+

Sudan University of Science and Technology
College of Graduate Studies

Performance Evaluation Of Open Network Operating System And Floodlight Controllers In Software Defined Network Based Internet Of Things

تقويم أداء متحكمات نظام تشغيل الشبكة المفتوحة والفلودلايت في إنترنت الأشياء المعتمدة على الشبكة المعرفة بالبرمجيات

A Thesis Submitted in Partial Fulfillment for the Requirement of the Degree of M.Sc. in Electronic Engineering (Computer and Networks Engineering)

Prepared By:
Sara Mohammed Bakri Mohammed
Supervised By:
Dr. Fath Elrahman Ismael Khalifa

March 2021

Sudan University of Science & Technology

College of Graduate Studies

جامعة السودان للعلوم والتكنولوجيا

كلية الدراسات العليا

بسم الله الرحمن الرحيم

كلية الدراسات العليا

Ref: SUST/ CGS/A11

Approval Page

(To be completed after the college council approval)

Name of Candidate: Sara | Mohammed | Bakri | Mohammed

Thesis title: Performance Evaluation of Open Network Operating System and Floodlight Controllers in Software Defind Network Based IoT

تقييم أداء متحكمات نظام تشغيل الشبكة المفتوحة والفلود لايت في البيئات الافتراضية للشبكة المعرفة بالبرمجيات

Degree Examined for: Master of Science in Electronics Engineering (Computer and Networks)

Approved by:

1. External Examiner

Name: Elsadig Saeed Gebreel Hamed

Signature: ..... Date: 8/4/2021

2. Internal Examiner

Name: Hisham Ahmed Ali

Signature: ..... Date: 8/4/2021

3. Supervisor

Name: Fath Elrahman Ismael Khalifa

Signature: ..... Date: 8/4/2021

الإستهلال

بِسْمِ اللهِ الرَّحْمنِ الرَّحِيمِ

(قَالُواْ سُبْحَانَكَ لاَ عِلْمَ لَنَا إِلاَّ مَا عَلَّمْتَنَا إِنَّكَ أَنتَ الْعَلِيمُ الْحَكِيمُ)

سورة البقرة-(32)

Dedication

I dedicate this work to those who were very caring, helpful and encouraging. To those who carried me for advancement and success.

To my Mother, my late father god blesses his soul, my brothers and sister who supported me and believed in my capabilities, my friends who were there for me.

Acknowledgement

First, I thank Allah for giving me the strength and ability to complete this research.

To the supervisor Dr. Fath Elrahman Ismael, the guider of my work, who gave me his trust in the knowledge and work, being determined and dedicated, helped and motivated me, to reach this form.

To my Mom, friends, family and stranger people those who pushed my goals till clarity supported me to the limit for it to be, sometimes over pushed it just so I succeed.

Abstract

The increase in the number of smart connected things has affected the performance of IoT applications, the problem statement of this research is that organizations requirements are becoming heterogeneous and stringent in terms of bandwidth, response time, throughput. Software Defined Networking (SDN) came as a solution that enables the network to adapt its behavior dynamically according to the traffic type which will increase the performance. In this research, the performances of the most used SDN controllers (ONOS, Floodlight) are analyzed and compared. Controllers analyzed based on some metrics such as throughput, delay, bandwidth using different topologies (single, linear, tree and SDN based IoT) using IoT traffic flow transmitted through MQTT protocol. ONOS shows throughput of 223.985, 340.3, 69.24 and 222.6 MB and floodlight shows throughput of 207.46 ,138.26, 255.33 and 228.5 MB in single, linear, tree and SDN-based IoT topologies respectively. ONOS shows bandwidth of 1.876, 2.85, 587.55 and 1.77 Gbit/sec and floodlight show bandwidth of 1.741, 1.4, 2.157 and 1.86 Gbit/sec in single, linear, tree and SDN-based IoT topologies respectively. ONOS show delay of 3.442, 7.15, 8.3and 0.062 msec and floodlight show delay of 2.85, 4.9, 5.28 and 0.039 msec in single, linear, tree and SDN-based IoT topologies respectively.

المستخلص

اثرت الزيادة في عدد الاشياء الذكية المتصلة بشبكة الانترنت على اداء تطبيقات انترنت الاشياء ومن اهم التحديات هي توفير إحتياجات المنظمات والتي قد تستخدم تقنيات مختلفة لتوفير خدمة معينة بمتطلبات معينة مثل النطاق الترددي ووقت الاستجابة والإنتاجية. جاءت البرمجيات المعرفة للشبكات كحل يمكّن الشبكة من تكييف سلوكها ديناميكيًا وفقًا لنوع البيانات المرسلة مما سيزيد من الأداء. في هذا البحث ، تم تحليل ومقارنة أداء وحدات تحكم البرمجيات المعرفة للشبكات الأكثر استخدامًا (نظام تشغيل الشبكة المفتوحة والفلودلايت) بناءً على بعض المقاييس مثل الإنتاجية والتأخير وعرض النطاق الترددي باستخدام بنيات مختلفة (فردي وخطي وشجري وانترنت الاشياء المبنية على البرمجيات المعرفة للشبكات) بإرسال البيانات بإستخدام بروتكول نقل الرسائل في قائمة الإنتظار عن بعد حيث يُظهر نظام تشغيل الشبكة المفتوحة إنتاجية تبلغ 223.985 و340.3 و 69.24 و 222.6 ميجابايت ويظهر فلودلايت 207.46 و 138.26 و 255.33 و 228.5 ميجابايت في البنية الفردية والخطية والشجرية وبنية انترنت الاشياء المبنية على البرمجيات المعرفة للشبكات على التوالي كما يُظهر نظام تشغيل الشبكة المفتوحة عرض نطاق ترددي 1.876 و 2.85 و 587.55 و 1.77 جيجابت / ثانية ويظهر فلودلايت عرض نطاق ترددي 1.741 و 1.4 و 2.157 و 1.86 جيجابت / ثانية في البنية الفردية والخطية والشجرية وبنية انترنت الاشياء المبنية على البرمجيات المعرفة للشبكات على التوالي واخيراً يظهرنظام تشغيل الشبكة المفتوحة تأخيرًا يبلغ 3.442 و 7.15 و 8.3 و 0.062 مللي ثانية ويظهر فلودلايت تأخير 2.85 و 4.9 و 5.28 0.039 مللي ثانية في البنية الفردية والخطية والشجرية وبنية انترنت الاشياء المبنية على البرمجيات المعرفة للشبكات على التوالي.

Table of Contents

List of Figures

List of Abbreviation

BYOD	Bring your own device
CoAP	Constrained Application Protocol
CP	Control plane
DP	Data Plane
FD	Forwarding Devices
GUI	Graphical User Interface
IERC	International Energy Research Centre
IOT	Internet of Things
ITU-T	International Telecommunication Union Telecommunication Standardization Sector
MQTT	Message Queue Telemetry Transport
NFV	Network Function Virtualization
NFV	Network Functions Virtualization
NI	Northbound Interface
NOS	The Network Operating System
QoS	Quality of Service
SDN	Software Defined Networking
SI	Southbound Interface
TCP	Transmission Control Protocol
VPN	Virtual Private Network

Chapter One

1-Introduction

1.1. Preface

The next wave in the era of computing will be outside the realm of the traditional desktop[1]. A growing number of physical objects are being connected to the Internet at an unprecedented rate realizing the idea of the Internet of Things (IoT)[2] which represent the Future of the internet. According to International Telecommunication Union Telecommunication Standardization Sector (ITU-T)and International Energy Research Centre (IERC) the IoT, first introduced in 1999 [3].

The Internet of Things (IoT) is a recent communication paradigm that envisions a near future, in which the objects of everyday life will be equipped with microcontrollers, transceivers for digital communication, and suitable protocol stacks that will make them able to communicate with one another and with the users and make decision using locally- or globally-gathered data.[4]

The recent surge in popularity of the Internet of Things (IoT) across multiple domains has stemmed from the spread of networking-enabled consumer devices that are deployed on a geographically wide-scale[5]it is estimated that by 2020 IoT is going to cover between 26 billion and 50 billion devices [6].

Advancement in wireless networking has let these thousands of smart devices connect to the internet anywhere and anytime. With the development of IoT, the amount of data produced per day increases exponentially[7, 8].

Nowadays we are also in the cloud computing and big data era in which most of computing and communication resources are shared and provided to users. The characteristics of diversity, dynamics, and big data explosion bring a big challenge for the design of the IoT architecture in the cloud and big data era. Networks should now be more intelligent, more powerful, more efficient, more secure, more reliable, and more scalable to meet the requirements of the characteristics of diversity and dynamics.

Also, the use of computing devices and communication technologies are growing exponentially with the decline in cost and size of hardware and software. Vendors and organizations are digging new domain in search of finding new ways of flexible computing and communication which provided by Software Defined Network (SDN). IoT and SDNs are two completely different communication and network domain whose merger is seeking for benefiting human kinds and developing smart systems. As the IoT implementation expectancy exceeds the limits of traditional network e.g., Virtual Private Network (VPN), the SDN promise to hold the traditional network with new service demands.[9].

1.2. Problem Statement

Due to the huge number of heterogeneous devices, it is required that IoT systems support the increasing number of connected devices; it is a necessary requirement that information exchange takes place between all the interconnected IoT devices taking into consideration bandwidth, response time, throughput.

1.3. Proposed Solution

By exploit the programmability and the flexibility of SDN an IoT model using SDN is proposed to enable the network to adapt its behavior

dynamically according to the traffic type which will increase the performance.

1.4. Aim and Objectives

The main objectives of this work is to analyze and evaluate the performance of software defined network controllers ONOS and floodlight in term of bandwidth, throughput and delay using different topologies through experiments and see how to utilize SDN controllers in IoT.

1.5. Methodology

Firstly, a comprehensive investigation on SDN and IoT is done to achieve our goal which is to evaluate the performance of SDN controllers ONOS and floodlight in single, linear, tree and SDN-based IoT topologies in term of bandwidth, throughput and delay.

To achieve this goal, we implemented networks using Mininet software which is software comes as a pre-built virtual machine (VM) image, The VM was allocated two 1.7 GHz Intel Core 7 processors and 4 gigabytes of RAM (Random Access Memory) then Mininet 2.2.0 VM image was downloaded and installed and controllers were installed on the Ubuntu 14.0 to be evaluated starting with ONOS controller that used in single, linear, tree and SDN-based IoT topologies then bandwidth and throughput was measured using iperf and delay was measured using ping between tow hosts in the network , for the IoT topology a python script is used to create the topology and Eclipse Mosquitto which is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol that used to carry out messaging using a publish/subscribe model in IoT .the same procedure is done for floodlight controller and each time the results was documented.

1.6. Thesis Outline

The thesis is reported in five chapters the first chapter includes Preface, problem statement, proposed solution, objectives and methodology. The second chapter is literature review which contains a general overview of SDN and IoT their architecture, protocols, services and applications in addition, some of previous studies of them brief definition of the tools and software used in the methodology. The third chapter is system implementation and configuration which contains detailed explanation of all stages to implement the proposed scenario .the fourth is result and discussion in which the performance of different SDN controllers (ONOS and FloodLight) was evaluated in term of bandwidth, throughput and delay using different topologies (single, linear , tree and SDN-based IoT topology) and the last chapter is conclusion and recommendation which concludes the work done in this research and presents the recommendation for future work.

Chapter Two

Literature Review

2. Overview

This chapter contains a general overview of SDN and IoT their Architecture, protocols, services and applications in addition, some of previous studies of them.

2.1. Software-Defined Networking (SDN)

This section contains a general overview of SDN its architecture, protocols, services and applications

2.1.1. SDN Definition

Software-Defined Networking (SDN) is a network architecture approach that enables the network to be intelligently and centrally controlled, or 'programmed,' using software applications. This helps operators manage the entire network consistently and holistically, regardless of the underlying network technology.

2.1.2. SDN Architecture

SDN is a layered architecture, consisting of three basic layers; application/services layer, a controller layer, and data plane layer called forwarding layer consisting of forwarding devices. These SDN layers communicate with each other via open APIs called Northbound Interface (NI) API and Southbound Interface (SI) API as in figure 2-3

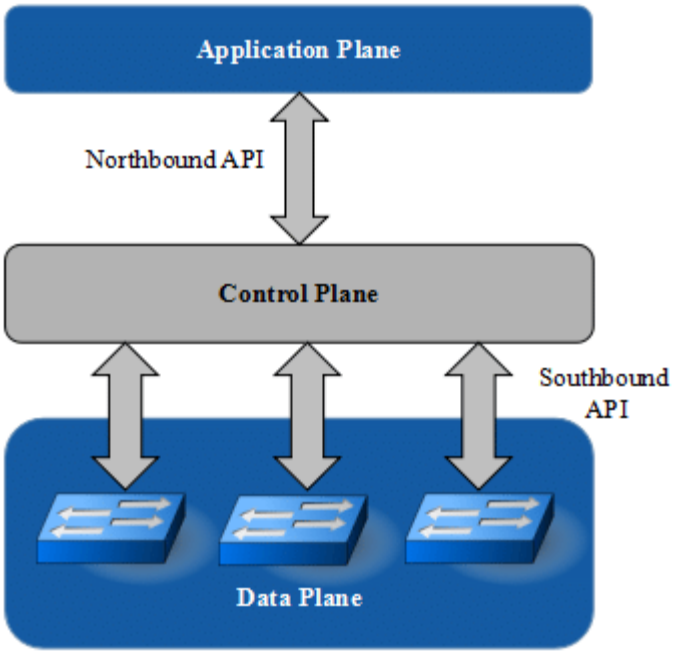

Figure 2-1. SDN Architecture

[10] describe them as follow:

2.1.2.1. Application Layer (AP):

The application plane also called management plane consist of applications that leverage the functions offered by the NI to implement network control and operation logic. Essentially, a management application defines the policies, which are ultimately translated to southbound-specific instructions that program the behavior of the forwarding devices[9].

2.1.2.2. Northbound Interface (NI):

Is the interface between applications and the controller. It provides access to network resources from the application level.

The Network Operating System (NOS) facilitate application developers to coordinate through these NI APIs. Typically, an NI APIs abstracts the low-level instruction sets and implementation of forwarding devices. So far NI

APIs is not well studied. Generally, RESTFull APIs are used as an interface between applications and control plane [11].

2.1.2.3. Control Plane (CP):

Control plane is the most intelligent and important layer of an SDN architecture. It contains one or various controllers (ONOS , Floodlight , NOX , Ryu and OpenDaylight) that forward the different types of rules and policies to the infrastructure data layer through the southbound interface [12].

2.1.2.4. Southbound Interface (SI):

Southbound interfaces provide a communication protocol between CP and forwarding device though the SI instruction set. Well established SI protocol help controller in programming forwarding devices and formalize rules for interaction between the two planes (CP & DP). Some examples are OpenFlow [13], Forwarding and Control Elements (ForCES)[14] , Protocol-oblivious forwarding [15].

2.1.2.5. Data Plane (DP)/Forwarding Plane:

Represents the forwarding devices on the network (routers, switches, load balancers, etc.). It uses the south-bound APIs to interact with the control plane by receiving the forwarding rules and policies to apply them to the corresponding devices[16].

2.1.3. SDN Basic Principles

Given the heterogeneity of networks, it is challenging to coordinate and optimize the use of the heterogeneous network resources with the goal of satisfying as many tasks and services as possible. It is conjecture that the SDN paradigm is a good candidate to solve the resource management needs for network environments for multiple reasons[17] SDN allows for a clear separation between services in the control plane

(that makes decisions about how traffic is managed) and the data plane (actual mechanisms for forwarding traffic to desired destinations). The decoupling of the control plane from the forwarding plane encourages abstractions of low-level network functionalities.

Logically centralized view of the network, which allows to perform network optimization techniques. Redundancy and other mitigation failures can be applied in order to avoid single points of failure.

Network programmability allows the dynamic and fast introduction of new network services

2.1.4. SDN Protocol (OpenFlow)

Sdn use openflow protocol which support a lot of technology .In the creators' own words, OpenFlow is a communications protocol that provides an abstraction of the forwarding plane of a switch or router in the network.[18] OpenFlow system initially created at Stanford University now under dynamic gauges improvement through the Open Networking Foundation (ONF). Open Networking foundation (ONF) defined that OpenFlow protocol is based on SDN layered architecture, it is in between control plane and forwarding plane as communication protocol (Open Flow Switch Specification. Open Networking Foundation ONF)[19]

Focusing on its main features, OpenFlow:

- Brings network control functions out of switches and routers, while allowing to directly access and manipulate the forwarding plane of those devices.
- species basic primitives that can be used by an external software application to actually program the forwarding plane of network devices, just
- like the instruction set of a CPU would program a computer system.

