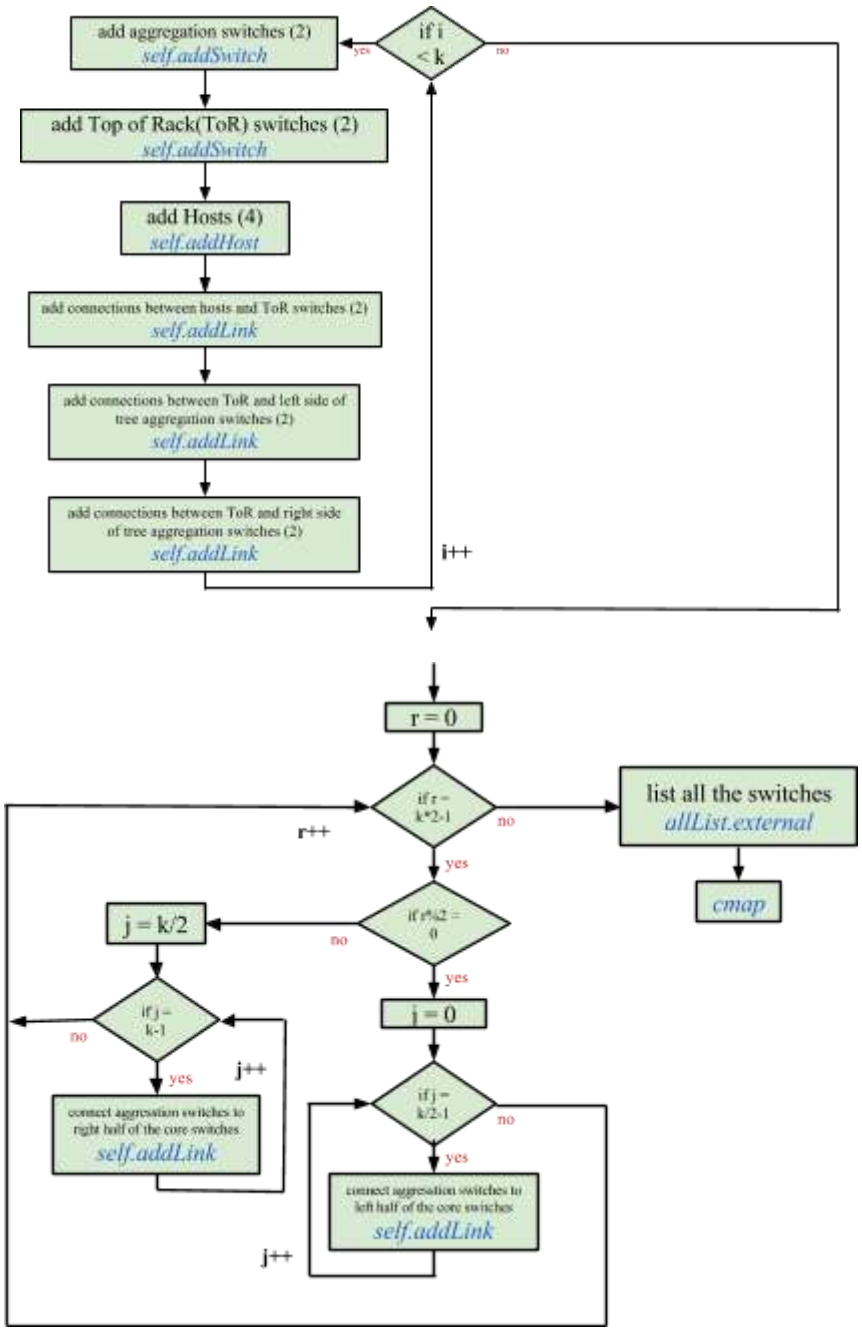


Flow Chart for Mininet




simpleTest
to define controller IP
RemoteController
for OVS switch
MultiSwitch
Create a fat tree topology with 2 pods
dcFatTreeTopo
create c controller map
cmap
adding controller
net.addController
to build
net.build
to start the processes
net.start


to resolve ARP
net.pingAll
For Test connectivity
net.pingAll
i = 0
yes
if
i<4
no
To get source & destination host
net.get
To call iperf & generate traffic
net.iperf
i++
CLI(net)
no
if
exit
yes
net.stop
dcFatTreeTopo
Module Load
init
specify link speed
dict
i = 0
Add core switches
self.addSwitch
if i =
(k-1)
no
i = 1
yes

