SDN IMPROVEMENTS AND SOLUTIONS FOR TRADITIONAL NETWORKS

MOHAMMED AMER YAHYA

SEPTEMBER 2017

SDN IMPROVEMENTS AND SOLUTIONS FOR TRADITIONAL NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES OF
ÇANKAYA UNIVERSITY

BY
MOHAMMED AMER YAHYA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING DEPARTMENT

SEPTEMBER 2017

Title of the Thesis: **SDN IMPROVEMENTS AND SOLUTIONS FOR TRADITIONAL NETWORKS.**

Submitted by **Mohammed Yahya**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.

[illegible]

Prof. Dr. Can ÇOĞUN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Erdoğan DOĞDU
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

[illegible]	[illegible]
Assist. Prof. Dr. Kasim ÖZTOPRAK Co-supervisor	Assist. Prof. Dr. Reza HASSANPOUR Supervisor

Examination Date: **14.09.2017**
Examining Committee Members:

Assist. Prof. Dr. Reza HASSANPOUR	(Çankaya Univ.)	[illegible]
Assist. Prof. Dr. Sibel TARIYAN ÖZYER	(Çankaya Univ.)	[illegible]
Assist. Prof. Dr. Shadi AL SHEHABI	(THK Univ.)	[illegible]

STATEMENT OF NON-PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: **Mohammed Yahya**

Signature: [illegible]

Date: 14.09.2017

ABSTRACT

SDN IMPROVEMENTS AND SOLUTIONS FOR TRADITIONAL NETWORKS

YAHYA, Mohammed Amer
M.Sc., Computer Engineering Department
Supervisor: Assist. Prof. Dr. Reza HASSANPOUR
Co-supervisor: Assist. Prof. Dr. Kasim ÖZTOPRAK

September 2017, 86 pages

The new trend of the internet has led to building a digital world, where everything is connected and from any place can be accessible. Nevertheless, the management of conventional networks is complicated and difficult. Because of its static characteristics, it is hard to configure and the reconfiguration causes errors, unwanted changes, and extra load. Another reason which makes the matters harder is that the traditional networks are also perpendicularly integrated which means the control and the data planes are embedded in the same device. Software defined networking (SDN) is a new trend of networking which guarantees to alter the behavior of the current network through decoupling the control plane from the data plane and gives the networks centralized management, and offers the capability to program the network. SDN makes it simpler to produce and present new concepts in networking, simplifies network control, and facilitates network development. In this thesis, we introduce the SDN improvements and solutions for traditional networks, and explain the current networks technologies and their features. Subsequently, we present the limitations of the current networks in managements, routing, devices dependency, capital and operational cost, and rigidness and complexity. We also give the proposed solutions from SDN to these limitations, cloud computing, and heterogeneous networks. We also look at SDN applications in network routing, security and access control, Internet research, mobile device offloading, and wireless virtual machines. In addition, we introduce one of the newest trends of SDN regarding traffic engineering which is Intent-based networking. Finally, we demonstrate through simulations the benefits which are provided by SDN to traditional networks. We select a single IP layer and a multilayer optical network (MPLS over DWDM) to demonstrate the advantages of SDN managements. We deployed an SDN controller that can illustrate how the test environment configuration is carried out and how beneficial it can be. The SDN controller used was the ONOS controller.

ÖZ

SDN IMPROVEMENTS AND SOLUTIONS FOR TRADITIONAL NETWORKS

YAHYA, Mohammed Amer
Yüksek Lisans, Bilgisayar Mühendisliği Anabilim Dalı
Tez Yöneticisi: Yrd. Doç. Dr. Reza HASSANPOUR
Ortak Tez Yönetiçisi: Yrd. Doç. Dr. Kasim ÖZTOPRAK

Eylül 2017, 86 sayfa

İnternetin yeni akımı, her şeyin bağlı olduğu ve her yerden erişilebilen dijital bir dünyanın inşasına yol açtı. Bununla birlikte, geleneksel ağların yönetimi daha karmaşıklaştı ve zorlaştı. Statik özelliklerinden dolayı, interneti yapılandırmak zordur ve yeniden yapılandırma hatalarına, istenmeyen değişikliklere ve aşırı yüklemeye neden olur. Konuyu zorlaştıran başka bir sebep de, geleneksel ağlar da dikey olarak entegre edilmiş durumdadır, kontrol ve veri düzlemi aynı cihaza yerleştiriliyor demektir. Yazılım tanımlı ağ (SDN), kontrol düzlemini veri düzleminden ayırarak mevcut ağın davranışını değiştirmeyi garanti eden ve ağlara merkezi yönetim ve ağ programlama olanağı sunan yeni bir trenddir. SDN, ağda yeni kavramlar üretmeyi ve sunmayı daha kolay hale getirir, ağ denetimini basitleştirir ve ağ geliştirmeyi kolaylaştırır. Bu tezde, geleneksel ağlar için SDN geliştirmeleri ve çözümleri tanıtıldı ve mevcut ağ teknolojileri ve bunların özellikleri açıklandı. Ardından, yönetimlerde, yönlendirmede, cihaz bağımlılığında, sermayede ve işletme maliyetinde, sertlik ve karmaşıklık gibi konularda mevcut ağların sınırlamalarını sunduk. Bunun yanında, SDN'den bu kısıtlamalara, bulut bilişim ve heterojen ağlar ile çözümler önerdik. Ayrıca, ağ yönlendirme, güvenlik ve erişim kontrolü, internet araştırması, mobil cihaz ayırma ve kablosuz sanal makinelerde SDN uygulamalarına baktık. Buna ilaveten, internet tabanlı bir ağ olan trafik mühendisliği ile ilgili olarak

SDN'in en yeni trendlerinden birisini tanıttık. Son olarak, geleneksel ağlara SDN tarafından sağlanan faydaları simülasyonlar yoluyla gösterdik. SDN yönetimlerinin avantajlarını göstermek için tek bir IP katmanı ve çok katmanlı optik ağ (DWDM yerine MPLS) seçtik. Test ortamının yapılandırılmasının nasıl olacağını ve ne kadar faydalı olabileceğini gösterebilecek bir SDN denetleyicisi kullandık. Kullanılan SDN denetleyicisi ONOS denetleyicisiydi.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1. Introduction

1.1 Overview

Communication and information technology [2] infrastructures are rapidly growing due to a rise in the implementation of cyber-physical systems and devices for the Internet of Things (IoT). Recent studies in networks have shown that the conventional structures are not able to satisfy the increasing requests such that all elements of this structure are implicated in vertically-forming complicated networks which are not easy to manage. Traditional networks only support vendor policies specifications and not offer flexibility or agility for dynamic network environments. Automating legacy networks has become difficult. Nowadays, in order to identify and locate servers and applications, networks are reliant upon using IP addresses. Although this method is applicable for static networks which use an IP address to recognize physical devices, it is very impractical for large virtual networks. In addition, it is very costly and time-consuming to manage very complex environments of that kind using traditional networks. The adamant structure of old networks prevents programmability from satisfying the range of client requirements or preferences and, at times, it forces vendors into implying complicated management systems which are programmable.

In today's world, optical networks are present in every part of our life, even in the urban access areas such as offices and houses. Optical networks have 4 ranking layers, namely backbone, regional, metro and access. They provide simultaneous control and communication; however, this simultaneous process is extremely complex. Old-fashioned management consoles and command lines interfaces are not appropriate for complicated, high-speed networks. Modern optical networks require software based programmable frameworks in order to provide easy management and flexible control.

Thousands of changes are made manually to network components. To perform this operation, large teams of network administrators are employed. Network usage and service demand are increasing rapidly in relation to this. Flexible, scalable, nimble and efficient networks are required by operators.

The main problem with the underlying communication network is their environment and the dynamic kind of network application. This means that there are performance requirements of the transmitted data flows, such as Quality of Service (QoS), an important way to designate this problem by means of Traffic Engineering (TE). Traffic Engineering (TE) is an application or a process of analyzing network status concerning a network system, balancing and predicting transferred data loads through the resources of a network. It is a mechanism used to adjust the traffic routing according to the change in the network situation. Traffic engineering aims to improve network performance, QoS and the experience of the user through the effective use of resources. The research community began working on traffic engineering to address this problem and suggested a new way to enhance network robustness in response to increasing traffic demand. Traffic engineering decreases the service disintegration caused by congestion and failure; e.g., link failure. An important property of any network is fault tolerance. It is to guarantee that if a failure occurs in a network, the requested traffic to the destination can still be delivered. The control management [8] plan and the forwarding data plan are tightly coupled in the conventional network, and the entire network is controlled through many devices, where it cannot achieve traffic management in a fine-grained manner and stretchability and flexibility are difficult to improve. To handle the above defined challenges, software-defined networking (SDN) and virtualization are suggested as solutions.

Software Defined networking (SDN) is a new network architecture to enable invention in a communication network and simplification of network management [5]. SDN is considered to be an independent hardware new age network model where networking devices from any provider may be managed by SDN. SDN separates the network control plane from the data plane. Logicality of control plan centralization and the forwarding plan is simply rendered to excite the instruction from the control plane. Two [2] main components of SDN are controller and switches. The entire network is managed by the controller, and the switches are responsible for delivering data to end points according to decisions from the controller. OpenFlow [6] is a communication protocol in the SDN environment that makes the controller manage and interact with infrastructures devices such as switches and routers. Technically, this is depicted as decoupling the control plane from the data forwarding plan and giving the switching architecture more flexibility that abstracts control to applications.

SDN can provide more than one solution in an integrated fashion for which older systems would require a number of interfaces. In optical networks, some functionalities, such as flexible and adaptive network behavior, secure and sustainable networks, measurement and display of the important parameters in optical networks, good resource allocation between main links and sub links that include clouds and data centers, and near-optimal traffic engineering and management, can be provided by SDN. Cross-layer issues such as testing and debugging can also be managed by SDN. Additionally, scalability of the regional and metro networks is another important requirement and it emerges because of the heterogeneous tributaries in the core. In spite of that, SDN can provide scalable and automated resource partitioning to overcome this issue. When enabling a dynamic and intelligent WSON (Wavelength Switching Optical Network), the control plane plays an essential role by minimizing the operational expense and latency caused by processing.

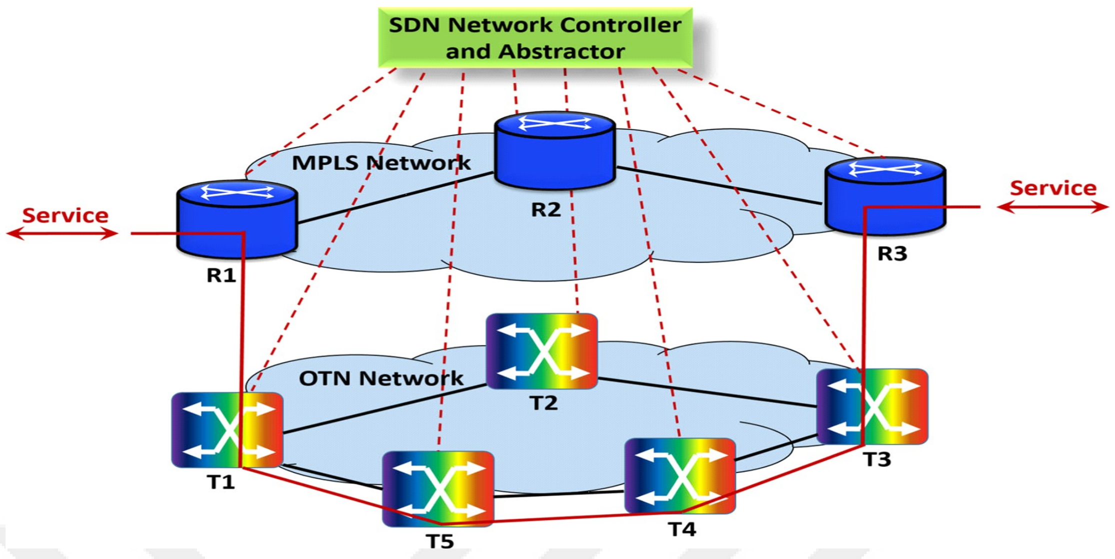


Figure (1) Multilayer Control in SDN

Due to the fact that one OpenFlow includes all useful functions that are integrated, an OpenFlow based control plane provides the best manageability and flexibility for carriers. Moreover, a natural choice for UCP (Uniform Control Plan) (Figure (1)) in IP/DWDM multilayer networks is the OpenFlow based control plane. Although both GMPLS and PCE have high technical maturity for optical networks, the OpenFlow based control plane is at the starting phase.

To overcome the problems of traffic engineering (TE) through the idea of a programmable network produced by SDN, the packet forwarding behavior can be programmed by network operators to enhance the potential of invention of network implementations. Compared with the conventional network architecture, SDN has specific features, as follows:

Concentricity of control: All network information, including dynamic changes of network states, network topology and overall application requirements, are saved by the SDN controller.

Programmability: The data forwarding plan devices can be programed in a dynamic manner to make the allotment of network assets more efficient.

Openness: The SDN controller communicates with the forwarding devices in a unified interface.

The features of SDN are beneficial for resolving the present tasks of network traffic engineering.

1.2 The Contributions of this Thesis:

This thesis is focused on improvements and solutions provided by SDN. The contributions of this thesis are as follows:

- Explain current network technologies.
- SDN architecture and OpenFlow.
- Explain the limitations of traditional networks in management, routing, device dependency, Capital & Operational costs, Rigidity and Complexity.
- SDN improvements and solutions for traditional networks limitations, Cloud Computing and Heterogeneous networks.
- Network applications of SDN.
- The presentation of a debate on open issues and advices which are required to detect the potential of SDN.

1.3 Thesis-Organization:

This thesis contains seven chapters. The second chapter presents an Overview of Traditional Networks, SDN and OpenFlow. Chapter 3 discusses previous studies to perform SDN over traditional networks. In Chapter 4, we explain the limitations of traditional networks. In Chapter 5, which is the core of the thesis, we survey the improvements and solutions for traditional networks, the applications of SDN and intent. Chapter 6 explains the simulation and performance of SDN over traditional networks. Finally, in Chapter 7, we present the Challenges and Conclusions.

CHAPTER 2

OVERVIEW OF TRADITIONAL NETWORKS, SDN AND OPENFLOW

2.1 Traditional Networks

2.1.1 Introduction

In today's world, optical networks are present in every part of our life, even in urban access areas such as offices and houses. Optical networks have four ranking layers, namely backbone, regional, metro and access. A traditional optical domain has a transmitter in telecommunications with which optical techniques are utilized. Wave modulation provides forwarding digital or analog signals in the range of gigabits per second on a transporter of very high frequency, exactly 186 THz to 196 THz. As a matter of fact, several transport waves that are propagating are used to increase the bit rate with less interaction in the same fiber. Apparently, every frequency represents a variety of wavelengths. This wavelength technique is called wavelength division multiplexing (WDM).

2.1.2 Wavelength Division Multiplexing (WDM)

Through a single optical fiber, WDM technology simultaneously transmits multiple optical channels at different wavelengths over a broad wavelength band. It is beneficial to make full use of the low-loss properties of optical fibers. According to published reports in 1970, this concept was first performed. However, in 1977 the essential research on WDM technology actually began.

The goal of WDM is to send signals simultaneously at multiple wavelengths to increase the transition capacity of an optical fiber to the width of an optical channel and the regulation of bandwidth of the transmitter source and the highest data rate possible for optical channels. For optical data, multiple types of multi/demultiplexers have been introduced, which are key devices in WDM transmission systems.

WDM systems contain two basic development methods. The first method is the dense WDM (DWDM). Traditional DWDM systems have a channel spacing of 50 GHz or 100 GHz (corresponding to 0.4 m or 0.8 m at wavelengths near 1.5 μm) (Figure (2)), and high bit rates can be provided by utilized optical channels, particularly, 2.5 or 10 Gbit/s, with the probability of working on 40 Gbit/ s technologies. Instead, ultra DWDM systems, in particular optical channels, assume more closely spaced channels at lower data rates. The second method is coarse WDM (CWDM). Complete key features of network structures (scalability, transparency and low cost) are ensured by CWDM systems, particularly to create a metro/access network [55] [56].

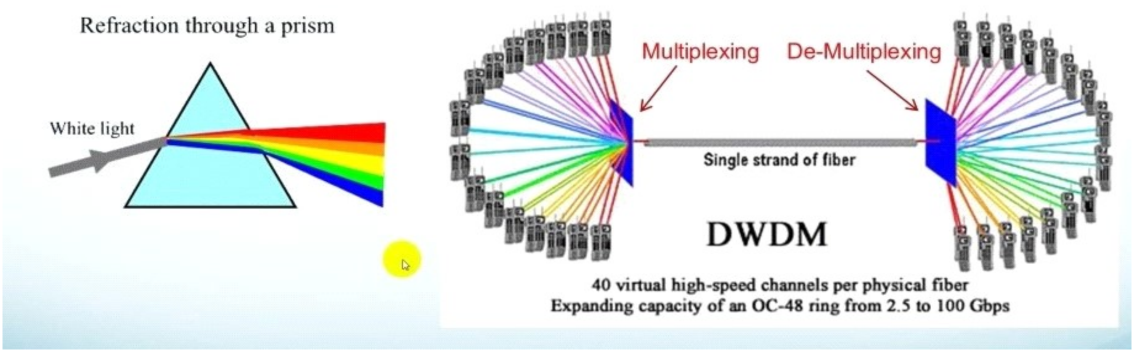

Refraction through a prism
White light
Multiplexing
De-Multiplexing
Single strand of fiber
DWDM
40 virtual high-speed channels per physical fiber
Expanding capacity of an OC-48 ring from 2.5 to 100 Gbps

Figure (2) DWDM Technology

2.1.2 Multi-Protocol Label Switching (MPLS)

MPLS is a technology that presents a mechanism to transmit packets for each network protocol. It was originally designed to give IP routers faster packet forwarding. Its abilities have grown massively to support traffic engineering and service creation such as VPNs.

Conventional IP networks are distributed control planes. When a packet arrives, the router uses the packet destination IP address to define the next hop beside the information from its own forwarding table. The information of the network topology is maintained in the router's forwarding tables and collected via an IP routing protocol, such as RIP, BGP, IS-IS, OSPF or static configureuration, which holds that information synchronized with network changes. Adding labels to all packets is the main concept of MPLS and then the packet is transported by way of the network routers. However, in MPLS (Figure (3)), the domain of the label presents the fundamental information to route the packet. Hence, MPLS technology directs and accelerates the network traffic and leads to simplifying the control [57] [60].

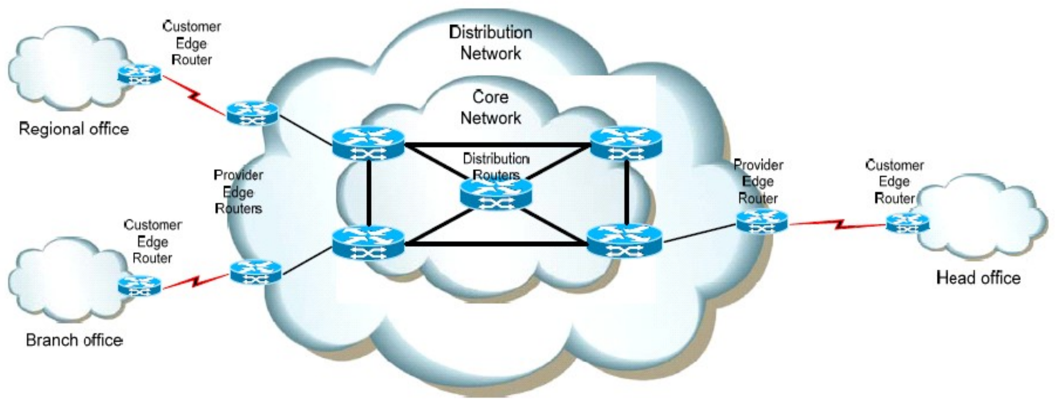


Figure (3) MPLS Network

2.1.2.1 MPLS Terminology

The main expression of MPLS technology is as follows (Figure (4)) [57]:

2.1.2.1.1 MPLS domain: A connecting set of routers that MPLS uses for switching of traffic flows and routing forwarding across a network under a single administrative domain. It is classified as the MPLS core nodes, which are defined as Label Switch Routers (LSRs). MPLS edge nodes are defined as Label Edge Routers (LERs).

2.1.2.1.2 LSR: manages the packet forwarding according to label switching which is found in the core of the MPLS domain. An LSR router can perform layer 3 packet routing.

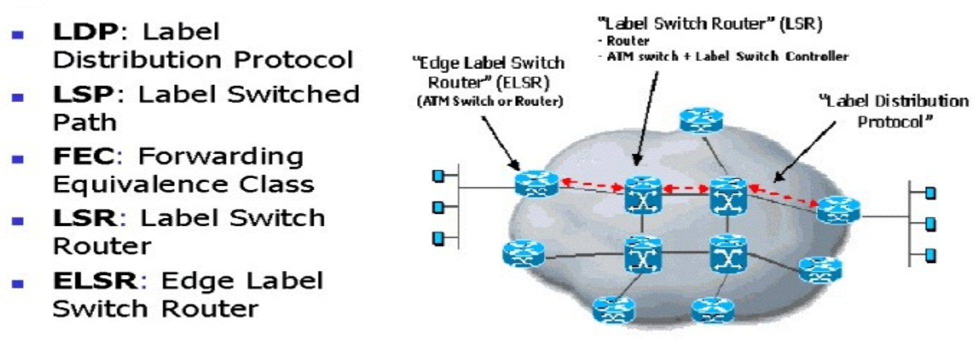
2.1.2.1.3 LER: The management of adding or removing labels from the packets while entering or leaving the MPLS domain being controlled by the LER. The LER can perform layer 3 routing. However, a packet normally enters the MPLS domain through the LER that is named as an ingress router and leaves the MPLS domain from the LER, which is named the egress router.

2.1.2.1.4 Label: is a short constant space identifier that is held by a packet within the MPLS domain and utilized to classify the packet flow to a certain forward equivalence class.

2.1.2.1.5 Shim: is a blank space in a packet between the layer 3 and layer 2 headers built into the MPLS framework. In the shim, a label is coded.

2.1.2.1.6 Forward Equivalence Class (FEC): FEC is a set of packets which have associated features and are transmitted along the same route with the same class. The same MPLS label is given to this set of packets at the ingress router. Each packet is classified with FEC only once in the MPLS network.

2.1.2.1.7 Label Distribution Protocol (LDP): LPD is the primary MPLS signaling protocol in the label routing information. It is switched between LSRs. It is overseen for taking care and building with labels.



- **LDP**: Label Distribution Protocol
- **LSP**: Label Switched Path
- **FEC**: Forwarding Equivalence Class
- **LSR**: Label Switch Router
- **ELSR**: Edge Label Switch Router

"Label Switch Router" (LSR)
- Router
- ATM switch + Label Switch Controller
"Edge Label Switch Router" (ELSR)
(ATM Switch or Router)
"Label Distribution Protocol"

Figure (4) MPLS Terminology

2.1.2.1.8 Label Switched Path (LSP): This is a route created by a series of routers and it starts at the ingress router and moves by one or more core LSRs and eliminates at the egress router. LSP is a special route which is usually held by a set of packets with the same related FEC to the LSP. Many LSPs are possible at any MPLS domain and they are installed using the LDP signaling protocol.

2.1.3 IP/MPLS over WDM

Due to the huge increase of data traffic made by the Internet, it becomes a necessity to have a network design that is more enhanced to support data traffic, so IP/MPLS over WDM is envisaged as the correct choice. MPLS supports improving the quality of service (QoS) by analyzing packets in the edge routers into forwarding equivalence classes and then sending the packets through the label switched paths (LSPs) with labels. This classification coupled with the specific routing of the bandwidth ensures that LSPs allow service suppliers to dynamically provision bandwidth guaranteed routes and to traffic engineer their networks. A conventional transport network transfers IP traffic through multiple layers that comprise a synchronous optical network (SONET) and asynchronous transfer mode (ATM) layers. Such networks may be rather static wherein constant bandwidth links (e.g., SONET) are normally set up manually. On the other hand, in IP/MPLS-over-WDM networks, dynamic IP/MPLS links are calculated into light routes that can be dynamically provided (See Figure (5)) [58].

Optical cross-connect (OXC), optical add-drop multiplexing (OADM) and the DWDM network interface card (DWDMNIC) for the customer devices (such as MPLS LSR or IP Router) are the main elements for WDM optical networking. OXC and OADM provide wavelength routing and switching, fiber/port switching, wavelength conversion and waveband switching. The optical switching purposes are performed in either with Optical-Electrical-Optical (O-E-O) architecture or the all-optical switching architecture. With respect to the architecture, there may be various limitations, such as the amount of wavelength conversion and the number of wavelength converters in an OXC as well as the optical frames being transferred to the higher protocol layers through the add-drop ports of OXC/OADM, such as the MPLS or IP layer [59].

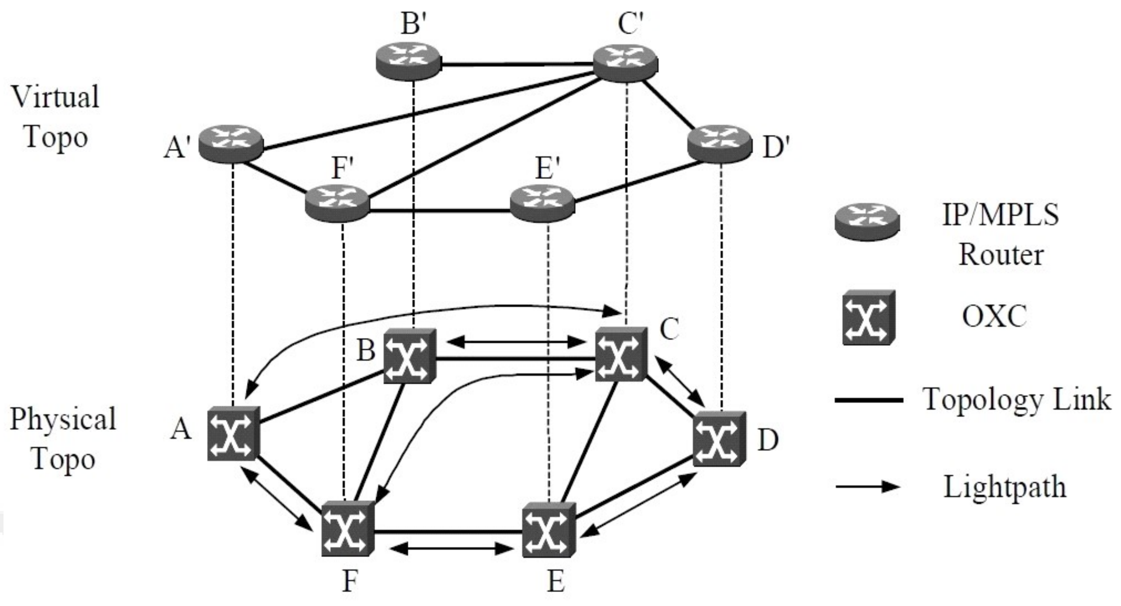


Figure (5) IP/MPLS over WDM

The light paths set up with respect to the demands via O-UNI signaling, delete, modify or status inquiry of the light paths is the most key part of the control plane of a WDM optical network. The control plane should have the functional modules of routing (such as OSPF or IS-IS), signaling (such as LDP or RSVP, BGP), and wavelength assignment for this light path handling [59].

