VIVEKANAND EDUCATION SOCIETY'S
INSTITUTE OF TECHNOLOGY

Department of Computer Engineering

Project Report on
A Comparative Study of Load Balancing using SDN and Traditional System

In partial fulfillment of the Fourth Year (Semester–VII), Bachelor of Engineering (B.E.) Degree in Computer Engineering at the University of Mumbai
Academic Year 2017-2018

Project Mentor
Dr.(Mrs.) Nupur Giri

Submitted by
Vikas Kukreja, D17B 34
Jatin Sajnani, D17B 63
Dinesh Panchi, D17A 54
Hitesh Seedani, D17A 68
(2017-18)

VIVEKANAND EDUCATION SOCIETY'S
INSTITUTE OF TECHNOLOGY

Department of Computer Engineering

CERTIFICATE of Approval

This is to certify that ____ of Fourth Year Computer Engineering studying under the University of Mumbai has satisfactorily presented the project on "**A Comparative Study of Load Balancing using SDN and Traditional System**" as a part of the coursework of PROJECT-I for Semester-VII under the guidance of **Dr. Mrs. Nupur Giri** in the year 2017-2018.

Date

Internal Examiner	External Examiner

Project Mentor	Head of the Department Dr. Mrs. Nupur Giri	Principal Dr. J. M. Nair

ACKNOWLEDGEMENT

We are thankful to our college Vivekanand Education Society's Institute of Technology for considering our project and extending help at all stages needed during our work of collecting information regarding the project.

It gives us immense pleasure to express our deep and sincere gratitude to Assistant Professor **Dr.(Mrs.) Nupur Giri (**Project Guide) for her kind help and valuable advice during the development of project synopsis and for her guidance and suggestions.

We are deeply indebted to Head of the Computer Department **Dr.(Mrs.) Nupur Giri** and our Principal **Dr. (Mrs.) J.M. Nair ,** for giving us this valuable opportunity to do this project.

We express our hearty thanks to them for their assistance without which it would have been difficult in finishing this project synopsis and project review successfully.

We convey our deep sense of gratitude to all teaching and non-teaching staff for their constant encouragement, support and selfless help throughout the project work. It is great pleasure to acknowledge the help and suggestion, which we received from the Department of Computer Engineering.

We wish to express our profound thanks to all those who helped us in gathering information about the project. Our families too have provided moral support and encouragement at several times.

Computer Engineering Department

COURSE OUTCOMES FOR B.E PROJECT

Learners will be to:-

Course Outcome	**Description of the Course Outcome**
CO 1	Able to apply the relevant engineering concepts, knowledge and skills towards the project.
CO2	Able to identify, formulate and interpret the various relevant research papers and to determine the problem.
CO 3	Able to apply the engineering concepts towards designing solution for the problem.
CO 4	Able to interpret the data and datasets to be utilized.
CO 5	Able to create, select and apply appropriate technologies, techniques, resources and tools for the project.
CO 6	Able to apply ethical, professional policies and principles towards societal, environmental, safety and cultural benefit.
CO 7	Able to function effectively as an individual, and as a member of a team, allocating roles with clear lines of responsibility and accountability.

CO 8	Able to write effective reports, design documents and make effective presentations.
CO 9	Able to apply engineering and management principles to the project as a team member.
CO 10	Able to apply the project domain knowledge to sharpen one's competency.
CO 11	Able to develop professional, presentational, balanced and structured approach towards project development.
CO 12	Able to adopt skills, languages, environment and platforms for creating innovative solutions for the project.

ABSTRACT

SDN is the only domain which brought the programmability, simplicity, and flexibility to the networks. Using SDN, the same architecture can be used for many applications like switches, routers, hubs, etc. This versatility of architecture was never before seen in any domain which makes SDN the future of Networking.Today due to the increase in the service requirement there is more load on the server this may result in Latency/Delay or sometimes the link failure or server crash. To avoid this latency or link failure or server crash there must be a Load Balancer between the client and server to handle this load. We will create a topology and deploy different load balancing applications to evaluate their performance.

INDEX

Chapter 1 : Introduction

1.1. Introduction to the project

Software-defined networking (SDN) technology is an approach to computer networking that allows network administrators to programmatically initialize, control, change, and manage network behaviour dynamically via open interfaces.SDN decouples or disassociates the Control plane and Data plane.

SDN Architecture has 3 Layers:

I. Infrastructure Layer.

II. Control Layer.

III. Application Layer.

Control Layer is the brain of the SDN, controlling of different Data Planes is done by this layer. Applications are installed over the Application Layer and the applications communicate with the Controller using the NorthBound APIs (Eg. Java APIs or REST APIs). Data Plane is to the south of the Controller, therefore the Data Plane will communicate with the Controller using the SouthBound APIs (Eg. OpenFlow Protocol). If there are more than one Controller than the Controllers can communicate with each other using EastBound and WestBound APIs.

1.2. Motivation for the project

Software defined networking (SDN) aims to simplify network management by removing the control plane from switches and running custom control applications at a logically central controller. Unfortunately, writing control applications that always maintain a set of network invariants (e.g., the network does not contain forwarding loops or black holes) is a challenging task.SDN can deliver a consistent, reliable and high-quality experience to all users, which has a significant impact on worker productivity.

Today’s business leaders are focused on enabling higher levels of worker productivity and improving business agility while lowering the overall cost of running IT.

Achieving this can be a daunting task because the IT environment has become increasingly complex. For decades, IT leaders have had to make trade-offs and choose products that either reduce costs or improve productivity because products rarely accomplish both. SDN can help create an environment where IT leaders no longer have to choose between IT projects that can either control cost or boost worker productivity.

With traditional networks, each location typically requires dedicated devices for each network service, no matter how small the location. This means each branch office would have its own separate router, firewall, VPN device and other devices to deliver all the services required. This "service chaining" is highly inefficient because each device must be configured individually, which increases management and troubleshooting complexity. In contrast, SDN enables multiple servers to run on a single hardware platform, reducing both troubleshooting and provisioning times. Service providers will increase the profitability of their services by leveraging SDN to reduce operational costs and hardware costs.

1.3. Drawback of the existing system

1.3.1. Traditional configuration is time-consuming and error-prone

Many steps are needed when an IT administrator needs to add or remove a single device in a traditional network. First, he will have to manually configure multiple devices (switches, routers, firewalls) on a device-by-device basis.

The next step is using device-level management tools to update numerous configuration settings, such as ACLs, VLANs and Quality of Service.

This configuration approach makes it that much more complex for an administrator to deploy a consistent set of policies. As a result, organizations are more likely to encounter security breaches, non-compliance with implications. Conclusion: the highly administrative 'hassle' that is traditional configuration interferes with meeting business networking standards.

1.3.2. Multi-vendor environments require a high level of expertise

The average organization owns a variety of equipment of different vendors. To successfully complete a configuration, an administrator will therefore need extensive knowledge of all present device types.

1.4. Problem Definition

With the growth and advancement in Industrialisation there is increase in the service requirement due to this the load on the server increases and this may result in latency/delay or sometimes the link failure or server crash.

To avoid this latency or link failure or server crash we must install a load balancer between the client and server to handle this load.

By using SDN we created a topology and installed a Load Balancer application on the Controller that will monitor the load on server as well as the links between the server and controller and controller and clients. Load Balancer Application uses 2-Threshold policy to transfer processes between the nodes where Threshold is calculated Dynamically and Altruistic Priority assignment policy in Underloaded Conditions.

1.5. Relevance of the Project

Weighted Round-Robin Load Balancing Using Software Defined Networking

The architecture of load balancing believes here consists of a load balancer that are connected to several target servers. The load balancer receives requests from the clients, the requests received from the client are then redirect to the target servers following a policy that are fixed by administer.The architecture of load balancing consists of OpenFlow switch connected to one POX controller and multiple servers that are linked through OpenFlow switch's ports. Static IP address is assigned to each server and the dictionary of live servers is maintained in POX controllers that are connected to the OpenFlow switch. Controller contains a virtual address. All requests coming from the clients are redirect to the virtual IP address .When the client dispatch a request packet to the virtual IP, information that are contained in the packet header, OpenFlow switch uses this information and contrast these information with the information stored in flow entries of switch, if the entries in flow table matches with client's packet header information then based on strategy implement on load balancer switch modifies the destination virtual IP address to the address of one of the servers and forward the packet to that particular server, If entries in flow table does not matches with any header information contained in packet flow entry, then the OpenFlow redirect packet to the controller. With the help of OpenFlow table the Controller inserts new flow entries to the switch's flow table. To implement load-balancing application, we wrote python modules that are executed by the POX controller.

1.6. Methodology used

1. Start OpenDaylight SDN Controller using command:

   ./karaf

2. Run the fat tree topology using Mininet:

   sudo mn --custom topology.py --topo mytopo --controller=remote,ip=127.0.0.1,port=6653

3. Run the pingall command to ensure all links are up and running.
4. Check the GUI of OpenDaylight using link:

   localhost.8181/index.html#/topology

5. Ping any two hosts to get real simulation of data transmission between nodes.
6. Run the load balancing script using command:

   python odl.py

7. Python file will run the application for load balancing between client and server application uses Altruistic Priority Assignment Policy and 2-threshold policy for deciding the workload of node.
8. Node can be in any of the following state
   a. Overloaded
   b. Normal
   c. underloaded
9. Threshold will be calculated dynamically and accordingly the load balancing application will work to tackle the load.

Chapter 2: Literature Survey

2.1. Research Papers

1. **Load Balancing algorithm by Process Migration**

   https://ijcsits.org/papers/vol2no62012/28vol2no6.pdf

   Abstract:

   In present scenario load balancing in distributed operating system is challenging topic. There are various methods to balance the load. But in this paper we have focus on process migration technique. This paper focus different algorithm for load balancing in distributed operating system.

   Inference:

   An algorithm has been present for load-balancing in distributed operating system with preemptive and nonpreemptive migration of process, and also it has been compared with available traditional algorithm based on parameter like CPU utilization and memory utilization of load balancing in distributed operating system. It shows that proposed algorithm is efficient than traditional algorithm for different type of application like CPU-bound process and memory bound process.

2. **OpenFlow Protocol**

   https://arxiv.org/ftp/arxiv/papers/1406/1406.0124.pdf

   Abstract:

   Real-time performance and high availability requirements have induced telecom networks to adopt the new concepts of the cloud model: software-defined networking (SDN) and network function virtualization (NFV). NFV introduces and deploys new network functions in an open and standardized IT environment, while SDN aims to transform the way networks function. SDN and NFV are complementary technologies; they do not depend on each other. However, both concepts can be merged and have the potential to mitigate the challenges of legacy networks. In this paper, our aim is to describe the benefits of using SDN in a multitude of environments such as in data centers, data center networks, and Network as Service offerings. We also present the various challenges facing SDN, from scalability to reliability and security concerns, and discuss existing solutions to these challenges.

Inference:

SDN provides a new concept and architecture for managing and configuring networks using a dynamic and agile infrastructure. But the networking area is not only experiencing the emergence of SDN but also network virtualization and network function virtualization. The three solutions build an automated, scalable, virtualized and agile networking and cloud environment.

3. **SDN Concepts and Challenges**

   http://sci-hub.io/http://ieeexplore.ieee.org/document/7821979/

   Abstract:

   Software Defined Networking (SDN) is an emerging networking paradigm that greatly simplifies network management tasks. In addition, it opens the door for network innovation through a programmable flexible interface controlling the behavior of the entire network. In the opposite side, for decades traditional IP networks were very hard to manage, error prone and hard to introduce new functionalities. In this paper, we introduce the concepts & applications of SDN with a focus on the open research challenges in this new technology.

   Inference:

   SDN is an emerging networking paradigm that allows the control of the network behavior through a centralized programming capability. SDN offers simplified and automated network management that meets the demand of increased network complexity & several application domains.

4. **SDN Security**

   http://sci-hub.io/http://ieeexplore.ieee.org/abstract/document/6702553/

   Abstract:

The pull of Software-Defined Networking (SDN) is magnetic. There are few in the networking community who have escaped its impact. As the benefits of network visibility and network device programmability are discussed, the question could be asked as to who exactly will benefit? Will it be the network operator or will it, in fact, be the network intruder? As SDN devices and systems hit the market, security in SDN must be raised on the agenda. This paper presents a comprehensive survey of the research relating to security in software-defined networking that has been carried out to date. Both the security enhancements to be derived from using the SDN framework and the security challenges introduced by the framework are discussed. By categorizing the existing work, a set of conclusions and proposals for future research directions are presented.

Inference:

There are two schools of thought on security in software-defined networking. The first is that significant improvements in network security can be achieved by simultaneously exploiting the programmability and the centralized network view introduced by SDN. The second is that these same two SDN attributes expose the network to a range of new attacks.

5. **Performance analysis of SDN**

http://sci-hub.io/http://ieeexplore.ieee.org/abstract/document/6730793/

Abstract:

Software-Defined Networking (SDN) approaches were introduced as early as the mid-1990s, but just recently became a well-established industry standard. Many network architectures and systems adopted SDN, and vendors are choosing SDN as an alternative to the fixed, predefined, and inflexible protocol stack. SDN offers flexible, dynamic, and programmable functionality of network systems, as well as many other advantages such as centralized control, reduced complexity, better user experience, and a dramatic decrease in network systems and equipment costs. However, SDN characterization and capabilities, as well as workload of the network traffic that the SDN-based systems handle, determine the level of these advantages. Moreover, the enabled flexibility of

SDN-based systems comes with a performance penalty. The design and capabilities of the underlying SDN infrastructure influence the performance of common network tasks, compared to a dedicated solution. In this paper we analyze two issues: a) the impact of SDN on raw performance (in terms of throughput and latency) under various workloads, and b) whether there is an inherent performance penalty for a complex, more functional, SDN infrastructure. Our results indicate that SDN does have a performance penalty; however, it is not necessarily related to the complexity level of the underlying SDN infrastructure.

Inference:

In this paper we analyzed the performance of two SDN architectures, OpenFlow and ProGFE, as a function of their complexity, flexibility, and potential functionality and capabilities. We saw that SDN flexibility does come at the expense of raw performance, enhanced by additional overhead for a more complex functionality

6. **Performance of OpenFlow Over Traditional Network :**

   http://sci-hub.io/http://ieeexplore.ieee.org/document/6873559/

   Abstract:

   We present a performance analysis of the SDN and OpenFlow over wireless networks. The aim is to evaluate potential advantages introduced by the SDN architecture in terms of Quality of Service (QoS) metrics such as throughput, packet loss, end-to-end delay, and packet delivery ratio.

   Inference:

   In this paper the impact of OpenFlow over wireless networks has been investigated in order to evaluate the enhancements introduced by the usage of the SDN architecture.

7. **Load Balancing using SDN**

   http://sci-hub.io/http://ieeexplore.ieee.org/document/7777454/

   Abstract:

   Current LTE networks use traditional load balancing algorithms to distribute traffic load among core network user plane nodes to access external data networks. The objective of this paper is to take advantage of a new SDN-based core architecture to perform SDN flow-based load balancing. The experimental results show that our application successfully balances the traffic between the nodes which preserved the high throughput and decreased latency among the new network nodes.

Inference:

In this paper, we have proposed a load balancing algorithm that is implemented using an SDN architecture for LTE network core. The collected statistics from the OF switches are sent periodically at 1-second intervals to the controller, which keeps the general state of traffic load in the network.The efficiency of our algorithm was discussed from two perspectives: the first is the success of traffic load balancing in a short time to preserve a high throughput, and the second is the latency, where the increase of network latency was due to the sudden increase of traffic load, but eventually it dropped back to the normal scale after the load was balanced.

8. **Method for variable Threshold Calculation**

http://ijcsit.com/docs/Volume%206/vol6issue02/ijcsit2015060220.pdf

Abstract:

VM migration is an important feature provided by the virtualization. It is the process of transferring the VM from one host to the host. VM migration is the costly operation which degrades the system performance. Proper load balancing can help to minimize the number migration as well as energy consumption. This paper present a load balancing approach based on the Virtual Machine migration. Lower and upper threshold are use for the load balancing. When the load on server is above the upper threshold or below the lower threshold, system is unbalanced and some VM has to be migrated. Our result show that the proposed method increase the resource utilization by applying the dynamic lower and upper threshold compare to the traditional VM migration algorithm. Keywords— Put your keyword

Inference:

In the cloud environment resource are provided to the clients on the basis of pay as you go model. User need to pay only for the resources which he used. Cost of the resources charged by the provider can be minimized by the proper utilization of the resource. Load balancing in the cloud is a very challenging task because of their dynamic nature. In this paper we present a load balancing method which use lower and upper threshold. Experiment result show that our method increased the resource utilization as compared to the threshold based method.

2.2. Interaction with Domain Expert

One day workshop was arranged by Dr. (Mrs.) Nupur Giri. The workshop was conducted by Gaurav Sharma from Pillai College of Engineering and was conducted at Pillai College of Engineering.

Chapter 3: Requirements Gathering

These days our networks have to handle large amount of traffic, serve thousands of clients. It is very difficult for a single server to handle such huge load. The solution is to use multiple servers with load balancer acting as a front end. The clients will send the requests to the load balancer. The load balancer will forward the client requests to different servers depending upon load balancing strategy. Load balancer use dedicated hardware. That hardware is expensive and inflexible. Currently available load balancers contain few algorithms that can be used. Network administrators can not create their own algorithms since traditional load balancer are vendor locked, non programmable. On the other hand SDN load balancers are programmable and allow you to design and implement your own load balancing strategy. Other advantages of SDN load balancer is we do not need dedicated hardware.

The requirements for the Load Balancing using SDN comprises of :-

3.1 Functional Requirements

1. **Coordination between layers of SDN**

   It is required to provide an orchestration functionality of the SDN application layer and SDN control layer, and multilayer management to handle the lifecycle management of software -based SDN functions.

2. **Scalability**

   It is required to provide scalability for supporting a large number of users.
   SDN controller plane can contain multiple SDN controller instances,arranged in tree or graph, with each level presenting a virtual view of the network to higher level, thereby further increasing scalability.

3. **Topology**

   It is required to discover underlying network topology to see how network resources (e.g., SDN-enabled switches) connect to each other.

4. **Efficient Route**

   It is required to build and update forwarding paths constituted by network resources, such as switches,routers and data processing entities.

5. **Interoperability**

   As interoperability is the primary goal of most standard,ensuring interoperability needs to be key focus item.Our proposed architecture can and should promote interoperability through standard protocols and API's like Infrastructure Layer lies to the South of the controller therefore this layer communicates with the controller using SouthBound API.

3.2. Non-Functional Requirements

1. **Security**

   In Order to provide secure functioning of the Controller,access to the Controller must be strictly controlled. In case of the attacks on the Controller (for example: DDoS), availability of the Controller needs to be maintained. Protecting the communications throughout the network is critical.This means ensuring the SDN Controller, the applications loaded on it, and the devices it manages are all trusted entities that are operating as they should.