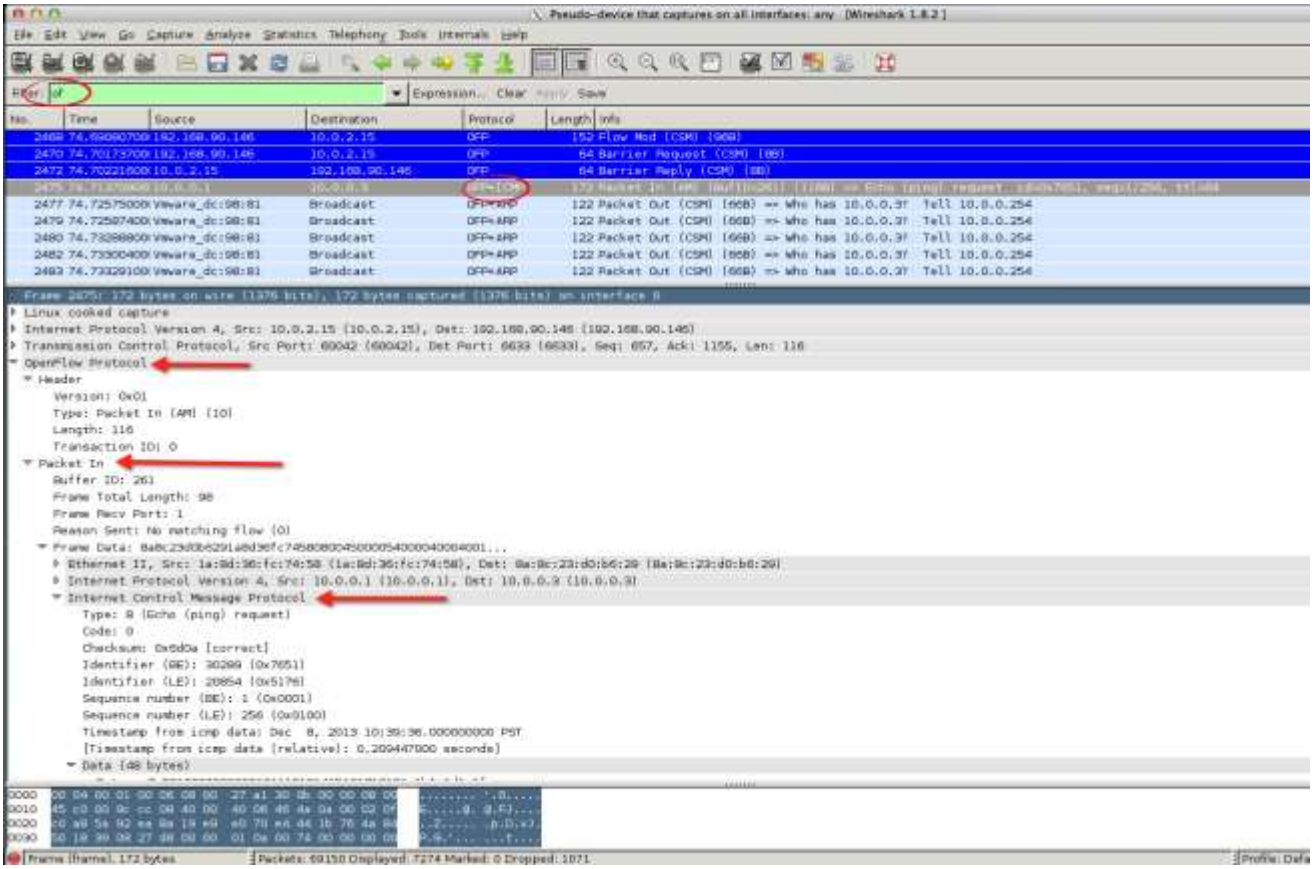


Data Analysis and discussion

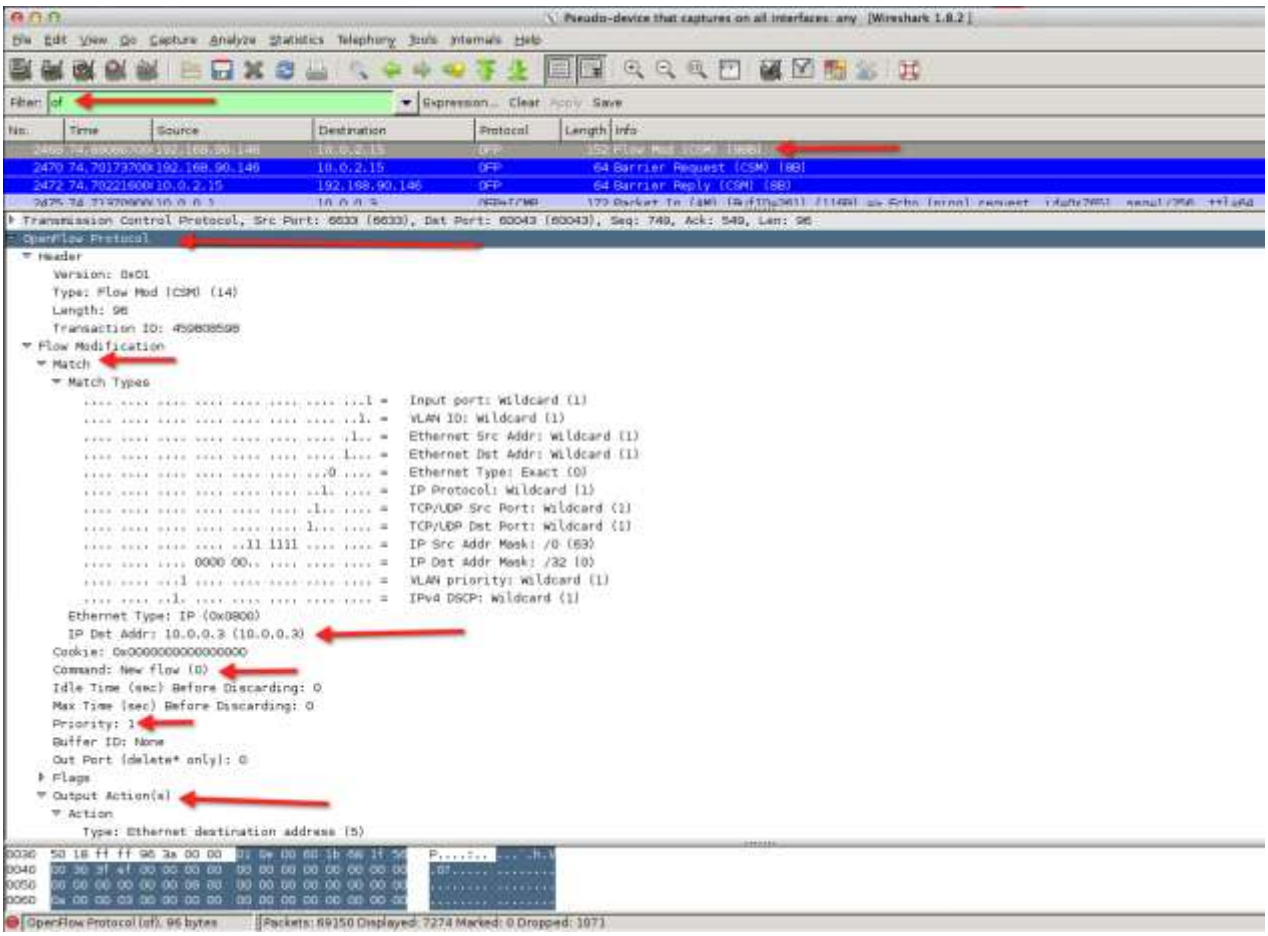
Output Generation & Analysis :

We are using iperf to generate TCP traffic to test the bandwidth of end to end connection.As we can see from the wireshark capture screen shot below,first ICMP

packet will be encapsulated with OFP(Open flow protocol) and will be sent to the controller as a PACKET_IN packet.