2.1.3.1 Interworking Models of the Optical Internet:

The internetworking of the IP/MPLS network and optical network can be considered in terms of [59]:

- A service model.
- An interaction model.
- Routing approaches.

2.1.3.1.1 Service model:

In the service model, the optical layer network and the IP layer network can work as [59]:

A client-server service model.

An integrated service model.

Client-server service model: A collection of larger bandwidth channels are presented by the optical network to the customer IP/MPLS layer.

Integrated service model: The optical layer and the IP layer networks are handled as one network and there is no difference between the optical nodes and the IP/MPLS nodes in so far as the control plane works.

2.1.3.1.2 Interaction model

Three scenarios in IP/MPLS over the optical interaction model [59]:

- Overlay model
- Peer model
- Augmented model

Overlay model: A point-to-point connection optical network supplies the IP/MPLS domain. The IP/MPLS routing protocols are separated from the signaling and the routing protocols of the optical layer. When the network administrators of the IP/MPLS layer network and the optical layer network are separate in the case of the privacy and security of network information, this overlay model will be utilized.

Peer model: The optical nodes and IP/MPLS nodes serve as equal nodes and there is a single example of the routing protocol operating in the IP/MPLS domain and optical domain.

Peer model: The optical nodes and IP/MPLS nodes serve as equal nodes and there is a single example of the routing protocol operating in the IP/MPLS domain and optical domain. For topology information exchange, a prevalent IGP, such as OSPF or IS-IS, can be employed. In this model, all IP/MPLS nodes and optical nodes have a common addressing scheme.

Augmented model: Decouple routing cases are placed in the optical and IP/MPLS domains; however, an example of single routing information is given in another routing example. For instance, an optical network can have identical information which is assigned to components and carried with optical routing protocols to support accessibility of information to deploy it by the IP domain so as to maintain it to a certain extent in an automated process.

2.1.3.1.3 Routing approaches:

Three routing models [59]:

- Fully peered routing model
- Domain specific routing model
- Overlay routing model

Fully peered routing: is utilized for the peer interaction model where one example of the routing protocol operates in the IP/MPLS and optical domains.

Domain specific routing model: works with the augmented interaction model where the routing instances are decoupled in the IP/MPLS domain and optical domain. To interchange information between the optical and IP/MPLS domains, the inter-domain uses BGP or OSPF routing protocols.

Overlay routing model: This works with the overlay interaction model. The optical routes for the IP packet transmission are set up over the optical network. The optical domain network can keep a record of the IP addresses and customers determine to which it is linked.

2.2 Software defined networking (SDN)

The evolution of SDN started in the 1990s with the concepts gathered from different supportive technologies. The control plane and data plane separation concept that was taken from the network control point, which performed the data and control, are decoupled with each other in telecomunication networks. This is demonstrated as a reasonably priced and reliable answer. Through an application interface, the programmability in the network was introduced by active networks. Tempest, Virtual Network Infrastructure (VINI) and the programmable router switch model are instances of active networking models which provide the elasticity to control various assignments and performances. Due to the lack of suitable hardware and infrastructure support, these models were found much earlier and they could not be completed. The significant contribution of SDN began in 2008 with the introduction of OpenFlow [54].

2.2.1 SDN Architecture

To understand the functionality of the SDN structure, there are three main layers, or SDN planes, which are presented in Figure (6) and formed of the following:

- Data plane
- Control plan
- Application plane

2.2.1.1 Data plane

An SDN data plane includes the network devices such as a switch, a router and an access point. Coexisting in this plane are both the physical switches and virtual switches such as Pica8, Indigo, Open vSwitch, Nettle and Open Flowj. The major function of the data plane is to send the packets with respect to the instruction rules or policies from the SDN controller.

2.2.1.2 Control plan

The control plane includes a controller which manages all SDN operations. This plane represents the mediator for the data plane and application plane. The controller is responsible for controlling the entire traffic flow and it is the only decision-maker on flow transference, routing, and packet dropping by way of a programmable interface. On the basis of east-bound and west-bound interfaces, the controllers in the decentralized medium connect with other controllers. The control plane and data plane connect with one another by way of a south-bound API, such as OpenFlow.

2.2.1.3 Application plane

This plane is the uppermost layer in the SDN. It oversees performance software associated business and security applications. Some instances of applications performed by this plane include (Intrusion Prevention Systems (IPS), Network virtualization, mobility management and Intrusion Detection Systems (IDS). This plane connects to the control layer using a northbound application interface, also known as the Application Control Plane Interface (A-CPI) [54], as shown in Figure (6).

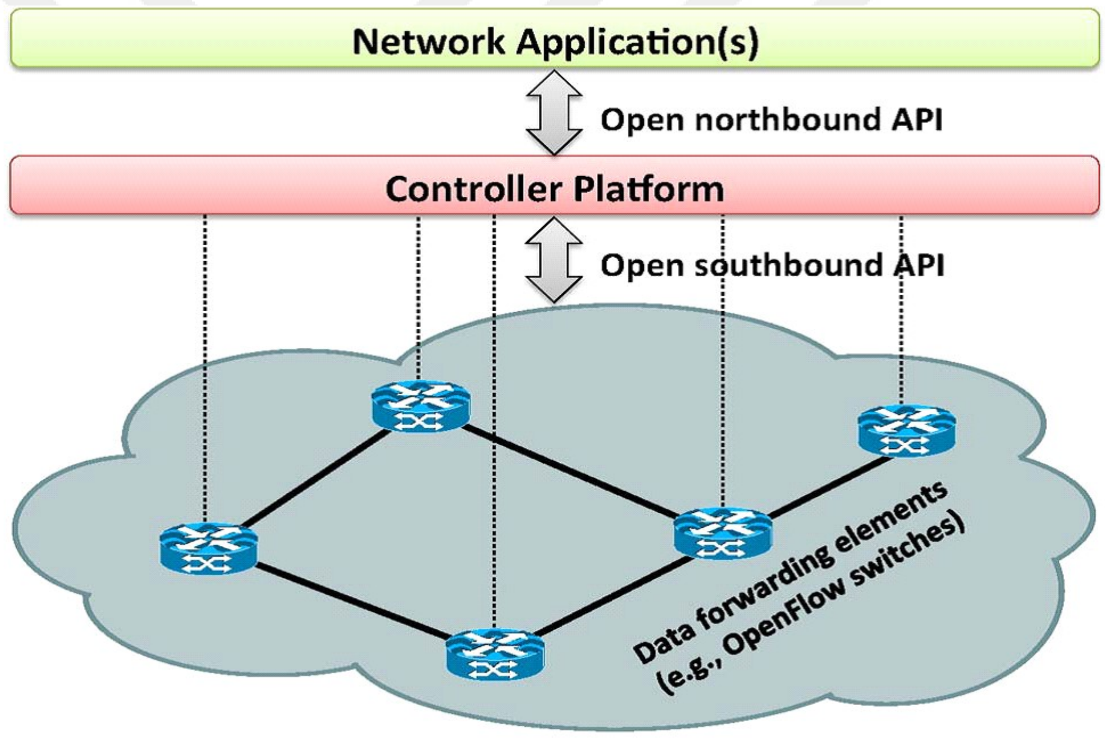


Figure (6) SDN Architecture

2.3 OpenFlow

OpenFlow [13] is a protocol that enables SDN based networks to handle connections between the SDN controller and Ethernet switches which have been standardized by the Open Network Foundation (ONF). OpenFlow came from SANE [61] and Ethane [62], which introduced the idea of separating the control and data planes, and it soon began to be a more famous open model which grew fast to handle many more functionalities.

A switch which enables OpenFlow originally included three basic parts [63] (see Figure (7)): Flow and group table(s), an OpenFlow channel, and an OpenFlow protocol. OpenFlow is almost new and it remains in development with enhancements and changes of OpenFlow's features and new operations to suit modern network needs.

While commanding its forwarding table manner by providing flow rules, the OpenFlow protocol enables the controller to learn connectivity statuses, details of the hardware of Ethernet switches and the topology of the network. Low and group tables define the manner of the switches by using data flow arriving from various interfaces (physical, virtual). Where the data communication flow is defined, the table includes a collection of rules. The switch responds to every flow with respect to the flow rules. An OpenFlow switch includes a group table for frame searches and forwarding as well as one or more flow tables. A flow rule includes three fields (see Figure (8)):

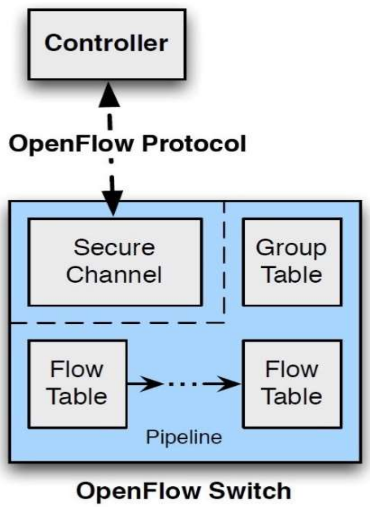


Figure (7) OpenFlow

Rule: A header for corresponding to the flow frames. OpenFlow specification supports many Ethernet headers [63]; however, expandable, specific headers can be arranged in the manner of OpenFlow. The switch simply supports a bit mask match. Therefore, non-IP traffic is forwarded freely by the OpenFlow switch.

Action: Constitutively and corresponding with traffic, the execution should be defined by the action. Actions are also open for expansion; however, some rule actions are currently supplied at the specification. Examples include drop the frame, forward to the controller, forwarding to one or more ports, and modifying the frame fields. The one necessary item is the data path which has to have elasticity at low cost and implemention of high performance to add modified actions.

Statistics: Every time the switch updates the frame counters when a flow rule is matched, it shows the popularity of a particular flow. Every table contains counters as well as all flows, each port and every queue. Moreover, an initial set of the flow and a timer for the last activity are kept.

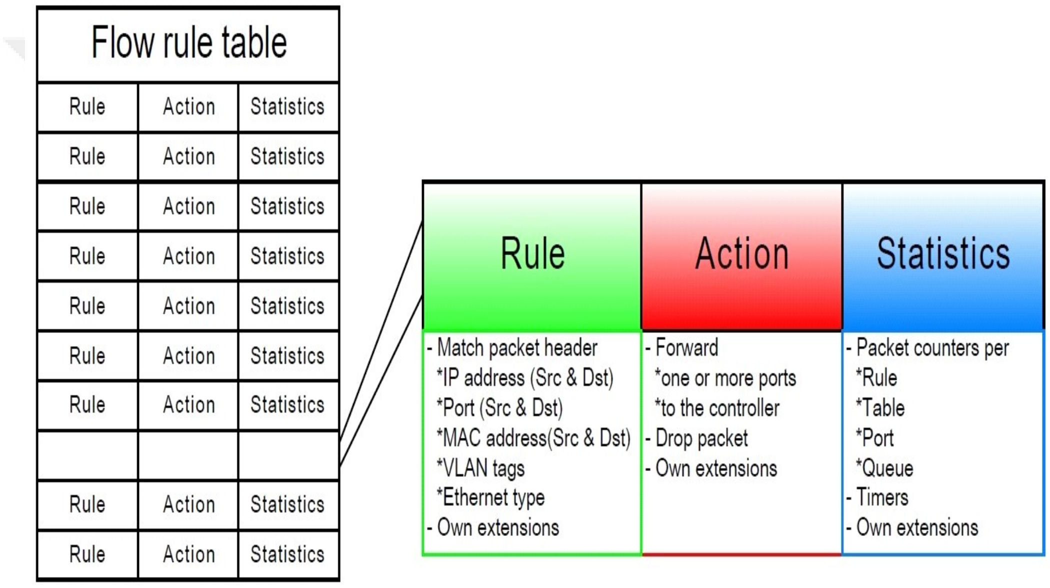


Figure (8) OpenFlow Fields

The Transport Layer Security (TLS) protocol the OpenFlow channel between the controller and the switch are normally encrypted. Nevertheless, utilizing a plain Transmission Control Protocol (TCP) can also execute the channel. An interface for the controller is supplied by the OpenFlow channel to control and improve the OpenFlow switch flow and group tables. Simultaneously, the switch also provides the controller which has a hardware status, ports connectivity state, and meter statistics of each flow rule. The OpenFlow protocol uses a connecting message plane which is predefined when communicating between the switches or the controller and the switch.

CHAPTER 3

LITERATURE REVIEW

Currently, the execution of SDN is one of the hot topics being researched. For SDN to replace current conventional networks, its performance features have to be considered and evaluated.

In Ref [30], the authors showed the characteristics and benefits of the SDN framework for optical networks. They explained the advantages and the execution issues of SDN in optical networks. Moreover, they presented the benefits of SDN-based network management.

In Ref [68], for intra data center switching, a software defined optical network (SDON) depending on multi-level WDM ring topology is presented. A routing approach and the contention solution are implemented on the basis of a multiplexed wavelength that is controlled by a designed Uniform Control Plane (UCP).

In Ref [69], the authors proposed a design to enable SDN features in combined packet optical networks in a control layer frame based on OpenFlow. A mechanism to execute OpenFlow on optical networks for abstraction was invented. The project determined that the preparation time and stability were enhanced when OpenFlow was immediately connected to the transport mechanism of the optics.

In Ref [70], the authors introduced an advanced structure for network performance and management recognized as OPCInet and its expansion to SDN. They discussed the enhanced elastic nature and improved the energy consumption with the high-speed switching ability of OPCInet. It was shown that the configureuration of the mapping and switching tables can be performed through an SDN system based on the packet according to the service provider’s needs by the web.

In Ref [71], the authors determined that a UCP of the GMPLS was absolutely un-usable as it is complicated, buttoned-down, and inelastic to be of any use from a service viewpoint over packets and circuits. IP networks would not connect with it. They introduced an SDN-based structure for a UCP as it is straightforward, expansible, and programmable and can be regularly utilized.

In Ref [72], the authors created, performed, and confirmed an SDN-based management mechanism with heterogeneous control planes that can handle the problem of GMPLS and OpenFlow interworking with different multidomain networks. The mechanism includes the description of operative and protocol structures depending on the ideas of network abstraction, stateful PCE and hierarchies.

In Ref [48], the authors described the SDN-based managements in optical transport networks. A framework was presented for the global improvement of available resources of the optical networks. The controller suggested uses in different processors placed at different critical positions in the networks to increase the expandability and reliability of the network. While presenting these superior performances, the network continued to be enhanced from the resource usage point of view.

In Ref [73], the authors introduced a control plane module with the concept of Open-Flow to enable SDN functions in optical networks. They mentioned the needs and portrayal of performances of Open-Flow convention extensions for transport optical systems utilizing business optical properties and they also looked into models of improving optical transport. Their explorations of the acceptances and performance testing of the introduced model showed an improved way to prepare times and control security when Open-Flow is communicated simply to optical transport advances.

In Ref [75], the authors developed different QoS associated characteristics in the Ryu controller. Simultaneously with network resource allocation/control and rate-limit mechanisms, they presented a flow-based QoS. The suggested system executes, controlling the network resources in a more effective manner and providing optimized rules for traffic forwarding.

In Ref [76], the authors developed an SDN network controller to solve the challenges of application-centric multi-layer transport network management. They gave a general review of its structure and demands, intent-based north-bound interface, and allocation algorithms of application-centric multi-layer resources.

In Ref [74], the iNDIRA tool was proposed, which communicates with SDN north-bound interfaces to allow intent-based networking. It presents reliable, uncomplicated, and technology-agnostic interaction between clients and networks. The authors showed a working tool to manage network intent into physical network provisioning commands and executing file transmits. This design was presened for probable future improvements by combining intelligence algorithms to predict user/application demands to configureure networks for best performance.

In Ref [77], the authors proposed an SDN controller that provided adaptive route supplying in dynamic assistance chaining to recover from congestion issues during service chain routes. Moreover, the SDN controller showed a north-bound interface based on REST rules and on the Intent-Based Network NEMO Language characteristics that enable applications to specify their demands for the network (including dynamic service chaining) without being affected by low-level execution details.

In Ref [78], the authors examined the NBI structure and the intent-based NBI rules guiding by ONF WG. The intent-based NBI method is required to evolve expanded and business-like network implementations; however, understanding the NBI in a practically structured manner rarely occurs. The authors introduced a structure based on micro service and service-oriented design rules and three-line application structure to understand the intent-based NBI. They integrated the ONOS and OSGi platform to build their model.

In Ref [79], to specify the network as a policy governed service and to perform an effective network virtualization structure, the authors introduced an intent-based modeling abstraction, Distributed Overlay Virtual Ethernet network (DOVE), obtaining the introduced abstraction. They explained the working model of the DOVE structure and published the results of the comprehensive simulation-based execution study, illustrating the extensibility and the competence of the solution.

In Ref [80], based on software defined Network Virtualization (NV), the authors introduced an intent-based virtual network control platform. The goal of the introduced NV scheme is to control automation and configureuration of virtual networks based on high-level clients demands characteristics, called intents. The model and execution of the scheme are created on ONOS, an open-source SDN controller, and OpenVirteX, a network hypervisor. The scheme was created to enable multiple VNs over the same physical devices for multiple clients. The VNs are separated from one another to provide clients with the ability to work and control their virtual networks separately in terms of network configureurations and control rules.

In Ref [85], the authors generally surveyed studies that test the SDN model in optical networks and viewed the field of software defined optical networks (SDONs). They ordered the SDON studies into the studies which were concentrated in the data layer, which was the control layer and the application layer. In addition, they included SDON studies concentrating on network virtualization. Furthermore, SDON studies focussed on the management of multidomain networking and multilayers.

In Ref [86], the writers had introduced the notion of SDN and important advantages of it in providing improved configureuration. Moreover, the notion of SDN developed performance and boosted invention. They had given a literature study of the latest SDN studies on the data layer, the control layer, the application layer. In fact, they presented OpenFlow as the actual SDN execution.

In Ref [87], the authors presented a software-defined optical data center solution for the accommodation of various DC implementations and meet their continuously growing demands. They demonstrated both the programmable optical data plane and the SDN-enabled control plane with optical expansions. Moreover, they clarified the optical DC virtualization work-flow.

In Ref [88], the authors examined hybrid SDN prototypes that combined SDN with a more conventional networking method established for distributed protocols. They displayed an amount of use cases in which hybrid designs can decrease the respective restrictions of conventional and SDN methods, giving incentives to (somewhat) shift to SDN. Moreover, they presented the qualitatively different tradeoffs that are easily applied in hybrid designs, making them suitable for various transition plans and long-term network configureurations.

In Ref [89], the authors surveyed modern studies in traffic engineering methods by focusing on traffic engineering for SDN. They concentrated on a number of traffic engineering techniques for the conventional network structure and the exercises that can be learned from them for better traffic engineering techniques for SDN-based networks.
In Ref [90], the authors introduced an iteration model for Elastic Optical Networks (EONs) that depended upon the SDN framework and the productivity being compared to the distribution of GMPLS recovery, GMPLS/PCE recovery, and a simple execution of an SDN installation. The introduced model gathers the signaling messages which are required to build the backup routes with fewer messages for the synchronization of the required node reconfigureurations and, accordingly, reduce the recovery period. An independent source-driven signaling session was always needed as a method to fit with the GMPLS control plane for each of the lightpaths to recover it. Moreover, a PCE was utilized for path calculation.
In Ref [91], the authors introduced a smart SDN that allowed heterogeneous network access that can supply connectivity to all femtocell, cellular and Wi-Fi communication technologies using Time-Wavelength Division Multiplexed Passive Optical Networks (TWDM-PON) hauling and near to the premises of intelligent access nodes. They introduced a control layer that allows current technologies to work collectively whilst allowing the most intelligent orchestration systems for all technology to maintain command.
In Ref [92], the authors introduced the Software-Defined Access Network (SDAN) concept, where they recognized industry directions that support this concept. The decouple sections handle the network control and the infrastructure plane control functions demonstrate the SDAN. A few particular instances of SDAN usage scenarios are presented, and some general results are illustrated. They concentrated on the controller, control layers and connections on the services layer.
In Ref [93], the authors introduced a scheme for both the physical plane and the SDN control plane for elastic optical network (EON) devices enabling spectrum defragmentation. They examine the OpenFlow message and signal execution, showing suitable and efficient control of SD-EON devices. The introduced management design

prevented the data layer details of defragmentation and allowed the SDN controller only to concentrate on the control of spectrum manipulation. It was also appropriate to other data-plane executions, showing that the data layer policies in the controller and the procedures of the node agent were updated.

In Ref [94], the authors introduced a control plane structure based on OpenFlow to allow SDN processes in combined packet/optical networks. An abstraction method for allowing OpenFlow on optical nodes was produced and performed on commercial hardware. They addressed demands and illustrated executions of OpenFlow protocol expansions for transport optical networks including commercial optical elements and research models of beginning optical transport technologies. Empirical validations and execution analyses of the introduced structure illustrated enhanced route installing times and control balance when OpenFlow was employed directly to optical transport technologies.

In Ref [95], the authors employed the software describing elastic optical networks (SD-EONs) and showed countless studies specifying the difficulty of locating Elastic Optical Network solutions built upon SDN. Moreover, the majority of studies confirm that the SDN model offers are far better (e.g., decreased light-path installation period) than GMPLS-based models. Additionally, they displayed some up-to-date knowledge about the point of view that operators apply real implants of that technology at the core network to predominate GMPLS existing solutions.

CHAPTER 4

LIMITATIONS OF TRADITIONAL NETWORKS

The impulse to adopt SDN technology qua Next Generation Networks (NGN) can be envisioned with the Limitations of Traditional Networks are given as follows:

4.1 Control and forwarding in the same device

Network intelligence is distributed in existing networks. The control plane (the decision maker for network traffic handling) and the data plane (which dispatches traffic with the help of decisions made by the control plane) are integrated into the same network equipment, thereby decreasing compatibility and obstructing evolution and renewal of the networking infrastructure. This cramped coupling is the fundamental reason for more complicated network design devices, essentially network switches and routers. Deployment and development of new networking features (e.g., routing algorithms) are made difficult because of this integration. When it would mean modifying the control plane in all network equipment by way of installing a new framework and in a few cases, hardware would develop. Such coupling compels network operators to hide their network controls so as to prevent needless jeopardies. Network implementations must reveal the network infrastructure device topology by attending detailed infrastructure device status information. The accomplishment of fine-grained network flow arrangements and operations then becomes unfeasible. It becomes impractical to deploy and test innovative ideas in an efficient network setting. In spite of the control plane abstraction of the network core equipment having made a little progression, there is still an absence of consolidated open APIs for various network equipment [1] [2] [9] [11].

4.2 Individual configureuration

The current communication networks are not easy to manage or control because of the perpendicular coupling and the configureuration of each layer being specific and separated. Current network protocol layers are coupled in a complex manner within networking devices. Operators of a network have to arrange every single piece of network equipment individually using low-level and vendor-specific orders in order to express the desired high-level network policies and to alter the traffic flow in response to altering network status (Figure (9)). Besides the installation complexity, the dynamics of errors has to be endured by network environments and adapted to load changes. Current IP networks are practically non-existent in terms of response mechanisms and automatic reconfigureuration, thereby compelling the necessary policies. For instance, the switch from IPv4 to IPv6 began more than 10 years ago and it is still mostly unfinished. In fact, IPv6 solely represents a protocol update. Due to the inertia of updated IP networks, five to ten years is needed for a fully designed, evaluated, and deployed [2] [30] new routing protocol.

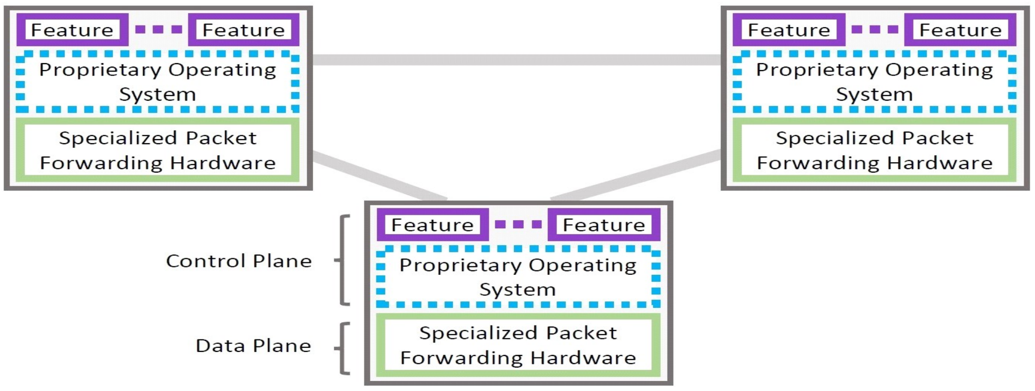

Feature
Feature
Proprietary Operating System
Specialized Packet Forwarding Hardware
Feature
Feature
Proprietary Operating System
Specialized Packet Forwarding Hardware
Control Plane
Data Plane
Feature
Feature
Proprietary Operating System
Specialized Packet Forwarding Hardware

Figure (9) Legacy Networks Architecture Predefined set of Distributed Control and Management Features

4.3 Non-programmable

Under heavy network traffic and the limitation of existing network devices, this methodology poses a problem, which is the rigid restriction on network performance. Cases such as reliance, trustworthiness, network speed and the increasing demand for scalability can virulently block the productivity of the updated network devices due to increasing network traffic. Existing network equipment lacks the elasticity to cope with various packet variations with different content because of the infrastructure hardware execution of routing rules. Furthermore, the networks which comprise the most important parts of the Internet need to be able to adjust to changes without being largely labor intensive in terms of hardware and software modifications. Nevertheless, traditional network processes are difficult to be re-reprogrammed [3] [33].

4.4 Same vendor devices

The architectures of the legacy network, which depend upon objective and vendor-specific systems, are composed of specialized forwarding chips and integrated predefined features and proprietary operating systems. As these networks are heterogeneous, the administration of these networks is too complex. Even though human operators also make contributions to network downtime (faults) because of manual setup of network elements (NE), network equipment outages can also be responsible. To apply new network policies, the vendor-specific command line interfaces (CLIs) are used by the operator to configureure each device. An operator might wait a long time before a device vendor announces a software upgrade that backs that predicted property in order to provide a new feature. Because of these inconveniences, a traditional manner of approaching setup, optimization and troubleshooting could grow into inefficiency and in some instances insufficiency [1] [49].

