

LOAD BALANCING IN DATA CENTER NETWORK USING SOFTWARE DEFINED NETWORKING

A PROJECT REPORT

Submitted by

**PRAKASH KUMAR [Roll No:GAU-C-15/244]
SOURAV BARMAN [RollNo: GAU-C-15/247]
KAUSHIK KUMAR KASHYAP [Roll No: GAU-C-15/256]
SUNITA BORO [RollNo: GAU-C-15/I-377]**

Under the guidance of

Mr. RANJAN PATOWARY
(Assistant Professor , Department of Information Technology)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY



CENTRAL INSTITUTE OF TECHNOLOGY KOKRAJHAR

(Deemed to be University, MHRD, Govt. of India)

MAY 2019

CERTIFICATE

Certified that this project report titled “**LOAD BALANCING IN DATA CENTER NETWORK USING SOFTWARE DEFINED NETWORKING**” is the bonafide work of “**PRAKASH KUMAR [Roll No:GAU-C-15/244]**, **SOURAV BARMAN [RollNo: GAU-C-15/247]**, **KAUSHIK KUMAR KASHYAP [Roll No: GAU-C-15/256]**, **SUNITA BORO [RollNo: GAU-C-15/I-377]**”, who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Mr. RANJAN PATOWARY
GUIDE
Assistant Professor
Dept. of Information Technology

SIGNATURE

Dr.Amitava Nag
HEAD OF THE DEPARTMENT
Dept. of Information Technology

Signature of the External Examiner

ABSTRACT

Software-defined networking (SDN) is a new network model, and it changes the current limit network facilities. In SDN, the control logic is separated from network devices such as routers and switches. Accordingly, network devices only need to transfer data (data layer), while the control logic is handled by a central controller. A data center network is a facility that centralizes an organizations IT operations and equipment, as well as where it stores, manages, and disseminates its data. Data centers house a networks most critical systems and are vital to the continuity of daily operations. Consequently, the security, reliability and load balancing of data centers and their information is a top priority for organizations.

For load balancing mechanisms in data center network a new concept of queue-length directed adaptive routing has been proposed. This approach handles the incoming packets of the switch by sending out to the output port with the smallest queue-length. We have modified this method to queue-weightage directed adaptive routing for better results, our proposed method handles the incoming packets of the switch by sending out to the output port with smallest queue weightage.

ACKNOWLEDGEMENTS

We would like to express my deepest gratitude to our guide, Mr. Ranjan Patowary for his valuable guidance, consistent encouragement, personal caring, timely help and providing me with an excellent atmosphere for doing research. All through the work, in spite of his busy schedule, he has extended cheerful and cordial support to us for completing this project work.

We express our heartfelt thanks to our Head of the Department, Dr. Amitiva Nag, who has been actively involved and very influential from the start till the completion of our project.

We would also like to thank all teaching and non-teaching staffs of the Information Technology Department for their constant support and encouragement given to us. Last but not the least it is our great pleasure to acknowledge the wishes of friends and well wishers, both in academic and non-academic spheres.

sign:

.....

.....

.....

.....

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENTS	iv
1 INTRODUCTION	1
1.1 Introduction	1
2 LITERATURE SURVEY	3
2.1 Traditional Network	3
2.2 SOFTWARE DEFINED NETWORKING	4
2.3 SDN FRAMEWORK	4
2.3.1 APPLICATION LAYER	6
2.3.2 CONTROL LAYER	6
2.3.3 INFRASTRUCTURE LAYER	6
2.4 BENEFITS OF SDN	6
2.5 SDN COMPONENTS	8
2.6 TRADITIONAL NETWORK VS SDN NETWORK	12
2.7 SDN ARCHITECTURE	13
2.8 OPENFLOW	14
2.8.1 BENEFITS OF OPENFLOW	15
2.9 DATA CENTER NETWORK	15
2.9.1 TOPOLOGIES IN DATA CENTER NETWORK	16
2.9.2 FAT TREE TOPOLOGY	18
2.10 LOAD BALANCING	20
3 WORKDONE	22
3.1 EXISTING SCHEME AND PROPOSED SCHEME:	22
3.2 TESTBED SETUP	23
3.3 IMPLEMENTATION	24
3.3.1 QUEUE IMPLEMENTATION ON HOST	26
3.3.2 FLOW TABLE	26
4 OUTPUT	29

4.1 RESULTS	29
5 CONCLUSION	35
6 FUTURE ENHANCEMENT	36

CHAPTER 1

INTRODUCTION

1.1 Introduction

Data center is a pool of resources (computational, storage, network) interconnected using a communication network. Data Center Network (DCN) holds a pivotal role in a data center, as it interconnects all of the data center resources together. DCNs need to be scalable and efficient to connect tens or even hundreds of thousands of servers to handle the growing demands of Cloud computing.

For a long time, networking technologies have evolved at a slower pace compared to other communication technologies. Network equipments such as switches and routers have been traditionally developed by manufacturers. Each vendor designs his own firmware and other software to operate their own hardware in a proprietary and closed way. This slowed the progress of innovations in networking technologies and caused an increase in management and operation costs whenever new services, technologies or hardware were to be deployed within existing networks. The architecture of today networks consists of three core logical planes: Control plane, data plane, and management plane. So far, networks hardware have been developed with tightly coupled control and data planes. Thus, traditional networks are known to be inside the box paradigm[13]. This significantly increases the complexity and cost of network administration and management. Being aware of these limitations, networking research communities and industrial market leaders have collaborated in order to rethink the design of traditional networks.

The principal endeavors of SOFTWARE DEFINED NETWORKING(SDN) are to separate the control plane from the data plane and to centralize networks intelligence and state. SDN philosophy is based on dissociating the control from the network forwarding elements (switches and routers), logically centralizing network intelligence and state (at the controller), and abstracting the underlying network infrastructure from the applications. SDN is very often linked to the OpenFlow protocol. OpenFlow (OF)

is considered one of the first software defined networking(SDN) standards. It originally defined the communication protocol in SDN environments that enables the SDN Controller to directly interact with the forwarding plane of network devices such as switches and routers, both physical and virtual (hypervisor-based). In this project we are working on the topic load balancing in Data center network using SDN. In[3], the author used fat-tree topology and investigated the flow-level valiant Load Balancing(VLB) technique that uses randomization to deal with highly volatile data center network traffics. There are two proposed methods, queue-length directed adaptive routing and probe-based adaptive routing. In probe-based adaptive routing a probe packet is used to investigate the efficient path to destination, it uses a probe packet with smallest size and send it to many paths and the packet which reaches first to server is received and all other packets are ignored and that server gives acknowledgement to the host who has send the packets, the acknowledgment contains the path from where probe packet has reached and so all other packet also follows the same path to reach the destination. In queue-length directed adaptive routing the concept of queue-length size is used for sending packets to destination. Here for every packet arrives at the switch is sent out to the output port having the smallest queue length at that instance. In our project we have modified the queue-length directed adaptive routing method to queue-weightage adaptive routing to achieve better results.

CHAPTER 2

LITERATURE SURVEY

2.1 Traditional Network

Traditional networks rely on physical infrastructure such as switches and routers to make connections and run properly. In contrast, a software-based network allows the user to control the allocation of resources at a virtual level through the control plane.

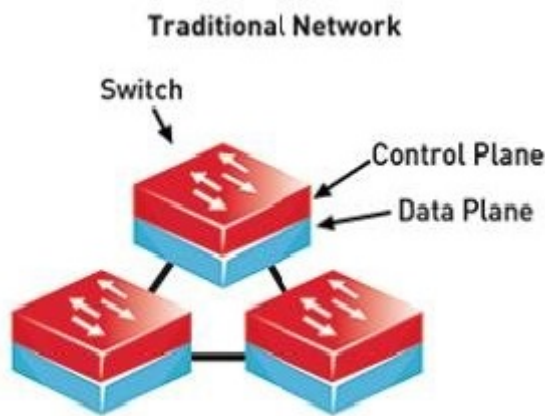


Figure 2.1: Traditional Network

COMPONENTS OF TRADITIONAL NETWORK

Data plane: The act of moving bits that constitute the packet the packet from an incoming port to an outgoing port is the responsibility of the data plane. This is also known as the forwarding plane. For example, in Ethernet switches, packets coming in from one port are forward out via one or more of the remaining ports.

Control plane: Using the previous example, to forward, to forward the packet to correct outgoing port, the data plane may need additional information. In the case of Ethernet switches, the outgoing port is identified using the destination MAC address learnt by

the switch. This act of learning and building awareness about the network is the responsibility of the control plane. The control plane learns gathers information about the network using various protocols.

2.2 SOFTWARE DEFINED NETWORKING

Software-defined networking (SDN) technology is an approach to cloud computing that facilitates network management and enables programmatically efficient network configuration in order to improve network performance and monitoring. SDN is meant to address the fact that the static architecture of traditional networks is decentralized and complex while current networks require more flexibility and easy troubleshooting. SDN attempts to centralize network intelligence in one network component by disassociating the forwarding process of network packets (data plane) from the routing process (control plane). The control plane consists of one or more controllers which are considered as the brain of SDN network where the whole intelligence is incorporated. However, the intelligence centralization has its own drawbacks when it comes to security, scalability and elasticity and this is the main issue of SDN.

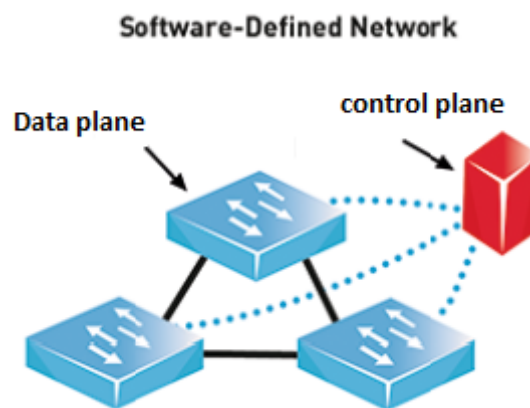


Figure 2.2: SOFTWARE DEFINED NETWORKING

2.3 SDN FRAMEWORK

A Software defined networking(SDN) architecture (or SDN architecture) defines how a networking and computing system can be built using a combination of open, software-based technologies and commodity networking hardware that separate the SDN control

plane and the SDN data plane of the networking stack. Traditionally, both the SDN control plane and data plane elements of a networking architecture were packaged in proprietary, integrated code distributed by one or a combination of proprietary vendors. The OpenFlow standard, created in 2008, was recognized as the first SDN architecture that defined how the control and data plane elements would be separated and communicates with each other using the OpenFlow protocol. The Open Network Foundation (ONF) is the body in charge of managing OpenFlow standards, which are open source.