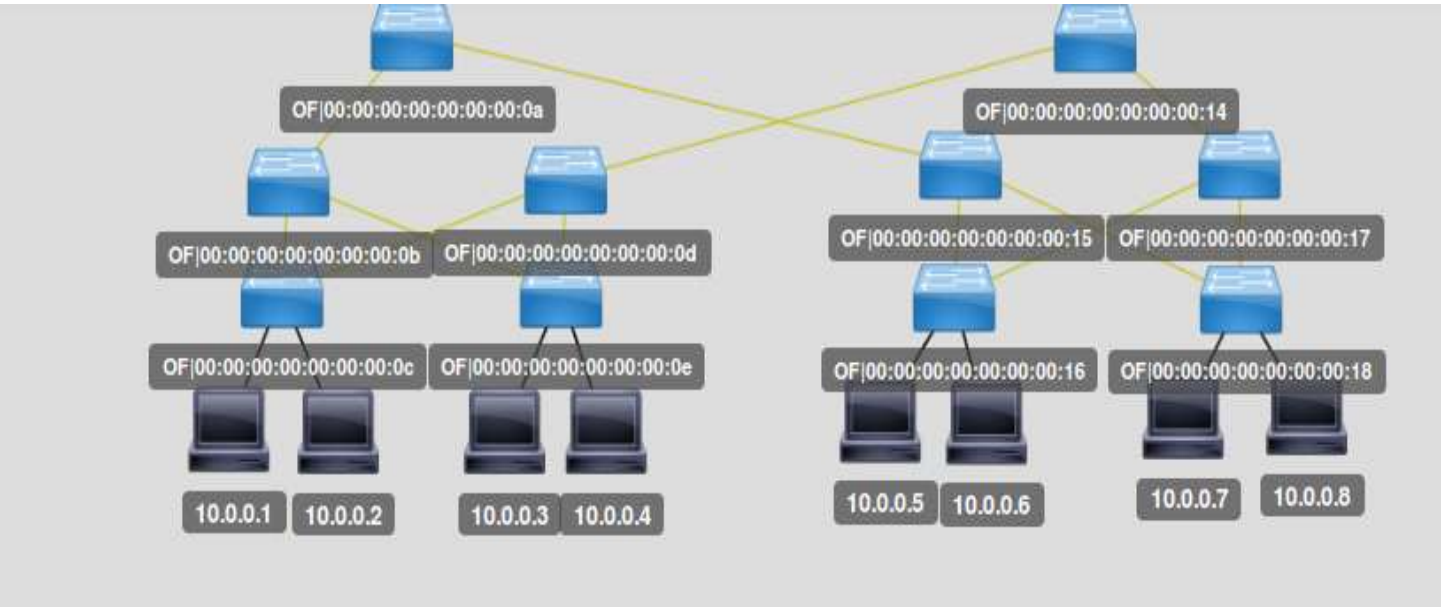


Controller then calculates the path with its logic based on statistics collector by the controller and it sends FLOW_MOD message back to the switch to program its flow,that can be seen in the Wireshark capture screen shot below



As we created network by python script in mininet the output can be collected from Open daylight controller and seen in GUI as below in which

- Switches a & 14 are core switches and b,d,15,17 are aggregation switches and c,e,16,18 are ToR switches
- Hosts are represented in black color



As soon as mininet connects to open daylight controller, controller learns nodes as below

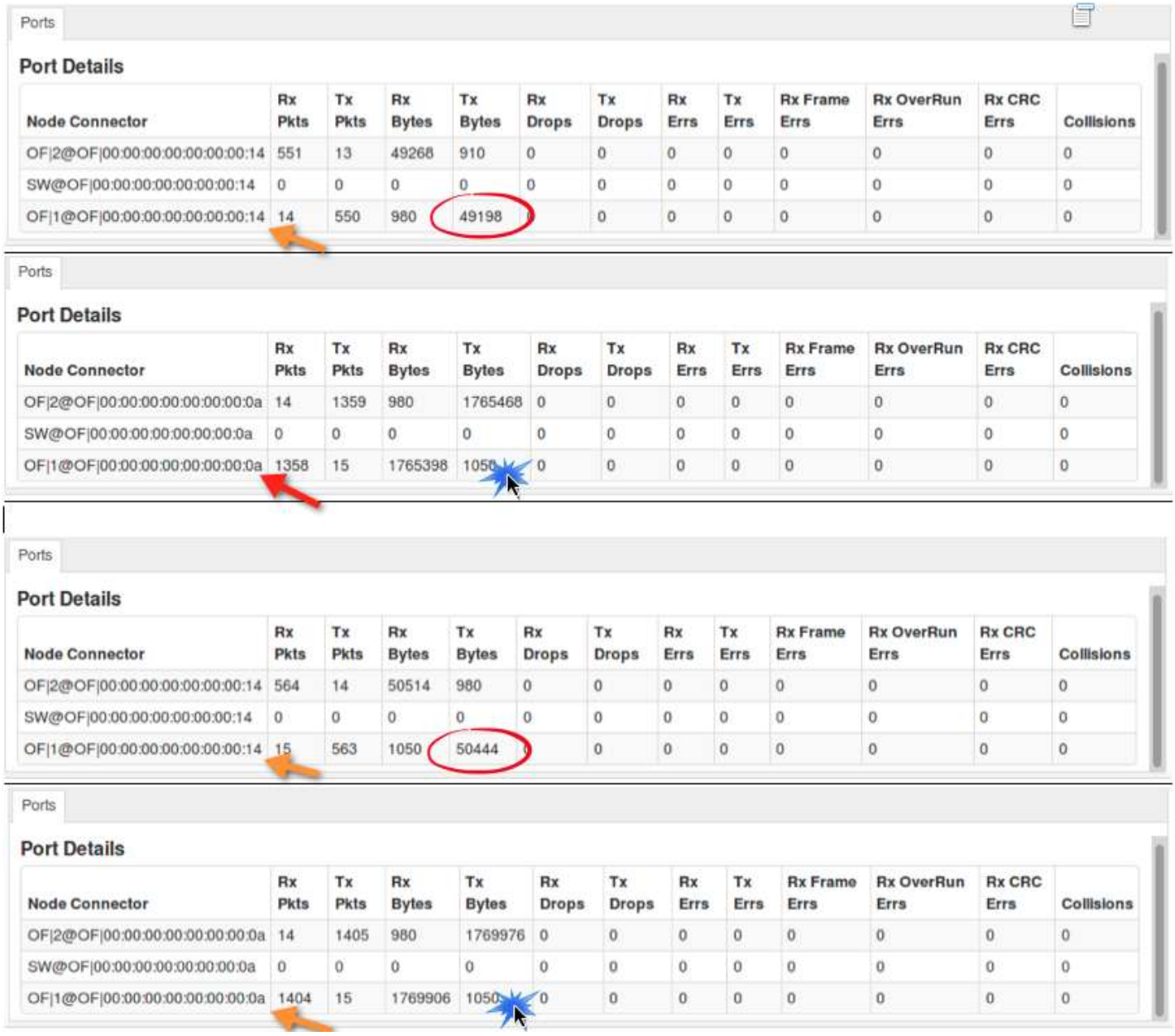
Nodes Learnt

Node ID	Node Name	Tier	Mac Address	Ports	Port Status
None	OF|00:00:00:00:00:00:00:15	Unknown (Tier-0)	000000000015	a21-eth1 (1) a21-eth2 (2) a21-eth3 (3)	
None	OF|00:00:00:00:00:00:00:0e	Access (Tier-1)	00000000000e	t14-eth1 (1) t14-eth2 (2) t14-eth3 (3) t14-eth4 (4)	
None	OF|00:00:00:00:00:00:00:0c	Access (Tier-1)	00000000000c	t12-eth1 (1) t12-eth2 (2) t12-eth3 (3) t12-eth4 (4)	
None	OF|00:00:00:00:00:00:00:14	Unknown (Tier-0)	000000000014	c20-eth1 (1) c20-eth2	

Comparison of output against hypothesis:

After controller has learnt all nodes and edges of network by using pingall command we have taken two core switches to test bandwidth utilization and automatic switching of traffic across less congested route.Below are the initial values of the two core switches 14 & a.

