International Journal of Advanced Engineering, Management and Science (IJAEMS)
Peer-Reviewed Journal
ISSN: 2454-1311 | Vol-9, Issue-9; Sep, 2023
Journal Home Page: https://ijaems.com/
Article DOI: https://dx.doi.org/10.22161/ijaems.99.1

Data Centre Network Optimization

Tajammul Hussain
itajammuleng@gmail.com

Received: 11 Jul 2023; Received in revised form: 15 Aug 2023; Accepted: 25 Aug 2023; Available online: 03 Sep Aug 2023

Abstract— *Technology has advanced quite quickly in recent years, and IT and telecom infrastructure is constantly growing. Due to an increased importance and emphasis of cloud computing many growing and emerging enterprises have shifted their computing needs and wants ultimately to the cloud, which leads to increase in inter-server data trafficking and also the bandwidth required for Data Centre Networking (DCCN). Actually this multi-tier hierarchical architecture used in modern data centres is based on traditional Ethernet/fabric switches. Researchers goal was to improve the data centre communication network design such that the majority of its problems can be solved while still using the existing network infrastructure and with lower capital expenditures. This is achieved through the deployment of OpenFlow (OF) switches and the Microelectromechanical Systems (MEMS) based all Optical Switching (MAOS). This will be beneficial in decreasing network latency, power consumption, and CAPEX costs in addition to helping with scalability concerns, traffic management, and congestion control. Additionally, by implementing new virtualization techniques, we may enhance DCCN's resource consumption and cable problems. In order to address data centre challenges, the researcher came up with an entirely new novel flat data centre coordination network architecture which is named as "Hybrid Flow based Packet filtering" (HFPFMAOS), forwarding, and MEMS which is entirely based on optical switching, and will be finally controlled by Software Defined Network (SDN) controller.*

Keywords— *Data Centre, Network Optimization, Software defined Networking, OpenFlow*

I. INTRODUCTION

1.1 Background

Data Centre is specialized building that houses various IT resources, such as servers, data storage systems, and networking and communication devices. Data centres have become increasingly important in IT and data networks over the last few years as a result of the explosive expansion of information technology and internet consumption. Additionally, as a result of the emergence of cloud computing, server virtualization, social media, and mobile data, the number of servers deployed, storage space, and interconnected network equipment in data centres is growing tremendously. Data centre IT network's primary elements include:

- Data Centre Communication Network
- High Performance Computing Network
- Storage Area Network

Three categories of traffic can be used to categorize a Data Centre.

- Traffic flows which are within the data centre.
- Traffic flows which travel between data centres.
- Traffic flows between data centre and the end user.

Data centre IP traffic is increasing at a fast rate [1] as depicted in the below figure. The primary driver of this expansion is cloud computing, which by 2023 is predicted to account for 4/5 of all traffic in data centres [2].

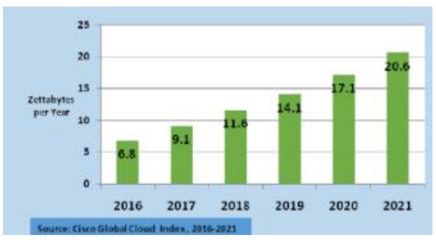


Fig 1.1: Global Data Centre Traffic, 2016–2021

Global data centres traffic by destination is shown in the figure below.

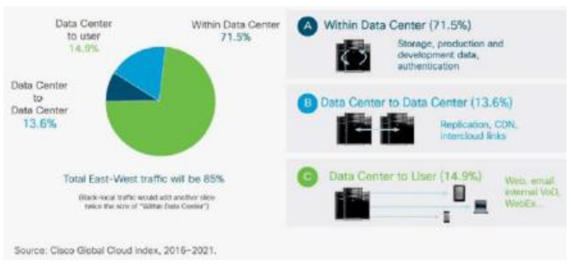


Fig 1.2: Global Data Centres Traffic by Destination

It is evident from the above chart that the majority of traffic occurs inside data centres, with the majority of that traffic travelling between servers and edge switches. Figure below depicts the traffic forecast between the various tiers of the data centre communication network.

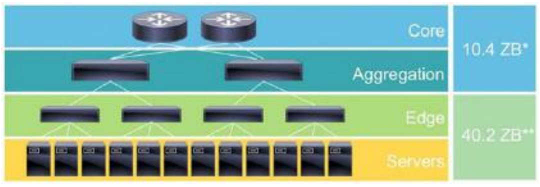


Fig 1.3 Traffic Forecast at Different Layers

1.2 Problem Statement

Limited server capacity and high oversubscription rates, network congestion and latency issues, management of internal data-centre traffic, support for a variety of traffic patterns, scalability, agility, effective resource utilization, management, fault handling, and troubleshooting are among the problems that DCCN must deal with. When building a data centre, these issues must be taken into consideration. The problems or difficulties that the DCCN will encounter are as follows:

- Limited capacity between servers and a high oversubscription rate as traffic moves through the layers of switches in a hierarchical fashion.
- Problems with congestion and high network latency have a negative impact on delay-sensitive applications and affect network performance as a whole.
- Controlling the nearly 75% of the overall traffic that is generated inside the data centre (i.e., east-west traffic between racks).
- Ease of Management, fault handling and troubleshooting in the network.
- Support for a range of traffic patterns, such as long-lasting, high-bandwidth "elephant" flows and short-lasting persistent "mouse" flows
- As a network grows, difficulties with scalability, agility, and efficient resource use arise.

1.3 Research Contribution

It relates to developing a low-cost, capital-efficient solution that will improve the scalability, traffic management, congestion control, and resource usage of data centre design. Academic contribution relates to proposing a solution HFPFMAOS using OpenFlow tools for:

- Congestion Reduction
- Improvement in Network Latency
- Implementing Scalability
- Traffic Management
- Reducing power Consumption
- CAPEX Reduction

Practical contribution of this research relates to coming up with an innovative data centre topology architecture with improved performance.

1.4 Objectives

- Application of Hybrid Flow to packet filtering /Forwarding and MEMS on the basis of an optical switching solution by utilizing different virtualization techniques.
- Efficient utilization of data centre network resources.
    - o Reducing Congestion
    - o Improving Network Latency
    - o Implementing Scalability
    - o Traffic Management
- Reducing Power Consumption and saving CAPEX.
- Provisioning of centralized intelligence of whole network.

1.5 Research Questions

- How to come up with a solution based on HFPFMAOS and MEMS based all optical switching?
- How to achieve efficient utilization of data centre network resources?
- How to reduce congestion, improving network latency, scalability and traffic management?
- How to reduce power consumption and save CAPEX?
- How to implement centralized intelligence of whole network?
- How to implement network virtualization techniques?

1.6 Research Methodology

OpenFlow (OF) switches were used in the access layer as TOR switches. OF messages were captured using Wire shark. Traffic was generated between various hosts (i.e. server machines) using iperf testing tool. Performance of the network and the delays were calculated/analyzed. For clarity, a reference data centre topology was used. Aggregation of the MEMS switch with conventional switch was done and connected it to the OpenFlow (OF) switches which were installed at the access layer. Network traffic was generated between various hosts and flow delays were observed. Researcher then directed flows via using MEMS switch and simultaneously monitored performance of the network.

1.7 Software's

The researcher used the following software's for carrying out the requisite analysis:

a) MININET
b) Putty
c) Wireshark
d) Oracle VM Virtual Box
e) Miniedit
f) ODL Controller
g) Xming

II. DATA CENTRE (DC)

2.1 Introduction

Data centre is basically an entirely one of its own kind of building which is invented to contains, supervises, and provides [3]. A data centre structure includes very unique building infrastructures, backup plans for power shortages or unavailability, cooling agents for systems, special equipment Chester's, servers, mainframes, and High performance Computing (HPC) networks to handle variety of communication data, as well as well-structured sophisticated cabling, application of various software, monitoring centers, and well equipped physical security systems.

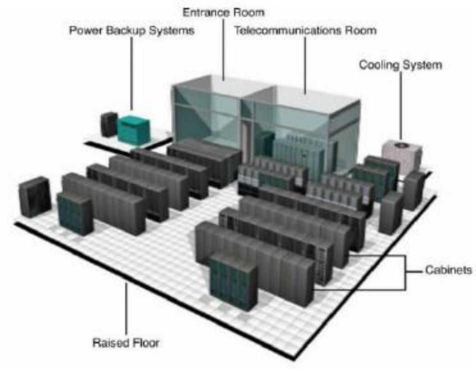


Fig. 2.1 Basic Data Centre Diagram

2.2 Evolution

First Data centre phase (called Phase 1.0) began in the 1950s and consisted of computer rooms with mainframe systems' CPUs and peripheral devices including storage, terminals, and printers, among other things. These monolithic software-based centralized systems give users little control over IT infrastructure and make heavy use of available resources. Phase 2.0 of data centres begins in the 1980s when client-server application models gain prominent. Currently, servers have replaced main frame computers, which are fairly compact and accessible through client PC-installed apps.

In this phase, servers that carry out specific tasks are typically installed close to clients rather than the main IT infrastructure to avoid paying excessive bandwidth costs. When the internet began to take off in the 1990s and the use of web-based applications increased, this mandated that servers be placed centrally in properly constructed data centres. Thus, the data centre that arises from the mainframe computer room has gained significance in the 1990s. Fig. 2.2 [3] depicts the timeline of data centre evolution.

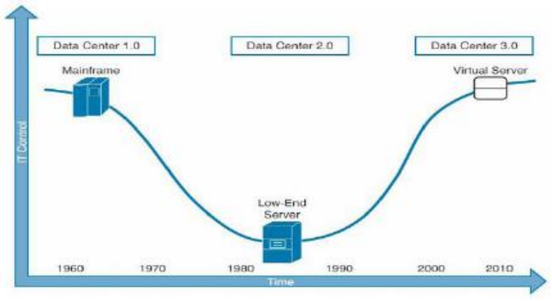


Fig. 2.2 Evolution Phases

Phase 3.0 began around the year 2000, and because technology evolved so quickly at this time, datacenter space, electricity, and the IT network are beginning to reach capacity. As a result, it is expensive to expand existing facilities or to build new ones. According to a 2005 Cisco IT research, around this time DC networks and servers were typically only used at 20% of their capacity. Application silos with discrete sets of servers, networks, and storage resources were the primary contributors to this problem. Later, a number of network consolidation projects are completed that provide new features, enhance resource utilization, decrease the number of network components and processes, and improve operational effectiveness.

2.3 Types

The are two in number [4][5]

- Private

- Cloud Based

2.3.1 Private

It is the on-site hardware that offers computing services and stores data within a local network that is managed by an organization's IT department. These are owned and run by small private/public businesses, governmental organization’s etc.

2.3.2 Cloud Based

Another name used for cloud data centres is co-location/managed services provider. Co-located Data Centers were created and maintained to offer specialized infrastructure and provide various services to external sources and parties. Major elements that force businesses to either go for cloud computing or develop their own data centers are:

- Business market requirements
- Data privacy issues
- The increasing cost of related equipment/infrastructure

2.4 Data Centre Communication Network (DCCN)

Data services and data transit from the server to clients or other servers are the major goals of DCCN. These qualities are now required from the perspectives of dependability, expansion, and efficiency for data centre communication networks.

- Availability
- Scalability
- Flexibility
- Efficiency
- Predictability

Ethernet is the most ubiquitous and well-liked DCCN protocol for data lines, with an interface range ranging from 10Mbps to 100Gbps. However, as the volume of traffic through the DCCN increased, various constraints compelled the development and adoption of new virtualization technologies as well as the optimization of the data centre network through the use of cutting-edge networking strategies like MEMS all optical switching and conventional packet switching.

2.5 DC Tiers

Various Tiers of Data Centres are shown in the figure below from Tier-1 to Tier-4.

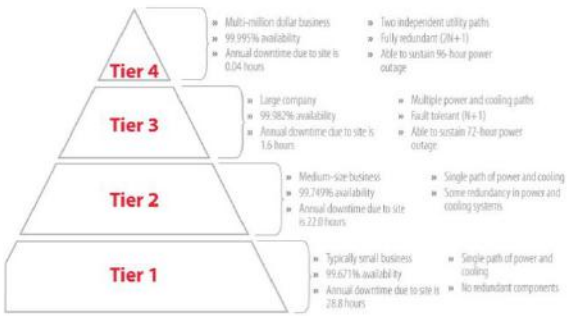

Tier 4
Multi-million dollar business
99.995% availability
Annual downtime due to site is 0.04 hours
Two independent utility paths
Fully redundant (2N+1)
Able to sustain 96-hour power outage
Tier 3
Large company
99.982% availability
Annual downtime due to site is 1.6 hours
Multiple power and cooling paths
Fault tolerant (N+1)
Able to sustain 72-hour power outage
Tier 2
Medium-size business
99.749% availability
Annual downtime due to site is 22.0 hours
Single path of power and cooling
Some redundancy in power and cooling systems
Tier 1
Typically small business
99.671% availability
Annual downtime due to site is 28.8 hours
Single path of power and cooling
No redundant components

Fig. 2.3 DC Tiers

2.6 Design Factors for Data Centre Network

- The following factors must be taken into account when planning and deploying DCCN architecture [3].
- Failure Impact & Application Resilience: All local and distant users will be unable to access their apps if DCCN fails.
- • Connectivity between the server and host: Servers must be linked together using several redundant Ethernet connections.
- Traffic Direction: In DCCN, the majority of traffic is between servers inside the data centre.
- Agility: This term refers to the capacity for assigning any service or application to any server at any moment in the data centre network while maintaining sufficient performance isolation and security amongst various applications.
- Growth Rate: Increasing number of servers, switches, and switch ports, etc., as customers and their data traffic increases.
- Application Bandwidth Demand & Oversubscription in case demand increases.

2.7 DCCN Challenges

Challenges for the Data Centers Communication Network fields are as follows:

- Due to the fact that router links carry traffic going in and out of the data centre, the bandwidth available over these links for server communication across various data centre areas is constrained [8].
- Network congestion and latency issues. As in a data centre, where many applications are active and producing distinct packet flows that move both within and across the DCCN, a multi-tier hierarchical structure with numerous hops, there are times when packet aggregation creates a bandwidth constraint. As the receiver's buffer space for absorbing packet

flows is running short, the incoming rate from numerous transmitters surpasses the rate at which it can handle packet flows. This congestion causes the delay time to lengthen and the receiver to begin dropping packets, which has an impact on various applications, particularly those that depend on latency, and lowers the network's overall performance.

- Managing the read/write, backup, and replication traffic that moves within the data centre due to the separation of application servers, databases, and storage, which accounts for over 75% of all traffic (i.e., east-west traffic between racks). Additionally, because jobs are distributed across numerous servers and assigned in parallel processing, DCCN's internal traffic is increased.
- Support for a range of traffic patterns, such as long-lasting, high-bandwidth "elephant" flows and short-lasting persistent "mouse" flows. Examples of entirely large data producers include particle accelerators, planes, trains, metros, undergrounds, self-driving cars, patient data in healthcare centers, etc. In the year 2014, the Boeing 787 produces 40 terabytes of onboard data every hour, of which 0.5 terabytes per hour is transmitted to a data centre. The majority of flows within a data centre are typically mouse flows, while only a few elephant flows contain the majority of the data. Managing every sort of traffic at once while keeping the overall network delay within bounds is therefore a major issue.
- As a network grows, difficulties with scalability, agility, and efficient resource use arise. In addition to making fault handling and debugging more challenging as the number of communication devices increases, this will also result in an increase in management overhead bytes on the network.
- As additional devices are added, energy requirements and consumption will increase. Currently, in traditional data centers, the top of the rack (TOR) switch, also known as the fabric interconnect, is where all of the servers in a rack are connected. Each rack has a number of servers, either rack mount servers or chassis-based servers. On one side, this aggregation layer provides communication between data centre clusters, allowing packet-based East-West traffic to pass through, while on the other, it also provides connectivity with the core network, allowing North-South traffic to pass. As a result, the aggregation layer is the layer where problems arise because 75% of the data center's east-west traffic passes through it, causing a bandwidth bottleneck & an increase in latency.

2.8 DCCN Topologies

The architecture of a data centre communication network (DCCN), which serves as the foundation for its applications, must be scalable, reliable, latency-free, and have adequate capacity to prevent network congestion. These qualities heavily rely on the network architecture in which the DCCN is installed and are crucial to the overall effectiveness of the data centre. Figure below depicts the Common DCCN Architecture, as stated by Cisco [9].

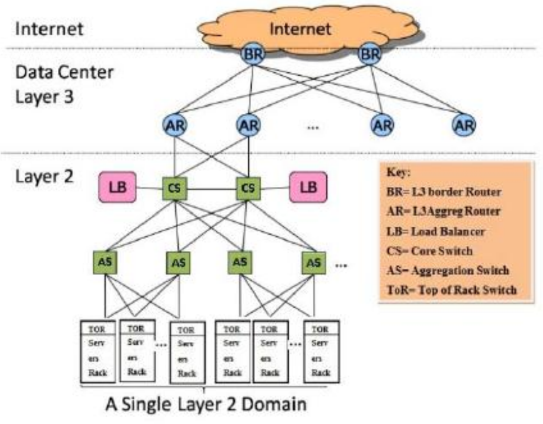


Fig 2.4 Common DCCN Architecture

Layered architecture has the advantage of enhancing the flexibility, modularity, and resilience of networks. Every layer in this design performs a separate role for unique profiles, such as the routers in the core layer (i.e., the aggregation router and the border router), which use routing protocols to decide how to forward traffic for both ingress and egress. Core switches, which offer incredibly flexible, scalable connections with numerous aggregation layer switches, make up the Layer 2 domain. This layer serves as a default gateway and a focal point for server IP subnets. Between numerous aggregation layer switches, it is typically used to transfer traffic between servers, and stateful network devices such server load balancers and firewalls are attached to this layer. The switches that make up the access layer are typically used by servers to connect to and communicate with one another. It contributes to better network administration and is in charge of exchanging any kind of traffic between servers, including broadcast, multicast, and unicast.