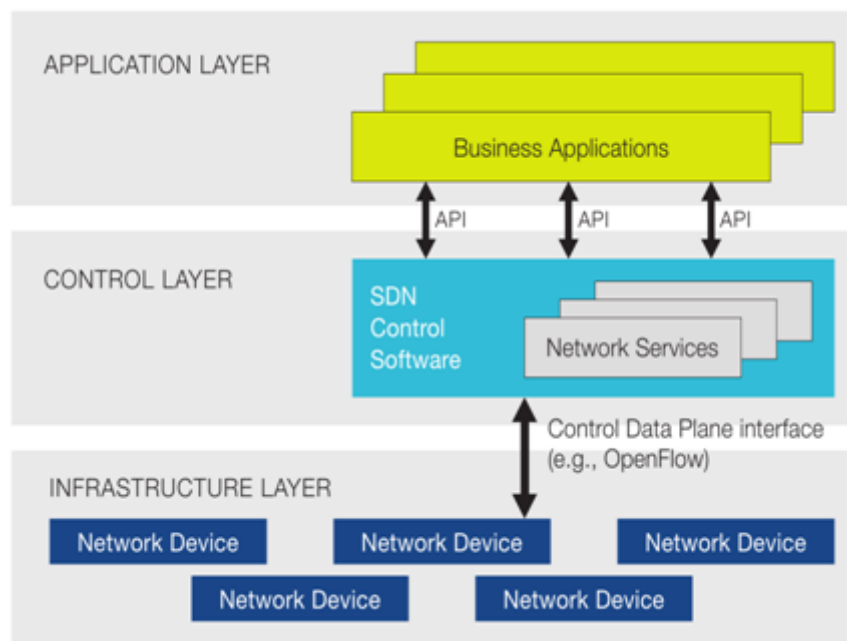


Figure 2.3: SDN FRAMEWORK

In the SDN architecture, the splitting of the control and data forwarding functions is referred to as disaggregation, because these pieces can be sourced separately, rather than deployed as one integrated system. This architecture gives the applications more information about the state of the entire network from the controller, as opposed to traditional networks where the network is application aware.

2.3.1 APPLICATION LAYER

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

2.3.2 CONTROL LAYER

The control layer represents the centralized SDN controller software that acts as the brain of the SDN network. This controller resides on a server and manages policies and the flow of traffic throughout the network.

2.3.3 INFRASTRUCTURE LAYER

The infrastructure layers is made up of the physical switches in the network. These three layers communicate using respective northbound and southbound applications programming interfaces. For example, applications talk to the controller through its northbound interface, while the controller and switches communicate using southbound interfaces, such as openflow although other protocols also exist.

2.4 BENEFITS OF SDN

1. **CENTRALIZED NETWORK PROVISIONING** Software defined networks provide a centralized view of the entire network, making it easier to centralize enterprise management and provisioning. For example, more VLANs are becoming part of physical LANs, creating a Gordian knot of links and dependencies. By abstracting the control and data planes, SDN can accelerate service delivery and provide more agility in provisioning both virtual and physical network devices from a central location.
2. **HOLISTIC ENTERPRISE MANAGEMENT** Enterprise networks have to set up new applications and virtual machines on demand to accommodate new processing requests such as those for big data. SDN allows IT managers to experiment

with network configuration without impacting the network. SDN also supports management of both physical and virtual switches and network devices from a central controller; something you cannot do with SNMP. SDN provides a single set of APIs to create a single management console for physical and virtual devices.

3. **MORE GRANULAR SECURITY** One of the advantages of security defined networking that appeals most to IT managers is centralized security. Virtualization has made network management more challenging. With virtual machines coming and going as part of physical systems, its more difficult to consistently apply firewall and content filtering policies. When you add in complexities such as securing BYOD devices, the security problem is compounded. The SDN Controller provides a central point of control to distribute security and policy information consistently throughout the enterprise. Centralizing security control into one entity, like the SDN Controller, has the disadvantage of creating a central point of attack, but SDN can effectively be used to manage security throughout the enterprise if it is implemented securely and properly.
4. **LOWER OPERATING COST** Administrative efficiency, improvements in server utilization, better control of virtualization, and other benefits should result in operational savings. Although it is still early to show real proof of savings, SDN should lower overall operating costs and result in administrative savings since many of the routine network administration issues can be centralized and automated.
5. **HARDWARE SAVINGS AND REDUCE CAPITAL EXPENDITURES** Adopting SDN also gives new life to existing network devices. SDN makes it easier to optimize commoditized hardware. Existing hardware can be repurposed using instructions from the SDN controller and less expensive hardware can be deployed to greater effect since new devices essentially become white box switches with all the intelligence centered at the SDN controller.
6. **CLOUD ABSTRACTION** Cloud computing is here to stay and it is evolving into a unified infrastructure. By abstracting cloud resources using software defined networking, its easier to unify cloud resources. The networking components that make up massive data center platforms can all be managed from the SDN controller.
7. **GUARANTEED CONTENT DELIVERY** The ability to shape and control data traffic is one of the primary advantages of software defined networking. Being able to direct and automate data traffic makes it easier to implement quality of services for voice over IP and multimedia transmissions. Streaming high quality video is easier because SDN improves network responsiveness to ensure a flawless user experience.

2.5 SDN COMPONENTS

SDN Consist of different components. Here, we will see all these components and their duty one by one. What are these SDN Components? The SDN Components are:

1. Network Devices (Data Plane)
2. SDN Controller (Control Plane)
3. Southbound Interface
4. Northbound Interface
5. Network Operating System (NOS)
6. Application and Services (Application Plane)

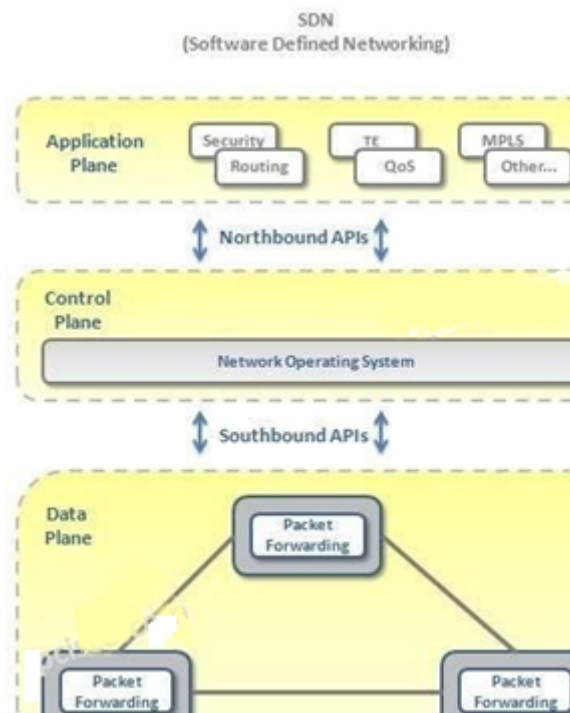


Figure 2.4: SDN COMPONENTS

NETWORK DEVICES (DATA PLANE): Data Plane is consist of various Network devices both physical and Virtual. The main duty of data plane is forwarding. In the previous traditional networks, both control and data plane was in the same device. But with SDN, network devices has only data plane. So, the main role of these network devices is only Forwarding the data. This provide a very efficient Forwarding mechanism.

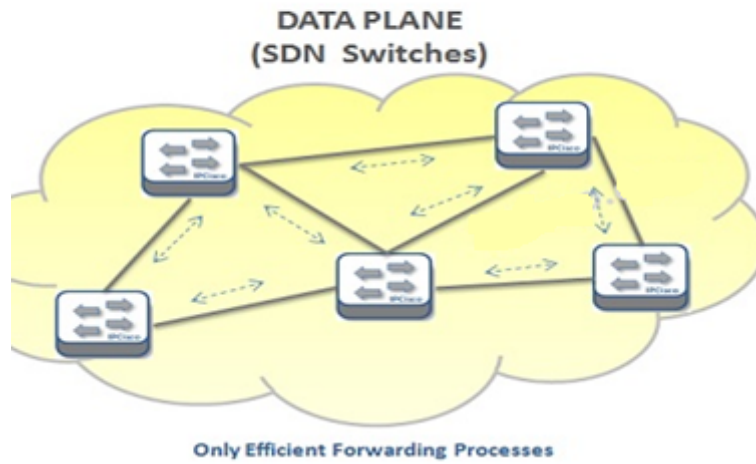


Figure 2.5: DATA PLANE

SDN CONTROLLER (CONTROL PLANE) [15] SDN Controller is the Center of the SDN Architecture. In other words, SDN Controller is the brain of the system. The control of all the data plane devices is done via SDN Controller. It also controls the Applications at Application Layer. SDN Controller communicates and controls this upper and lower layer with APIs through Interfaces.

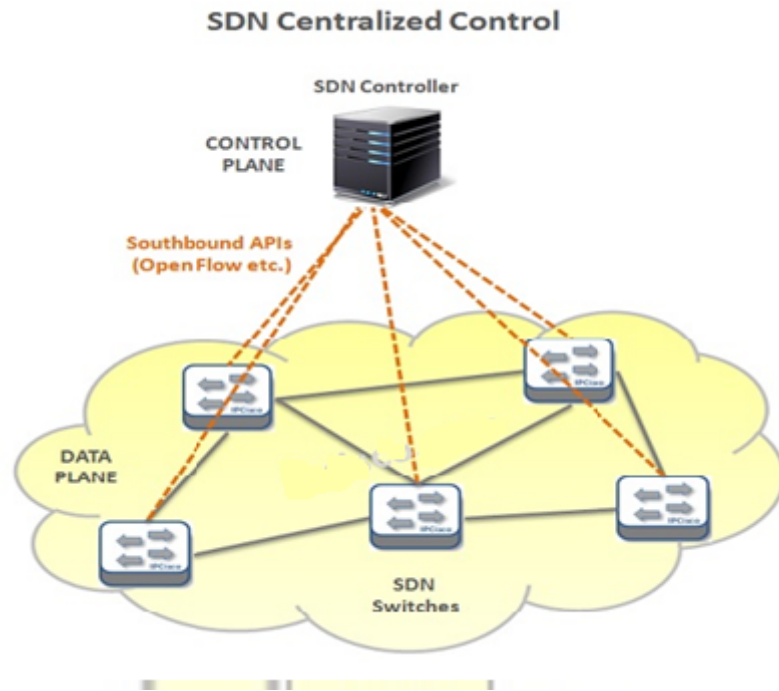


Figure 2.6: CONTROL PLANE

SOUTHBOUND INTERFACES

A southbound interface (SBI) is a component's lower level interface layer. It is directly connected to that lower layer's northbound interface. It breaks down the concepts into smaller technical details that are specifically geared toward a lower layer component within the architecture. In software-defined networking (SDN), the southbound interface serves as the OpenFlow or alternative protocol specification. It allows a network component to communicate with a lower level component.