2. **Performance**

   Load balancing application will balance the load between clients and servers and Maintain the overall performance of the system.

3. **Robustness**

   SDN provides an environment that is both highly flexible and more robust in the face of constantly changing workloads.

4. **Maintenance**

The system should be updated from time to time so that system can work efficiently for longer period and provide better service.

3.3. Constraints

- Good Knowledge About Networking Commands.
- Active Links Between Components of Network like Switches,Hosts and Controller.
- Port Number 6653 and 8181 or 8080 Should be Active.
- Active Internet Connection on Hosts (Servers and Clients).

3.4. Hardware & Software Requirements

3.4.1. Hardware Requirements

➢ **For SDN Controller**

- A Computer with 8GB RAM.
- Octa-core Processor.
- Core I5 or Higher Versions of Microprocessor.

➢ **For Server and Clients**

- A Computer Machine with 4GB of RAM.
- Normal cores and processors.

3.4.2. Software Requirements

- Ubuntu 14.04 LTS with Linux Kernel 3.16.0-77-genetic.
- OpenDayLight Controller - Carbon.
- Setup Environment for OpenJDK or Oracle java.
- Mininet Version 2.2.1.
- Any Browser For Graphical Unit Interface.

3.5. Techniques utilized till date for the proposed system

Load balancing using OpenDaylight SDN Controller

Methods used in the technique:

- Altruistic Property
- 2 threshold Policy

3.6. Tools utilized till date for the proposed system

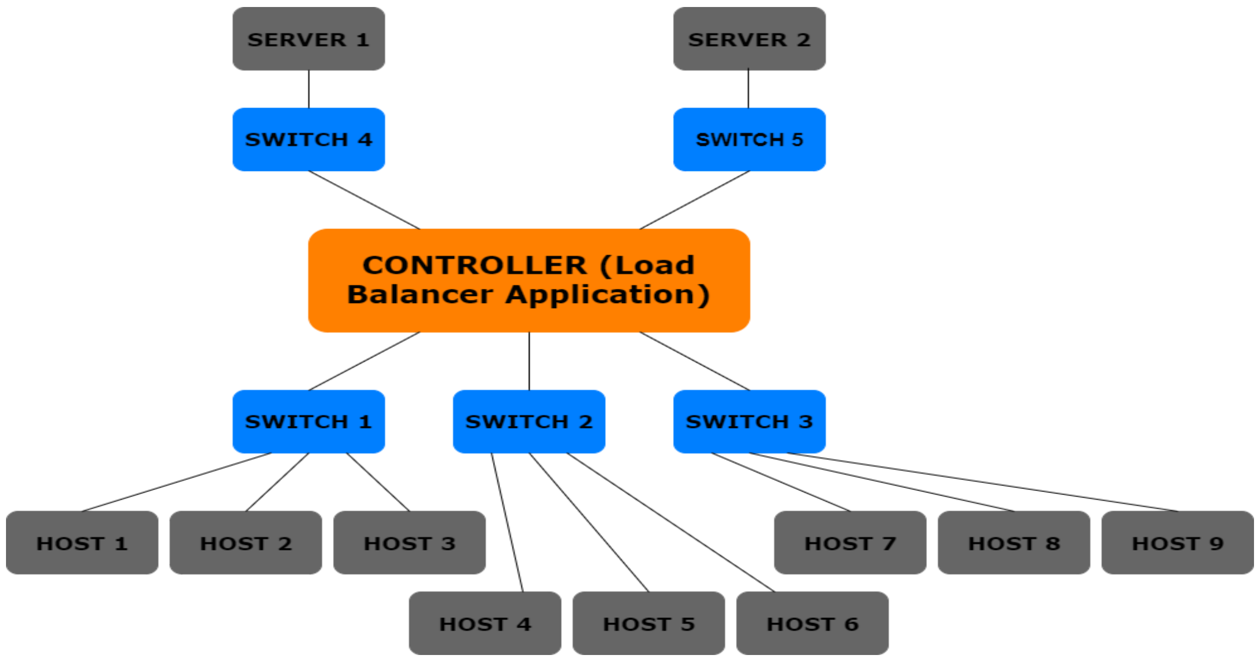
- Ubuntu
- OpenDayLight Controller
- Oracle java
- Mininet

3.7. Algorithms utilized in the existing systems

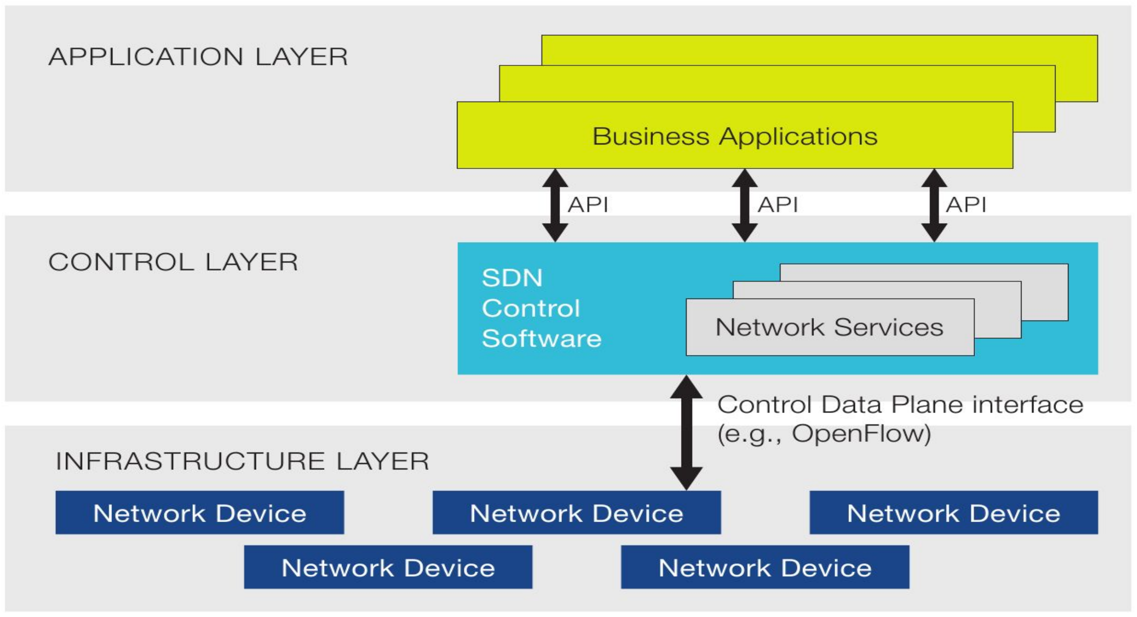
- Round Robin Algorithm
- Randomized Algorithm
- Threshold Algorithm
- Central Queue Algorithm
- Local Queue Algorithm
- Least Queue Algorithm

Chapter 4: Proposed Design

4.1. Architecture Diagram



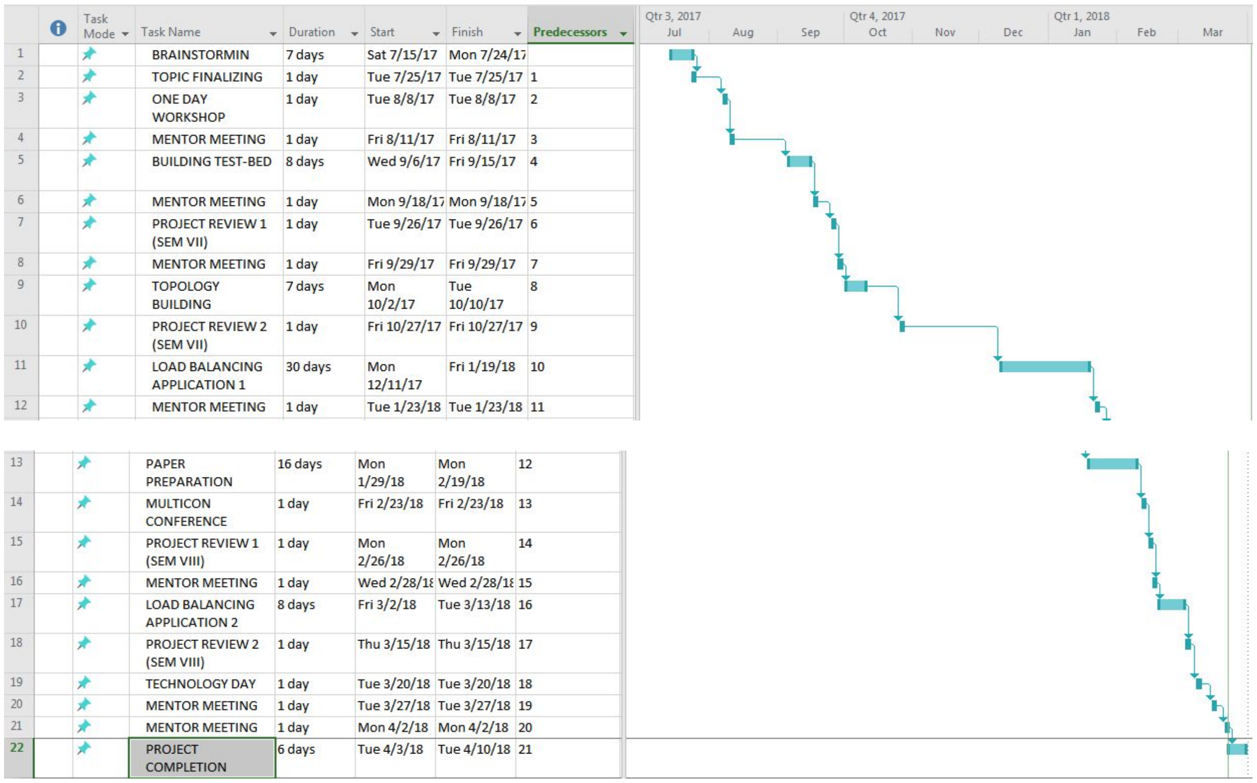
4.2. Block Diagram:



SDN has 3 layers:

1. Infrastructure Layer: It contains the forwarding devices (i.e, Data Plane) which forwards the packets. This Layer lies to the South of the Controller therefore this layer communicates with the Controller using SouthBound APIs (OpenFlow Protocol).
2. Control Layer: This layer is the brain of the SDN. It controls all the forwarding devices.
3. Application Layer: Different applications can be installed on the Controller (i.e, Control Plane) as per the need. This Layer lies to the North of the Controller therefore this layer communicates with the Controller using NorthBound APIs (REST or Java APIs).

4.3. GANTT CHART



	ℹ	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		📌	BRAINSTORMIN	7 days	Sat 7/15/17	Mon 7/24/17	
2		📌	TOPIC FINALIZING	1 day	Tue 7/25/17	Tue 7/25/17	1
3		📌	ONE DAY WORKSHOP	1 day	Tue 8/8/17	Tue 8/8/17	2
4		📌	MENTOR MEETING	1 day	Fri 8/11/17	Fri 8/11/17	3
5		📌	BUILDING TEST-BED	8 days	Wed 9/6/17	Fri 9/15/17	4
6		📌	MENTOR MEETING	1 day	Mon 9/18/17	Mon 9/18/17	5
7		📌	PROJECT REVIEW 1 (SEM VII)	1 day	Tue 9/26/17	Tue 9/26/17	6
8		📌	MENTOR MEETING	1 day	Fri 9/29/17	Fri 9/29/17	7
9		📌	TOPOLOGY BUILDING	7 days	Mon 10/2/17	Tue 10/10/17	8
10		📌	PROJECT REVIEW 2 (SEM VII)	1 day	Fri 10/27/17	Fri 10/27/17	9
11		📌	LOAD BALANCING APPLICATION 1	30 days	Mon 12/11/17	Fri 1/19/18	10
12		📌	MENTOR MEETING	1 day	Tue 1/23/18	Tue 1/23/18	11
13		📌	PAPER PREPARATION	16 days	Mon 1/29/18	Mon 2/19/18	12
14		📌	MULTICON CONFERENCE	1 day	Fri 2/23/18	Fri 2/23/18	13
15		📌	PROJECT REVIEW 1 (SEM VIII)	1 day	Mon 2/26/18	Mon 2/26/18	14
16		📌	MENTOR MEETING	1 day	Wed 2/28/18	Wed 2/28/18	15
17		📌	LOAD BALANCING APPLICATION 2	8 days	Fri 3/2/18	Tue 3/13/18	16
18		📌	PROJECT REVIEW 2 (SEM VIII)	1 day	Thu 3/15/18	Thu 3/15/18	17
19		📌	TECHNOLOGY DAY	1 day	Tue 3/20/18	Tue 3/20/18	18
20		📌	MENTOR MEETING	1 day	Tue 3/27/18	Tue 3/27/18	19
21		📌	MENTOR MEETING	1 day	Mon 4/2/18	Mon 4/2/18	20
22		📌	PROJECT COMPLETION	6 days	Tue 4/3/18	Tue 4/10/18	21

Chapter 5: Implementation Details

5.1. Algorithms for the respective modules developed.

Application 1:

Base Idea: Make use of REST APIs to collect operational information of the topology and its devices.

1. Enable statistics collection in case of Floodlight (TX i.e. Transmission Rate, RX i.e. Receiving Rate, etc). This step is not applicable for OpenDaylight
2. Find information about hosts connected such as their IP, Switch to which they are connected, MAC Addresses, Port mapping, etc
3. Obtain path/route information (using Dijkstra thereby limiting search to shortest paths and only one segment of fat tree topology) from Host1 to Host2 i.e. the hosts between load balancing has to be performed.
4. Find total link cost for all these paths between Host 1 and Host 2. OpenDaylight it gives only transmitted data. So subsequent REST requests are made to compute this. This adds latency to the application (when using OpenDaylight).
5. The flows are created depending on the minimum transmission cost of the links at the given time.
6. Based on the cost, the best path is decided and static flows are pushed into each switch in the current best path. Information such as In-Port, Out-Port, Source IP, Destination IP, Source MAC, Destination MAC is fed to the flows.
7. The program continues to update this information every minute thereby making it dynamic.

Application 2:

Base Idea: Make use of REST APIs to collect operational information of the topology and its devices.

1. Enable statistics collection in case of Floodlight (TX i.e. Transmission Rate, RX i.e. Receiving Rate, etc). This step is not applicable for OpenDaylight
2. Find information about hosts connected such as their IP, Switch to which they are connected, MAC Addresses, Port mapping, etc
3. Obtain path/route information (using A* thereby limiting search to shortest paths and only one segment of fat tree topology) from Host1 to Host2 i.e. the hosts between load balancing has to be performed.
4. Find total link cost for all these paths between Host 1 and Host 2. OpenDaylight it gives only transmitted data. So subsequent REST requests are made to compute this. This adds latency to the application (when using OpenDaylight).
5. The flows are created depending on the minimum transmission cost of the links at the given time.
6. Based on the cost, the best path is decided and static flows are pushed into each switch in the current best path. Information such as In-Port, Out-Port, Source IP, Destination IP, Source MAC, Destination MAC is fed to the flows.
7. The program continues to update this information every minute thereby making it dynamic.

5.2. Comparative Analysis with the existing algorithms

Traditional Load Balancing	**SDN Load Balancing**
1. Path Cost is more between nodes.	1. Path Cost is less between nodes.
2. Response Time is High.	2. Response Time is Low.
3. TurnAround Time is High.	3. TurnAround Time is Low.

4. Throughput is Low.	4. Throughput is High.
5. Implementation Cost is High.	5. Implementation Cost is Low.
6. Low Flexibility.	6. High Flexibility.

6. Testing

Test case 1: Testing the reachability of the Controller

Test case ID	**Test case Name**	**Expected Output**	**Actual Output**	**Test Result**
1	Valid Port Numbers	Connection Established	Connection Established	Pass
2	Invalid Port Numbers	Connection Failed	Connection Failed	Pass

Test case 2: Testing the Hosts

Test case ID	**Test case Name**	**Expected Output**	**Actual Output**	**Test Result**
1	Pingall command was successful	All the hosts are up and running	All the hosts are up and running	Pass
2	Pingall command was unsuccessful	Hosts are down	Hosts are down	Pass

Test case 3: Load Balancing Application

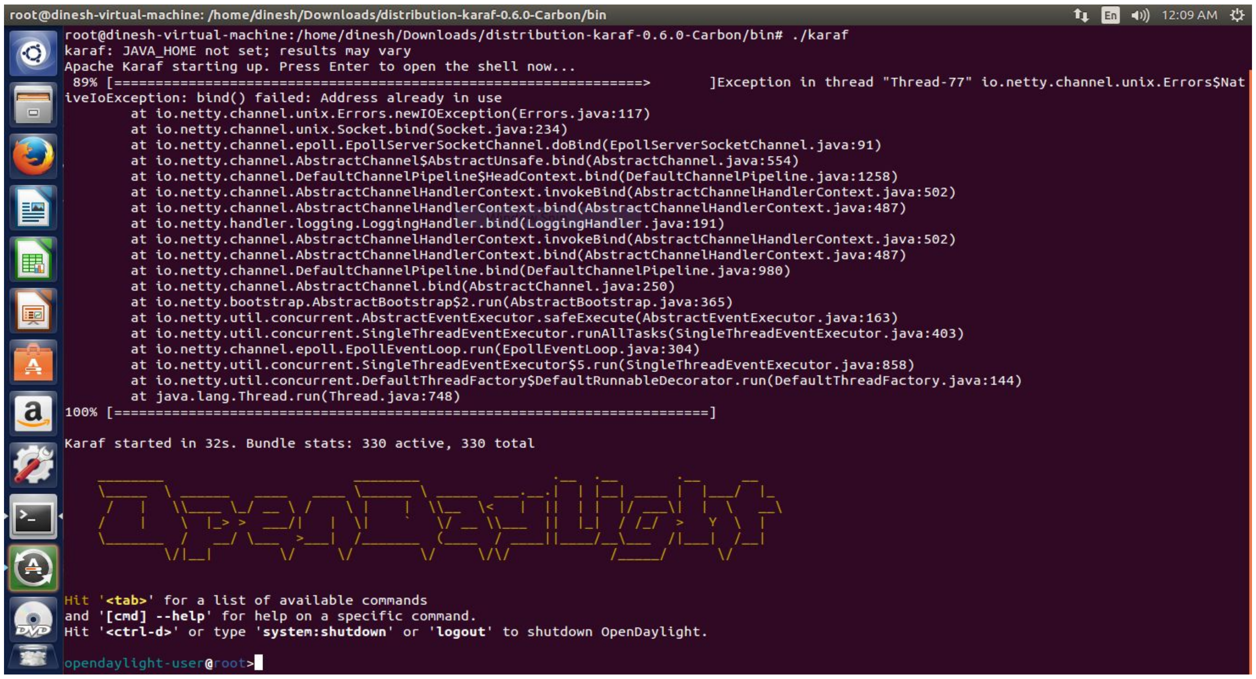
Algorithms	**Test case ID**	**Test case Name**	**Expected Output**	**Actual Output**	**Test Result**
Dijkstra's	**1**	Proper Hosts are Entered	Lightly Loaded Paths Selected	Lightly Loaded Paths Selected	Pass

			and Load is Balanced	and Load is Balanced	
	2	Proper Hosts are not Entered	Load was not Balanced on desired hosts	Load was not Balanced on desired hosts	Pass
A*	**1**	Proper Hosts are Entered	Lightly Loaded Paths Selected and Load is Balanced	Lightly Loaded Paths Selected and Load is Balanced	Pass
	2	Proper Hosts are not Entered	Load was not Balanced on desired hosts	Load was not Balanced on desired hosts	Pass