The bottom of the above illustration shows a server layer stacked in server racks.

Numerous Virtual machines are being operated by these servers and assigned to various data centre applications. To manage and respond to requests from external clients arriving through the internet, an application is typically linked to several public IP addresses. These requests are split up among the pool of servers for processing by specialized equipment called a load balancer that is redundantly connected to the core switches [10].

2.8.1 Fat-Tree

Al-Fareset and colleagues [11] have proposed this DCCN architecture. It makes use of the fat-tree idea and aims to boost fault tolerance, end-to-end cross sectional bandwidth, scalability, and cost-effectiveness. With this topology, the core and several pods make up the entire infrastructure. Servers, TORS (Access switches), and aggregation switches make up the pods.

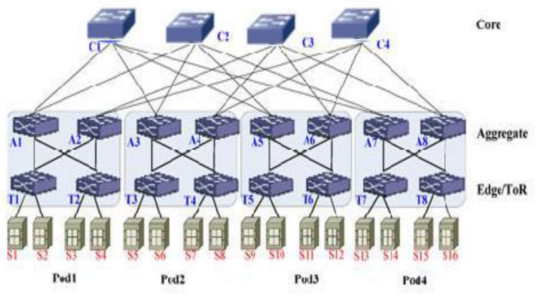


Fig 2.5 Fat-Tree

The topology adapts the data lines in the DCCN architecture to give a customized IP address scheme and multipath routing algorithm. Below figure compares 3 tier and fat-tree systems.

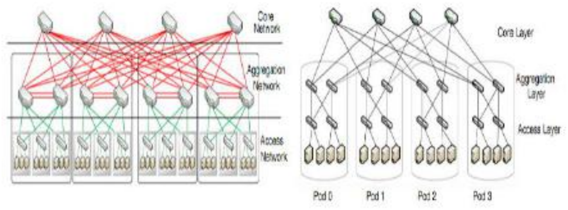


Fig 2.6 Comparison of 3 Tier/Fat-Tree Topologies

2.8.2 B-Cube

The structure [12] is suggested for the modular DCs created within containers and offers quick installation and migration, but scalability is limited because container data centres are not meant to be scaled up. Layers of COTS (commodity off the shelf) switches and servers make up this architecture, and they are in charge of packet forwarding. At level 0, the BCube architecture is made up of many BCube0 modules, each of which has a switch with n ports connected to n servers. n switches make up the BCube1 module at level 1, which is connected to n BCube0 modules or networks. One server from the BCube0 module is linked to each switch of BCube1.

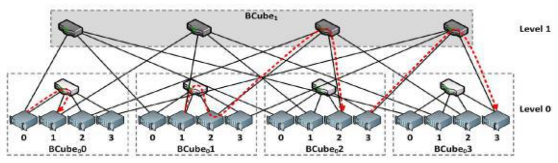


Fig 2.7 B-Cube Topology

2.8.3 DCell

Guo et al proposed a recursively specified design. Mini switches and servers are used in this very marketable design to forward packets. According to below figure, Dcell1 is made up of n+1 DCell0 modules, each of which is connected to the others by a single link via its servers.

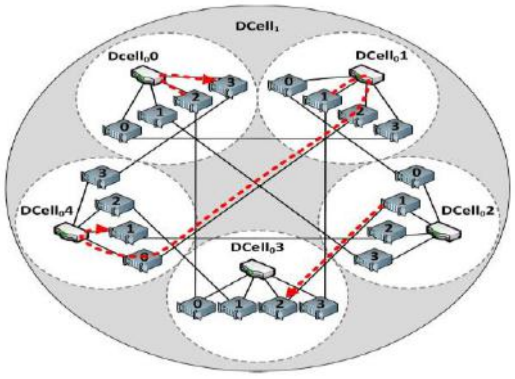


Fig 2.8 DCell Topology Architecture

2.8.4 VL2

A Fat-Tree based DCCN design called Virtual Layer 2 (VL2) aims to provide a flat automated IP addressing scheme that makes it possible to install servers anywhere in the network without manually configuring their IP addresses. This makes it possible for any service to be allocated to any network server. In order to scale to a large server pool, it makes use of [14] address resolution based on end systems.

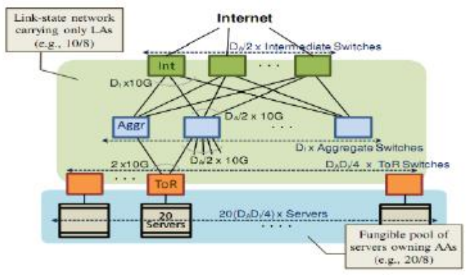

Link-state network carrying only LAs (e.g., 10/8)
Internet
DA/2 x Intermediate Switches
Int
DI x10G
Aggr
DA/2 x 10G
DI x Aggregate Switches
2 x10G
DA/2 x 10G
DADI/4 x ToR Switches
ToR
20 Servers
20(DADI/4) x Servers
Fungible pool of servers owning AAs (e.g., 20/8)

Fig 2.9 Network Architecture VL2

In addition to all architectures, there is a need for architecture which can handle the present issues the DCCN is encountering as well as give backward compatibility with existing infrastructure and its architecture. In this thesis, I suggested the HFPFMOS data centre design, which can partially address the shortcomings of the existing architecture while simultaneously being backwards compatible with it.

2.8.5 HFPFMAOS

The data centre communication network architecture (DCCN) called HFPFMAOS, is suggested as a solution to the various problems that data centres in the present face while maintaining the infrastructure. Particularly,in this architecture, standard switches are used at the aggregation layer. By monitoring all of the flows across the OF equipped switches, this will give us consolidated network intelligence and enable us to dynamically establish High bandwidth data paths between the data centre servers whenever and wherever they are needed. Additionally, it will lower network management overhead bytes and aid in quick defect detection and correction. The conceptualized design is shown below.

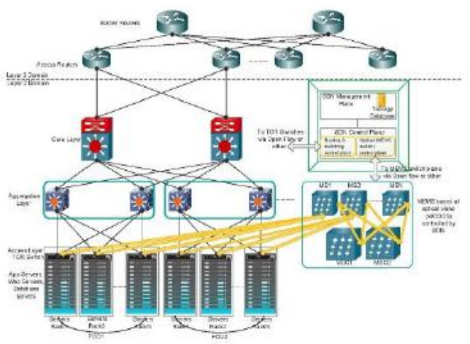


Fig 2.10 Architecture – HFPFMAOS - DCCN's

III. MEMS OPTICAL SWITCHING

3.1 Introduction

Interconnection between DCN demands a significant capacity increase due to the exponential daily growth of traffic. Therefore, the best way to relieve network congestion caused by electronic switched is to use optical switching. Although optical fibre has a very high bandwidth, it is constrained by electronic switching and transmission capacity. As switching is done in the optical domain, all optical switching can therefore play an important role.

3.2 OOO Switching (All Optical)

Modifying optical cross connections, a circuit is created from the ingress node to the egress node in all optical switching, and data travels via this circuit entirely in optical form. This can assist in addressing electronic switched network bottleneck since data is switched in the optical domain directly rather than going through many optical-electrical-optical conversions.

3.3 OOO Switching Benefits

- POD-based architecture is typically used in data centres, which results in low use of computing resources. However, optical switching enables sharing of computing resources among several PODs for optimal effectiveness.
- Increases revenue by quickly deploying new services.
- Less power is lost compared to a traditional electrical switch.
- On-demand capacity creation and reallocation.
- A remarkable increase in the data centre's operational efficiency as a result of the smooth functioning of applications and the efficient use of computational resources and 3D optical MEMS switching technology.
- Boost defect detection and quality of service.

3.4 Technology using OOO Switching

Different optical switching methods [17] are used by many switches. The same are shown in the below table:

Table 3.1 Contrast of various Optical switching technologies

	MEMS	Liquid-crystal	Bubble	Electro-Holographic
Wavelength Range	1290 ~ 1625 nm	1525 ~ 1575 nm	1270 ~ 1650 nm	1310 ~ 1550 nm
Insertion Loss (IL)	1dB ~ 5dB	<1dB	<7.5dB	<3dB
Scalability	Higher scalability with 3D MEMS to thousands of ports but limited with 2D MEMS to about 64x64	low	Low (to about 100 ports)	Low
Switching Speed	1ms ~ 5ms	4ms	<10 ms	<30ns
Power Consumption & Technical Performance	Low power consumption	Low power consumption about 50mW, Prefer for single mode instead of multimode, require polarization splitter, suffer from PMD, thermal fluctuation and moisture sensitivity	High power consumption about 25W, Wavelength dependent phase shift caused by bubble and hence amplitude dispersion in signal output, tradeoff between low loss and high isolation required.	Low power consumption, Polarization and wavelength dependent, hence dispersion and loss.
Cost & Packaging	Low cost & light weight	Costly & inconvenient	Low cost but some degree of precision required	Low cost & easily integrated with signal equalization & monitoring
Polarization Dependent Loss	< . 2dB	. 2dB	< . 3dB	
Maturity of Technology	Medium but Mature	Medium	Medium	Mature

3.5 Micro Electro Mechanical Systems (MEMS)

The branch of engineering MEMS (Micro electro mechanical system) which merged computer technology with minuscule mechanical components found in semiconductor chips, including actuators, gears, sensors, valves, mirrors, and valves [31]. It contains mechanical components, like micro-mirrors or sensors which h can be easily adjusted as needed, and reflect optical signals from input to output fibres placed within a tiny silicon chip that embedded micro-circuitry. Light beam can be reflected from the input to output by tilting/rotating the mirror at varied angles thus directing the traffic to the ports.

3.6 Design and Principle

Below figure shows the Structure of 3D MEMS optical switch [16].

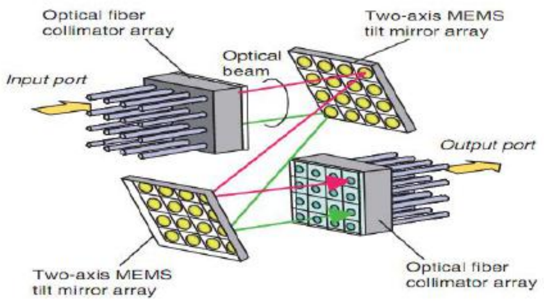

Optical fiber collimator array
Two-axis MEMS tilt mirror array
Optical beam
Input port
Output port
Two-axis MEMS tilt mirror array
Optical fiber collimator array

Fig. 3.1 Optical Switch

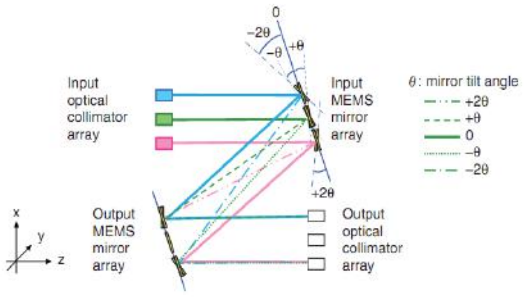


Fig 3.2 Standard Format OS

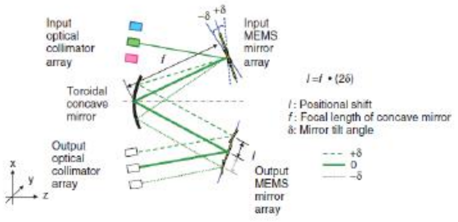


Fig 3.3 A toroidal concave mirror is used in a 3D MEMS optical switch.

The tilt angle of MEMS mirror measures the incident angle at which the optical beam from the I/P collimator array meets the concave mirror after reflecting by the I/P MEMS mirror array. Then light beam is again converged with a shift in position "/" by this toroidal concave mirror, that does an optical Fourier transform. This shift"/" can be quantitatively expressed as follows using the concave mirror focal length and the MEMS mirror tilt angle:

$$/ = f \times 2\delta$$

512 ports can be obtained by using a 2X2 array of collimator arrays with 128 optical ports and MEMS mirror arrays with 128 ports. This makes it easier to fabricate high-capacity optical switches with a negligibly small cumulative pitch error.

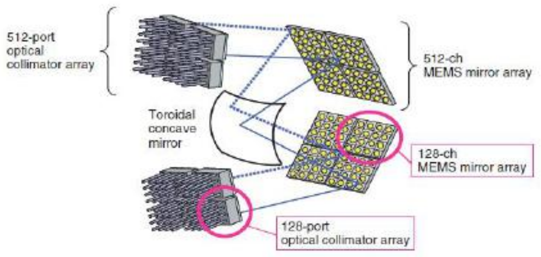


Fig 3.4 Schematic of 3D 512 MEMS optical ports switch

3.7 MEMS Mirror Structure

The same is shown in the below figure.

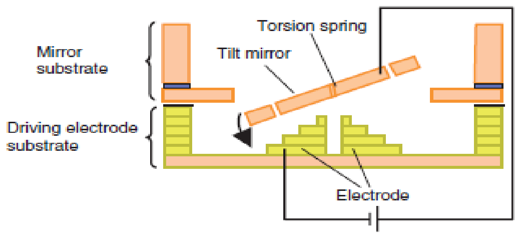


Fig 3.5 MEMS tilt mirror cross-sectional schematic

In order to generate an air gap between the electrodes and the mirrors, substrates are individually produced, then linked together by flip chips. Electrostatic force is produced when electrodes and mirrors are connected by a voltage, and this force moves the mirrors. Therefore, we are able to adjust the tilt angle of these mirrors by supplying a driving voltage to each electrode.

3.7.1 Mirror Substrate

They are formed of a single silicon crystal, giving the mirror incredibly steady movement. MEMS mirror is supported on X-Axis by two folded torsion springs, and the gimbal ring is connected to the base on the Y axis by a second pair of folded torsion springs [16]. The mirror cannot be dragged down and come into contact with the electrodes because of the great rigidity of these springs in the Z direction relative to torsion direction. As a result, moving the mirror on the X and Y axes makes it simple to reflect an optical beam in 3D space in a particular direction.

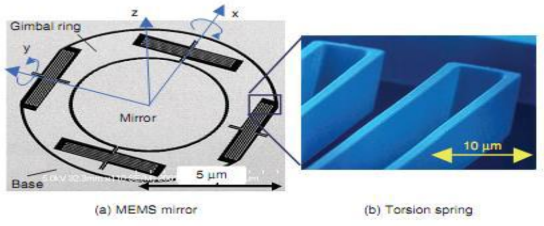


Fig 3.6 High aspect torsion spring and MEMS mirror are snapped together via SEM

3.7.1.1 Mirror Electrode Substrate Fabrication Method

A layer of polyimide is then spun over the created mirror pattern in step 7[16] (a) (b). The third process involves dry etching to create the pattern for the mirror opening (d) and resistant mask (c) on the opposite side of the bulk Si. The polyimide coating serves as an etching stopper. Following the dicing procedure, the polyimide layer is removed using oxygen plasma (g). "In-process sticking of the mirror" is the name of the dry procedure used to fabricate mirrors. AS Since the top surface of the mirror has an au coating, which gives optical beams high reflectivity, both sides of the mirror have this coating. Peak-to-valley difference can be utilized to assess the mirror surface's flatness, which in our case is 0.05 m, as the optical features of the switch made up of a MEMS mirror array depend on it.

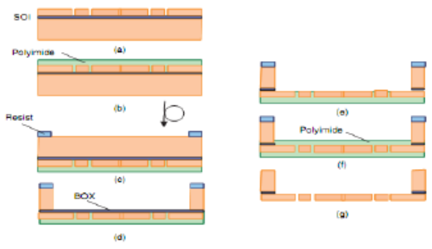


Fig 3.7 Mirror substrate fabrication process flow diagram

3.7.2 Driving of Electrode Substrate

Mirrors move due to this electrostatic force.

3.7.2.1 Fabrication of Driving Electrode Substrate

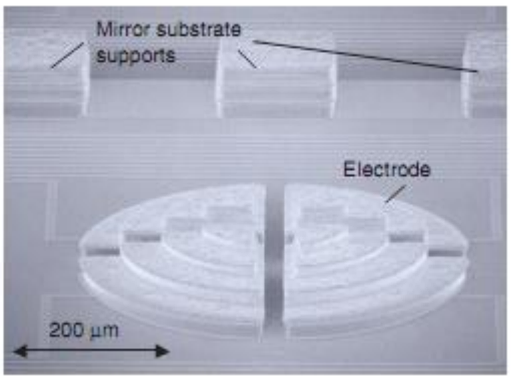


Fig 3.8 Electrodes' SEM snap

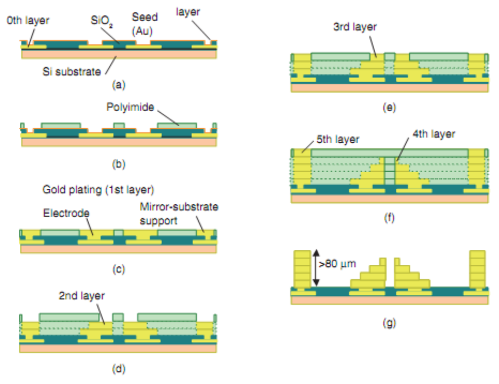


Fig 3.9 Driving Electrode Substrate's Fabrication flow process

3.8 MEMS Mirror Movement

Figure below shows connection between the voltage applied and tilt angle of mirror.