NORTHBOUND INTERFACE

A northbound interface (NBI) is the interface to a component of higher function or level layer. The lower layer's NBI links to the higher layer's southbound interface (SBI). In an architectural overview, a NBI is drawn on the top portion of the component or layer in question and can be thought of as flowing upward, while a SBI is drawn at the bottom, symbolizing a downward flow.

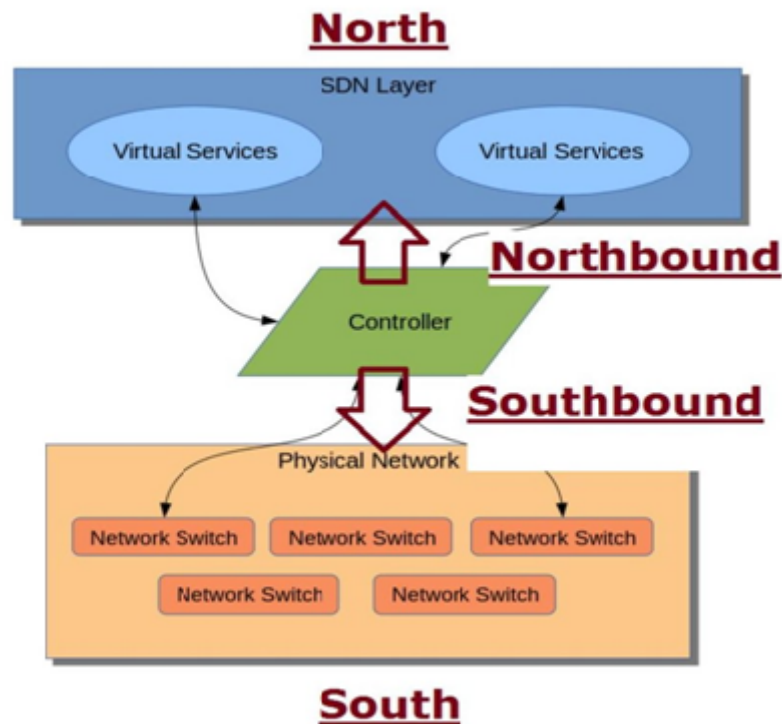


Figure 2.7: INTERFACES

NETWORK OPERATING SYSTEM A network operating system (NOS) is a computer operating system that is designed primarily to support workstation, personal computer, and in some instances, older terminal that are connected on a local area network(LAN).

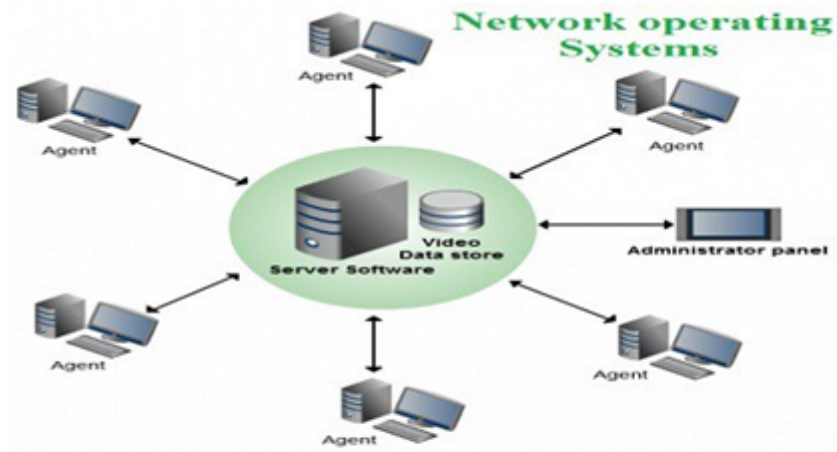


Figure 2.8: NETWORK OPERATING

2.6 TRADITIONAL NETWORK VS SDN NETWORK

In Traditional networks, all network devices has Data plane and Control plane. In other words, Traditional networks are using integrated hardware and software. They makes their own decision about routing and switching. Whereas, Software-defined networking

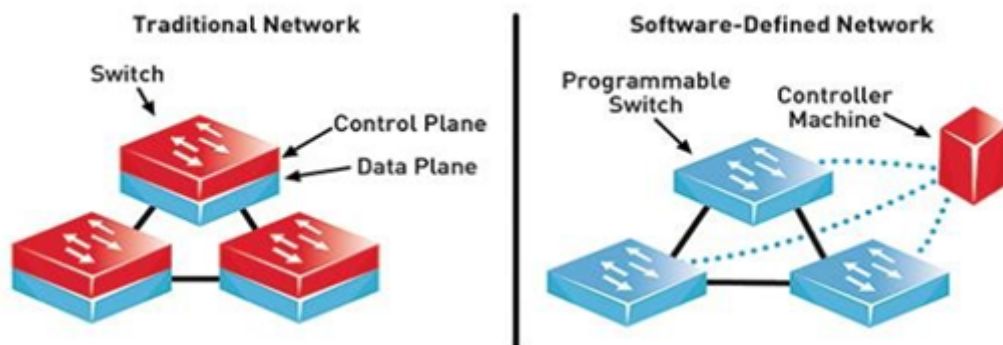


Figure 2.9: TRADITIONAL VS SDN NETWORK

(SDN) , is an emerging networking paradigm that gives hope to change the limitations of current network infrastructures. First, it breaks the vertical integration by separating

the network's control logic (the control plane) from the underlying routers and switches that forward the traffic (the data plane). Second, with the separation of the control and data planes, network switches become simple forwarding devices and the control logic is implemented in a logically centralized controller (or network operating system), simplifying policy enforcement and network (re)configuration and evolution. A simplified view of this architecture is shown in Fig 2.9. It is important to emphasize that a logically centralized programmatic model does not postulate a physically centralized system. In fact, the need to guarantee adequate levels of performance, scalability, and reliability would preclude such a solution.

2.7 SDN ARCHITECTURE

The separation of the control plane and the data plane can be realized by means of a well-defined programming interface between the switches and the SDN controller. The controller exercises direct control over the state in the data plane elements via this well-defined application programming interface (API).

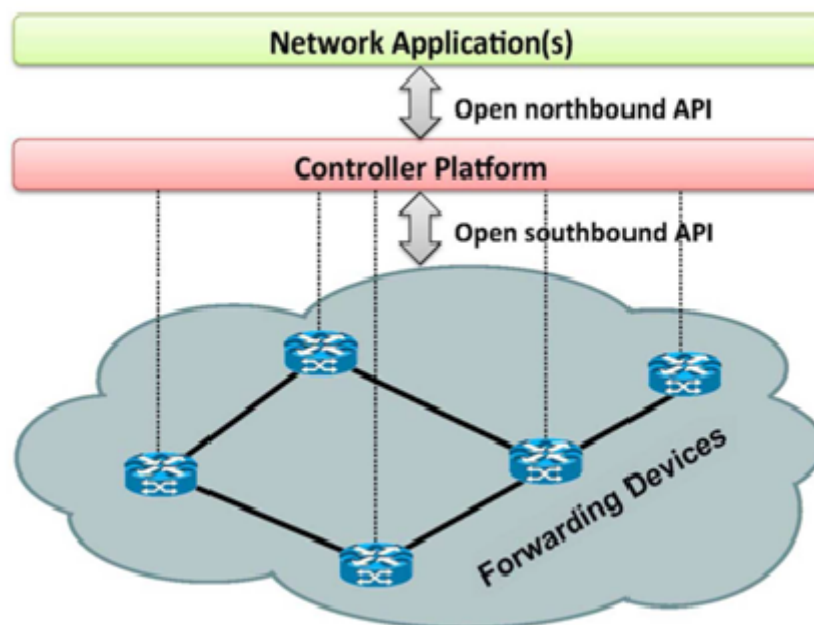


Figure 2.10: SDN ARCHITECTURE

2.8 OPENFLOW

OpenFlow is a protocol that allows a server to tell network switches where to send packets. In a conventional network, each switch has proprietary software that tells it what to do. With OpenFlow, the packet-moving decisions are centralized, so that the network can be programmed independently of the individual switches and data center gear. In a conventional switch, the data path and the control path occur on the same device. An OpenFlow switch separates the data path from the control path. The data path portion resides on the switch itself; a separate controller makes high-level routing decisions. The switch and controller communicate by means of the OpenFlow protocol. This methodology, known as SDN, allows for more effective use of network resources than is possible with traditional networks. OpenFlow has gained favor in applications such as VM (virtual machine) mobility, mission-critical networks and next generation IP-based mobile networks.

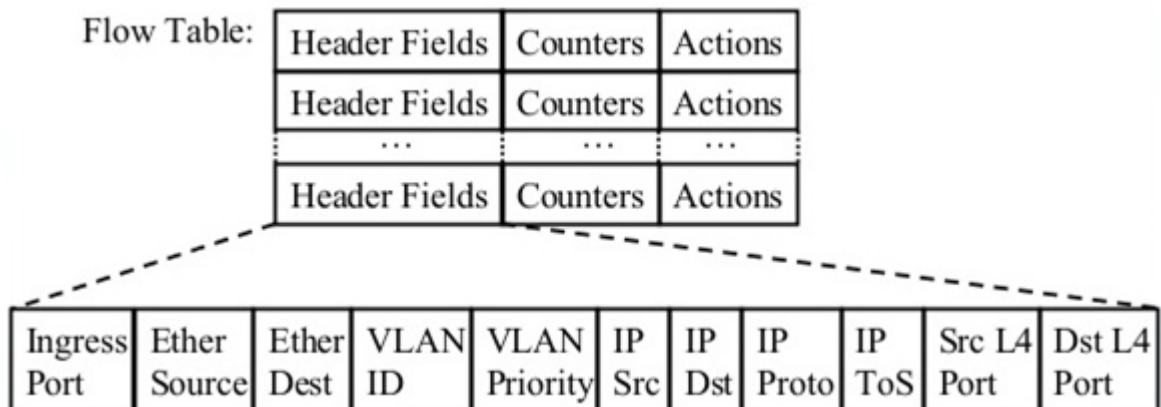


Figure 2.11: FLOWTABLE

Header Feilds: fields againts which packet can be matched.