Further, we are sending ICMP traffic across two hosts.It is observed that switch 14 is taking the load while switch ‘a’ values remain the same. On adding additional iperf across 2 other hosts which could use the same path it is finally observed that once meeting the bandwidth optimization of switch 14 traffic is routed to switch ‘a’



Ports

Port Details

Node Connector	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
OF|2@OF|00:00:00:00:00:00:00:14	551	13	49268	910	0	0	0	0	0	0	0	0
SW@OF|00:00:00:00:00:00:00:14	0	0	0	0	0	0	0	0	0	0	0	0
OF|1@OF|00:00:00:00:00:00:00:14	14	550	980	49198	0	0	0	0	0	0	0	0

Ports

Port Details

Node Connector	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
OF|2@OF|00:00:00:00:00:00:00:0a	14	1359	980	1765468	0	0	0	0	0	0	0	0
SW@OF|00:00:00:00:00:00:00:0a	0	0	0	0	0	0	0	0	0	0	0	0
OF|1@OF|00:00:00:00:00:00:00:0a	1358	15	1765398	1050	0	0	0	0	0	0	0	0

Ports

Port Details

Node Connector	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
OF|2@OF|00:00:00:00:00:00:00:14	564	14	50514	980	0	0	0	0	0	0	0	0
SW@OF|00:00:00:00:00:00:00:14	0	0	0	0	0	0	0	0	0	0	0	0
OF|1@OF|00:00:00:00:00:00:00:14	15	563	1050	50444	0	0	0	0	0	0	0	0

Ports

Port Details

Node Connector	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
OF|2@OF|00:00:00:00:00:00:00:0a	14	1405	980	1769976	0	0	0	0	0	0	0	0
SW@OF|00:00:00:00:00:00:00:0a	0	0	0	0	0	0	0	0	0	0	0	0
OF|1@OF|00:00:00:00:00:00:00:0a	1404	15	1769906	1050	0	0	0	0	0	0	0	0

Conclusion:

We have successfully implemented traffic switching using OpenFlow using tools like mininet, iperf, opendaylight, OSGi,MAVEN.Link statistics are generated on a periodic basis and efficient switching is accomplished using opendaylight controller thus proving our hypothesis.

Bibliography

[1] Li, Y., & Pan, D. OpenFlow based Load Balancing for Fat-Tree Networks with Multipath Support.

[2] van der Pol, R., Boele, S., Dijkstra, F., Barczyk, A., van Malenstein, G., Chen, J. H., & Mambretti, J. (2012, November). Multipathing with MPTCP and OpenFlow. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:* (pp. 1617-1624). IEEE.

[3] Wang, R., Butnariu, D., & Rexford, J. (2011, March). OpenFlow-based server load balancing gone wild. In *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services* (pp. 12-12). USENIX Association.

[4] Nguyen, S. C., Zhang, X., Nguyen, T. M. T., & Pujolle, G. (2011, May). Evaluation of throughput optimization and load sharing of multipath tcp in heterogeneous networks. In *Wireless and Optical Communications Networks (WOCN), 2011 Eighth International Conference on* (pp. 1-5). IEEE.

[5] http://en.wikipedia.org/wiki/Software-defined_networking

[6] https://www.opennetworking.org/index.php?option=com_content&view=article&id=686&Itemid=272&lang=en

[7] http://clusterdesign.org/fat-trees/

[8] http://www.opendaylight.org/project/technical-overview

[9] http://en.wikipedia.org/wiki/Iperf

[10] https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[11] http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

[12] http://jackbrowntelecomprofessional.wordpress.com/2011/10/19/software-defined-networking-sdn/

[13] http://vlkan.com/blog/post/2013/07/31/openflow-controllers/

Appendices

Code:

Code for Mininet :

```
#!/usr/bin/python

#1. arguments
#3. spline

from optparse import OptionParser
import os
import sys
import time


from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import irange,dumpNodeConnections
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.link import TCLink
#from mininet.node import CPULimitedHost
from mininet.node import OVSSwitch, Controller, RemoteController

#global variable here
########################################

coreList = [ ]
aggList = [ ]
torList = [ ]

spineList = [ ]
leafList = [ ]

allList = [ ]


########################################

#Define topologies here##################
##########################################

#Data center Fat Tree Network Topology
class dcFatTreeTopo(Topo):
```

```
"Linear topology of k switches, with one host per switch."

def __init__(self, k=2, **opts):
    """Init.
        k: number of switches (and hosts)
        hconf: host configuration options
        lconf: link configuration options"""

    super(dcFatTreeTopo, self).__init__(**opts)

    self.k = k


    link1 = dict(bw=10, delay='1ms', loss=0, max_queue_size=1000, use_htb=True)
    link2 = dict(bw=5, delay='10ms', loss=0, max_queue_size=500, use_htb=True)
    link3 = dict(bw=1, delay='15ms', loss=0, max_queue_size=100, use_htb=True)

    # Creating an array of core switches
    # Adding them in an array so that they can be refered to later

    for i in irange(0, k-1):
        "core switch"
        coreSwitch = self.addSwitch('c%s%s' % (i+1, 0))
        #coreList.insert(i, coreSwitch)
        coreList.append(coreSwitch)


    #lastSwitch = None
    for i in irange(1, k):

        "aggregation switches"
        aggSwitch1 = self.addSwitch('a%s%s' % (i, 1))
        aggSwitch2 = self.addSwitch('a%s%s' % (i, 3))
        aggList.append(aggSwitch1)
        aggList.append(aggSwitch2)

        torSwitch1 = self.addSwitch('t%s%s' % (i, 2))
        torSwitch2 = self.addSwitch('t%s%s' % (i, 4))
        torList.append(torSwitch1)
        torList.append(torSwitch2)

        "host = self.addHost('h%s%s' % (i, i+1))"
        host11 = self.addHost('h%s%s' % (i, 1))
        host12 = self.addHost('h%s%s' % (i, 2))
        host13 = self.addHost('h%s%s' % (i, 3))
        host14 = self.addHost('h%s%s' % (i, 4))
        #hosts1 = [ net.addHost( 'h%d' % n ) for n in 3, 4 ]

        "connection of the hosts to the left tor switch "
        self.addLink(host11, torSwitch1, **link3)
        self.addLink(host12, torSwitch1, **link3)

        "connection of the hosts to the right tor switch "
        self.addLink(host13, torSwitch2, **link3)
        self.addLink(host14, torSwitch2, **link3)

        "connection of the the left tor switch to aggregation switches"
        self.addLink(torSwitch1, aggSwitch1, **link2)
```

```
            self.addLink(torSwitch1, aggSwitch2, **link2)

            "connection of the the right tor switch to aggregation switches"
            self.addLink(torSwitch2, aggSwitch1, **link2)
            self.addLink(torSwitch2, aggSwitch2, **link2)

        "connect the aggregation switch to top pod core switch"
        if k == 1:
            for r in irange(0, k): #this is to go through the agg switches
                self.addLink(aggList[r], coreList[0])
        else:
            for r in irange(0, (k*2)-1): #this is to go through the agg switches
                if  r % 2  == 0: #if agg switch is even then connect to first half
                    for j in irange(0, ((k/2)-1)): #this is to go through the core switches
                        self.addLink(aggList[r], coreList[j], **link1)
                else:
                    for j in irange((k/2), k-1): #this is to go through the core switches
                        self.addLink(aggList[r], coreList[j], **link1)

        allList.extend(coreList)
        allList.extend(aggList)
        allList.extend(torList)


def evenSimpleTest():

    for sw in allList:
        print allList[sw]

def simpleTest():

    # argument to put in either remote or local controller

    "Create and test a simple network"
    c0 = RemoteController( 'c0', ip='192.168.90.146' )

    # the cmap here needs to dynamically take the switch name from the switchLists[] so that it is not static
    #cmap = { 'a11': c0, 'a12': c0, 'a21': c0, 'a22': c0, 'a31': c0, 'a32': c0, 'a41': c0, 'a42': c0, 'c11': c0, 'c21': c0, 'c31': c0,
'c41': c0, 't11': c0, 't12': c0, 't21': c0, 't22': c0, 't31': c0, 't32': c0, 't41': c0, 't42': c0}


    class MultiSwitch( OVSSwitch ):
        "Custom Switch() subclass that connects to different controllers"
    def start( self, controllers ):
        return OVSSwitch.start( self, [ cmap[ self.name ] ] )

    #section for handling the differnt argumetns.... simpleTest(arg1, arg2, ...) will take in arguments from user

    topo = dcFatTreeTopo(k=2)
    net = Mininet(  topo=topo, switch=MultiSwitch, build=False, link=TCLink )

    print "connecting all SWITCHES to controller with cmap"
    cString = "{"
    for i in irange(0, len(allList)-1):
        if i != len(allList)-1:
            tempCString = "'" + allList[i] + "'" + " : c0, "
        else:
            tempCString = "'" + allList[i] + "'" + " : c0 "
```

```
        cString += tempCString

    cmapString = cString + "}"

    #print "wowzer" + cmapString
    cmap = cmapString

    net.addController(c0)

    net.build()
    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"

    #def perfTest():

    # if user test argument is active then pick the correct test

    net.pingAll()
    net.pingAll()
    print "Testing bandwidth between h11 and h12....."
    #h11, h12 = net.get('h11', 'h12')
    #net.iperf((h11, h12)

    #print "Testing bandwidth between h11 and h14....."
    h11, h14 = net.get('h11', 'h14')
    net.iperf((h11, h14))

    #print "Testing bandwidth between h11 and h16....."
    h11, h22 = net.get('h11', 'h22')
    net.iperf((h11, h22))

    #print "Testing bandwidth between h11 and h18....."
    h11, h24 = net.get('h11', 'h24')
    net.iperf((h11, h24))

    # also argument for generating traffic

    # arugment for stat analysis

    CLI( net )
    net.stop()

if __name__ == '__main__':
    # get arguments here to make the code configurable
    # pass in the arguments into simpleTest() so that they can be processed in SimpleTest

    import sys

    #print (sys.argv[1:])
    # Tell mininet to print useful information
    setLogLevel('info')
    simpleTest()
    #evenSimpleTest()
```