Chapter 7: Result Analysis

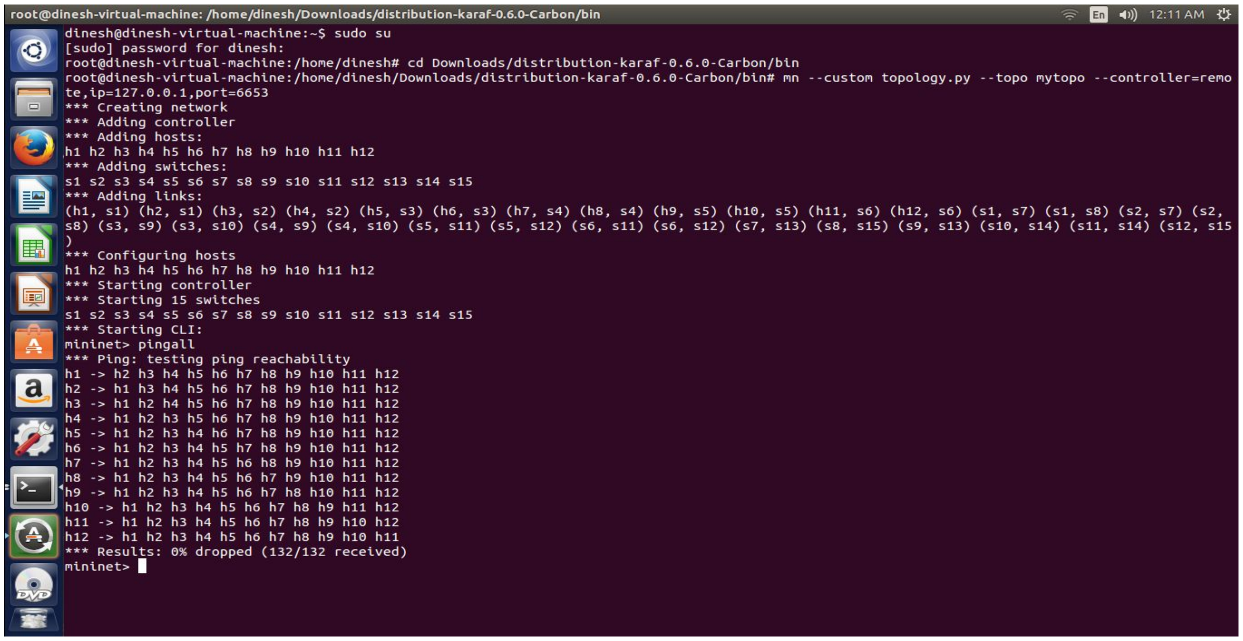
7.1. Simulation Model

OpenDayLight Controller



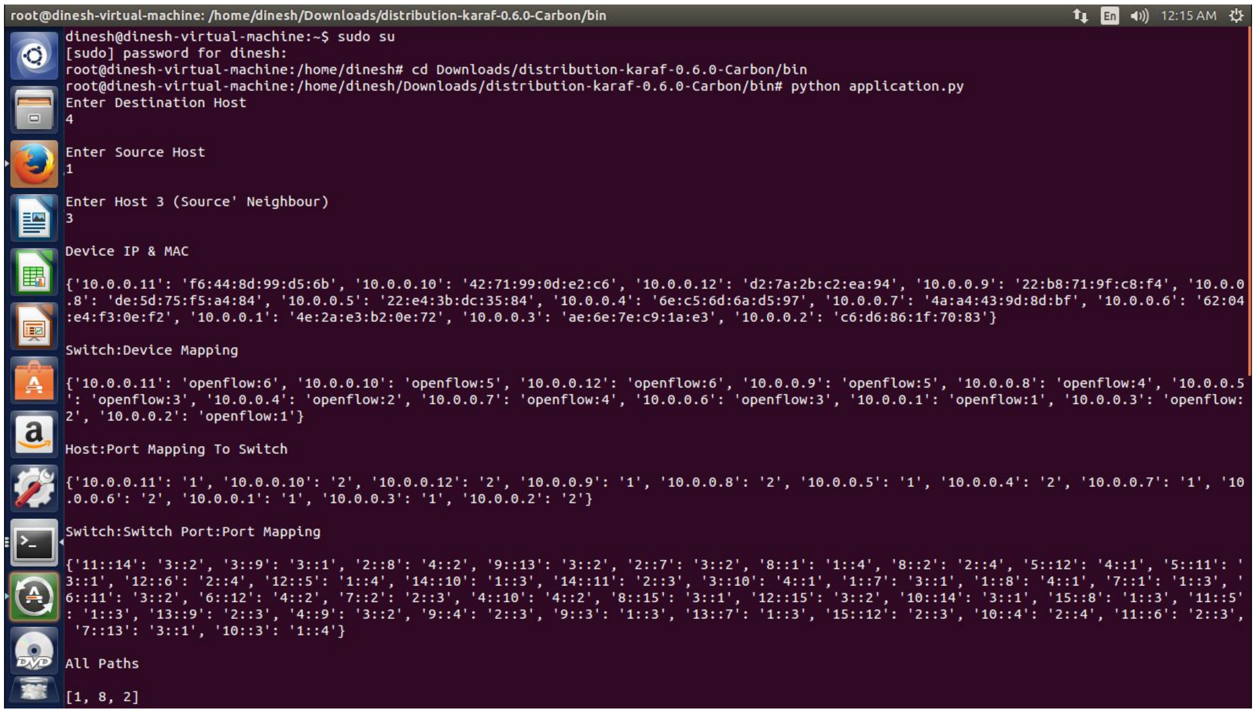
```
root@dinesh-virtual-machine: /home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin          12:09 AM
root@dinesh-virtual-machine:/home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin# ./karaf
karaf: JAVA_HOME not set; results may vary
Apache Karaf starting up. Press Enter to open the shell now...
 89% [=================================================================>       ]Exception in thread "Thread-77" io.netty.channel.unix.Errors$Nat
iveIoException: bind() failed: Address already in use
        at io.netty.channel.unix.Errors.newIOException(Errors.java:117)
        at io.netty.channel.unix.Socket.bind(Socket.java:234)
        at io.netty.channel.epoll.EpollServerSocketChannel.doBind(EpollServerSocketChannel.java:91)
        at io.netty.channel.AbstractChannel$AbstractUnsafe.bind(AbstractChannel.java:554)
        at io.netty.channel.DefaultChannelPipeline$HeadContext.bind(DefaultChannelPipeline.java:1258)
        at io.netty.channel.AbstractChannelHandlerContext.invokeBind(AbstractChannelHandlerContext.java:502)
        at io.netty.channel.AbstractChannelHandlerContext.bind(AbstractChannelHandlerContext.java:487)
        at io.netty.handler.logging.LoggingHandler.bind(LoggingHandler.java:191)
        at io.netty.channel.AbstractChannelHandlerContext.invokeBind(AbstractChannelHandlerContext.java:502)
        at io.netty.channel.AbstractChannelHandlerContext.bind(AbstractChannelHandlerContext.java:487)
        at io.netty.channel.DefaultChannelPipeline.bind(DefaultChannelPipeline.java:980)
        at io.netty.channel.AbstractChannel.bind(AbstractChannel.java:250)
        at io.netty.bootstrap.AbstractBootstrap$2.run(AbstractBootstrap.java:365)
        at io.netty.util.concurrent.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java:163)
        at io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:403)
        at io.netty.channel.epoll.EpollEventLoop.run(EpollEventLoop.java:304)
        at io.netty.util.concurrent.SingleThreadEventExecutor$5.run(SingleThreadEventExecutor.java:858)
        at io.netty.util.concurrent.DefaultThreadFactory$DefaultRunnableDecorator.run(DefaultThreadFactory.java:144)
        at java.lang.Thread.run(Thread.java:748)
100% [========================================================================]

Karaf started in 32s. Bundle stats: 330 active, 330 total

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
```

Fat-tree Topology

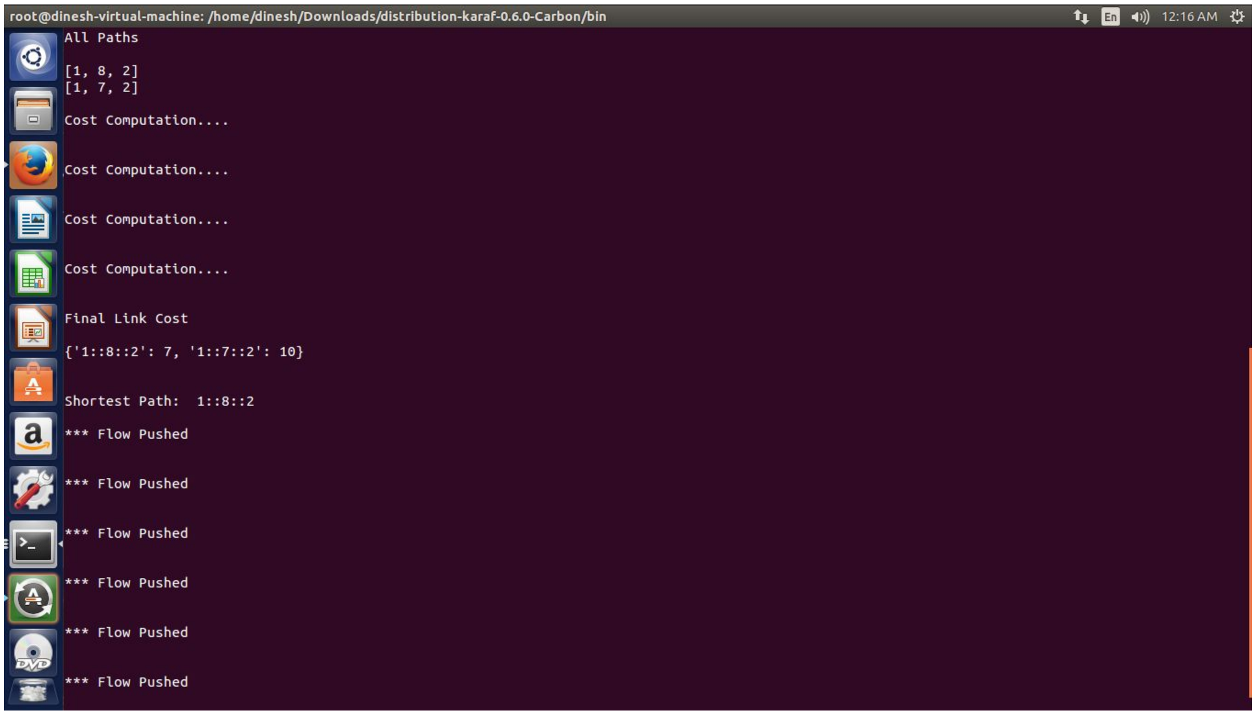


```
root@dinesh-virtual-machine: /home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin          12:11 AM
dinesh@dinesh-virtual-machine:~$ sudo su
[sudo] password for dinesh:
root@dinesh-virtual-machine:/home/dinesh# cd Downloads/distribution-karaf-0.6.0-Carbon/bin
root@dinesh-virtual-machine:/home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin# mn --custom topology.py --topo mytopo --controller=remo
te,ip=127.0.0.1,port=6653
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s3) (h6, s3) (h7, s4) (h8, s4) (h9, s5) (h10, s5) (h11, s6) (h12, s6) (s1, s7) (s1, s8) (s2, s7) (s2,
s8) (s3, s9) (s3, s10) (s4, s9) (s4, s10) (s5, s11) (s5, s12) (s6, s11) (s6, s12) (s7, s13) (s8, s15) (s9, s13) (s10, s14) (s11, s14) (s12, s15
)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Starting controller
*** Starting 15 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11
*** Results: 0% dropped (132/132 received)
mininet>
```

Load Balancing Application (Dijkstra's Algorithm)

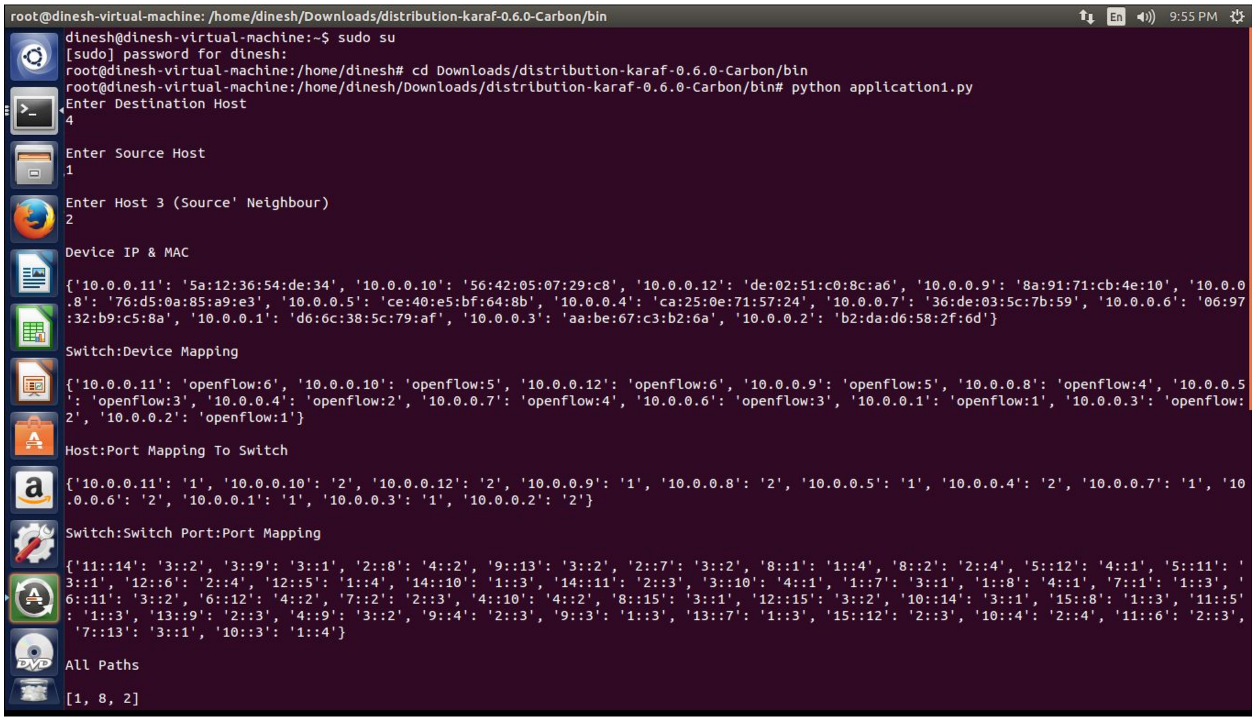


```
root@dinesh-virtual-machine: /home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin                12:15 AM
dinesh@dinesh-virtual-machine:~$ sudo su
[sudo] password for dinesh:
root@dinesh-virtual-machine:/home/dinesh# cd Downloads/distribution-karaf-0.6.0-Carbon/bin
root@dinesh-virtual-machine:/home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin# python application.py
Enter Destination Host
4

Enter Source Host
1

Enter Host 3 (Source' Neighbour)
3

Device IP & MAC

{'10.0.0.11': 'f6:44:8d:99:d5:6b', '10.0.0.10': '42:71:99:0d:e2:c6', '10.0.0.12': 'd2:7a:2b:c2:ea:94', '10.0.0.9': '22:b8:71:9f:c8:f4', '10.0.0
.8': 'de:5d:75:f5:a4:84', '10.0.0.5': '22:e4:3b:dc:35:84', '10.0.0.4': '6e:c5:6d:6a:d5:97', '10.0.0.7': '4a:a4:43:9d:8d:bf', '10.0.0.6': '62:04
:e4:f3:0e:f2', '10.0.0.1': '4e:2a:e3:b2:0e:72', '10.0.0.3': 'ae:6e:7e:c9:1a:e3', '10.0.0.2': 'c6:d6:86:1f:70:83'}

Switch:Device Mapping

{'10.0.0.11': 'openflow:6', '10.0.0.10': 'openflow:5', '10.0.0.12': 'openflow:6', '10.0.0.9': 'openflow:5', '10.0.0.8': 'openflow:4', '10.0.0.5
': 'openflow:3', '10.0.0.4': 'openflow:2', '10.0.0.7': 'openflow:4', '10.0.0.6': 'openflow:3', '10.0.0.1': 'openflow:1', '10.0.0.3': 'openflow:
2', '10.0.0.2': 'openflow:1'}

Host:Port Mapping To Switch

{'10.0.0.11': '1', '10.0.0.10': '2', '10.0.0.12': '2', '10.0.0.9': '1', '10.0.0.8': '2', '10.0.0.5': '1', '10.0.0.4': '2', '10.0.0.7': '1', '10
.0.0.6': '2', '10.0.0.1': '1', '10.0.0.3': '1', '10.0.0.2': '2'}

Switch:Switch Port:Port Mapping

{'11::14': '3::2', '3::9': '3::1', '2::8': '4::2', '9::13': '3::2', '2::7': '3::2', '8::1': '1::4', '8::2': '2::4', '5::12': '4::1', '5::11': '
3::1', '12::6': '2::4', '12::5': '1::4', '14::10': '1::3', '14::11': '2::3', '3::10': '4::1', '1::7': '3::1', '1::8': '4::1', '7::1': '1::3', '
6::11': '3::2', '6::12': '4::2', '7::2': '2::3', '4::10': '4::2', '8::15': '3::1', '12::15': '3::2', '10::14': '3::1', '15::8': '1::3', '11::5'
: '1::3', '13::9': '2::3', '4::9': '3::2', '9::4': '2::3', '9::3': '1::3', '13::7': '1::3', '15::12': '2::3', '10::4': '2::4', '11::6': '2::3',
 '7::13': '3::1', '10::3': '1::4'}

All Paths

[1, 8, 2]
```