Counters : Statistics reporting capabilities.

Actions : Defining how the packet should be treated(forward, drop, modified).

2.8.1 BENEFITS OF OPENFLOW

1. OpenFlow-based SDN creates flexibility in how the network is used, operated, and sold. The software that governs it can be written by enterprises and service providers using ordinary software environments.
2. It promotes rapid service introduction through customization, because network operators can implement the features they want in software they control, rather than having to wait for a vendor to put it in plan in their proprietary products.
3. It lowers operating expenses and results in fewer errors and less network downtime because it enables automated configuration of the network and reduces manual configuration.
4. OpenFlow-based SDN enables virtualization of the network, and therefore the integration of the network with computing and storage. This allows the entire IT operation to be governed more sleekly with a single viewpoint and toolset.
5. It can be easily integrated with computing for resource management and maintenance.
6. OpenFlow-based SDN can better align the network with business objectives.
7. As a standard way of conveying flow-table information to the network devices, it fosters open, multi-vendor markets.

2.9 DATA CENTER NETWORK

Data center networking is the process of establishing and interconnecting the entire physical and network-based devices and equipment within a data center facility. It enables a digital connection between data center infrastructure nodes and equipment to ensure that they can communicate and transfer data between each other and on an external network or Internet.

1. Typically, data center networking creates a network infrastructure that is: Stable, secure and reliable.
2. In line with the industry regulations and meets organization/customer/users needs
3. Supports networking requirements for modern technologies such as cloud computing and virtualization
4. Scalable and can easily meet the requirements of network communications in peak usage.
5. The components and technologies that make up data center networking generally include:

6. Networking equipment (routers, switches, modems, etc.)
7. Network cabling (LAN/WAN and network interface cabling)
8. Network addressing scheme such as IP V4 or IP V6
9. Network security (security protocols/encryption algorithms, firewalls, IDS)
10. Internet connectivity (satellite, DSL, wireless, optical)

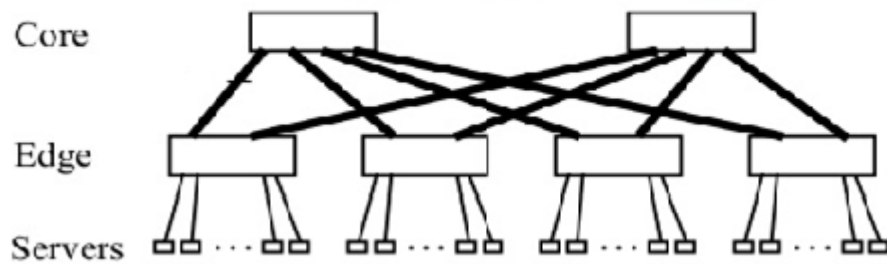


Figure 2.12: DATA NETWORK

11. 6 identical 36-port switches. All ports 1Gbps. 72 Servers.
12. Each edge switch connects to 18 servers. 9 uplinks to first core switch. Other 9links to 2nd core switch.
13. Throughput between any two servers=1 Gbps using ECMP Identical bandwidth at any bisection.
14. Negative: Cabling complexity.

2.9.1 TOPOLOGIES IN DATA CENTER NETWORK

1. Three-Tier Topology
2. Fat-Tree Topology.
3. D-Cell Topology, etc.

THREE-TIER TOPOLOGY

The legacy three-tier DCN architecture follows a multi-routed tree based network topology composed of three layers of network switches, namely access, aggregate, and core layers. The servers in the lowest layers are connected directly to one of the edge layer switches. The aggregate layer switches interconnects multiple access layer switches

together. All of the aggregate layer switches are connected to each other by core layer switches. Core layer switches are also responsible for connecting the data center to the Internet. The three-tier is the common network architecture used in data centers. However, three-tier architecture is unable to handle the growing demand of cloud computing. The higher layers of the three-tier DCN are highly oversubscribed. Moreover, scalability is another major issue in three-tier DCN. Major problems faced by the three-tier architecture include, scalability, fault tolerance, energy efficiency, and cross-sectional bandwidth. The three-tier architecture uses enterprise-level network devices at the higher layers of topology that are very expensive and power hungry.

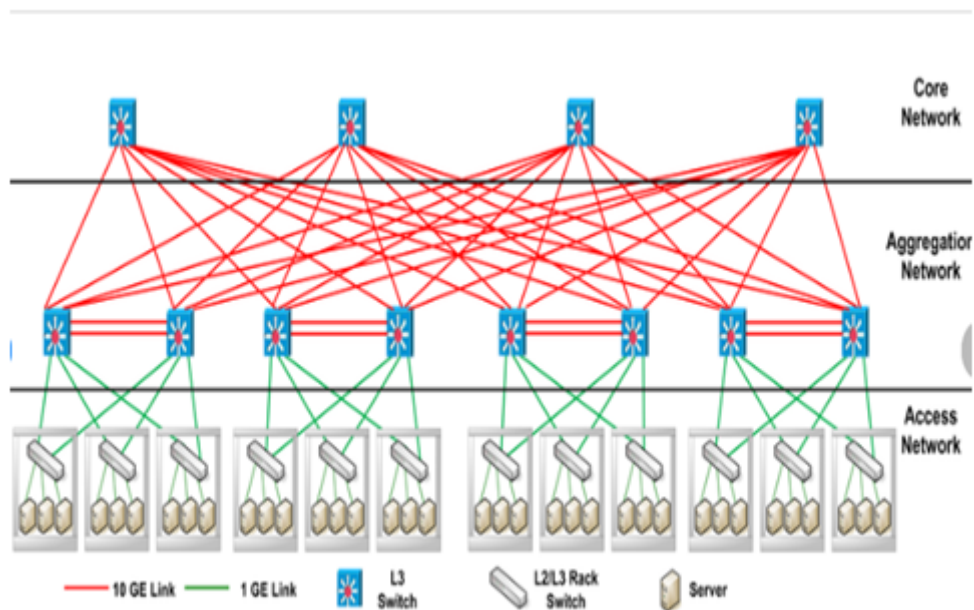


Figure 2.13: THREE-TIER TOPOLOGY

2.9.2 FAT TREE TOPOLOGY

Fat tree DCN architecture handles the oversubscription and cross section bandwidth problem faced by the legacy three-tier DCN architecture. Fat tree DCN employs commodity network switches based architecture using Clos topology. The network elements in fat tree topology also follows hierarchical organization of network switches in access, aggregate, and core layers. However, the number of network switches is much larger than the three-tier DCN. The architecture is composed of k pods, where each pod contains, $(k/2)2$ servers, $k/2$ access layer switches, and $k/2$ aggregate layer switches in the topology. The core layers contain $(k/2)2$ core switches where each of the core switches is connected to one aggregate layer switch in each of the pods. The fat tree topology offers 1:1 oversubscription ratio and full bisection bandwidth. The fat tree architecture uses a customized addressing scheme and routing algorithm. The scalability is one of the major issues in fat tree DCN architecture and maximum number of pods is equal to the number of ports in each switch.

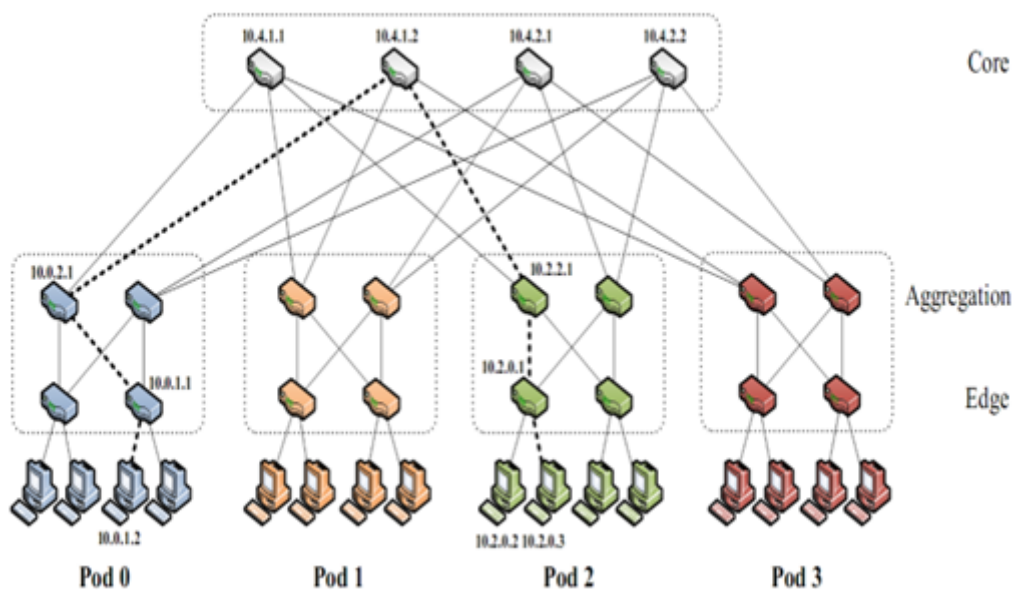


Figure 2.14: FAT-TREE TOPOLOGY

D-CELL TOPOLOGY

DCell is a server-centric hybrid DCN architecture where one server is directly connected to many other servers. A server in the DCell architecture is equipped with multiple Network Interface Cards (NICs). The DCell follows a recursively built hierarchy of cells. A cell0 is the basic unit and building block of DCell topology arranged in multiple levels, where a higher level cell contains multiple lower layer cells. The cell0 is building block of DCell topology, which contains n servers and one commodity network switch. The network switch is only used to connect the server within a cell0. A cell1 contains $k=n+1$ cell0 cells, and similarly a cell2 contains $k * n + 1$ cell1. The DCell is a highly scalable architecture where a four level DCell with only six servers in cell0 can accommodate around 3.26 million servers. Besides very high scalability, the DCell architecture depicts very high structural robustness. However, cross section bandwidth and network latency is a major issue in DCell DCN architecture.