4.5 Misconfigureurations and Errors

In today’s networks, misconfigureurations and related errors are frequently common. In border gateway protocol (BGP) routers, more than 1000 configureuration errors have been observed. Very undesired network behavior may result (including setting up unintended paths, packet losses, forwarding loops, or service contract violations) from one misconfigureured device. Indeed, while rare, one misconfigureured router is capable of compromising the correct operation of the entire Internet for hours [2].

4.6 Rigidness and Complexity

Small limitations, including maladjusted policies, failure to scale and vendor reliance, are caused by the complexity in existing network architecture.

A. The current network design is established based on packet accessibility. To meet the requirements of security, scalability, reliability, and QoS from different applications, various network protocols are developed and designed independently to solve single application problems. On the other hand, network devices (routers, switches, etc.) supporting these protocols have become increasingly complicated. This complication increases the maintenance overhead and the difficulty for further innovation and risk of error. Consequently, network operators have become conservative when introducing network changes for risk management. It becomes difficult for the existing network architecture to adjust to the new directions of mobility and virtualization.

B. Network traffic patterns have changed much because of mobile devices with wireless networking and cloud computing. The wide use of mobile devices and virtual machines creates regular network topology updates and dynamic network node migrations. These types of topology change pose challenges to the existing network.

C. Internet corporations, such as Facebook and Google, are required to provide network services at much larger scales than ordinary service providers. They are also required to request load balancing and dynamic traffic scheduling in their network infrastructures. Unfortunately, the current network falls short of these challenges because of its rigidity and complexity [10].

4.7 Control and Monitor

Traditional networks use particular algorithms performed on dedicated devices (hardware components) to monitor and control the traffic flow in the network. They manage routing paths and locate how various devices are linked in the network. Generally, these sets of rules and routing algorithms are executed in dedicated hardware components such as Application Specific Integrated Circuits (ASICs). ASICs are prepared for implemention of specific operations. Packet forwarding is a simple example [32].

4.8 EMS/NMS control

The control constructs for a conventional circuit, as well as for optical or packet switches that utilize a centralized network management system (NMS) for their modification to add an application interface for modification to support application-driven (re)reconfigureurability of network resources. The overall flexibility will be located by the physical limitations of every architecture. An element management system (EMS/NMS) manages conventional optical networks. The application layer is still able to demand resources on this network, but an API that performs the northbound interface of the EMS/NMS is used in such interactions. Since the control plane is still embedded in the network elements, the control is limited to the functions that the EMS/NMS can access [48].

4.9 Capital and Operational costs

To support network management, a small number of vendors offer proprietary solutions for operating systems, specialized hardware and control programs (network applications). Network operators need to maintain and acquire identical specialized teams and various management solutions. The capital and operational costs of creating and preserving a networking infrastructure are important, with long-term returns on investment cycles, which obstruct invention and the addition of new features and services (for instance, load balancing, access control, traffic engineering and energy efficiency). To relieve the lack of in-path functionalities within the network, a myriad of middleboxes and specialized components (such as deep packet inspection engines, intrusion detection systems, and firewalls) proliferate in current networks [2].

4.10 Routing

When routing devices receive packets in a traditional network, the network appoints a group of rules which are already embedded in the devices to reveal the path for those packets as well as the address of the end device in the network. In general, data packets are handled in an analogous manner, which may be transferred to the same destination, which can happen on a cheap routing device. More expensive routing devices can address various packet types in different manners based on their type and contents [31] [49].

4.11 Packet Recognition

In traditional networks, web content flow and video content flowpaths cannot be separated automatically and directly. Because the control plane is distributed in each router, every router checks for the best route by using default parameters. Moreover, routers cannot easily recognize the differences between video content and web content. This means routers cannot readily separate web content from video. Therefore, it cannot decide to which interface it should send the traffic in relation to the traffic type. Most significantly of all, they are unable to do all these operations in dynamical form [50].

CHAPTER 5

5.1 SDN improvements and solutions for traditional networks

SDN offers several benefits to the management and control of conventional networks.

5.1.1 Separating (Solutions for Control and forwarding in the same device)

Nowadays, there is a newly emerging network model called SDN. The main advantage of this network model is the change of the strict restrictions of current network infrastructures. The data plane and control plane are decoupled by SDN. Therefore, perpendicular integration is fractured. By means of decoupling, the control plane can be performed in a network operating system or by a logical central controller. Network devices can be used as forwarding devices, which is a very simple process. This simplifies evolution, execution of the policy and reconfigureuration of the network.

Some of the concerns, such as inserting between switching hardware, network rule definitions and the enforcement of these network rules in forwarding, are decoupled by SDN. This decoupling is very important because many specifications, which require flexibility, easy network management, simple network development and invention, simple establishing and inserting new abstractions into the network and separating the network control problems into tractable segments, are obtained by decoupling.

In an SDN, separating the data plane from the network control logic has an important role in a high-performance computing (HPC) network. In the data plane, flow tables are used for forwarding packets using switches. These flow tables hold a list of the flow entries. Each of the flow entries consists of three fields, namely counter, instruction and matching. It provides enhanced network performance in relation to data execution [2] [3] [31].

5.1.2 Centralization (Solutions for Individual configuration, Non-programmable and Routing)

The main priority of SDN is to centralize control and management. In SDN controllers, which are software based and maintain a global view of networks, network intelligence is logical central.

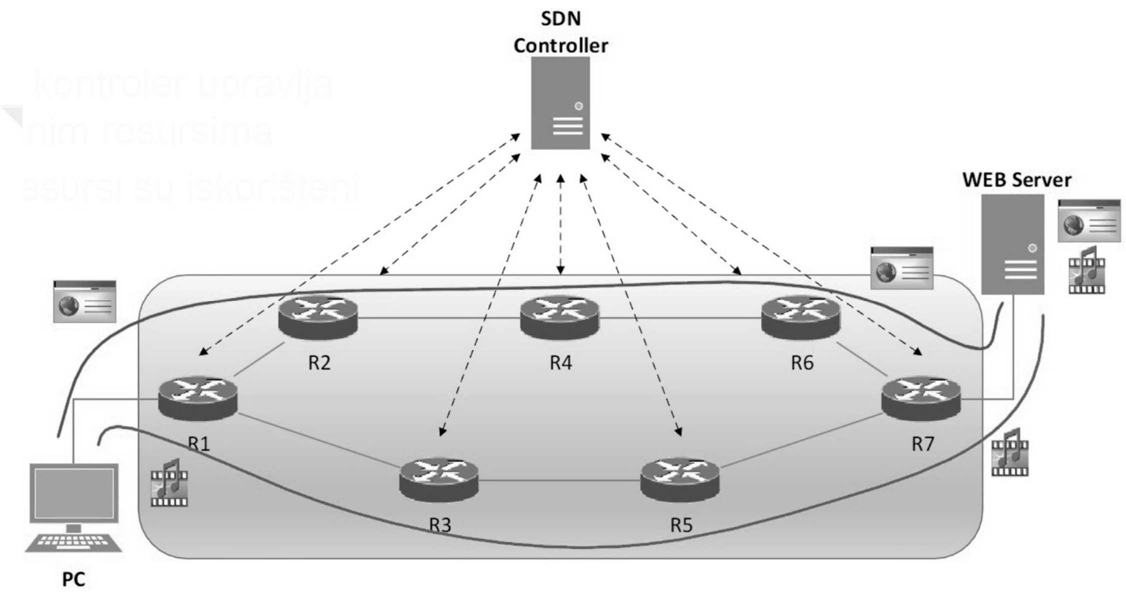


Figure (10) Network Management with an SDN Controller

A centralized SDN controller has a holistic network map, seen in Figureure 10. To enable the holistic optimization of network resources, routing and switching are easily and strategically altered in relation to various applications, such as QoS (quality of service), traffic load balancing,

to repair the network rules through advanced standard languages and software parts by being less error-prone and simpler, checked with lower level tool specific forms. Evolution in advanced network functions, services and applications is facilitated by the control plane in a controller with holistic knowledge of the status of the network. A control program can react to the pseudo alterations of the network status automatically. In this way, high level policies can be preserved intact. An abstract forwarding paradigm is accomplished with SDN controllers. This forwarding paradigm includes a forwarding task and the holistic network map is used as the input. The controller can make informed and intelligent decisions about switching and forwarding through the availability of the holistic network map; then it is possible in a distributed scenario. Switches acquire the switching information via the flow channel, which is a secure channel with the controller; then the switches execute the packet forwarding. An SDN controller can calculate a route on a network that satisfies wavelength constraints, circuits, or connections by using topology information. Additionally, optimization of the connections or circuits can be performed better with SDN because of central calculation in SDN. It can also calculate various routes regardless of whether they share the same destination. When there are some restrictions between nodes regarding independent connections or when there is optimization of the holistic network source, SDN is very useful over the control plane, which is distributed throughout the elements of the network. The limitations of the distributed control plane can be eliminated with centralized control and with the centralized control's holistic knowing about the requirements of the application and resources of the network. The SDN controller may compensate for the disability to fulfil the requirements of the application, such as calculating a route in the IGP (Interior Gateway Protocol) or calculating a route which satisfies a latency constraint. In addition, an SDN can collect some suitable measurements about any lost packets and delays. Moreover, the performance monitoring ability of the routers can be configureured with SDN. If there is congestion or increased delays or degraded links, a route which avoids problematic segments can be recalculated using SDN. Coordination of the application network demands various technologies that can be done easily with a centralized SDN control; here, centralized SDN control shows its power in this area. Each of the centralized SDN controls has specific features to recognize the serving network. This type of position emerges in multi-tenant data centers. In these data centers, dynamical adjustment or creation are needed in a tenant network because selecting the nodes of the network is needed and there is connectivity. Additionally, this position also emerges in optical layers, Time-Division Multiplexing and multilayer networks [2] [48] [53].

5.1.3 Configureuration and management (Solutions for Control and Monitor)

SDN separates the data plane and control plane. In this way, an SDN grows into an external structure which is a Network Operating System or SDN controller. The controller supplies the abstract and programming languages to the network, which can participate because that application programming is becoming easier. The benefits of the information on the same network can be gathered by all of the applications. The applications make more harmonious and efficient rule decisions when the software modules of the control framework are re-utilized. Actions such as reconfigureuring the forwarding switches can be performed by these applications from any part of the network. Thus, there is no requirement to innovate an accurate strategy regarding a new functionalities position. Therefore, the combination of various applications becomes simpler [8]. For instance, the sequential integration of routing applications and load balancing can be performed with decisions of the load balancing having some priorities on routing policies [2]. Through a monocular software controller, a device for the multiple forwarding plane becomes controllable. With OpenFlow, the direct control of data devices in the network is expanded by the controller plane of the SDN, including interface controls, switch controls and the data plane through the API. This can be seen in (Figure (11)). Some cloud resources such as storage, processing, virtual machines, bandwidth or scientific experiments conducting processes, are utilized by cloud users. A single controller or multiple controllers may exist in the control plane. In a state of a multiple controller environment that is peer-to-peer, SDN can create fast and dependable distributed network control [49]. From the central position, network routing and switching device features can be changed by SDN. Thus, data flow can be controlled by managers. With control applications which are executed as software modules, there is no need to deal with every component individually. This enables

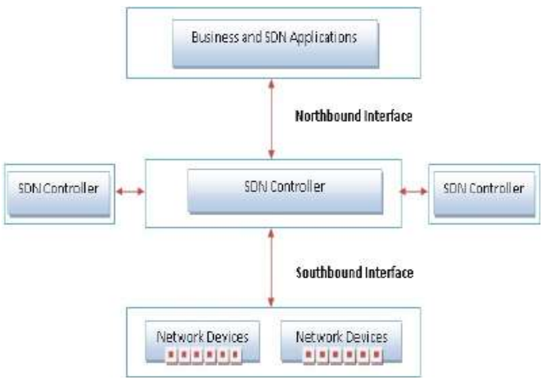


Figure (11) API Directionality in SDN architecture.

network managers to enforce changes of routing in routing elements. Network managers can control the primacy of confirmed packets or they can allow the packet or block it from flowing. This feature gives them an additional control layer through the data [3] [31-33].

5.1.4 Decreasing the network core's complexity (Solutions for EMS/NMS control)

SDN technology separates the control plane and forwarding plane, where the network operating system is utilized by controllers in order to control network resources in a consolidated approach from a logical center. For the upper application layer, an SDN also defines controller APIs. Before distribution to individual switches, the controller manages and implements network policies, removing the requirement to manage each switch manually. Meanwhile, the updating costs for the devices are avoided by the providers of the network core devices by adding the wide range of management protocols. An effective, automatic management framework and elastic to treat the complication in the network configureuration and control is provided by the SDN architecture by substituting the large number of manual processes. The SDN also improves security management and error control with the automated systems and centralization [9].

5.1.5 OpenFlow

In OpenFlow, network administrators can remotely manage routing tables. When comparing between protocols, which creates software-defined networks, the most popular one is OpenFlow. In this protocol, there is a centralized packet switching system. Thus, the network can be independently programmed from a cloud data center and the individual switches.

When the OpenFlow protocol is implemented, the OpenFlow switch can be the software or the hardware. With regard to connecting OpenFlow switches and computing devices, the SDN paradigm can be constructed. Although traditional switches and routers do the high-level routing (control path) and fast packet forwarding (data path) on the same device, OpenFlow switches separate these operations. Fast packet forwarding is still performed on the switch, but the control path is on a separate controller as the standard server. In addition to that, in the OpenFlow protocol, there are some messages such as send-packet-out, get stats, packet received and modify-forwarding-table. The controller communicates with the OpenFlow switches by using these messages [9].

5.1.6 For more reliable and secure application, improving the capabilities (Solutions for Misconfigurations and Errors)

With a holistic view, an SDN creates network abstraction and control network resources. It guarantees flow scheduling, detection of errors, controls access and network security planning. Thus, an SDN efficiently reduces errors and network configureuration conflicts. Moreover, an SDN supplies flow-based fine-grained control. Thus, for more accurate processes, capabilities of the application are granted [9].

5.1.7 Providing vendor independence to improve application supports (Solutions for Same vendor devices)

To execute interactions of the control plane and for management of the control plane and the forwarding plane by united protocols such as Open-Flow, harmonious APIs are provided by SDN. Meanwhile in the production system, SDN technology allows researchers to examine innovative ideas independently without assuming their operation. Additionally, the IT department can apply network policies at different levels such as at device, user, session or application levels in an automated fashion and in a highly abstracted manner by OpenFlow's low-based management paradigm [10]. With this ability, developers can build unmatched network applications. Until now, there have been ingrained causes for deploying this type of network. Firstly, multicore technology production has some advantages including decreasing the cost of network elements and enabling the choice to use a general-objective CPU in order to forward packets and calculate the network policy. Secondly, for interoperability, heterogeneous networks (wireless, IPv6) with various functions, which are rapidly increasing, are needed for the elasticity of network devices. Thirdly, in network management, cloud computing and virtual flourishing have created some new needs, including traffic scheduling and flow monitoring [9].

5.1.8 Revenue Generation and Costreduction (Solutions for Capital and Operational costs)

From a network operator's perspective, the possibility of business paradigms enabled by SDN: A set of opportunities for network managers has opened through centralized control and management. Since the improvement and expansion of networks has now become easier, new business paradigms can be executed to maximize revenue.

Virtualize the network (networking as service): In a cloud computing module, computing and storage are supplied according to user demand, and networking resources should also be according to user demand. Thus, virtualizing can be a powerful tool enabled by SDN.

Less staff requirements: With small teams depending on automation and APIs, in order to operate highly configureured tasks, service providers and managers of a network are enabled by SDN instead of manually setting network devices.

Higher availability of service: Violations of the lower service level agreement are enabled by SDN, including insurance of availability, and as a logical unit, the network in its entirety can be abstracted and an effective distribution of resources becomes possible. Moreover, the client can by itself make SLA supply change without the interference of the network operator through the API. The benefits of compliance with the service level agreement (SLA) can be applied to supply security, speed of time for deployment, insurance of the uptime, used bandwidth and so on.

Energy consumption and cost: Optimizing the power utilization of the network and using green energy, i.e., energy maintenance by using renewable resources, can be conducted with assistance from centralized control [53].

5.1.9 SDN for Cloud Computing

Great benefits are provided by SDN. **First**, the deployment of devices of various vendors has become easier for the cloud provider. Conventionally, from the same vendor, providers of big clouds such as Google, Amazon, etc. have to have improved (high performance) routers and switches to easily reconfigureure the parameters of the routing update period, routing information base, etc. Devices of different sellers have their own advantages and disadvantages. However, the customization of each device is the greatest challenge because each seller might have a number of particular specifications including the syntax of the language. If each sellers' devices are pursuing the standards of SDN for rapid resource distribution or re-policy of the routing issues, a cloud provider is allowed. **Second**, through creating virtual flow slices, an SDN allows cloud users to apply scientific experiments or more effectively utilize cloud resources. The OpenFlow protocol harmonizes with the standards of GENI. This allows users arbitrarily to make slivers or slices without knowing anything about the substructure of the physical network. It is not important whether a substructure is wired or wireless. Moreover, it is not important how various storage units are deployed in different locations by cloud providers. With the concept of virtual flow, data flow is transparent along all cloud devices thanks to SDN.

An SDN is less expensive due to its being global and its provision of more management over network traffic flow and switching of data forwarding which follows confirmed standards when it is compared with the traditional devices of the network. Great SDN advantages include [35-42].

1) **Speed and Intelligence:** Through a powerful control panel, SDNs have some capacity for traffic load distribution optimization. This outcome in high speed transitions makes more effective use of resources.

2) **Easy Network Management:** Managers can be remotely in control of a network and they can also change the features of the network such as workload patterns, which is the connectivity based on service configureurations. With this advantage, managers can access the configureuration instantly and more effectively.

3) **Multi Tenancy:** The concept of an SDN can be extended as in data centers and data clouds across multiple partitions of a network. For example, across multiple partitions of a network, the concept of SDN can be extended to data clouds and data centers. For instance, cloud applications are required for deployment in virtual machines through multiple sites by multiple data center hirers. Cloud providers will want to ensure that each tenant has perfect cross-site execution isolation for improvement of traffic for a specific hirer. Network control ability of the inter-tenant and joint intra-tenant is not supported by

currently used cloud architectures. SDN supported cross-tenant data centers can be improved by taking advantage of the separation of the forwarding plane and control plane and visualization of the resource.

4) **Virtual Application Networks:** The virtualization of network resources utilized by Virtual application networks (such as distributed storage units, traffic queues in each router, etc.) as a result, the user's application avoids the low-level physical details. Thus, without any direct migration management problems through multiple data sites and resource separation, a network user can smoothly use the overall resources for distributed applications. A DOVE (distributed overlay virtual network) enables virtual application networks to be implemented by network managers. This helps via automation, transparency and virtualized network loads' better mobility [3] [34]. A large proportion of SDN is comprised of virtualization logic. The details of network devices, which are low level, can be prevented by virtualization. Moreover, network issues can be re-policied easily by users. This specification is also provided by virtualization. Many networks which are specific use virtualization. Inside the meaning of WSNs (wireless sensor networks), VITRO, a commendable European action, worked on virtualization. From the application, sensor deployment details are segregated by the virtual wireless sensor networks (WSN) concept [46]. Therefore, on same physical sensors, a multiple logic sensing application can be run. This allows multiple applications to be served by the same WSN.

5.1.10 SDN in heterogeneous networks

With the increasing numbers of network connectivity options, the interconnection and integration of non-homogeneous networks has become increasingly prominent. Hardware devices, transmission latencies, throughputs and protocols are examples of various heterogeneous network structures. Combinations increase network scalability and network coverage. Current network resources provide advantages to integration so as to reduce the cost of the operation and to improve competitiveness such as integration of the current power grid and SDN [23]. Moreover, the integration of a heterogeneous network will meet the needs of future network users. The traditional manner of integration and network interconnection is very complex. It consumes a great amount of time and it is error-prone.

Optical circuit network domains and packet network domains can be integrated through the OpenFlow control framework [24]. Both packets and circuit traffic are managed by a controller in an SDN. Therefore, it becomes a logical place to combine and manage heterogeneous networks. Nowadays, dynamic ability and different optical transmission technologies are supported by GMPLS (Generalized Multi-Protocol Label Switching). Because of that, GMPLS is the most improved available optical network management technology.

In Ref. [24], a unified network control plane, which is with OpenFlow and GMPLS, is introduced by the authors. If the flow tables of the network switches are empty, the OpenFlow controller receives the first packet entered for extension. The controller decides the flow's destination and sends the destination to the controller of the GMPLS because, for optical flows, this GMPLS controller computes a light route in relation to the end point. To send the remaining flows before the GMPLS controller finishes its task, the edge switch flow tables are updated by the OpenFlow controller, which has been extended.

There are also other research purposes of network interconnections and integration [25–27].

Moreover, for circuit and packet network integration, wireless networks have been redesigned by OpenRadio using the SDN concept [28]. Wireless network protocols, which are currently in use, are closely integrated with some special ASIC chips; they are also often the hardware replacement for any upgrades. The essence of OpenRadio design is the software abstraction layer, which is defined. To program MAC and PHY layers, a number of programmable interfaces are provided by OpenRadio. MAC and PHY layers can be used by operators. Operators can develop some policies for similar actions and some specific flows. Therefore, the protocol updates can be achieved. The most important aim is to design a wireless forwarding plane with a programmable characteristic that comprises routers, gateways, switches, and base stations, thereby developing cellular underlying devices that are software defined [29]. Large companies such as Huawei and Centec Networks employ an effective method to meet network integration and interconnection challenges. Centec Networks concentrates on the providers of the white box switch and SDN-switching-metal in China. To benefit open SDN architecture and its improved and high performance, Centec provides clients with networks to smoothly transfer to the new SDN track3 from the traditional architecture of MPLS/MPLS-TP, L2 and L3.

5.2 Network applications of SDN

Unique network controllability has been provided through the SDN architecture. A considerable number of SDN based applications have been brought through the liberation of controllability.

5.2.1 SDN network routing

In an SDN, when routing purposes with a holistic view are compared to traditional designs, it can be seen that there are many benefits of routing purposes with a holistic view. When multi-rooted tree structure topology is followed, the network of the data center generally connects its own nodes. Multi rooted tree topological features cannot be fully exploited with existing bandwidth improvements of single root forwarding, which is static between the host pairs. In this manner, a number of data centers use the Equal Cost Multi Path (ECMP) method, which increases the likelihood of system crashes and limiting the availability of the network.

Hedera [14] serves the multi-rooted tree topology with a dynamic flow scheduling system design. To gather the information of the flow, component switches are used. It calculates the unconflicted routes and to reroute the traffic, it commands the switches accordingly. To one or multi-rooted tree topologies, Hedera can be applied. Scheduling traffic by using a holistic view approach prevents network execution degeneration, which is caused by traditional multipath algorithms. To direct traffic to server instances, conventional data centers usually use load balancers which are dedicated front-end. Dedicated and expensive hardware needs this solution. An alternative method endeavors to provide the Plug-n-Serve [15] system, which has an SDN-based architecture. The controllers receive the first packet of all flows in order to check this system and define forwarding rules with respect to the relevant policy. Thus, in accordance with the currently used network conditions, it can perform dynamic load balancing. However, this solution suffers from scalability restrictions because of the deep participation of controllers for checking each flow's initial packet and the many rules in all switches. In Ref [16], an expandable proactive network load balancing approach is proposed. This design consists of two parts, namely the "partitioning algorithm" and the "transitioning algorithm." The purpose of the "partitioning algorithm" is to create wildcard policies. The purpose of the "transitioning algorithm" is to change one group of policies with another. The "partitioning algorithm" divides the traffic of the client in relation to the various weights depending on the dynamic features of the OpenFlow switches. While forwarding policies are applied by the system and to guarantee all of the same TCP link packets reach the same end point, it considers the client's IP. Controllers will keep the switch regulations while current TCP links are not finalized. The algorithm of "transitioning" utilizes two methods to achieve regulation transmissions: the initial one for a faster transition routes several packets for the controller; the second within a slower crossing cost enables the switch to deal with all packets. With less participation of controllers, this design proactively performs the mapping of the initial IP-to-server situations.

As well as load balancing, path improvements are another vital research subject. Data center applications, peer-to-peer and content distribution networks (CDNs) require server selection services [17]. The current application of the layer traffic optimization (ALTO) protocol supplies an abstraction outlook of network in the type of cost functions and network functions to support implementations to select the best server situations. Entire routing applications mentioned previously are restricted to related meshes or autonomous systems (AS). According to reference [18], to remove the AS boundary, a conveying service outsourcing paradigm is introduced to the same routing service operator. Numerous AS systems may outsource their transmitting control levels. This might create a logically centered transmitting control stage that may simplify the finalization of cross-domain transmission and execution.

5.2.2 SDN network security and access control

Nowadays, most actual SDNs apply to campus networks and data center company mesh structures, such as the Stanford premises network and Google data centers. By applying SDN in a multi-tenant open cloud, company meshes or bearer networks will experience difficulties in the active network structure acquired by the extensively utilized virtualization technology. Static status and the inherent physical limits of core mesh are substituted by SDN’s active and imaginary logical features. Thus, network safety in the cloud becomes overly complex by the dynamic distribution and control of the safety rules and elements, as well as to the response to network flow and the decision-making function [19]. The decoupling of the control layer in an SDN allows recent network protection paradigms.

In Ethane [12], a sole controller performs connection control by setting up various rules for individual switches. Resonance [20] to a new security framework for this design was expanded. It combines an observation infrasystem into the network operating system with an SDN controller. Similarly, network managers may implement protection methods from a holistic appearance with no necessity for coactions among individual switches. This makes active more fine-grained management together with more specification in addition to simplifying security rules performance.

Ref. [21] proposed FRESCO, an SDN network security application improvement framework. It enables deployment and sharing of security application modules for network security researchers. By creating different security monitoring and threat detection logics, this framework produces APIs and re-utilizable library modules. It includes an implementation layer and a protection execution core. The implementation layer performs an advancement surrounded simultaneously with a source controller liable to the infrastructures control. With FRESCO, when a packet is received, an instance will be triggered, and the security kernel receives the results of performance.