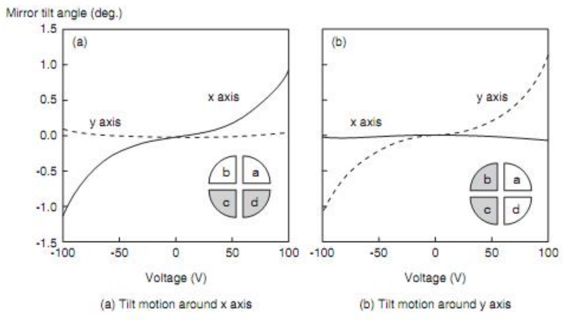


Fig 3.10 Relationship between Voltage applied & Tilt Angle of Mirror+

Size, flatness, fill factor, and scan angle of the micro mirrors, along with their scan angle and fill factor, all have an impact on how many ports the 3D MEMS switch can support.

3.9 Key Advantages of MEMS Switches in DCCN

- They offer any-to-any communication between servers, allowing for very low latency and the transfer of enormous amounts of data. which is much lower than IP switches' latency.
- Can handle any data rate.
- They can circumvent the packet-based aggregation network and offer direct, high-capacity pure optical data links between any TOR switches to reduce network latency and shift sensitive traffic between servers as needed.

IV. SOFTWARE DEFINED NETWORKING (SDN)

4.1 Introduction

Adoption of practice of creating and maintaining networks results in an architecture which is called Software Defined Networking (SDN). The major aim of this kind of architecture is to facilitate programmability for the control plane by separating the devices' that is control plane from their data/forwarding plane.

4.2 SDN Architecture

Control and Data planes at present coexist on the same network device, or, to put it another way, all data flow functions, such as switching, forwarding, and routing, as well as the various protocols used to make these decisions, are housed on that one device. "Provide open user-controlled administration of the forwarding hardware of a network element," according to the definition of SDN in [7], is the fundamental objective. The following elements are part of an architecture based on software defined networking (SDN) are shown in the below figure:

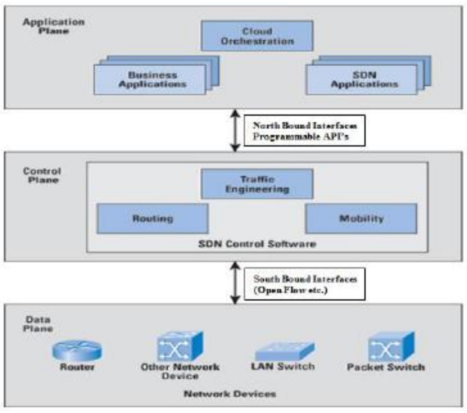


Fig 4.1: Architecture for Software Defined Networking (SDN).

4.2.1 Application Layer

Applications can include network characteristics like forwarding schemes, manageability, and security policies, among others. With the aid of the controller, the application layer can abstract the overall view of all network components and use that data to provide the control layer the proper direction. In our approach, management programs that offer CLI/GUI for the network devices are operating along with network monitoring tools.

4.2.2 Control Plane

It is network's "brain" and is in charge of managing and programming forwarding plane. The control plane employs a topology database to give an abstract picture of all the network components.

4.2.3 Data Plane

The infrastructure layer, which is the lowest layer in the SDN network architecture and includes forwarding Network Elements like (Router, Switches). Data transmission, statistics collection, and information monitoring are among its primary functions. MEMS switches are present at the aggregate layer while OF switches are located at the access.

4.2.4 API's which are North Bound

These are software interfaces b/w controller's software modules and SDN applications. The informal name used for the interface between the application/control plane is called the Northbound interface.

4.2.5 API's which are South Bound

South bound APIs are standardized APIs (application program interfaces) that allow SDN controllers to communicate with switches and routers and other forwarding network hardware [18].

4.2.6 East-West Protocols

In a multi-controller-based architecture, these protocols are used to control how different controllers communicate with one another. In a broader sense, SDN presents a networking architecture in which choices regarding the routing of data traffic are made external to the actual switching hardware. Therefore, genuine network intelligence can be achieved in data centres when SDN, PFFS, and MOOOS are applied. Second, a secure standard protocol should exist to enable communication between SDN controllers and network devices. This logical architecture may be implemented differently by different vendor equipment, but from the standpoint of an SDN controller, it performs like a consistent logical switch. AS It is explained that OpenFlow (OF) is the first and most extensively utilized interface between the infrastructure layer (data plane) and the control plane because it satisfies both of these needs.

4.3 Open Flow (OF)

The Open Networking Foundation (ONF) has standardized OpenFlow as the most popular southbound interface of SDN architecture. OF's first version, version 1.0, was created by Stanford University and then acquired by ONF. Version 1.5 of OF was released in December 2014.

It is an open standard communication protocol that defines how one or more control servers interact with switches that are SDN compliant. The flow table entries are installed in the OF compliance switches by an OF controller so that traffic is sent in line with these flow entries. These flow tables can be used by network administrators to alter network configuration and data traffic patterns. Additionally, this protocol offers management tools for packet filtering and topology change control. The OF protocol is supported by nearly all of the well-known vendors, including Cisco, HP, IBM, Brocade, and others. There are two types of switches for OF or SDN compliance.

4.3.1 Open Flow-Only Switches

These switches, which solely support OF operation and process all packets using the OF pipeline, are also known as Pure SDN switches or SDN-only switches.

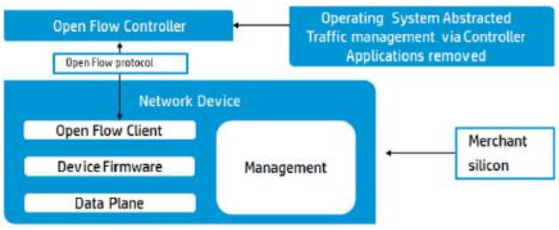


Fig 4.2 Architecture of Open Flow-Only Switch

4.3.2 Hybrid Switches with Open Flow

The same are shown in the below figure.

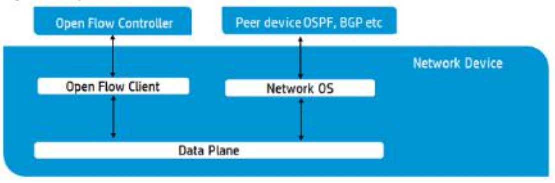


Fig 4.3 Architecture of Hybrid Switch

4.3.3 Basic Architecture of Open Flow (OF)

The below figure shows the basic OpenFlow architecture.

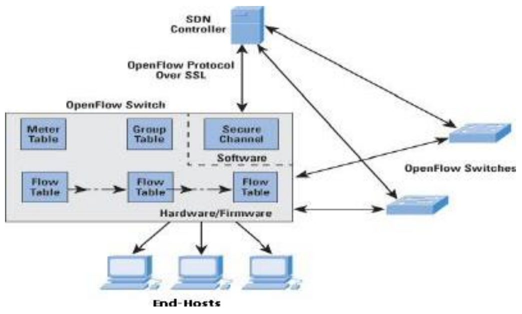


Fig 4.4 Architecture of Basic Open Flow (OF)

Open Flow switches are deployed in the Access layer as TORs, which are connected to servers or end hosts on one end and traditional switches and MEMS switches at the Aggregation layer on the other. As a control plane, the ODL controller is set up to connect with the OF switch using the OF protocol via a secure channel using SSL or TSL (Transport Layer Security). Three tables make up the OF switch's logical design. Flow Table, Group Table, Meter Table, and Table of Groups

4.3.4 Flow Table

It is the fundamental component of logical switch design that determines the action to be taken on each incoming packet by comparing it to a specific flow table made up of numerous flow entries. This packet might go via one or more flow tables that operate as pipelines. As demonstrated in fig. 4.5, a controller can add, remove, and alter flow entries from an OF switch's flow tables either proactively or reactively (in reaction to an incoming packet).

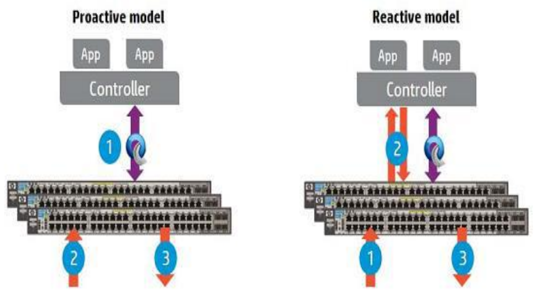


Fig 4.5 Flow Table Entries Model

Flow entries can't be generated until something happens, or after the receipt of the respective packet, thus subsequent action is taken in compliance with the instructions by a controller, while in the case of proactive model, flow entries are generated as required in advance and the process is completed without checking from the controller. When a packet first reaches a switch, the switches OF agent software looks for flow table (ASIC for hardware switches) and software flow table (virtual switches). This information exchange is shown in the figure below.

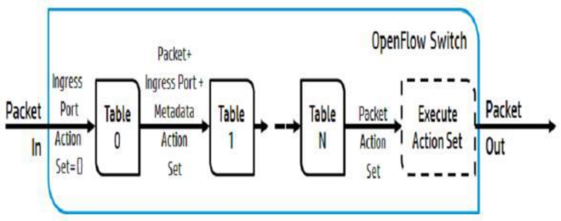


Fig 4.6 Pipeline Processing

Figure below depicts the packet processing flow chart.

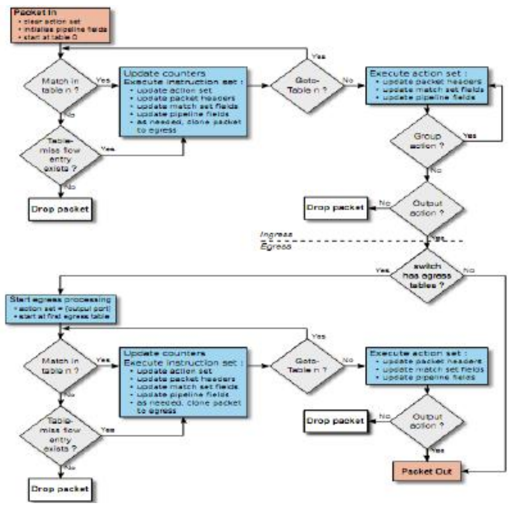


Fig 4.7 Open Flow(OF)-Packet Processing Flow Chart.

Table 4.1 Main components of Flow Entry

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

4.3.4.1 Match Felds Values

It is important here that not to pick any packets without matching field values as shown in the below table:

Table 4.2 Match Field of Flow Entry

IN_PORT	IN_PHY_PORT	METADATA	ETH_DST	ETH_SRC	ETH_TYPE	VLAN_VID
VLAN_PCP	IP_DSCP	IP_ECN	IP_PROTO	IPV4_SRC	IPV4_DST	TCP_DST
TCP_DST	UDP_SRC	UDP_DST	SCTP_SRC	SCTP_DST	ICMPV4_TYPE	ICMPV4_CODE
ARP_OP	ARP_SPA	ARP_TPA	ARP_SHA	ARP_THA	IPV6_SRC	IPV6_DST
IPV6_FLABEL	ICMPV6_TYPE	ICMPV6_CODE	IPV6_ND_TAR	IPV6_ND_SLL	IPV6_ND_TLL	MPLS_LABEL
MPLS_TC	MPLS_BOS	PBB_ISID	TUNNEL_ID	IPV6_EXTHDR		

4.3.4.2 Priority Allocation

Setting a priority along with Match fields results in a unique identity of flow entry.

4.3.4.3 Counter Checks

It contains information such as the number of bytes actually received and number of missed packets too.

4.3.4.4 Directions

Instructions are provided for side by side modifications in pipeline processing or desired set of actions.

4.3.4.5 Timeouts

It is defined as the maximum time for which any switch remains idol before which the flow of packets also expires.

4.3.4.6 Cookies

Any opaque data values selected by a relevant controller. The OF controller can utilize it to filter requests for flow deletion and modification as well as flow statistics entries. When processing packets, these values are not used.

4.3.5 Group Table

In order to perform various operations that may have an impact on several flows, Flow Table may send its flows to Group Table. A group table is made up of entries for groups that have group identifiers. The principal elements of a group entry shown below:

Table 4.3 Group Table

Group Identifier	Group Type	Counters	Action Bucket

4.3.5.1 Identifier

32-bit unique identifier identifies the group entry.

4.3.5.2 Type of Group

It is used to manage/maintain group types depicting them as “Required” and “Optional”.

4.3.5.3 Packet Counters

Provide actual packets count which are generated and processed by a group.

4.3.5.4 Action buckets

AB consists of complete flow of processes which are performed in a specific pattern for changing packets and then sending them to a port. This collection of actions is always carried out as a set of acts. A group entry may have zero or more buckets, with the exception of groups specified as "Required: Indirect," which only include one bucket. If a group doesn't have a bucket, it will immediately throw the packets into the air.

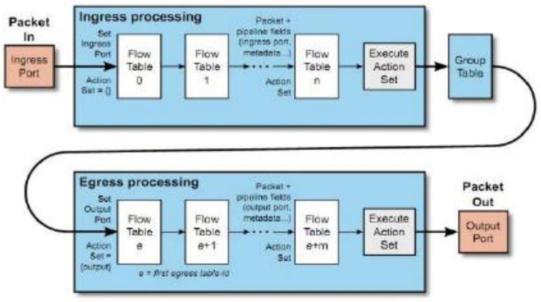


Fig.4.8 OpenFlow packets pipeline Processing

4.3.6 Meter Table

Having the ability to perform flow-related actions. It is made up of meter entries that specify per-flow meters. Personalized Service Code Point based metering also enables the division of packets into different according to data rate. Instead of using ports, meters are directly connected to flow entries. After measuring, they can influence the overall rate of all packet flows and be configured to perform a specific action. The key parts of a meter entry are presented below:

Table 4.4 Main components of Meter Entry

Meter Identifier	Meter Bands	Counters

4.3.6.1 Identifier of Meter

Meter entry is identified by a 32-bit identifier but in a distinct way.

4.3.6.2 Meter Bands

Defines how to handle packets by listing rates of each band's meter bands in a list that is not ordered. Meter Band is used for measuring the response of the meter towards packets of various meter rate ranges which are calculated from all packets received by that particular meter from all inputs. Meter bands are used to specify. Only one-meter band processes a packet.

Table 4.5 Meter Band

Band Type	Rate	Burst	Counters	Type specific arguments

4.3.6.2.1 Band Types

Packets processing ways.

4.3.6.2.2 Target Rate

It is the required rate of a meter band.

4.3.6.2.3 Burst

It establishes the metre band's granularity.

4.3.6.2.4 Counters

Processing of packets by a meter band updates the counters.

4.3.6.2.5 Type specific arguments

Parameters which are in essence optional for particular band types. Meter bands are ranked from 0 to 1, with 1 being the default band, depending on how much the desired rate increases. The packet is processed by only one-meter band when the measured rate exceeds the intended rate.

4.3.6.3 Counters

They are used to figure out how many packets a meter has processed. In the same flow table, multiple flow entries may utilize various meters, the same meters, or no meters at all.

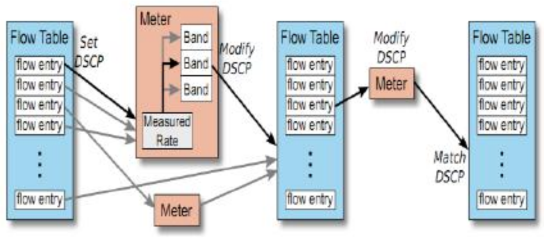


Fig. 4.9 Hierarchical DSCP metering and metres.

4.3.7 Protocols for an Open Flow Channel

Totally safe Open Flow channel is provided by the OF channel which is typically encrypted by TLS (Transport Layer Security) or SSL. The protocol that it employs for these purposes is known as the OF protocol [20].

4.3.7.1 Controller_to_Switch Messages

Messages that the controller initializes to control the switch's logical state, such as configuration, flow tables, group tables.

4.3.7.2 Asynchronous Messages

Initiated by the switch, these messages comprise status updates to inform the controller of changes to the switch's condition and network events. Additionally, it has a "Packet-In" message that switches utilize to transfer packets to OF controllers when their flow tables do not match.

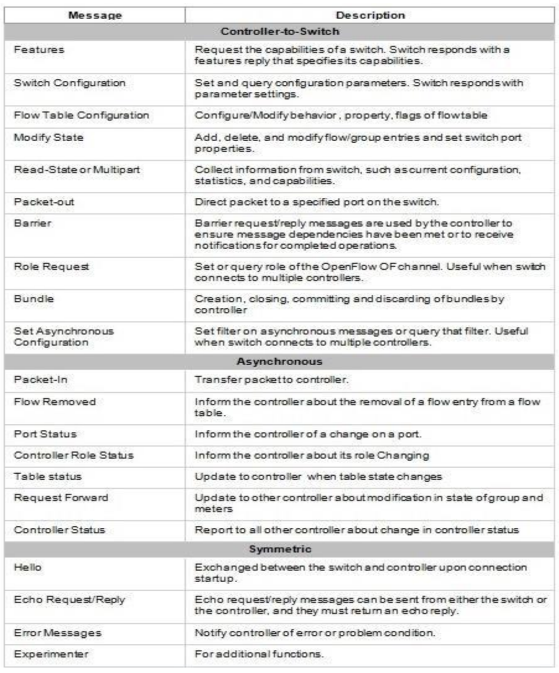
4.3.7.3 Symmetric Messages

Hello messages are initiated by controller/switch, right after when the connection is made between two specific devices, or Echo messages are used to measure the latency and bandwidth of the connection between the switch and controller as well as to confirm that the device is functioning as intended.