- works on a per-ow basis to identify network track.
- Forwards flows according to preened match rules statically or dynamically programmed by the SDN control software.

2.1.5. SDN Controllers

the SDN controller is a software entity that has exclusive control over an abstract set of data plane resources.An SDN controller may be implemented as any number of software components, which reside on any number of physical platforms[17].

Several open-source implementations of an SDN controller are available, being the most important the following:

2.1.5.1. ONOS Conrtoller:

ONOS controller is a dynamic flexible platform easy operates in any OSs capabilities and strongly supports switching, routing, and distributed application architected for performance, high availability, scale-out and well-defined northbound and southbound abstractions and interfaces. ONOS was open- sourced on December 5th, 2014 [20].

2.1.5.2. Floodlight Controller:

The Floodlight Open SDN Controller is a company-class, open source, Java-based OpenFlow Controller. It is supported by a group of developers among them of engineers from Big Switch Networks. OpenFlow protocols are an open standard managed by ONF. It specifies a protocol through switch a remote controller can modify the behavior of networking devices through a well-defined “forwarding instruction set” Floodlight is designed to work with the increasing number of switches, routers, virtual switches, and access points that support the OpenFlow standard [21].

2.1.5.3. OpenDaylight Controller

OpenDaylight controller is an open-source project supported by IBM, Cisco, Juniper, VMWare and several other major networking vendors.It is SDN controller platform implemented in Java. As such, it can be deployed on any hardware and operating system platform that supports Java. It robust and provides production-level performance and support. Its main drawback is the complexity and the fact that it takes time for learning to develop applications [22].

2.1.5.4. Ryu Controller

The Ryu Controller is open source and under the Apache 2.0 license, written completely based on Python, supported and deployed by NTT cloud data centers. Main source code can be found on GitHub, provided and supported by Open Ryu community. It supports NETCONF and OF-config network management protocols, as well as OpenFlowis .it is component-based software defined networking framework that provides software components with well-defined API that make it easy for developers to create new network management and control applications [23].

2.1.6. SDN Applications

In this section SDN applications are introduced including Software defined ICN, cloud and data center and BlueCat DNS director.

2.1.6.1. Software Defined Information Centric Network (SDICN)

1.In recent years many researchers claimed that current internet architecture is not able to response the emerging and future need of users. Based on this claim, new architectures were introduced. Information centric network is one of these architectures. In ICN, the information name is unique and independent of locations, applications, storages and

distribution and network primitives are done based on the names. To retrieve named information, various transmission techniques are introduced, including name-based routing, name-based resolution, etc. To support these techniques and exploit the advantages of ICN, dramatic changes to the network devices deployed in current Internet are needed, which leads to challenge of ICN implementation. Implementing ICN over SDN enables innovation and optimization of network resources and functionalities. This leads to decreasing in implementation costs. It also enables innovation and optimization of network resources and functionalities[24]

2.1.6.2. Cloud and Data Center

One area that SDN has been attended a lot is Cloud Services and data center. One of the main characteristics of cloud is that users gain the adequate resources based on requirement in real time [109]. Cloud management is the most important challenge that has always been, and many solutions have been proposed for that. SDN is highly regarded as one of the newest solutions, which makes it possible to configure and manage cloud and data center easily[25].

2.1.6.3. Bluecat DNS Director

This application is targeted towards security threats caused by BYOD (Bring your own device). DNS director programs Openflow switches in the network using HP VAN SDN controller to redirect requests for non-corporate DNS servers towards

BlueCat’s DNS server. BlueCat’s DNS server sends back proper DNS response and the requestor will not even know that the DNS request was intercepted [26].

2.2. Internet of Thing(IoT)

This section contains a general overview of IoT its architecture, protocols, services and applications

2.2.1. IoT Definition

IoT Defined as a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to throughput data over a network without requiring human-to-human or human-to-computer interaction[27]

2.2.2. Iot Architecture:

In this section, an overview of the IoT architecture was briefly highlighted

The structure of the IoT consists of five layers: Business layer, Application layer, Middleware layer, Network layer and Perception layer.

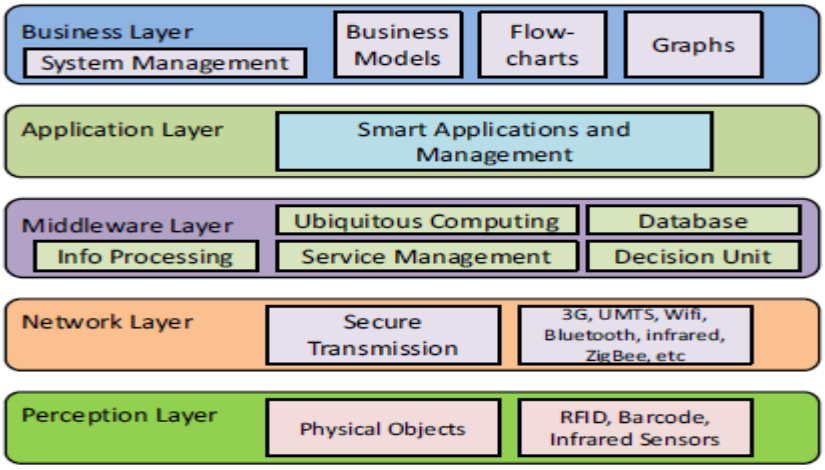

Figure 2-2. IoT Architecture

2.2.2.1. Business Layer

This layer is responsible for the management of overall IoT system including the applications and services. It builds business models, graphs, flowcharts. etc based on the data received from Application layer. The real

success of the IoT technology also depends on the good business models. Based on the analysis of results, this layer will help to determine the future actions and business strategies [28].

2.2.2.2. Application Layer

This is the layer that delivers IoT services to end users through IoT applications and has been placed at the top of the ACO architecture. It acts as an interface for the users to remotely send commands and receive data and information from the objects. Users are also able to visualize the IoT data, which has been analyzed in the cloud services layer, through the applications.

The applications also allow administrators and users to configure devices, and define access control polices for securing access to IoT resources and data [29].

2.2.2.3. Middleware Layer

The devices over the IoT implement different type of services. Each device connects and communicates with only those other devices which implement the same service type. This layer is responsible for the service management and has link to the database.It receives the information from Network layer and store in the database. It performs information processing and ubiquitous computation and takes automatic decision based on the results[30].

2.2.2.4. Network Layer

The network layer is like the neural network and brain of loT, its main function is transmitting and processing information. The network layer includes a convergence network of communication and Internet network, network management center, information center and intelligent processing

center, etc. The network layer will transmit and process the information obtained from perception layer[31].

2.2.2.5. Perception Layer

Perception layer is the lowest layer in the IoT architecture. As the name suggests, its purpose is to perceive the data from environment. All the data collection and data sensing part is done on this layer Sensors, bar code labels, RFID tags, GPS, and camera, lie in this layer. Identifying object/thing and gathering data is the main purpose of this layer [32].

2.2.3. IoT Protocols:

This section contains a general overview of IoT

2.2.3.1. Message Queue Telem etry Transport (MQTT)

The Message Queuing Telemetry Transport (MQTT) is a standard protocol of Organization for the Advancement of Structured Information Standards (OASIS) standard. It is light weights publish/subscribe messaging transport application level protocol. It uses TCP/IP protocol and designed for machine-to-machine and IoT purpose[33].

2.2.3.2. Constrained Application Protocol (CoAP)

The Constrained Application Protocol (CoAP) has been designed by Internet Engineering Task Force (IETF) to facilitate message throughput between machine-to-machine (M2M) applications by offering various features such as built-in discovery, multicast support and asynchronous message exchanges. The main goal of CoAP is to design a web protocol on top of UDP for special environments that consist of constrained nodes (e.g., resources, computing power) and networks (e.g., low-power, loss). There are more than 30 CoAP implementations written in various languages including C,C++, Java, Python, JavaScript, etc[34].

2.2.3.3. Websocket Protocol

the WebSocket Protocol also has been designed by IETF to provide full-duplex communication between the web browser and server to overcome the existing bidirectional communication technologies using HTTP while supporting existing HTTP infrastructures over HTTP ports—80 and 443. Although the WebSocket Protocol has been designed to supersede existing HTTP technologies, it recently has been used for the IoT applications that require real-time communication[35].

2.2.4. IoT Application

Potential applications of the IoT are numerous and diverse, permeating into practically all areas of every-day life of individuals, enterprises, and society as a whole. The IoT application covers “smart” environments/spaces in domains such as: Transportation, Building, City, Lifestyle, Retail, Agriculture, Factory, Supply chain, Emergency, Healthcare, User interaction, Culture and tourism, Environment and Energy. Below are some of the IOT applications:

2.2.4.1. Smart Cities

The IoT play a vital role to improve the smartness of cities includes many applications to monitoring of parking spaces availability in the city, monitoring of vibrations and material conditions in buildings and bridges, sound monitoring in sensitive areas of cities, monitoring of vehicles and pedestrian levels, intelligent and weather adaptive lighting in street lights, detection of waste containers levels and trash collections, smart roads, intelligent highways with warning messages and diversions according to climate conditions and unexpected events like accidents or traffic jams [36].

2.2.4.2. Smart Agriculture and Smart Water

The IoT can help to improve and strengthen the agriculture work by monitoring soil moisture and trunk diameter in vineyards to control and maintain the amount of vitamins in agricultural products, control micro climate conditions to maximize the production of fruits and vegetables and its quality, study of weather conditions in fields to forecast ice information, rail, drought, snow or wind changes, control of humidity and temperature level to prevent fungus and other microbial contaminants. The role of IoT in water management includes study of water suitability in rivers and the sea for agriculture and drinkable use, detection of liquid presence outside tanks and pressure variations along pipes and monitoring of water level variations in rivers, dams and reservoirs[37].

2.2.4.3. Retail and Logistics Implementing

The IoT in Retail/Supply Chain Management has many advantages which include monitoring of storage conditions along the supply chain and product tracking for traceability purposes and payment processing based on location or activity duration for public transport, gyms, theme park, etc. In the shop itself, IoT offers many applications like guidance in the shop according to a preselected shopping list, fast payment solutions like automatically check-out using biometrics, detection of potential allergen in a given product and control of rotation of products in shelves and warehouses to automate restocking processes. The IoT elements used in this kind of application are RFID and WSN and the bandwidth range is small. The example retail IoT reported in literature is SAP future retail center. The IoT in logistics includes quality of shipment conditions, item location, storage incompatibility detection, fleet tracking, etc. The IoT elements used in the field of logistics are RFID, WSN and single sensors

and the bandwidth ranges from medium to large[38].

2.2.4.4. Healthcare

Many benefits provided by the IoT technologies to the healthcare domain are classified into tracking of objects, staff and patients, identification and authentication of people, automatic data collection and sensing. Tracking is the function used to identify a person or an object in motion. This includes the case of patient flow monitoring to improve workflow in hospitals. The identification and authentication include patient identification to reduce incidents harmful to patients, comprehensive and current electronic medical record maintenance, and infant identification in hospitals to prevent mismatching. The automatic data collection and throughput is mostly aimed at reducing form processing time, process automation, automated care and procedure auditing, and medical inventory management. Sensor devices enable function centered on patients, and in particular on diagnosing patient conditions, providing real-time information on patient health indicators. Application domains include different telemedicine solutions, monitoring patient compliance with medication regiment prescriptions, and alerting for patient well-being. In this capacity, sensors can be applied both in in-patient and out-patient care. The elements of IoT in Health Care are RFID, NFC, WSN, WiFi, Bluetooth, etc. significantly improve the measurement and monitoring methods of vital functions such as temperature, blood pressure, heart rate, cholesterol level, blood glucose, etc. [39].

2.3. Related Work

Recently, SDN has proved its benefits and thus applied to many networking environments such as IoT. Here, SDN has an impact in IoT by

its flexibility to adapt the components and the applications dynamically. So that many researchers proposed an SDN-based architecture for IoT:

The author of [40] reviewed SDN controller and Take advantages of the decentralized feature of it and proposed Ubiflow software defined IoT system flow control and mobility management in a heterogeneous network.

The author s in [5]also proposed an original SDN controller design in IoT Multi-networks that enables flexible, effective, and efficient management on task, flow, network, and resources.

Although, others proposed software defined based framework for IoT. Jararweh and al in [41] proposed a SDIoT software defined based framework model to simplify the IoT management process and challenges in the traditional IoT architecture to forward, store, and secure the produced data from the IoT objects by integrating the software defined network software defined storage, and software defined security into one software defined based control model.

The authors of [42] developed an SDN-Based layered architecture for horizontal IoT services with open and programmable devices and data at different levels they proposed two network models, the man-like nervous (MLN) model and the social organization framework (SOF) model. The idea of MLN model is similar to a reverse tree in which the sensors are in the low rank of the model, and it becomes a distributed node to send sensing data to the centralized data center. Additionally, the SOF model consists of local IoT, industrial IoT, and national IoT that manages the distributed nodes in the specific region and the data center.

The theme of [42]is that IoT is based on the data center and manages the data from the sensors based on the MLN model. However, the specific

usage of the IoT network is not provided yet.

Another architecture of the IoT network was described by Catellani et al. [43]. In this work,the authors created a test bed for an IoT environment by utilizing the legacy sensors and actuators by modifying it with TinyOS. The authors envisioned that the future IoT network would be based on Internet Protocol version 6 (IPv6), and every communication will be conducted with IP addresses.

Moreover, instead of enumerating the future networking protocol, the authors categorized the nodes in terms of base station node (BSN), mobile node (MN), and specialized node (SN). BSN refers to the IPv6 sink and router, MN refers to wireless dongle to add wireless sensor network (WSN) connectivity to a standard laptop, and SN refers to nodes offering services such as temperature readings or actuation.

Finally, Gubbi et al. [11] claimed that future IoT network will be based on the current WSN technology. The nodes are expected to be deployed in an ad-hoc manner, and a novel cyber infrastructure is based on a service-oriented architecture (SOA) and sensor networks to acquire the data from the sensors and the devices. Furthermore, the addressing schemes will be based on IPv6 and possible adaptation with the uniform resource name (URN) system for development of the IoT network. The authors claimed that cloud-based centralized storage is required to support the storage and analysis for IoT. Consequently, the authors noted that the future IoT network will be based on current WSN technology with the middleware to support heterogeneous devices and a centralized storage for the data.

2.4. System Tools

This section describes software tools that have been used in the simulation

2.4.1. VM VirtualBox

VirtualBox is free and open-source software that can be installed on a number of host operating systems including Linux, mac-OS and Windows VirtualBox is used here to run mininet hosted in Ubuntu Linux operating system which is used to implement the different topologies under evaluation.

2.4.2. Ubuntu Linux

Ubuntu Linux is a Linux distribution based on Debian and mostly composed of free and open-source software. Ubuntu is officially released in three editions: *Desktop*, *Server* and *Core* for Internet of things devices and robots. All the editions can run on the computer alone, or in a virtual machine. Ubuntu is a popular operating system for cloud computing, with support for OpenStack

2.4.3. SDN Hub

SDN Hub created a starter kit tutorial VM, which comprises various components that will facilitate SDN development.

The VM is Ubuntu 64-bit based, which is preinstalled with various software and tools Controllers (OpenDaylight, ONOS, RYU, Floodlight, Floodlight-OF1.3, POX, and Trema), Open VSwitch, Mininet, Wireshark 1.12.1 with native support for OpenFlow parsing, JDK 1.8, Eclipse Luna, and Maven 3.3.

2.4.4. Mininet

Mininet is a network emulator which is able to emulate a linked set of virtual hosts, switches, and controllers. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly exile custom routing and SDN.

Mininet supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete

experimental network on a laptop or other PC.

2.4.5. Iperf

Iperf is a tool for network performance measurement and tuning. It is a cross-platform tool that can produce standardized performance measurements for any network. Iperf has client and server functionality and can create data streams to measure the throughput between the two ends in one or both directions. Typical iperf output contains a time-stamped report of the amount of data throughputred and the throughput measured.

2.4.6. Eclipse Mosquitto

Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers.

The MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for Internet of Things messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers.

The Mosquitto project also provides a C library for implementing MQTT clients, and the very popular mosquitto_pub and mosquitto_sub command line MQTT clients.

2.4.7. Paho Python Client

The Paho Python Client provides a client class with support for both MQTT v3.1 and v3.1.1 on Python 2.7 or 3.x. It also provides some helper functions to make publishing one off messages to an MQTT server very straightforward

Chapter Three

Methodology

3. System Implementation and Configuration

In this chapter, we discuss about software's that we used in simulations and finally the implementations of SDN networks.