Recently, the works of creating new SDN protection models suffer from many defects. For example, Resonance is still lacking in sensitivity and scalability. In the FRESCO framework library, the number of available modules is only twelve [21]. Moreover, researchers are concerned about utilizing logical management centers in the SDN mesh to clarify network protection problems in the conventional mesh structure. Furthermore, when creating a new SDN protection paradigm, some works are available regarding attempts of tracking on SDN. Another important primary stage of many network attacks is IP and port scanning. In Ref. [9], for performance of IP address alteration to aid IP protection, OpenFlow is incidental to the change of host (OF-RHM) technology being offered. This technology performs the next two aims, which are difficult to achieve in the conventional meshes: 1) that address alterations are clear to term hosts, followed by 2) address alterations inability to be predicted. Within OF-RHM, the submesh gateway, being in charge of exchanging the right and fake directions and the controller, is liable to regulate the shaping of network address alterations for the entire network. SDN's logically centralized controller may change the orders on OpenFlow devices to regulate alterations efficiently, by containing the management of current links and DNS updates.

Unexpected surges of network flow in the data center or on the cloud platform can cause service failures or performance degradation. NetFuse [22] proposes an overload prevention mechanism for the SDN network flow. To detect active flows, it generally uses Open-Flow checking messages and then uses a multidimensional flow combining algorithm automatically to define an ideal group of flow piles and define the irregular flows.

5.2.3 Internet Research

Continuous Internet Updates brings many defenses that are continuously being utilized; it is difficult to examine fresh strategies and ideas to resolve existing issues in a current mesh. SDN technologies give a means to examine opinions for a new generation of the Internet with no change in the available mesh [44]. From the SDN's separated management and exchange of data with OpenFlow devices, it is very easy to separate hardware in distinction to software. This decoupling allows for testing with new forwarding designs; therefore, recent Internet structure designs may be examined.

Generally, the testing of new models of networks is difficult. From the time that new network models frequently use several dispatching schemes and content alternatives in non-standard ways, the differences are difficult to cover in existing networks. OpenFlow provides switches, routers and access points from several various vendors to use for decoupling of the management and data layers. These devices only transfer data packs according to the rules from the controller. The switch forwards the packet to the controller if the data pack reaches its goal and the switch does not have a criterion for it to determine what will be done with the packet, and if needed, the controller delivers a new criterion to the switch, thereby possibly similarly addressing any following data packets [43].

5.2.4 Mobile Device Offloading

Privacy is important for business apps as people often work with information that requires that information to remain secure. Some information can be distributed to only a small number of recipients, whereas other information does not need to be as secure. For example, in [45], writers used Enterprise-Centric Offloading System (ECOS) to allay those concerns. The purpose of creating ECOS was to offload data to empty PCs when guaranteeing the apps together with extra reliability needs, which are only offloaded in approved machines. The execution was kept in view of the consideration of many applications and PC users [45]. To select resources and manage the network, SDN is used. The resources that are selected have to be capable of compensating privacy demands. A controller will decide whether a device is capable of offloading, which takes into account any privacy demands when preserving energy. If no such device exists, the data does not provide for offloading from the portable tool. If preservation of energy is not required, afterwards some of the resources with sufficient capability are utilized if present. An OpenFlow switch is operated because the controller might control the flows. ECOS may discharge by paying attention to security demands even if there is no overly complicated design.

5.2.5 Wireless Virtual Machines

It is becoming increasingly common to run applications on wireless and virtual machines in organizations. These virtual machines enable organizations to have lower operational costs and be more resilient. There are demands to make them more portable in order to obtain the full potential of a virtual machine. In this process, the main concern of virtual machines is how to keep IP addresses. Present approaches to managing virtual machines

have not been sufficiently effective. Proposed resolutions in [46] involve utilizing an IP which is a mobile or dynamic DNS. The primary matter among the two resolutions consists of a manual reconfigureuration of network settings after removing the virtual machine. This restricts companies and datacenters when it is necessary simply to port the virtual machines to different locations. In [36], to solve the problem of virtual machine mobility, an application named Cross-Roads was proposed. Cross-Roads was created to enable on-line and off-line virtual machine mobility. There are three basic goals in Cross-Roads. The **initial** goal consists of enabling the management of flow from both data centers and external clients. **The next** goal is to manage the exploitation of OpenFlow assuming that every data center uses an OpenFlow controller. **The final** goal implies exploiting false MAC and IP addresses to have them continue if they are being ported and providing the actual IP to change.
Their fundamental software execution was performed according to the rules of building with the aim of scanning various networks to find virtual machines. The Cross-Roads controller is supposed to have records of actual MAC and IP addresses for virtual machines being run on its own network and also the controllers within every data center. If an application program which is running in a specific virtual machine is requested, the request is transmitted so as to be processed by the controllers. When a request for a virtual machine is received by the controller and if it is not on its list, the request is transmitted to another controller. The controller obtaining the actual address of the IP of the virtual machine subsequently receives a false MAC address transmitted to the initial controller, enabling the primary controller to upgrade its list if another request is received by it afterwards. In many various models of applications, SDN has been proved to be a helpful resource. SDN enables users' instant adaptation of networks in accordance with the new conditions and also to examine new protocols. In the majority of applications, OpenFlow was used because of its versatility. Data centers have been evolving to be a continuously essential component of many large corporations and on the Internet. Column mobile applications refer to cell phones, tablets, and other non-traditional media formats rather than laptops and other typical computing platforms. The cloud is used by several applications. Changes in hardware are difficult to perform within a traditional network,

which is essential since a model for shutting down in the process of an upgrade is needed. However, suitability for the upgrades is given by SDN in accordance with its decoupling of control planes and data [31].

5.3 Intent

One of the newest trends of SDN regarding traffic engineering is intent. This framework gives some permissions to an application regarding specifying its network control desires in the form of policy instead of as a mechanism. These directives, which are policy based, are referred to as **intents**. An intent is a constant modal object. The requests of an application are described to the SDN controller by this modal object. With these application requests, at the low levels, the behavior of the network is altered. The idea of intent-based networking is to tell the network controller what the requirements are and allow the controller to sort them in terms of being available and possible. Building policies determines the direct actions needed, and a user does nothing while the controller does everything.

5.3.1 Intent design specifications:

5.3.1.1 Intent is invariant: Some conditions change, such as in firmware upgrades, link fail downs, crashed servers, switch merchant changes, and cloud provider changes or any other infrastructural changes; these do not change the intent. This description shows that applications are independent from any underlying network details. Therefore, it simplifies developing, testing and deploying an application.

The basic intent has not yet been caught because by its design, it has a neutral infrastructure if the expression must be changed according to the state of the infrastructure.

5.3.1.2 Intent is portable: Intent, which describes the needs of an application, is independent from any type of media, merchants, protocols, infrastructure providers, etc. As mentioned previously, it is isolated from infrastructural changes. Thus, the effects of this type of change are eliminated by intent-based networking. Intent satisfies the needs of service providers, enterprises and telecom carriers because it is portable across many different solutions inclusive of the SDN controllers. Moreover, in consequence of inevitable infrastructural changes, changes to lengthy application integration and run time complexity are eliminated by intent.

5.3.1.3 Intent scales out, not up: If one were to switch from one domain of a network controller (which has millions of ports) to one-thousand network controllers (which have thousands of ports for every domain), the intent remains the same. One single intent description is sufficient for all of them. The scale out approach to building is enabled. For instance, small failures and maintenance domains are supported simultaneously by SDN domains. Nowadays, establishing scale up systems by using massive, single and clustered controller domains is assumed by many SDN architectures. However, it will entail a number of operational and deployment difficulties. When collective intent is effectively shared across an optional number of independent domains, end to end fulfillment is achieved. This allows high survivability of locally autonomous controllers and fault handling [66] [65] [67].

5.3.2 Intent Benefits:

1. It provides a high-level interface which focuses on what should be done instead of how it is programmed.
2. It isolates network complexity from applications.
3. It has easy production of complex functionality by combining other intents.
4. Operators and applications can describe policy instead of describing device instructions.
5. It is universal and non-prescriptive.
6. Implementations of a controller plane and application plane renders are independent of each other.
7. For network controller resources, intent systems are more secure in comparison to non-intent systems. In traditional software-defined networking (SDN) interfaces, the main vulnerabilities, such as manipulation, raw flow table, etc., are not possible in a system with only NBI; it must also be intent based.

8. Intents are more effective compared to solely installing flows. It keeps one’s intent when retaining the ability to install them.

5.4 ONOS implementation of intents

5.4.1 Open Network Operation System (ONOS)

ON.Lab (Open Networks Laboratory), a non-commercial foundation established in 2012 has produced ONOS, which is an open source SDN controller supported by the University of Berkeley and Stanford University to evolve the SDN controller of the ONOS type. ONOS is employed as an SDN controller used for telecommunication corporations as well as providers of service, focusing on great availability, performance and scalability. A set of OSGi Java application programs, which run as parts of Apache Karaf, communicate using REST APIs and Java APIs, which are used to execute the controller.

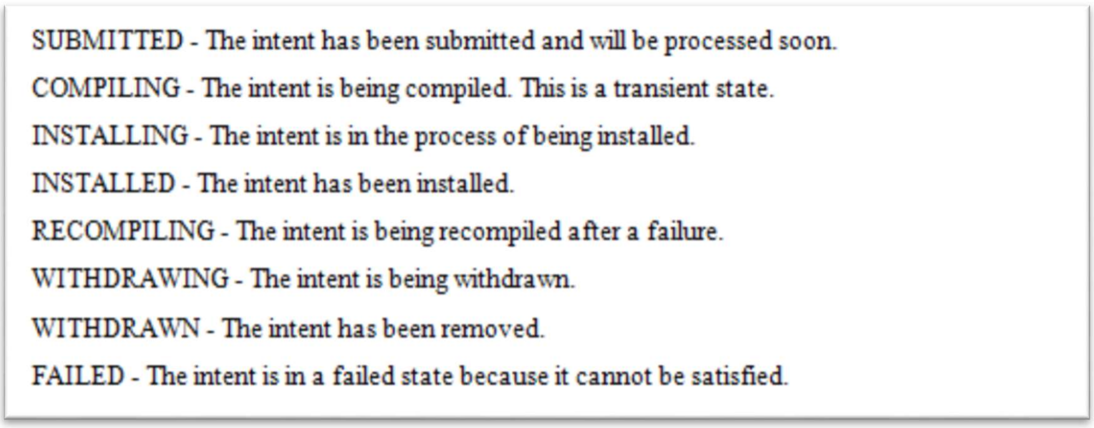
A wide range of devices are supported by ONOS with the aim of improving new applications based on the controller. The controller includes graphical user interfaces or reference patterns and an uncomplicated unification to the command line. ONOS presents a pair of major interfaces being northbound: the topology view of the holistic network and framework of the intent. Controller API is utilized by applications for the purpose of calculating paths, flows of provisions and execution of other processes in a network. The case of distributed controllers is the most important feature in ONOS. The topology view is the primary concern for maintaining the view consistency. Every distributed controller maintains a holistic view and with another randomly selected controller instance, the controller is synchronized. The intents of ONOS are used as an abstract concept to show the necessity of connecting several nodes; however, they do not define specific routes between such nodes. After establishing an intent, the optimal route is computed by the controller, which also configureures data plane devices to maintain important paths or flows in order to perform the intent. The controller should, under dynamic conditions, update the flows of the data plane devices if the topology of the network changes.

For network management, ONOS supplies a Command Line Interface (CLI), REST API and a graphical, web-enabled GUI interface. Essentially, REST API is utilized by developers, but it can be utilized in daily routine management of networks (with minor issues). ONOS CLI enables Apache Karaf CLI for version extensions, with access after being authorized to the major functions of network such as managing data-flows on switches as well as providing components of the network for instance hosts and switches. For network managers, the CLI also provides for a framework of intent in a suitable manner. Presenting illustrations of the topology of a network in a graphical form and enabling simple setups (without changing or removing) of intents are the main functions of the GUI [64] [65].

5.4.2 ONOS implementation of Intents

ONLAB was the first SDN company to implement the notion of intents. Even more, the ONOS developer team took this one step further, allowing the application itself to make its intent known. This intent is treated as an object inside the ONOS controller, comprised of elements such as network resources, constraints, criteria and instructions. The figureure below illustrates the compilation process of each intent implemented by ONOS [65]:

As described on the ONOS website, an intent shares the following statuses:



SUBMITTED - The intent has been submitted and will be processed soon.

COMPILING - The intent is being compiled. This is a transient state.

INSTALLING - The intent is in the process of being installed.

INSTALLED - The intent has been installed.

RECOMPILING - The intent is being recompiled after a failure.

WITHDRAWING - The intent is being withdrawn.

WITHDRAWN - The intent has been removed.

FAILED - The intent is in a failed state because it cannot be satisfied.

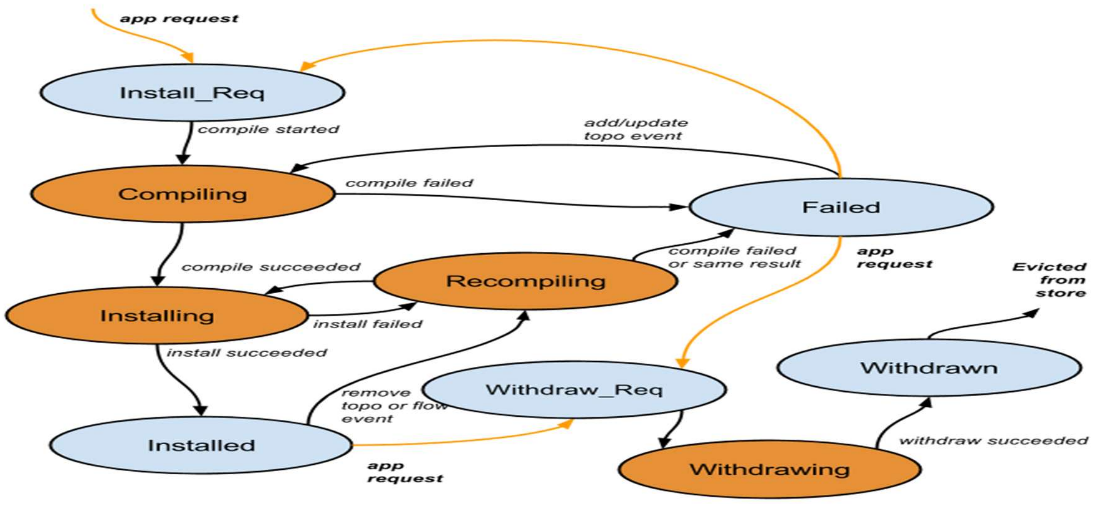


Figure (12) ONOS Iintent Framework

5.4.3 How the ONOS Intent Framework works

- ONOS provides an application intent framework as the NBI to describe network connectivity as network policy. It is a sub-system of ONOS.
- Intents in the framework are organized into a hierarchy; developers will select the intents that are closest to their network models.
- Each intent has its compiler to compile the intent into flow rules; flow rules are installed as table entries in network devices based on the OpenFlow protocol.
- Coordinates and verifies installation of instructions.
- Responds to network conditions that are changing.
- Gives permission to the optimization of intent objects.

- Allows dynamic changes in functionality (e.g., compilers, installers) or adds functionality to the Intent Framework.

CHAPTER 6
SIMULATION AND PERFORMANCE

Experiment performance

In this section, we present the benefits provided by SDN to traditional networks. We selected a single IP layer and a multilayer optical network (MPLS over DWDM) to demonstrate the advantages of SDN managements. We employed an SDN controller that can assess a description of how to configureure the test environment and show the benefits. The SDN controller used was the ONOS controller.

6.1 Test Environment

We took advantage of virtualization to run test cases. The Ubuntu 14.04 operating system was running in a virtual machine using VirtualBox 5.1.22. Depending on the test case, we created a realistic virtual network using Mininet, which allows users to create SDN networks with different topologies, examined with OpenFlow. The ONOS controller was used to manage the network. Other software requirements were Python to execute the network topology. Oracle Java 8 JDK, Apache Maven, Apache Karaf, git, and bash (for packaging and testing) were used to run the controller.

6.2 Prerequisites and Setup for ONOS

6.2.1 Installing VirtualBox, Ubuntu and Mininet

At first, we downloaded the Virtual Machine. We can use either Oracle VM VirtualBox or VMware Player. There is no problem in either case since we will obtain the same result. We will not explain in detail the installation method of Oracle VM VirtualBox, Ubuntu 14.04 and Mininet. For the download and installation, we can follow the links in references [81], [82], [83] and [84].

6.2.2 Installing ONOS

6.2.2.1 Installation of Maven, Karaf and Java

The ONOS set up method depended on the circumference which was changeable. JAVA_HOME was appropriately installed. That is, either mvn-type or java-type has to be the same as the Java version:

Install Karaf, Maven:

~/Downloads and ~/Applications should be created. Download and extract into the file that is ~/Downloads and extract it to ~/Applications the, Karaf3.0.5, and Maven 3.3.9 binaries.

```
build:~$ cd; mkdir Downloads Applications

build:~$ cd Downloads

build:~$ wget http://archive.apache.org/dist/karaf/3.0.5/apache-karaf-3.0.5.tar.gz

build:~$ wget http://archive.apache.org/dist/maven/maven-3/3.3.9/binaries/apache-
maven-3.3.9-bin.tar.gz

build:~$ tar -zxvf apache-karaf-3.0.5.tar.gz -C ../Applications/

build:~$ tar -zxvf apache-maven-3.3.9-bin.tar.gz -C ../Applications/
```

Afterwards, set up “Oracle-Java-8”:

```
build:~$ sudo apt-get install software-properties-common -y

build:~$ sudo add-apt-repository ppa:webupd8team/java -y

build:~$ sudo apt-get update

build:~$ sudo apt-get install oracle-java8-installer oracle-java8-set-default -y
```

Upgrading the installation of Java 8:

```
$ su -
$ echo "deb http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main" | tee
/etc/apt/sources.list.d/webupd8team-java.list
$ echo "deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main" | tee
-a /etc/apt/sources.list.d/webupd8team-java.list
$ apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys EEA14886
$ apt-get update
$ apt-get install oracle-java8-installer oracle-java8-set-default -y
$ exit
```

Setting “JAVA_HOME”

```
$ /usr/libexec/java_home
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home
```

If “JAVA_HOME” is not configureured, then false, or execute, and put in the command below:

```
$ export JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

ONOS building using Maven:

To manage the build process, ONOS uses Maven. For ONOS code-base building from the top level, just type the following:

```
$ cd ~/onos
$ mvn clean install
```

6.3 Test scenarios

In this section, we will test the implementation of SDN based networking in two scenarios, the first of which is the packet IP environment including a one-layer network topology. The second was an SDN over the optical network environment including two layers (MPLS/DWDM) of network topology. We demonstrated the benefits and flexibility produced by SDN as follows:

6.3.1 Packet IP environment

To perform this scenario, we prepared a network topology containing the part of 6 switches whose edge switches are attached to 2 core switches, the place where they are attached to each other. Figure (13) below demonstrates the topology. Every side switch has 6 hosts which are attached to it. Everything was working using the OpenFlow 1.3 protocol.

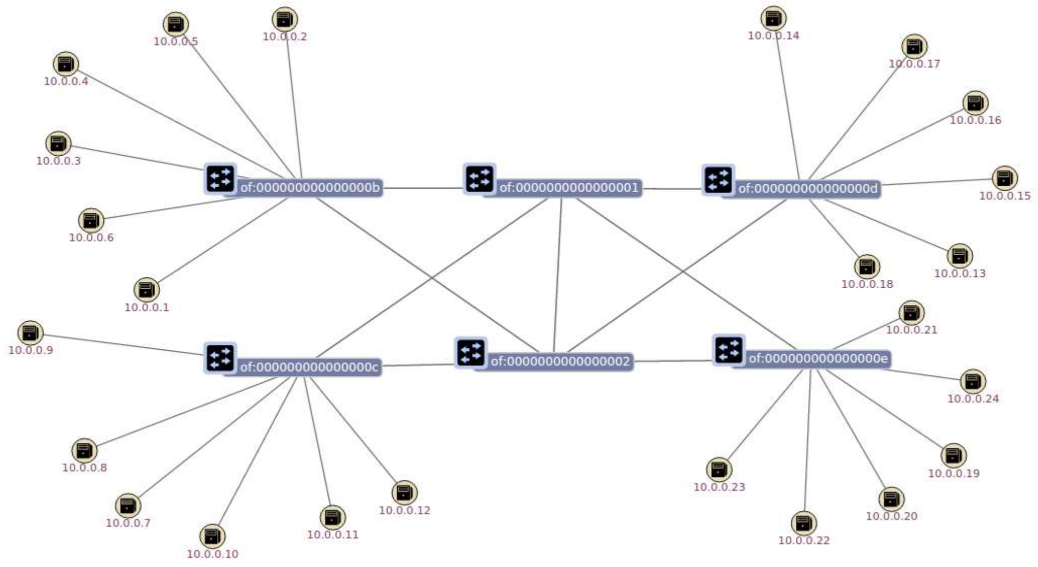
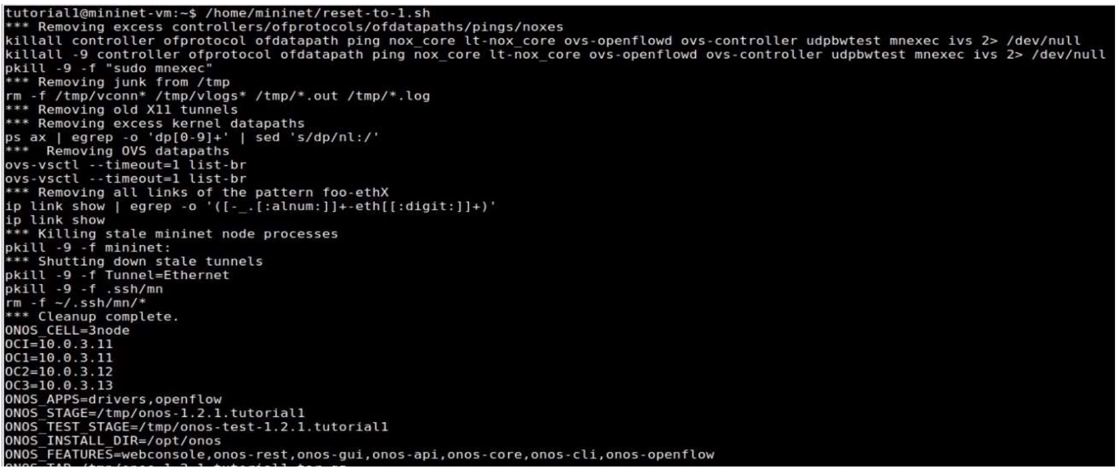


Figure (13) Network Topology

Running ONOS controller

Before running the ONOS controller, we reset the controller to guarantee that the surroundings were pure and complete. The resetting was simple and we just typed the following script in the terminal (Figure (14)):

```
$ /home/mininet/reset-to-1.sh
```

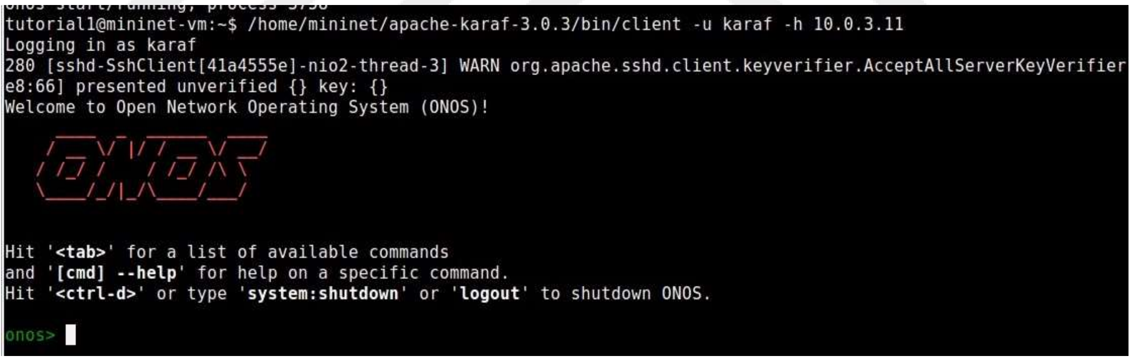


```
tutorial1@mininet-vm:~$ /home/mininet/reset-to-1.sh
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
ONOS_CELL=3node
OCI=10.0.3.11
OC1=10.0.3.11
OC2=10.0.3.12
OC3=10.0.3.13
ONOS_APPS=drivers,openflow
ONOS_STAGE=/tmp/onos-1.2.1.tutorial1
ONOS_TEST_STAGE=/tmp/onos-test-1.2.1.tutorial1
ONOS_INSTALL_DIR=/opt/onos
ONOS_FEATURES=webconsole,onos-rest,onos-gui,onos-api,onos-core,onos-cli,onos-openflow
```

Figure (14) ONOS Resetting

We then ran the ONOS controller by typing the following script into the terminal (Figure (15)):

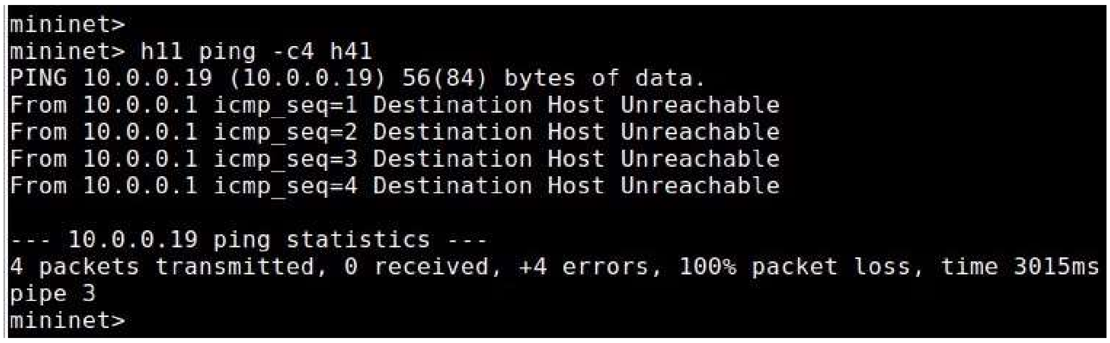
```
$ /home/mininet/apache-karaf-3.0.3/client -u karaf -h 10.0.3.11
```



```
tutorial1@mininet-vm:~$ /home/mininet/apache-karaf-3.0.3/bin/client -u karaf -h 10.0.3.11
Logging in as karaf
280 [sshd-SshClient[41a4555e]-nio2-thread-3] WARN org.apache.sshd.client.keyverifier.AcceptAllServerKeyVerifier
e8:66] presented unverified {} key: {}
Welcome to Open Network Operating System (ONOS)!