Opendaylight controller :

```
/**
 * @file   LbRoutingImplementation.java
 *
 *
 * @brief  Implementation of a routing engine using
 * lb_routing. Implementation of lb_routing come from Jung2 library
 *
 */


package org.opendaylight.controller.routing.lb_routing_implementation.internal;

import org.opendaylight.controller.sal.core.Bandwidth;
import org.opendaylight.controller.sal.core.ConstructionException;
import org.opendaylight.controller.sal.core.Edge;
import org.opendaylight.controller.sal.core.Node;
import org.opendaylight.controller.sal.core.NodeConnector;
import org.opendaylight.controller.sal.core.Path;
import org.opendaylight.controller.sal.core.Property;
import org.opendaylight.controller.sal.core.UpdateType;
import org.opendaylight.controller.sal.reader.IReadService;
import org.opendaylight.controller.sal.routing.IListenRoutingUpdates;
import org.opendaylight.controller.sal.routing.IRouting;
import org.opendaylight.controller.sal.topology.TopoEdgeUpdate;
import org.opendaylight.controller.switchmanager.ISwitchManager;
import org.opendaylight.controller.topologymanager.ITopologyManager;
import org.opendaylight.controller.topologymanager.ITopologyManagerAware;
import org.opendaylight.controller.statisticsmanager.internal.StatisticsManager;
import edu.uci.ics.jung.algorithms.shortestpath.DijkstraShortestPath;
import edu.uci.ics.jung.graph.Graph;
import edu.uci.ics.jung.graph.SparseMultigraph;
import edu.uci.ics.jung.graph.util.EdgeType;
import java.lang.Exception;
import java.lang.IllegalArgumentException;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
import java.util.Set;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;
import java.util.Timer;
import java.util.TimerTask;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.apache.commons.collections15.Transformer;

public class LbRoutingImplementation implements IRouting, ITopologyManagerAware {
    private static Logger log = LoggerFactory
            .getLogger(LbRoutingImplementation.class);
    private ConcurrentMap<Short, Graph<Node, Edge>> topologyBWAware;
    private ConcurrentMap<Short, DijkstraShortestPath<Node, Edge>> sptBWAware;
    DijkstraShortestPath<Node, Edge> mtp; // Max Throughput Path
    private Set<IListenRoutingUpdates> routingAware;
    private ISwitchManager switchManager;
```

```
private ITopologyManager topologyManager;
private IReadService readService;
private static final long DEFAULT_LINK_SPEED = Bandwidth.BW1Gbps;
private StatisticsManager statMgr;
private Timer lbRoutingTimer;
private TimerTask lbRoutingTimerTask;


public void setListenRoutingUpdates(IListenRoutingUpdates i) {
    if (this.routingAware == null) {
        this.routingAware = new HashSet<IListenRoutingUpdates>();
    }
    if (this.routingAware != null) {
        log.debug("Adding routingAware listener: {}", i);
        this.routingAware.add(i);
    }
}

public void unsetListenRoutingUpdates(IListenRoutingUpdates i) {
    if (this.routingAware == null) {
        return;
    }
    log.debug("Removing routingAware listener");
    this.routingAware.remove(i);
    if (this.routingAware.isEmpty()) {
        // We don't have any listener lets dereference
        this.routingAware = null;
    }
}

@Override
public synchronized void initMaxThroughput(
        final Map<Edge, Number> EdgeWeightMap) {
    if (mtp != null) {
        log.error("Max Throughput Dijkstra is already enabled!");
        return;
    }
    Transformer<Edge, ? extends Number> mtTransformer = null;
    if (EdgeWeightMap == null) {
        mtTransformer = new Transformer<Edge, Double>() {
            public Double transform(Edge e) {
                if (switchManager == null) {
                    log.error("switchManager is null");
                    return (double) -1;
                }
                NodeConnector srcNC = e.getTailNodeConnector();
                NodeConnector dstNC = e.getHeadNodeConnector();
                if ((srcNC == null) || (dstNC == null)) {
                    log.error("srcNC:{} or dstNC:{} is null", srcNC, dstNC);
                    return (double) -1;
                }
                Bandwidth bwSrc = (Bandwidth) switchManager
                        .getNodeConnectorProp(srcNC,
                                Bandwidth.BandwidthPropName);
                Bandwidth bwDst = (Bandwidth) switchManager
                        .getNodeConnectorProp(dstNC,
                                Bandwidth.BandwidthPropName);
```

```
            long srcLinkSpeed = 0, dstLinkSpeed = 0;
            if ((bwSrc == null)
                    || ((srcLinkSpeed = bwSrc.getValue()) == 0)) {
                log.debug(
                        "srcNC: {} - Setting srcLinkSpeed to Default!",
                        srcNC);
                srcLinkSpeed = DEFAULT_LINK_SPEED;
            }

            if ((bwDst == null)
                    || ((dstLinkSpeed = bwDst.getValue()) == 0)) {
                log.debug(
                        "dstNC: {} - Setting dstLinkSpeed to Default!",
                        dstNC);
                dstLinkSpeed = DEFAULT_LINK_SPEED;
            }

            long avlSrcThruPut = srcLinkSpeed
                    - readService.getTransmitRate(srcNC);
            long avlDstThruPut = dstLinkSpeed
                    - readService.getTransmitRate(dstNC);

            // Use lower of the 2 available thruput as the available
            // thruput
            long avlThruPut = avlSrcThruPut < avlDstThruPut ? avlSrcThruPut
                    : avlDstThruPut;

            if (avlThruPut <= 0) {
                log.debug("Edge {}: Available Throughput {} <= 0!", e,
                        avlThruPut);
                return (double) -1;
            }
            return (double) (Bandwidth.BW1Pbps / avlThruPut);
        }
    };
  } else {
    mtTransformer = new Transformer<Edge, Number>() {
        public Number transform(Edge e) {
            return EdgeWeightMap.get(e);
        }
    };
  }
  Short baseBW = Short.valueOf((short) 0);
  // Initialize mtp also using the default topo
  Graph<Node, Edge> g = this.topologyBWAware.get(baseBW);
  if (g == null) {
    log.error("Default Topology Graph is null");
    return;
  }
  mtp = new DijkstraShortestPath<Node, Edge>(g, mtTransformer);
}

@Override
public Path getRoute(Node src, Node dst) {
  log.debug("getRoute called");
  if (src == null || dst == null) {
    return null;
  }
```

```
    //return getRoute(src, dst, (short) 0);
    return getMaxThroughputRoute(src, dst);
}

@Override
public synchronized Path getMaxThroughputRoute(Node src, Node dst) {
    if (mtp == null) {
        log.error("Max Throughput Path Calculation Uninitialized!");
        return null;
    }
    log.debug("getMaxThroughputRoute called");
    List<Edge> path;
    try {
        path = mtp.getMaxThroughputPath(src, dst);
    } catch (IllegalArgumentException ie) {
        log.debug("A vertex is yet not known between {} {}", src, dst);
        return null;
    }
    Path res;
    try {
        res = new Path(path);
    } catch (ConstructionException e) {
        log.debug("A vertex is yet not known between {} {}", src, dst);
        return null;
    }
    return res;
}

@Override
public synchronized Path getRoute(Node src, Node dst, Short Bw) {
    DijkstraShortestPath<Node, Edge> spt = this.sptBWAware.get(Bw);
    if (spt == null)
        return null;
    List<Edge> path;
    try {
        path = spt.getPath(src, dst);
    } catch (IllegalArgumentException ie) {
        log.debug("A vertex is yet not known between {} {}", src, dst);