4.4 Benefits of SDN

SDN has the potential to be a promising technology for managing and solving different problems in data centre networks. Using the SDN technique, the network administrator only needs to declare these settings on a single place from which all devices are managed and directed by the SDN controller. Data centres are having scaling problems, particularly as the number of servers and virtual machines (VMs) that run on them rises and, later, as the requirement for migration (VM motion) rises. A significant bandwidth is needed for virtual machine migration and updating the MAC address table, which raises network latency and lowers overall network performance. In typical data centre architectures, users may encounter interruptions when accessing apps. Consequently, this researcher suggests that SDN, "All optical switching," and other virtualization technologies like OTV can be used to address this study area/gap.

Table 4.6 OpenFlow Messages



Message	Description
Controller-to-Switch	
Features	Request the capabilities of a switch. Switch responds with a features reply that specifies its capabilities.
Switch Configuration	Set and query configuration parameters. Switch responds with parameter settings.
Flow Table Configuration	Configure/Modify behavior , property, flags of flowtable
Modify State	Add, delete, and modify flow/group entries and set switch port properties.
Read-State or Multipart	Collect information from switch, such as current configuration, statistics, and capabilities.
Packet-out	Direct packet to a specified port on the switch.
Barrier	Barrier request/reply messages are used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.
Role Request	Set or query role of the OpenFlow OF channel. Useful when switch connects to multiple controllers.
Bundle	Creation, closing, committing and discarding of bundles by controller
Set Asynchronous Configuration	Set filter on asynchronous messages or query that filter. Useful when switch connects to multiple controllers.
Asynchronous	
Packet-In	Transfer packet to controller.
Flow Removed	Inform the controller about the removal of a flow entry from a flow table.
Port Status	Inform the controller of a change on a port.
Controller Role Status	Inform the controller about its role Changing
Table status	Update to controller when table state changes
Request Forward	Update to other controller about modification in state of group and meters
Controller Status	Report to all other controller about change in controller status
Symmetric	
Hello	Exchanged between the switch and controller upon connection startup.
Echo Request/Reply	Echo request/reply messages can be sent from either the switch or the controller, and they must return an echo reply.
Error Messages	Notify controller of error or problem condition.
Experimenter	For additional functions.

V. HFPFMAOS

5.1 Conceptualized HFPFMAOS Solution

As per this research contribution, suggested is the HFPFMAOS as a remedy for the problems and challenges facing DCCN (Hybrid Flow based packet filtering, forwarding & MEMS based all optical switching). Eight phases make up the implementation of this research concept:

- The deployment of an SDN controller and an OF in TOR switches that link to the controller through SSL.
- Aggregation-level MOOOS plane deployment and connection to SDN controller.
- Development of database for the conceptualized network topology.
- Flow based Packet Filtering and Forwarding – FPFSF
- Monitoring of outgoing ports, computation of links' usage, and alerting the SDN controller.
- Flow Table lookup and flow entry inspection are used to find the flows consuming more bandwidth and to inform the source and destination of those flows.
- Building high-bandwidth data connections between TOR switches that correspond to the source and destination indicated.
- Moving a particular flow of traffic to a high-bandwidth link constructed using a MOOOS plane, then moving it back to its normal path and breaking down the high-bandwidth link when that particular flow vanishes or resumes operating at its regular data rate.

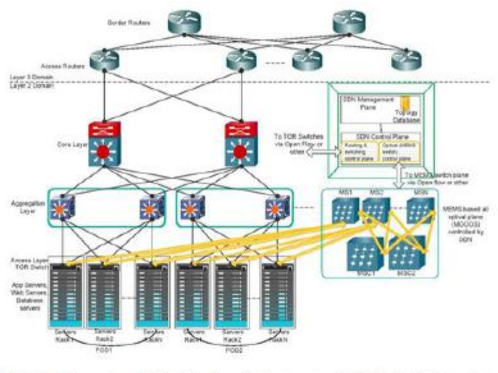
5.1.1 Installation of Open Flow in Access and SDN controller

At initial stages, the respective researcher first deploys an SDN controller and then turn on an Open Flow in all of the access layer switches (TOR), without disturbing the arrangement of the rest of the network's switches and allow them to work just as before inclusion of this setup. Then I establish a secure encrypted SSL (secure socket layer)/TCP connection between each OpenFlow OF enabled Access switch and the SDN controller. Through the OF protocol, this link is utilized to communicate between an SDN controller and an OF switch.

This connectivity is further utilized by SDN controller as well as by a number of north-bound applications, which includes initial hello /hi messages (like OFPT HELLO, FEATURE REQUEST MESSAGE from controller to switch & then FEATURE REPLY MESSAGE in the same manner from switch to controller), topology discovery using LLDP (Link Layer Discovery Protocol) & BDDP (Broadcast Domain Discovery Protocol), thus start building data bases, message alerts (PACKET IN to controller & PACKET OUT) messages from table to switch which specifically flows over all optical path between OF switches.

5.1.2 Setting up Plane

At the aggregation level, researcher deployed MAOS (optical switching based on MEMS) plane alongside conventional switches. The current packet-based switching continues to function in this approach together with the newly proposed MAOS to provide dynamically high speed data routes between servers for switching elephant traffic as needed. Plus, centralized control is exercised by connecting it to a central SDN controller, network elements(NEs) supervision, and control of traffic flow. Figure below shows the conceptualized architecture.

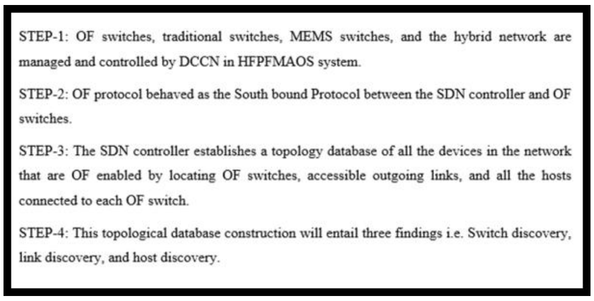

Border Routers
Access Routers
Layer 3 Domain
Layer 2 Domain
Core Layer
SDN Management Plane
Topology Database
SDN Control Plane
To TOR Switches via Open Flow or other
To MEMS switch plane via Open flow or other
Aggregation Layer
MEMS based all optical plane (MOOOS) controlled by SDN
Access Layer TOR Switch
App Servers, Web Servers, Database servers
POD1
POD2

OF switches (TOR) are paired with aggregation and MEMS switches. Orange lines shown represent an idle high bandwidth TORS connections between MEMS switches in MAOS Plane to manage prolonged persistent data flows. Black lines relate to active links of a standard packet switched network.

Fig 5.1 Conceptualized Architecture of HPMOOOS for DCCN

At the aggregation layer, the MAOS plane is implemented side by side the conventional switches.

5.1.3 Developing Topology Database



STEP-1: OF switches, traditional switches, MEMS switches, and the hybrid network are managed and controlled by DCCN in HFPFMAOS system.

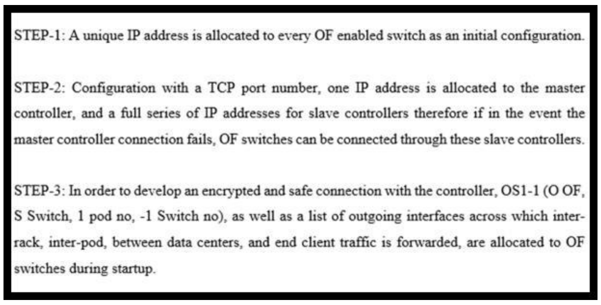
STEP-2: OF protocol behaved as the South bound Protocol between the SDN controller and OF switches.

STEP-3: The SDN controller establishes a topology database of all the devices in the network that are OF enabled by locating OF switches, accessible outgoing links, and all the hosts connected to each OF switch.

STEP-4: This topological database construction will entail three findings i.e. Switch discovery, link discovery, and host discovery.

Fig 5.2 Database Topology Development Steps

5.1.3.1 Revelation of OpenFlow OF Switches



STEP-1: A unique IP address is allocated to every OF enabled switch as an initial configuration.

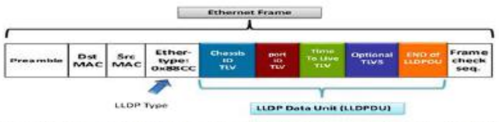
STEP-2: Configuration with a TCP port number, one IP address is allocated to the master controller, and a full series of IP addresses for slave controllers therefore if in the event the master controller connection fails, OF switches can be connected through these slave controllers.

STEP-3: In order to develop an encrypted and safe connection with the controller, OS1-1 (O OF, S Switch, 1 pod no, -1 Switch no), as well as a list of outgoing interfaces across which inter-rack, inter-pod, between data centers, and end client traffic is forwarded, are allocated to OF switches during startup.

Fig 5.3 OpenFlow OF Switches Process

5.1.3.2 Revelation of Active Links

OFDP can detect indirect accessible multi-hop links (routes) between OpenFlow OF switches by leveraging the L2 discovery protocol known as BDDP [22]. Most well-known Open Source controllers, including ODL and Floodlight, adopt this technique.



In destination MAC address field, BDDP utilizes broadcast address (ff:ff:ff:ff:ff:ff) rather than the multicast address (01:80:c2:00:00:0e, 01:80:c2:00:00:03, or 01:80:c2:00:00:00) used in LLDP, and it uses the field value of 0x8999 rather than 0x88cc in the case of LL Figure shows frame structures of the LLDP and BDDP.

Fig 5.4 Structural Framework of LLDP

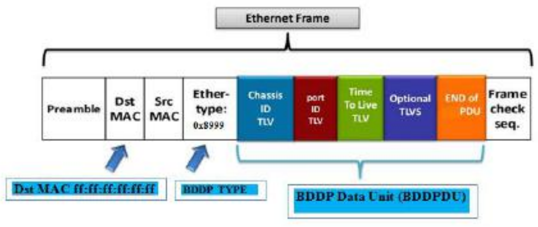


Fig 5.5 BDDP Frame Structure

A switch's identification, basic capabilities, and other characteristics are determined by an OF controller via the OF protocol, which includes sending "OFPT FEATURES REQUEST" messages to all switches as part of the first handshake.

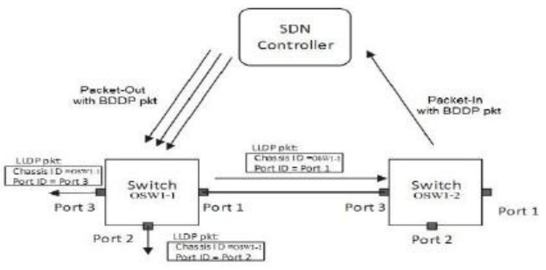

SDN Controller
Packet-Out with BDDP pkt
Packet-In with BDDP pkt
LLDP pkt: Chassis ID =oswi-1 Port ID = Port 3
LLDP pkt: Chassis ID =OSW1-1 Port ID = Port 1
LLDP pkt: Chassis ID =osw1-1 Port ID = Port 2
Switch OSW1-1
Switch OSW1-2
Port 3
Port 1
Port 2
Port 3
Port 1
Port 2

Fig 5.6 Links discovery Via OFDP

Messages exchanged between OpenFlow OF switches and controller for Links are shown below.

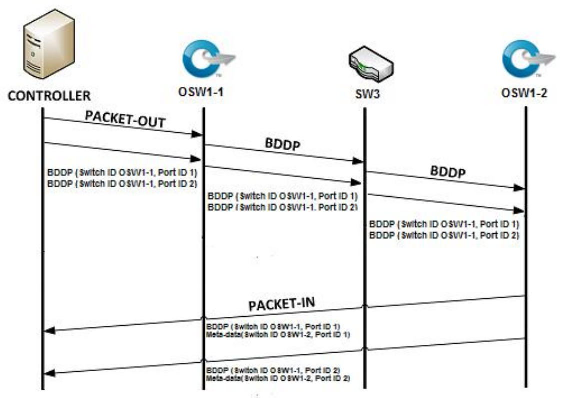

CONTROLLER
OSW1-1
SW3
OSW1-2
PACKET-OUT
BDDP
BDDP
BDDP (Switch ID OSW1-1, Port ID 1)
BDDP (Switch ID OSW1-1, Port ID 2)
BDDP (Switch ID OSW1-1, Port ID 1)
BDDP (Switch ID OSW1-1, Port ID 2)
BDDP (Switch ID OSW1-1, Port ID 1)
BDDP (Switch ID OSW1-1, Port ID 2)
PACKET-IN
BDDP (Switch ID OSW1-1, Port ID 1)
Meta-data(Switch ID OSW1-2, Port ID 1)
BDDP (Switch ID OSW1-1, Port ID 2)
Meta-data(Switch ID OSW1-2, Port ID 2)

Fig 5.7 Messages flow between controller and Open Flow OF switches for Links

The metadata of OpenFlow OF switches where BDDP packets are received are included in "OFPT PACKET IN" messages. Between OpenFlow OF switches and their controller, messages are only transmitted in one direction. These massages are also exchanged in the other direction. Based on data from BDDP and metadata, the controller can identify indirect multi-hop links between OpenFlow OF switches after receiving these "OFPT PACKET IN" signals and store them in its database to create a network topology. Most crucially, the controller counts the number of hops between OpenFlow OF switches using the TTL value. The default time interval for this topology finding operation is 5 seconds. The total number of "OFPT PACKET IN" messages received by the controller during this entire discovery process is equal to double the number of "L" active links accessible in the domain, and this number may be calculated as below:

$$\text{Total}_{\text{Rx OFPT\_PACKET\_IN}} = 2 \text{ x } L$$

However, the total messages “OFPT_PACKET_OUT” sent by controller can be computed as:

$$\text{Total}_{\text{Tx OFPT\_PACKET\_OUT}} = \sum_{i=0}^{S} Pi$$

Number of Open Flow OF switches is denoted by S, whereas active ports are denoted by P. All switches has P. 4 OpenFlow OF switches with 2 active ports each make up our reference topology, hence "OFPT PACKET OUT" messages issued by the controller are 2+2+2+2=8 in total. The total number of BDDP OFPT PACKET OUT messages sent by a controller to a switch can be reduced by implementing OFDPv2[23], in which the Port ID TLV field is set to 0 and will be ignored while the source MAC address field has been set with the MAC address of the port through which it is to be sent out. One "OFPT PACKET OUT" message is sent by the controller for each switch, and the total number of messages transmitted is equal to the number of switches determined by OFDPv2.

$$\text{Total}_{\text{Tx OFPT\_PACKET\_OUT}} = S$$

As each OpenFlow OF switch's port is physically connected to a MEMS switch's port, we don't need link discovery to establish connectivity with the MAOS plane; instead, the controller can create paths between any two points dynamically, and topology database can statically store all the data related to flows and port connections.

Table 5.1 Topology Database

Switch ID	Port	Direct connected Port MAC address	Far End OSW ID & Port Rx "Packet_IN" MSG	Type	Status
OSW1-1	Eth 1	SW3 port0 MAC address	OSW1-2, Eth 1 ,1	Dynamic	Up
	Eth 1	SW3 port0 MAC address	OSW2-1, Eth 1 ,3	Dynamic	Up
	Eth 1	SW3 port0 MAC address	OSW2-2, Eth 1 ,3	Dynamic	Up
	Eth 1	SW3 port0 MAC address	OSW1-2, Eth 1 ,3	Dynamic	Up
	Eth 2	SW4 port0 MAC address	OSW1-2, Eth 2 ,1	Dynamic	Up
	Eth 2	SW4 port0 MAC address	OSW2-1, Eth 2 ,3	Dynamic	Up
	Eth 2	SW4 port0 MAC address	OSW2-2, Eth 2 ,3	Dynamic	Up
	Eth 2	SW4 port0 MAC address	OSW1-2, Eth 2 ,3	Dynamic	Up
	Eth 3	MSW1 port 1		Static	Down
	Eth 4	Controller		Static	Up
	Eth 5				Down
	Eth 6				Down
	Eth 7	bb:bb:bb:bb:bb:bb		Dynamic	Up
	Eth 8	aa:aa:aa:aa:aa:aa		Dynamic	Up
OSW1-2	Eth 1	SW3 port1 MAC address		Dynamic	Up
	Eth 2	SW4 port1 MAC address		Dynamic	Up
	Eth 3	MSW1 port 1		Static	Down
	Eth 4	Controller		Static	Up
	Eth 5				Down
	Eth 6				Down
	Eth 7	cc:cc:cc:cc:cc:cc		Dynamic	Up
	Eth 8	dd:dd:dd:dd:dd:dd		Dynamic	Up

5.1.3.3 Discovery of Host

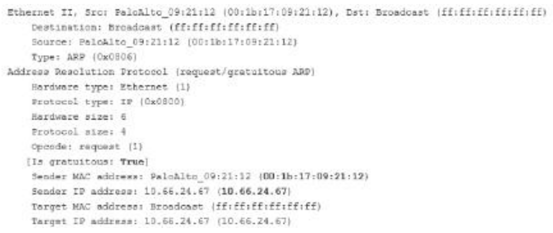
Two methods have been applied for available host discovery which is connected to the OpenFlow OF switches.