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

onos> 
```

Figure (15) Running ONOS

The topology was connected to the controller, but there was no communication between the hosts. We could see that by using a ping between any two hosts in the Mininet prompt as follows (Figure (16)):

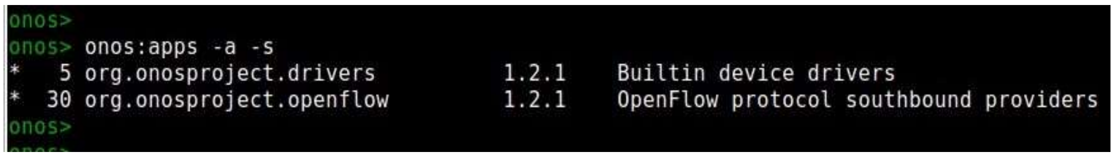


```
mininet>
mininet> h11 ping -c4 h41
PING 10.0.0.19 (10.0.0.19) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable

--- 10.0.0.19 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3015ms
pipe 3
mininet>
```

Figure (16) No Connection between Hosts

In order to create the connections between the hosts and switches, we used two methods. First, we used the Reactive Forwarding application (fwd) by activating it from the controller command line interface (CLI). However, before activating the application, we checked the active application in the controller prompt, as in the following (Figures. (17) and (18):

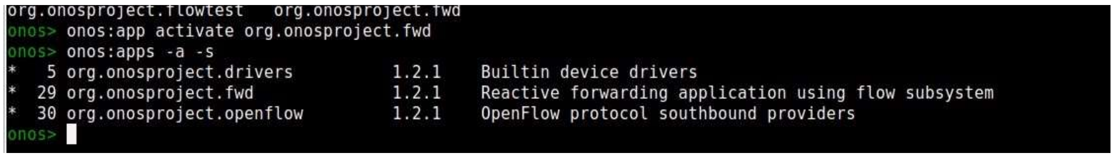
```
Onos>onos:apps -a -s
```



```
onos>
onos> onos:apps -a -s
*   5 org.onosproject.drivers          1.2.1    Builtin device drivers
*  30 org.onosproject.openflow         1.2.1    OpenFlow protocol southbound providers
onos>
```

Figure (17) ONOS Startup Applications

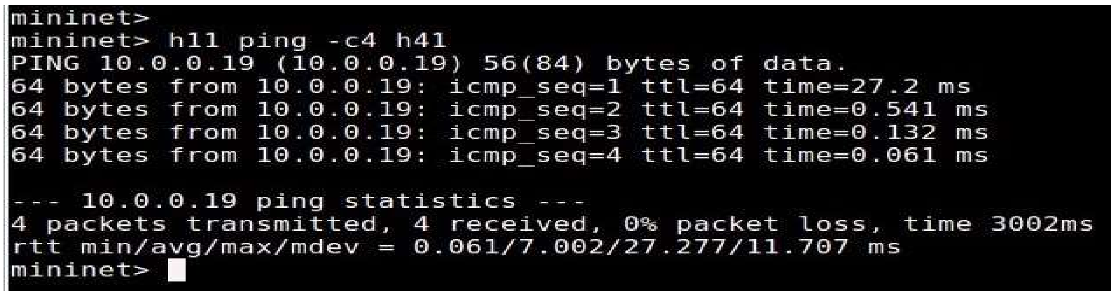
```
Onos>onos:app activate org.onosproject.fwd
```



```
org.onosproject.flowtest   org.onosproject.fwd
onos> onos:app activate org.onosproject.fwd
onos> onos:apps -a -s
*   5 org.onosproject.drivers         1.2.1    Builtin device drivers
*  29 org.onosproject.fwd             1.2.1    Reactive forwarding application using flow subsystem
*  30 org.onosproject.openflow        1.2.1    OpenFlow protocol southbound providers
onos> █
```

Figure (18) Reactive Forwarding application (fwd) Activation

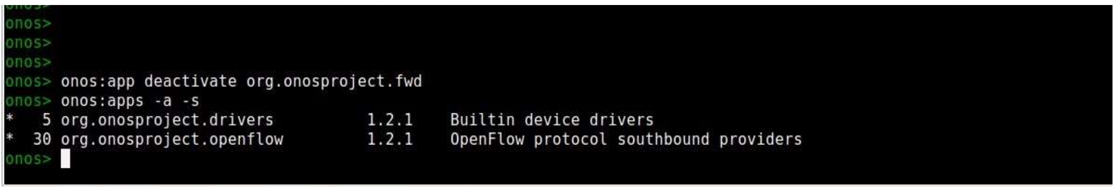
Now, the connections are created and we can check it by using a ping between any two hosts in the Mininet prompt, as follows (Figure (19)):



```
mininet>
mininet> h11 ping -c4 h41
PING 10.0.0.19 (10.0.0.19) 56(84) bytes of data.
64 bytes from 10.0.0.19: icmp_seq=1 ttl=64 time=27.2 ms
64 bytes from 10.0.0.19: icmp_seq=2 ttl=64 time=0.541 ms
64 bytes from 10.0.0.19: icmp_seq=3 ttl=64 time=0.132 ms
64 bytes from 10.0.0.19: icmp_seq=4 ttl=64 time=0.061 ms

--- 10.0.0.19 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.061/7.002/27.277/11.707 ms
mininet>
```

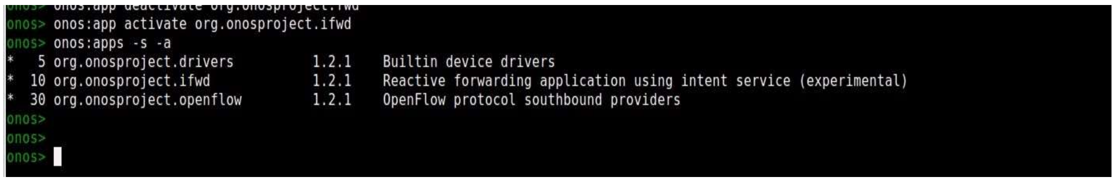
Figure (19) Connection is Created

We used the intent feature in two ways: first, if we want to add connections between all the hosts in one command, it was by using the Intent Reactive Forwarding application (ifwd) and activating it from the controller command line interface (CLI). Instead of pushing flow entries for every packet it sees, the intent reactive forwarding application provides an intent.. In particular, it provides a host-to-host intent which is an easy connectivity intent which allows connectivity between two hosts. However, before active application, we deactivated the Reactive Forwarding application (fwd) in the controller prompt, as follows (Figure (20) and (21)):



```
onos>
onos>
onos>
onos>
onos> onos:app deactivate org.onosproject.fwd
onos> onos:apps -a -s
*   5 org.onosproject.drivers         1.2.1    Builtin device drivers
*  30 org.onosproject.openflow        1.2.1    OpenFlow protocol southbound providers
onos>
```

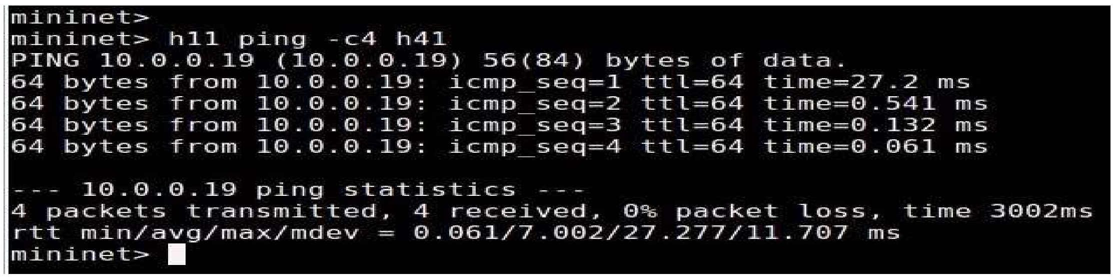
Figure (20) Deactivate Reactive Forwarding application (fwd)



```
onos> onos:app deactivate org.onosproject.fwd
onos> onos:app activate org.onosproject.ifwd
onos> onos:apps -s -a
*   5 org.onosproject.drivers         1.2.1    Builtin device drivers
*  10 org.onosproject.ifwd            1.2.1    Reactive forwarding application using intent service (experimental)
*  30 org.onosproject.openflow        1.2.1    OpenFlow protocol southbound providers
onos>
onos>
onos>
```

Figure (21) Activate Intent Reactive Forwarding application (ifwd)

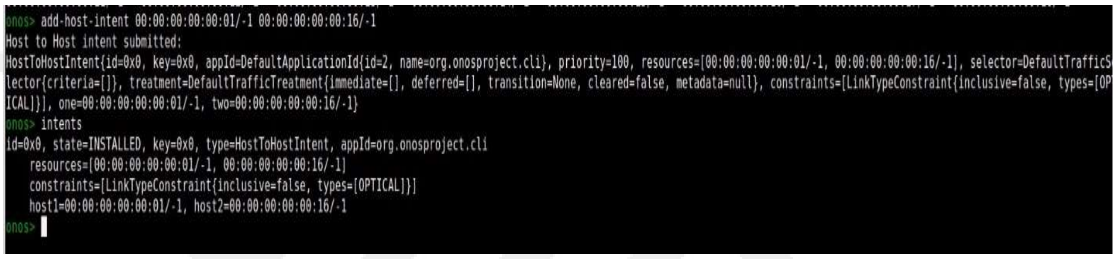
Now, the connections are created and we can check them by using a ping between any two hosts in the Mininet prompt, as follows (Figure (22)):



```
mininet>
mininet> h11 ping -c4 h41
PING 10.0.0.19 (10.0.0.19) 56(84) bytes of data.
64 bytes from 10.0.0.19: icmp_seq=1 ttl=64 time=27.2 ms
64 bytes from 10.0.0.19: icmp_seq=2 ttl=64 time=0.541 ms
64 bytes from 10.0.0.19: icmp_seq=3 ttl=64 time=0.132 ms
64 bytes from 10.0.0.19: icmp_seq=4 ttl=64 time=0.061 ms

--- 10.0.0.19 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.061/7.002/27.277/11.707 ms
mininet>
```

Figure (22) Connection is Created

For the second way, we used the add host intent by using the selected two hosts and typing the script in the controller prompt, as follows (Figure (23)):



```
onos> add-host-intent 00:00:00:00:00:01/-1 00:00:00:00:00:16/-1
Host to Host intent submitted:
HostToHostIntent{id=0x0, key=0x0, appId=DefaultApplicationId{id=2, name=org.onosproject.cli}, priority=100, resources=[00:00:00:00:00:01/-1, 00:00:00:00:00:16/-1], selector=DefaultTrafficS
lector{criteria=[]}, treatment=DefaultTrafficTreatment{immediate=[], deferred=[], transition=None, cleared=false, metadata=null}, constraints=[LinkTypeConstraint{inclusive=false, types=[OP
ICAL]}], one=00:00:00:00:00:01/-1, two=00:00:00:00:00:16/-1}
onos> intents
id=0x0, state=INSTALLED, key=0x0, type=HostToHostIntent, appId=org.onosproject.cli
    resources=[00:00:00:00:00:01/-1, 00:00:00:00:00:16/-1]
    constraints=[LinkTypeConstraint{inclusive=false, types=[OPTICAL]}]
    host1=00:00:00:00:00:01/-1, host2=00:00:00:00:00:16/-1
onos>
```

Figure (23) Host to Host Iintent Creation

The most important features of intent is that it is **invariant**; that is, it does not change as a result of a link going down, a server crashing, changing cloud providers, changing switch vendors, upgrading firmware, or any other change to the infrastructure. In the following figureures, we stopped one link between two switches, and the intent automatically changed its path to another link without cutting the connection, (as in Figures. (24) and (25)).

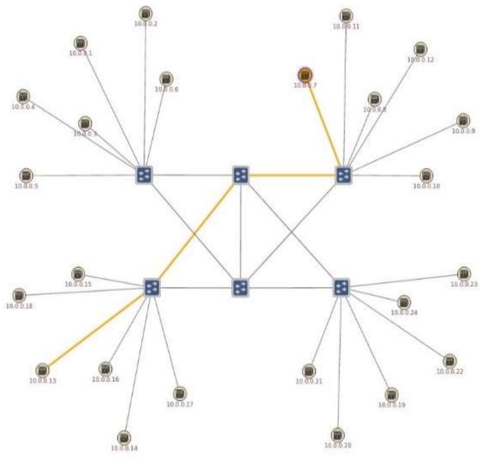


Figure (24) Intent Path

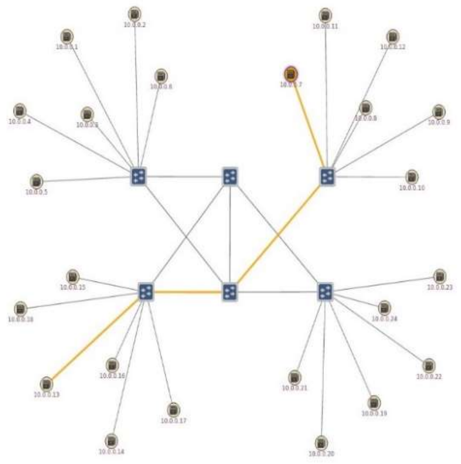
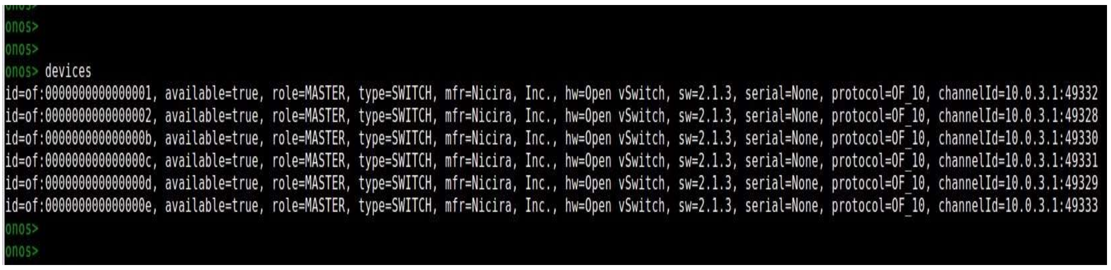


Figure (25) Intent Path after Stopping the link

Through the logical centralization of the controller with the global view of the network infrastructure devices, we can check and configureure devices, links, path costs between two nodes, ports, flows in each device, hosts and every physical device in the network topology by using the controller (CLI), as in the figureures below:

- The SDN Controller might have been of no use if there had not been any devices to manage. ONOS can have suitable commands to show existing tools within the given the system.

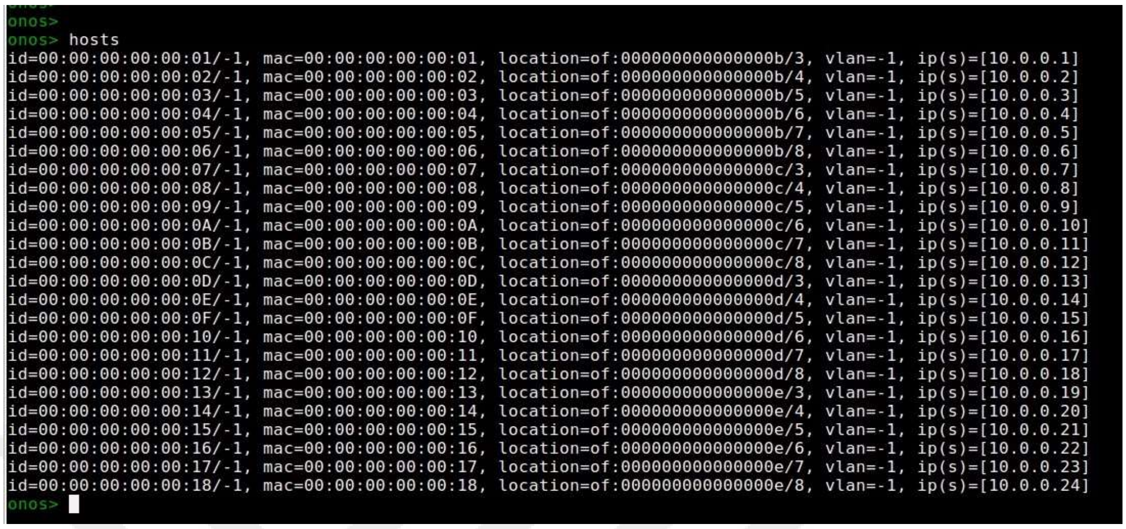


```
onos>
onos>
onos> devices
id=of:0000000000000001, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:49332
id=of:0000000000000002, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:49328
id=of:000000000000000b, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:49330
id=of:000000000000000c, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:49331
id=of:000000000000000d, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:49329
id=of:000000000000000e, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:49333
onos>
onos>
```

Figure (26) Devices Information

The command line output contains device IDs and Boolean values, and indications of whether these tools are now up. It also displays the types of devices and the roles of their respective connections by using ONOS as an example in this case.

- ONOS has the capability of showing the hosts existing in the system,

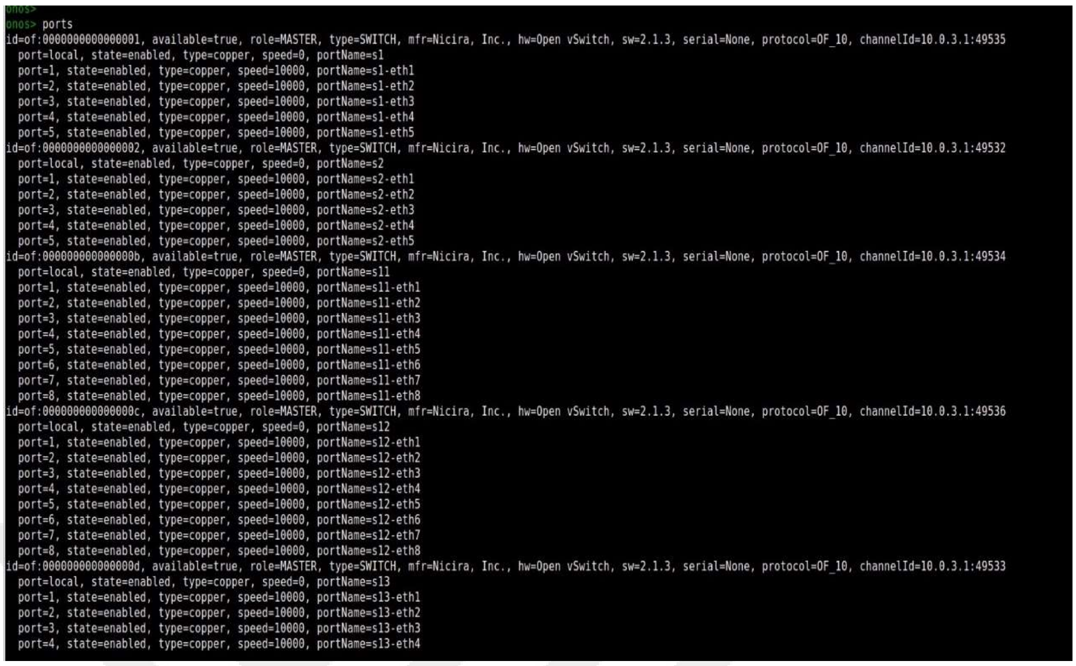


```
onos>
onos> hosts
id=00:00:00:00:00:01/-1, mac=00:00:00:00:00:01, location=of:000000000000000b/3, vlan=-1, ip(s)=[10.0.0.1]
id=00:00:00:00:00:02/-1, mac=00:00:00:00:00:02, location=of:000000000000000b/4, vlan=-1, ip(s)=[10.0.0.2]
id=00:00:00:00:00:03/-1, mac=00:00:00:00:00:03, location=of:000000000000000b/5, vlan=-1, ip(s)=[10.0.0.3]
id=00:00:00:00:00:04/-1, mac=00:00:00:00:00:04, location=of:000000000000000b/6, vlan=-1, ip(s)=[10.0.0.4]
id=00:00:00:00:00:05/-1, mac=00:00:00:00:00:05, location=of:000000000000000b/7, vlan=-1, ip(s)=[10.0.0.5]
id=00:00:00:00:00:06/-1, mac=00:00:00:00:00:06, location=of:000000000000000b/8, vlan=-1, ip(s)=[10.0.0.6]
id=00:00:00:00:00:07/-1, mac=00:00:00:00:00:07, location=of:000000000000000c/3, vlan=-1, ip(s)=[10.0.0.7]
id=00:00:00:00:00:08/-1, mac=00:00:00:00:00:08, location=of:000000000000000c/4, vlan=-1, ip(s)=[10.0.0.8]
id=00:00:00:00:00:09/-1, mac=00:00:00:00:00:09, location=of:000000000000000c/5, vlan=-1, ip(s)=[10.0.0.9]
id=00:00:00:00:00:0A/-1, mac=00:00:00:00:00:0A, location=of:000000000000000c/6, vlan=-1, ip(s)=[10.0.0.10]
id=00:00:00:00:00:0B/-1, mac=00:00:00:00:00:0B, location=of:000000000000000c/7, vlan=-1, ip(s)=[10.0.0.11]
id=00:00:00:00:00:0C/-1, mac=00:00:00:00:00:0C, location=of:000000000000000c/8, vlan=-1, ip(s)=[10.0.0.12]
id=00:00:00:00:00:0D/-1, mac=00:00:00:00:00:0D, location=of:000000000000000d/3, vlan=-1, ip(s)=[10.0.0.13]
id=00:00:00:00:00:0E/-1, mac=00:00:00:00:00:0E, location=of:000000000000000d/4, vlan=-1, ip(s)=[10.0.0.14]
id=00:00:00:00:00:0F/-1, mac=00:00:00:00:00:0F, location=of:000000000000000d/5, vlan=-1, ip(s)=[10.0.0.15]
id=00:00:00:00:00:10/-1, mac=00:00:00:00:00:10, location=of:000000000000000d/6, vlan=-1, ip(s)=[10.0.0.16]
id=00:00:00:00:00:11/-1, mac=00:00:00:00:00:11, location=of:000000000000000d/7, vlan=-1, ip(s)=[10.0.0.17]
id=00:00:00:00:00:12/-1, mac=00:00:00:00:00:12, location=of:000000000000000d/8, vlan=-1, ip(s)=[10.0.0.18]
id=00:00:00:00:00:13/-1, mac=00:00:00:00:00:13, location=of:000000000000000e/3, vlan=-1, ip(s)=[10.0.0.19]
id=00:00:00:00:00:14/-1, mac=00:00:00:00:00:14, location=of:000000000000000e/4, vlan=-1, ip(s)=[10.0.0.20]
id=00:00:00:00:00:15/-1, mac=00:00:00:00:00:15, location=of:000000000000000e/5, vlan=-1, ip(s)=[10.0.0.21]
id=00:00:00:00:00:16/-1, mac=00:00:00:00:00:16, location=of:000000000000000e/6, vlan=-1, ip(s)=[10.0.0.22]
id=00:00:00:00:00:17/-1, mac=00:00:00:00:00:17, location=of:000000000000000e/7, vlan=-1, ip(s)=[10.0.0.23]
id=00:00:00:00:00:18/-1, mac=00:00:00:00:00:18, location=of:000000000000000e/8, vlan=-1, ip(s)=[10.0.0.24]
onos>
```

Figure (27) Host Information

which illustrates the hosts' IPs as the same as their own respective MAC IDs, IPs, and the locations in the network in which they are linked. The sign '-1' within that ID domain is used to illustrate vlan knowledge, but here there is no vlan.

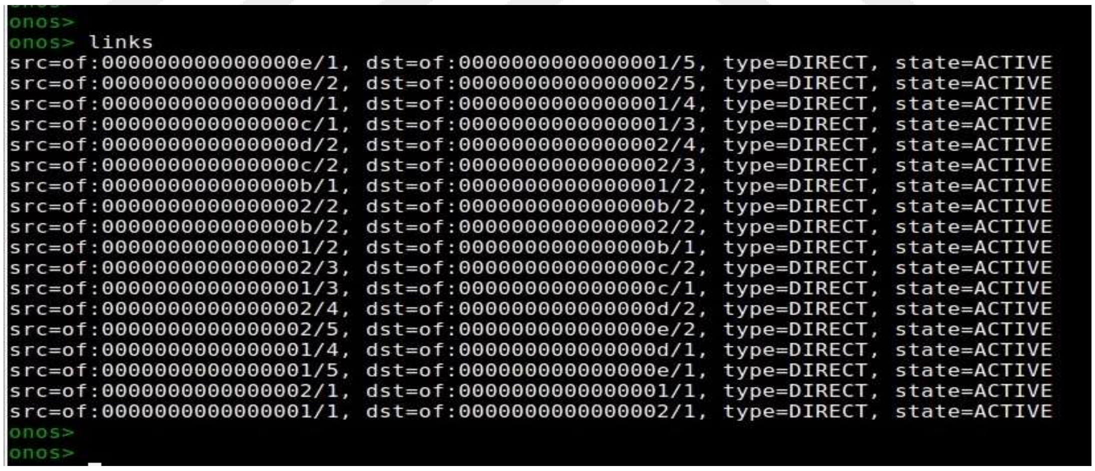
- ONOS has the capability of showing the ports of each device in the system containing the information for each device and their respective port connections.