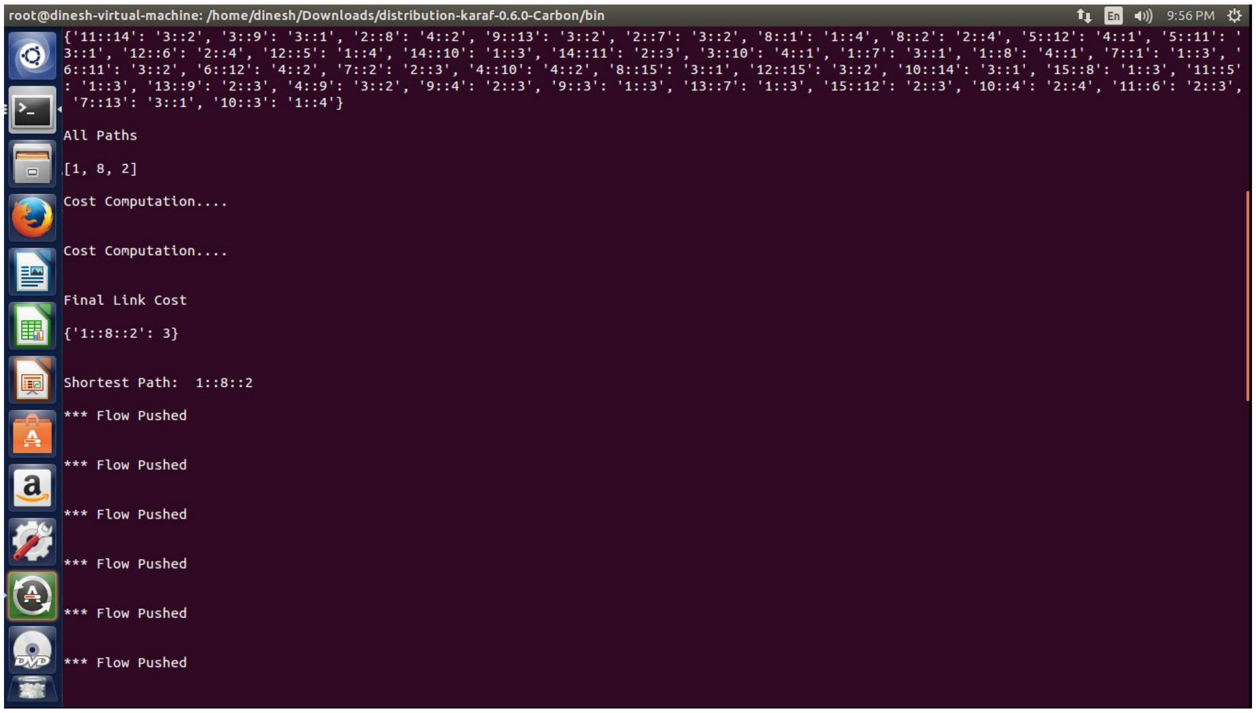


```
root@dinesh-virtual-machine: /home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin                12:16 AM
All Paths

[1, 8, 2]
[1, 7, 2]

Cost Computation....


Cost Computation....


Cost Computation....


Cost Computation....


Final Link Cost

{'1::8::2': 7, '1::7::2': 10}


Shortest Path:  1::8::2

*** Flow Pushed


*** Flow Pushed


*** Flow Pushed


*** Flow Pushed


*** Flow Pushed


*** Flow Pushed
```

Load Balancing Application (A-star Algorithm)



```
dinesh@dinesh-virtual-machine:~$ sudo su
[sudo] password for dinesh:
root@dinesh-virtual-machine:/home/dinesh# cd Downloads/distribution-karaf-0.6.0-Carbon/bin
root@dinesh-virtual-machine:/home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin# python application1.py
Enter Destination Host
4

Enter Source Host
1

Enter Host 3 (Source' Neighbour)
2

Device IP & MAC

{'10.0.0.11': '5a:12:36:54:de:34', '10.0.0.10': '56:42:05:07:29:c8', '10.0.0.12': 'de:02:51:c0:8c:a6', '10.0.0.9': '8a:91:71:cb:4e:10', '10.0.0
.8': '76:d5:0a:85:a9:e3', '10.0.0.5': 'ce:40:e5:bf:64:8b', '10.0.0.4': 'ca:25:0e:71:57:24', '10.0.0.7': '36:de:03:5c:7b:59', '10.0.0.6': '06:97
:32:b9:c5:8a', '10.0.0.1': 'd6:6c:38:5c:79:af', '10.0.0.3': 'aa:be:67:c3:b2:6a', '10.0.0.2': 'b2:da:d6:58:2f:6d'}

Switch:Device Mapping

{'10.0.0.11': 'openflow:6', '10.0.0.10': 'openflow:5', '10.0.0.12': 'openflow:6', '10.0.0.9': 'openflow:5', '10.0.0.8': 'openflow:4', '10.0.0.5
': 'openflow:3', '10.0.0.4': 'openflow:2', '10.0.0.7': 'openflow:4', '10.0.0.6': 'openflow:3', '10.0.0.1': 'openflow:1', '10.0.0.3': 'openflow:
2', '10.0.0.2': 'openflow:1'}

Host:Port Mapping To Switch

{'10.0.0.11': '1', '10.0.0.10': '2', '10.0.0.12': '2', '10.0.0.9': '1', '10.0.0.8': '2', '10.0.0.5': '1', '10.0.0.4': '2', '10.0.0.7': '1', '10
.0.0.6': '2', '10.0.0.1': '1', '10.0.0.3': '1', '10.0.0.2': '2'}

Switch:Switch Port:Port Mapping

{'11::14': '3::2', '3::9': '3::1', '2::8': '4::2', '9::13': '3::2', '2::7': '3::2', '8::1': '1::4', '8::2': '2::4', '5::12': '4::1', '5::11': '
3::1', '12::6': '2::4', '12::5': '1::4', '14::10': '1::3', '14::11': '2::3', '3::10': '4::1', '1::7': '3::1', '1::8': '4::1', '7::1': '1::3', '
6::11': '3::2', '6::12': '4::2', '7::2': '2::3', '4::10': '4::2', '8::15': '3::1', '12::15': '3::2', '10::14': '3::1', '15::8': '1::3', '11::5'
: '1::3', '13::9': '2::3', '4::9': '3::2', '9::4': '2::3', '9::3': '1::3', '13::7': '1::3', '15::12': '2::3', '10::4': '2::4', '11::6': '2::3',
 '7::13': '3::1', '10::3': '1::4'}

All Paths

[1, 8, 2]
```

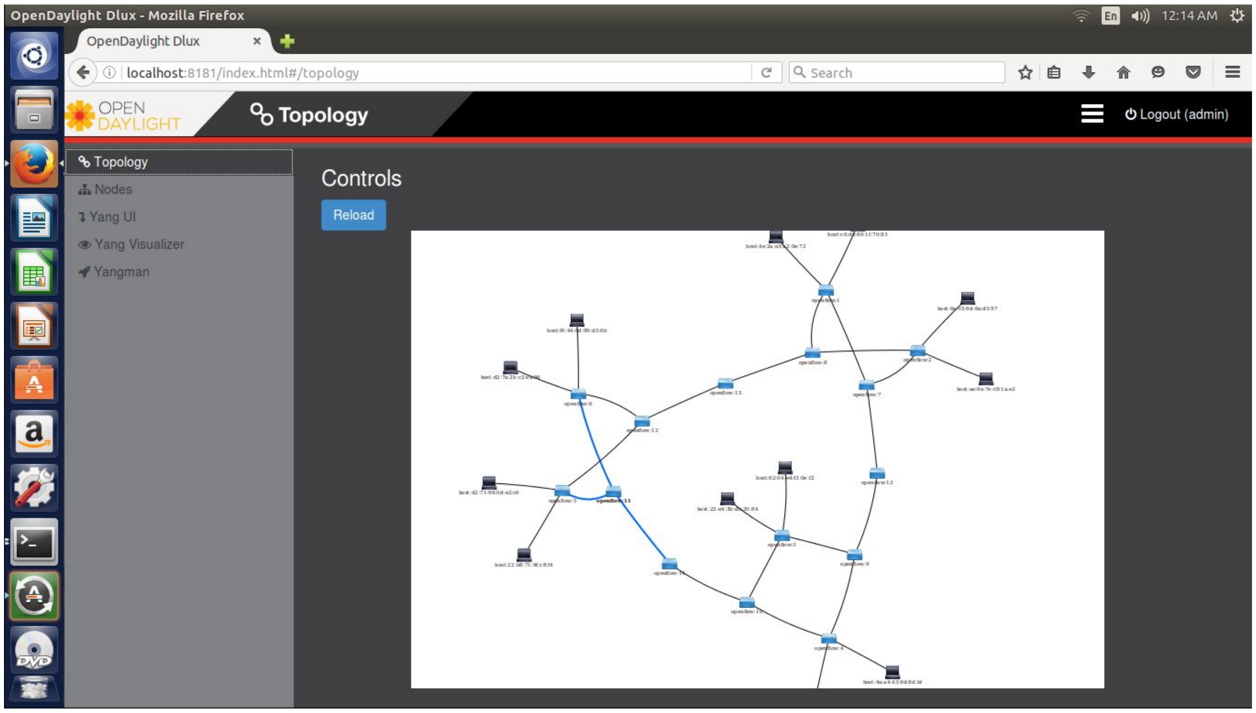


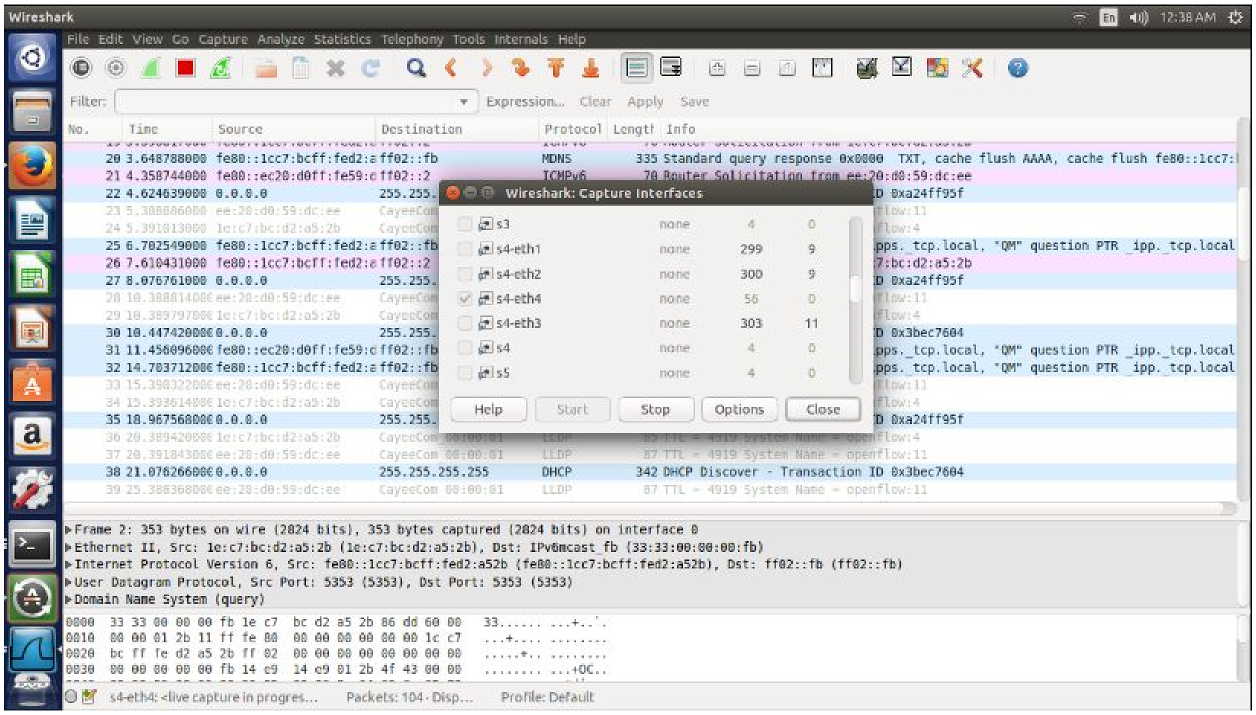
```
{'11::14': '3::2', '3::9': '3::1', '2::8': '4::2', '9::13': '3::2', '2::7': '3::2', '8::1': '1::4', '8::2': '2::4', '5::12': '4::1', '5::11': '
3::1', '12::6': '2::4', '12::5': '1::4', '14::10': '1::3', '14::11': '2::3', '3::10': '4::1', '1::7': '3::1', '1::8': '4::1', '7::1': '1::3', '
6::11': '3::2', '6::12': '4::2', '7::2': '2::3', '4::10': '4::2', '8::15': '3::1', '12::15': '3::2', '10::14': '3::1', '15::8': '1::3', '11::5'
: '1::3', '13::9': '2::3', '4::9': '3::2', '9::4': '2::3', '9::3': '1::3', '13::7': '1::3', '15::12': '2::3', '10::4': '2::4', '11::6': '2::3',
 '7::13': '3::1', '10::3': '1::4'}

All Paths

[1, 8, 2]

Cost Computation....


Cost Computation....


Final Link Cost

{'1::8::2': 3}

Shortest Path:  1::8::2

*** Flow Pushed


*** Flow Pushed


*** Flow Pushed


*** Flow Pushed


*** Flow Pushed


*** Flow Pushed
```

7.2. Parameters considered

- Scalability
- Number of users
- Throughput
- Performance

7.3. Screenshots of User Interface




Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save
Wireshark: Capture Interfaces
Help Start Stop Options Close
12:38 AM

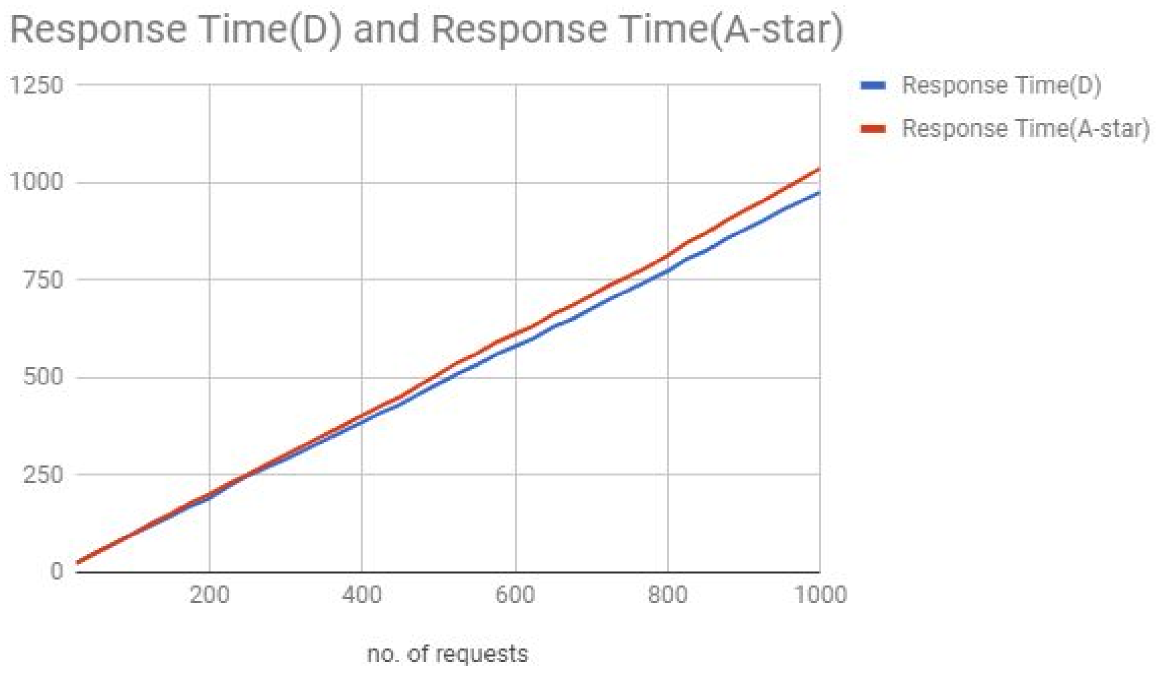
```
No.   Time         Source                 Destination        Protocol Length Info
   20 3.648788000  fe80::1cc7:bcff:fed2:a ff02::fb           MDNS       335 Standard query response 0x0000  TXT, cache flush AAAA, cache flush fe80::1cc7:
   21 4.358744000  fe80::ec20:d0ff:fe59:c ff02::2            ICMPv6      70 Router Solicitation from ee:20:d0:59:dc:ee
   22 4.624639000  0.0.0.0                255.255.                          D 0xa24ff95f
   23 5.388686000  ee:20:d0:59:dc:ee      CayeeCom                          flow:11
   24 5.391013000  1e:c7:bc:d2:a5:2b      CayeeCom                          flow:4
   25 6.702549000  fe80::1cc7:bcff:fed2:a ff02::fb                          pps._tcp.local, "QM" question PTR _ipp._tcp.local
   26 7.610431000  fe80::1cc7:bcff:fed2:a ff02::2                           7:bc:d2:a5:2b
   27 8.076761000  0.0.0.0                255.255.                          D 0xa24ff95f
   28 10.38881400€ ee:20:d0:59:dc:ee      CayeeCom                          flow:11
   29 10.38979700€ 1e:c7:bc:d2:a5:2b      CayeeCom                          flow:4
   30 10.44742000€ 0.0.0.0                255.255.                          D 0x3bec7604
   31 11.45609600€ fe80::ec20:d0ff:fe59:c ff02::fb                          pps._tcp.local, "QM" question PTR _ipp._tcp.local
   32 14.70371200€ fe80::1cc7:bcff:fed2:a ff02::fb                          pps._tcp.local, "QM" question PTR _ipp._tcp.local
   33 15.39832200€ ee:20:d0:59:dc:ee      CayeeCom                          flow:11
   34 15.39361400€ 1e:c7:bc:d2:a5:2b      CayeeCom                          flow:4
   35 18.96756800€ 0.0.0.0                255.255.                          D 0xa24ff95f
   36 20.38942000€ 1e:c7:bc:d2:a5:2b      CayeeCom_00:00:01  LLDP        85 TTL = 4919 System Name = openflow:4
   37 20.39184300€ ee:20:d0:59:dc:ee      CayeeCom 00:00:01  LLDP        87 TTL = 4919 System Name = openflow:11
   38 21.07626600€ 0.0.0.0                255.255.255.255    DHCP       342 DHCP Discover - Transaction ID 0x3bec7604
   39 25.38836800€ ee:20:d0:59:dc:ee      CayeeCom 00:00:01  LLDP        87 TTL = 4919 System Name = openflow:11
```

```
☐ s3        none    4    0
☐ s4-eth1   none  299    9
☐ s4-eth2   none  300    9
☑ s4-eth4   none   56    0
☐ s4-eth3   none  303   11
☐ s4        none    4    0
☐ s5        none    4    0
```