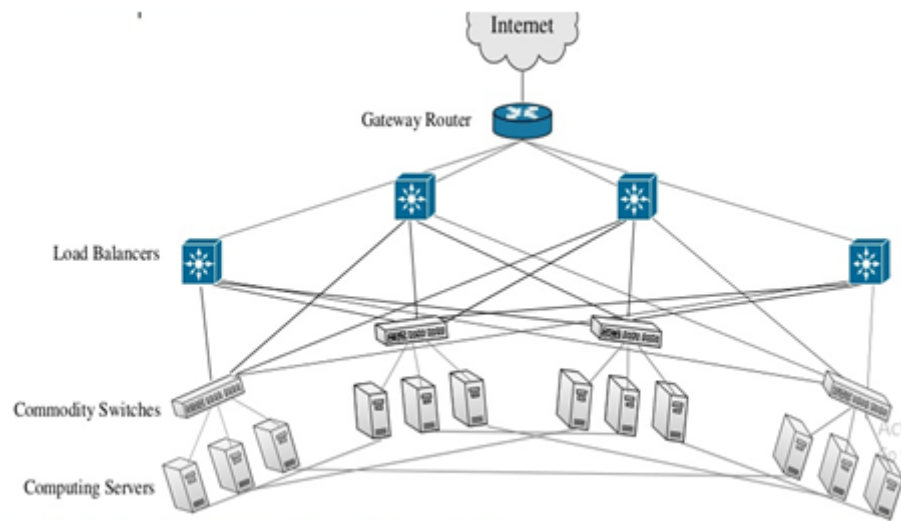


Figure 2.15: D-CELL TOPOLOGY

2.10 LOAD BALANCING

Load balancing is a technique used to distribute workloads uniformly across servers or other compute resources to optimize network efficiency, reliability and capacity. Load balancing is performed by an appliance – either physical or virtual – that identifies in real time which server in a pool can best meet a given client request, while ensuring heavy network traffic doesn't unduly overwhelm a single server. In SDN load balancing implied a productive and clever blockage mindful directing resolution. In the light of SDN condition it is a fundamental limitation to enhance the adaptability and accessibility of network which prompts accomplish greatest number of packets dealt with by the controller in insignificant time for any application.

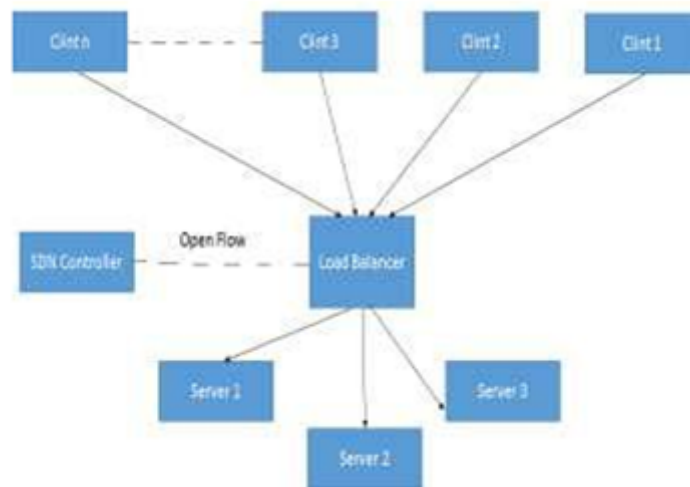


Figure 2.16: LOAD BALANCING

With the SDN controller the load balancing design comprises of open flow switch in which various servers are associated with it. SDN controller keeps p rundown of live servers that are associated with the open flow switch, and every server is appointed with static IP address. On an outstanding port 80 web facility is running on every server, virtual address covers by controller and to the virtual IP address all requests from the customers are sent. When the customers sends a request to the virtual IP, open flow

switch utilized the data contained in bundle header and contrast it and flow passages in switch and if the customers bundle header data coordinates with flow section, at that point switch alter the goal virtual IP deliver to the address of one of the servers in view of load balancing procedure and forward the request to that server.

CHAPTER 3

WORKDONE

3.1 EXISTING SCHEME AND PROPOSED SCHEME:

In [3] the author used fat-tree topology and investigated the flow-level valiant Load Balancing(VLB) technique that uses randomization to deal with highly volatile data center network traffics. Their two methods have been proposed, queue-length directed adaptive routing and probe-based adaptive routing. In queue-length directed adaptive routing technique, the concept of real time queue-length size is used for sending packets to the destination. And probe-based adaptive routing uses a probe packet first to investigate the efficient path to destination it uses a probe packet with smallest size and send it to many paths and the packet which reaches first to server is received and all other packets are ignored and that server gives acknowledgement to the host who has send the packets ,the acknowledgment also contains the path from where probe packet has reached and so all other packet also follows the same path to reach the destination. In our project we have modified the queue-length directed adaptive routing method to achieve better results. In queue-length directed adaptive routing method they have used the concept of queue-length size, queue length size is the no.of packets present in that queue in real time . For example consider there are four ports in a switch and the three ports are monitored by a queue, let q_1, q_2, q_3 be three queue whose real time queue length at an instance found to be 5,3,7 respectively on the ports 1,2,3 respectively and a packet has arrived in port 4 of the switch then according to the queue-length directed adaptive routing among three out-port it will choose the port whose queue length is smallest hence packet will go out from port 2, because for now the smallest queue length is in port 2. According to this queue-length directed adaptive routing only queue-length is considered but for example suppose the port 2 whose length is smallest one and having 3 packets in its queue Have the total size of 20kb and in the port 1, the queue q_1 whose queue length is 5 have total size of 13kb then this existing solution might give wrong paths for some packets. Hence we are considering the weightage of queue which

means total size of the packets inside the queue, which determines the weightage of each queue.

3.2 TESTBED SETUP

We have used Virtual box as a platform to run mininet. A VirtualBox or VB is a software virtualization package that installs on an operating system as an application. VirtualBox allows additional operating systems to be installed on it, as a Guest OS, and run in a virtual environment. Using mininet we have created our custom topology that is fat-tree topolgoy and have used pox controller at the control plane.

1. MININET
2. POX CONTROLLER

MININET [11]

Mininet is a network emulator, or perhaps more precisely a network emulation orchestration system. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. A Mininet host behaves just like a real machine; we can ssh into it (if we start up sshd and bridge the network to your host) and run arbitrary programs (including anything that is installed on the underlying Linux system.) In short, Mininets virtual hosts, switches, links, and controllers are the real thing they are just created using software rather than hardware and for the most part their behavior is similar to discrete hardware elements. It is usually possible to create a Mininet network that resembles a hardware network, or a hardware network that resembles a Mininet network, and to run the same binary code and applications on either platform. Mininet supports parametrized topologies. With a few lines of Python code, we can create a flexible topology which can be configured based on the parameters you pass into it, and reused for multiple experiments.

POX CONTROLLER [12]

POX is an open source development platform for Python-based software-defined networking (SDN) control applications, such as OpenFlow SDN controllers. POX, which enables rapid development and prototyping, is becoming more commonly used than NOX

1. Pythonic OpenFlow interface.
2. Reusable sample components for path selection, topology discovery, etc.
3. Runs anywhere, Can bundle with install-free PyPy runtime for easy deployment.
4. Specifically targets Linux, Mac OS, and Windows.
5. Topology discovery.
6. Supports the same GUI and visualization tools as NOX.

POX started life as an OpenFlow controller, but can now also function as an OpenFlow switch, and can be useful for writing networking software in general.

POX officially requires Python 2.7 (though much of it will work fine with Python 2.6), and should run under Linux, Mac OS, and Windows.

3.3 IMPLEMENTATION

FAT-TREE TOPOLOGY

To implement Fat-tree topology we have used Mininet as platform. The Fat-tree topology is also called as k-ary or k-port Fat-tree topology Where k denotes the no. Of ports in the switches. A Fat-tree with k number identifies about no. of consisting core switches, aggregate switches and edge switches. For k-number of ports in a switch the number of core layer switch is $(k/2)$ and $(k/2)$ no. of pods , pods are the set of $k/2$ aggregate switches and $k/2$ edge switches. And then each aggregate switch is connected

with $(k/2)$ core layer switch and $(k/2)$ layer of edge switch And then each edge switch inside the pods are connected with $(k/2)$ no. Of hosts and also associated with $(k/2)$ no.of aggregate switches.

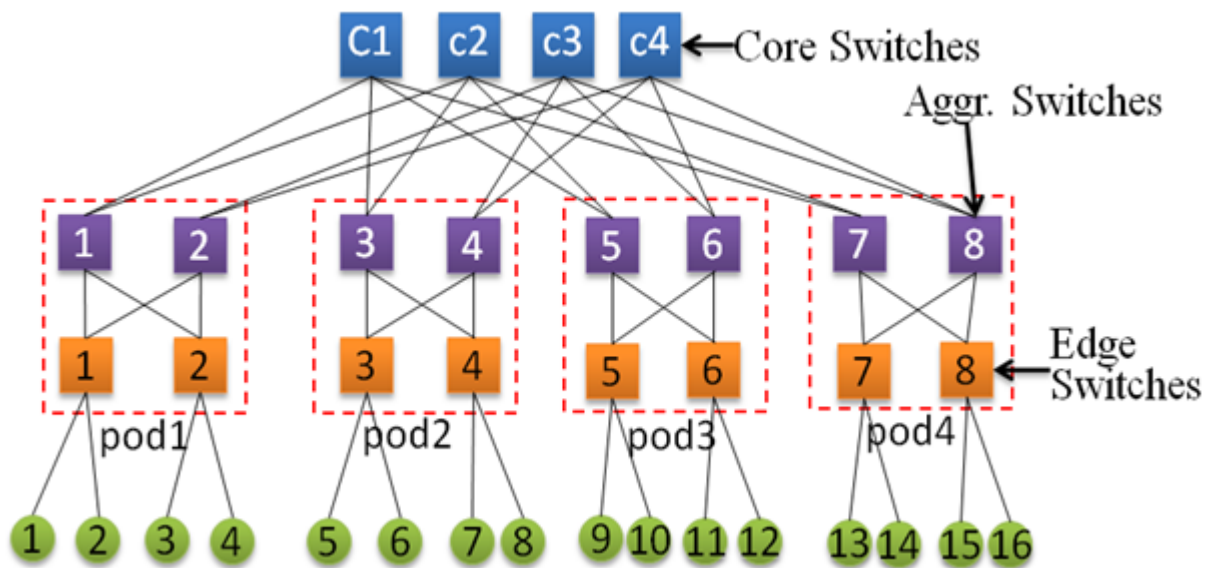


Figure 3.1: FAT-TREE

3.3.1 QUEUE IMPLEMENTATION ON HOST

After implementing Fat-tree we have pinged a server from multiple hosts and found that there was decreased in bandwidth. So to solve this problem we have created multiple queues on a single input ports. After creating queues we have again pinged a server with multiple host and found that there was slightly decreasing in the bandwidth as compared to earlier results.