As all active ports on all switches send BDDP "Packet Out" messages, I Ports for which OpenFlow OF Switches do not send "Packet In" messages are recognized as host ports during link discovery. (ii) GARP message is sent by the Host when Host is connected to an OpenFlow OF Switch. HSRP and VRRP[24] use GARP to update the MAC address table of L2 switches, In a broadcast domain GARP is an advance notification mechanism to keep the controller aware about host discovery and inserting flow entries of MAC address in the flow tables of Open Flow OF switches, thus updating other hosts’ ARP tables before their ARP requests are made, and finally updating ARP tables of other hosts when a new host is also connected to a switch,however a host IP address or MAC address is changed due to failover. Generated ARP requests are actually special ARP packets with the source that is (host) IP and destination.The destination broadcast MAC address (ff:ff:ff:ff:ff:ff) is present in the MAC address field and the Ethertype field fixt to 0x0806. The parameters listed below are part of a GARP request message:

• FF:FF:FF:FF:FF:FF:FF:FF is the destination MAC address (broadcast)

• Source MAC address: MAC address of the host

Example of GARP is shown in the below figure. IP address of a Source = IP address of a Destination: Host transmitting GARP Type ,IP address is: ARP (0x0806).



```
Ethernet II, Src: PaloAlto_09:21:12 (00:1b:17:09:21:12), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
    Destination: Broadcast (ff:ff:ff:ff:ff:ff)
    Source: PaloAlto_09:21:12 (00:1b:17:09:21:12)
    Type: ARP (0x0806)
Address Resolution Protocol (request/gratuitous ARP)
    Hardware type: Ethernet (1)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
   [Is gratuitous: True]
    Sender MAC address: PaloAlto_09:21:12 (00:1b:17:09:21:12)
    Sender IP address: 10.66.24.67 (10.66.24.67)
    Target MAC address: Broadcast (ff:ff:ff:ff:ff:ff)
    Target IP address: 10.66.24.67 (10.66.24.67)
```

Fig 5.8 GARP Request Message

Each host will ping its neighbor after GARP is finished to verify connectivity and reachability. Currently, all hosts are aware of one another's MAC addresses and assigned IP addresses, which are visible in their ARP tables. To stop BDDP packet propagation, SDN controllers disable link discovery on the OpenFlow OF switch ports to which hosts are connected after discovery. Similar to BPDU guard security function in conventional switches, this suppression also prevents BPDU propagation [27].

Table 5.2 Host-1 linked with OSW1-1 - ARP

IP Address	Physical Address
10.0.0.2 (Host 2 IP Address)	bb:bb:bb:bb:bb:bb (Host 2 MAC Address)

Table 5.3 Host 2 linked with OSW1-1 – ARP

IP Address	Physical Address
10.0.0.1 (Host 1 IP Address)	aa:aa:aa:aa:aa:aa (Host 1 MAC Address)

5.1.4 Discovered Routes & Forwarding Table Buildup

Information relating to interface of OpenFlow(OF) switchs is received by Controller, plus list of all routes as well as destinations (MAC/IP addresses) also become visible through it, because it contains map of topology discovery of the entire network and a centralised database too. Each route found is given a Route-Tag by the SDN controller following topology discovery. Allocating a Route-Tag has the main aim of to distinctly recognize each respectively occured route and then generate a forwarding table that gives information about all of the destination (MAC/IP address) addresses that may be reached using the same route. In order to send a flow to an outgoing interface based on Route-Tag,this forwarding table is installed by an SDN Controller into the OF switch. The source address and destination address are used to traditionally forward flow frame to next hop because its content addressable memory table also includes a list of MAC addresses that can be reached over each link (port).later on when the same flow touches an OpenFlow OF switch, it searches again all the flow tables and look for the best possible match before forwarding the flow to the accurate Host port after assuring its destination address.

5.1.5 Building up of Flow Table by inserting Flow entries

After entering into an OpenFlow OF switch the flow itself again compares each incoming packet with one or more flow tables, each of which contains multiple flow entries, and then determines the action to be taken. Any sort of matching can be used, for example matching an ingress port, a source or destination MAC address or IP address, a VLAN ID, a TCP/UDP port number, etc.

For providing maximum flexibility and support to diverse types of data traffic passing through DCCN while supplying delay-sensitive traffic with minimal latency, I will employ a hybrid model of flow table entries in this thesis that combines proactive and reactive modalities. The use of a proactive model for flow entry is required for delay-sensitive applications that demand low latency, for example while making audio/video calls, live radio/TV transmissions, financial banks transactions, and routine heavy traffic like web browsing, data file/folder transfer, and peer-to-peer traffic.

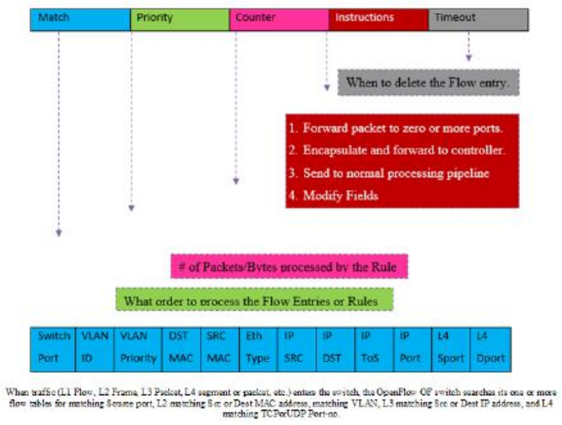

Match
Priority
Counter
Instructions
Timeout
When to delete the Flow entry.
1. Forward packet to zero or more ports.
2. Encapsulate and forward to controller.
3. Send to normal processing pipeline
4. Modify Fields
# of Packets/Bytes processed by the Rule
What order to process the Flow Entries or Rules
Switch Port
VLAN ID
VLAN Priority
DST MAC
SRC MAC
Eth Type
IP SRC
IP DST
IP ToS
IP Port
L4 Sport
L4 Dport
When traffic (L1 Flow, L2 Frame, L3 Packet, L4 segment or packet, etc.) enters the switch, the OpenFlow OF switch searches its one or more flow tables for matching Source port, L2 matching Src or Dest MAC address, matching VLAN, L3 matching Src or Dest IP address, and L4 matching TCP or UDP Port-no.

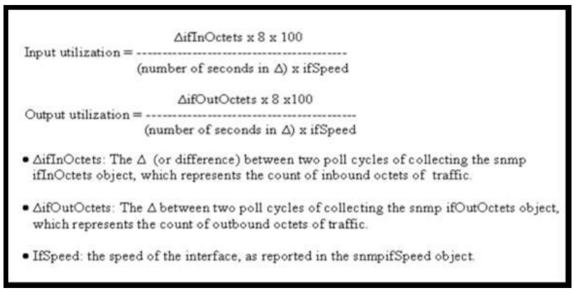
Fig 5.9 Anatomy - OpenFlow OF

Discovery of flow entries which have close match with the related fields of the data traffic, it either executes the particular action of flow entry or forwards it to a group table to execute multiple actions. The importance of the priority parameter cannot be overstated since flow entries are prioritised, and if there are many flow entries, the one with the highest priority will be used and the others will be ignored. Table below shows different types of counters.

Table 5.4 Types of Counters

Per Table	Per Flow	Per Port	Per Queue
Active Entries	Received Packets	Received Packets	Transmit Packets
Packet Lookups	Receive Bytes	Transmitted Packets	Transmit Bytes
Packet Matches	Duration (Secs)	Receive Bytes	Transmit Overrun Errors
	Duration (nsecs)	Transmitted Bytes	
		Receive Drops	
		Transmitted Drops	
		Received Errors	
		Transmitted Errors	
		Receive Frame allignment error	
		Receive Overrun Errors	
		Receive CRC Errors	
		Collisions	

By issuing the command OFPMP PORT STATS to the OpenFlow OF switch, the SDN controller continuously receives the statistics of outgoing active links (Ports), such is the quantity of Rx packets, Tx packets, Rx bytes, and Tx bytes, as well as the time in seconds, dropped packets, and Tx/Rx errors. Link utilisation of all the desired ports is calculated by the controller based on the statistics that were returned. The Link Utilization is calculated using the following formula[28]:



$$\text{Input utilization} = \frac{\Delta\text{ifInOctets} \times 8 \times 100}{(\text{number of seconds in } \Delta) \times \text{ifSpeed}}$$

$$\text{Output utilization} = \frac{\Delta\text{ifOutOctets} \times 8 \times 100}{(\text{number of seconds in } \Delta) \times \text{ifSpeed}}$$

- ΔifInOctets: The Δ (or difference) between two poll cycles of collecting the snmp ifInOctets object, which represents the count of inbound octets of traffic.
- ΔifOutOctets: The Δ between two poll cycles of collecting the snmp ifOutOctets object, which represents the count of outbound octets of traffic.
- IfSpeed: the speed of the interface, as reported in the snmpifSpeed object.

Fig 5.10 Formula to calculate Link Utilization

For each of the outgoing interfaces, the controller maintains a specific link utilisation threshold. When the threshold is exceeded, it consults its flowtable to filter flows with outgoing interfaces that have exceeded the threshold. Following the specification of ports, the SDN controller determines whether any ports are available before creating a Highband dat pathpath across the MEMS Plane.

If there is no match, the entry in the table is considered to be incorrect, and the controller is notified by default. The controller instructs the switch to take certain actions. The "Flow Mod" message contains a variety of information, including Buffer IDs, Timeouts, Actions, Priorities, and more. Additionally, flow entries can be permanent or for a limited period of time, and there are two types: "Hard Timeout" and "Idle Timeout." Idle timeout refers to the removal of an entry from the flow table if there is no matching flow entry request during that time period. Hard timeout refers to the maximum amount of time an entry can remain in the flow table, regardless of whether a matching entry is present. If Hard timeout is set to 0, it is deactivated.

As an illustration, in our reference topology diagram, Host 1 sends an HTTP request to Host 2 (let's say a web server) following host discovery by GARP. It begins with a SYN message. Host 1 sends a SYN message to the OSW1-1 switch, which checks its flow table upon receiving the packet because it is the initial packet and likely has no flow entries that match the packet. Table miss flow entry is the term for this. Therefore, by encasing this TCP packet in a "Packet IN" message, the switch passes it to the controller by default. This Packet IN message contains the entire TCP packet or its Buffer ID (for example, Buffer ID = 250, which designates the location where the switch stores the whole TCP message). Therefore, the controller will take a few actions, such as returning a "Flow Mod" or "Packet Out" message to the switch, where "Packet Out" includes information regarding the switch's handling of that particular as well as the whole encapsulated TCP packet or the reference buffer ID that the switch uses to store this packet. Send the TCP SYNpacket reference with buffer ID=250 out of port 8 to host 2 if the switch OSW1-1 receives a "Packet Out" message.Then the switch is directed to add another new flow entry to its flow table through the "Flow Mod" message. When a similar packet arrives at the switch in the future and matches its fields and masks, this flow entry guides the switch what to do now.In this way the message informs the related switch to route any TCP requests from Host1's IP address or MAC address to Host4's IP address or MAC address and finally to Port 8. Additionally, it tells the switch to release the packet it had been buffering with the BufferID of 250 and to carry out the instructions in this message. H4 replies by sending a SYN/ACK packet to the switch, which receives it but finds no flow entry from Host2 to Host1 (yet another table miss). In order to send this SYN/ACK packet to the controller for additional analysis, the switch encapsulates it in a "Packet In" message and gives it the reference buffer ID BufferID=251. In response, the controller instructs the switch to add a flow entry to the flow table and perform some action, which is to forward the SYN/ACK message to port7. The controller also sends the switch a Packet Out and a Flow mod message.The remainder of the communication between Host1 and Host2 would not reach the controller after all of this because switches have flow entries in their flow tables that tell them what to do with packets. The switch routed HTTP reply and ACK messages directly, as demonstrated in the following figs:

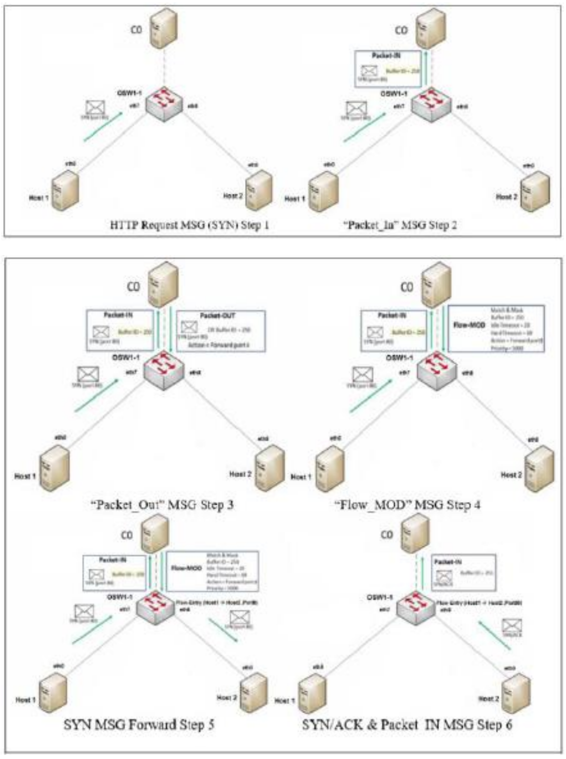


Fig 5.11 (a) HTTP request with Open Flow messaging

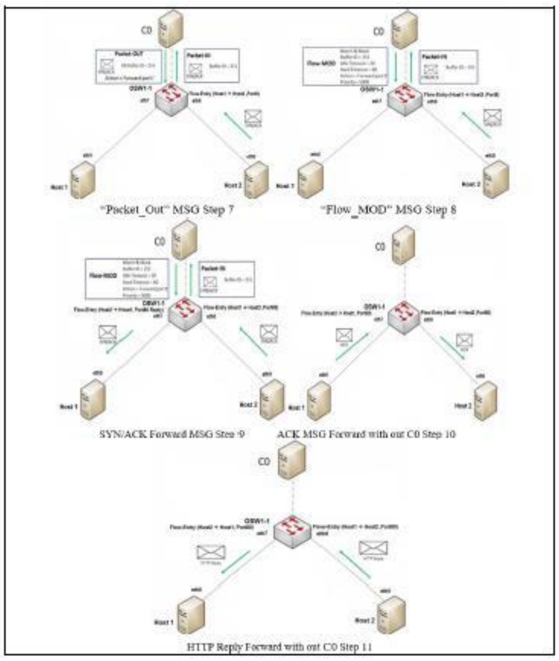

CO
OSW1-1
Host 1
Host 2
"Packet_Out" MSG Step 7
"Flow_MOD" MSG Step 8
SYN/ACK Forward MSG Step 9
ACK MSG Forward with out C0 Step 10
HTTP Reply Forward with out C0 Step 11

Fig 5.11 (b) HTTP Reply-Open Flow messages

Following table is showin OF switch flow entries:

Table 5.5 Flow Table of OF Switch OSW1-1 Flow Entries

I/P Port	Src MAC	Dst MAC	VLAN ID	VLAN PCP	Ether_Type	SRC IP	DST IP	IP Prot	IP ToS	L4 SPort	L4 DPort	Priority	Action	Counter
contr	*	ff:ff	*	*	0x8999	*	*	*	*	*	*	10	Local	11
*	*	*	*	*	0x0806	*	*	*	*	*	*	10	Local	2
*	*	aa:aa	*	*	*	*	*	*	*	*	*	80	Port 7	50
*	*	bb:bb	*	*	*	*	*	*	*	*	*	80	Port 8	60
*	*	dd:dd	*	*	*	*	*	*	*	*	*	80	Port 1,2	60
*	*	*	10	*	*	*DSTrouting	10.0.0.9/8	*	*	*	*	67	Port 1	20
*	*	*	20	*	*	*DSTrouting	10.0.0.16/8	*	*	*	*	67	Port 2	15
*	*	*	*	*	*	*	*default route	*	*	*	*	50	Contr, Port1	100
Port7	aa:aa	bb:bb	20	0	0x0800	10.0.0.1	10.0.0.2	0x06	*	38661	80	67	Port 8 TCP	20
Port8	bb:bb	aa:aa	20	0	0x0800	10.0.0.2	10.0.0.1	0x06	*	80	38661	67	Port 7 TCP	20

The very first entry instructs switch to broadcast packets that arrive from the controller port, regardless of their ether type, to all other switch ports, indicating that they are BDDP messages, and to update the counter. Sec entry instructs the switch to issue a GARP request and update the counter regardless of the switch's fields if the ether type is 0x0806. The third, fourth, and fifth lines are L2 matching and forwarding, which instructs the switch to perform a specific action specified by the Action field and update the counter regardless of other fields if the packet arrives with a certain DMAC. The sixth and seventh lines instruct IP to route packets to a particular port based on their destination IP. The eighth line is the default route, as shown for our reference topology, and the final two flow entries are TCP flow entries for HTTP requests and answers. When the bulk of brief flows, sometimes referred to as "Mouse flows," start flowing alongside long persistent high bandwidth data flows, or "elephant flows," This results in a bandwidth bottleneck. High bandwidth data flows, which can be caused by, among other things, VM migration, database backups, or other software like Hadoop, cause switches' buffers to overflow, which slows down packet processing and negatively affects applications that require low latency. Nowadays, there are many virtual machines running on a single server for various applications. For whatever reason, one virtual machine running one application in Server Rack 1 started using a lot of bandwidth and CPU, which had an adverse effect on other applications operating on the same server. Since the switch buffers start to overflow, the administrator is compelled to move the virtual machine to Server Rack 2 of the same POD, which results in a bandwidth bottleneck and increased latency on network links, as shown in the figure below.

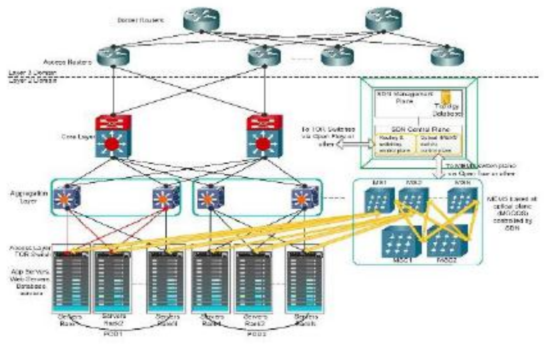


Fig 5.12 Congestion

Colored red indicate congestion on an active packet-switched network link that a packet is attempting to cross. As seen in the figure, switches alert their management/controlled plane to network congestion and increased delay.