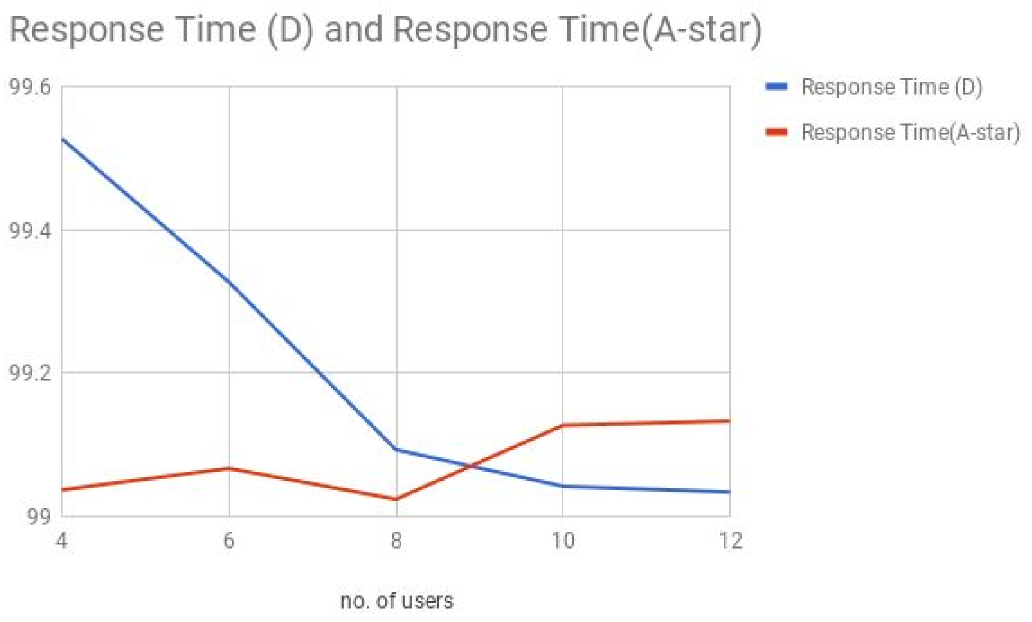
```
▶Frame 2: 353 bytes on wire (2824 bits), 353 bytes captured (2824 bits) on interface 0
▶Ethernet II, Src: 1e:c7:bc:d2:a5:2b (1e:c7:bc:d2:a5:2b), Dst: IPv6mcast_fb (33:33:00:00:00:fb)
▶Internet Protocol Version 6, Src: fe80::1cc7:bcff:fed2:a52b (fe80::1cc7:bcff:fed2:a52b), Dst: ff02::fb (ff02::fb)
▶User Datagram Protocol, Src Port: 5353 (5353), Dst Port: 5353 (5353)
▶Domain Name System (query)
0000  33 33 00 00 00 fb 1e c7  bc d2 a5 2b 86 dd 60 00   33...... ...+..`.
0010  00 00 01 2b 11 ff fe 80  00 00 00 00 00 00 1c c7   ...+.... ........
0020  bc ff fe d2 a5 2b ff 02  00 00 00 00 00 00 00 00   .....+.. ........
0030  00 00 00 00 00 fb 14 e9  14 e9 01 2b 4f 43 00 00   ........ ...+OC..
s4-eth4: <live capture in progres...   Packets: 104 · Disp...   Profile: Default
```

7.4. Graphical outputs

Performance Graph



Scalability Graph


Response Time (D) and Response Time(A-star)
99.6
99.4
99.2
99
4
6
8
10
12
no. of users
Response Time (D)
Response Time(A-star)

Chapter 8: Conclusion

1. Limitation

Centralized architecture of SDN based networks itself is a huge challenge.Other challenges of immature code base, lack of features, lack of support etc are temporary disadvantages which will go away with time.

However, SDN based networks will always have a centralized architecture because you want to pull out the intelligence from the boxes and move it to an application on the controller. One controller manages the network. Becomes a single point of failure so you add clustering and HA support. Now 5 controllers function in a cluster to support a network. These 5 will only support anywhere between 200-2000 devices. As you add elements to the network you need to add more controllers. Now these different clusters need to work together to manage your network. So you keep on adding intelligence through other nodes which do not reside on the device. Internet/Networks inherently is distributed in nature. You take away the flavor of distributed architecture once you remove all the intelligence from these devices.

SDN has a problematic future as it has issues to overcome. The first issue is the console/remote capability with today's security issues many will not want to expose their network to a potential hacker takeover. It is an Open Source Technology.

Another issue is the current state of Network Management policies and practices with single device or single path focus. When a Network Manager looks at SDN he only sees how it can help or hurt his network but SDN is much bigger and a lot of education still remains to be done to make SDN or similar technologies palatable.

SDN is a Human Centric Technology where today's technology is Device Centric which is and always will be a challenge to get managers to adopt especially when one person can completely change your network, storage, WAN, etc. fabric.

Many talk about SDN's ability to help with the "Cloud" but before we get SDN involved we need to get the "Cloud" under control.

2. Conclusion

SDN has brought about a revolution in the field of Networking because now the brain of the devices i.e., the Control Plane is separated from Data Plane. Developers can now deploy their own applications on the Controller that makes the Networks more flexible. SDN is considered to be the future of Networking because SDN provides many functions to Developers over Traditional Networks. Even intelligent applications can be programmed and deployed on the Controller to make the network intelligent which was not possible in traditional networks.

Due to very small topology the results here may not seem very promising but as the network grows, SDN proves to be the best solution.

Load Balancing using SDN is the most efficient way to monitor and balance the load between Clients and Servers. The Developer only needs to built an efficient application and deploy that application on the Controller and the Application will do its job.

3. Future Scope

Looking at the current trends and increasing demands in the service sector, Traditional Networks do not satisfy the needs of service sector in terms of speed. SDN is very fast as compared to Traditional ones because of simplicity of it's architecture.

SDN is very flexible as the same architecture (Control Plane and Data Plane) can be used for various applications. The developers need an efficient algorithms for the applications which they want to deploy on the Controller then the Controller and the application will communicate to perform its operation. Even Intelligent Applications can be programmed and deployed on the Controller to make the network more flexible and intelligent which was not possible in Traditional Networks.

SDN brings Software Virtualizations to the Networks which was not in Traditional Networks because in Traditional Networks the Router, Switches, Hubs, etc. behave like one applications and not like flexible model. So SDN can revolutionize this aspect of the Networks.

Scalability of the Networks is much more simple in SDN as compared to Traditional Ones.

9. Appendix

9.1. List of Figures

9.2. Paper Publications

9.2.1. Draft of the Paper

A Comparative Study of Load Balancing using SDN and Traditional System

Vikas Kukreja
Department of Computer Engineering,
Vivekanand Education Society Institute of Technology,
Chembur, India
vikas.kukreja@ves.ac.in

Dinesh Panchi
Department of Computer Engineering,
Vivekanand Education Society Institute of Technology,
Chembur, India
dinesh.panchi@ves.ac.in

Jatin Sajnani
Department of Computer Engineering,
Vivekanand Education Society Institute of Technology,
Chembur, India
jatin.sajnani@ves.ac.in

Hitesh Seedani
Department of Computer Engineering,
Vivekanand Education Society Institute of Technology,
Chembur, India
hitesh.seedani@ves.ac.in

Dr.(Mrs).Nupur Giri
Department of Computer Engineering,
Vivekanand Education Society Institute of Technology,
Chembur, India
nupur.giri@ves.ac.in

***Abstract*--With the growth and advancement in Industrialisation there is increase in the service requirement, due to this the load on the server increases and this may result in latency/delay or sometimes the link failure or server crash. To avoid this latency or link failure or server crash there must be a Load Balancer between the client and server to handle this load. By using SDN, a topology can be created and a Load Balancer Application can be deployed on the Controller that will monitor the load on server, links between the server controller and clients. ODL Controller and Mininet are used to display the topology and simulate the topology created. The Load Balancer application will calculate the threshold value dynamically and will compare the results obtained from both traditional system and SDN using certain parameters. And Plot the graphs for same.**

Keywords: SDN, Load Balancing, ODL, Mininet.
General Terms: Link failure, Graphs.

I. Introduction

A. Introduction

Software-defined networking (SDN) is a networking technology that brings programmability in a network by separating control plane from data plane and deploying a single controller (commonly known as SDN controller) to control flow of data packets among data plans.

SDN Architecture has 3 Layers:

I. Infrastructure Layer.
II. Control Layer.
III. Application layer.

Control Layer is the brain of the SDN, controlling of different Data Planes is done by this layer. Applications are installed in the Application Layer, over the Control Layer and the communication between Controller and Application is done using the NorthBound APIs (Eg. Java APIs or REST APIs) and that of between Controller and Switches using SouthBound APIs (Eg. OpenFlow Protocol). If there are more than one Controller than the Controllers can communicate with each other using EastBound and WestBound APIs.

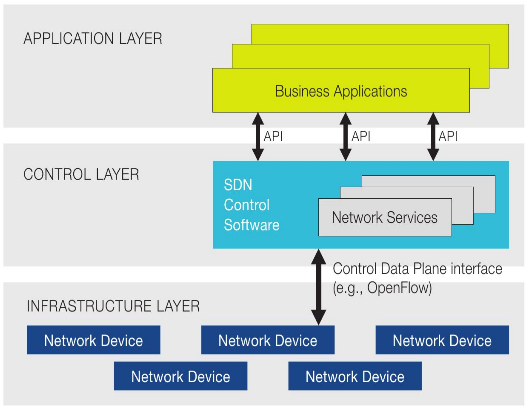


Fig 1. Architecture of SDN

B. Motivation for the project

Software defined networking (SDN) aims to simplify network management by removing the control plane from switches and running custom control applications at a logically central controller. It is a challenging task to code an application which can detect loops or black holes in network. Reliability, Consistency and High-quality products are provided by SDN to all users.

Today's IT leaders are focused on getting more benefits and improving the business agility while lowering the cost of running IT technology. Achieving this is challenging task as IT environment has become complex.SDN technology can help the IT leaders to control cost and boost worker productivity.

With traditional networks, each location requires dedicated devices to provide the services required.For this each location should have their own set of devices for eg router, Firewall etc. SDN can run multiple services on a single hardware platform. This will reduce the consumption of time and hardware as well as operational costs.

C. Problem Definition

With the growth and improvement in Industrialisation there is increase in the service requirement due to this the load on the server increases and this may result in server crashes or link failure.

To avoid this latency or link failure or server crash this paper will install a load balancer between the client and server to handle this load.

By using SDN a topology is created and Load Balancer application is installed on the Controller that will monitor the load on server as well as the links between the server and controller and controller and clients. Load Balancer Application calculates Threshold value dynamically using 2-Threshold policy to transfer processes between the nodes, and Altruistic Priority assignment policy is used in Under loaded Conditions to prioritize the clash between Local and Remote Process.

II. CONCEPT AND METHODOLOGY

A. Concept

Software-defined networking (SDN) is a networking technology that brings programmability in a network by separating control plane from data plane and deploying a single controller (commonly known as SDN controller) to control flow of data packets among data planes. By using SDN, this paper creates a custom topology and installed a Load Balancer application on the Controller that will monitor the load on server and also on the links between the server, controller and clients. Load Balancer Application uses 2-Threshold policy to transfer processes between the nodes where Threshold is calculated Dynamically and Altruistic Priority assignment policy for migration of processes from one node to another or transferring via some other links. This paper provides a solution for controlling the load in this era of increasing industrialization.

Mininet is a tool which is used as network simulator for openflow and SDN applications. It is a software program which allows an entire network consisting of virtual hosts, controllers. switches, and links to be created and emulated on single PC. Mininet uses lightweight virtualization. Minnet connects hosts and switches using virtual ethernet pairs. New network applications such as firewall . load balancer. Can be developed and tested on Mininet. The same application code can be moved to another infrastructure.

It emulates the network using :

A. Command Line Interface.
B. Interactive User Interface.
C. Python Application.

Mininet has few inbuilt topologies:

A. Simple.
B. Linear.
C. Tree.
D. Minimal.
E. Custom.

B. *Methodology*

1. Run OpenDaylight SDN Controller using command:

   ./karaf

2. Run the custom topology using Mininet:

   sudo mn --custom topology.py --topo mytopo --controller=remote,ip=127.0.0.1,port=6653

3. Run the pingall command to ensure all hosts are up and running.
4. Check the GUI of OpenDaylight using link:

   localhost.8181/index.html#/topology

5. Ping any two hosts to get real simulation of data transmission between nodes.
6. Run the load balancing application using command:

   python odl.py

7. Python file will run the application for load balancing between client and server, application uses Altruistic Priority Assignment Policy and 2-threshold policy for deciding the workload of node.
8. Node can be in any of the following state
   a. Overloaded
   b. Normal
   c. Underloaded
9. Dynamically threshold value is calculated and the load balancing application will handle the load based on the node states described in the previous step.

III. IMPLEMENTATION

1. **Creation of Topology :** This paper had considered 12 hosts named as Host 1 to Host 2 and 15 switches named as switch 1 to switch 15 which are connected as shown in *Fig 2* below.

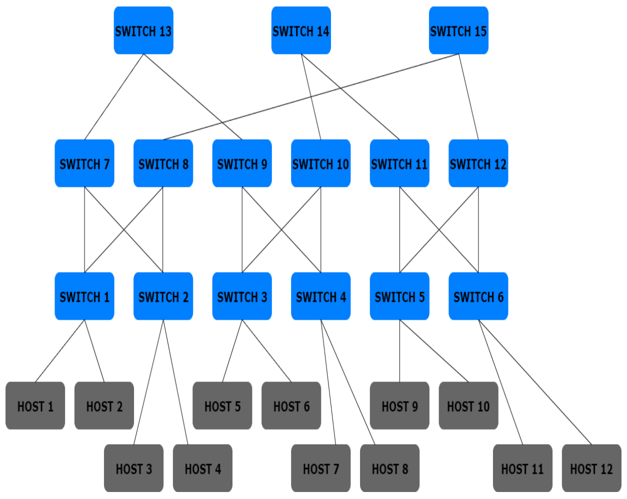


Fig 2 . Topology.

2. **Finding the Paths:** find all the possible paths from destination to source along with load on each path. There can be more than one path to reach the destination from source.

3. **Selecting the Shortest path:** Out of all the paths select the one with the least load.Packets will be transferred through this path from source to destination.

4. **Dynamically Change Of Path :** As the load on paths depends upon the topology created and the packets present in the topology. As the time flies and number of packets increases it might be possible that path selected as shortest may have more load than the alternative path which was rejected first, It will automatically switch to the path with least load.

IV. PARAMETERS

1. Overload Rejection: In situations where node is Overloaded, certain measures need to be taken in order to balance the links. Overload rejection measures are stopped when node which is overload situation terminates and after a short interval Load Balancing is also closed down.

2. Process Migration : This factor decides when does a system should transfer a process based on threshold value used in algorithms.

3. Reliability: In case of failure, this factor determines the reliability of Static and Dynamic Algorithms. Static load balancing algorithms distribute the workload in the start based on their performance evaluation and therefore it can not transfer the workload to other machine in case of failure occurs at run time. While dynamic algorithms distribute the workload at run time and therefore it can balance the load in case of system failure.

4. 2-Threshold Policy: The two threshold values (upper and lower) of various parameters (CPU Queue Length, CPU Utilization and Memory Utilization) are computed using multiplication of average load of every parameter and a constant value.

$$tH = H*Lavg$$
$$tL = L*Lavg$$

Where, tH is high threshold, tL is low threshold, H and L are constants(calculated based on average load).

5. There are different parameters on which this paper will compare the Load Balancing using Traditional and SDN:

a. Throughput
b. Threshold
c. Number of Users
d. Process Migration
e. Response Time
f. Waiting Time
g. Robustness
h. Scalability

The graph of above parameters can be plot for different loads on a single node.

V. Proposed Graph

1. Load vs Throughput for Threshold parameter.
2. Number of Users vs Throughput for Number of Users.
3. Load vs Number of Users for Scalability and Robustness.
4. Load vs Threshold for Process Migration.

VI. Conclusion

SDN has brought about a revolution in the field of Networking because now the brain of the devices i.e., the Control Plane is separated from Data Plane. Flexibility is achieved in the Networks as Developers can now develop and deploy their own Applications on the Controller. The reason SDN is called Future of Networking is because SDN provides lots of functionalities to the Developers and Network Administrators. Even intelligent applications can be programmed and deployed on the Controller to maintain the network intelligent which was not possible in traditional networks.

Due to very small topology the results here may not seem promising but as the network grows, SDN proves to be the best solution.

Load Balancing using SDN is the most efficient way to monitor and balance the load between Clients and Servers. The overhead now remains with the Developers is to develop an efficient Application as per the requirements and deploy that application on the Controller and the Application will perform its function.

References

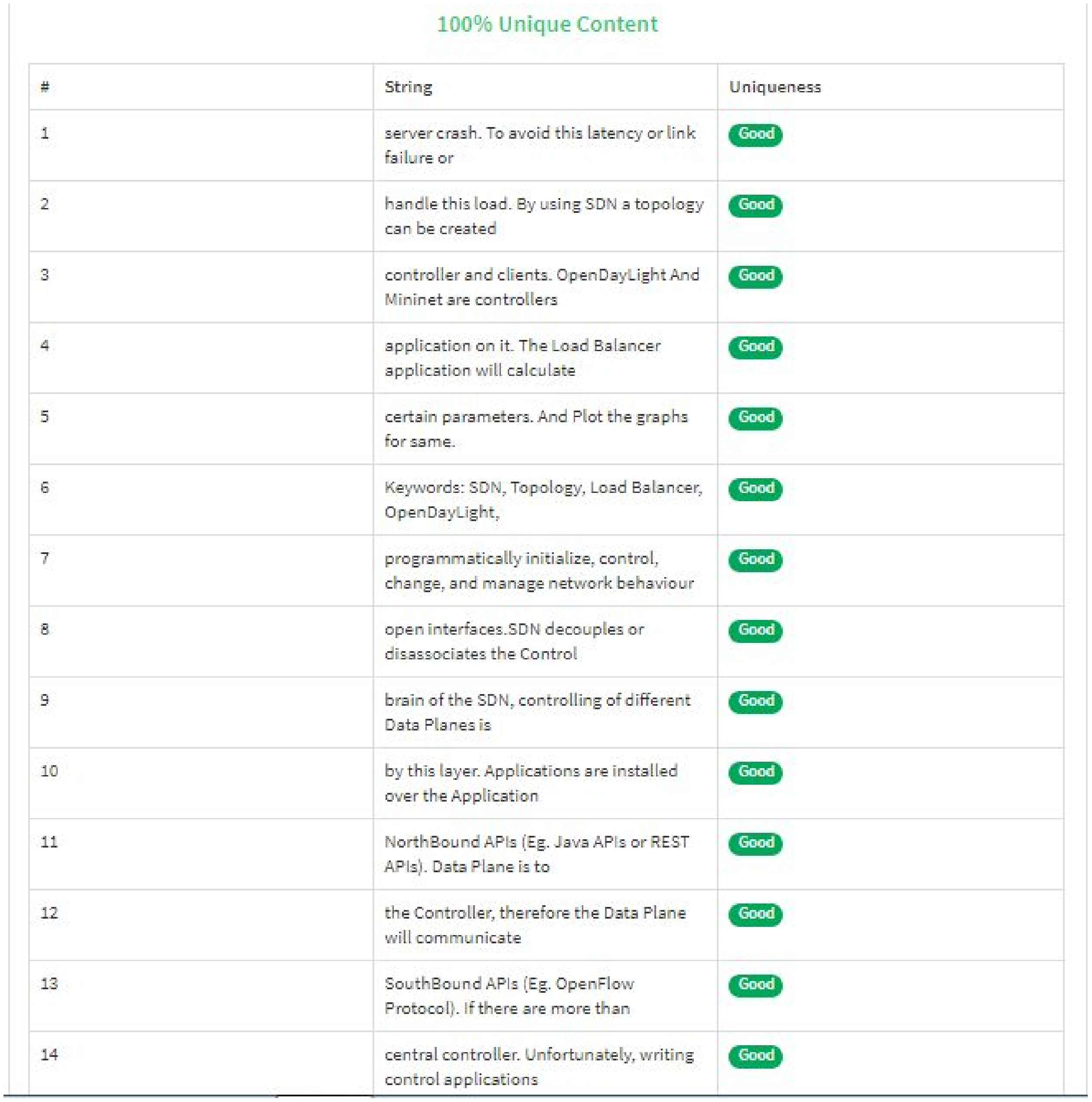
[1] Mr. Marlon Esteban Olaya, Mr. Ivan Bernal, Mr. David Mejia, “Application of Load Balancing in SDN”, 10.1109/EATIS.2016.7520102.