        return null;
    }
    Path res;
    try {
        res = new Path(path);
    } catch (ConstructionException e) {
        log.debug("A vertex is yet not known between {} {}", src, dst);
        return null;
    }
    return res;
}

@Override
public synchronized void clear() {
    DijkstraShortestPath<Node, Edge> spt;
    for (Short bw : this.sptBWAware.keySet()) {
        spt = this.sptBWAware.get(bw);
        if (spt != null) {
            spt.reset();
        }
```

```
    }
    clearMaxThroughput();
}

@Override
public synchronized void clearMaxThroughput() {
    if (mtp != null) {
        mtp.reset(); // reset maxthruput path
    }
}

@SuppressWarnings({ "rawtypes", "unchecked" })
private synchronized boolean updateTopo(Edge edge, Short bw, boolean added) {
    Graph<Node, Edge> topo = this.topologyBWAware.get(bw);
    DijkstraShortestPath<Node, Edge> spt = this.sptBWAware.get(bw);
    boolean edgePresentInGraph = false;
    Short baseBW = Short.valueOf((short) 0);

    if (topo == null) {
        // Create topology for this BW
        Graph<Node, Edge> g = new SparseMultigraph();
        this.topologyBWAware.put(bw, g);
        topo = this.topologyBWAware.get(bw);
        this.sptBWAware.put(bw, new DijkstraShortestPath(g));
        spt = this.sptBWAware.get(bw);
    }

    if (topo != null) {
        NodeConnector src = edge.getTailNodeConnector();
        NodeConnector dst = edge.getHeadNodeConnector();
        if (spt == null) {
            spt = new DijkstraShortestPath(topo);
            this.sptBWAware.put(bw, spt);
        }

        if (added) {
            // Make sure the vertex are there before adding the edge
            topo.addVertex(src.getNode());
            topo.addVertex(dst.getNode());
            // Add the link between
            edgePresentInGraph = topo.containsEdge(edge);
            if (edgePresentInGraph == false) {
                try {
                    topo.addEdge(new Edge(src, dst), src.getNode(),
                            dst.getNode(), EdgeType.DIRECTED);
                } catch (ConstructionException e) {
                    log.error("", e);
                    return edgePresentInGraph;
                }
            }
        } else {
            // Remove the edge
            try {
                topo.removeEdge(new Edge(src, dst));
            } catch (ConstructionException e) {
                log.error("", e);
                return edgePresentInGraph;
            }
```

```
            // If the src and dst vertex don't have incoming or
            // outgoing links we can get ride of them
            if (topo.containsVertex(src.getNode())
                    && topo.inDegree(src.getNode()) == 0
                    && topo.outDegree(src.getNode()) == 0) {
                log.debug("Removing vertex {}", src);
                topo.removeVertex(src.getNode());
            }

            if (topo.containsVertex(dst.getNode())
                    && topo.inDegree(dst.getNode()) == 0
                    && topo.outDegree(dst.getNode()) == 0) {
                log.debug("Removing vertex {}", dst);
                topo.removeVertex(dst.getNode());
            }
        }
        spt.reset();
        if (bw.equals(baseBW)) {
            clearMaxThroughput();
        }
    } else {
        log.error("Cannot find topology for BW {} this is unexpected!", bw);
    }
    return edgePresentInGraph;
}

private boolean edgeUpdate(Edge e, UpdateType type, Set<Property> props) {
    String srcType = null;
    String dstType = null;

    if (e == null || type == null) {
        log.error("Edge or Update type are null!");
        return false;
    } else {
        srcType = e.getTailNodeConnector().getType();
        dstType = e.getHeadNodeConnector().getType();

        if (srcType.equals(NodeConnector.NodeConnectorIDType.PRODUCTION)) {
            log.debug("Skip updates for {}", e);
            return false;
        }

        if (dstType.equals(NodeConnector.NodeConnectorIDType.PRODUCTION)) {
            log.debug("Skip updates for {}", e);
            return false;
        }
    }

    Bandwidth bw = new Bandwidth(0);
    boolean newEdge = false;
    if (props != null)
        props.remove(bw);

    if (log.isDebugEnabled()) {
      log.debug("edgeUpdate: {} bw: {}", e, bw.getValue());
    }
```

```
    Short baseBW = Short.valueOf((short) 0);
    boolean add = (type == UpdateType.ADDED) ? true : false;
    // Update base topo
    newEdge = !updateTopo(e, baseBW, add);
    if (newEdge == true) {
        if (bw.getValue() != baseBW) {
            // Update BW topo
            updateTopo(e, (short) bw.getValue(), add);
        }
    }
    return newEdge;
}

@Override
public void edgeUpdate(List<TopoEdgeUpdate> topoedgeupdateList) {
    boolean callListeners = false;
    for (int i = 0; i < topoedgeupdateList.size(); i++) {
        Edge e = topoedgeupdateList.get(i).getEdge();
        Set<Property> p = topoedgeupdateList.get(i).getProperty();
        UpdateType type = topoedgeupdateList.get(i).getUpdateType();
        if ((edgeUpdate(e, type, p)) && (!callListeners)) {
            callListeners = true;
        }
    }
    if ((callListeners) && (this.routingAware != null)) {
        /*Map<Edge, Number> edgeWeightMap = new  Map<Edge, Number >();
        Bandwidth bw = new Bandwidth(0);
        Graph<Node, Edge> topo = this.topologyBWAware.get(bw);
        Collection<Edge> edges = topo.getEdges();
        Iterator<Edge> iterEdge;
        Number i;
        for(iterEdge = edges.iterator(), i = 0;iterEdge.hasNext();){
            Edge e = iterEdge.next();
            edgeWeightMap.put(e,i);
            i = i.intValue()+1;
        }*/
        clearMaxThroughput();
        mtp = null;
        initMaxThroughput(null);
        for (IListenRoutingUpdates ra : this.routingAware) {
            try {
                ra.recalculateDone();
            } catch (Exception ex) {
                log.error("Exception on routingAware listener call", ex);
            }
        }
    }
}

private void updateEdgeUtilization() {
    log.debug("UpdateEdgeUtilization called");
    clearMaxThroughput();
    mtp = null;
    initMaxThroughput(null);
}

/**
 * Function called by the dependency manager when all the required
```

```
 * dependencies are satisfied
 *
 */
@SuppressWarnings({ "unchecked", "rawtypes" })
public void init() {
    log.debug("Routing init() is called");
    this.topologyBWAware = (ConcurrentMap<Short, Graph<Node, Edge>>) new ConcurrentHashMap();
    this.sptBWAware = (ConcurrentMap<Short, DijkstraShortestPath<Node, Edge>>) new ConcurrentHashMap();
    // Now create the default topology, which doesn't consider the
    // BW, also create the corresponding Dijkstra calculation
    Graph<Node, Edge> g = new SparseMultigraph();
    Short sZero = Short.valueOf((short) 0);
    this.topologyBWAware.put(sZero, g);
    this.sptBWAware.put(sZero, new DijkstraShortestPath(g));
    // Topologies for other BW will be added on a needed base
    this.statMgr = new StatisticsManager();
    this.lbRoutingTimer = new Timer();
    this.lbRoutingTimerTask = new TimerTask() {
        @Override
        public void run() {
            updateEdgeUtilization();
        }
    };
}

/**
 * Function called by the dependency manager when at least one dependency
 * become unsatisfied or when the component is shutting down because for
 * example bundle is being stopped.
 *
 */
void destroy() {
    log.debug("Routing destroy() is called");
}

/**
 * Function called by dependency manager after "init ()" is called and after