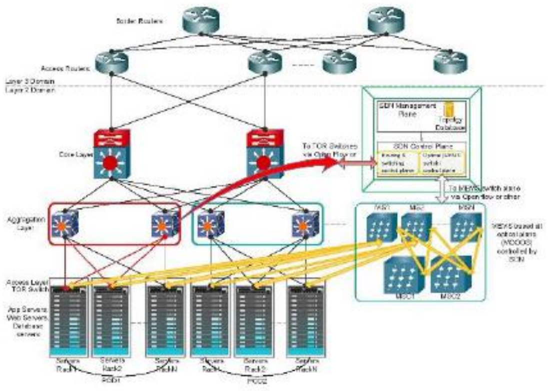
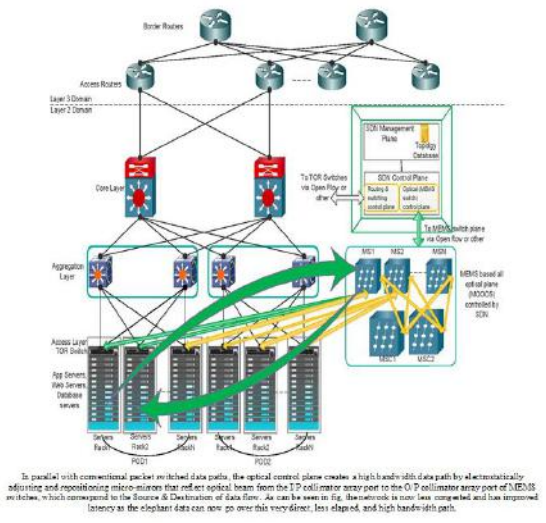


Fig 5.13 Congestion Notification by Switches to Control Plane

In reply, the management/control plane investigates the packets to make clear their source and destination, consults its topology database, and calculates the locations between which high bandwidth data routes must be developed.

Resultantly, the optical control plane gives messages to the MAOS control plane, which is made up of MEMS-based optical switches.


Border Routers
Access Routers
Layer 3 Domain
Layer 2 Domain
Core Layer
SDN Management Plane
Topology Database
To TOR Switches via Open Flow or other
SDN Control Plane
Routing & switching control plane
Optical MEMS switch controllers
To MEMS switch plane via Open flow or other
Aggregation Layer
MS1
MS2
MSN
MEMS based all optical plane (MOOCS) controlled by SDN
Access Layer TOR Switch
App Servers, Web Servers, Database servers
MSC1
MSC2
Servers Rack1
Servers Rack2
Servers RackN
POD1
POD2

In parallel with conventional packet switched data paths, the optical control plane creates a high bandwidth data path by electrostatically adjusting and repositioning micro-mirrors that reflect optical beam from the I/P collimator array port to the O/P collimator array port of MEMS switches, which correspond to the Source & Destination of data flow. As can be seen in fig, the network is now less congested and has improved latency as the elephant data can now go over this very direct, less elapsed, and high bandwidth path.

Fig 5.14 MEMS switches produce High data rate optical path

As seen above, data is travelling across both packet-switched network and a temporary, high-bandwidth channel, which is depicted by the green colour and is enabled by MEMS-based switches. Once the high persistent flow has subsided and traffic flow has returned to normal, the control plane will demolish the temporarily established optical link and reallocate it as needed.

VI. SOFTWARE IMPLEMENTATION

6.1 Reference Network Topology

In order to implement the proposed software of the proposes solution, the researcher came up with the below conceptualized architecture.

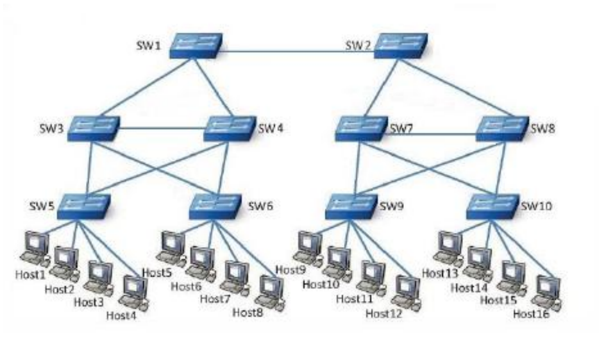
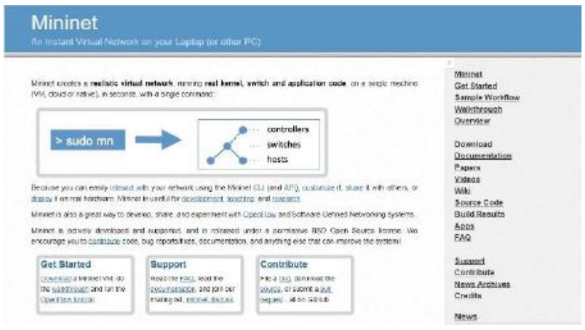


Fig 6.1 Prototype testing Reference network topology

Our network topology consists of 2 pods, POD 1 and POD 2, with each pod expected to include eight hosts or servers, two aggregation switches, and two access switches. POD1 is made up of two aggregation switches (SW3 & SW4), two access (TOR) switches (SW5 & SW6), and eight hosts (Hosts 1 through Host8) that produce traffic. POD2 consists of 8 hosts, Host1 to Host8, 2 TOR switches, named SW9 and SW10, 2 aggregation switches, named as SW7 and SW8, and 2 hosts.

6.2 Software used

The tools incorporated are shown in in the figure below:


Mininet

An Instant Virtual Network on your Laptop (or other PC)

Mininet creates a **realistic virtual network**, running **real kernel, switch and application code**, on a single machine (VM, cloud or native), in seconds, with a single command:

> sudo mn

controllers
switches
hosts

Because you can easily interact with your network using the Mininet CLI (and API), customize it, share it with others, or deploy it on real hardware, Mininet is useful for development, teaching, and research.

Mininet is also a great way to develop, share, and experiment with OpenFlow and Software Defined Networking systems.

Mininet is actively developed and supported, and is released under a permissive BSD Open Source license. We encourage you to contribute code, bug reports/fixes, documentation, and anything else that can improve the system!

Get Started
Download a Mininet VM, do the walkthrough and run the OpenFlow tutorial

Support
Read the FAQ, read the documentation, and join our mailing list, mininet-discuss

Contribute
File a bug, download the source, or submit a pull request - all on GitHub

Mininet
Get Started
Sample Workflow
Walkthrough
Overview

Download
Documentation
Papers
Videos
Wiki
Source Code
Build Results
Apps
FAQ

Support
Contribute
News Archives
Credits

News

Fig 6.2 Software Tool Used Mininet

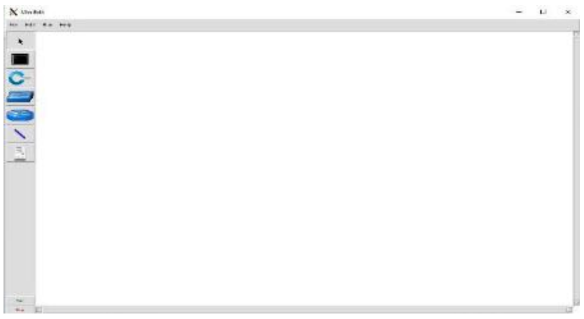


Fig 6.3 Software Tool Used MiniEdit

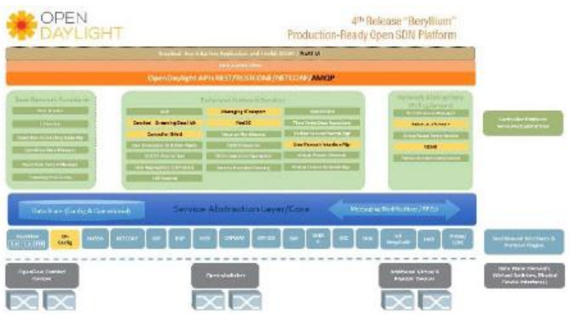

OPEN DAYLIGHT
4th Release "Beryllium"
Production-Ready Open SDN Platform
OpenDaylight APIs REST/RESTCONF/NETCONF/AMQP
Service Abstraction Layer/Core

Fig 6.4 Software Tool Used Open Day Light

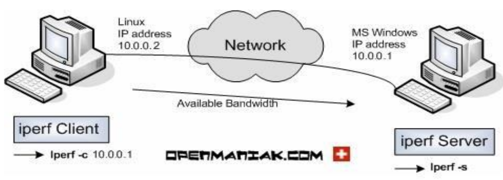


Fig 6.5 Software Tool Used IPERF

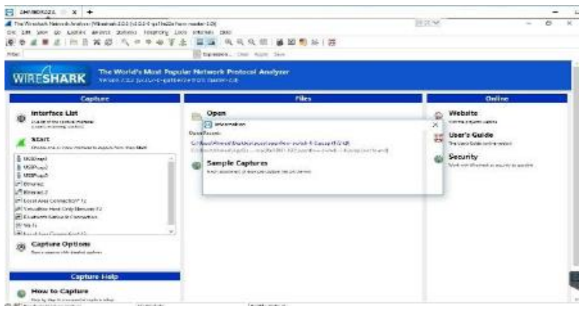
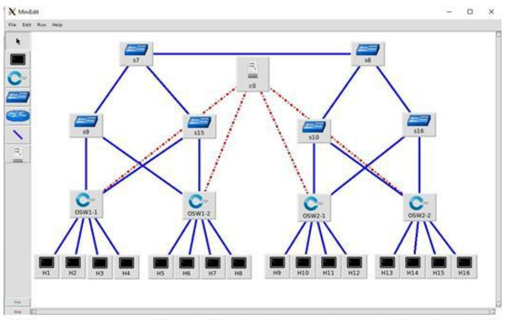


Fig 6.6 Software Tool Used WireShark



Host H1 through H16 are servers that are mounted in corresponding racks and connected to each other via OF switches and traditional switches, while C0 is the SDN controller with an IP address of 192.168.56.102 that is connected to each OF switch via a malicious OF port number 6633 over TCP. H1, H2, H3, and H4 are regarded as servers that are mounted in a rack and connected to a TOR switch (OSW1-1); H5, H6, H7, and H8 are connected to an OSW1-2; H9, H10, H11, and H12 are connected to an OSW2-1; and H13, H14, H15, and H16 are connected to an OSW2-1; (OSW2-2). The IP addresses for each host between H1 and H16 range from 10.0.0.1 to 10.0.0.16, while the IP addresses for each OF switch are OSW1-1=10.0.1.1/16, OSW1-2=10.0.1.2/16, OSW2-1=10.0.2.1/16, and OSW2-2=10.0.2.2/16.

Fig 6.8 OF switches based Network topology - Access as TOR switches

6.3 Preparation/Implementation

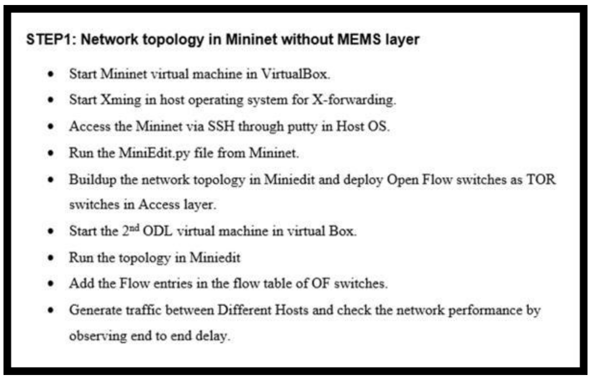
Below actions must be taken to prepare for this setup:

- Before running my Mininet and Open Day Light virtual machines, I first downloaded and installed Oracle VM VirtualBox.
- Next, construct a VM in VirtualBox, download the Mininet virtual machine image, mount it on top of the fresh virtual machine, and then install Mininet.
- Third, create a second VirtualBox virtual machine and install the ODL controller setup.

Mininet and its GUI program MiniEdit (running on o1 VM on VirtualBox) is used by researcher, to plan out and simulate the reference topology. I require X forwarding in order to run Miniedit and connect to Mininet over SSH. To do this, I've used Putty and XMing. Open DayLight, an external SDN-based OpenFlow controller, was utilised by me to operate OpenFlow virtual switches and MEMS switches (ODL Beryllium). I have generated data flow from hosts and servers and measured several performance characteristics using the IPERF, such as link bandwidth and network latency. I used Wireshark to investigate data packets and analyse variety of protocol messages on several interfaces during the whole network topology.

6.4 Applications

The real deployment of HFPFMAOS was gained in two main steps



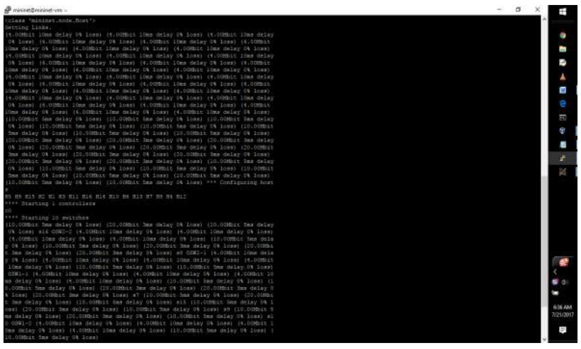
STEP1: Network topology in Mininet without MEMS layer

- Start Mininet virtual machine in VirtualBox.
- Start Xming in host operating system for X-forwarding.
- Access the Mininet via SSH through putty in Host OS.
- Run the MiniEdit.py file from Mininet.
- Buildup the network topology in Miniedit and deploy Open Flow switches as TOR switches in Access layer.
- Start the 2nd ODL virtual machine in virtual Box.
- Run the topology in Miniedit
- Add the Flow entries in the flow table of OF switches.
- Generate traffic between Different Hosts and check the network performance by observing end to end delay.

Fig 6.7 Deployment Steps of HFPFMAOS

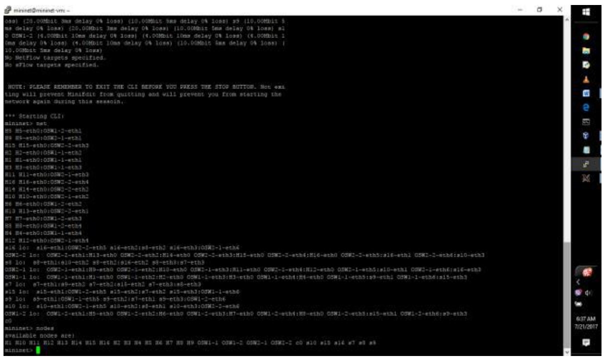
Links between hosts and OF switches are 4 Mb/s with a 10 msec delay, while links between all traditional switches and between OF switches are 10 Mb/s with a 5 msec delay, as demonstrated in the following figures:

Host, Nodes, Links and interfaces verification can be done by these commands, mininet>nodes, mininet>net and mininet>dump.