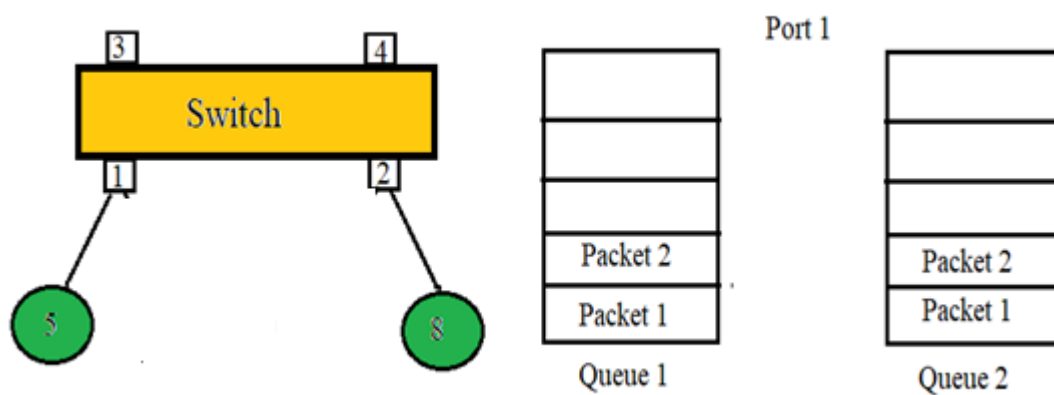


Figure 3.2: QUEUE CREATION

3.3.2 FLOW TABLE

The OpenFlow table is a data structure that resides in the high speed data plane of an open flow switch. Its contents determine the forwarding behavior and packet handling behavior of that switch. An open flow table has one or more flow entries.

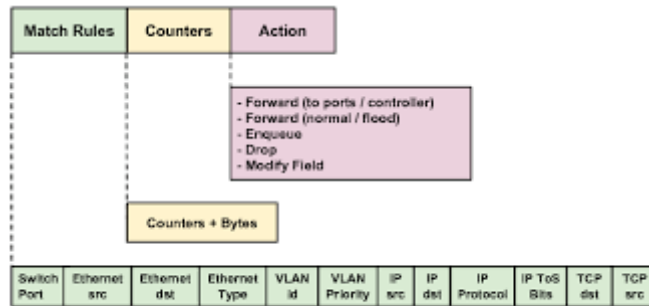


Figure 3.3: FLOW TABLE

WORKING

Initially a fat-tree topology of $k=4$ is created, which consists of 4 core switches 8 aggregate switches, 8 edge switches and 16 hosts. All operations have been done on this fat-tree topology only. In first experiment on this fat-tree topology, a server is requested from a host and the estimated bandwidth between them is resulted around 9.71 MBits/sec, again when the server is requested from another host at the same time, then there was decrease in throughput. we used Qos policy to deal with this problem. Egress traffic shaping is configured on the server, where two queues have been created on the interface of the edge switch connected to the server and an actions have been associated with both of the queue, for both of the host first action specifies to enter the host 1 in queue 1 and second action specifies to enter the host 2 in queue 2. once again when the server is requested from both of the host then there was no decrease in the bandwidth.

Second task of our project was to create a flow table for the switches, A flow table consists of flow entries, Each flow table entry contains:

1. match fields: to match against packets. These consist of the ingress port and packet headers, and optionally metadata specified by a previous table.
2. priority: matching precedence of the flow entry.
3. counters: updated when packets are matched.
4. instructions: to modify the action set or pipeline processing.
5. timeouts: maximum amount of time or idle time before flow is expired by the switch.
6. cookie: opaque data value chosen by the controller. May be used by the controller to filter flow statistics, flow modification and flow deletion. Not used when processing packets.

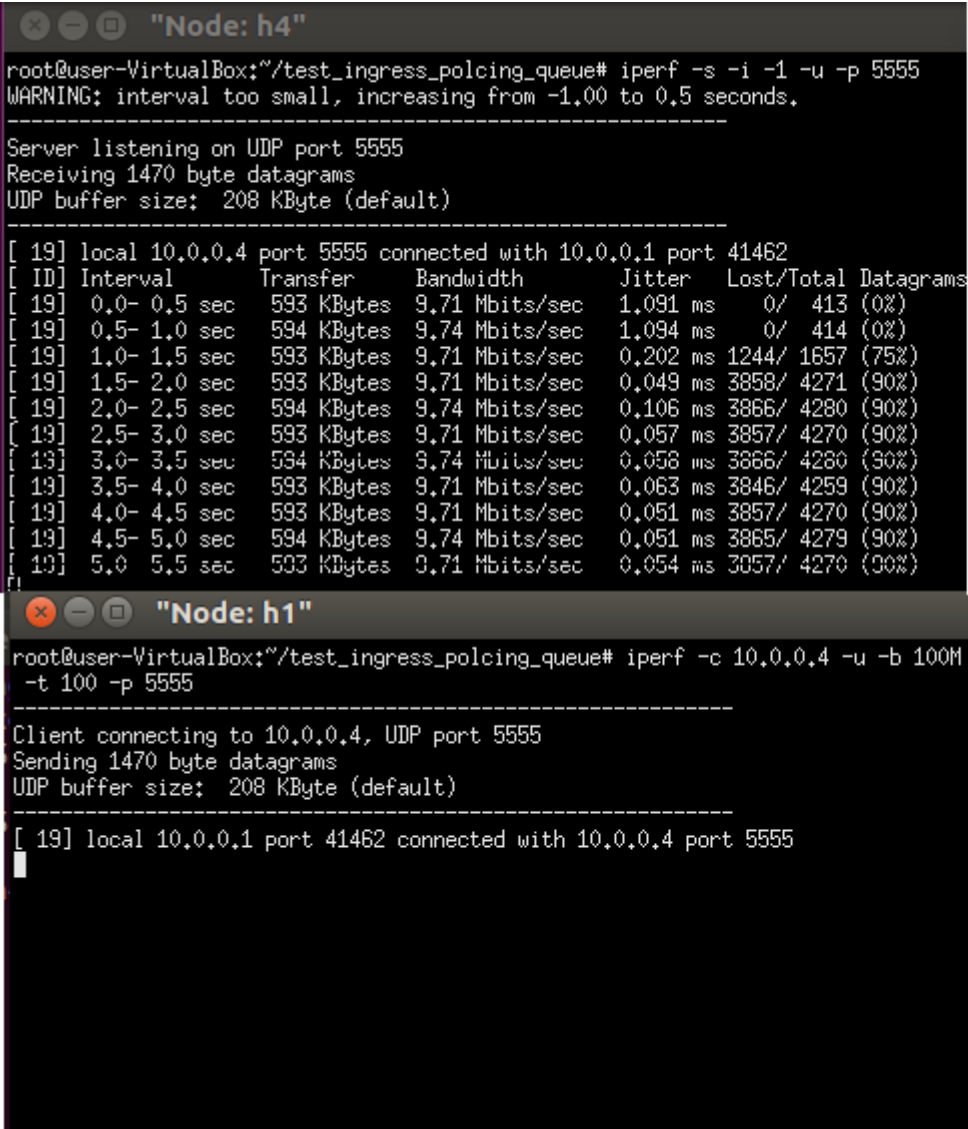
A flow table entry is identified by its match fields and priority: the match fields and priority taken together to identify a unique flow entry in the flow table. The flow entry that wildcards all fields (all fields omitted) and has priority equal to 0 is called the table-miss flow entry. a flow table is consisted of many flows and each packet arrived at the switch is matched with these flows in the flow table, and if a packet matches with any of the flow then corresponding action is applied and the counter is updated whenever a packet matches the flow,if a packet matches with two flows in a table then the flow with the highest priority is selected and if there is no flow associated with that packet then it checks for the table miss entry and the action associated with the table miss is performed on that packet.and if for a packet does not have flow entry and the table does not contain the table miss entry then the packet is dropped. We have created a flow table for the switches containing flows for different operations like switching,flow switching,firewall,routing and vlan switching.

CHAPTER 4

OUTPUT

4.1 RESULTS

We have created h4 as a server and h1 as a host. When host h1 sends packets to server h4, then the throughput is around 9.71 Mbps.



```
root@user-VirtualBox:~/test_ingress_polcing_queue# iperf -s -i -1 -u -p 5555
WARNING: interval too small, increasing from -1.00 to 0.5 seconds.
-----
Server listening on UDP port 5555
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 19] local 10.0.0.4 port 5555 connected with 10.0.0.1 port 41462
[ ID] Interval      Transfer    Bandwidth   Jitter    Lost/Total Datagrams
[ 19] 0.0- 0.5 sec  593 KBytes  9.71 Mbits/sec  1.091 ms  0/ 413 (0%)
[ 19] 0.5- 1.0 sec  594 KBytes  9.74 Mbits/sec  1.094 ms  0/ 414 (0%)
[ 19] 1.0- 1.5 sec  593 KBytes  9.71 Mbits/sec  0.202 ms 1244/ 1657 (75%)
[ 19] 1.5- 2.0 sec  593 KBytes  9.71 Mbits/sec  0.049 ms 3858/ 4271 (90%)
[ 19] 2.0- 2.5 sec  594 KBytes  9.74 Mbits/sec  0.106 ms 3866/ 4280 (90%)
[ 19] 2.5- 3.0 sec  593 KBytes  9.71 Mbits/sec  0.057 ms 3857/ 4270 (90%)
[ 19] 3.0- 3.5 sec  594 KBytes  9.74 Mbits/sec  0.058 ms 3866/ 4280 (90%)
[ 19] 3.5- 4.0 sec  593 KBytes  9.71 Mbits/sec  0.063 ms 3846/ 4259 (90%)
[ 19] 4.0- 4.5 sec  593 KBytes  9.71 Mbits/sec  0.051 ms 3857/ 4270 (90%)
[ 19] 4.5- 5.0 sec  594 KBytes  9.74 Mbits/sec  0.051 ms 3865/ 4279 (90%)
[ 19] 5.0- 5.5 sec  593 KBytes  9.71 Mbits/sec  0.054 ms 3057/ 4270 (90%)

root@user-VirtualBox:~/test_ingress_polcing_queue# iperf -c 10.0.0.4 -u -b 100M
-t 100 -p 5555
-----
Client connecting to 10.0.0.4, UDP port 5555
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 19] local 10.0.0.1 port 41462 connected with 10.0.0.4 port 5555
```