[2] Mr. Mohammad Mousa, Mr. Ayman .Bahaa-Eldin, Mr. Mohamed Sobh, “Software Defined Networking concepts and challenges”, 10.1109/ICCES.2016.7821979.

[3] Mr. Alexander Gelberger, Mr. Niv Yemini, Mr. Ran Giladi, “Performance Analysis of Software Defined Networking(SDN)”, 10.1109/MASCOTS.2013.58.

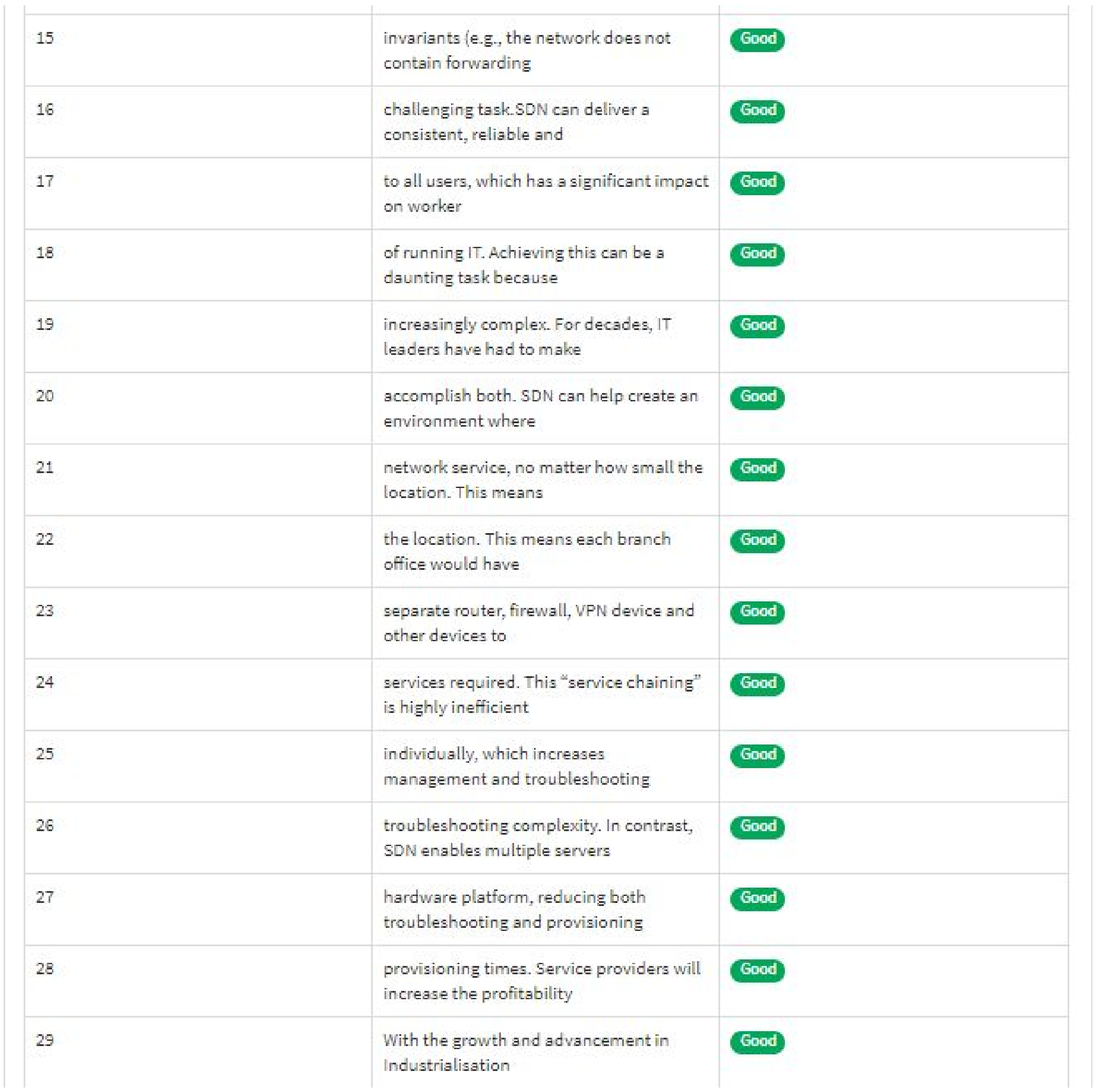
[4] Mr. G. Araniti, Mr. J. Cosmas, Mr. A. Lera, “OpenFlow over wireless networks: Performance analysis”, 10.1109/BMSB.2014.6873559.

9.2.2. Plagiarism Report of the Paper

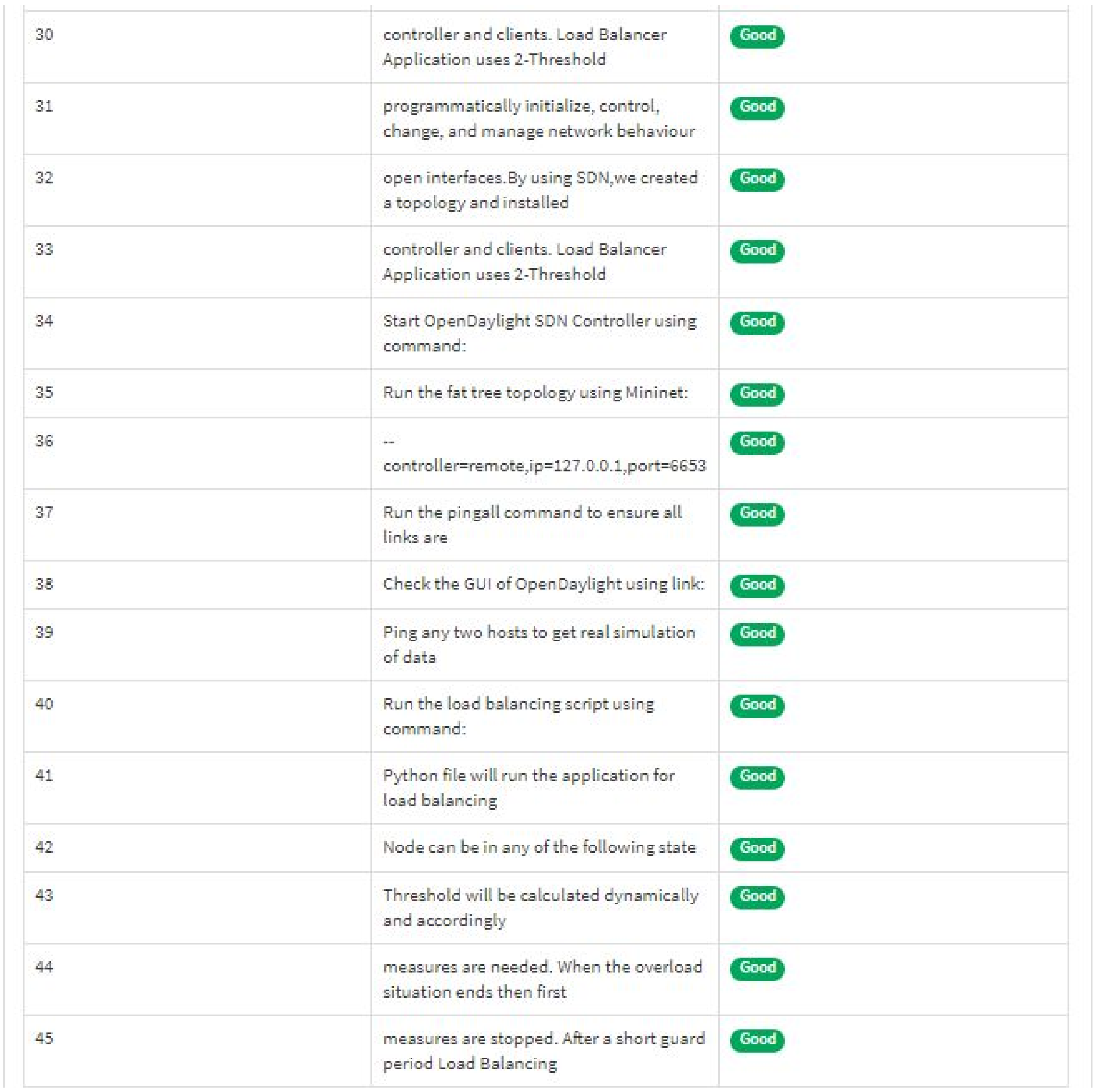


100% Unique Content

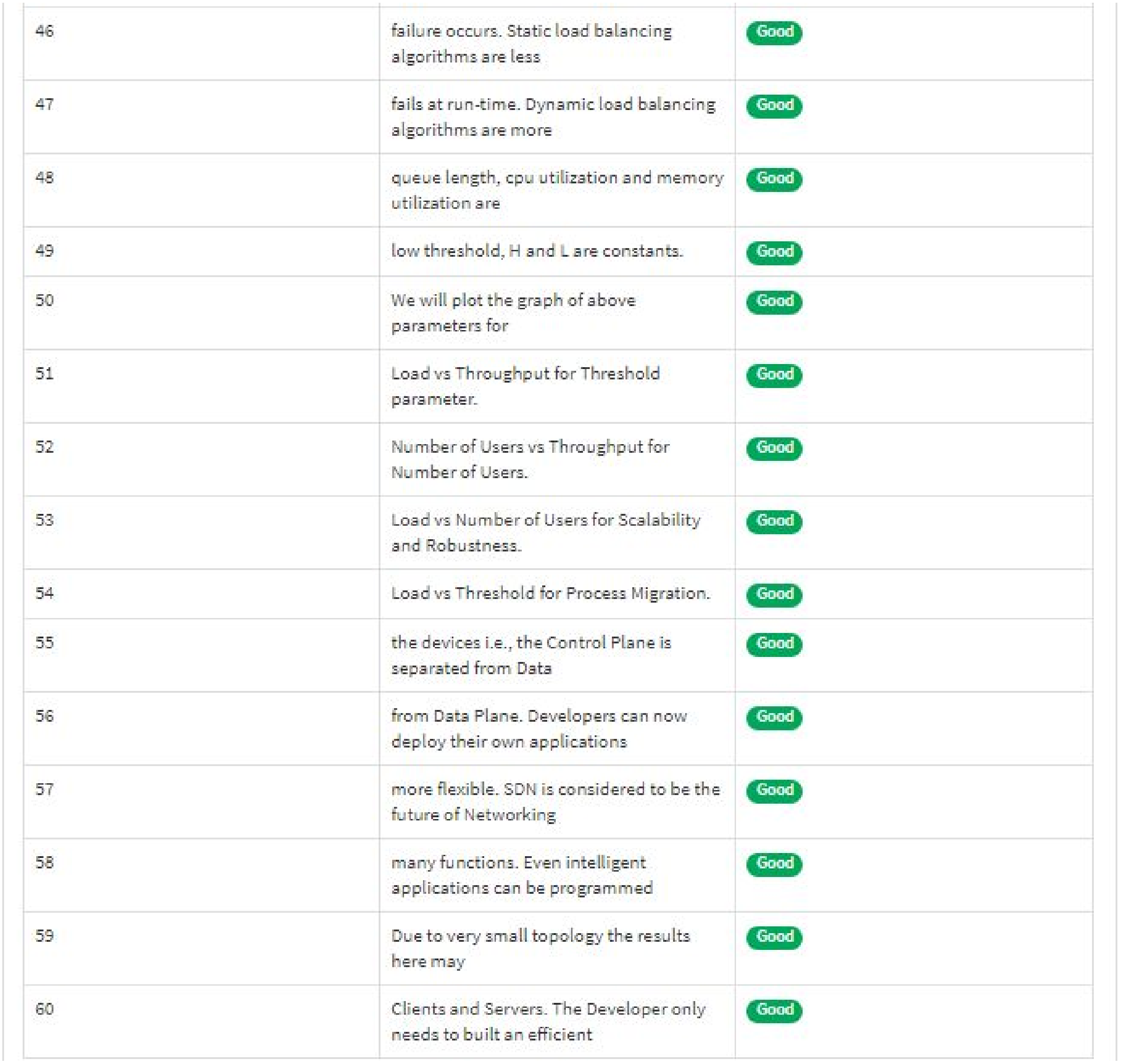
#	String	Uniqueness
1	server crash. To avoid this latency or link failure or	Good
2	handle this load. By using SDN a topology can be created	Good
3	controller and clients. OpenDayLight And Mininet are controllers	Good
4	application on it. The Load Balancer application will calculate	Good
5	certain parameters. And Plot the graphs for same.	Good
6	Keywords: SDN, Topology, Load Balancer, OpenDayLight,	Good
7	programmatically initialize, control, change, and manage network behaviour	Good
8	open interfaces.SDN decouples or disassociates the Control	Good
9	brain of the SDN, controlling of different Data Planes is	Good
10	by this layer. Applications are installed over the Application	Good
11	NorthBound APIs (Eg. Java APIs or REST APIs). Data Plane is to	Good
12	the Controller, therefore the Data Plane will communicate	Good
13	SouthBound APIs (Eg. OpenFlow Protocol). If there are more than	Good
14	central controller. Unfortunately, writing control applications	Good



15	invariants (e.g., the network does not contain forwarding	Good
16	challenging task.SDN can deliver a consistent, reliable and	Good
17	to all users, which has a significant impact on worker	Good
18	of running IT. Achieving this can be a daunting task because	Good
19	increasingly complex. For decades, IT leaders have had to make	Good
20	accomplish both. SDN can help create an environment where	Good
21	network service, no matter how small the location. This means	Good
22	the location. This means each branch office would have	Good
23	separate router, firewall, VPN device and other devices to	Good
24	services required. This "service chaining" is highly inefficient	Good
25	individually, which increases management and troubleshooting	Good
26	troubleshooting complexity. In contrast, SDN enables multiple servers	Good
27	hardware platform, reducing both troubleshooting and provisioning	Good
28	provisioning times. Service providers will increase the profitability	Good
29	With the growth and advancement in Industrialisation	Good



30	controller and clients. Load Balancer Application uses 2-Threshold	Good
31	programmatically initialize, control, change, and manage network behaviour	Good
32	open interfaces.By using SDN,we created a topology and installed	Good
33	controller and clients. Load Balancer Application uses 2-Threshold	Good
34	Start OpenDaylight SDN Controller using command:	Good
35	Run the fat tree topology using Mininet:	Good
36	-- controller=remote,ip=127.0.0.1,port=6653	Good
37	Run the pingall command to ensure all links are	Good
38	Check the GUI of OpenDaylight using link:	Good
39	Ping any two hosts to get real simulation of data	Good
40	Run the load balancing script using command:	Good
41	Python file will run the application for load balancing	Good
42	Node can be in any of the following state	Good
43	Threshold will be calculated dynamically and accordingly	Good
44	measures are needed. When the overload situation ends then first	Good
45	measures are stopped. After a short guard period Load Balancing	Good



46	failure occurs. Static load balancing algorithms are less	Good
47	fails at run-time. Dynamic load balancing algorithms are more	Good
48	queue length, cpu utilization and memory utilization are	Good
49	low threshold, H and L are constants.	Good
50	We will plot the graph of above parameters for	Good
51	Load vs Throughput for Threshold parameter.	Good
52	Number of Users vs Throughput for Number of Users.	Good
53	Load vs Number of Users for Scalability and Robustness.	Good
54	Load vs Threshold for Process Migration.	Good
55	the devices i.e., the Control Plane is separated from Data	Good
56	from Data Plane. Developers can now deploy their own applications	Good
57	more flexible. SDN is considered to be the future of Networking	Good
58	many functions. Even intelligent applications can be programmed	Good
59	Due to very small topology the results here may	Good
60	Clients and Servers. The Developer only needs to built an efficient	Good

9.2.3. Draft of the 2nd paper published.

Performance Evaluation of Load Balancing Algorithms for SDN

Vikas Kukreja
Department of Computer Engineering,
Vivekanand Education Society Institute of Technology,
Chembur, India
vikas.kukreja@ves.ac.in

Dinesh Panchi
Department of Computer Engineering,
Vivekanand Education Society Institute of Technology,
Chembur, India
dinesh.panchi@ves.ac.in

Jatin Sajnani
Department of Computer Engineering,
Vivekanand Education Society Institute of Technology,
Chembur, India
jatin.sajnani@ves.ac.in

Hitesh Seedani
Department of Computer Engineering,
Vivekanand Education Society Institute of Technology,
Chembur, India
hitesh.seedani@ves.ac.in

Dr.(Mrs).Nupur Giri
Department of Computer Engineering,
Vivekanand Education Society Institute of Technology,
Chembur, India
nupur.giri@ves.ac.in

Abstract- SDN is the only domain which brought the programmability, simplicity, and flexibility to the networks. Using SDN, the same architecture can be used for many applications like switches, routers, hubs, etc. This versatility of architecture was never before seen in any domain which makes SDN the future of networking. Today due to the increase in the service requirement there is more load on the server this may result in Latency/Delay or sometimes the link failure or server crash. To avoid this latency or link failure or server crash there must be a Load Balancer between the client and server to handle this load. We will create a topology and deploy different load balancing applications to evaluate their performance.

Keywords: SDN, Fat Tree Topology, Dijkstra's, A*.

I. Introduction

A. *Introduction*

Software Defined Networking(SDN) is a revolutionary networking technology that brought programmability in the networking by decoupling Control Plane and Data Plane and deploying a single SDN Controller to monitor the entire network.