```
onos>
onos> ports
id=of:0000000000000001, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:49535
  port=local, state=enabled, type=copper, speed=0, portName=s1
  port=1, state=enabled, type=copper, speed=10000, portName=s1-eth1
  port=2, state=enabled, type=copper, speed=10000, portName=s1-eth2
  port=3, state=enabled, type=copper, speed=10000, portName=s1-eth3
  port=4, state=enabled, type=copper, speed=10000, portName=s1-eth4
  port=5, state=enabled, type=copper, speed=10000, portName=s1-eth5
id=of:0000000000000002, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:49532
  port=local, state=enabled, type=copper, speed=0, portName=s2
  port=1, state=enabled, type=copper, speed=10000, portName=s2-eth1
  port=2, state=enabled, type=copper, speed=10000, portName=s2-eth2
  port=3, state=enabled, type=copper, speed=10000, portName=s2-eth3
  port=4, state=enabled, type=copper, speed=10000, portName=s2-eth4
  port=5, state=enabled, type=copper, speed=10000, portName=s2-eth5
id=of:000000000000000b, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:49534
  port=local, state=enabled, type=copper, speed=0, portName=s11
  port=1, state=enabled, type=copper, speed=10000, portName=s11-eth1
  port=2, state=enabled, type=copper, speed=10000, portName=s11-eth2
  port=3, state=enabled, type=copper, speed=10000, portName=s11-eth3
  port=4, state=enabled, type=copper, speed=10000, portName=s11-eth4
  port=5, state=enabled, type=copper, speed=10000, portName=s11-eth5
  port=6, state=enabled, type=copper, speed=10000, portName=s11-eth6
  port=7, state=enabled, type=copper, speed=10000, portName=s11-eth7
  port=8, state=enabled, type=copper, speed=10000, portName=s11-eth8
id=of:000000000000000c, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:49536
  port=local, state=enabled, type=copper, speed=0, portName=s12
  port=1, state=enabled, type=copper, speed=10000, portName=s12-eth1
  port=2, state=enabled, type=copper, speed=10000, portName=s12-eth2
  port=3, state=enabled, type=copper, speed=10000, portName=s12-eth3
  port=4, state=enabled, type=copper, speed=10000, portName=s12-eth4
  port=5, state=enabled, type=copper, speed=10000, portName=s12-eth5
  port=6, state=enabled, type=copper, speed=10000, portName=s12-eth6
  port=7, state=enabled, type=copper, speed=10000, portName=s12-eth7
  port=8, state=enabled, type=copper, speed=10000, portName=s12-eth8
id=of:000000000000000d, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:49533
  port=local, state=enabled, type=copper, speed=0, portName=s13
  port=1, state=enabled, type=copper, speed=10000, portName=s13-eth1
  port=2, state=enabled, type=copper, speed=10000, portName=s13-eth2
  port=3, state=enabled, type=copper, speed=10000, portName=s13-eth3
  port=4, state=enabled, type=copper, speed=10000, portName=s13-eth4
```

Figure (28) Ports Fnformation

• ONOS has the capability of showing the links existing in the system:

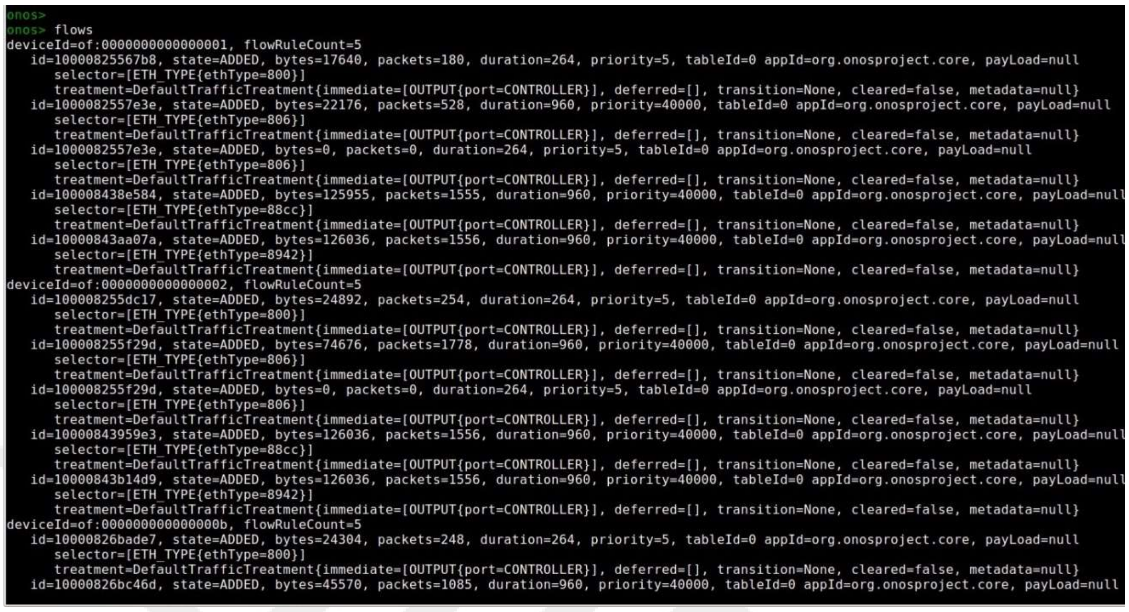


```
onos>
onos> links
src=of:000000000000000e/1, dst=of:0000000000000001/5, type=DIRECT, state=ACTIVE
src=of:000000000000000e/2, dst=of:0000000000000002/5, type=DIRECT, state=ACTIVE
src=of:000000000000000d/1, dst=of:0000000000000001/4, type=DIRECT, state=ACTIVE
src=of:000000000000000c/1, dst=of:0000000000000001/3, type=DIRECT, state=ACTIVE
src=of:000000000000000d/2, dst=of:0000000000000002/4, type=DIRECT, state=ACTIVE
src=of:000000000000000c/2, dst=of:0000000000000002/3, type=DIRECT, state=ACTIVE
src=of:000000000000000b/1, dst=of:0000000000000001/2, type=DIRECT, state=ACTIVE
src=of:0000000000000002/2, dst=of:000000000000000b/2, type=DIRECT, state=ACTIVE
src=of:000000000000000b/2, dst=of:0000000000000002/2, type=DIRECT, state=ACTIVE
src=of:0000000000000001/2, dst=of:000000000000000b/1, type=DIRECT, state=ACTIVE
src=of:0000000000000002/3, dst=of:000000000000000c/2, type=DIRECT, state=ACTIVE
src=of:0000000000000001/3, dst=of:000000000000000c/1, type=DIRECT, state=ACTIVE
src=of:0000000000000002/4, dst=of:000000000000000d/2, type=DIRECT, state=ACTIVE
src=of:0000000000000002/5, dst=of:000000000000000e/2, type=DIRECT, state=ACTIVE
src=of:0000000000000001/4, dst=of:000000000000000d/1, type=DIRECT, state=ACTIVE
src=of:0000000000000001/5, dst=of:000000000000000e/1, type=DIRECT, state=ACTIVE
src=of:0000000000000002/1, dst=of:0000000000000001/1, type=DIRECT, state=ACTIVE
src=of:0000000000000001/1, dst=of:0000000000000002/1, type=DIRECT, state=ACTIVE
onos>
onos>
```

Figure (29) Links Information

The output presents a group of found links. These related links are formed by a resource node-port couple for ending node-port two. The 'type' field shows whether the connection is a direct contact between these two devices.

• ONOS has the capability of showing the flows existing in the system:

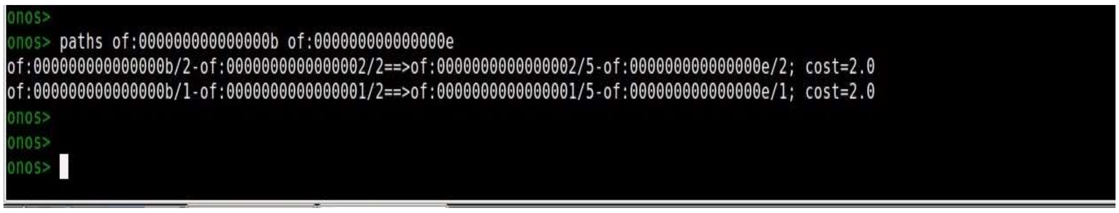


```
onos>
onos> flows
deviceId=of:0000000000000001, flowRuleCount=5
   id=10000825567b8, state=ADDED, bytes=17640, packets=180, duration=264, priority=5, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=800}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=1000082557e3e, state=ADDED, bytes=22176, packets=528, duration=960, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=806}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=1000082557e3e, state=ADDED, bytes=0, packets=0, duration=264, priority=5, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=806}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=100008438e584, state=ADDED, bytes=125955, packets=1555, duration=960, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=88cc}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=10000843aa07a, state=ADDED, bytes=126036, packets=1556, duration=960, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=8942}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
deviceId=of:0000000000000002, flowRuleCount=5
   id=100008255dc17, state=ADDED, bytes=24892, packets=254, duration=264, priority=5, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=800}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=100008255f29d, state=ADDED, bytes=74676, packets=1778, duration=960, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=806}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=100008255f29d, state=ADDED, bytes=0, packets=0, duration=264, priority=5, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=806}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=10000843959e3, state=ADDED, bytes=126036, packets=1556, duration=960, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=88cc}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=10000843b14d9, state=ADDED, bytes=126036, packets=1556, duration=960, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=8942}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
deviceId=of:000000000000000b, flowRuleCount=5
   id=10000826bade7, state=ADDED, bytes=24304, packets=248, duration=264, priority=5, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=800}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=10000826bc46d, state=ADDED, bytes=45570, packets=1085, duration=960, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
```

Figure (30) Flows Information

ONOS gives several details regarding flows on switches. For instance, any flow input will determine the dimmer switch and the method that is a collection of the traffic met via a flow input as well as how the traffic has to be operated. For all flow inputs that are followed via an appId (app Id), the appId knows that the app settled the flow input. This is a beneficial characteristic due to its possible support of the admin to recognize that the application could be misbehaving or wasting resources.

• ONOS has the capability of showing path costs between existing devices in the system:



```
onos>
onos> paths of:000000000000000b of:000000000000000e
of:000000000000000b/2-of:0000000000000002/2==>of:0000000000000002/5-of:000000000000000e/2; cost=2.0
of:000000000000000b/1-of:0000000000000001/2==>of:0000000000000001/5-of:000000000000000e/1; cost=2.0
onos>
onos>
onos> ▌
```

Figure (31) Path costs between two Nodes

6.3.2 SDN over optical network environment

To perform this scenario, we prepared the supporting software for the optical network and topology of the network, as shown in Figure (32). The network topology includes six switches on a higher plane of the packet. Additionally, there are ten optic switches on the low part plane of the optical layer. Every switch of the packet is connected to the host, which represents a data center. Traffic usually began from the hosts connected to the network of the packet. There are no physical connections within the switches of the packet. All traffic from the center of the data origins are forwarded by the plane that is optical. The plane that is optical demonstrates the infrastructure of the physical links connected by ROADMs. Additionally, the network of the packet network is connected by logical connections solely when the circuit is built across a physical layer. The packet layer works in the OpenFlow 1.3 protocol.

6.3.2.1 Supporting software for optical network in ONOS

Set up, configureure and create LINC Switch: https://github.com/FlowForwarding/LINC-Switch

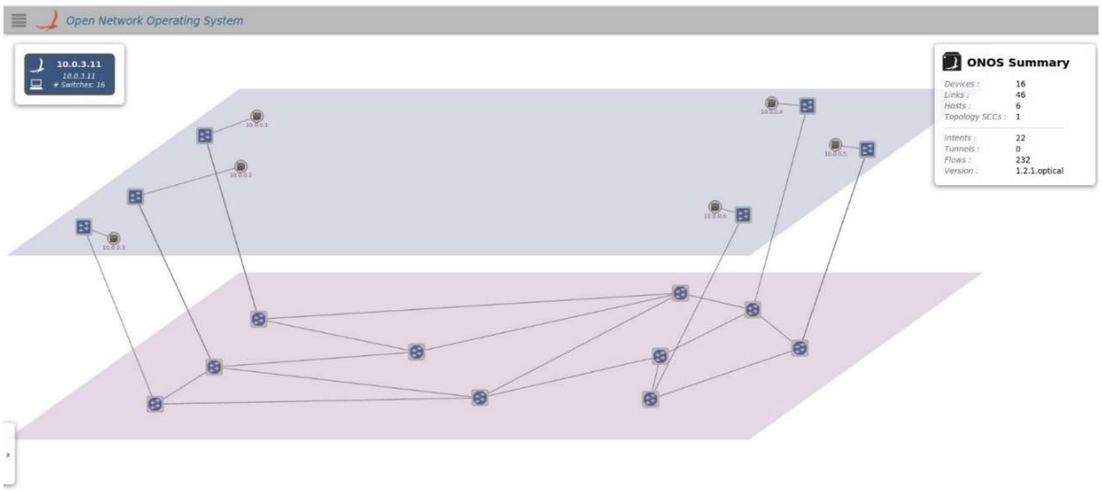
```
$ cd
$ sudo apt-get install erlang git-core bridge-utils libpcap0.8 libpcap-dev libcap2-bin
uml-utilities curl
$ git clone https://github.com/FlowForwarding/LINC-Switch.git linc-oe
$ cd linc-oe
$ sed -i s/3000/300000/ rel/files/vm.args
$ cp rel/files/sys.config.orig rel/files/sys.config
$ make
$ cd
```

Set up LINC-configure-generator: https://github.com/FlowForwarding/LINC-configure-generator

```
$ cd
$ git clone https://github.com/FlowForwarding/LINC-configure-generator.git
$ cd LINC-configure-generator
$ cppriv/* .
$ make
$ cd
```

Clone ONOS and compile the oecfg application:

```
$ cd

$ git clone https://github.com/opennetworkinglab/onos

$ cd onos/apps/oecfg

$ mci

$ cd
```



ONOS Summary

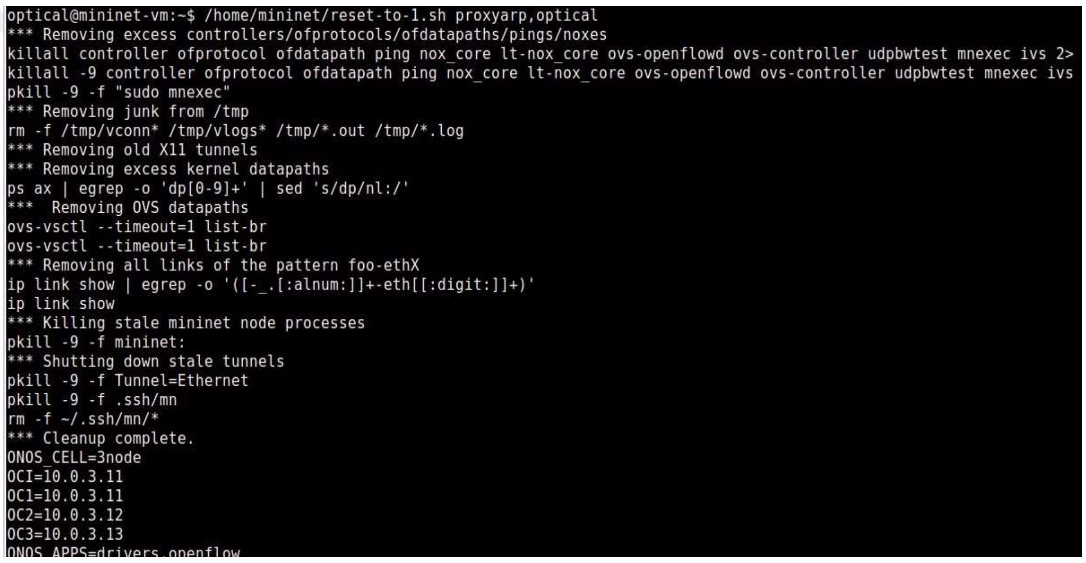
```
Devices :        16
Links :          46
Hosts :          6
Topology SCCs :  1

Intents :        22
Tunnels :        0
Flows :          232
Version :        1.2.1.optical
```

Figure (32) Network Topology

Run ONOS controller

Before running the ONOS controller, we reset the controller and configureured it by using the applications collection, which has to be automatically promoted at the beginning of the startup in order to guarantee that the environment is clean and will start ONOS. The resetting is simple: we just type the following script into the terminal (Figures. (33) and (34):

```
$ /home/mininet/reset-to-1.sh proxyarp,optical
```

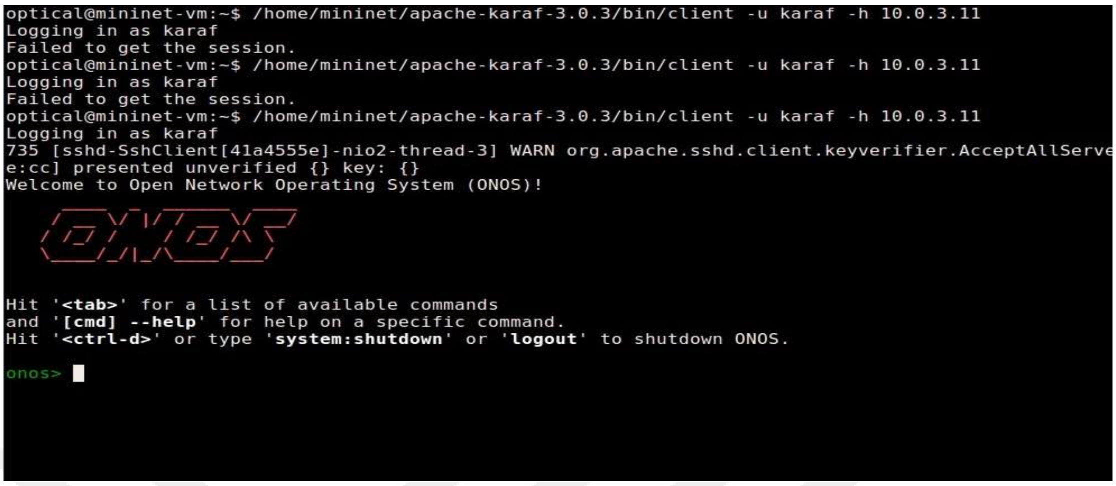


```
optical@mininet-vm:~$ /home/mininet/reset-to-1.sh proxyarp,optical
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2>
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
ONOS_CELL=3node
OCI=10.0.3.11
OC1=10.0.3.11
OC2=10.0.3.12
OC3=10.0.3.13
ONOS_APPS=drivers,openflow
```

Figure (33) ONOS Resetting

Running ONOS:

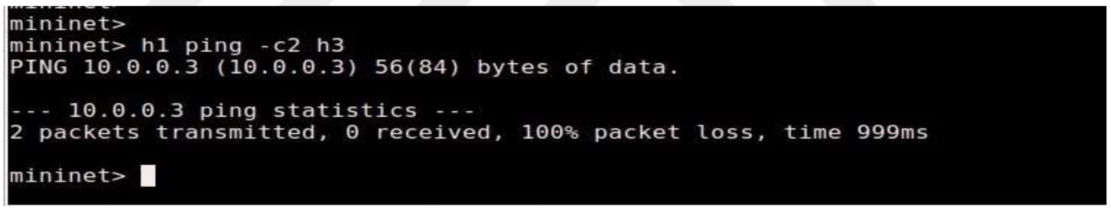
```
$ /home/mininet/apache-karaf/bin/client -u karaf -h 10.0.3.11
```



```
optical@mininet-vm:~$ /home/mininet/apache-karaf-3.0.3/bin/client -u karaf -h 10.0.3.11
Logging in as karaf
Failed to get the session.
optical@mininet-vm:~$ /home/mininet/apache-karaf-3.0.3/bin/client -u karaf -h 10.0.3.11
Logging in as karaf
Failed to get the session.
optical@mininet-vm:~$ /home/mininet/apache-karaf-3.0.3/bin/client -u karaf -h 10.0.3.11
Logging in as karaf
735 [sshd-SshClient[41a4555e]-nio2-thread-3] WARN org.apache.sshd.client.keyverifier.AcceptAllServe
e:cc] presented unverified {} key: {}
Welcome to Open Network Operating System (ONOS)!

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

onos>
```

Figure (34) Running ONOS

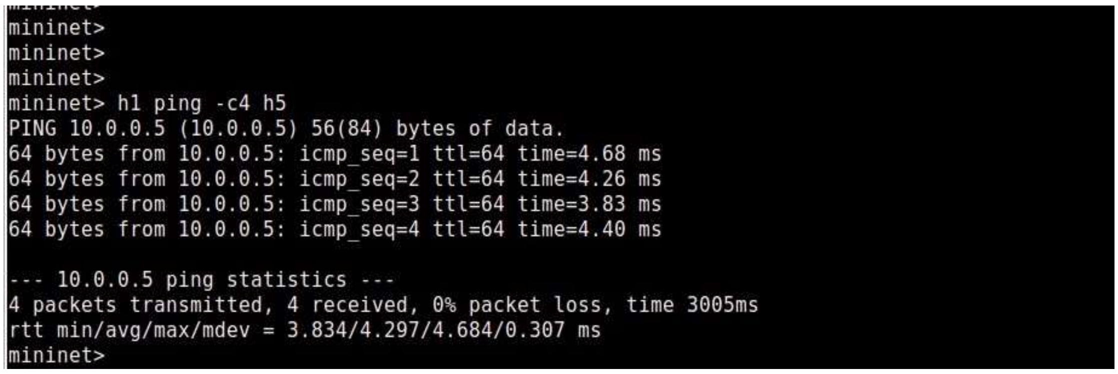
When the topology is connected to the controller but there is no communication between the hosts, we can use a ping between any two hosts in the Mininet prompt, as follows (Figure (35)):



```
mininet>
mininet> h1 ping -c2 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 999ms

mininet>
```

Figure (35) No Connection between Hosts

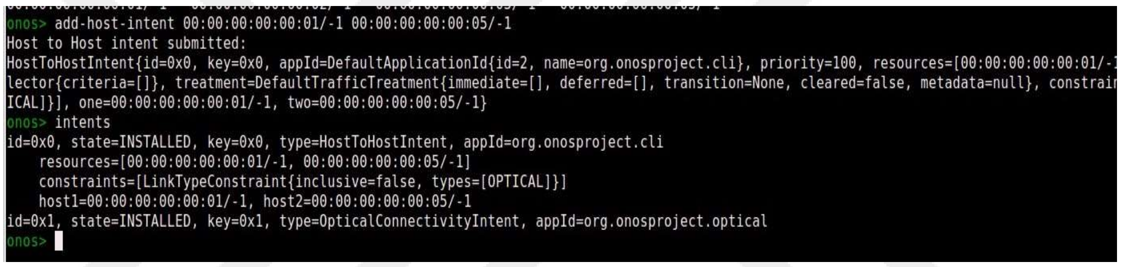
In order to create the connections between the hosts and switches, there is only one method which involves using the intent feature in two ways. First, if we want to add connections between all the hosts in one command, it can be done by using the Intent Reactive Forwarding application (ifwd) and activating it from the controller command line interface (CLI). In the optical scenario, we cannot use the Reactive Forwarding application (fwd) because it only works in IP packet environments. For its activation, we use the same steps in the ex-scenario. After activation, connections are created and we can check them by using a ping between any two hosts in the Mininet prompt, as follows (Figure (36)):



```
mininet>
mininet>
mininet>
mininet> h1 ping -c4 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=4.68 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=4.26 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=3.83 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=4.40 ms

--- 10.0.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 3.834/4.297/4.684/0.307 ms
mininet>
```

Figure (36) Connection Created

The second way to use the add host intent is by using the selected two hosts and typing the script into the controller prompt, as follows (Figure (37)):



```
onos> add-host-intent 00:00:00:00:00:01/-1 00:00:00:00:00:05/-1
Host to Host intent submitted:
HostToHostIntent{id=0x0, key=0x0, appId=DefaultApplicationId{id=2, name=org.onosproject.cli}, priority=100, resources=[00:00:00:00:00:01/-1
lector{criteria=[]}, treatment=DefaultTrafficTreatment{immediate=[], deferred=[], transition=None, cleared=false, metadata=null}, constrai
ICAL]}], one=00:00:00:00:00:01/-1, two=00:00:00:00:00:05/-1}
onos> intents
id=0x0, state=INSTALLED, key=0x0, type=HostToHostIntent, appId=org.onosproject.cli
    resources=[00:00:00:00:00:01/-1, 00:00:00:00:00:05/-1]
    constraints=[LinkTypeConstraint{inclusive=false, types=[OPTICAL]}]
    host1=00:00:00:00:00:01/-1, host2=00:00:00:00:00:05/-1
id=0x1, state=INSTALLED, key=0x1, type=OpticalConnectivityIntent, appId=org.onosproject.optical
onos> 
```

Figure (37) Host to Host Intent Creation

In the following figureures, to prove that the intent is **invariant**, we stopped one link between two switches in the optical layer. The intent between H1 and H5 automatically changed its path to another link without cutting the connection, as follows (Figures. (38) and (39)):

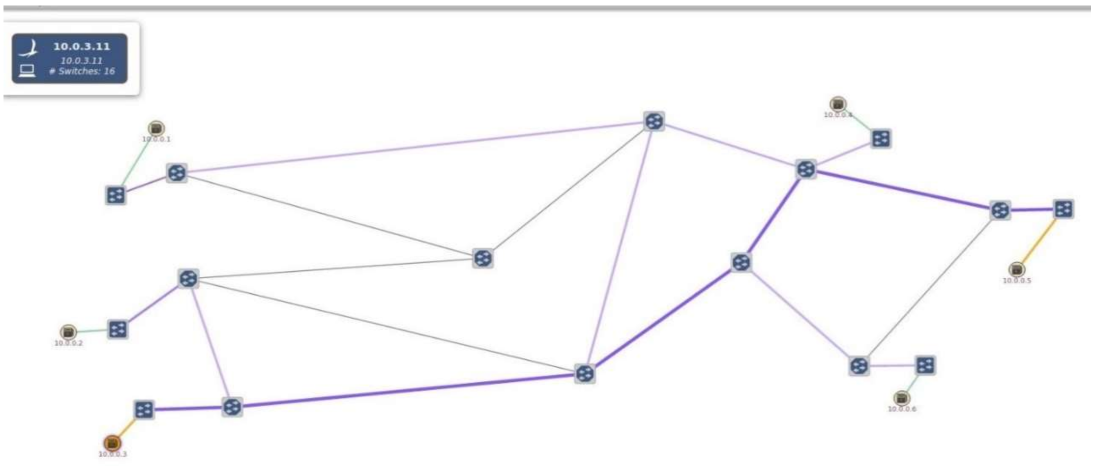


Figure (38) H1 and H5 Path Intent

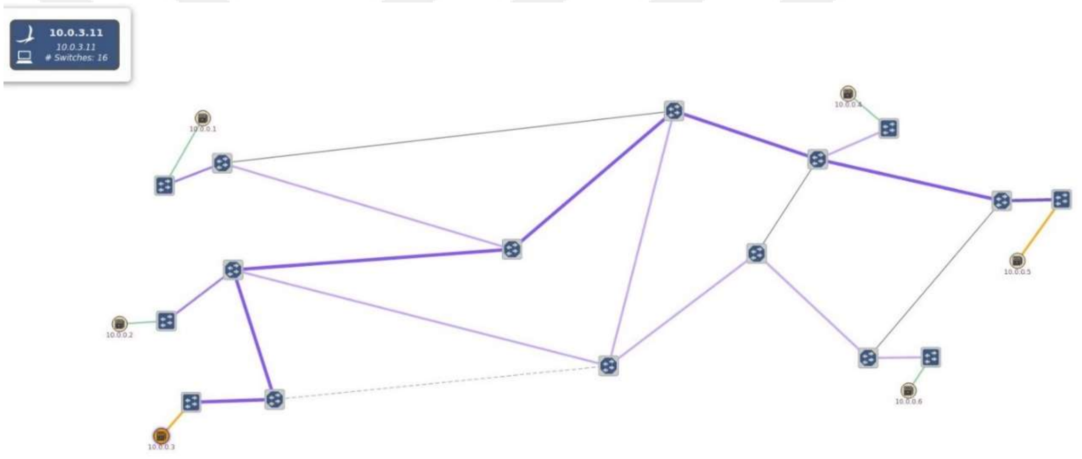
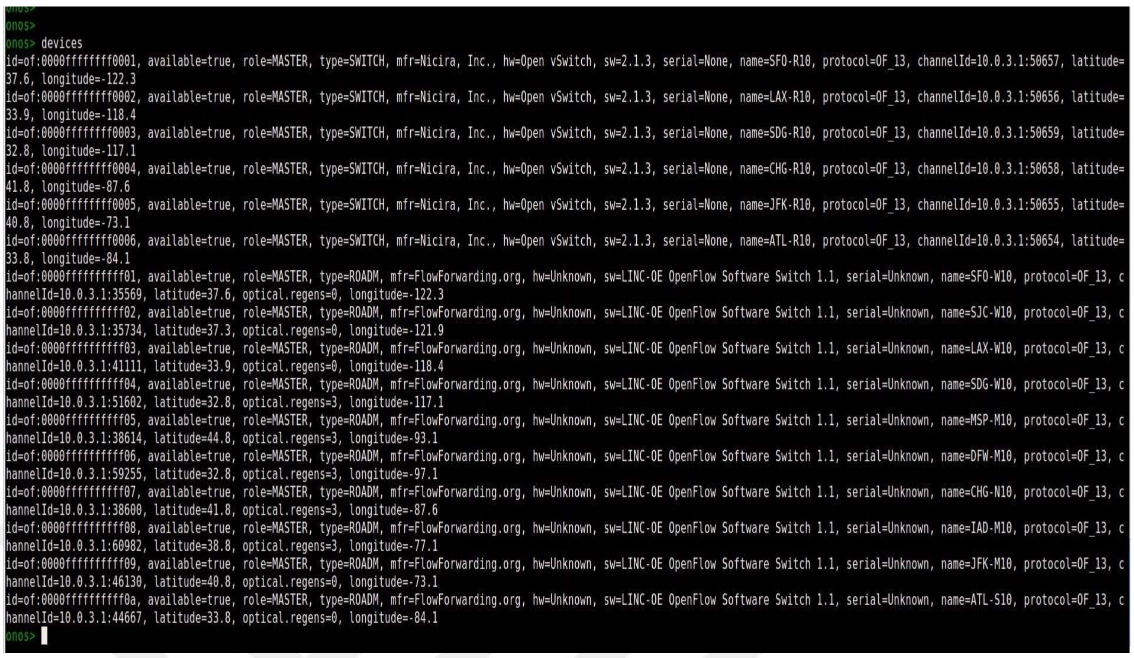


Figure (39) H1 and H5 Path Intent Changed

Through the logical centralization of the controller with the global view to the network infrastructure devices, we can check the configure devices, links, path costs between two nodes, ports, flows in each device, hosts and every physical device in the network topology by using the controller (CLI), as shown in the figureures below:

- One SDN of the controller would not be useful without any tools to manage. ONOS has suitable software commands to show the existing tools in its own system.

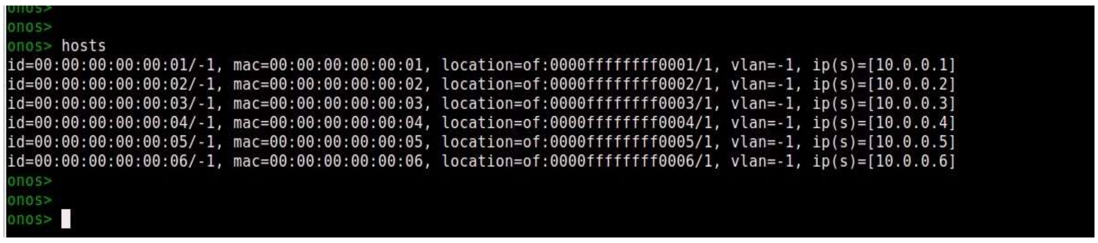