Figure 4.1: In the beginning, the h1 is sending to h4. The throughput is around 9.71Mbps

This time we have requested the server h4 from two host h1 and h3 at the same time. Initially when only h1 is sending traffic to h4 the throughput is around 9.71Mbps and when another host h3 sends traffic at the same time then the throughput between h1-h4 drops and result is around 4.10-5.80 Mbps.

```

"Node: h4"
[ 19] 1.5- 2.0 sec 594 KBytes 9.74 Mbits/sec 0.052 ms 3866/ 4280 (90%)
[ 19] 2.0- 2.5 sec 593 KBytes 9.71 Mbits/sec 0.076 ms 3857/ 4270 (90%)
[ 19] 7.5- 8.0 sec 593 KBytes 9.71 Mbits/sec 0.052 ms 3857/ 4270 (90%)
[ 19] 8.0- 8.5 sec 594 KBytes 9.74 Mbits/sec 0.056 ms 3866/ 4280 (90%)
[ 19] 8.5- 9.0 sec 593 KBytes 9.71 Mbits/sec 0.056 ms 3857/ 4270 (90%)
[ 19] 9.0- 9.5 sec 593 KBytes 9.71 Mbits/sec 0.050 ms 3857/ 4270 (90%)
[ 19] 9.5-10.0 sec 593 KBytes 9.74 Mbits/sec 0.100 ms 3866/ 4280 (90%)
[ 19] 10.0-10.5 sec 593 KBytes 9.71 Mbits/sec 0.047 ms 3857/ 4270 (90%)
[ 19] 10.5-11.0 sec 594 KBytes 9.71 Mbits/sec 0.063 ms 3866/ 4280 (90%)
[ 19] 11.0-11.5 sec 343 KBytes 5.82 Mbits/sec 0.036 ms 3989/ 4228 (94%)
[ 19] 11.5-12.0 sec 290 KBytes 4.75 Mbits/sec 0.052 ms 4068/ 4270 (95%)
[ 19] 12.0-12.5 sec 266 KBytes 4.35 Mbits/sec 0.076 ms 4116/ 4301 (96%)
[ 19] 12.5-13.0 sec 322 KBytes 5.27 Mbits/sec 0.052 ms 3856/ 4290 (95%)
[ 19] 13.0-13.5 sec 112 KBytes 1.83 Mbits/sec 0.164 ms 3866/ 1603 (95%)
[ 19] 13.5-14.0 sec 0.00 KBytes 0.00 bits/sec 0.164 ms 3857/ 4270 (91%)
[ 19] 14.0-14.5 sec 44.5 KBytes 729 Kbits/sec 0.177 ms 3857/ 4270 (91%)
[ 19] 14.5-15.0 sec 257 KBytes 4.21 Mbits/sec 0.042 ms 4092/ 4280 (90%)
[ 19] 15.0-15.5 sec 312 KBytes 5.10 Mbits/sec 0.034 ms 4063/ 4280 (90%)
[ 19] 15.5-16.0 sec 299 KBytes 4.89 Mbits/sec 0.035 ms 4062/ 4270 (90%)
[ 19] 16.0-16.5 sec 327 KBytes 5.36 Mbits/sec 0.034 ms 4041/ 4269 (94%)
[ 19] 16.5-17.0 sec 350 KBytes 5.74 Mbits/sec 0.052 ms 4037/ 4281 (95%)
[ 19] 17.0-17.5 sec 304 KBytes 4.99 Mbits/sec 0.035 ms 4057/ 4269 (95%)
[ 19] 17.5-18.0 sec 319 KBytes 5.22 Mbits/sec 0.028 ms 4048/ 4270 (95%)

"Node: h1"
root@user-VirtualBox:~/test_ingress_polcing_queue# iperf -c 10.0.0.4 -u -b 100M
-t 100 -p 5555
-----
Client connecting to 10.0.0.4, UDP port 5555
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 19] local 10.0.0.1 port 50782 connected with 10.0.0.4 port 5555

```

Figure 4.2: When only h1 sends traffic to h4 upto 11.0sec the throughput is constant.

```

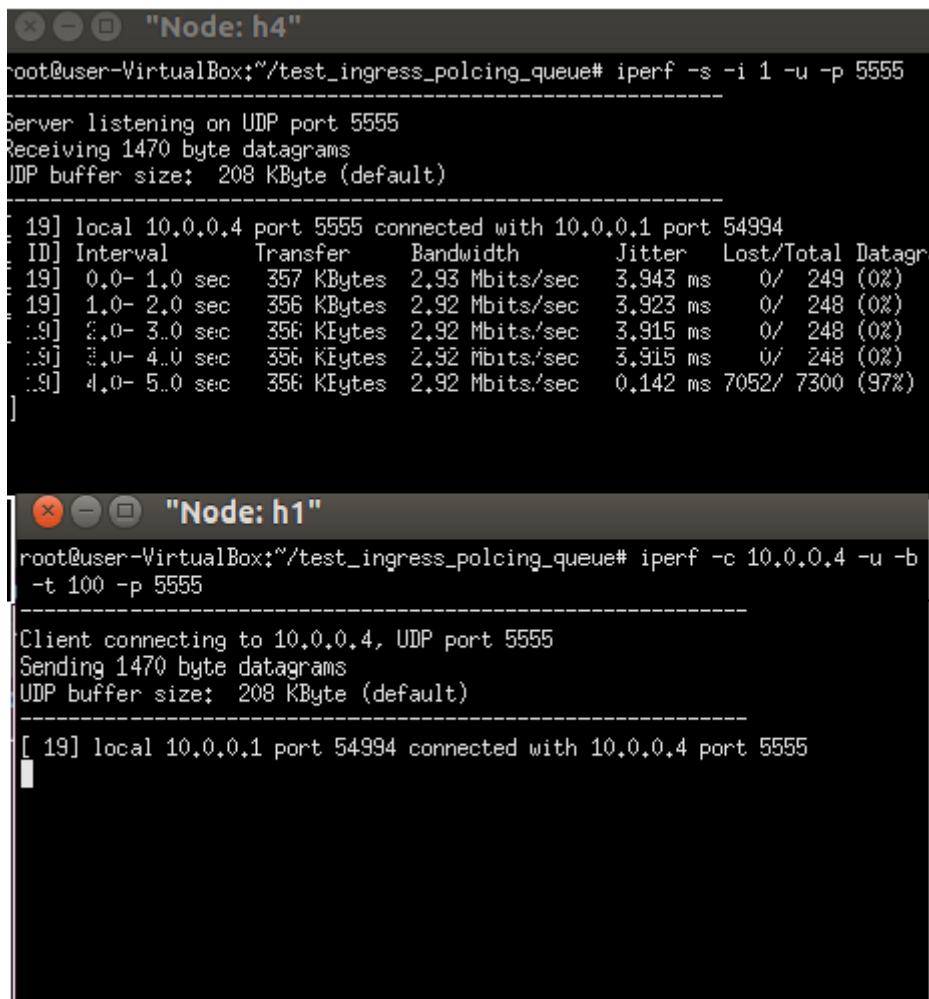
"Node: h4"
root@user-VirtualBox:~/test_ingress_polcing_queue# iperf -s -i 1 -u -p 6666
-----
Server listening on UDP port 6666
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 19] local 10.0.0.4 port 6666 connected with 10.0.0.3 port 53232
[ ID] Interval      Transfer      Bandwidth      Jitter    Lost/Total Datagrams
[ 19] 0.0- 1.0 sec   636 KBytes    5.21 Mbits/sec 0.050 ms  8070/ 8513 (95%)
[ 19] 1.0- 2.0 sec   584 KBytes    4.79 Mbits/sec 0.206 ms  8153/ 8560 (95%)
[ 19] 2.0- 3.0 sec   1.12 MBytes   9.38 Mbits/sec 0.172 ms  7761/ 8559 (91%)
[ 19] 3.0- 4.0 sec   808 KBytes    6.62 Mbits/sec 0.047 ms  7960/ 8523 (94%)
[ 19] 4.0- 5.0 sec   583 KBytes    4.77 Mbits/sec 0.033 ms  8142/ 8548 (95%)
[ 19] 5.0- 6.0 sec   511 KBytes    4.10 Mbits/sec 0.043 ms  8194/ 8550 (95%)
[ 19] 6.0- 7.0 sec   550 KBytes    4.57 Mbits/sec 0.045 ms  8099/ 8400 (95%)
:

"Node: h3"
root@user-VirtualBox:~/test_ingress_polcing_queue# iperf -c 10.0.0.4 -u -b 100M
-t 50 -p 6666
-----
Client connecting to 10.0.0.4, UDP port 6666
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 19] local 10.0.0.3 port 53232 connected with 10.0.0.4 port 6666

```

Figure 4.3: When the h3 starts to send the traffic to h4, the throughput of H1-h4 drops

We have configured egress traffic shaping on the switch connected to server h4, where we have created two queues q1 and q2 on the interface connected to server h4 for both the host h1 and h3. And again we have repeated the same process.



```
root@user-VirtualBox:~/test_ingress_polcing_queue# iperf -s -i 1 -u -p 5555
-----
Server listening on UDP port 5555
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 19] local 10.0.0.4 port 5555 connected with 10.0.0.1 port 54994
[ ID] Interval      Transfer    Bandwidth   Jitter    Lost/Total Datagr
[ 19] 0.0- 1.0 sec  357 KBytes  2.93 Mbits/sec  3.943 ms  0/ 248 (0%)
[ 19] 1.0- 2.0 sec  356 KBytes  2.92 Mbits/sec  3.923 ms  0/ 248 (0%)
[ 19] 2.0- 3.0 sec  356 KBytes  2.92 Mbits/sec  3.915 ms  0/ 248 (0%)
[ 19] 3.0- 4.0 sec  356 KBytes  2.92 Mbits/sec  3.915 ms  0/ 248 (0%)
[ 19] 4.0- 5.0 sec  356 KBytes  2.92 Mbits/sec  0.142 ms 7052/ 7300 (97%)
]

root@user-VirtualBox:~/test_ingress_polcing_queue# iperf -c 10.0.0.4 -u -b
-t 100 -p 5555
-----
Client connecting to 10.0.0.4, UDP port 5555
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 19] local 10.0.0.1 port 54994 connected with 10.0.0.4 port 5555
|
```

Figure 4.4: When only H1 is sending traffic to H4, this flow gets 2.92 Mbps.