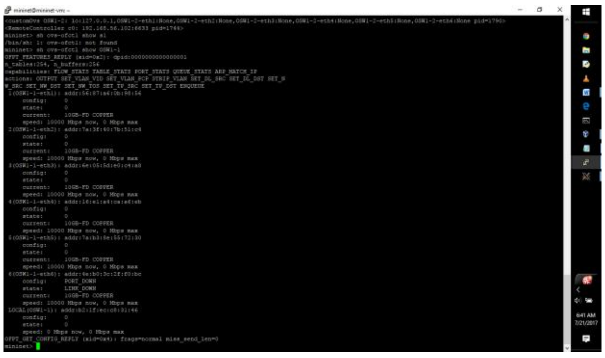
```
<class 'mininet.node.Host'>
Getting Links.
(4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay
 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit
10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss)
(4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay
 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit
10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss)
(4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay
 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit
10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss)
(4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay
 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit
10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss)
(10.00Mbit 5ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) (10.00Mbit 5ms delay
 0% loss) (10.00Mbit 5ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) (10.00Mbit
5ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) (10.00Mbit 5ms delay 0% loss)
(20.00Mbit 3ms delay 0% loss) (20.00Mbit 3ms delay 0% loss) (20.00Mbit 3ms delay
 0% loss) (20.00Mbit 3ms delay 0% loss) (20.00Mbit 3ms delay 0% loss) (20.00Mbit
3ms delay 0% loss) (20.00Mbit 3ms delay 0% loss) (20.00Mbit 3ms delay 0% loss)
(20.00Mbit 3ms delay 0% loss) (20.00Mbit 3ms delay 0% loss) (10.00Mbit 5ms delay
 0% loss) (10.00Mbit 5ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) (10.00Mbit
5ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) (10.00Mbit 5ms delay 0% loss)
(10.00Mbit 5ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) *** Configuring host
s
H5 H9 H15 H2 H1 H3 H11 H16 H14 H10 H6 H13 H7 H8 H4 H12
*** Starting 1 controllers
c0
*** Starting 10 switches
(10.00Mbit 5ms delay 0% loss) (20.00Mbit 3ms delay 0% loss) (20.00Mbit 5ms delay
 0% loss) s16 OSW2-2 (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss)
 (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (10.00Mbit 5ms dela
y 0% loss) (10.00Mbit 5ms delay 0% loss) (20.00Mbit 3ms delay 0% loss) (20.00Mbi
t 3ms delay 0% loss) (20.00Mbit 3ms delay 0% loss) s8 OSW2-1 (4.00Mbit 10ms dela
y 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit
 10ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) (10.00Mbit 5ms delay 0% loss)
 OSW1-1 (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10
ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) (1
0.00Mbit 5ms delay 0% loss) (20.00Mbit 3ms delay 0% loss) (20.00Mbit 3ms delay 0
% loss) (20.00Mbit 3ms delay 0% loss) s7 (10.00Mbit 5ms delay 0% loss) (20.00Mbi
t 3ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) s15 (10.00Mbit 5ms delay 0% l
oss) (20.00Mbit 3ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) s9 (10.00Mbit 5
ms delay 0% loss) (20.00Mbit 3ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) s1
0 OSW1-2 (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 1
0ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) (
10.00Mbit 5ms delay 0% loss)
```

Fig 6.9 Creating Links Switches



```
oss) (20.00Mbit 3ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) s9 (10.00Mbit 5
ms delay 0% loss) (20.00Mbit 3ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) s1
0 OSW1-2 (4.00Mbit 10ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (4.00Mbit 1
0ms delay 0% loss) (4.00Mbit 10ms delay 0% loss) (10.00Mbit 5ms delay 0% loss) (
10.00Mbit 5ms delay 0% loss)
No NetFlow targets specified.
No sFlow targets specified.


 NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exi
ting will prevent MiniEdit from quitting and will prevent you from starting the
network again during this session.

*** Starting CLI:
mininet> net
H5 H5-eth0:OSW1-2-eth1
H9 H9-eth0:OSW2-1-eth1
H15 H15-eth0:OSW2-2-eth3
H2 H2-eth0:OSW1-1-eth2
H1 H1-eth0:OSW1-1-eth1
H3 H3-eth0:OSW1-1-eth3
H11 H11-eth0:OSW2-1-eth3
H16 H16-eth0:OSW2-2-eth4
H14 H14-eth0:OSW2-2-eth2
H10 H10-eth0:OSW2-1-eth2
H6 H6-eth0:OSW1-2-eth2
H13 H13-eth0:OSW2-2-eth1
H7 H7-eth0:OSW1-2-eth3
H8 H8-eth0:OSW1-2-eth4
H4 H4-eth0:OSW1-1-eth4
H12 H12-eth0:OSW2-1-eth4
s16 lo:  s16-eth1:OSW2-2-eth5 s16-eth2:s8-eth2 s16-eth3:OSW2-1-eth6
OSW2-2 lo:  OSW2-2-eth1:H13-eth0 OSW2-2-eth2:H14-eth0 OSW2-2-eth3:H15-eth0 OSW2-2-eth4:H16-eth0 OSW2-2-eth5:s16-eth1 OSW2-2-eth6:s10-eth3
s8 lo:  s8-eth1:s10-eth2 s8-eth2:s16-eth2 s8-eth3:s7-eth3
OSW2-1 lo:  OSW2-1-eth1:H9-eth0 OSW2-1-eth2:H10-eth0 OSW2-1-eth3:H11-eth0 OSW2-1-eth4:H12-eth0 OSW2-1-eth5:s10-eth1 OSW2-1-eth6:s16-eth3
OSW1-1 lo:  OSW1-1-eth1:H1-eth0 OSW1-1-eth2:H2-eth0 OSW1-1-eth3:H3-eth0 OSW1-1-eth4:H4-eth0 OSW1-1-eth5:s9-eth1 OSW1-1-eth6:s15-eth3
s7 lo:  s7-eth1:s9-eth2 s7-eth2:s15-eth2 s7-eth3:s8-eth3
s15 lo:  s15-eth1:OSW1-2-eth5 s15-eth2:s7-eth2 s15-eth3:OSW1-1-eth6
s9 lo:  s9-eth1:OSW1-1-eth5 s9-eth2:s7-eth1 s9-eth3:OSW1-2-eth6
s10 lo:  s10-eth1:OSW2-1-eth5 s10-eth2:s8-eth1 s10-eth3:OSW2-2-eth6
OSW1-2 lo:  OSW1-2-eth1:H5-eth0 OSW1-2-eth2:H6-eth0 OSW1-2-eth3:H7-eth0 OSW1-2-eth4:H8-eth0 OSW1-2-eth5:s15-eth1 OSW1-2-eth6:s9-eth3
c0
mininet> nodes
available nodes are:
H1 H10 H11 H12 H13 H14 H15 H16 H2 H3 H4 H5 H6 H7 H8 H9 OSW1-1 OSW1-2 OSW2-1 OSW2-2 c0 s10 s15 s16 s7 s8 s9
mininet>
```

Fig 6.10 Network/Nodes Verification



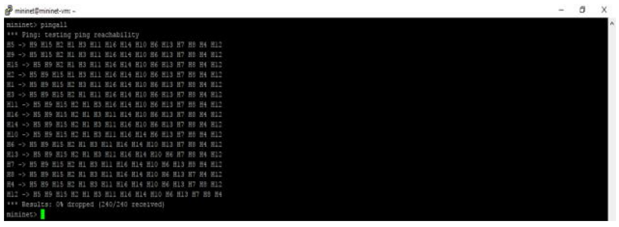
```
<customOvs OSW1-2: lo:127.0.0.1,OSW1-2-eth1:None,OSW1-2-eth2:None,OSW1-2-eth3:None,OSW1-2-eth4:None,OSW1-2-eth5:None,OSW1-2-eth6:None pid=1790>
<RemoteController c0: 192.168.56.102:6633 pid=1744>
mininet> sh ovs-ofctl show s1
/bin/sh: 1: ovs-ofctl: not found
mininet> sh ovs-ofctl show OSW1-1
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_N
W_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
 1(OSW1-1-eth1): addr:56:87:a6:0b:98:56
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(OSW1-1-eth2): addr:7a:3f:40:7b:51:c4
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(OSW1-1-eth3): addr:6e:05:5d:e0:c4:a8
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 4(OSW1-1-eth4): addr:16:e1:a4:ca:a6:cb
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 5(OSW1-1-eth5): addr:7a:b3:8e:55:72:30
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 6(OSW1-1-eth6): addr:4e:b0:3c:2f:f0:bc
     config:     PORT_DOWN
     state:      LINK_DOWN
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(OSW1-1): addr:b2:1f:ec:c8:31:46
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
mininet>
```

Fig 6.11 OF Switch interfaces verification

Flow entries are shown by the researcher in the flow table of OF switches OSW1-1, OSW1-2, OSW1-3 & OSW1-4 for my proposed solution HFPFMAOS which are as under:

For Discovery of HOST and ARP Table

mininet> ping all



```
mininet> pingall
*** Ping: testing ping reachability
H5 -> H9 H15 H2 H1 H3 H11 H16 H14 H10 H6 H13 H7 H8 H4 H12
H9 -> H5 H15 H2 H1 H3 H11 H16 H14 H10 H6 H13 H7 H8 H4 H12
H15 -> H5 H9 H2 H1 H3 H11 H16 H14 H10 H6 H13 H7 H8 H4 H12
H2 -> H5 H9 H15 H1 H3 H11 H16 H14 H10 H6 H13 H7 H8 H4 H12
H1 -> H5 H9 H15 H2 H3 H11 H16 H14 H10 H6 H13 H7 H8 H4 H12
H3 -> H5 H9 H15 H2 H1 H11 H16 H14 H10 H6 H13 H7 H8 H4 H12
H11 -> H5 H9 H15 H2 H1 H3 H16 H14 H10 H6 H13 H7 H8 H4 H12
H16 -> H5 H9 H15 H2 H1 H3 H11 H14 H10 H6 H13 H7 H8 H4 H12
H14 -> H5 H9 H15 H2 H1 H3 H11 H16 H10 H6 H13 H7 H8 H4 H12
H10 -> H5 H9 H15 H2 H1 H3 H11 H16 H14 H6 H13 H7 H8 H4 H12
H6 -> H5 H9 H15 H2 H1 H3 H11 H16 H14 H10 H13 H7 H8 H4 H12
H13 -> H5 H9 H15 H2 H1 H3 H11 H16 H14 H10 H6 H7 H8 H4 H12
H7 -> H5 H9 H15 H2 H1 H3 H11 H16 H14 H10 H6 H13 H8 H4 H12
H8 -> H5 H9 H15 H2 H1 H3 H11 H16 H14 H10 H6 H13 H7 H4 H12
H4 -> H5 H9 H15 H2 H1 H3 H11 H16 H14 H10 H6 H13 H7 H8 H12
H12 -> H5 H9 H15 H2 H1 H3 H11 H16 H14 H10 H6 H13 H7 H8 H4
*** Results: 0% dropped (240/240 received)
mininet>
```

Fig 6.12 Host Discovery by Pingall command for

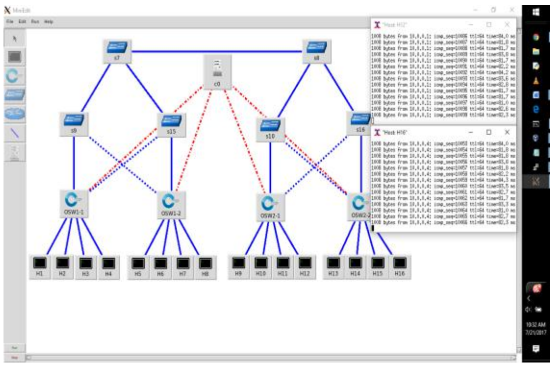
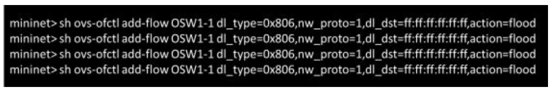


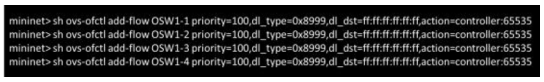
Fig 6.13 Host pinging

For Links Discovery GARP Requests:



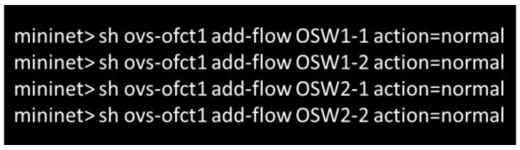
```
mininet> sh ovs-ofctl add-flow OSW1-1 dl_type=0x806,nw_proto=1,dl_dst=ff:ff:ff:ff:ff:ff,action=flood
mininet> sh ovs-ofctl add-flow OSW1-1 dl_type=0x806,nw_proto=1,dl_dst=ff:ff:ff:ff:ff:ff,action=flood
mininet> sh ovs-ofctl add-flow OSW1-1 dl_type=0x806,nw_proto=1,dl_dst=ff:ff:ff:ff:ff:ff,action=flood
mininet> sh ovs-ofctl add-flow OSW1-1 dl_type=0x806,nw_proto=1,dl_dst=ff:ff:ff:ff:ff:ff,action=flood
```

For Links Discovery BDDP Requests:



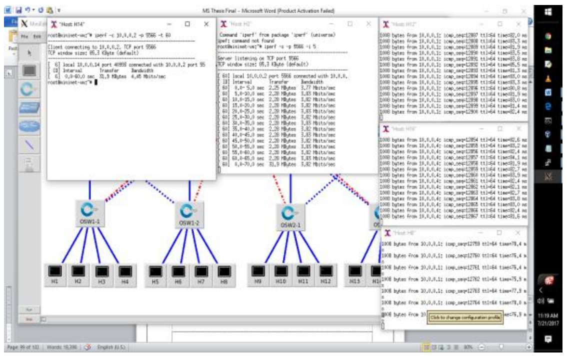
```
mininet> sh ovs-ofctl add-flow OSW1-1 priority=100,dl_type=0x8999,dl_dst=ff:ff:ff:ff:ff:ff,action=controller:65535
mininet> sh ovs-ofctl add-flow OSW1-2 priority=100,dl_type=0x8999,dl_dst=ff:ff:ff:ff:ff:ff,action=controller:65535
mininet> sh ovs-ofctl add-flow OSW1-3 priority=100,dl_type=0x8999,dl_dst=ff:ff:ff:ff:ff:ff,action=controller:65535
mininet> sh ovs-ofctl add-flow OSW1-4 priority=100,dl_type=0x8999,dl_dst=ff:ff:ff:ff:ff:ff,action=controller:65535
```

For Enable Layer 2 Forwarding



```
mininet> sh ovs-ofct1 add-flow OSW1-1 action=normal
mininet> sh ovs-ofct1 add-flow OSW1-2 action=normal
mininet> sh ovs-ofct1 add-flow OSW2-1 action=normal
mininet> sh ovs-ofct1 add-flow OSW2-2 action=normal
```

Generation of Traffic by Iperf



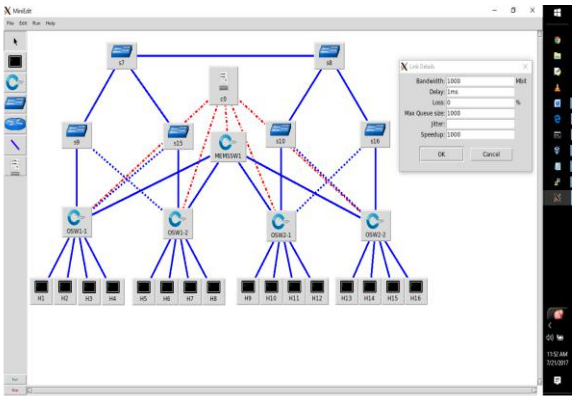
```
root@mininet-vm:~# iperf -c 10.0.0.2 -p 5566 -t 60
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5566
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  6] local 10.0.0.14 port 40958 connected with 10.0.0.2 port 55
[ ID] Interval       Transfer     Bandwidth
[  6]  0.0-60.0 sec  31.5 MBytes  4.45 Mbits/sec
root@mininet-vm:~#
```

```
Command 'iperf' from package 'iperf' (universe)
iperf: command not found
root@mininet-vm:~# iperf -s -p 5566 -i 5
------------------------------------------------------------
Server listening on TCP port 5566
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 60] local 10.0.0.2 port 5566 connected with 10.0.0.
[ ID] Interval       Transfer     Bandwidth
[ 60]  0.0- 5.0 sec  2.25 MBytes  3.77 Mbits/sec
[ 60]  5.0-10.0 sec  2.28 MBytes  3.83 Mbits/sec
[ 60] 10.0-15.0 sec  2.28 MBytes  3.82 Mbits/sec
[ 60] 15.0-20.0 sec  2.28 MBytes  3.82 Mbits/sec
[ 60] 20.0-25.0 sec  2.28 MBytes  3.83 Mbits/sec
[ 60] 25.0-30.0 sec  2.28 MBytes  3.82 Mbits/sec
[ 60] 30.0-35.0 sec  2.28 MBytes  3.83 Mbits/sec
[ 60] 35.0-40.0 sec  2.28 MBytes  3.82 Mbits/sec
[ 60] 40.0-45.0 sec  2.28 MBytes  3.83 Mbits/sec
[ 60] 45.0-50.0 sec  2.28 MBytes  3.82 Mbits/sec
[ 60] 50.0-55.0 sec  2.28 MBytes  3.83 Mbits/sec
[ 60] 55.0-60.0 sec  2.28 MBytes  3.82 Mbits/sec
[ 60] 60.0-65.0 sec  2.28 MBytes  3.83 Mbits/sec
[ 60]  0.0-70.0 sec  31.9 MBytes  3.82 Mbits/sec
```

Fig 6.14 Generation of Traffic and Latency (Delay)

STEP2: Mininet's Network topology with MEMS

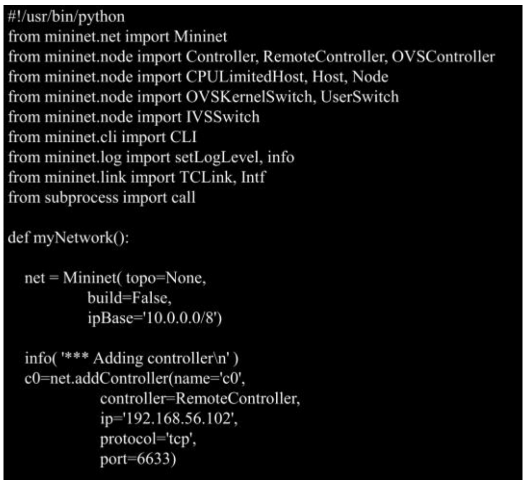
In the second phase, I add another Open flow switch is added which is called MEMSSW1 to the aggregation layer beside conventional switches and connect it to OF switches to test/apply the concept of MEMS switching (TOR). I intentionally introduced the links' bandwidth and latency with MEMSSW1 and keep it on a higher side, that is 1000Mb/s and 1msec, to testify the idea of MEMS. I set the buffer size and throughput (speedup) of the links to be equal to the link's bandwidth, or 1000 Mb/s, to demonstrate all optical switching. This is how the network topology is displayed:



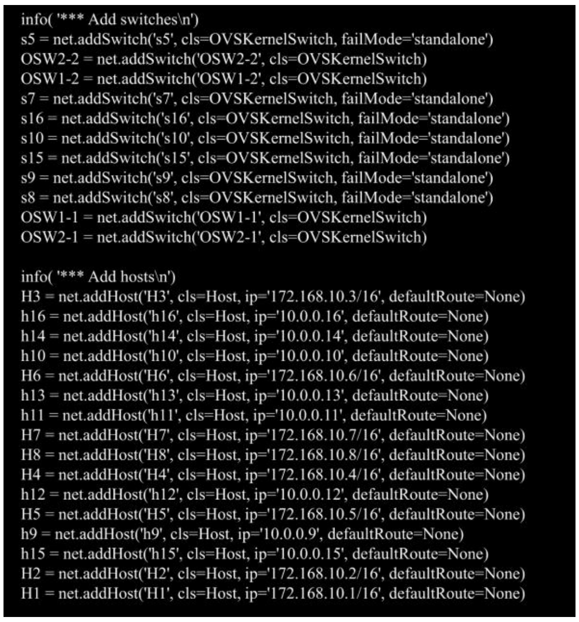
```
Bandwidth: 1000 Mbit
Delay: 1ms
Loss: 0 %
Max Queue size: 1000
Jitter:
Speedup: 1000
```