```
onos>
onos>
onos> devices
id=of:0000ffffffff0001, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, name=SFO-R10, protocol=OF_13, channelId=10.0.3.1:50657, latitude=
37.6, longitude=-122.3
id=of:0000ffffffff0002, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, name=LAX-R10, protocol=OF_13, channelId=10.0.3.1:50656, latitude=
33.9, longitude=-118.4
id=of:0000ffffffff0003, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, name=SDG-R10, protocol=OF_13, channelId=10.0.3.1:50659, latitude=
32.8, longitude=-117.1
id=of:0000ffffffff0004, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, name=CHG-R10, protocol=OF_13, channelId=10.0.3.1:50658, latitude=
41.8, longitude=-87.6
id=of:0000ffffffff0005, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, name=JFK-R10, protocol=OF_13, channelId=10.0.3.1:50655, latitude=
40.8, longitude=-73.1
id=of:0000ffffffff0006, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, name=ATL-R10, protocol=OF_13, channelId=10.0.3.1:50654, latitude=
33.8, longitude=-84.1
id=of:0000ffffffffff01, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=SFO-W10, protocol=OF_13, c
hannelId=10.0.3.1:35569, latitude=37.6, optical.regens=0, longitude=-122.3
id=of:0000ffffffffff02, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=SJC-W10, protocol=OF_13, c
hannelId=10.0.3.1:35734, latitude=37.3, optical.regens=0, longitude=-121.9
id=of:0000ffffffffff03, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=LAX-W10, protocol=OF_13, c
hannelId=10.0.3.1:41111, latitude=33.9, optical.regens=0, longitude=-118.4
id=of:0000ffffffffff04, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=SDG-W10, protocol=OF_13, c
hannelId=10.0.3.1:51602, latitude=32.8, optical.regens=3, longitude=-117.1
id=of:0000ffffffffff05, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=MSP-M10, protocol=OF_13, c
hannelId=10.0.3.1:38614, latitude=44.8, optical.regens=3, longitude=-93.1
id=of:0000ffffffffff06, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=DFW-M10, protocol=OF_13, c
hannelId=10.0.3.1:59255, latitude=32.8, optical.regens=3, longitude=-97.1
id=of:0000ffffffffff07, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=CHG-N10, protocol=OF_13, c
hannelId=10.0.3.1:38600, latitude=41.8, optical.regens=3, longitude=-87.6
id=of:0000ffffffffff08, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=IAD-M10, protocol=OF_13, c
hannelId=10.0.3.1:60982, latitude=38.8, optical.regens=3, longitude=-77.1
id=of:0000ffffffffff09, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=JFK-M10, protocol=OF_13, c
hannelId=10.0.3.1:46130, latitude=40.8, optical.regens=0, longitude=-73.1
id=of:0000ffffffffff0a, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=ATL-S10, protocol=OF_13, c
hannelId=10.0.3.1:44667, latitude=33.8, optical.regens=0, longitude=-84.1
onos>
```

Figure (40) Devices Information

The command line output contains tool IDs and Boolean values, and whether these tools are now up. It also displays the types of devices, the protocol types and their names and their latitudes and longitudes if available.

• ONOS has the capability of showing the hosts existing in the system,

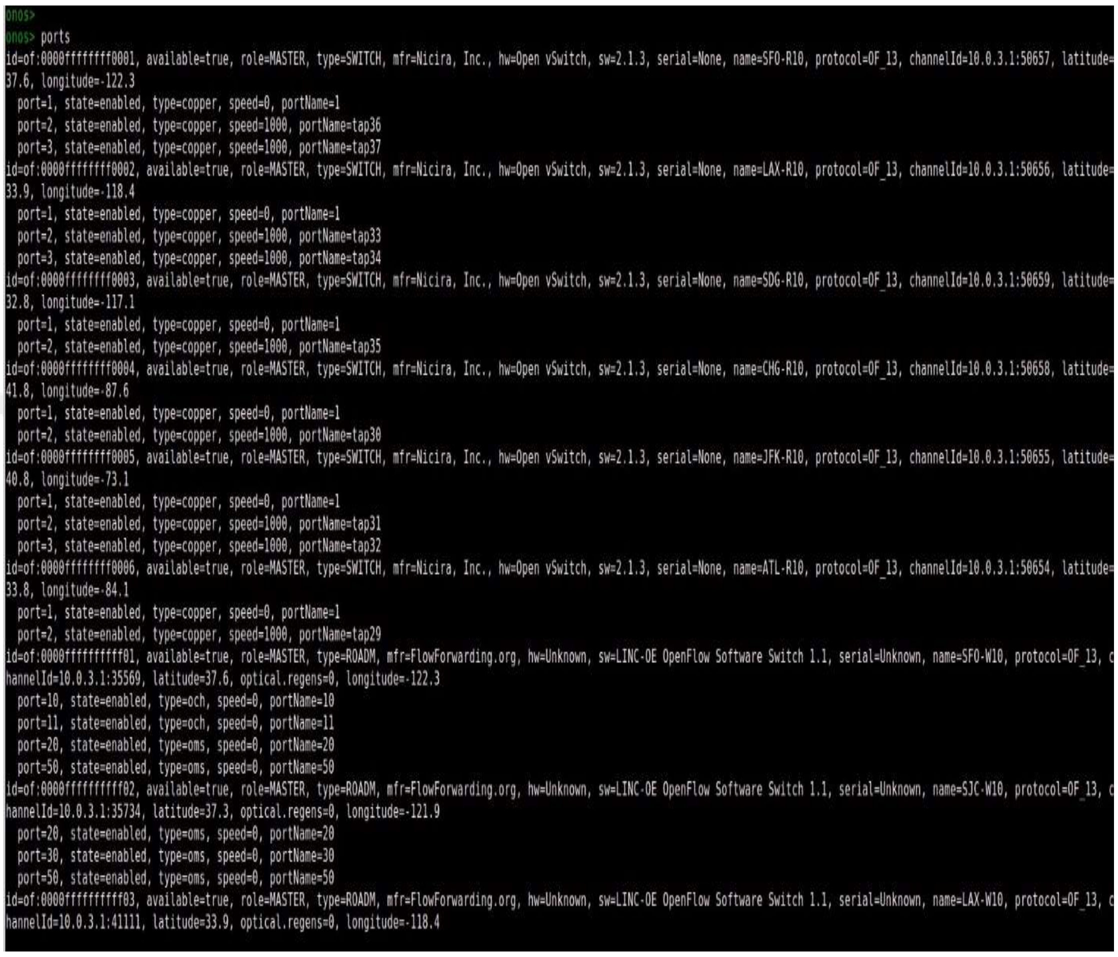


```
onos>
onos>
onos> hosts
id=00:00:00:00:00:01/-1, mac=00:00:00:00:00:01, location=of:0000ffffffff0001/1, vlan=-1, ip(s)=[10.0.0.1]
id=00:00:00:00:00:02/-1, mac=00:00:00:00:00:02, location=of:0000ffffffff0002/1, vlan=-1, ip(s)=[10.0.0.2]
id=00:00:00:00:00:03/-1, mac=00:00:00:00:00:03, location=of:0000ffffffff0003/1, vlan=-1, ip(s)=[10.0.0.3]
id=00:00:00:00:00:04/-1, mac=00:00:00:00:00:04, location=of:0000ffffffff0004/1, vlan=-1, ip(s)=[10.0.0.4]
id=00:00:00:00:00:05/-1, mac=00:00:00:00:00:05, location=of:0000ffffffff0005/1, vlan=-1, ip(s)=[10.0.0.5]
id=00:00:00:00:00:06/-1, mac=00:00:00:00:00:06, location=of:0000ffffffff0006/1, vlan=-1, ip(s)=[10.0.0.6]
onos>
onos>
onos>
```

Figure (41) Hosts Information

which illustrates the hosts' IPs as the same as their own respective MAC IDs, IPs, and the locations in the network where they are linked. Sign '-1' within that id domain is used to illustrate vlan knowledge, but here there is no vlan.

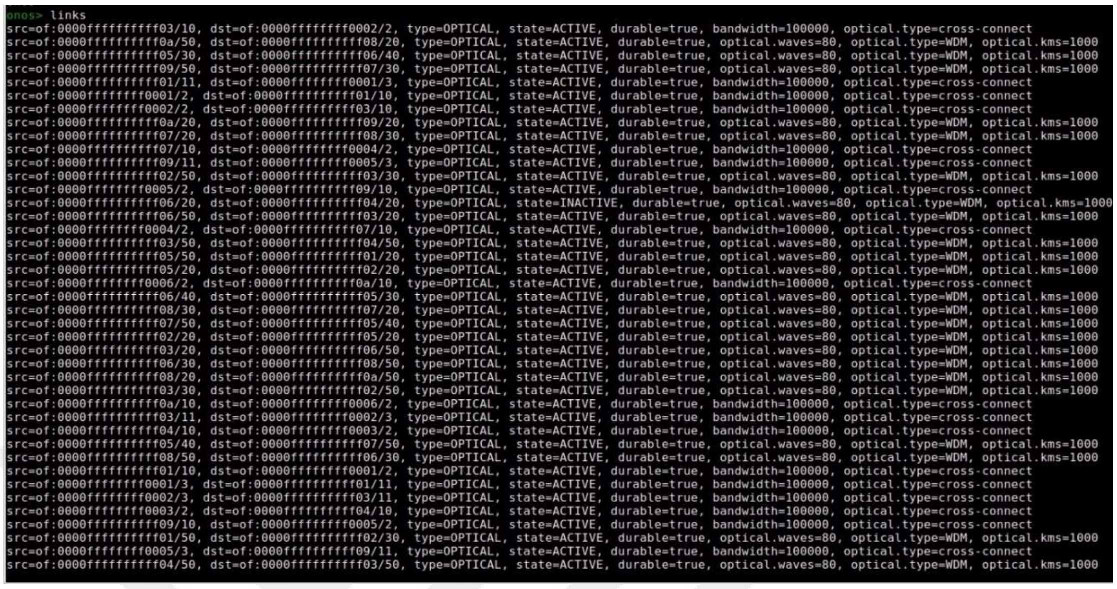
- ONOS has capability of showing the ports of each device in the system which contains the information for each device and its port connections.



```
onos>
onos> ports
id=of:0000ffffffff0001, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, name=SFO-R10, protocol=OF_13, channelId=10.0.3.1:50657, latitude=
37.6, longitude=-122.3
  port=1, state=enabled, type=copper, speed=0, portName=1
  port=2, state=enabled, type=copper, speed=1000, portName=tap36
  port=3, state=enabled, type=copper, speed=1000, portName=tap37
id=of:0000ffffffff0002, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, name=LAX-R10, protocol=OF_13, channelId=10.0.3.1:50656, latitude=
33.9, longitude=-118.4
  port=1, state=enabled, type=copper, speed=0, portName=1
  port=2, state=enabled, type=copper, speed=1000, portName=tap33
  port=3, state=enabled, type=copper, speed=1000, portName=tap34
id=of:0000ffffffff0003, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, name=SDG-R10, protocol=OF_13, channelId=10.0.3.1:50659, latitude=
32.8, longitude=-117.1
  port=1, state=enabled, type=copper, speed=0, portName=1
  port=2, state=enabled, type=copper, speed=1000, portName=tap35
id=of:0000ffffffff0004, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, name=CHG-R10, protocol=OF_13, channelId=10.0.3.1:50658, latitude=
41.8, longitude=-87.6
  port=1, state=enabled, type=copper, speed=0, portName=1
  port=2, state=enabled, type=copper, speed=1000, portName=tap30
id=of:0000ffffffff0005, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, name=JFK-R10, protocol=OF_13, channelId=10.0.3.1:50655, latitude=
40.8, longitude=-73.1
  port=1, state=enabled, type=copper, speed=0, portName=1
  port=2, state=enabled, type=copper, speed=1000, portName=tap31
  port=3, state=enabled, type=copper, speed=1000, portName=tap32
id=of:0000ffffffff0006, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.1.3, serial=None, name=ATL-R10, protocol=OF_13, channelId=10.0.3.1:50654, latitude=
33.8, longitude=-84.1
  port=1, state=enabled, type=copper, speed=0, portName=1
  port=2, state=enabled, type=copper, speed=1000, portName=tap29
id=of:0000ffffffffff01, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=SFO-W10, protocol=OF_13, c
hannelId=10.0.3.1:35569, latitude=37.6, optical.regens=0, longitude=-122.3
  port=10, state=enabled, type=och, speed=0, portName=10
  port=11, state=enabled, type=och, speed=0, portName=11
  port=20, state=enabled, type=oms, speed=0, portName=20
  port=50, state=enabled, type=oms, speed=0, portName=50
id=of:0000ffffffffff02, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=SJC-W10, protocol=OF_13, c
hannelId=10.0.3.1:35734, latitude=37.3, optical.regens=0, longitude=-121.9
  port=20, state=enabled, type=oms, speed=0, portName=20
  port=30, state=enabled, type=oms, speed=0, portName=30
  port=50, state=enabled, type=oms, speed=0, portName=50
id=of:0000ffffffffff03, available=true, role=MASTER, type=ROADM, mfr=FlowForwarding.org, hw=Unknown, sw=LINC-OE OpenFlow Software Switch 1.1, serial=Unknown, name=LAX-W10, protocol=OF_13, c
hannelId=10.0.3.1:41111, latitude=33.9, optical.regens=0, longitude=-118.4
```

Figure (42) Ports Information

• ONOS has capability of showing the links existing in the system:

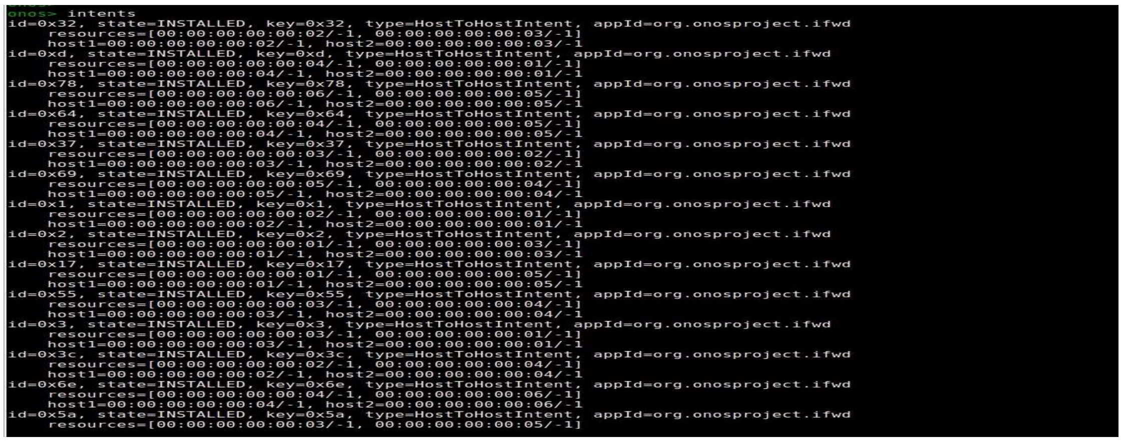


```
onos> links
src=of:0000ffffffffff03/10, dst=of:0000ffffffff0002/2, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff0a/50, dst=of:0000ffffffffff08/20, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff05/30, dst=of:0000ffffffffff06/40, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff09/50, dst=of:0000ffffffffff07/30, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff01/11, dst=of:0000ffffffff0001/3, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffff0001/2, dst=of:0000ffffffffff01/10, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffff0002/2, dst=of:0000ffffffffff03/10, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff0a/20, dst=of:0000ffffffffff09/20, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff07/20, dst=of:0000ffffffffff08/30, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff07/10, dst=of:0000ffffffff0004/2, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff09/11, dst=of:0000ffffffff0005/3, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff02/50, dst=of:0000ffffffffff03/30, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffff0005/2, dst=of:0000ffffffffff09/10, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff06/20, dst=of:0000ffffffffff04/20, type=OPTICAL, state=INACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff06/50, dst=of:0000ffffffffff03/20, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffff0004/2, dst=of:0000ffffffffff07/10, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff03/50, dst=of:0000ffffffffff04/50, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff05/50, dst=of:0000ffffffffff01/20, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff05/20, dst=of:0000ffffffffff02/20, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffff0006/2, dst=of:0000ffffffffff0a/10, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff06/40, dst=of:0000ffffffffff05/30, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff08/30, dst=of:0000ffffffffff07/20, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff07/50, dst=of:0000ffffffffff05/40, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff02/20, dst=of:0000ffffffffff05/20, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff03/20, dst=of:0000ffffffffff06/50, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff06/30, dst=of:0000ffffffffff08/50, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff08/20, dst=of:0000ffffffffff0a/50, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff03/30, dst=of:0000ffffffffff02/50, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff0a/10, dst=of:0000ffffffff0006/2, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff03/11, dst=of:0000ffffffff0002/3, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff04/10, dst=of:0000ffffffff0003/2, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff05/40, dst=of:0000ffffffffff07/50, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff08/50, dst=of:0000ffffffffff06/30, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffffff01/10, dst=of:0000ffffffff0001/2, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffff0001/3, dst=of:0000ffffffffff01/11, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffff0002/3, dst=of:0000ffffffffff03/11, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffff0003/2, dst=of:0000ffffffffff04/10, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff09/10, dst=of:0000ffffffff0005/2, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff01/50, dst=of:0000ffffffffff02/30, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
src=of:0000ffffffff0005/3, dst=of:0000ffffffffff09/11, type=OPTICAL, state=ACTIVE, durable=true, bandwidth=100000, optical.type=cross-connect
src=of:0000ffffffffff04/50, dst=of:0000ffffffffff03/50, type=OPTICAL, state=ACTIVE, durable=true, optical.waves=80, optical.type=WDM, optical.kms=1000
```

Figure (43) Links Information

The output presents a group of met links, one of which is related to the link formed by the resource tool port couple to the end tool port couple. Subject 'type' area shows whether or not the links are direct coupling points between those two tools. It also shows the link bandwidth and optical distance.

• ONOS has the capability of showing the intents existing in the system:

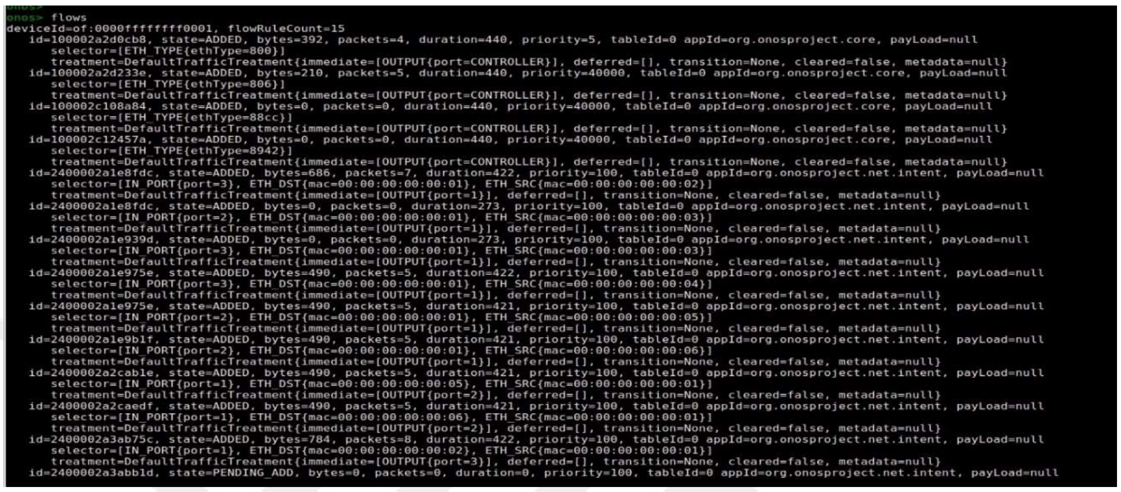


```
onos> intents
id=0x32, state=INSTALLED, key=0x32, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:02/-1, 00:00:00:00:00:03/-1]
    host1=00:00:00:00:00:02/-1, host2=00:00:00:00:00:03/-1
id=0xd, state=INSTALLED, key=0xd, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:04/-1, 00:00:00:00:00:01/-1]
    host1=00:00:00:00:00:04/-1, host2=00:00:00:00:00:01/-1
id=0x78, state=INSTALLED, key=0x78, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:06/-1, 00:00:00:00:00:05/-1]
    host1=00:00:00:00:00:06/-1, host2=00:00:00:00:00:05/-1
id=0x64, state=INSTALLED, key=0x64, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:04/-1, 00:00:00:00:00:05/-1]
    host1=00:00:00:00:00:04/-1, host2=00:00:00:00:00:05/-1
id=0x37, state=INSTALLED, key=0x37, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:03/-1, 00:00:00:00:00:02/-1]
    host1=00:00:00:00:00:03/-1, host2=00:00:00:00:00:02/-1
id=0x69, state=INSTALLED, key=0x69, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:05/-1, 00:00:00:00:00:04/-1]
    host1=00:00:00:00:00:05/-1, host2=00:00:00:00:00:04/-1
id=0x1, state=INSTALLED, key=0x1, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:02/-1, 00:00:00:00:00:01/-1]
    host1=00:00:00:00:00:02/-1, host2=00:00:00:00:00:01/-1
id=0x2, state=INSTALLED, key=0x2, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:01/-1, 00:00:00:00:00:03/-1]
    host1=00:00:00:00:00:01/-1, host2=00:00:00:00:00:03/-1
id=0x17, state=INSTALLED, key=0x17, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:01/-1, 00:00:00:00:00:05/-1]
    host1=00:00:00:00:00:01/-1, host2=00:00:00:00:00:05/-1
id=0x55, state=INSTALLED, key=0x55, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:03/-1, 00:00:00:00:00:04/-1]
    host1=00:00:00:00:00:03/-1, host2=00:00:00:00:00:04/-1
id=0x3, state=INSTALLED, key=0x3, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:03/-1, 00:00:00:00:00:01/-1]
    host1=00:00:00:00:00:03/-1, host2=00:00:00:00:00:01/-1
id=0x3c, state=INSTALLED, key=0x3c, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:02/-1, 00:00:00:00:00:04/-1]
    host1=00:00:00:00:00:02/-1, host2=00:00:00:00:00:04/-1
id=0x6e, state=INSTALLED, key=0x6e, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:04/-1, 00:00:00:00:00:06/-1]
    host1=00:00:00:00:00:04/-1, host2=00:00:00:00:00:06/-1
id=0x5a, state=INSTALLED, key=0x5a, type=HostToHostIntent, appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:03/-1, 00:00:00:00:00:05/-1]
```

Figure (44) Intents Information

The output presents a list of found intents which contain the intent ID, the intent state, the type of intent, which application created it and the source-destination devices.

• ONOS has the capability of showing the flows existing in the system:



```
onos> flows
deviceId=of:0000ffffffff0001, flowRuleCount=15
   id=100002a2d0cb8, state=ADDED, bytes=392, packets=4, duration=440, priority=5, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=800}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=100002a2d233e, state=ADDED, bytes=210, packets=5, duration=440, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=806}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=100002c108a84, state=ADDED, bytes=0, packets=0, duration=440, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=88cc}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=100002c12457a, state=ADDED, bytes=0, packets=0, duration=440, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=8942}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=2400002a1e8fdc, state=ADDED, bytes=686, packets=7, duration=422, priority=100, tableId=0 appId=org.onosproject.net.intent, payLoad=null
      selector=[IN_PORT{port=3}, ETH_DST{mac=00:00:00:00:00:01}, ETH_SRC{mac=00:00:00:00:00:02}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=1}], deferred=[], transition=None, cleared=false, metadata=null}
   id=2400002a1e8fdc, state=ADDED, bytes=0, packets=0, duration=273, priority=100, tableId=0 appId=org.onosproject.net.intent, payLoad=null
      selector=[IN_PORT{port=2}, ETH_DST{mac=00:00:00:00:00:01}, ETH_SRC{mac=00:00:00:00:00:03}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=1}], deferred=[], transition=None, cleared=false, metadata=null}
   id=2400002a1e939d, state=ADDED, bytes=0, packets=0, duration=273, priority=100, tableId=0 appId=org.onosproject.net.intent, payLoad=null
      selector=[IN_PORT{port=3}, ETH_DST{mac=00:00:00:00:00:01}, ETH_SRC{mac=00:00:00:00:00:03}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=1}], deferred=[], transition=None, cleared=false, metadata=null}
   id=2400002a1e975e, state=ADDED, bytes=490, packets=5, duration=422, priority=100, tableId=0 appId=org.onosproject.net.intent, payLoad=null
      selector=[IN_PORT{port=3}, ETH_DST{mac=00:00:00:00:00:01}, ETH_SRC{mac=00:00:00:00:00:04}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=1}], deferred=[], transition=None, cleared=false, metadata=null}
   id=2400002a1e975e, state=ADDED, bytes=490, packets=5, duration=421, priority=100, tableId=0 appId=org.onosproject.net.intent, payLoad=null
      selector=[IN_PORT{port=2}, ETH_DST{mac=00:00:00:00:00:01}, ETH_SRC{mac=00:00:00:00:00:05}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=1}], deferred=[], transition=None, cleared=false, metadata=null}
   id=2400002a1e9b1f, state=ADDED, bytes=490, packets=5, duration=421, priority=100, tableId=0 appId=org.onosproject.net.intent, payLoad=null
      selector=[IN_PORT{port=2}, ETH_DST{mac=00:00:00:00:00:01}, ETH_SRC{mac=00:00:00:00:00:06}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=1}], deferred=[], transition=None, cleared=false, metadata=null}
   id=2400002a2ca1e, state=ADDED, bytes=490, packets=5, duration=421, priority=100, tableId=0 appId=org.onosproject.net.intent, payLoad=null
      selector=[IN_PORT{port=1}, ETH_DST{mac=00:00:00:00:00:05}, ETH_SRC{mac=00:00:00:00:00:01}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=2}], deferred=[], transition=None, cleared=false, metadata=null}
   id=2400002a2caedf, state=ADDED, bytes=490, packets=5, duration=421, priority=100, tableId=0 appId=org.onosproject.net.intent, payLoad=null
      selector=[IN_PORT{port=1}, ETH_DST{mac=00:00:00:00:00:06}, ETH_SRC{mac=00:00:00:00:00:01}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=2}], deferred=[], transition=None, cleared=false, metadata=null}
   id=2400002a3ab75c, state=ADDED, bytes=784, packets=8, duration=422, priority=100, tableId=0 appId=org.onosproject.net.intent, payLoad=null
      selector=[IN_PORT{port=1}, ETH_DST{mac=00:00:00:00:00:02}, ETH_SRC{mac=00:00:00:00:00:01}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=3}], deferred=[], transition=None, cleared=false, metadata=null}
   id=2400002a3abb1d, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=100, tableId=0 appId=org.onosproject.net.intent, payLoad=null
```

Figure (45) Flow Information

ONOS gives several detailed items of information of flows on switches. For example, any flow input determines the separator and the method that is the collection of the traffic paired by a flow input and how the traffic has to be managed. For all flow inputs that are pursued by the appId (application id), the appId knows which of the application has settled the flow input, which is a beneficial characteristic in as much as it can support the admin to recognize which of the applications might be misbehaving or wasting many resources.

6.4 Summary

From the above test scenarios, SDN showed its strength and flexibility for network management by instantaneously establishing connections between nodes and handling link failures or crashed switches. The global view of the central controller allows access to all infrastructure devices in the data layer. In addition, the programmability provided by SDN gives a dynamic style to the network in order to solve any emergency event immediately.

To create the connections between nodes, SDN provides many ways to perform this in single steps by using the available applications such as (Reactive Forwarding application (fwd)) or (Intent Reactive Forwarding application (ifwd)) in the controller, as in Figures. (18) and (21), which can establish the connections between every node or between two specific nodes. In comparison, the method for traditional networks establishes connections in a distributed style where each router uses the packet destination IP address to define the next hop in parallel with information from its own forwarding table. The information of the network topology is maintained in the router's forwarding tables and is collected via an IP routing protocol, such as RIP, BGP, IS-IS, OSPF or static configureuration, which holds that information synchronized with the network changing.
SDN can handle link fail downs, crashed servers or switches in an instantaneous manner through the intent Reactive Forwarding application (ifwd) (Figures. (25) and (39)) without losing data or time or disconnecting between the nodes. The intent shows that applications are independent of any underlying network details. In traditional networks, if there is a link fail down or crashed server or switch, it will need manual configureuration according to the device vendor's specifications to handle the problem, which costs time, money and effort and it loses data.
A centralized SDN controller has a global view of every infrastructure device. Through controller CLI comments, SDN enables the network operator to access, check and manage everything connected to the network, including devices, links, flows, ports, hosts and the path cost between nodes (Figures. (26), (27), (28), (29), (30) and (31)). This property does not exist in traditional networks.
The programmability given to SDN allows the network operator, through controller CLI comments, to deal with the network in a dynamic style to solve emergency events such as congestion, addition of new routes, new intents, etc. Traditional networks do not support a programmable style because they are static.
Through the Graphic User Interface (GUI) (a single page web application, providing a visual front to the SDN controller), the monitoring of the network has become simple as it presents all the network information and everything connected to the network in

graphical form, as in Figures. (13) and (32). The SDN GUI provides the same ability as the controller CLI. This property does not exist in traditional networks.

Finally, SDN gives service providers greater ability through API to control networks as far as application availability.

CHAPTER 7
CHALLENGES AND CONCLUSIONS

7.1 Challenges in SDN and Open Issues

SDN is regarded as an emerging technology for network performance, security and energy efficiency. However, there are many challenges and open problems prior to realizing its full potential. Here, we discuss the important problems and challenges in SDN [54].

7.1.1 Scalability of the SDN Controller

In SDN, network functions are centralized to the SDN controller. The SDN controller is responsible for handling all important operations in the network. Thus, finding the optimal number of controllers and their optimal position/placement in the network for better performance, scalability, security and energy efficiency is still an active research topic.

7.1.2 Flow-table Management in SDN

SDN is a flat network where SDN switches lack efficiency to work independently, which makes them dependent on the rules set by the SDN controller. An SDN switch matches the packet with the flow entries in the table to make a decision. The flow tables in SDN have limited space. The storage of greater numbers of flow entries may impose overheads on the flow tables thereby increasing costs and degrading overall performance. This necessitates intelligent flow table management methods that can store a greater number of rules without increasing the cost and degrading the overall network performance, network security and energy consumption in SDN.

7.1.3 Performance of SDN Switches

SDN switches are not intelligent and they operate based on the rules set by the SDN controller. The performance of switches in SDN directly impacts the overall performance of the network, including network security [7]. The issues of concern in the switches include maximum output that can be obtained using available OpenFlow switches being limited to 38-1000 flow-mod per second. Switches need high capacity processors to work efficiently. If a greater number of CPUs are included in the switch, power consumption will increase.

7.1.4 Real-time Change Update in SDN

SDN is reconfigureurable on the fly based on its operating conditions. In any SDN, it is a real challenge to address dynamic real-time change and the deployment of rules. As previously discussed, SDN has the ability to automate the provisioning of new converged infrastructures in minutes and impact multiple devices in real time. This results in gaps in the visibility of SDN changes and there will be a huge impact when small things go wrong. In SDN, another challenge is the accommodation of rapid on-demand growth, which poses a risk to SDN monitoring platforms. Defense and monitoring solutions must be able to accommodate the rapid growth of SDN infrastructures. Otherwise, they can quickly become over-subscribed, resulting in failure of whole systems.

7.1.5 Integration of SDN and Traditional Networks

Traditional networks are hierarchical and SDN is flat. Thus, the integration of SDN and legacy networks is another challenge where defense solutions and monitoring systems should be compatible and robust for both networks to enhance overall performance, network security and energy savings. It is important that one system should not be a bottleneck for another. SDN is expected to be able to work based on the context of a particular customer or tenant of the network to serve them better and meet their QoS requirements. This may degrade the network performance when there is a hybrid network with physical and virtual services.

7.2 Conclusions

The complexity of conventional networks has made its management too difficult. The important reasons are the controlling and the forwarding functionality in the same device and the vendor dependency. In addition, a concurring reason is that the line products and versions are tightly tied to typical networking devices, which means that each vendor may have its own management interfaces and configureurations, implying a long time for the release of new product updates (e.g., new firmware) or upgrades (e.g., new versions of devices). All this has given rise to problems for vendors and network infrastructure owners, as well as affecting heavy limitations to change and innovation.

SDN has created opportunities to solve these longstanding problems. Some of the main ideas of SDN are the provision of dynamical programming in the forwarding devices through open southbound interfaces, the separation of data and control plane and the overall view of the network by the logic central of the network controller. However, while the data plane devices become simple but extremely efficient and programmable packet forwarding elements, the control plane has become a new single entity represented by the controller or Network Operating System (NOS). The network logic is implemented by applications which run on top of the controller, which has made deployment and development much easier than traditional networks. Given the overall view, the consistency of policies is easy to enforce. The prime paradigm shift represented by SDN is the evolution and development of networks, providing a rapidity of innovation in networking infrastructure.

Despite recent and interesting efforts to survey this new chapter in the history of networks, the literature is as yet lacking, and to the best of our knowledge, there is only a single comprehensive and extensive overview of the building blocks, connotations, and challenges of SDNs. Attempting to handle this gap, this thesis has endeavored to give coverage to a vast range of existing solutions provided by SDN as well as future directions for researchers.

REFERENCES

1. **R. Alvizu et al., (2017)** "*Comprehensive survey on T-SDN: Software-defined Networking for Transport Networks*" IEEE Commun. Surv. pp. 1-52.

2. **D. Kreutz et al., (2015)** "*Software-defined networking: A comprehensive survey*" Proceedings of the IEEE, vol. 103, pp. 14–76, no. 1.

3. **H. Kim and N. Feamster, (2013)** ''*Improving network management with software defined networking,*'' IEEE Commun. Mag., vol. 51, pp. 114–119, no. 2.

4. **A. Lara, A. Kolasani, and B. Ramamurthy, (2014)** ''*Network innovation using OpenFlow: A survey,*'' IEEE Commun. Surv. Tut., vol. 16, pp. 493–512, no. 1.

5. **B. Nunes, et al., (2014)** ''*A survey of software-defined networking: Past, present, future of programmable networks,*'' IEEE Commun. Surv. Tut., vol. 16, pp. 1617–1634, no. 3.

6. **Y. Jarraya, T. Madi, and M. Debbabi, (2014)** ''*A survey and a layered taxonomy of software-defined networking,*'' IEEE Commun. Surv. Tut., vol. 16, pp. 1955–1980, no. 4.

7. **N. Feamster and H. Balakrishnan, (2005)** ''*Detecting BGP configureuration faults with static analysis,*'' in Proc. 2nd Conf. Symp. Netw. Syst. Design Implement., vol. 2, pp. 43–56.

8. **M. Casado, N. Foster, and A. Guha, (2014)** ''*Abstractions for software-defined networks,*'' ACM Commun., vol. 57, no. 10, pp. 86–95.

9. **Y. GONG, (2015)** "*A survey on software defined networking and its applications*" Springer-Verlag Berlin Heidelberg, Inc. pp. 827-845.

10. **ONF Market Education Committee, (2012)** "*Software-defined networking: the new norm for networks*" ONF White Paper, 1-12.

11. **Zuo Q, Chen M. (2013)** "*Openflow-based SDN technologies*" Journal of Software, 13, 24(5): 1078–1097

12. **Casado M. et al., (2007)** "*taking control of the enterprise*" ACM SIGCOMM Computer Communication Review, 37(4): 1–12.

13. **McKeown N., et al., (2008)** "*OpenFlow: enabling innovation in campus networks*" ACM SIGCOMM Computer Communication Review, 38(2): 69–74.

14. **Al-Fares M. et al., (2010)** "*Dynamic flow scheduling for data center networks*" 7th USENIX Conference on Networked Systems Design and Implementation. 1–19.

15. **Handigol N. et al., (2009)** "*Plug-n-Serve: load-balancing web traffic using OpenFlow*" ACM SIGCOMM Demo, 4(5): 6.

16. **Wang R., et al., (2011)** "*Openflow-based server load balancing gone wild*" 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services. 2011, 1–12.

17. **Gurbani V. et al., (2012)** "*Abstracting network state in software defined networks (SDN) for rendezvous services*" IEEE International Conference on Communications. pp. 6627–6632.

18. **Kotronis V, et al., (2012)** "*Outsourcing the routing control logic: better internet routing based on SDN principles*" 11th ACM Workshop on Hot Topics in Networks. 2012, 55–60.

19. **Kloti R, et al., (2013)** "*OpenFlow: a security analysis*" 21st IEEE International Conference on Network Protocols.

20. **Nayak A K, et al., (2009)** "*Resonance: dynamic access control for enterprise networks*" 1st ACM Workshop on Research on Enterprise Networking, 11–18.

21. **Shin S, et al., (2012)** "*FRESCO: modular composable security services for software-defined networks*" Internet Society NDSS. 2013 51. Jafarian J H, Al-Shaer E, Duan Q. Openflow random host mutation: transparent moving target defense using software defined networking. In: Proceedings of the 1st ACM Workshop on Hot Topics in Software Defined Networks. 127–132.

22. **Wang Y, et al., (2013)** "*NetFuse: shortcircuiting traffic surges in the cloud*" IEEE International Conference on Communications. pp. 3514–3518.

23. **Sydney A, et al., (2014)** "*Using geni for experimental evaluation of software defined networking in smart grids*" Computer Networks, 63: 5–16.

24. **Simeonidou D, et al., (2011)** "*Enabling the future optical internet with OpenFlow: a paradigm shift in providing intelligent optical network services*" 13th International Conference on Transparent Optical Networks. 1–4.

25. **Das S, et al., (2010)** "*Packet and circuit network convergence with OpenFlow*" Optical Fiber Communication Conference.

26. **Azodolmolky S, et al., (2011)** "*Integrated OpenFlow-GMPLS control plane: an overlay model for software defined packet over optical networks*" Optics Express, 19(26): 421–428.

27. **Gudla V, et al., (2010)** "*Experimental demonstration of OpenFlow control of packet and circuit switches*" Optical Fiber Communication Conference.

28. **Bansal M, et al., (2012)** "*Openradio: a programmable wireless dataplane*" 1st ACM Workshop on Hot Topics in Software Defined Networks. 109–114.

29. **Sharafat A, et al., (2011)** "*Mpls-te and mpls vpns with openflow*". ACM SIGCOMM Computer Communication Review, 41(4): 452–453.

30. **Sudhir K., et al., (2016)** "*Software Defined Networking for Optical Networks*" IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics, pp. 133-137.

31. **Hu F., et al., (2014)** "*A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation*" IEEE COMMUNICATION SURVEYS & TUTORIALS, VOL. 16, NO. 4, pp. 2181-2206.

32. **S. Ortiz, (2013)** "*Software-defined networking: On the verge of a breakthrough?*" Computer, vol. 46, no. 7, pp. 10–12.

33. **K. Bakshi, (2013)** "*Considerations for software defined networking (SDN): Approaches and use cases,*" in Proc. IEEE Aerosp. Conf., pp. 1–9.

34. **S. Fang, et al., (2013)** "*A loss-free multipathing solution for data center network using software-defined networking approach,*" IEEE Trans. Magn., vol. 49, no. 6, pp. 2723–2730.

35. **C. S. Li and W. Liao, (2013)** "*Software defined networks,*" IEEE Comm. Mag., vol. 51, no. 2, p. 113.

36. **M. Casado, et al., (2012)** "*Fabric: A retrospective on evolving SDN*," in Proc. Workshop Hot Topics Softw. Defined Netw., pp. 85–90.

37. **Y. Kanaumi, S. Saito, and E. Kawai, (2010)** "*Toward large-scale programmable networks: Lessons learned through the operation and management of a wide-area OpenFlow-based network*," in Proc. Int. Conf. Netw. Serv.Manage., pp. 330–333.

38. **H. Fei, (2014)** "*Network Innovation Through OpenFlow and SDN: Principles and Design*" New York, NY, USA: Taylor & Francis.

39. **B. Lantz, et al., (2010)** "*A network in a laptop: Rapid prototyping for software-defined networks,*" in Proc. ACM SIGCOMM Workshop Hot Topics Netw., New York,

NY, USA, pp. 19:1–19:6.

40. **T. D. Nadeau and P. Pan, (2011)** "*Software driven networks problem statement,*" IETF Internet-Draft (Work-in-Progress), raft-nadeau-sdnproblem-statement-01.

41. **M. Yu, et al., (2010)** "*Scalable flow-based networking with DIFANE*," Proc. ACM SIGCOMM Comput. Commun. Review, vol. 40, no. 4, pp. 351–362.

42. **K. Yap et al., (2010)** "*OpenRoads: Empowering research in mobile networks,*" ACMSIGCOMM Comput. Commun. Review, vol. 40, no. 1, pp. 125–126.

43. **Y. Hu, et al. (2013)** "*Reliability-aware controller placement for software-defined networks,*" in Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage, pp. 672–675.

44. **F. de O. Silva, et al., (2012)** "*Enabling future Internet architecture research and experimentation by using software defined networking*," in Proc. Eur. Workshop Softw. Defined Netw, pp. 73–78.

45. **A. Gember, et al., (2012)** "*ECOS: Leveraging softwaredefined networks to support mobile application offloading,*" in Proc. Eighth ACM/IEEE Symp. Architectures Netw. Commun. Syst., pp. 199–210.

46. **V. Mann, rt al., (2012)** "*CrossRoads: Seamless VM mobility across data centers through software defined networking,*" in Proc. IEEE Netw. Operations Manage. Symp., pp. 88–96.

47. **Z. Liu, et al., (2013)** "*M2cloud: Software defined multi-site data center network control framework for multi-tenant,*" SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pp. 517–518.

48. **S. Gringerl, et al., (2013)** "*Extending Software Defined Network Principles to Include Optical Transport,*" IEEE Communications Magazine, vol., no. 3, pp. 324 – 334.

49. **Singh S., Kumar R., (2016)** "*A Survey on Software Defined Networking: Architecture for Next Generation Network*" Springer Science Business Media NY, pp. 321-374.

50. **Dizdarevic S., et al., (2016)** "*A Survey on transition from GMPLS control plane for optical multilayer networks to SDN control plane*" IEEE 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 537-544.

51. **T. Anderson, et al., (2005)** "*Overcoming the internet impasse through virtualization,*" Computer 38 (4).

52. **HP, (2012)** "*Realizing the Power of SDN with HP Virtual Application Networks*".

53. **Bhaumik P., et al., (2014)** "*Software-defined optical networks (SDONs): a survey*" Springer Science Business Media New York, pp. 4-18.

54. **Danda B., et al., (2017)** "*Software Defined Networking Architecture, Security and Energy Efficiency: A Survey*" IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 19, NO. 1, pp. 325-346.

55. **Ishio H., et al., (1984)** "*WaveIength-Division-Multiplexing Technology and Its Application*" IEEE JOURNAL OF LIGHTWAVE TECHNOLOGY, VOL. LT-2, NO. 4, pp, 448-463.

56. **R. Roka, (2003)** "*The utilization of the DWDWCWDM combination in the metro/access networks*" IEEE Mobile Future and Symposium on Trends in Communications, pp, 160-162.

57. **Azeddien M., (2014)** "*Modeling and Simulating MPLS Networks*" IEEE International Symposium on Networks, Computers and Communications.

58. **QIN Z., et al., (2003)** "*PROTECTION APPROACHES FOR DYNAMIC TRAFFIC IN IPMPLS-OVER-WDM NETWORKS*" IEEE Optical Communications, pp, S24-S29.

59. **Kim Y., et al.,** "*GLASS (GMPLS Lightwave Agile Switching Simulator) - A Scalable Discrete Event Network Simulator for GMPLS-based Optical Internet*" white paper, pp, 1-15.

60. **Neil Jerram and Adrian Farrel, (2001)** *"MPLS in Optical Networks"* An analysis of the features of MPLS and Generalized MPLS and their Application to Optical Networks, with reference to Link Management Protocol and Optical UNI.

61. **M. Casado et al., (2006)** ''*SANE: A protection architecture for enterprise networks,*'' in Proc. 15th Conf. USENIX Security Symp., vol. 15, Article 10.

62. **M. Casado et al., (2007)** ''*Ethane: Taking control of the enterprise,*'' in Proc. Conf. Appl. Technol. Architect. Protocols Comput. Commun., DOI: 10.1145/1282380. 1282382.

63. **Open Network Foundation, (2012)** "*OpenFlow Switch Specification*" v1.3.0.

64. **Bondkovskii A, et al., (2016)** "*Qualitative Comparison of Open-Source SDN Controllers*" IEEE/IFIP Network Operations and Management Symposium.

65. **ON.Labs** "*ONOS Open Network Operating System*", http://onosproject.org Online; accessed 2015..

66. **Lenrow D., (2015)** "*Intent-Based Networking Seeks Network Effect*" available online. https://www.sdxcentral.com/articles/contributed/intent-based-networking-seeks-network-effect-david-lenrow/2015/09/

67. **Lenrow D., (2015)** "*Intent: Don't Tell Me What to Do! (Tell Me What You Want)*" available online. https://www.sdxcentral.com/articles/contributed/network-intent-summit-perspective-david-lenrow/2015/02/

68. **Sun X., et al., (2015)** "*SOFTWARE DEFINED OPTICAL NETWORK BASED ON MULTI-LEVEL WDM RING TOPOLOGY FOR INTRA DATA CENTER SWITCHING*" IEEE International Conference on Optical Communications and Networks (ICOCN).

69. **M. Channegowda, R. et al., (2013)** "*Software-defined optical networks technology and infrastructure: Enabling software-defined optical network operations Invited.*," Journal of Optical Communications and Networking, vol. 5, no. 10, pp. A274-A282.

70. **Harai, H., et al., (2014)** "*Optical packet and circuit integrated networks and software defined networking extension*" J. Lightwave Technol. 32(16), 2751–2759.

71. **Das S., et al., (2012)** "*Why OpenFlow/SDN Can Succeed Where GMPLS Failed*" OSA ECOC Technical Digest.

72. **R. Casellas et al., (2015)** "*SDN orchestration of OpenFlow and GMPLS flexigrid networks with a stateful hierarchical PCE invited.*," IEEE/OSA Journal of Optical Communications and Networking, vol. 7, no. 1, pp. A106–A117.

73. **Khot S., et al., (2016)** "*Network Virtualization on Optical Networks*" IEEE International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), pp, 568-573.

74. **Kiran M., et al., (2016)** "*Enabling Intent to Configureure Scientific Networks for High Performance Demands*" Energy Sciences Network (ESnet), Lawrence Berkeley National Labs, CA, USA.

75. **Huang N., et al., (2016)** "*Bandwidth Distribution for Applicitons in Slicing Network Toward SDN on vCPE Framework*" IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS).

76. **F. Pederzolli, et al., (2016)** "*SDN Application-Centric Orchestration for Multi-Layer Transport Networks*" IEEE International Conference on Transparent Optical Networks (ICTON).
77. **A. Mohammed., et al., (2016)** "*SDN controller for Network-aware Adaptive Orchestration in Dynamic Service Chaining*" IEEE NetSoft Conference and Workshops (NetSoft).

78. **Minh P., et al., (2016)** "*SDN applications - the intent-based Northbound interface realisation for extended applications*" IEEE NetSoft Conference and Workshops (NetSoft).

79. **R. Cohen, K., et al., (2013)** "*An intent-based approach for network virtualization*" IFIP/IEEE Int. Sym. IntegratedNetwork Management, pages 42- 50.

80. **Han Y. et al., (2016)** "*An Intent-based Network Virtualization Platform for SDN*" IEEE International Conference on Network and Service Management (CNSM).

81. *https://www.virtualbox.org/wiki/Downloads*

82. *https://www.ubuntu.com/*

83. *https://askubuntu.com/questions/142549/how-to-install-ubuntu-on-virtualbox*

84. *http://mininet.org/download/*

85. **Akhilesh S., et al., (2016)** "*Software Defined Optical Networks (SDONs): A Comprehensive Survey*" IEEE Communications Surveys & Tutorials, pp, 2738-2786.

86. **Xia W., et al., (2015)** "*A Survey on Software-Defined Networking*" IEEE COMMUNICATION SURVEYS & TUTORIALS, VOL. 17, NO. 1, pp, 27-51.

87. **Yi S., et al., (2015)** "*Software-Defined Optical Data Centre Networks*" IEEE China Communications, pp, 1-9.

88. **S. Vissicchio, et al., (2014)** ''*Opportunities and research challenges of hybrid software defined networks,*'' SIGCOMM Comput. Commun. Rev., vol. 44, no. 2, pp. 70–75.

89. **Mohammad R., et al., (2016)** "*Traffic Engineering in Software Defined Networks: A Survey*" Journal of Telecommunications and information technology, pp, 3-12.

90. **A. Giorgetti, et al., (2015)** "*Dynamic restoration with GMPLS and SDN control plane in elastic optical networks Invited.,*" IEEE/OSA Journal of Optical Communications and Networking, vol. 7, no. 2, pp. A174–A182.

91. **Robinson M., et al., (2016)** "*Software Defined Networking for Heterogeneous Access Networks*" IEEE International Conference on Transparent Optical Networks (ICTON), pp,1-4.

92. **Kenneth J., et al., (2014)** "*Software-Defined Access Networks*" IEEE Communications Magazine, pp,152-159.

93. **Zhu P., et al., (2017)** "*Software-Defined Elastic Optical Network Node Supporting Spectrum Defragmentation*" IEEE/OSA Journal of Optical Communications and Networking, VOL. 9, NO. 1, pp, A63-A70.

94. **M. Channegowda, et al., (2013)** "*Softwaredefined optical networks technology and infrastructure: Enabling software-defined optical network operations Invited.*," IEEE/OSA Journal of Optical Communications and Networking, vol. 5, no.10, pp. A274–A282.

95. **Roberto J., et al., (2016)** "*Technical Challenges and Deployment Perspectives of SDN Based Elastic Optical Networks*" IEEE International Conference on Transparent Optical Networks (ICTON), pp, 1-5.

96. **L. Sarakis, et al., (2012)** "*A framework for service provisioning in virtual sensor networks*," in EURASIP J. Wireless Commun. Netw., Special Issue Recent Advances Mobile Lightw. Wireless Syst., 135.