Here host h1 and h3 sends traffic to the server h4 at the same time.

```

"Node: h4"
[ 19] 4.0-5.0 sec 356 KBytes 2.92 Mbits/sec 0.148 ms 7652/ 7360 (97%)
[ 19] 5.0-6.0 sec 356 KBytes 2.92 Mbits/sec 0.064 ms 8299/ 8547 (97%)
[ 19] 6.0-7.0 sec 356 KBytes 2.92 Mbits/sec 0.091 ms 8280/ 8528 (97%)
[ 19] 7.0-8.0 sec 356 KBytes 2.92 Mbits/sec 0.095 ms 8297/ 8545 (97%)
[ 19] 8.0-9.0 sec 356 KBytes 2.92 Mbits/sec 0.065 ms 8299/ 8547 (97%)
[ 19] 9.0-10.0 sec 356 KBytes 2.92 Mbits/sec 0.062 ms 8267/ 8510 (97%)
[ 19] 10.0-11.0 sec 356 KBytes 2.92 Mbits/sec 0.447 ms 8299/ 8547 (97%)
[ 19] 11.0-12.0 sec 356 KBytes 2.92 Mbits/sec 0.067 ms 8223/ 8471 (97%)
[ 19] 12.0-13.0 sec 356 KBytes 2.92 Mbits/sec 0.287 ms 8294/ 8542 (97%)
[ 19] 13.0-14.0 sec 356 KBytes 2.92 Mbits/sec 0.051 ms 8298/ 8543 (97%)
[ 19] 14.0-15.0 sec 356 KBytes 2.92 Mbits/sec 0.076 ms 8298/ 8546 (97%)
[ 19] 15.0-16.0 sec 356 KBytes 2.92 Mbits/sec 0.127 ms 8299/ 8547 (97%)
[ 19] 16.0-17.0 sec 356 KBytes 2.92 Mbits/sec 0.141 ms 8256/ 8504 (97%)
[ 19] 17.0-18.0 sec 356 KBytes 2.92 Mbits/sec 0.057 ms 8299/ 8547 (97%)
[ 19] 18.0-19.0 sec 356 KBytes 2.92 Mbits/sec 0.051 ms 8297/ 8545 (97%)
[ 19] 19.0-20.0 sec 356 KBytes 2.92 Mbits/sec 0.076 ms 8297/ 8545 (97%)
[ 19] 20.0-21.0 sec 356 KBytes 2.92 Mbits/sec 0.125 ms 8290/ 8548 (97%)
[ 19] 21.0-22.0 sec 356 KBytes 2.92 Mbits/sec 0.507 ms 8256/ 8546 (97%)
[ 19] 22.0-23.0 sec 356 KBytes 2.92 Mbits/sec 0.510 ms 8299/ 8547 (97%)
[ 19] 23.0-24.0 sec 356 KBytes 2.92 Mbits/sec 0.547 ms 8293/ 8545 (97%)
[ 19] 24.0-25.0 sec 356 KBytes 2.92 Mbits/sec 0.510 ms 8298/ 8545 (97%)
[ 19] 25.0-26.0 sec 356 KBytes 2.92 Mbits/sec 0.523 ms 8299/ 8547 (97%)
[ 19] 26.0-27.0 sec 356 KBytes 2.92 Mbits/sec 0.536 ms 8299/ 8547 (97%)

"Node: h1"
root@user-VirtualBox:~/test_ingress_polcing_queue# iperf -c 10.0.0.4 -u -b 100M
-t 100 -p 5555
-----
Client connecting to 10.0.0.4, UDP port 5555
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 19] local 10.0.0.1 port 54994 connected with 10.0.0.4 port 5555

```

Figure 4.5: When H1 is sending traffic to H4 at the same time with h3, the flow of H3-H4 is not affected by H1-H4 flow.


```

"Node: h4"
root@user-VirtualBox:~/test_ingress_polcing_queue# iperf -s -i 1 -u -p 6666
-----
Server listening on UDP port 6666
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[19] local 10.0.0.4 port 6666 connected with 10.0.0.3 port 54049
[19] ID Interval Transfer Bandwidth Jitter Lost/Total Datagrams
[19] 0.0- 1.0 sec 828 KBytes 6.79 Mbits/sec 1.581 ms 0/ 577 (0%)
[19] 1.0- 2.0 sec 831 KBytes 6.81 Mbits/sec 0.584 ms 1183/ 1762 (67%)
[19] 2.0- 3.0 sec 831 KBytes 6.81 Mbits/sec 0.566 ms 7972/ 8551 (93%)
[19] 3.0- 4.0 sec 830 KBytes 6.80 Mbits/sec 0.572 ms 7959/ 8537 (93%)
[19] 4.0- 5.0 sec 831 KBytes 6.81 Mbits/sec 0.574 ms 7958/ 8537 (93%)
[19] 5.0- 6.0 sec 831 KBytes 6.81 Mbits/sec 0.574 ms 7972/ 8551 (93%)
[19] 6.0- 7.0 sec 831 KBytes 6.81 Mbits/sec 0.564 ms 7972/ 8551 (93%)

"Node: h3"
root@user-VirtualBox:~/test_ingress_polcing_queue# iperf -c 10.0.0.4 -u -b 100M
-t 100 -p 6666
-----
Client connecting to 10.0.0.4, UDP port 6666
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[19] local 10.0.0.3 port 54049 connected with 10.0.0.4 port 6666

```

Figure 4.6: When H3 starts to send traffic to H4, the flow of H1-H4 is not affected by H3-H4 flow.

CHAPTER 5

CONCLUSION

The present research was planned to find out the efficient load balancing algorithm in Software Defined Networking (SDN) technology. Our project primarily presents about the load balancing approaches in software defined networking (SDN), and its implementation in pox controller. Load balancing strategy was designed in the SDN controller which deals with the various type of load effectively. Different load balancing parameters were utilized for comparing the performance of load balancing in software defined networking (SDN). Such parameters are response time, throughput, and availability.

In this project we have modified an given approach that is queue-length directed adaptive routing to queue-weightage directed adaptive routing.

To achieve our goal we have implemented some of the core steps and these are:

we have successfully configured QoS policy in the switches and have also experimented that how QoS policy can be used for load balancing. Next step was to create flow table for the Openflow switches, we have successfully created flow table by both of the approach without using the controller (directly accessing the switches) and by using the controller also. At the end of our project we summarize with better results for load balancing.

CHAPTER 6

FUTURE ENHANCEMENT

Future work concerns deeper analysis on queue-weightage directed adaptive routing. Our future work comprises of finding out the queue-weightage in real time to implement the queue-weightage directed adaptive routing and to consider optimum length of the queue for different type of traffics.

REFERENCES

- [1] Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti *"A Survey of Software Defined Networking: Past, Present, and Future of Programmable Networks"* Sep 2014
- [2] Yosr Jarraya, Member, IEEE, Taous Madi, and Mourad Debbabi, Member, IEEE *"A Survey and a Layered Taxonomy of Software Defined Networking"* Dec 2014
- [3] Fei Hu, Qi Hao, and Ke Bao *"A Survey on Software Defined Network and Open-Flow: From Concept to Implementation"* Nov 2014
- [4] Diego Kreutz, Member, IEEE, Fernando M. V. Ramos, Member, IEEE, Paulo Verissimo, Fellow, IEEE, Christian Esteve Rothenberg, Member, IEEE, Siamak Azodolmolky, Senior Member, IEEE, and Steve Uhlig, Member, IEEE *"Software-Defined Networking: A Comprehensive Survey"* Oct 2014
- [5] United States Patent *"Flow Based Queuing Of Network"* Oct 2011
- [6] John P. Lehoczky Department of Statistics Carnegie Mellon University Pittsburgh *"Real-Time Queueing Theory"*
- [7] Anand V Akella and Kaiqi Xiong *"Quality of Service (QoS) Guaranteed Network Resource Allocation via Software Defined Networking (SDN)"* 2014
- [8] Yuanhao Zhou, Mingfa Zhu, Limin Xiao, Li Ruan, Wenbo Duan, Deguo Li, Rui Liu, Mingming Zhu *"A Load Balancing Strategy for SDN Controller based on Distributed Decision"* 2014
- [9] Seunghoe Gu, Jonghwan Kim, Younghoon Kim, Ikjun Yeom *"Controlled Queue Management in Software-Defined Networks"*
- [10] Rihab JMAL and Lamia CHAARI FOURATI *"Implementing Shortest Path Routing Mechanism using Openflow POX Controller"*

- [11] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet De-meester *"In-Band Control, Queuing, and Failure Recovery Functionalities for OpenFlow"*
- [12] Santosh Mahapatra Xin Yuan *"Load Balancing Mechanisms in Data Center Networks"*
- [13] Karamjeet Kaur, Japinder Singh and Navtej Singh Ghumman *"Mininet as Software Defined Networking Testing Platform"*
- [14] Jose David Anzanello *"Load Balancing Strategy For SDN Controller Based On Distributed Decision"* Jun 2018
- [15] By Yagiz Kaymak, Roberto Rojas-Cessa *"Per-Packet Load Balancing In Data Center Networks"* Nov 2015
- [16] Hilmi E. Egilmez, Student Member, IEEE, and A. Murat Tekalp *"Distributed QoS Architectures for Multimedia Streaming Over Software Defined Networks"* Oct 2014
- [17] Liming Wang, Gang Lu *"The Dynamic sub-topology load balancing algorithm for Data center network"*
- [18] K Jeong, J Kim, YT Kim *"QoS-aware Network Operating System for Software Defined Networking with generalized openflows"* 2012
- [19] W Braun, M Menth *"Software Defined Networking using openflow: protocols, applications and architectural design choices"* 2014
- [20] C Arad *"Method for weighted Load Balancing Among network interfaces"* 2011
- [21] H Kim, N Feamster *"Improving Network Management using Software Defined Networking"* 2013