Fig 6.15 Network Topology with MEMS

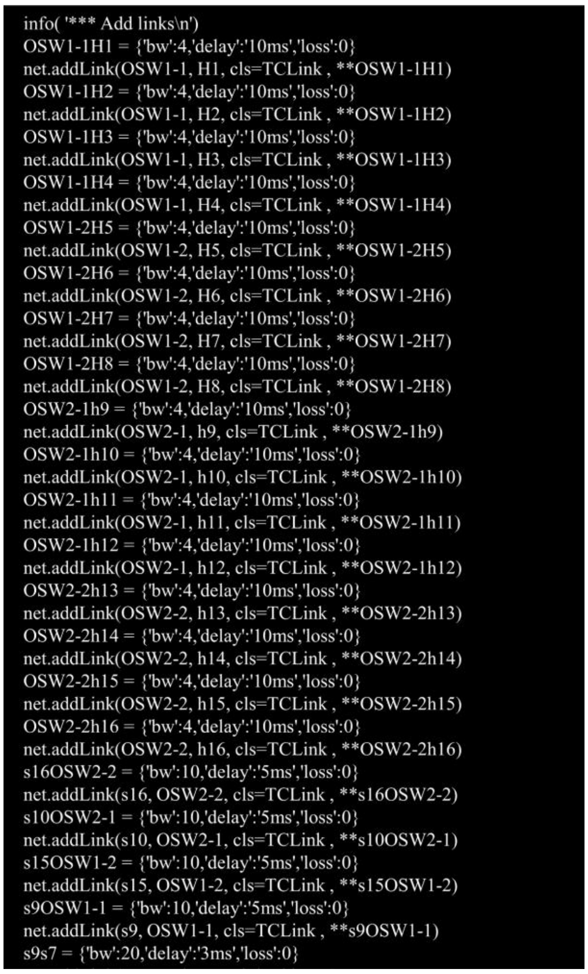
6.5 Code for Topology Creation:



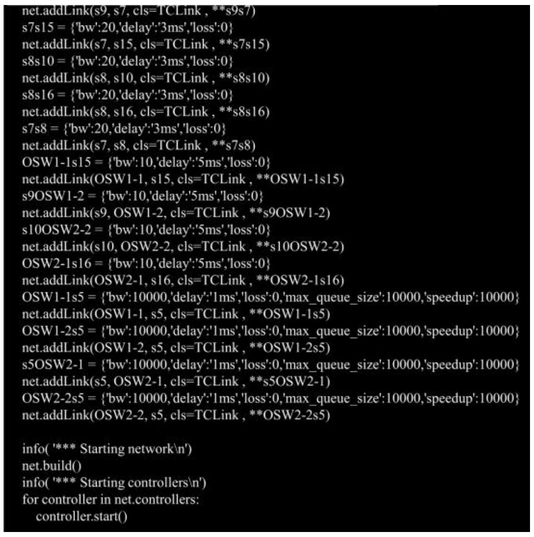
```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                      controller=RemoteController,
                      ip='192.168.56.102',
                      protocol='tcp',
                      port=6633)
```



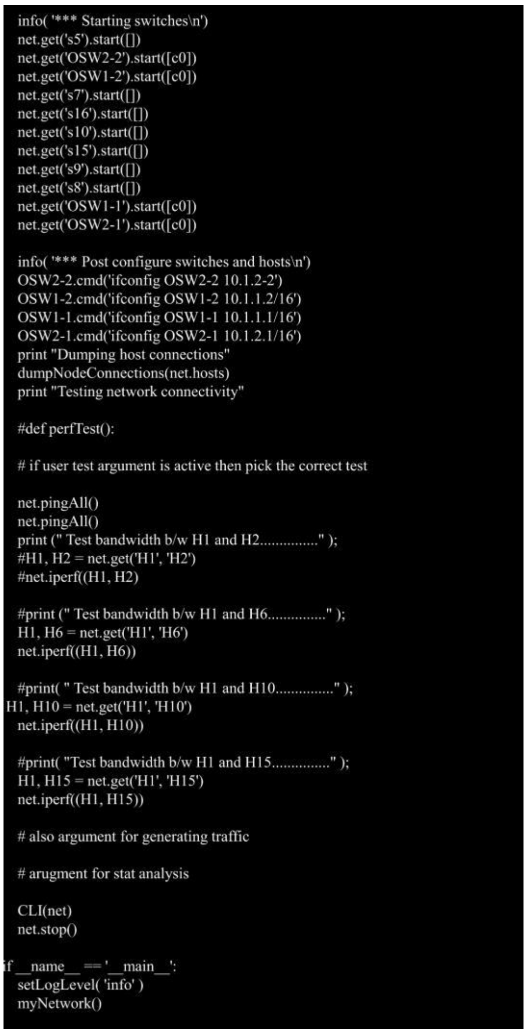
```
info( '*** Add switches\n')
s5 = net.addSwitch('s5', cls=OVSKernelSwitch, failMode='standalone')
OSW2-2 = net.addSwitch('OSW2-2', cls=OVSKernelSwitch)
OSW1-2 = net.addSwitch('OSW1-2', cls=OVSKernelSwitch)
s7 = net.addSwitch('s7', cls=OVSKernelSwitch, failMode='standalone')
s16 = net.addSwitch('s16', cls=OVSKernelSwitch, failMode='standalone')
s10 = net.addSwitch('s10', cls=OVSKernelSwitch, failMode='standalone')
s15 = net.addSwitch('s15', cls=OVSKernelSwitch, failMode='standalone')
s9 = net.addSwitch('s9', cls=OVSKernelSwitch, failMode='standalone')
s8 = net.addSwitch('s8', cls=OVSKernelSwitch, failMode='standalone')
OSW1-1 = net.addSwitch('OSW1-1', cls=OVSKernelSwitch)
OSW2-1 = net.addSwitch('OSW2-1', cls=OVSKernelSwitch)

info( '*** Add hosts\n')
H3 = net.addHost('H3', cls=Host, ip='172.168.10.3/16', defaultRoute=None)
h16 = net.addHost('h16', cls=Host, ip='10.0.0.16', defaultRoute=None)
h14 = net.addHost('h14', cls=Host, ip='10.0.0.14', defaultRoute=None)
h10 = net.addHost('h10', cls=Host, ip='10.0.0.10', defaultRoute=None)
H6 = net.addHost('H6', cls=Host, ip='172.168.10.6/16', defaultRoute=None)
h13 = net.addHost('h13', cls=Host, ip='10.0.0.13', defaultRoute=None)
h11 = net.addHost('h11', cls=Host, ip='10.0.0.11', defaultRoute=None)
H7 = net.addHost('H7', cls=Host, ip='172.168.10.7/16', defaultRoute=None)
H8 = net.addHost('H8', cls=Host, ip='172.168.10.8/16', defaultRoute=None)
H4 = net.addHost('H4', cls=Host, ip='172.168.10.4/16', defaultRoute=None)
h12 = net.addHost('h12', cls=Host, ip='10.0.0.12', defaultRoute=None)
H5 = net.addHost('H5', cls=Host, ip='172.168.10.5/16', defaultRoute=None)
h9 = net.addHost('h9', cls=Host, ip='10.0.0.9', defaultRoute=None)
h15 = net.addHost('h15', cls=Host, ip='10.0.0.15', defaultRoute=None)
H2 = net.addHost('H2', cls=Host, ip='172.168.10.2/16', defaultRoute=None)
H1 = net.addHost('H1', cls=Host, ip='172.168.10.1/16', defaultRoute=None)
```



```
info( '*** Add links\n')
OSW1-1H1 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW1-1, H1, cls=TCLink , **OSW1-1H1)
OSW1-1H2 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW1-1, H2, cls=TCLink , **OSW1-1H2)
OSW1-1H3 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW1-1, H3, cls=TCLink , **OSW1-1H3)
OSW1-1H4 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW1-1, H4, cls=TCLink , **OSW1-1H4)
OSW1-2H5 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW1-2, H5, cls=TCLink , **OSW1-2H5)
OSW1-2H6 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW1-2, H6, cls=TCLink , **OSW1-2H6)
OSW1-2H7 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW1-2, H7, cls=TCLink , **OSW1-2H7)
OSW1-2H8 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW1-2, H8, cls=TCLink , **OSW1-2H8)
OSW2-1h9 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW2-1, h9, cls=TCLink , **OSW2-1h9)
OSW2-1h10 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW2-1, h10, cls=TCLink , **OSW2-1h10)
OSW2-1h11 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW2-1, h11, cls=TCLink , **OSW2-1h11)
OSW2-1h12 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW2-1, h12, cls=TCLink , **OSW2-1h12)
OSW2-2h13 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW2-2, h13, cls=TCLink , **OSW2-2h13)
OSW2-2h14 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW2-2, h14, cls=TCLink , **OSW2-2h14)
OSW2-2h15 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW2-2, h15, cls=TCLink , **OSW2-2h15)
OSW2-2h16 = {'bw':4,'delay':'10ms','loss':0}
net.addLink(OSW2-2, h16, cls=TCLink , **OSW2-2h16)
s16OSW2-2 = {'bw':10,'delay':'5ms','loss':0}
net.addLink(s16, OSW2-2, cls=TCLink , **s16OSW2-2)
s10OSW2-1 = {'bw':10,'delay':'5ms','loss':0}
net.addLink(s10, OSW2-1, cls=TCLink , **s10OSW2-1)
s15OSW1-2 = {'bw':10,'delay':'5ms','loss':0}
net.addLink(s15, OSW1-2, cls=TCLink , **s15OSW1-2)
s9OSW1-1 = {'bw':10,'delay':'5ms','loss':0}
net.addLink(s9, OSW1-1, cls=TCLink , **s9OSW1-1)
s9s7 = {'bw':20,'delay':'3ms','loss':0}
```



```
net.addLink(s9, s7, cls=TCLink , **s9s7)
s7s15 = {'bw':20,'delay':'3ms','loss':0}
net.addLink(s7, s15, cls=TCLink , **s7s15)
s8s10 = {'bw':20,'delay':'3ms','loss':0}
net.addLink(s8, s10, cls=TCLink , **s8s10)
s8s16 = {'bw':20,'delay':'3ms','loss':0}
net.addLink(s8, s16, cls=TCLink , **s8s16)
s7s8 = {'bw':20,'delay':'3ms','loss':0}
net.addLink(s7, s8, cls=TCLink , **s7s8)
OSW1-1s15 = {'bw':10,'delay':'5ms','loss':0}
net.addLink(OSW1-1, s15, cls=TCLink , **OSW1-1s15)
s9OSW1-2 = {'bw':10,'delay':'5ms','loss':0}
net.addLink(s9, OSW1-2, cls=TCLink , **s9OSW1-2)
s10OSW2-2 = {'bw':10,'delay':'5ms','loss':0}
net.addLink(s10, OSW2-2, cls=TCLink , **s10OSW2-2)
OSW2-1s16 = {'bw':10,'delay':'5ms','loss':0}
net.addLink(OSW2-1, s16, cls=TCLink , **OSW2-1s16)
OSW1-1s5 = {'bw':10000,'delay':'1ms','loss':0,'max_queue_size':10000,'speedup':10000}
net.addLink(OSW1-1, s5, cls=TCLink , **OSW1-1s5)
OSW1-2s5 = {'bw':10000,'delay':'1ms','loss':0,'max_queue_size':10000,'speedup':10000}
net.addLink(OSW1-2, s5, cls=TCLink , **OSW1-2s5)
s5OSW2-1 = {'bw':10000,'delay':'1ms','loss':0,'max_queue_size':10000,'speedup':10000}
net.addLink(s5, OSW2-1, cls=TCLink , **s5OSW2-1)
OSW2-2s5 = {'bw':10000,'delay':'1ms','loss':0,'max_queue_size':10000,'speedup':10000}
net.addLink(OSW2-2, s5, cls=TCLink , **OSW2-2s5)

info( '*** Starting network\n')
net.build()
info( '*** Starting controllers\n')
for controller in net.controllers:
    controller.start()
```



```
  info( '*** Starting switches\n')
  net.get('s5').start([])
  net.get('OSW2-2').start([c0])
  net.get('OSW1-2').start([c0])
  net.get('s7').start([])
  net.get('s16').start([])
  net.get('s10').start([])
  net.get('s15').start([])
  net.get('s9').start([])
  net.get('s8').start([])
  net.get('OSW1-1').start([c0])
  net.get('OSW2-1').start([c0])

  info( '*** Post configure switches and hosts\n')
  OSW2-2.cmd('ifconfig OSW2-2 10.1.2-2')
  OSW1-2.cmd('ifconfig OSW1-2 10.1.1.2/16')
  OSW1-1.cmd('ifconfig OSW1-1 10.1.1.1/16')
  OSW2-1.cmd('ifconfig OSW2-1 10.1.2.1/16')
  print "Dumping host connections"
  dumpNodeConnections(net.hosts)
  print "Testing network connectivity"

  #def perfTest():

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

  net.pingAll()
  net.pingAll()
  print (" Test bandwidth b/w H1 and H2..............." );
  #H1, H2 = net.get('H1', 'H2')
  #net.iperf((H1, H2)

  #print (" Test bandwidth b/w H1 and H6..............." );
  H1, H6 = net.get('H1', 'H6')
  net.iperf((H1, H6))

  #print( " Test bandwidth b/w H1 and H10..............." );
H1, H10 = net.get('H1', 'H10')
  net.iperf((H1, H10))

  #print( "Test bandwidth b/w H1 and H15..............." );
  H1, H15 = net.get('H1', 'H15')
  net.iperf((H1, H15))

  # also argument for generating traffic

  # arugment for stat analysis

  CLI(net)
  net.stop()

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

REFERENCES

[1] https://cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-indexgci/Cloud_Index_White_Paper.pdf

[2] https://www.ciscoknowledgenetwork.com/files 477_11-11-2014-CiscoGCIDraftDeck2013-2018_CKN.pdf

[3] http://www.ciscopress.com/store/data-centre-virtualization-fundamentals-understanding-9781587143243

[4] http://www.graybar.com/applications/data-centres/types

[5] http://www.buusinessn ewsdaily.com/4982-cloud-vs-data-centre.html

[6] http://www.cyberciti.biz/faq/data-centre-standard-overview/

[7] http://www.graybar.com/applications/data-centres/tiers

[8] https://www.microsoft.com/en-us/research/publication/the-cost-of-a-cloud-research-problems-in-data-centre-networks/.

[9] Cisco systems: Data centre: Load balancing data centre services, 2004.

[10] http://research.microsoft.com/en-us/um/people/dmaltz/papers/DC-Costs-CCR-editorial.pdf

[11] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity datacentre network architecture," in ACM SIGCOMM, Aug. 2008.

[12] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: a high performance, server-centric network architecture for modular data centres," in ACM SIGCOMM, Aug. 2009.

[13] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: A scalable and fault-tolerant network structure for data centres," in ACM SIGCOMM, Aug. 2008, pp. 75–86.

[14] http://research.microsoft.com/en-us/um/people/srikanth/data/vl2_sigcomm09.pdf

[15] "3D MEMS Optical switch with toroidal concavemirror" https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201211ra1_s.html

[16] "High-yield Fabrication Methods for MEMS Tilt Mirror Array for Optical Switches" by Joji Yamaguchi †, Tomomi Sakata, Nobuhiro Shimoyama, "State of the Art of Optical Switching Technology for All-Optical Networks" http://home.deib.polimi.it/bregni/papers/cscc2001_optswitch.pdf.

[17] http://sanmapyourtech.blogspot.com/2014_08_01_archive.html

[18] https://www.sdxcentral.com/wp-content/uploads/2015/11/2015_SDxCentral_-SDN_Controllers-Report_Cisco_FINAL.pdf

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

[20] http://h17007.www1.hpe.com/docs/networking/solutions/sdn/devcentre/03_HP_OpenFlow OF_Technical_Overview_TSG_v1_2013-10-01.pdf

[21] http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html

[22] https://arxiv.org/ftp/arxiv/papers/1406/1406.0124.pdf

[23] http://upcommons.upc.edu/bitstream/handle/2117/77672/Current+Trends+of+Discovery+Topology+in+SDN.pdf?sequence=1

[24] https://www.researchgate.net/file.PostFileLoader.html?id=554cd4f7d2fd64b73e8b456c&assetKey=AS%3A273773314412545%401442284052403

[25] http://networkengineering.stackexchange.com/questions/7713/how-does-gratuitous-arp-work.

[26] http://www.taos.com/2014/07/16/understanding-gratuitous-arps/

[27] https://live.paloaltonetworks.com/t5/Management-Articles/Trigger-a-Gratuitous-ARP-GARP-from-a-Palo-Alto-Networks-Device/ta-p/61962

[28] Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures http://www.internetsociety.org/sites/default/files/10_4_2.pdf

[29] http://www.cisco.com/c/en/us/support/docs/ip/simple-network-management-protocol-snmp/8141-calculate-bandwidth-snmp.html

[30] http://internetofthingsagenda.techtarget.com/definition/micro-electromechanical-systems-MEMS

[31] https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/OpenFlow OF/OpenFlow OF-switch-v1.3.1.pdf

[32] https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/OpenFlow OF/OpenFlow OF-switch-v1.5.1.pdf

[33] https://www.mininet.org