3.1. System Implementation

SDN hub installed in the virtual machine as in figure (3-1) and different topologies (single, linear and tree) with different SDN controllers (ONOS and Floodlight) are constructed using mininet tool

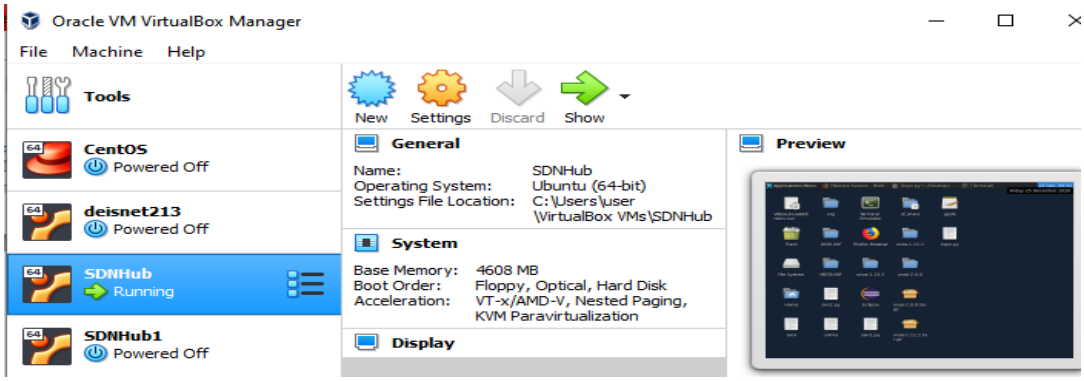

Figure 3-1creates SDNhub virtual machine

ONOS controller installed using the commands:

root@sdnhumvm:/home/ubuntu/Desktop/onos-2.0.0/bin# ./onos-service start

FloodLight controller installed using the commands:

root@sdnhumvm:/home/ubuntu/Desktop/floodlight# java - jar target/floodlight.jar

3.2. System Configuration

In this section configuration of the different topologies is detailed as follows:

- Single topology: The single topology in Mininet environment consists of a single OpenFlow-enabled switch and number of hosts. The switch in turn gets connected with a control plane available on the topology. A single topology that connected to a floodlight controller or ONOS controller can be created by using a command-line tool as shown in Fig. 3-2 with 5-hosts follow the structure command below:

Sudo mn –topo single, 5 - -controller remote, ip=127.0.0.1,port=6653

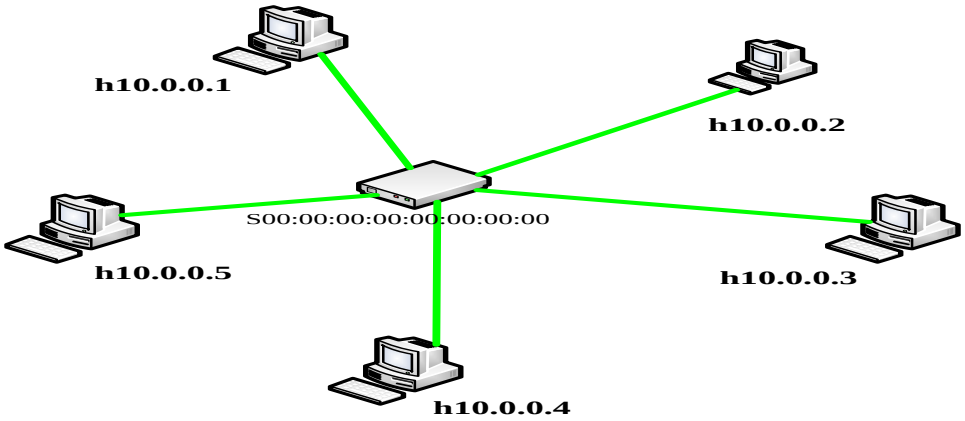

Figure 3-2 Single Topology

- Linear Topology: Is including linear connection between switches and hosts. Each host connects with its particular switch and the switches are connected with each other linearly. All the OpenFlow-enabled switches in turn gets connected with a remote controller. A linear topology having 5-hosts connected with Floodlight controller or ONOS controller topology that connected to a floodlight controller or ONOS controller can be created by using a command-line tool as shown in Fig. 3-3.

Sudo mn –topo linear, 5 - -controller remote, ip=127.0.0.1,port=6653

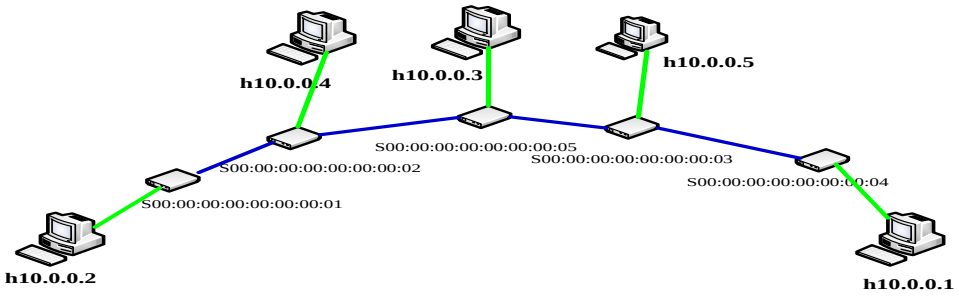

Figure 3-3 Linear Topology

- Tree Topology: All OpenFlow-enabled switches and hosts are linked with each other in a hierarchical fashion. A tree topology having 32-hosts in floodlight controller or ONOS controller topology that connected to a floodlight controller or ONOS controller can be created by using a command-line tool as shown in Fig. 3-4.

Sudo mn –topo tree,3 - -controller remote, ip=127.0.0.1,port=6653

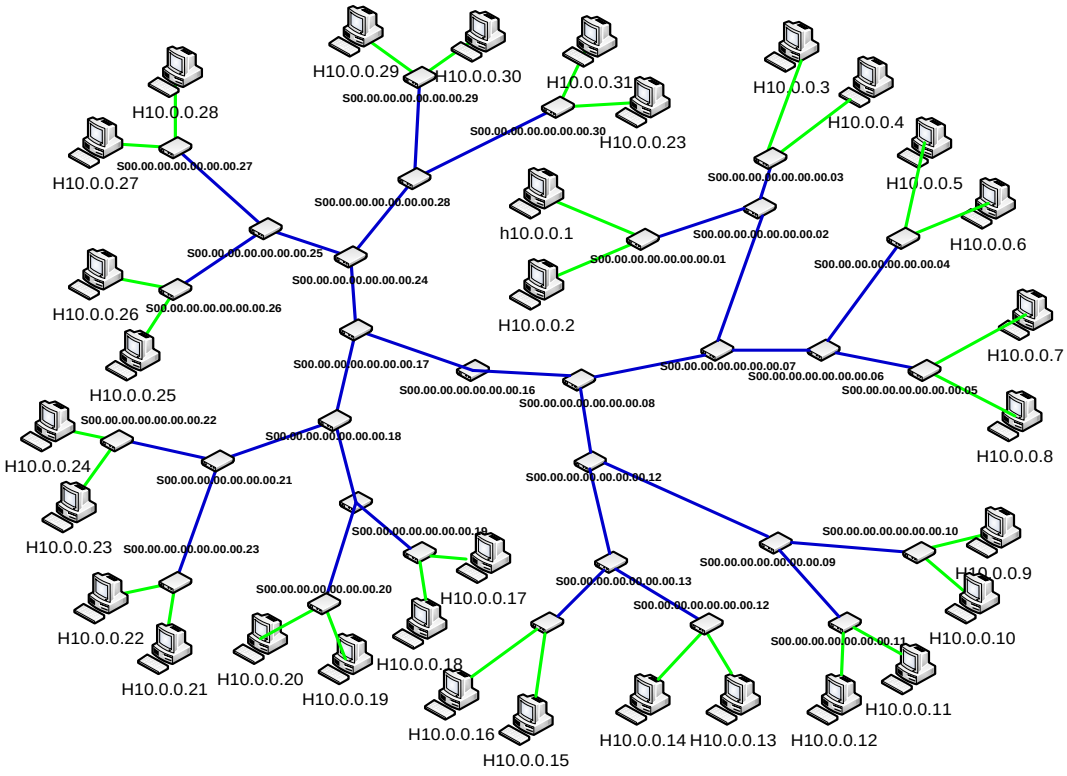

Figure 3-4 Tree Topology

- IoT Topology:In this section, An SDN-based Iot topology that uses MQTT protocol which can be used in IoT devices from different manufacture that uses different technologies is created using python script as shown in Fig. 3-5.

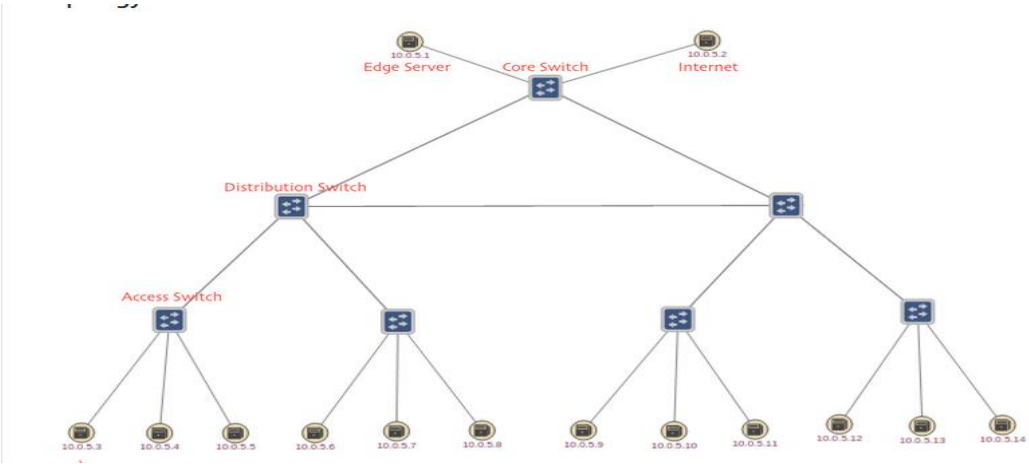

Figure 3-5 IoT Topology

MQTT use mosquito and mosqitto-clients and paho mqtt python which are a light weight open source message broker that Implements MQTT. mosquito configured using the command:

```
Sudo apt-get install mosquito
```

mosqitto-clients configured using the command:

```
Sudo apt-get install mosquito- clients
```

paho mqtt python configured using the command:

```
git clone https://github.com/eclipse/paho.mqtt.python
cd paho.mqtt.python
python setup.py install
```

Then part of the hosts used to create a very simple two state controllable sensors using python script as shown in Fig. 3-6. The sensors can be used to simulate real world objects like lights, doors using the following command:

```
root@sdnhubvm:$python simple-sensor.py -h test.mosquitto.org -n door -s
```

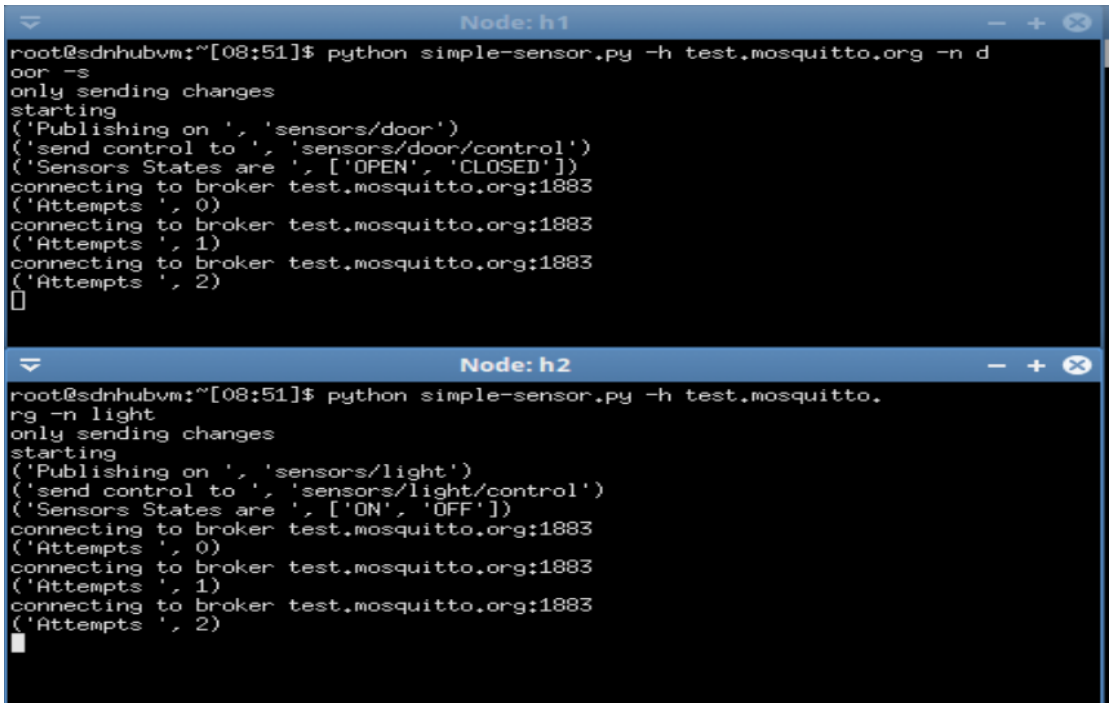

Figure 3-6 create sensors using python

All the sensors controlled by the remote SDN Controller use MQTT which is an application protocol that runs over the TCP/IP protocol using the publish-subscribe pattern. Then test.mosquitto.org which is online MQTT broker and two hosts (sensors) are used as publisher and subscriber. Both can publish messages or subscribe to topic request it from broker (server) who is listening at port 1883 as shown in Fig. 3-7.

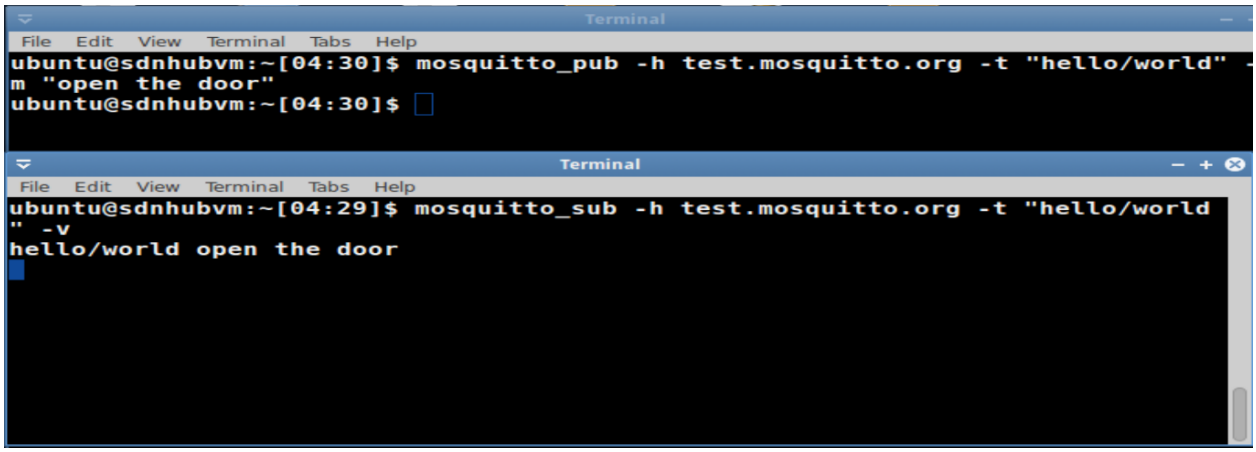

Figure 3-7: subscribe and publish to topic by sensors

Chapter Four

Results and Discussion

4. Results

In this chapter the performance of different SDN controllers (ONOS and FloodLight) was evaluated in terms of bandwidth, throughput and delay using different topologies (single, linear and tree and SDN-based architecture for IoT as follow:

4.1. Single Topology

The following section provides the results obtained by testing the ONOS and Floodlight controllers in simulation environment that implemented in Mininet.