Software Defined Networks consists of 3 layers:

1. Data/Infrastructure Layer.
2. Control Layer.
3. Application Layer.

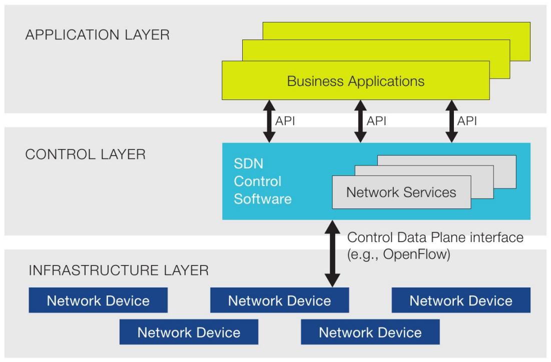


Fig 1. Architecture of SDN

Infrastructure Layer basically consists of all the forwarding devices which forwards the packets to other forwarding device. Control Layer monitors the entire network by sending flow rules to direct switches. Controller and switches communicates with each other through OpenFlow protocol. Over the Control Layer there lies the Application Layer in which different applications can be deployed. The Controller and Application Layer will communicate with each other through REST APIs.

B. *Problem Definition*

The ever increasing load on the servers(because of increase in the demand of the service) has to be balanced in order to provide efficient and consistent services to those who need it. If the load is not properly balanced it may lead to link failures and server crash.Therefore to balance the load and provide the services efficiently a Load Balancing Application can be deployed on the Controller to balance the load and monitor the traffic so that the entire network works seamlessly.

II. Methodology

A comparison of load balancing application using SDN with two different algorithms is made by deploying a custom topology on SDN Controller which represents same for both algorithms to compare results.The custom topology implemented for performing load balancing can be shown as below:

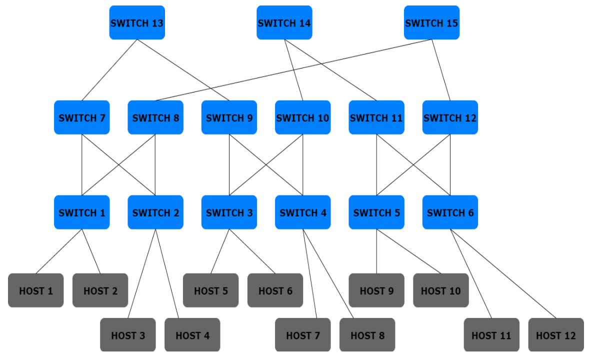


Fig 2. Custom fat tree topology

Two algorithms Dijkstra's and A* are compared based on parameters such as Number of users, Load on each Node, Response time, Number of requests nodes can handle etc. to test scalability, performance and reliability of SDN network performing load balancing using these algorithms.

The SDN network consisting of virtual hosts, switches, controller, an application that needs to be tested for Dijkstra's or A* deployed on controller is simulated using Minnet tool.

Mininet is a network simulator that creates a network of virtual hosts, switches, controllers and links to provide network testbed for developing OpenFlow applications. Mininet switches support OpenFlow protocol that is used for custom routing and software defined networking.

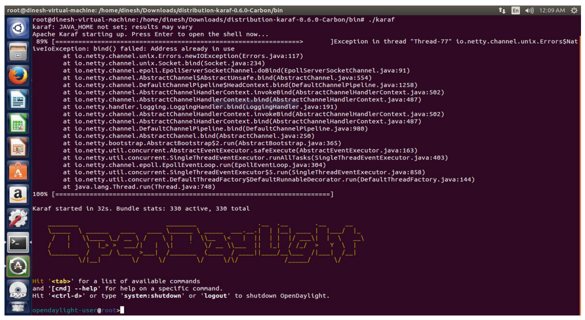
After building testbed for SDN using Mininet, following steps are implemented to run a load balancing application:

A. A SDN controller called OpenDaylight is installed in linux kernel and initiated.
B. Custom topology is deployed using Mininet.
C. Ping each and every host in the network using pingall command to test their existence.
D. A complete network of hosts and switches can be viewed in GUI of OpenDaylight controller, after performing above steps successfully.
E. To check the whether load is balanced between any two hosts say H1 and H2, perform following steps: I. H1 ping H2
II. Run the load balancing application deployed on the controller.
III. Check the results in wireshark.

III. Results

A. Screenshots

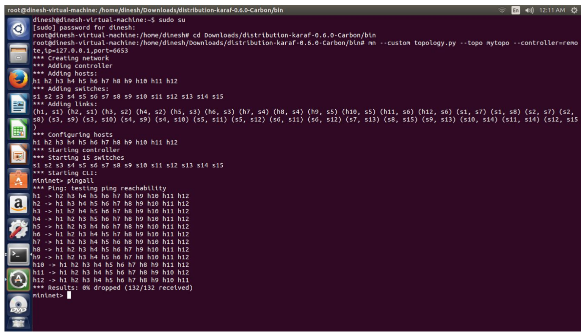
Fig 3 shows the OpenDayLight Controller which is up and running and on which a Topology is created and Load Balancing Application is deployed.



```
root@dinesh-virtual-machine: /home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin
root@dinesh-virtual-machine:/home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin# ./karaf
karaf: JAVA_HOME not set; results may vary
Apache Karaf starting up. Press Enter to open the shell now...
 89% [=================================================================>          ]Exception in thread "Thread-77" io.netty.channel.unix.Errors$Nat
iveIoException: bind() failed: Address already in use
        at io.netty.channel.unix.Errors.newIOException(Errors.java:117)
        at io.netty.channel.unix.Socket.bind(Socket.java:234)
        at io.netty.channel.epoll.EpollServerSocketChannel.doBind(EpollServerSocketChannel.java:91)
        at io.netty.channel.AbstractChannel$AbstractUnsafe.bind(AbstractChannel.java:554)
        at io.netty.channel.DefaultChannelPipeline$HeadContext.bind(DefaultChannelPipeline.java:1258)
        at io.netty.channel.AbstractChannelHandlerContext.invokeBind(AbstractChannelHandlerContext.java:502)
        at io.netty.channel.AbstractChannelHandlerContext.bind(AbstractChannelHandlerContext.java:487)
        at io.netty.handler.logging.LoggingHandler.bind(LoggingHandler.java:191)
        at io.netty.channel.AbstractChannelHandlerContext.invokeBind(AbstractChannelHandlerContext.java:502)
        at io.netty.channel.AbstractChannelHandlerContext.bind(AbstractChannelHandlerContext.java:487)
        at io.netty.channel.DefaultChannelPipeline.bind(DefaultChannelPipeline.java:980)
        at io.netty.channel.AbstractChannel.bind(AbstractChannel.java:250)
        at io.netty.bootstrap.AbstractBootstrap$2.run(AbstractBootstrap.java:365)
        at io.netty.util.concurrent.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java:163)
        at io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:403)
        at io.netty.channel.epoll.EpollEventLoop.run(EpollEventLoop.java:304)
        at io.netty.util.concurrent.SingleThreadEventExecutor$5.run(SingleThreadEventExecutor.java:858)
        at io.netty.util.concurrent.DefaultThreadFactory$DefaultRunnableDecorator.run(DefaultThreadFactory.java:144)
        at java.lang.Thread.run(Thread.java:748)
100% [========================================================================]

Karaf started in 32s. Bundle stats: 330 active, 330 total

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
```

Fig 3. OpenDayLight Controller

Fat Tree Topology is created and is running on Mininet Network Simulator. Fig 4 shows the 12 hosts and 15 switches which are connected to each other and responding to ping command.



```
root@dinesh-virtual-machine: /home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin
dinesh@dinesh-virtual-machine:~$ sudo su
[sudo] password for dinesh:
root@dinesh-virtual-machine:/home/dinesh# cd Downloads/distribution-karaf-0.6.0-Carbon/bin
root@dinesh-virtual-machine:/home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin# mn --custom topology.py --topo mytopo --controller=remo
te,ip=127.0.0.1,port=6653
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s3) (h6, s3) (h7, s4) (h8, s4) (h9, s5) (h10, s5) (h11, s6) (h12, s6) (s1, s7) (s1, s8) (s2, s7) (s2,
s8) (s3, s9) (s3, s10) (s4, s9) (s4, s10) (s5, s11) (s5, s12) (s6, s11) (s6, s12) (s7, s13) (s8, s15) (s9, s13) (s10, s14) (s11, s14) (s12, s15
)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Starting controller
*** Starting 15 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11
*** Results: 0% dropped (132/132 received)
mininet>
```

Fig 4. Fat Tree Topology running on Mininet

Fig 5 shows the Visualization of Fat Tree Topology in the Web based GUI of ODL Controller. Blue coloured components are the switches while black ones are the hosts.

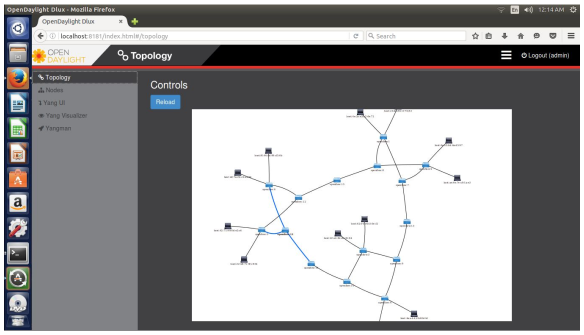
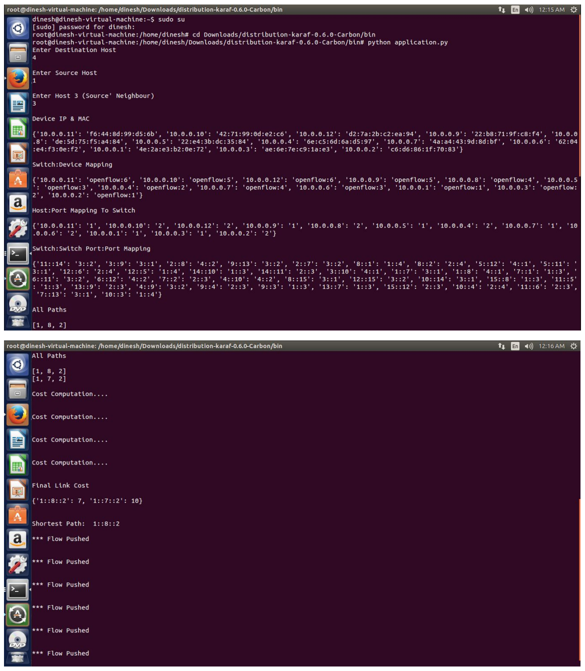


Fig 5. Web based GUI of OpenDayLight Controller showing Fat Tree Topology

This Paper compares 2 Algorithms one is Dijkstra’s and the second one is A* for finding the lightly loaded paths in the network. Fig 6 shows the Load Balancing Application using Dijkstra’s Algorithm which finds all the possible paths in the network and out of them the algorithm selects the best available path to transfer the load on that path.

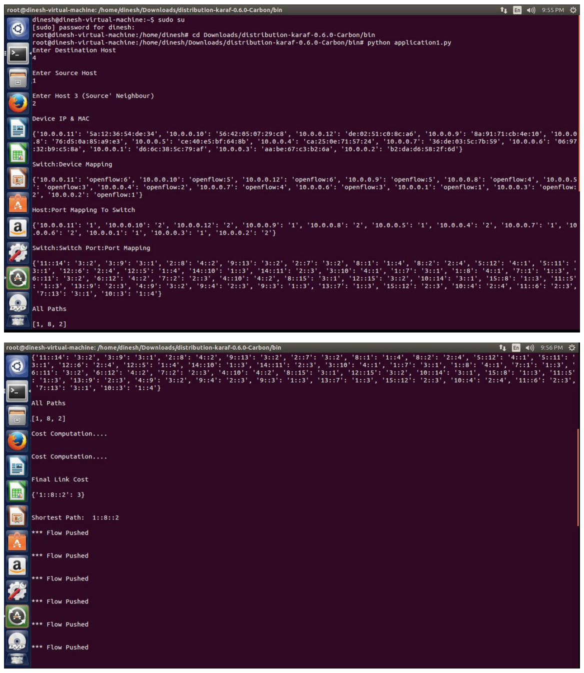


```
root@dinesh-virtual-machine: /home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin
dinesh@dinesh-virtual-machine:~$ sudo su
[sudo] password for dinesh:
root@dinesh-virtual-machine:/home/dinesh# cd Downloads/distribution-karaf-0.6.0-Carbon/bin
root@dinesh-virtual-machine:/home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin# python application.py
Enter Destination Host
4

Enter Source Host
1

Enter Host 3 (Source' Neighbour)
3

Device IP & MAC

{'10.0.0.11': 'f6:44:8d:99:d5:6b', '10.0.0.10': '42:71:99:0d:e2:c6', '10.0.0.12': 'd2:7a:2b:c2:ea:94', '10.0.0.9': '22:b8:71:9f:c8:f4', '10.0.0
.8': 'de:5d:75:f5:a4:84', '10.0.0.5': '22:e4:3b:dc:35:84', '10.0.0.4': '6e:c5:6d:6a:d5:97', '10.0.0.7': '4a:a4:43:9d:8d:bf', '10.0.0.6': '62:04
:e4:f3:0e:f2', '10.0.0.1': '4e:2a:e3:b2:0e:72', '10.0.0.3': 'ae:6e:7e:c9:1a:e3', '10.0.0.2': 'c6:d6:86:1f:70:83'}

Switch:Device Mapping

{'10.0.0.11': 'openflow:6', '10.0.0.10': 'openflow:5', '10.0.0.12': 'openflow:6', '10.0.0.9': 'openflow:5', '10.0.0.8': 'openflow:4', '10.0.0.5
': 'openflow:3', '10.0.0.4': 'openflow:2', '10.0.0.7': 'openflow:4', '10.0.0.6': 'openflow:3', '10.0.0.1': 'openflow:1', '10.0.0.3': 'openflow:
2', '10.0.0.2': 'openflow:1'}

Host:Port Mapping To Switch

{'10.0.0.11': '1', '10.0.0.10': '2', '10.0.0.12': '2', '10.0.0.9': '1', '10.0.0.8': '2', '10.0.0.5': '1', '10.0.0.4': '2', '10.0.0.7': '1', '10
.0.0.6': '2', '10.0.0.1': '1', '10.0.0.3': '1', '10.0.0.2': '2'}

Switch:Switch Port:Port Mapping

{'11::14': '3::2', '3::9': '3::1', '2::8': '4::2', '9::13': '3::2', '2::7': '3::2', '8::1': '1::4', '8::2': '2::4', '5::12': '4::1', '5::11': '
3::1', '12::6': '2::4', '12::5': '1::4', '14::10': '1::3', '14::11': '2::3', '3::10': '4::1', '1::7': '3::1', '1::8': '4::1', '7::1': '1::3', '
6::11': '3::2', '6::12': '4::2', '7::2': '2::3', '4::10': '4::2', '8::15': '3::1', '12::15': '3::2', '10::14': '3::1', '15::8': '1::3', '11::5'
: '1::3', '13::9': '2::3', '4::9': '3::2', '9::4': '2::3', '9::3': '1::3', '13::7': '1::3', '15::12': '2::3', '10::4': '2::4', '11::6': '2::3',
 '7::13': '3::1', '10::3': '1::4'}

All Paths

[1, 8, 2]
```

```
root@dinesh-virtual-machine: /home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin
All Paths

[1, 8, 2]
[1, 7, 2]

Cost Computation....


Cost Computation....


Cost Computation....


Cost Computation....


Final Link Cost

{'1::8::2': 7, '1::7::2': 10}


Shortest Path:  1::8::2
*** Flow Pushed


*** Flow Pushed


*** Flow Pushed


*** Flow Pushed


*** Flow Pushed


*** Flow Pushed
```

Fig 6. Load Balancing Application(Dijkstra’s Algorithm)

Fig 7 shows the snapshots of the Load Balancing Application using A* Algorithm which returns the best available path in the network and transfers the load on that path to balance the load.

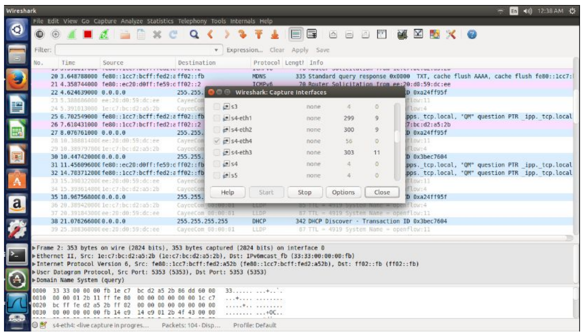


```
root@dinesh-virtual-machine: /home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin
dinesh@dinesh-virtual-machine:~$ sudo su
[sudo] password for dinesh:
root@dinesh-virtual-machine:/home/dinesh# cd Downloads/distribution-karaf-0.6.0-Carbon/bin
root@dinesh-virtual-machine:/home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin# python application1.py
Enter Destination Host
4

Enter Source Host
1

Enter Host 3 (Source' Neighbour)
2

Device IP & MAC

{'10.0.0.11': '5a:12:36:54:de:34', '10.0.0.10': '56:42:05:07:29:c8', '10.0.0.12': 'de:02:51:c0:8c:a6', '10.0.0.9': '8a:91:71:cb:4e:10', '10.0.0
.8': '76:d5:0a:85:a9:e3', '10.0.0.5': 'ce:40:e5:bf:64:8b', '10.0.0.4': 'ca:25:0e:71:57:24', '10.0.0.7': '36:de:03:5c:7b:59', '10.0.0.6': '06:97
:32:b9:c5:8a', '10.0.0.1': 'd6:6c:38:5c:79:af', '10.0.0.3': 'aa:be:67:c3:b2:6a', '10.0.0.2': 'b2:da:d6:58:2f:6d'}

Switch:Device Mapping

{'10.0.0.11': 'openflow:6', '10.0.0.10': 'openflow:5', '10.0.0.12': 'openflow:6', '10.0.0.9': 'openflow:5', '10.0.0.8': 'openflow:4', '10.0.0.5
': 'openflow:3', '10.0.0.4': 'openflow:2', '10.0.0.7': 'openflow:4', '10.0.0.6': 'openflow:3', '10.0.0.1': 'openflow:1', '10.0.0.3': 'openflow:
2', '10.0.0.2': 'openflow:1'}

Host:Port Mapping To Switch

{'10.0.0.11': '1', '10.0.0.10': '2', '10.0.0.12': '2', '10.0.0.9': '1', '10.0.0.8': '2', '10.0.0.5': '1', '10.0.0.4': '2', '10.0.0.7': '1', '10
.0.0.6': '2', '10.0.0.1': '1', '10.0.0.3': '1', '10.0.0.2': '2'}

Switch:Switch Port:Port Mapping

{'11::14': '3::2', '3::9': '3::1', '2::8': '4::2', '9::13': '3::2', '2::7': '3::2', '8::1': '1::4', '8::2': '2::4', '5::12': '4::1', '5::11': '
3::1', '12::6': '2::4', '12::5': '1::4', '14::10': '1::3', '14::11': '2::3', '3::10': '4::1', '1::7': '3::1', '1::8': '4::1', '7::1': '1::3', '
6::11': '3::2', '6::12': '4::2', '7::2': '2::3', '4::10': '4::2', '8::15': '3::1', '12::15': '3::2', '10::14': '3::1', '15::8': '1::3', '11::5'
: '1::3', '13::9': '2::3', '4::9': '3::2', '9::4': '2::3', '9::3': '1::3', '13::7': '1::3', '15::12': '2::3', '10::4': '2::4', '11::6': '2::3',
 '7::13': '3::1', '10::3': '1::4'}

All Paths

[1, 8, 2]
```

```
root@dinesh-virtual-machine: /home/dinesh/Downloads/distribution-karaf-0.6.0-Carbon/bin
{'11::14': '3::2', '3::9': '3::1', '2::8': '4::2', '9::13': '3::2', '2::7': '3::2', '8::1': '1::4', '8::2': '2::4', '5::12': '4::1', '5::11': '
3::1', '12::6': '2::4', '12::5': '1::4', '14::10': '1::3', '14::11': '2::3', '3::10': '4::1', '1::7': '3::1', '1::8': '4::1', '7::1': '1::3', '
6::11': '3::2', '6::12': '4::2', '7::2': '2::3', '4::10': '4::2', '8::15': '3::1', '12::15': '3::2', '10::14': '3::1', '15::8': '1::3', '11::5'
: '1::3', '13::9': '2::3', '4::9': '3::2', '9::4': '2::3', '9::3': '1::3', '13::7': '1::3', '15::12': '2::3', '10::4': '2::4', '11::6': '2::3',
 '7::13': '3::1', '10::3': '1::4'}

All Paths

[1, 8, 2]

Cost Computation....


Cost Computation....


Final Link Cost

{'1::8::2': 3}


Shortest Path:  1::8::2
*** Flow Pushed


*** Flow Pushed


*** Flow Pushed


*** Flow Pushed


*** Flow Pushed


*** Flow Pushed
```

Fig 7. Load Balancing Application(A Algorithm)*

Load on each hosts and how packets are moving in the network can be seen on Wireshark. The input given to the Load Balancing is Host 4 i.e., Load is to be balanced on Host 4 so using Wireshark loads on all the hosts can be determined and the different information related to the ping packets can also be gathered. Fig 8 show the tracking of load on Host 4.


Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save
Wireshark: Capture Interfaces
Help Start Stop Options Close

```
No.  Time         Source                  Destination      Protocol Length Info
  20 3.648788000  fe80::1cc7:bcff:fed2:a ff02::fb          MDNS      335 Standard query response 0x0000  TXT, cache flush AAAA, cache flush fe80::1cc7:
  21 4.358744000  fe80::ec20:d0ff:fe59:c ff02::2           ICMPv6     70 Router Solicitation from ee:20:d0:59:dc:ee
  22 4.624639000  0.0.0.0                255.255.                                                                   D 0xa24ff95f
  25 6.702549000  fe80::1cc7:bcff:fed2:a ff02::fb                                                                   pps._tcp.local, "QM" question PTR _ipp._tcp.local
  26 7.610431000  fe80::1cc7:bcff:fed2:a ff02::2                                                                    7:bc:d2:a5:2b
  27 8.076761000  0.0.0.0                255.255.                                                                   D 0xa24ff95f
  30 10.447420000 0.0.0.0                255.255.                                                                   D 0x3bec7604
  31 11.456096000 fe80::ec20:d0ff:fe59:c ff02::fb                                                                   pps._tcp.local, "QM" question PTR _ipp._tcp.local
  32 14.703712000 fe80::1cc7:bcff:fed2:a ff02::fb                                                                   pps._tcp.local, "QM" question PTR _ipp._tcp.local
  35 18.967568000 0.0.0.0                255.255.                                                                   D 0xa24ff95f
  38 21.076266000 0.0.0.0                255.255.255.255  DHCP      342 DHCP Discover - Transaction ID 0x3bec7604

s3        none    4    0
s4-eth1   none  299    9
s4-eth2   none  300    9
s4-eth4   none   56    0
s4-eth3   none  303   11
s4        none    4    0
s5        none    4    0

▸Frame 2: 353 bytes on wire (2824 bits), 353 bytes captured (2824 bits) on interface 0
▸Ethernet II, Src: 1e:c7:bc:d2:a5:2b (1e:c7:bc:d2:a5:2b), Dst: IPv6mcast_fb (33:33:00:00:00:fb)
▸Internet Protocol Version 6, Src: fe80::1cc7:bcff:fed2:a52b (fe80::1cc7:bcff:fed2:a52b), Dst: ff02::fb (ff02::fb)
▸User Datagram Protocol, Src Port: 5353 (5353), Dst Port: 5353 (5353)
▸Domain Name System (query)

0000  33 33 00 00 00 fb 1e c7  bc d2 a5 2b 86 dd 60 00   33...... ...+..`.
0010  00 00 01 2b 11 ff fe 80  00 00 00 00 00 00 1c c7   ...+.... ........
0020  bc ff fe d2 a5 2b ff 02  00 00 00 00 00 00 00 00   .....+.. ........
0030  00 00 00 00 00 fb 14 e9  14 e9 01 2b 4f 43 00 00   ........ ...+OC..

s4-eth4: <live capture in progres...   Packets: 104 · Disp...   Profile: Default
```

Fig 8. Wireshark

IV. Graphs

Fig 9 compares the performance of Dijkstra’s and A* Algorithm by constantly increasing the Number of Requests. Dijkstra’s Algorithm selects the best available path from the list of all the possible paths while that of A* selects the path which is best locally. The application runs dynamically i.e., the path is switched when the application detects the load on a certain path. Therefore performance of Dijkstra's Algorithm is better in the long run than that of A*. When the requests are not that high both the Algorithms performed the same way but as the number of requests increases Dijkstra's Algorithm proves to be better as it returns all the possible paths.

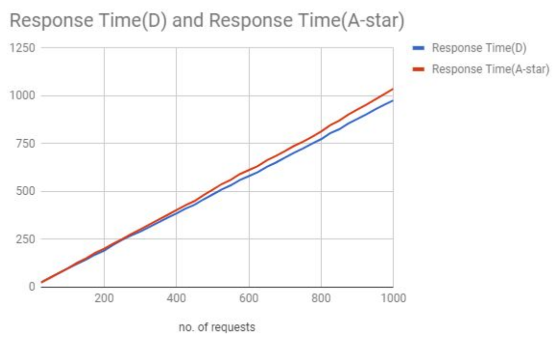


Fig 9. Performance of Algorithms(Dijkstra's vs A)*

Initially, Dijkstra's may not perform well due to switching between different paths but as the load goes on increasing on the network A*'s performance reduces as it does not detect other paths and continue on the same while this is not the case with Dijkstra's, it switches when the path faces traffic and other path seems to be better option. The reason why Dijkstra's should be prefered is, in real networks there are more 12 hosts so Dijkstra's Algorithm can easily can easily handle them.

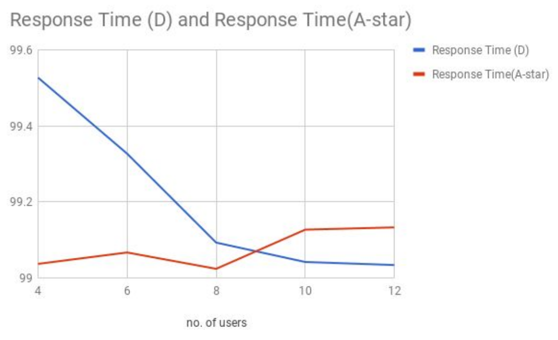


Fig 10. Scalability of Algorithms(Dijkstra's vs A)*

V. Conclusion

SDN has brought about a revolution in the field of Networking because now the brain of the devices i.e., the Control Plane is distinct from Data Plane. Developers can now deploy their own applications on the Controller that makes the Networks more flexible. SDN is considered to be the future of Networking because SDN provides many functions to Developers over Traditional Networks. Even intelligent applications can be programmed and deployed on the Controller to make the network intelligent which was not possible in traditional networks.

We have implemented 2 application on our topology Dijkstra's and A*. Dijkstra's seems to better than A*, above graphs are plotted using various parameters which shows that even though A* seems to be good initially but as number of nodes increases Dijkstra's gives the better results.

VI. References

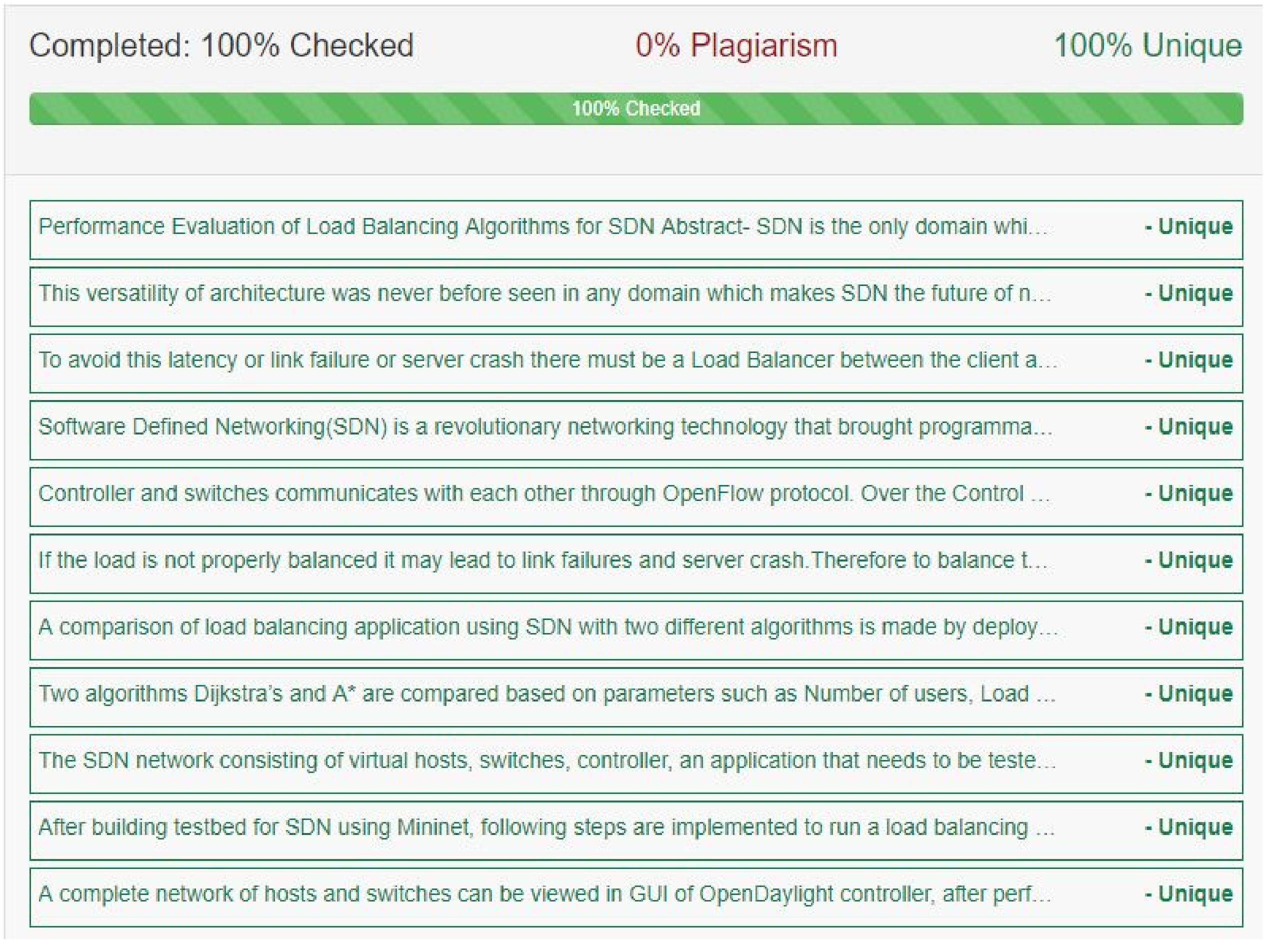
[1] Mr. Marlon Esteban Olaya, Mr. Ivan Bernal, Mr. David Mejia, "Application of Load Balancing in SDN", 10.1109/EATIS.2016.7520102.

[2] Mr. Mohammad Mousa, Mr. Ayman .Bahaa-Eldin, Mr. Mohamed Sobh, "Software Defined Networking concepts and challenges", 10.1109/ICCES.2016.7821979.

[3] Mr. Alexander Gelberger, Mr. Niv Yemini, Mr. Ran Giladi, "Performance Analysis of Software Defined Networking(SDN)", 10.1109/MASCOTS.2013.58.

[4] Mr. G. Araniti, Mr. J. Cosmas, Mr. A. Lera, "OpenFlow over wireless networks: Performance analysis", 10.1109/BMSB.2014.6873559.

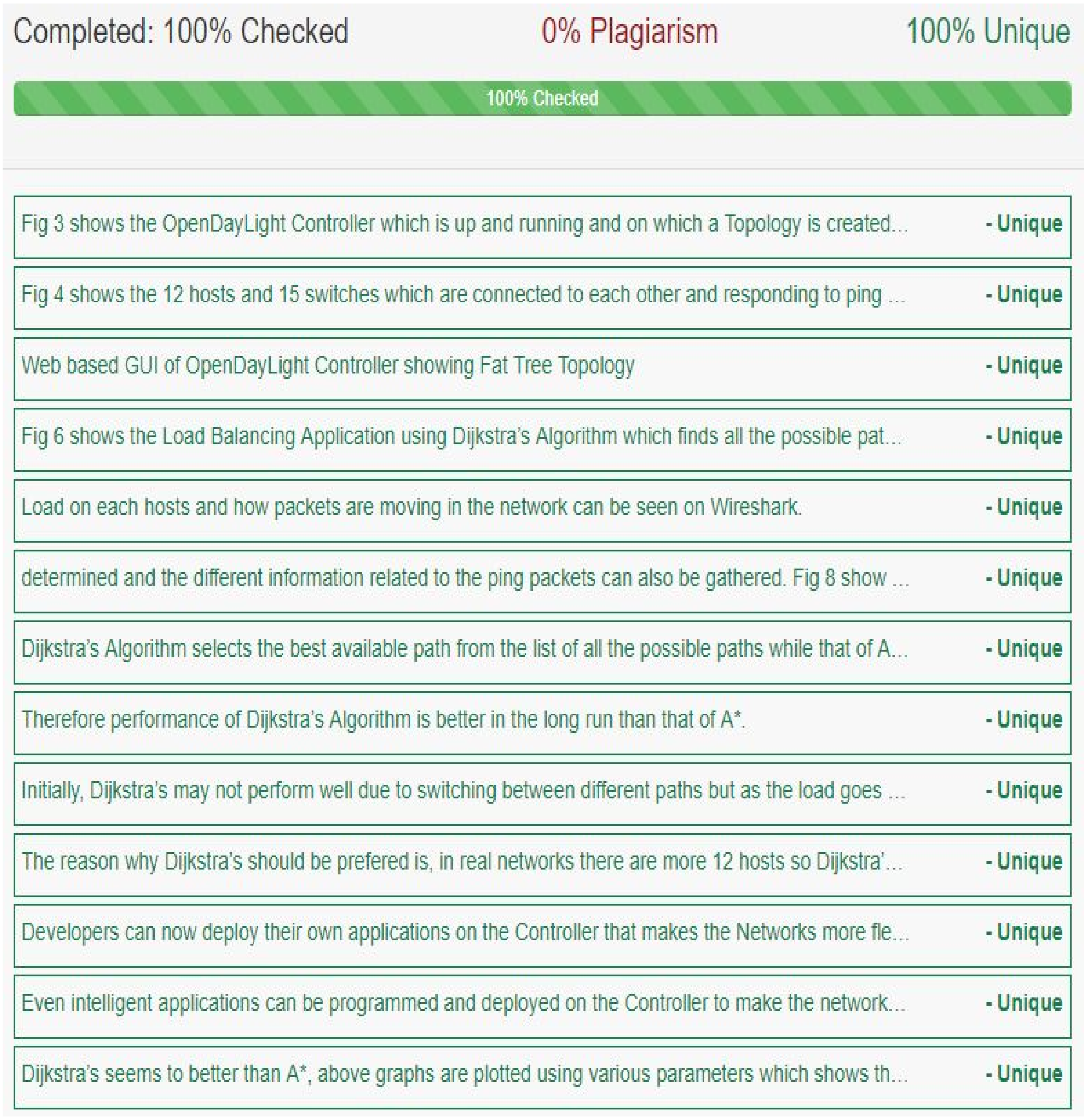
9.2.3. Plagiarism Report of Paper



Completed: 100% Checked | 0% Plagiarism | 100% Unique

100% Checked

Performance Evaluation of Load Balancing Algorithms for SDN Abstract- SDN is the only domain whi...	- Unique
This versatility of architecture was never before seen in any domain which makes SDN the future of n...	- Unique
To avoid this latency or link failure or server crash there must be a Load Balancer between the client a...	- Unique
Software Defined Networking(SDN) is a revolutionary networking technology that brought programma...	- Unique
Controller and switches communicates with each other through OpenFlow protocol. Over the Control ...	- Unique
If the load is not properly balanced it may lead to link failures and server crash.Therefore to balance t...	- Unique
A comparison of load balancing application using SDN with two different algorithms is made by deploy...	- Unique
Two algorithms Dijkstra's and A* are compared based on parameters such as Number of users, Load ...	- Unique
The SDN network consisting of virtual hosts, switches, controller, an application that needs to be teste...	- Unique
After building testbed for SDN using Mininet, following steps are implemented to run a load balancing ...	- Unique
A complete network of hosts and switches can be viewed in GUI of OpenDaylight controller, after perf...	- Unique



Completed: 100% Checked 0% Plagiarism 100% Unique

100% Checked

Fig 3 shows the OpenDayLight Controller which is up and running and on which a Topology is created...	- Unique
Fig 4 shows the 12 hosts and 15 switches which are connected to each other and responding to ping ...	- Unique
Web based GUI of OpenDayLight Controller showing Fat Tree Topology	- Unique
Fig 6 shows the Load Balancing Application using Dijkstra's Algorithm which finds all the possible pat...	- Unique
Load on each hosts and how packets are moving in the network can be seen on Wireshark.	- Unique
determined and the different information related to the ping packets can also be gathered. Fig 8 show ...	- Unique
Dijkstra's Algorithm selects the best available path from the list of all the possible paths while that of A...	- Unique
Therefore performance of Dijkstra's Algorithm is better in the long run than that of A*.	- Unique
Initially, Dijkstra's may not perform well due to switching between different paths but as the load goes ...	- Unique
The reason why Dijkstra's should be prefered is, in real networks there are more 12 hosts so Dijkstra'...	- Unique
Developers can now deploy their own applications on the Controller that makes the Networks more fle...	- Unique
Even intelligent applications can be programmed and deployed on the Controller to make the network...	- Unique
Dijkstra's seems to better than A*, above graphs are plotted using various parameters which shows th...	- Unique