 * the services provided by the class are registered in the service registry
 *
 */
void start() {
    log.debug("Routing start() is called");
    // build the routing database from the topology if it exists.
    Map<Edge, Set<Property>> edges = topologyManager.getEdges();
    if (edges.isEmpty()) {
        return;
    }
    List<TopoEdgeUpdate> topoedgeupdateList = new ArrayList<TopoEdgeUpdate>();
    log.debug("Creating routing database from the topology");

    for (Iterator<Map.Entry<Edge, Set<Property>>> i = edges.entrySet()
            .iterator(); i.hasNext();) {
        Map.Entry<Edge, Set<Property>> entry = i.next();
        Edge e = entry.getKey();
        Set<Property> props = entry.getValue();
        TopoEdgeUpdate topoedgeupdate = new TopoEdgeUpdate(e, props,
                UpdateType.ADDED);
        topoedgeupdateList.add(topoedgeupdate);
```

```
    }
    edgeUpdate(topoedgeupdateList);
 // StatsTimer is set to 10s
    lbRoutingTimer.scheduleAtFixedRate(lbRoutingTimerTask, 0, 10000);

}

/**
 * Function called by the dependency manager before the services exported by
 * the component are unregistered, this will be followed by a "destroy ()"
 * calls
 *
 */
public void stop() {
    log.debug("Routing stop() is called");
}

@Override
public void edgeOverUtilized(Edge edge) {
    // TODO Auto-generated method stub

}

@Override
public void edgeUtilBackToNormal(Edge edge) {
    // TODO Auto-generated method stub

}

public void setSwitchManager(ISwitchManager switchManager) {
    this.switchManager = switchManager;
}

public void unsetSwitchManager(ISwitchManager switchManager) {
    if (this.switchManager == switchManager) {
        this.switchManager = null;
    }
}

public void setReadService(IReadService readService) {
    this.readService = readService;
}

public void unsetReadService(IReadService readService) {
    if (this.readService == readService) {
        this.readService = null;
    }
}

public void setTopologyManager(ITopologyManager tm) {
    this.topologyManager = tm;
}

public void unsetTopologyManager(ITopologyManager tm) {
    if (this.topologyManager == tm) {
        this.topologyManager = null;
    }
}
```

```
}


<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <prerequisites>
    <maven>3.0</maven>
  </prerequisites>

  <scm>
    <connection>scm:git:https://git.opendaylight.org/gerrit/p/controller.git</connection>
    <developerConnection>scm:git:ssh://git.opendaylight.org:29418/controller.git</developerConnection>
  </scm>
  <parent>
    <groupId>org.opendaylight.controller</groupId>
    <artifactId>commons.opendaylight</artifactId>
    <version>1.4.0-SNAPSHOT</version>
    <relativePath>../../commons/opendaylight</relativePath>
  </parent>

  <repositories>
    <!-- To get SVNKit -->
    <repository>
      <id>svnkit-snapshots</id>
      <name>svnkit-snapshots</name>
      <url>${nexusproxy}/repositories/svnkit-snapshots/</url>
    </repository>
  </repositories>

  <artifactId>distribution.opendaylight</artifactId>
  <version>0.1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <modules>
    <module>../../forwarding/staticrouting</module>
    <module>../../clustering/services</module>
    <module>../../clustering/services_implementation</module>
    <module>../../clustering/integrationtest</module>
    <module>../../clustering/stub</module>
    <module>../../clustering/test</module>
    <module>../../configuration/api</module>
    <module>../../configuration/implementation</module>
    <module>../../arphandler</module>
    <module>../../forwardingrulesmanager/api</module>
    <module>../../forwardingrulesmanager/implementation</module>
    <module>../../forwardingrulesmanager/integrationtest</module>
    <module>../../hosttracker/api</module>
    <module>../../hosttracker/implementation</module>
    <module>../../hosttracker/integrationtest</module>
    <module>../../containermanager/api</module>
    <module>../../containermanager/implementation</module>
    <module>../../switchmanager/api</module>
    <module>../../switchmanager/implementation</module>
    <module>../../switchmanager/integrationtest</module>
    <module>../../statisticsmanager/api</module>
    <module>../../statisticsmanager/implementation</module>
```

```
  <module>../../statisticsmanager/integrationtest</module>
  <module>../../topologymanager</module>
  <module>../../usermanager</module>
  <module>../../security</module>


  <module>../../../third-party/openflowj</module>
  <module>../../../third-party/net.sf.jung2</module>
  <module>../../../third-party/jersey-servlet</module>

  <!-- SAL bundles -->
  <module>../../sal/api</module>
  <module>../../sal/implementation</module>

  <!--  Web bundles -->
  <module>../../web/root</module>
  <module>../../web/flows</module>
  <module>../../web/devices</module>
  <module>../../web/troubleshoot</module>
  <module>../../web/topology</module>

  <!-- Northbound bundles -->
  <module>../../northbound/commons</module>
  <module>../../northbound/topology</module>
  <module>../../northbound/staticrouting</module>
  <module>../../northbound/statistics</module>
  <module>../../northbound/flowprogrammer</module>
  <module>../../northbound/hosttracker</module>
  <module>../../northbound/subnets</module>
  <module>../../northbound/switchmanager</module>

  <!-- Northbound integration tests -->
  <module>../../northbound/integrationtest</module>
  <!-- Debug and logging -->
  <module>../../logging/bridge</module>

  <!-- Southbound bundles -->
  <module>../../protocol_plugins/openflow</module>
  <module>../../protocol_plugins/stub</module>

  <!-- Samples -->
  <module>../../samples/simpleforwarding</module>
  <module>../../samples/loadbalancer</module>
  <module>../../samples/northbound/loadbalancer</module>
  <module>../../commons/concepts</module>
  <module>../../routing/lb_routing_implementation</module>
</modules>

<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>buildnumber-maven-plugin</artifactId>
      <version>1.2</version>
      <executions>
        <execution>
          <phase>validate</phase>
          <goals>
```

```
            <goal>create</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <doCheck>false</doCheck>
        <doUpdate>false</doUpdate>
        <providerImplementations>
          <svn>javasvn</svn>
        </providerImplementations>
        <revisionOnScmFailure>VersionUnknown</revisionOnScmFailure>
      </configuration>
      <dependencies>
        <dependency>
          <groupId>com.google.code.maven-scm-provider-svnjava</groupId>
          <artifactId>maven-scm-provider-svnjava</artifactId>
          <version>2.0.5</version>
        </dependency>
        <dependency>
          <groupId>org.tmatesoft.svnkit</groupId>
          <artifactId>svnkit</artifactId>
          <version>1.7.4-v1</version>
        </dependency>
        <dependency>
          <groupId>org.apache.maven.scm</groupId>
          <artifactId>maven-scm-provider-svn-commons</artifactId>
          <version>1.7</version>
        </dependency>
      </dependencies>
    </plugin>
    <plugin>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>2.3</version>
      <executions>
        <execution>
          <id>distro-assembly</id>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
          <configuration>
            <descriptors>
              <descriptor>src/assemble/bin.xml</descriptor>
            </descriptors>
            <finalName>${project.artifactId}-${build.suffix}</finalName>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
 </build>
</project>
```