To measure the throughput, bandwidth and delay single topology is created as shown in figure 4-1 using onos controller and figure 4-4 using floodligt controller and then iperf which use one of the host as server and the other as client was used as in figure 4-2, figure 4-3, figure 4-5 and figure 4-6

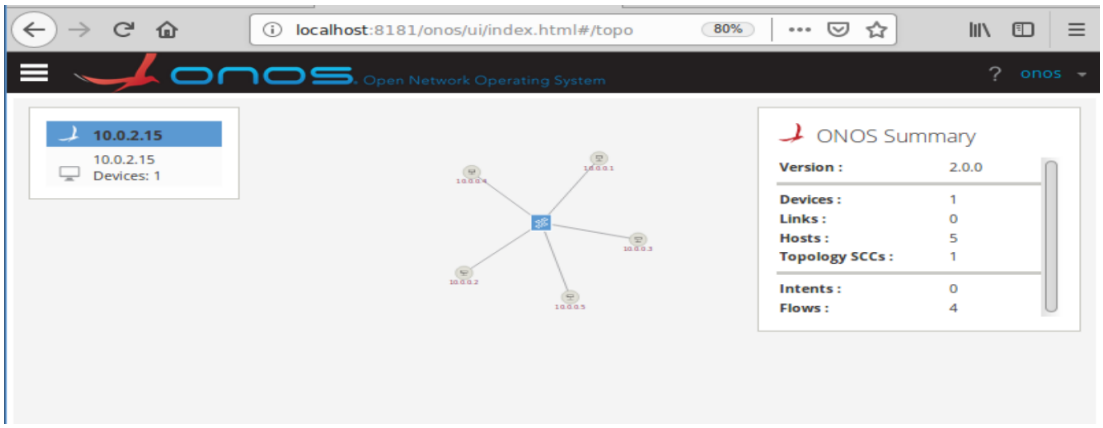

Figure 4-1 Single Topology ONOS GUI

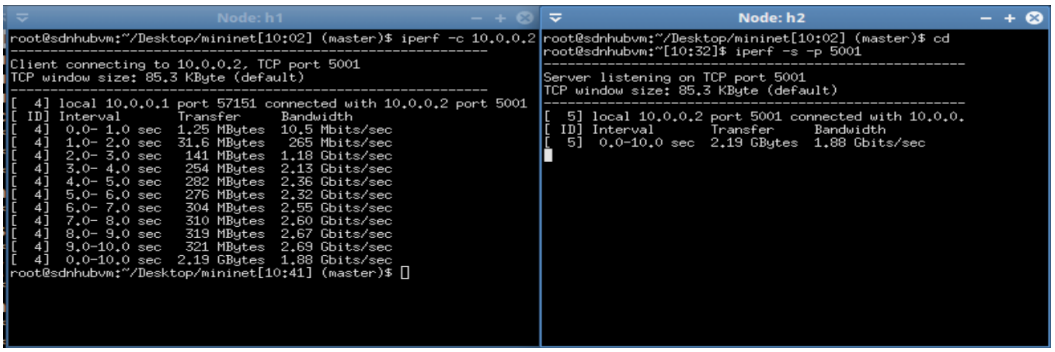

Figure 4-2 throughput and bandwidth in onos- based single topology

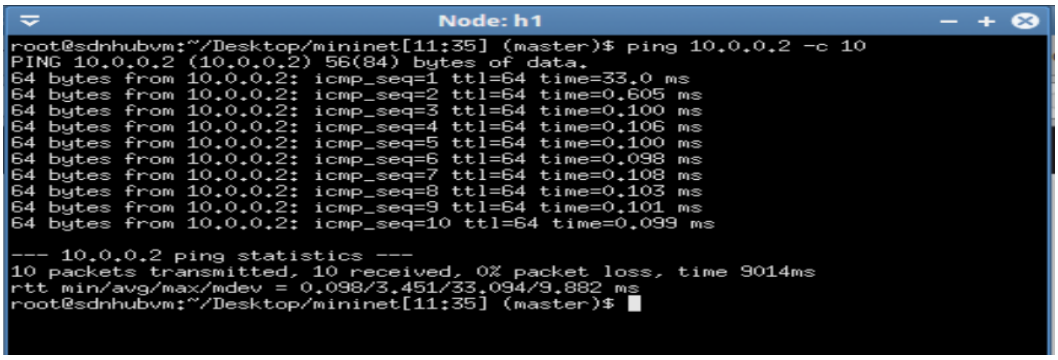

Figure 4-3 delay in onos- based single topology

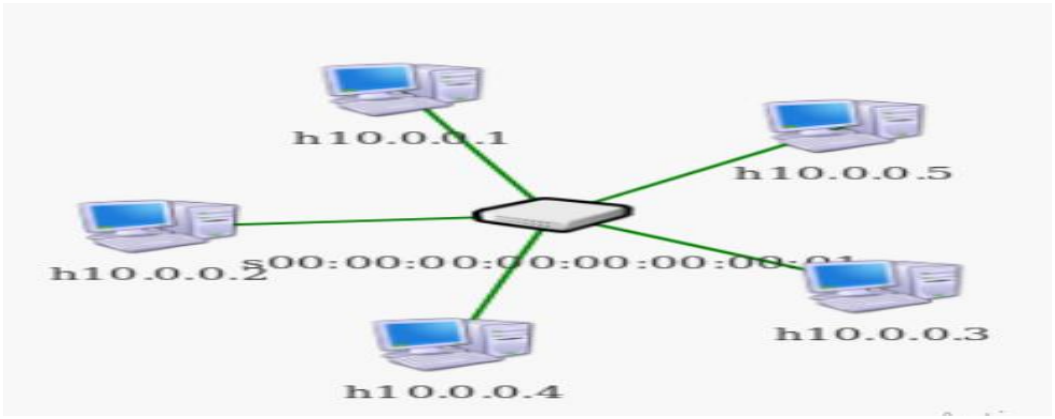

Figure 4-4 Single Topology Floodlight GUI

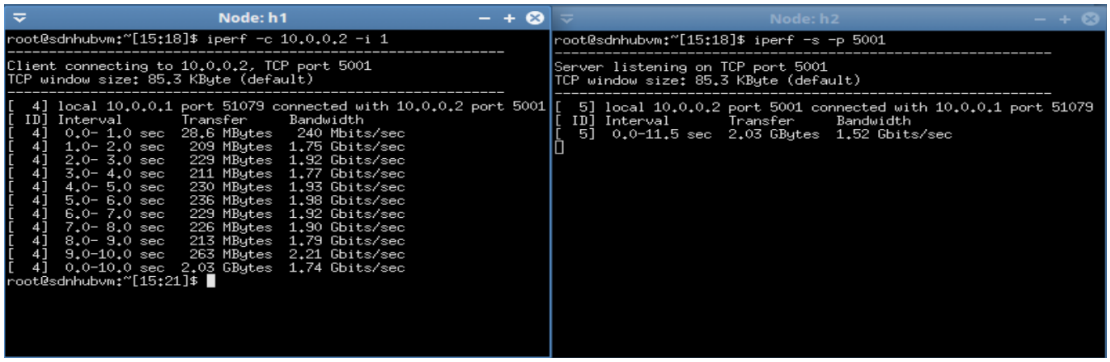

Figure 4-5 throughput and bandwidth in Floodlight- based single topology

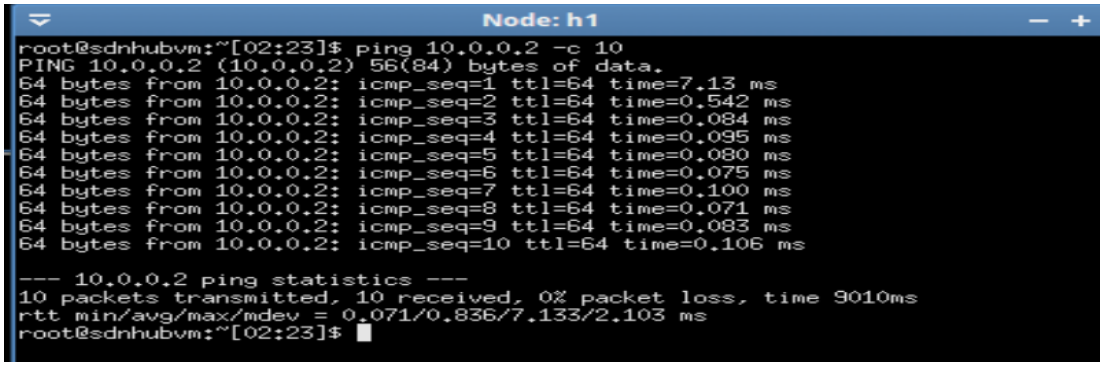

Figure 4-6 delay in Floodlight- based single topology

Single topology reflects the simple way for evaluating controller behavior. Fig 4-7, Fig 4-8 and Fig 4-9 show the throughput, bandwidth and delay over a single topology, respectively. It's clear that ONOS controller gives performance better than floodlight controller when it comes to Throughput and Bandwidth and Floodlight give better performance when it comes to delay.

The traffic Throughput of ONOS ranged is from 1.25 to 321 MB and increase dramatically from 1.25 to 282 MB at the first five seconds then slightly decrease to 276 MB at second number six then continue to increase until reach the highest throughput value which is 321 MB at the last time interval, On the other hand the throughput of floodlight ranged is from 28.6 to 263 MB and increase dramatically from 28.6 to 229 MB at

the first three seconds then slightly decrease to 211 MB at the next time interval then increase to 236 at the following two time interval and decrease to 213 at the following three time interval until reach the highest throughput value which is 263 MB at the last time interval .

The Bandwidth of ONOS ranged is from 0.01 to 2.69 Gbits/sec. and increase dramatically from 0.01 to 2.36 Gbits/sec MB at the first five seconds then slightly decrease to 2.32 Gbits/sec at the next time interval then continue to increase until reach the highest Bandwidth value which is 2.69 Gbits/sec at the last time interval; On the other hand the Bandwidth of floodlight ranged is from 0.24 to 2.21 Gbit/sec and increase dramatically from 0.24 to 1.92 Gbit/sec at the first three seconds then slightly decrease to 1.77 at the next time interval then increase to 1.98 Gbit/sec at the following two time interval and decrease to 1.79 Gbit/sec at the following three time interval until reach The highest value of the Bandwidth which is 2.21 Gbit/sec at the last time interval .

The Delay of ONOS ranged is from 0.099 to 33 ms. the highest value of the delay is 33 ms which happen at the first-time interval because put the ping request on hold to send out an ARP broadcast to learn the MAC address of the remote device, then wait for a response, and then send the first ping through. This delay is usually too long, then the delay decrees as the time went by until it reaches the lowest value 0.099 at the last time interval; on the other hand, the Delay of floodlight ranged was from 0.099 to 27.5ms. The highest value of the Delay is 27.5 ms which happen at the first-time interval as described then the delay decrees as the time went by until it reaches the lowest value 0.099 at the last time interval.

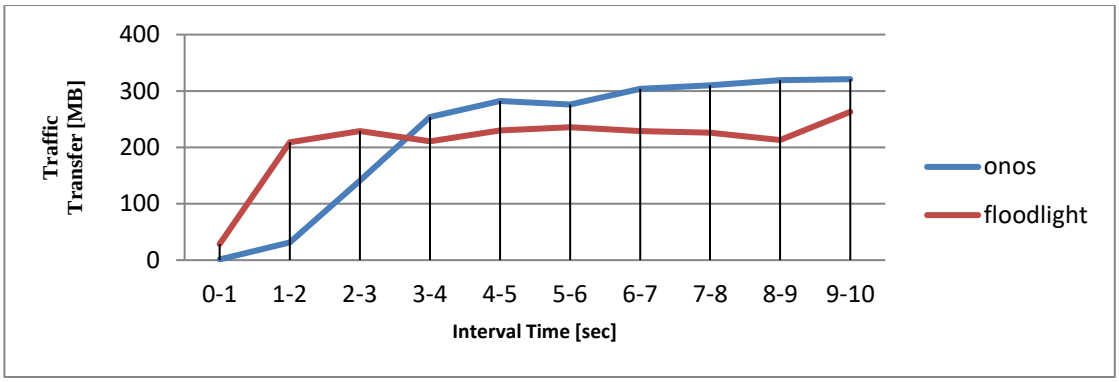

Figure 4-7 Traffic Throughput of Single Topology

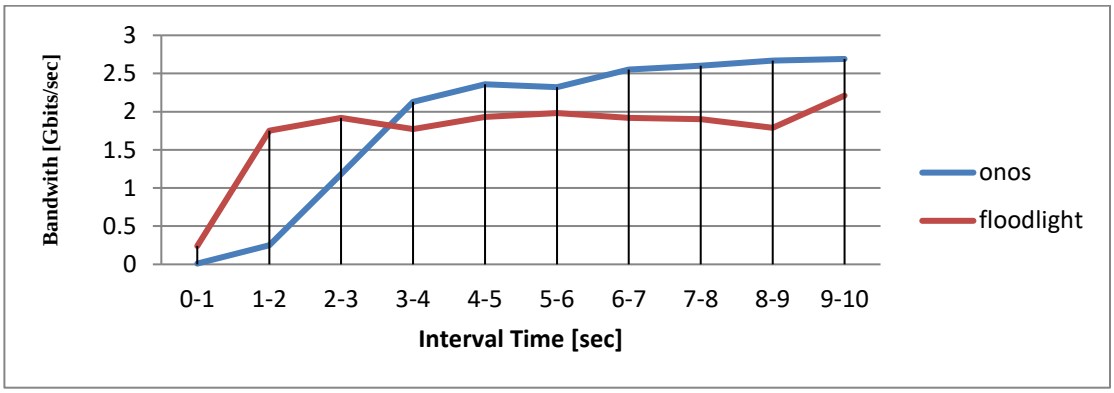

Figure 4-8 Bandwidth of Single Topology

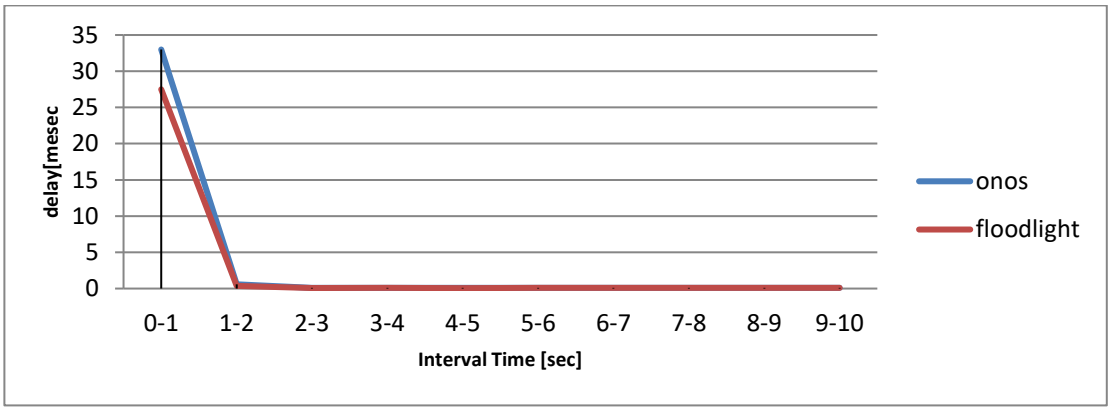

Figure 4-9 Delay of Single Topology

4.2. Linear Topology

A linear topology was created as showed in figure 4-10 using onos controller and figure 4-13 using floodligt controller and Iperf was used to measure the throughput, bandwidth as showed in figure 4-11 and Figure

4-14 and ping used to measure delay as showed in figure 4-12 and Figure 4-15:

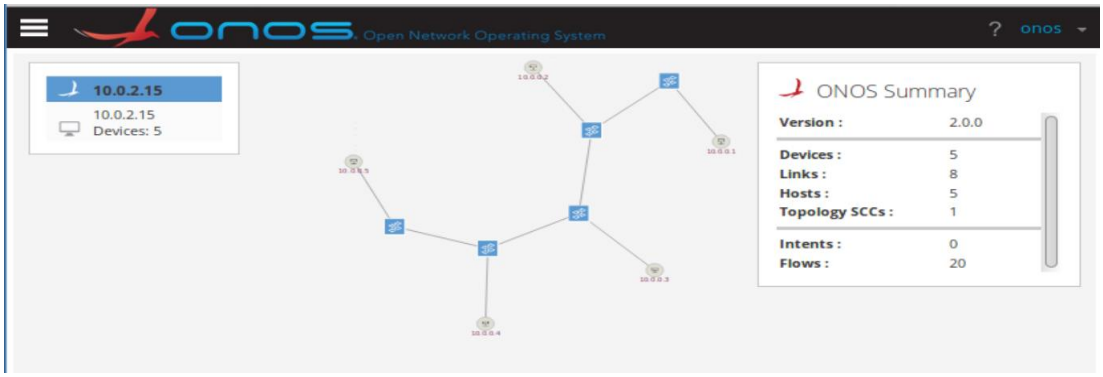

Figure 4-10 Linear Topology ONOS GUI

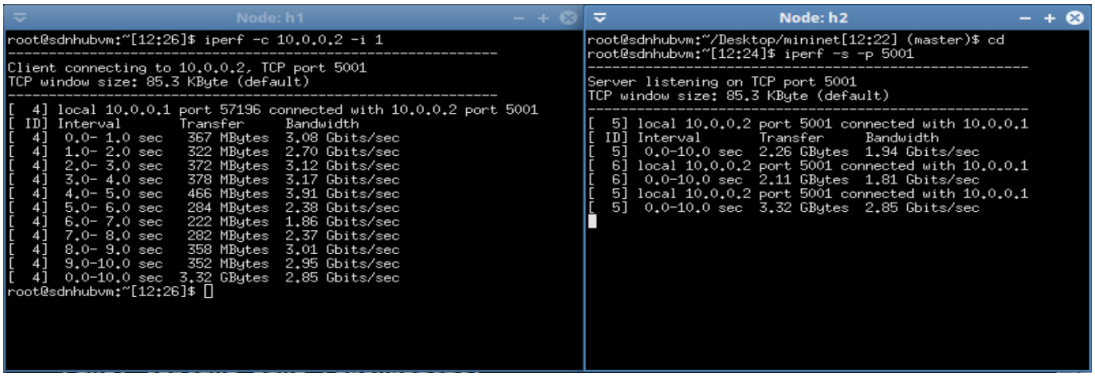

Figure 4-11 throughput and bandwidth in onos based linear topology

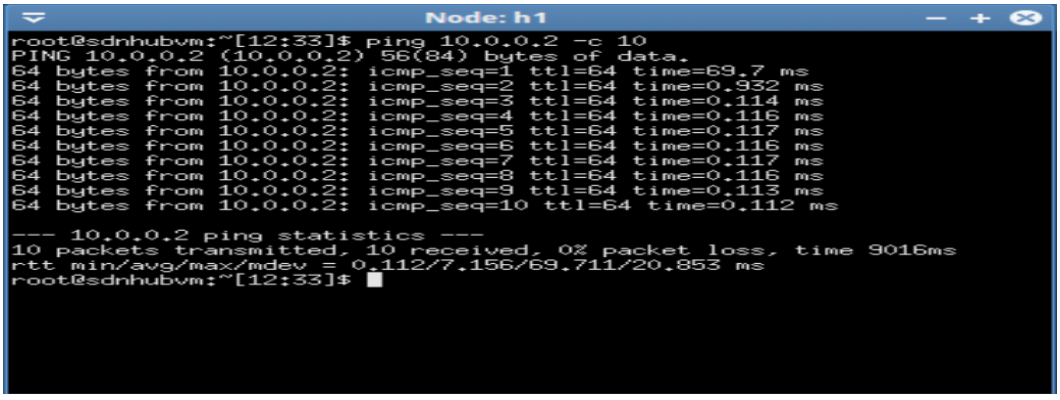

Figure 4-12 delay in onos based linear topology

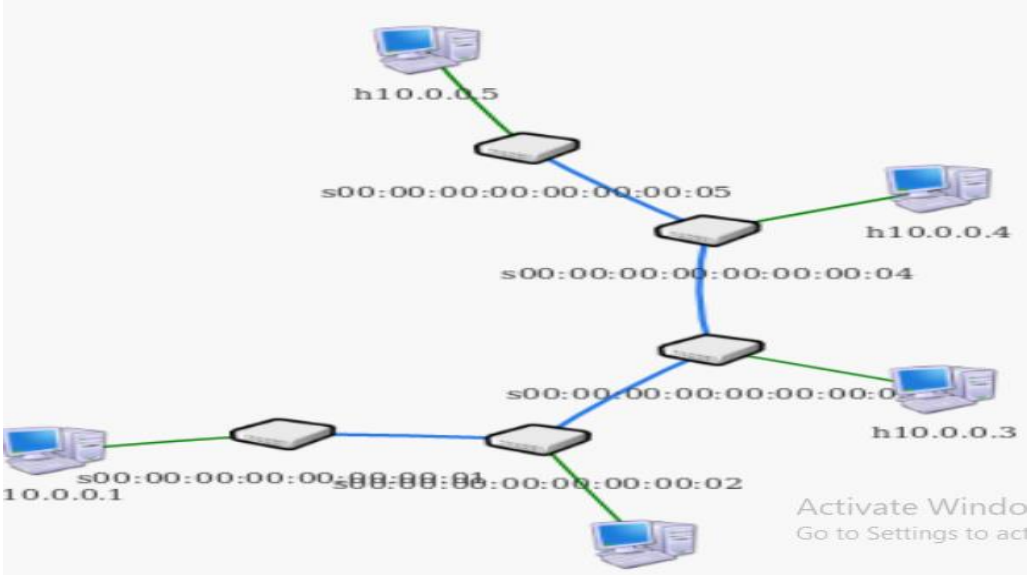

Figure 4-13 Linear Topology Floodlight GUI

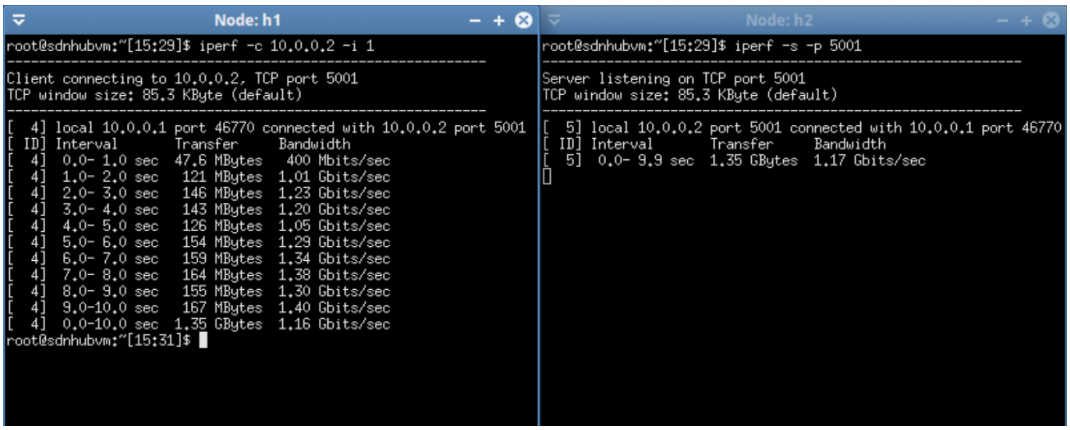

Figure 4-14 throughput and bandwidth in Floodlight based linear topology

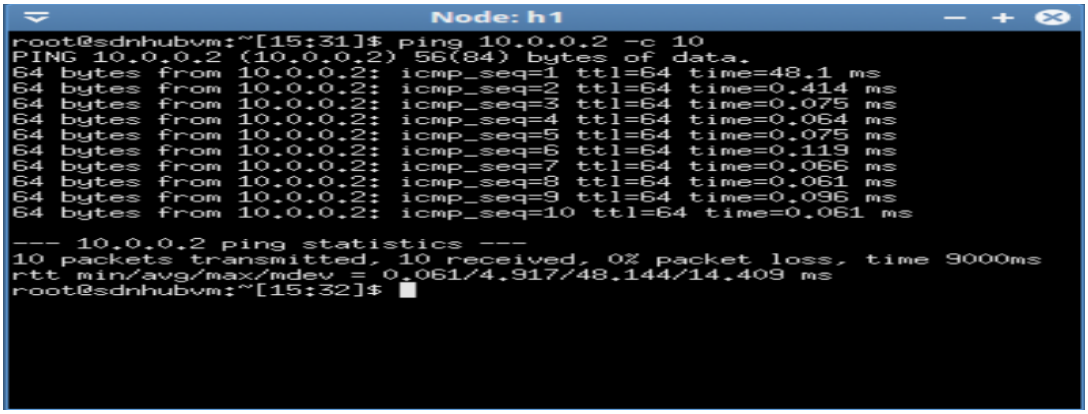

Figure 4-15 delay in Floodlight based linear topology

figure 4-16, figure 4-17 and figure 4-18 display the results of linear topology based on throughput, bandwidth and delay respectively.

figure 4-16 display the results of the throughput of ONOS which show inconsistency in increasing and decreasing through all the time interval it ranged from 222 which happened at time interval number seven to 466 MB which happened at time interval number five, On the other hand the Throughput of floodlight show increasing from the lowest value 47.6 to 146 Mbytes at the first three time interval then decrease to 126 Mbytes after two time interval and increasing to the highest value of the Throughput which is 167 MB that happened at the last interval.

figure 4-17 display the results of the bandwidth of ONOS which show inconsistency in increasing and decreasing through all the time interval it ranged from 1.86 Gbit/sec which happened at time interval number seven to 3.91Gbit/sec which happened at time interval number five, on the other hand the Bandwidth of floodlight show increasing from the lowest value 0.39 to 1.23 91Gbit/sec at the first three time interval then decrease to 1.05 91Gbit/sec after two time interval and increasing to the highest value of the bandwidth which is 1.4 Gbit/sec that happened at the last interval.

figure 4-18 display the results of the the controllers delay in ONOS it ranged from the highest value which is 69.7 ms that's happen at the first time interval as described then the delay decrees as the time went by until it reach the lowest value 0.112 ms at the last time interval, On the other hand the Delay of floodlight ranged from the highest value which is 48.1 ms ms that's happen at the first time interval as described then the delay decrees as the time went by until it reach the lowest value 0.061 ms at the last time .

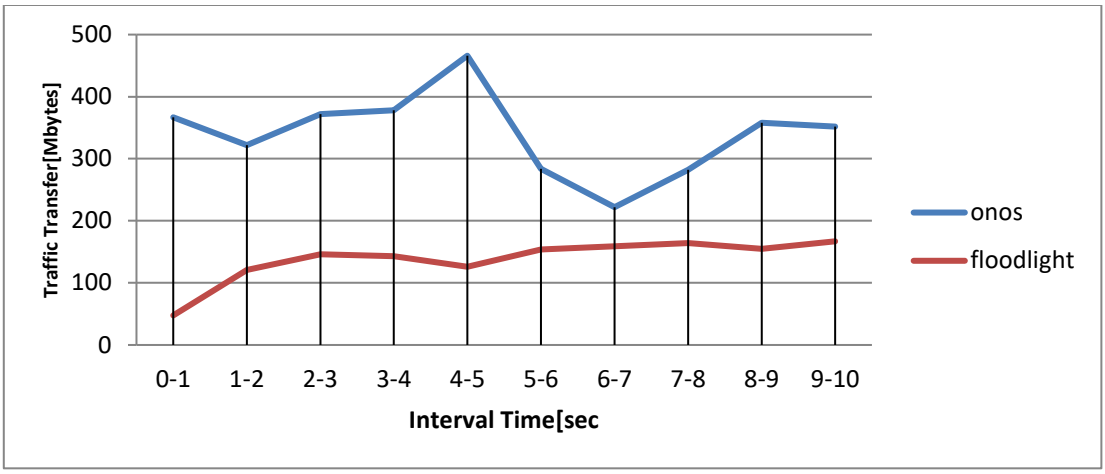

Figure 4-16 Traffic Throughput of Linear Topology

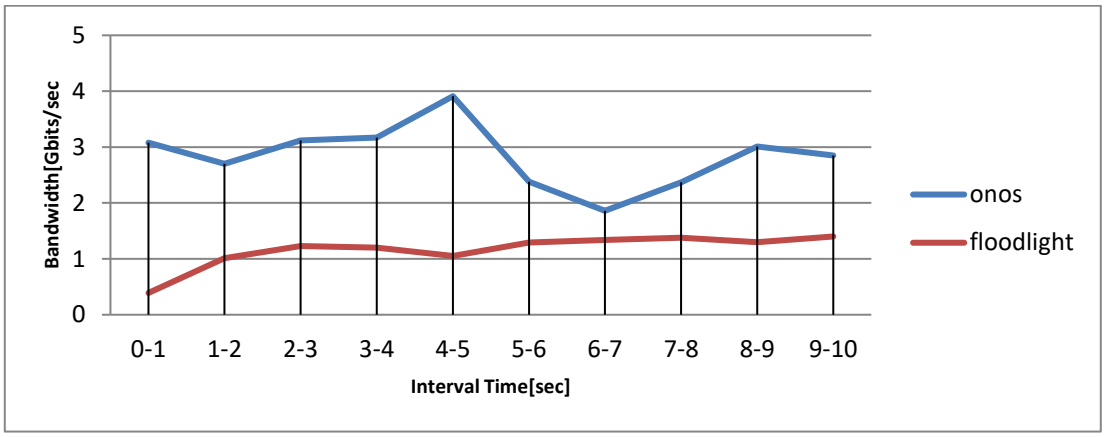

Figure 4-17 Bandwidth of Linear Topology

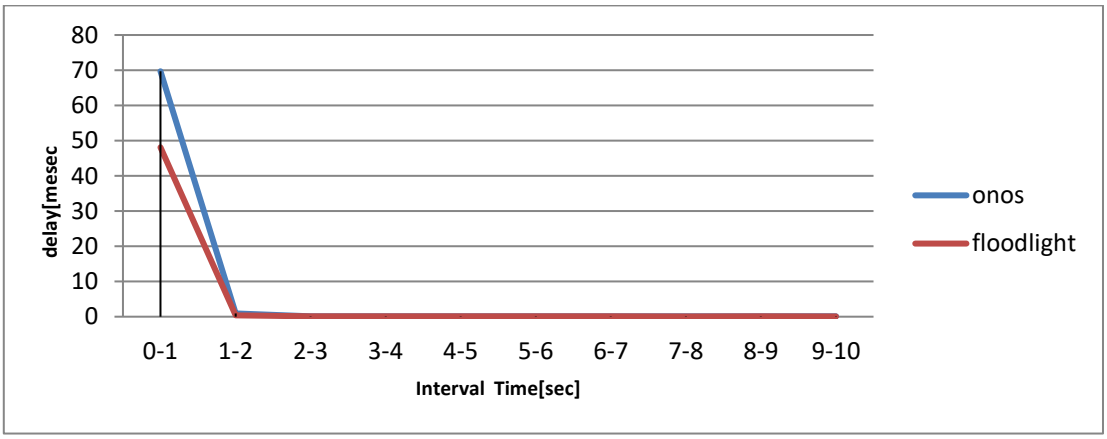

Figure 4-18 Delay of Linear Topology

4.3. Tree Topology

A tree topology was created as showed in figure 4-19 using onos controller

and figure 4-22 using floodligt controller and Iperf was used to measure the throughput, bandwidth as showed in figure 4-20 and figure 4-23 and ping used to measure delay as showed in figure 4-21 and figure 4-24:

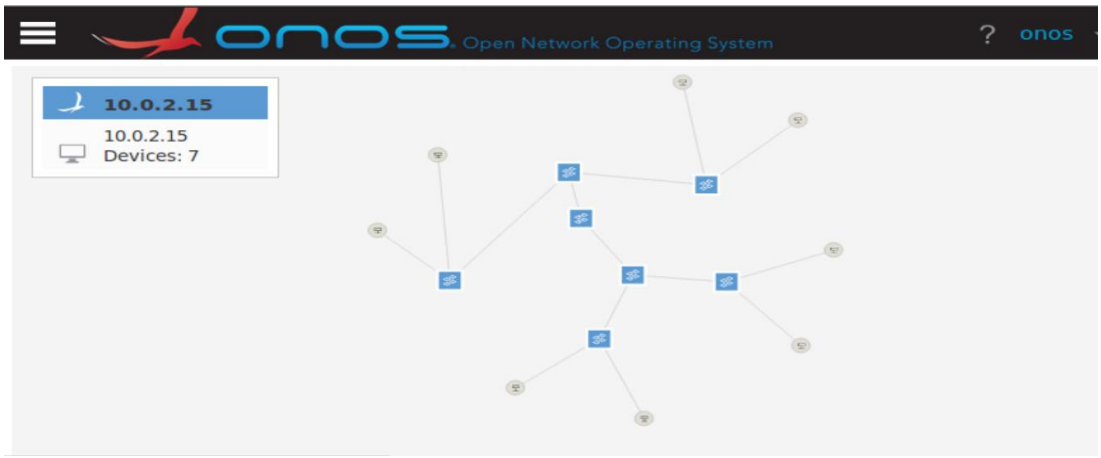

Figure 4-19 Tree Topology ONOS GUI

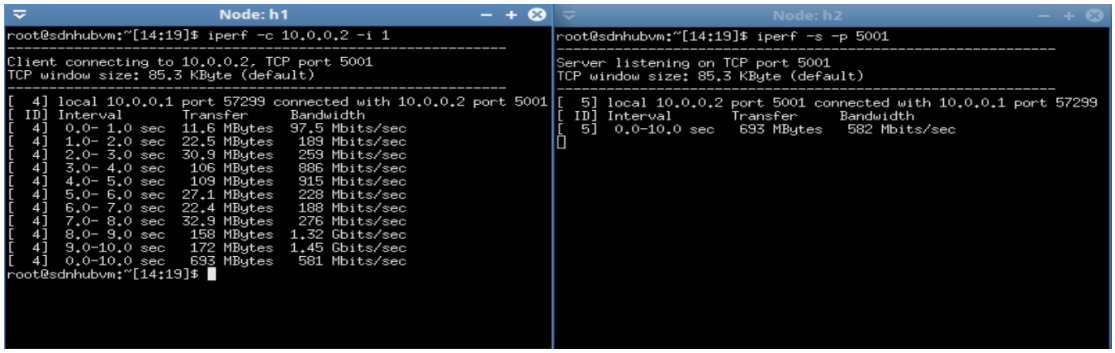

Figure 4-20 throughput and bandwidth in ONOS based Tree topology

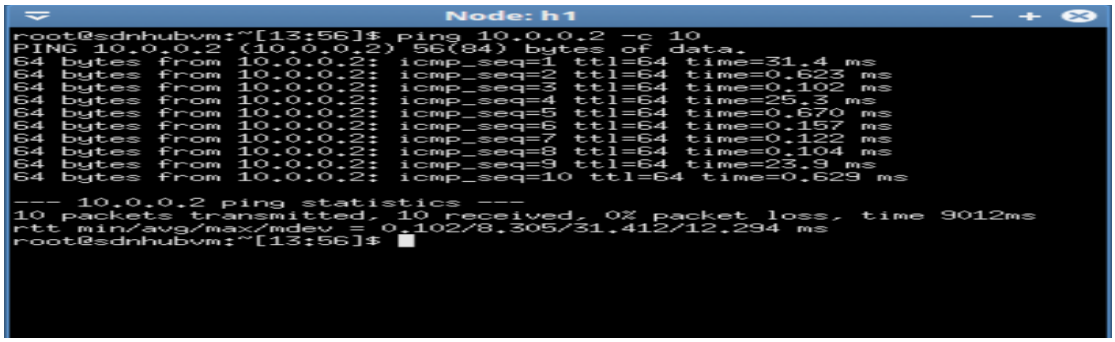

Figure 4-21 delay in ONOS based tree topology

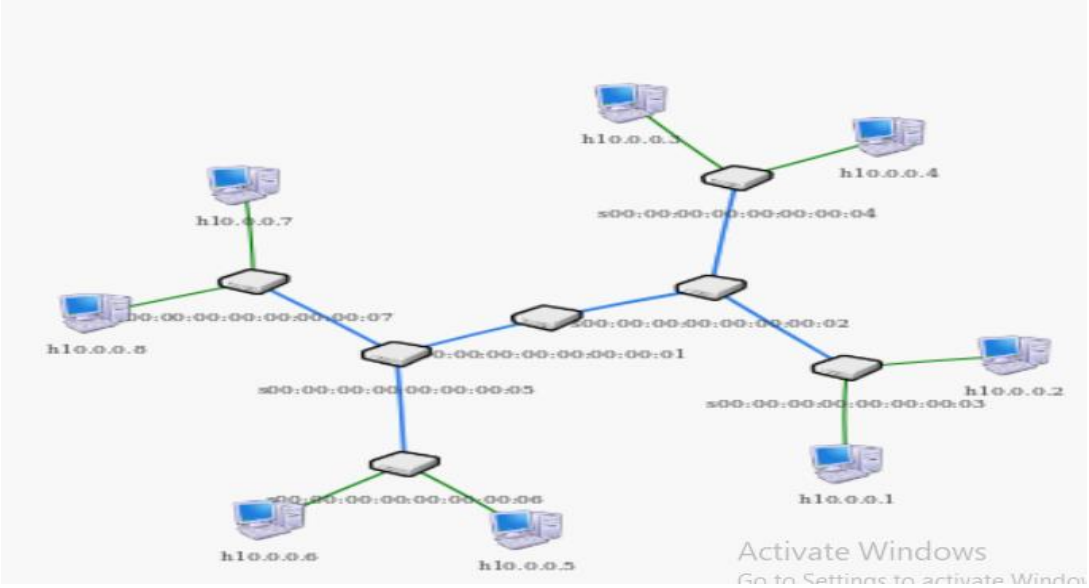

Figure 4-22 Tree Topology Floodlight GUI

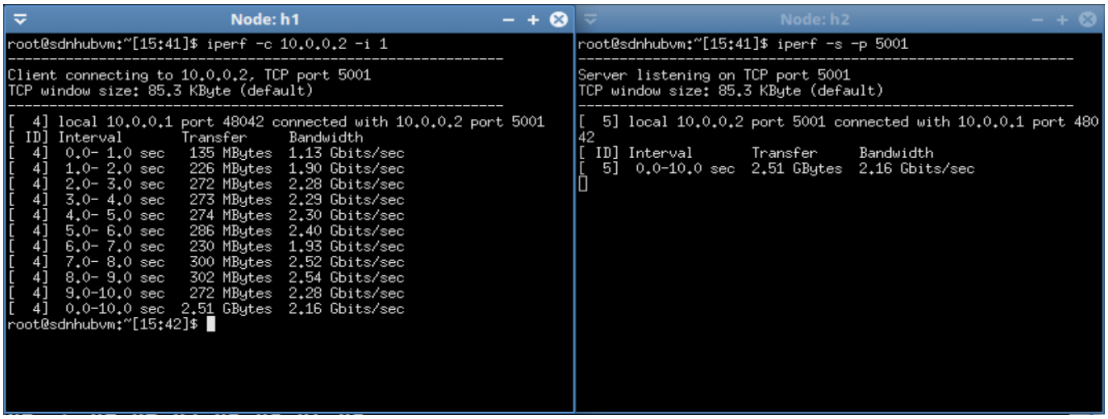

Figure 4-23 throughput and bandwidth in Floodlight based Tree topology

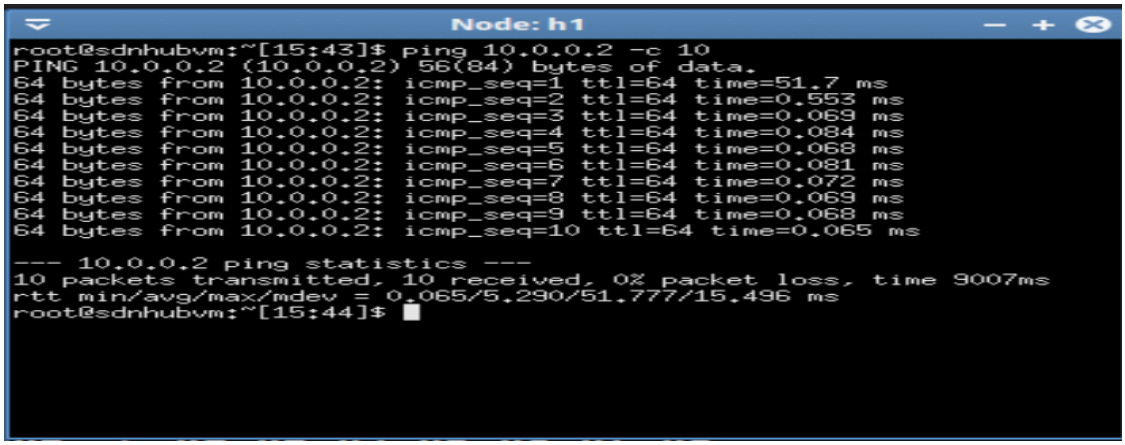

Figure 4-24 delay in Floodlight based tree topology

Figure 4-25, figure 4-26 and figure 4-27 illustrate the outcome of tree topology based on throughput, bandwidth and delay respectively.

Figure 4-25 illustrate the Throughput of ONOS which show increasing from the lowest value 11.6 to 109 Mbytes at the first five time interval then decrease to 32.9 Mbytes after three time interval and increasing to the highest value of the Throughput which is 172 MB that happened at the last time interval; on the other hand the Throughput of floodlight increasing from the lowest value 135 to 286 Mbytes at the first five time interval then decrease to 230 Mbytes for the next time interval and increasing to the highest value of the Throughput which is 302 MB that happened at time interval number nine then decrease to 272 anged is from 135 to 300 Mbyte. The highest value of the Throughput is 300Mbyte as in figure 4-26.

Figure 4-26 present the Bandwidth of ONOS which started at 97.5 Gbits/sec and increase dramatically to 915 Gbits/sec at the first five seconds then slightly decrease to 256 Gbits/sec at the next three time interval then continue to increase until reach the highest Bandwidth value which is 1485Gbit/sec at the last time interval; On the other hand the Bandwidth of floodlight started at 1.13 Gbits/sec and increase dramatically to 2.4 Gbits/sec at the first six time interval then slightly decrease to 1.93 Gbits/sec at the next time interval then continue to increase to 2.54 Gbits/sec at the next two time interval then slightly decrease to 2.28 Gbits/sec at the last time interval.

figure 4-27 display the results of the controllers delay in ONOS it showed inconsistency in increasing and decreasing through all the time interval it decrease from 31.4 ms which happened at the first time interval to 0.102 at the time interval number three and increase to 25.3 for the next interval

then decrease to 0.104 at the time interval number eight then increase to 23.9 for the next interval then decrease to 0.629 at the last time interval, On the other hand the Delay of floodlight ranged from the highest value which is 51.7 ms that's happen at the first time interval as described then the delay decrees as the time went by until it reach the lowest value 0.065 ms at the last time interval .

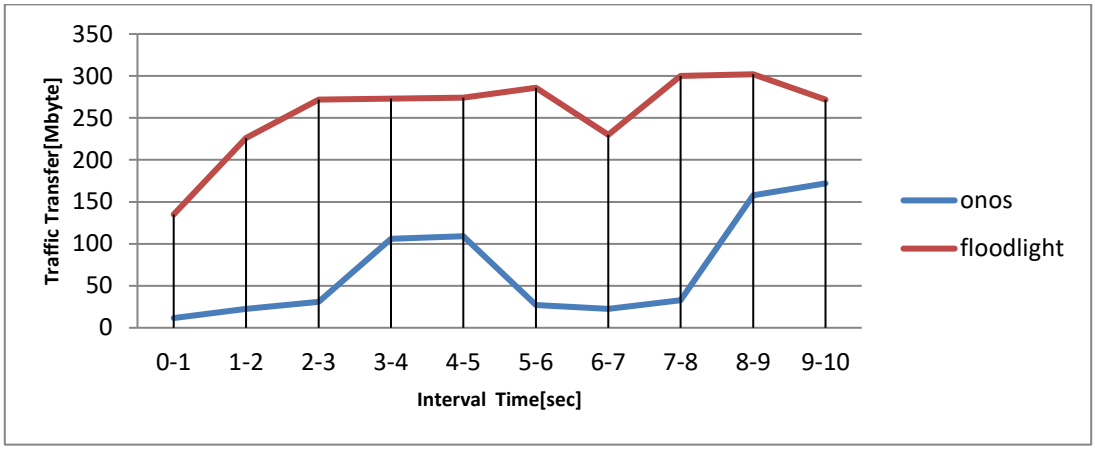

Figure 4-25 Traffic Throughput of Tree Topology

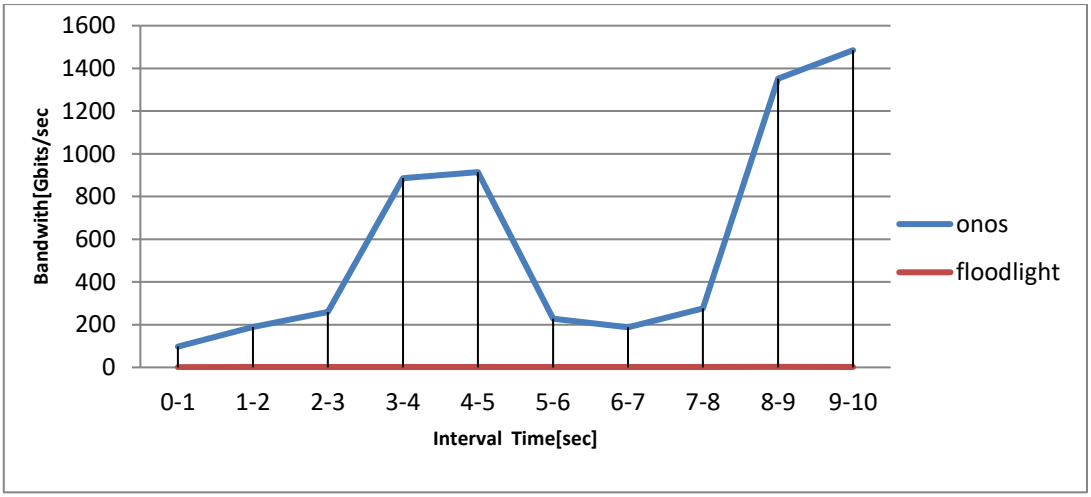

Figure 4-26 Bandwidth of Tree Topology

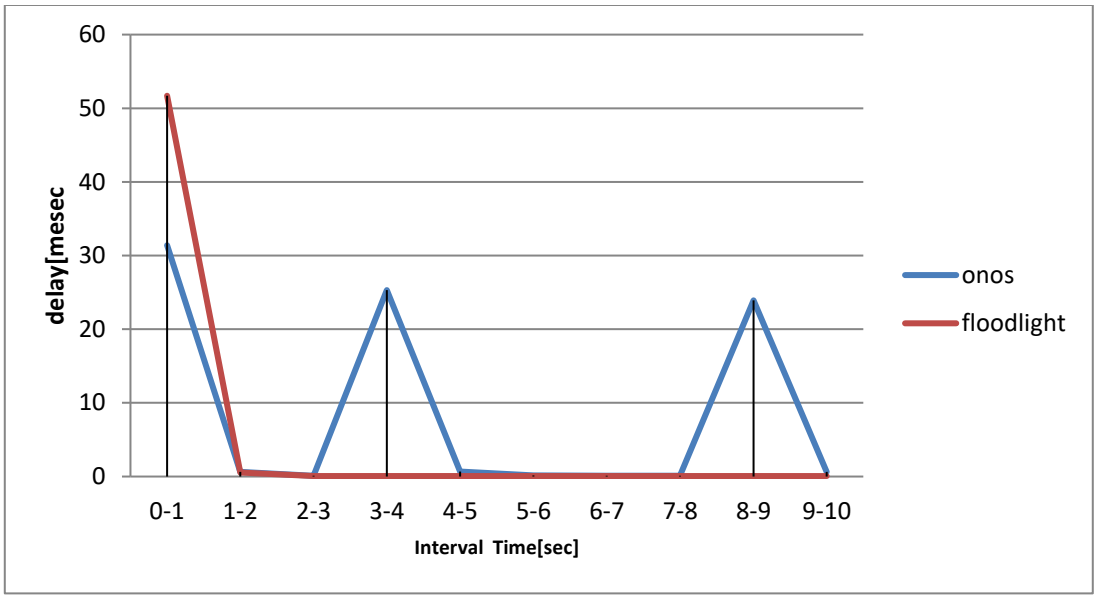

Figure 4-27 Delay of Tree Topology

4.4. IoT topology

An IoT topology was created as showed in figure 4-28 using ONOS controller and Iperf was used to measure the throughput and bandwidth as showed in figure 4-29 and figure 4-31 and ping used to measure delay as showed in figure 4-30 and figure 4-32:

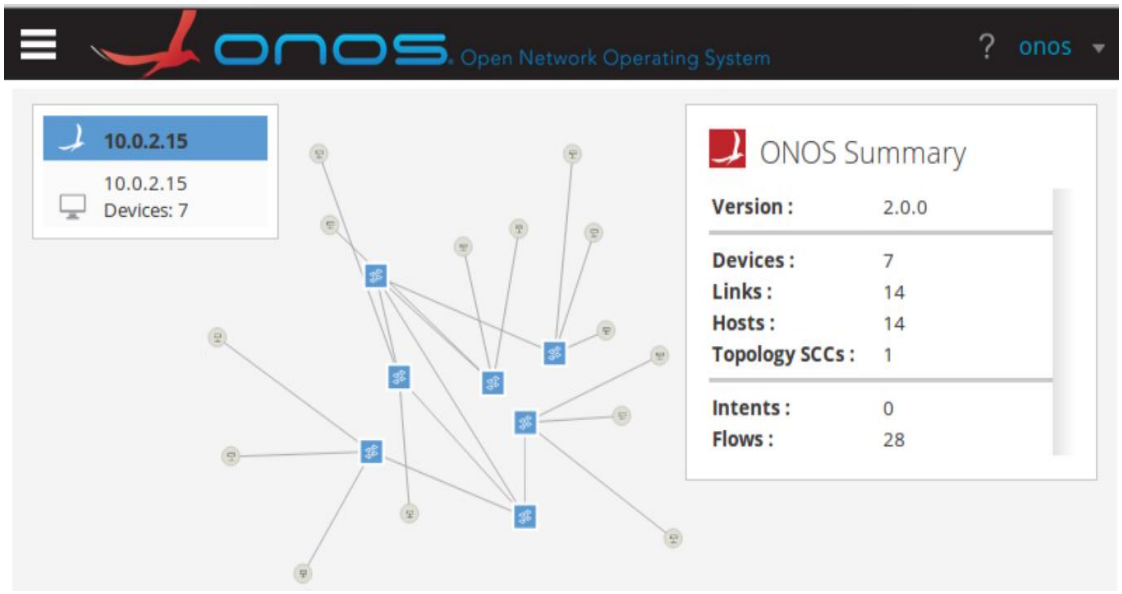

Figure 4-28 SDN-based iot Topology ONOS GUI

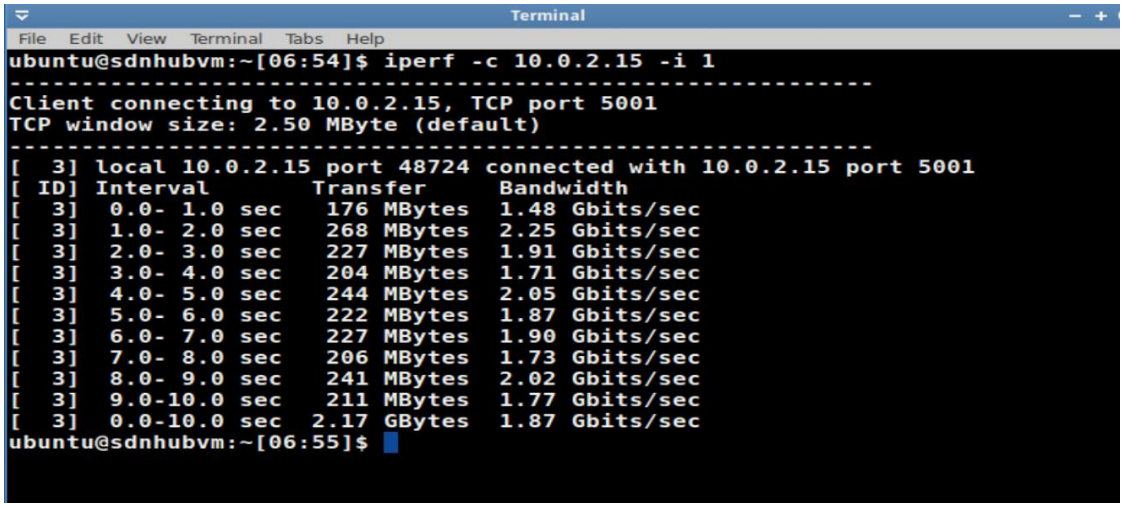

Figure 4-29 throughput and bandwidth in ONOS- based IoT topology

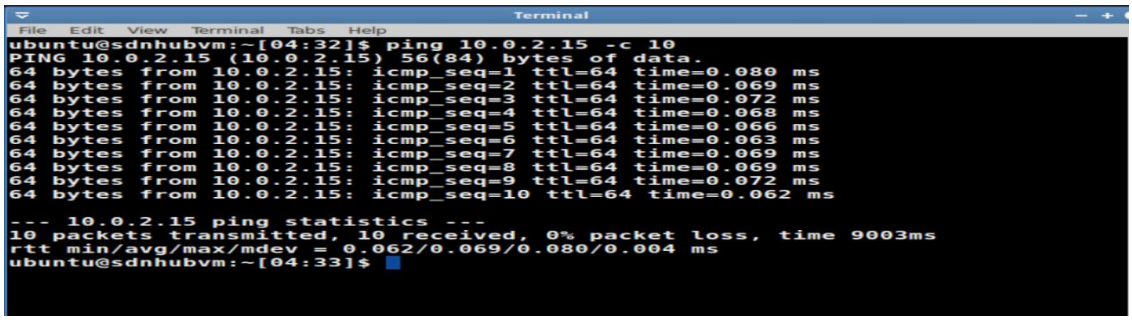

Figure 4-30 Delay in ONOS- based IoT topology

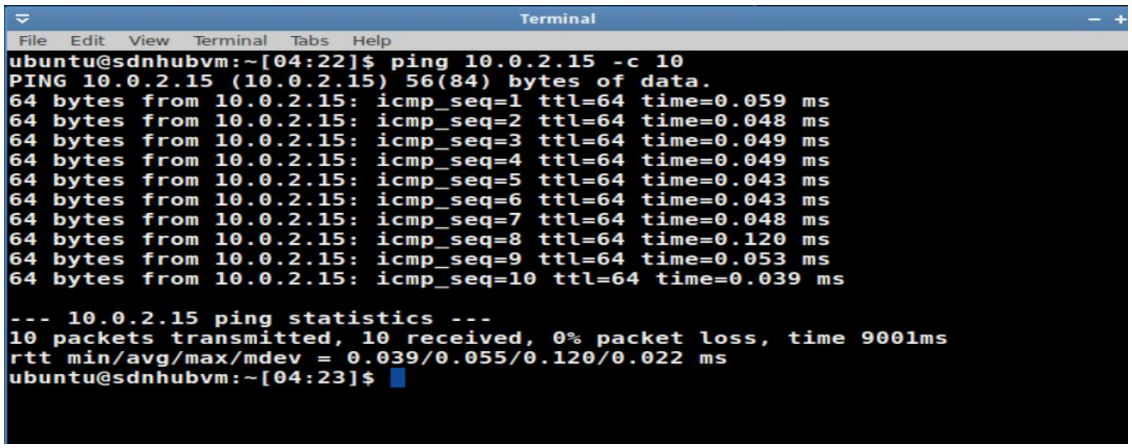

Figure 4-31 throughput and bandwidth in floodlight- based IoT topology

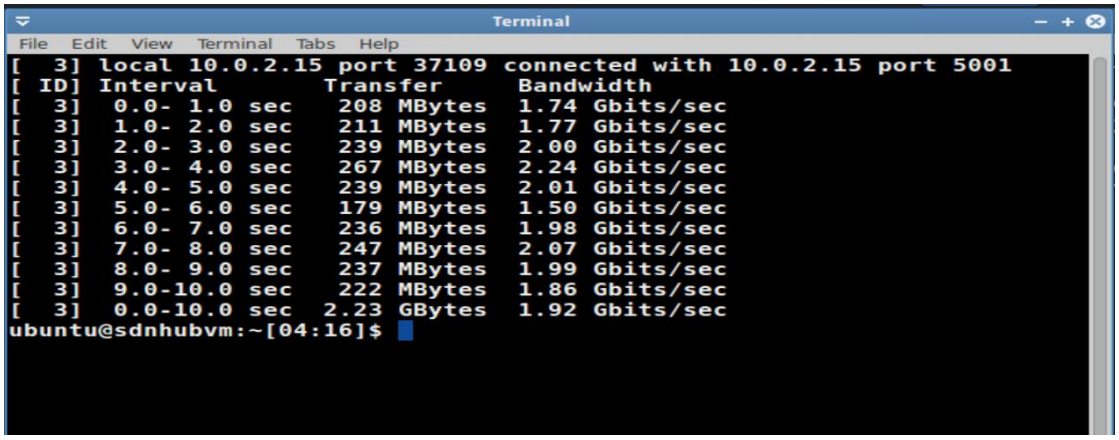

Figure 4-32 Delay in floodlight- based IoT topology

Figure 4-33, figure 4-34 and figure 4-35 illustrate the outcome of IoT topology based on throughput, bandwidth and delay respectively.

Figure 4-33 illustrate the Throughput of ONOS which show increasing from the lowest value 176 to 268 Mbytes at the first time interval then decrease and increasing as the time interval goes until the last time interval which shows throughput of 211 Mbytes; on the other hand the Throughput of floodlight increasing from the lowest value 179 to 267 Mbytes at the first four time interval then decrease and increasing as the time interval goes until the last time interval which shows throughput of 222 Mbytes.

Figure 4-34 present the bandwidth of ONOS which started at 1.48 Gbits/sec and increase to2.25 Gbits/sec which is the highest value then decrease and increasing as the time interval goes until the last time interval which shows bandwidth of 1.77; On the other hand the bandwidth of floodlight started at 1.74 Gbits/sec increasing to 2.24 Gbits/secwhich is the highest value at the first four time interval then decrease and increasing as the time interval goes until the last time interval which shows 1.86 Gbits/sec at the last time interval.

Figure 4-35 display the results of the controllers delay in ONOS it showed

inconsistency in increasing and decreasing through all the time interval it decrease from 0.08 ms which happened at the first time interval to 0.062 at the last time interval, On the other hand the Delay of floodlight ranged from the highest value which is 0.059 ms that's happen at the first time interval as described then the delay showed inconsistency in increasing and decreasing through all the time interval and reach 0.039ms delay at the last time interval .

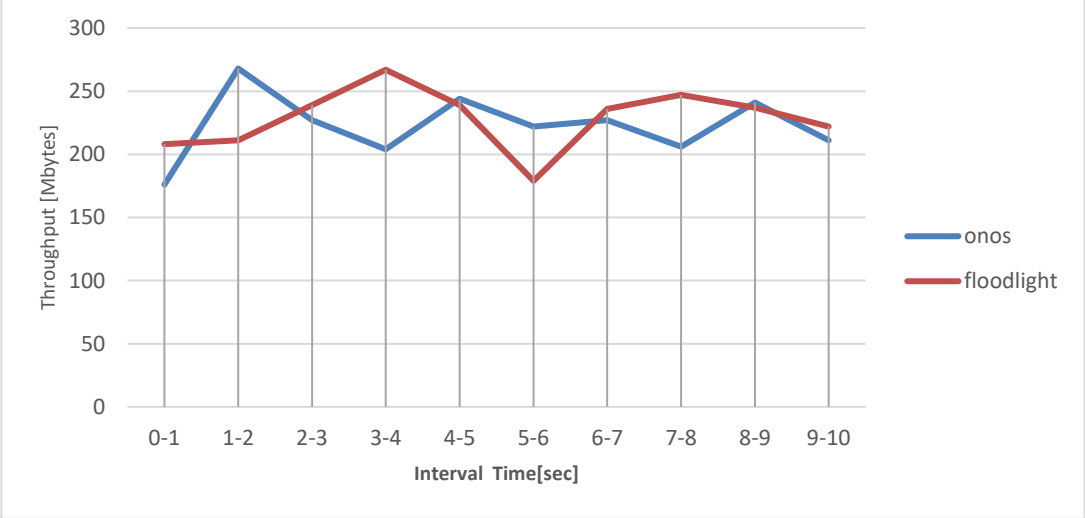

Figure 4-33Traffic Throughput of IoT Topology

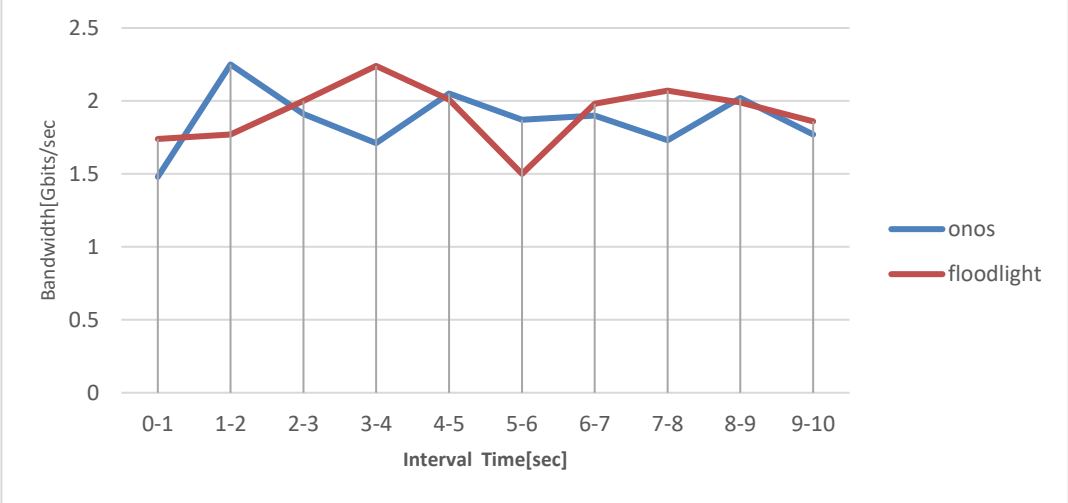

Figure 4-34 Bandwidth of IoT Topology

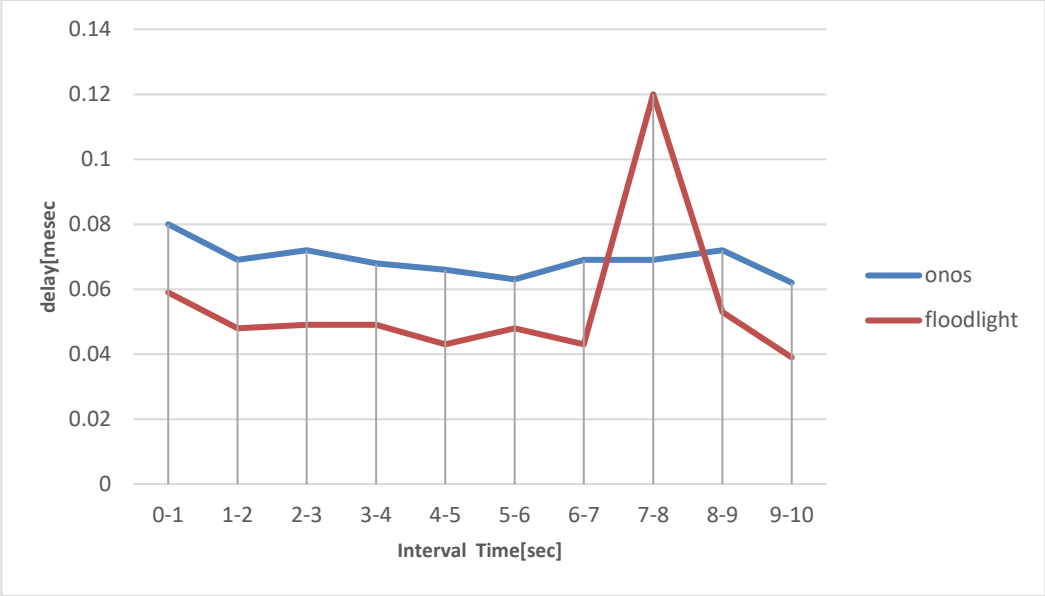

Figure 4-35 Delay of IoT Topology

Chapter 5

5. Conclusion and Recommendations

5.1. Conclusion

SDN seems to be the promising technology thanks to its flexibility, programmability and the global view given by its centralized controller, in the first part of this Thesis tow SDN controllers are evaluated especially in terms of throughput, bandwidth and delay and since some of the IoT applications require high band width due to the big data generated. ONOS show the highest bandwidth in all the evaluated topologies which are 1.876, 2.85 and 587.55 Gbit/sec comparing to floodlight which are 1.741, 1.4 and 2.157 Gbit/sec in single, linear and tree topologies respectively and show its feasibility to be used in the SDN-based IoT topology and show average throughput of 222.6 MB and average bandwidth of 1.869 Gbit/sec and average delay of 0.069 ms.

5.2. Recommendations

Internet of Things is a new revolution of the Internet and it is a key research topic for researcher in embedded, computer science & information technology area due to its very diverse area of application& heterogeneous mixture of various communications and embedded technology in its architecture

In the future a special attention should be put to studying performances of SDN-based architecture for IoT in a concrete smart scenario and with IoT devices such as Raspberry Pi and sensors. In addition, the behavior of different SDN controllers within a smart IoT environment.

Reference

1. Gubbi, J., et al., *Internet of Things (IoT): A vision, architectural elements, and future directions.* Future generation computer systems, 2013. **29**(7): p. 1645-1660.
2. Al-Fuqaha, A., et al., *Internet of things: A survey on enabling technologies, protocols, and applications.* IEEE communications surveys & tutorials, 2015. **17**(4): p. 2347-2376.
3. Domingo, M.C., *An overview of the Internet of Things for people with disabilities.* Journal of Network and Computer Applications, 2012. **35**(2): p. 584-596.
4. Zanella, A., et al., *Internet of things for smart cities.* IEEE Internet of Things journal, 2014. **1**(1): p. 22-32.
5. Qin, Z., et al. *A software defined networking architecture for the internet-of-things*. in *2014 IEEE network operations and management symposium (NOMS)*. 2014. IEEE.
6. Evans, D., *The internet of things how the next evolution of the internet is changing everything (april 2011).* White Paper by Cisco Internet Business Solutions Group (IBSG), 2012.
7. Atzori, L., A. Iera, and G. Morabito, *The internet of things: A survey.* Computer networks, 2010. **54**(15): p. 2787-2805.
8. Giusto, D., et al., *The internet of things: 20th Tyrrhenian workshop on digital communications*. 2010: Springer Science & Business Media.
9. Tayyaba, S.K., et al., *Software-defined networks (SDNs) and Internet of Things (IoTs): A qualitative prediction for 2020.* network, 2016. **7**(11).
10. Lara, A., A. Kolasani, and B. Ramamurthy, *Network innovation using openflow: A survey.* IEEE communications surveys & tutorials, 2013. **16**(1): p. 493-512.
11. Cabaj, K., et al. *SDN Architecture Impact on Network Security*. in *FedCSIS (Position Papers)*. 2014.
12. Kreutz, D., et al., *Software-defined networking: A comprehensive survey.* Proceedings of the IEEE, 2014. **103**(1): p. 14-76.
13. Da Xu, L., W. He, and S. Li, *Internet of things in industries: A survey.* IEEE Transactions on industrial informatics, 2014. **10**(4): p. 2233-2243.
14. Perera, C., et al., *A survey on internet of things from industrial*

market perspective. IEEE Access, 2014. **2**: p. 1660-1679.
15. Yang, Z., et al. *Study and application on the architecture and key technologies for IOT*. in *2011 International Conference on Multimedia Technology*. 2011. IEEE.
16. Blial, O., M. Ben Mamoun, and R. Benaini, *An overview on SDN architectures with multiple controllers.* Journal of Computer Networks and Communications, 2016. **2016**.
17. Bonomi, F., et al. *Fog computing and its role in the internet of things*. in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 2012.
18. McKeown, N., et al., *OpenFlow: enabling innovation in campus networks.* ACM SIGCOMM Computer Communication Review, 2008. **38**(2): p. 69-74.
19. Othman, W.M., et al. *Implementation and performance analysis of SDN firewall on POX controller*. in *2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*. 2017. IEEE.
20. Salman, O., et al. *SDN controllers: A comparative study*. in *2016 18th Mediterranean Electrotechnical Conference (MELECON)*. 2016. IEEE.
21. Taher, A., *Testing of floodlight controller with mininet in sdn topology.* ScienceRise, 2014(5 (2)): p. 68-73.
22. Khattak, Z.K., M. Awais, and A. Iqbal. *Performance evaluation of OpenDaylight SDN controller*. in *2014 20th IEEE international conference on parallel and distributed systems (ICPADS)*. 2014. IEEE.
23. Morita, K., I. Yamahata, and V. Linux. *Ryu: Network operating system*. in *OpenStack Design Summit & Conference*. 2012.
24. Nguyen, X.N., D. Saucez, and T. Turletti, *Providing CCN functionalities over OpenFlow switches.* 2013.
25. Rowshanrad, S., et al., *A survey on SDN, the future of networking.* Journal of Advanced Computer Science & Technology, 2014. **3**(2): p. 232-248.
26. Kulmala, M., *Improving network security with software-defined networking*, 2016.
27. Berte, D.-R. *Defining the iot*. in *Proceedings of the International Conference on Business Excellence*. 2018. Sciendo.
28. Khan, R., et al. *Future internet: the internet of things architecture, possible applications and key challenges*. in *2012 10th international conference on frontiers of information technology*. 2012. IEEE.

29. Bhatt, S., F. Patwa, and R. Sandhu. *An access control framework for cloud-enabled wearable Internet of Things*. in *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*. 2017. IEEE.
30. Tan, L. and N. Wang. *Future internet: The internet of things*. in *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*. 2010. IEEE.
31. Wu, M., et al. *Research on the architecture of Internet of Things*. in *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*. 2010. IEEE.
32. Uckelmann, D., M. Harrison, and F. Michahelles, *Architecting the internet of things*. 2011: Springer Science & Business Media.
33. Mun, D.-H., M. Le Dinh, and Y.-W. Kwon. *An assessment of internet of things protocols for resource-constrained applications*. in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*. 2016. IEEE.
34. Shelby, Z., K. Hartke, and C. Bormann, *The constrained application protocol (CoAP).* 2014.
35. Fette, I. and A. Melnikov, *The websocket protocol*, 2011, RFC 6455, December.
36. Kidd, C.D., et al. *The aware home: A living laboratory for ubiquitous computing research*. in *International Workshop on Cooperative Buildings*. 1999. Springer.
37. Bainbridge, S., C. Steinberg, and M. Furnas. *GBROOS—an ocean observing system for the Great Barrier Reef*. in *International Coral Reef Symposium*. 2010.
38. Zhang, M., T. Yu, and G.F. Zhai. *Smart transport system based on "The Internet of Things"*. in *Applied mechanics and materials*. 2011. Trans Tech Publ.
39. Vilamovska, A., et al., *Rfid application in healthcare–scoping and identifying areas for rfid deployment in healthcare delivery.* RAND Europe, February, 2009: p. 26.
40. Wu, D., et al. *UbiFlow: Mobility management in urban-scale software defined IoT*. in *2015 IEEE conference on computer communications (INFOCOM)*. 2015. IEEE.
41. Jararweh, Y., et al., *SDIoT: a software defined based internet of things framework.* Journal of Ambient Intelligence and Humanized Computing, 2015. **6**(4): p. 453-461.
42. Huang, H., J. Zhu, and L. Zhang, *An SDN_based management framework for IoT devices.* 2014.

43. Castellani, A.P., et al. *Architecture and protocols for the internet of things: A case study*. in *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. 2010. IEEE.

6. Appendices

Appendix A

Create SDN-based IoT topology

```
"""Custom topology example
Two directly connected switches plus a host for each switch:
  host --- switch --- switch --- host
Adding the 'topos' dict with a key/value pair to generate our newly
defined
topology enables one to pass in '--topo=mytopo' from the command line.
"""
from mininet.topo import Topo
class MyTopo( Topo ):
   "Simple topology example."
   def __init__( self ):
      "Create custom topo."
      # Initialize topology
      Topo.__init__( self )
      # Add hosts and switches
      edgeServer = self.addHost( 'h1' )
        internet = self.addHost( 'h2' )
      Host1 = self.addHost( 'h3' )
      Host2 = self.addHost( 'h4' )
      Host3 = self.addHost( 'h5' )
      Host4 = self.addHost( 'h6' )
      Host5 = self.addHost( 'h7' )
      Host6 = self.addHost( 'h8' )
```

```
Host7 = self.addHost( 'h9' )
Host8 = self.addHost( 'h10' )
Host9 = self.addHost( 'h11' )
Host10 = self.addHost( 'h12' )
Host11 = self.addHost( 'h13' )
Host12 = self.addHost( 'h14' )
coreSwitch = self.addSwitch( 's1' )
disSwitch1 = self.addSwitch( 's2' )
disSwitch2 = self.addSwitch( 's3' )
accSwitch1 = self.addSwitch( 's4' )
accSwitch2 = self.addSwitch( 's5' )
accSwitch3 = self.addSwitch( 's6' )
accSwitch4 = self.addSwitch( 's7' )
# Add links
self.addLink( edgeServer, coreSwitch)
self.addLink( internet, coreSwitch)
self.addLink( coreSwitch, disSwitch1 )
self.addLink( coreSwitch, disSwitch2 )
self.addLink( disSwitch1, disSwitch2 )
self.addLink( disSwitch1, accSwitch1 )
self.addLink( disSwitch1, accSwitch2 )
self.addLink( disSwitch2, accSwitch3 )
self.addLink( disSwitch2, accSwitch4 )
self.addLink( accSwitch1, Host1 )
self.addLink( accSwitch1, Host2 )
self.addLink( accSwitch1, Host3 )
self.addLink( accSwitch2, Host4 )
```

```
        self.addLink( accSwitch2, Host5 )
        self.addLink( accSwitch2, Host6 )
        self.addLink( accSwitch3, Host7 )
        self.addLink( accSwitch3, Host8 )
        self.addLink( accSwitch3, Host9 )
        self.addLink( accSwitch4, Host10 )
        self.addLink( accSwitch4, Host11 )
        self.addLink( accSwitch4, Host12 )
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Appendix B

Create simple sensors

```
#! python3.4
#Simple Light or door type Sensor that can receive control Information to change state
##sensor uses loop and standard reconnect
import paho.mqtt.client as mqtt
#import testclient as mqtt
import json
import os
import time
import logging,random,os
import sys,getopt
#from mqtt_functions import *
options=dict()
brokers=["192.168.1.206","192.168.1.157","192.168.1.204","192.168.1.185","test.mosquitto.org",\
        "broker.hivemq.com","iot.eclipse.org"]
options["broker"]=brokers[1]
options["port"]=1883
options["verbose"]=False
options["username"]=""
options["password"]=""
options["cname"]=""
options["sensor_type"]="light"
```

```
options["topic_base"]="sensors"
options["interval"]=10 #loop time when sensor publishes in verbose
options["interval_pub"]=300 # in non chatty mode publish
# status at this interval if 0 then ignore
options["keepalive"]=120
options["loglevel"]=logging.ERROR
cname=""
QOS0=0
mqttclient_log=False
username=""
password=""
chatty=False
interval=2 #loop time when sensor publishes
sensor_pub_interval=300# how often to publish if status is unchanged
##
def command_input(options):
    topics_in=[]
    qos_in=[]
    valid_options="  -h <broker> -b <broker> -p <port>-t <topic> -q QOS
-v -h <help>\
 -d logging debug  -n Client ID or Name -i loop Interval\
-s <set states to open and closed> -u Username -P Password --h <help>"
    print_options_flag=False
    try:
      opts, args = getopt.getopt(sys.argv[1:],"h:b:i:dk:p:t:q:l:vsn:r:u:P:")
    except getopt.GetoptError:
      print (sys.argv[0],valid_options)
```

```
  sys.exit(2)
qos=0
for opt, arg in opts:
    if opt == '-h':
        options["broker"] = str(arg)
    elif opt == "-b":
        options["broker"] = str(arg)
    elif opt == "-i":
        options["interval"] = int(arg)
    elif opt == "-k":
        options["keepalive"] = int(arg)
    elif opt=="-r":
       options["topic_base"]=str(arg)
    elif opt =="-p":
       options["port"] = int(arg)
    elif opt =="-t":
       topics_in.append(arg)
    elif opt =="-q":
        qos_in.append(int(arg))
    elif opt =="-n":
        options["cname"]=arg
    elif opt =="-d":
       options["loglevel"]=logging.DEBUG
    elif opt =="-v":
       options["verbose"]=True
    elif opt =="-s":
       options["sensor_type"]="door"
```

```
        elif opt == "-P":
            options["password"] = str(arg)
        elif opt == "-u":
            options["username"] = str(arg)
    lqos=len(qos_in)
    for i in range(len(topics_in)):
        if lqos >i:
            topics_in[i]=(topics_in[i],int(qos_in[i]))
        else:
            topics_in[i]=(topics_in[i],0)
    if topics_in:
        options["topics"]=topics_in
#######
##callback all others defined in mqtt-functions.py
def on_message(client,userdata, msg):
    topic=msg.topic
    m_decode=str(msg.payload.decode("utf-8","ignore"))
    logging.debug("Message Received "+m_decode)
    message_handler(client,m_decode,topic)
def message_handler(client,msg,topic):
    if topic==topic_control: #got control message
        print("control message ",msg)
        update_status(client,msg)
def on_connect(client, userdata, flags, rc):
    logging.debug("Connected flags"+str(flags)+"result code "\
    +str(rc)+"client1_id")
    if rc==0:
```

```
            client.connected_flag=True
            client.publish(connected_topic,1,retain=True)
            #publish connection status
            client.subscribe(options["topics"])
        else:
            client.bad_connection_flag=True
    def on_disconnect(client, userdata, rc):
        logging.debug("disconnecting reason  " + str(rc))
        client.connected_flag=False
        client.disconnect_flag=True
        client.subscribe_flag=False
    #######
    def update_status(client,status):
        status=status.upper()
        if status==states[0] or status==states[1]: #Valid status
            client.sensor_status=status #update
            print("updating status",client.sensor_status)
    def publish_status(client):
        global start_flag #used to publish on start
        pubflag=False
        if start_flag:
            start_flag=False
            pubflag=True
        if time.time()-client.last_pub_time >=options["interval_pub"]:
            pubflag=True
        if time.time()-client.last_pub_time >=options["interval"] and chatty:
            pubflag=True
```

```
    logging.debug("old "+str(client.sensor_status_old))
    logging.debug("new "+ str(client.sensor_status))
    if client.sensor_status_old!=client.sensor_status or pubflag:
        client.publish(sensor_status_topic,client.sensor_status,0,True)
        print("publish on",sensor_status_topic,\
              " message  ",client.sensor_status)
        client.last_pub_time=time.time()
        client.sensor_status_old=client.sensor_status
def Initialise_client_object():
    mqtt.Client.last_pub_time=time.time()
    mqtt.Client.topic_ack=[]
    mqtt.Client.run_flag=True
    mqtt.Client.subscribe_flag=False
    mqtt.Client.sensor_status=states[1]
    mqtt.Client.sensor_status_old=None
    mqtt.Client.bad_connection_flag=False
    mqtt.Client.connected_flag=False
    mqtt.Client.disconnect_flag=False
    mqtt.Client.disconnect_time=0.0
    mqtt.Client.disconnect_flagset=False
    mqtt.Client.pub_msg_count=0
def Initialise_clients(cname):
    #flags set
    client= mqtt.Client(cname)
    if mqttclient_log: #enable mqqt client logging
        client.on_log=on_log
    client.on_connect= on_connect        #attach function to callback
```

```
    client.on_message=on_message        #attach function to callback
    client.on_disconnect=on_disconnect
    #client.on_subscribe=on_subscribe
    #client.on_publish=on_publish
    return client
def Connect(client,broker,port,keepalive,run_forever=False):
    """Attempts connection set delay to >1 to keep trying
    but at longer intervals  """
    connflag=False
    delay=5
    #print("connecting ",client)
    badcount=0 # counter for bad connection attempts
    while not connflag:
        logging.info("connecting to broker "+str(broker))
        print("connecting to broker "+str(broker)+":"+str(port))
        print("Attempts ",badcount)
        try:
            res=client.connect(broker,port,keepalive)      #connect to broker
            if res==0:
                connflag=True
                return 0
            else:
                logging.debug("connection failed ",res)
                badcount +=1
                if badcount>=3 and not run_forever:
                    return -1
                    raise SystemExit #give up
```

```
            elif run_forever and badcount<3:
                delay=5
            else:
                delay=30
        except:
            client.badconnection_flag=True
            logging.debug("connection failed")
            badcount +=1
            if badcount>=3 and not run_forever:
                return -1
                raise SystemExit #give up
            elif run_forever and badcount<3:
                delay=5*badcount
            elif delay<300:
                delay=30*badcount
        time.sleep(delay)
    return 0
def
wait_for(client,msgType,period=.25,wait_time=40,running_loop=False):
    #running loop is true when using loop_start or loop_forever
    client.running_loop=running_loop #
    wcount=0
    while True:
        logging.info("waiting"+ msgType)
        if msgType=="CONNACK":
            if client.on_connect:
                if client.connected_flag:
```

```
                return True
            if client.bad_connection_flag: #
                return False
        if not client.running_loop:
            client.loop(.01)  #check for messages manually
        time.sleep(period)
        #print("loop flag ",client.running_loop)
        wcount+=1
        if wcount>wait_time:
            print("return from wait loop taken too long")
            return False
################
#####
if __name__ == "__main__" and len(sys.argv)>=2:
    command_input(options)
chatty=options["verbose"]
logging.basicConfig(level=options["loglevel"]) #error logging
#use DEBUG,INFO,WARNING,ERROR
if not options["cname"]:
    r=random.randrange(1,10000)
    r=3542
    cname="sensor-"+str(r)
else:
    cname=str(options["cname"])
##May want to change topics
connected_topic=options["topic_base"]+"/connected/"+cname
sensor_status_topic=options["topic_base"]+"/"+cname
```

```
topic_control=sensor_status_topic+"/control"
#########
options["topics"]=[(topic_control,0)]
#print(options["topics"])
if not options["verbose"]:
   print("only sending changes")
if options["sensor_type"]=="light":
   states=["ON","OFF"] #possible sensor states
else:
   states=["OPEN","CLOSED"] #possible sensor states
Initialise_client_object() # add extra flags
logging.info("creating client"+cname)
client=Initialise_clients(cname)#create and initialise client object
if options["username"] !="":
   client.username_pw_set(options["username"],options["password"])
client.will_set(connected_topic,0, qos=0, retain=True) #set will
print("starting")
print("Publishing on ",sensor_status_topic)
print("send control to ",topic_control)
print("Sensors States are ",states)
start_flag=True #used to always publish when starting
run_flag=True
#connecting_flag=False
bad_conn_count=0
try:
   while run_flag:
      client.loop(0.05)
```

```
        if not client.connected_flag:
            if Connect(client,options["broker"],options["port"],\
                    options["keepalive"],run_forever=True) !=-1:
                if not wait_for(client,"CONNACK"):
                  run_flag=False #break
            else:
                run_flag=False #break
        #subbscribes to control in on_connect calback
        if client.connected_flag:
            publish_status(client)
except KeyboardInterrupt:
    print("interrrupted by keyboard")
if client.connected_flag:
    client.publish(connected_topic,0,retain=True)
    time.sleep(1)
    client.disconnect()
```