Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Technologies

Bachelor Thesis

Comprehensive assessment of software-defined network implementation using OpenFlow protocol

Author

Arpana Shanta

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Arpana Shanta

Systems Engineering and Informatics
Informatics

Thesis title

Comprehensive assessment of software defined network implementation using OpenFlow protocol

Objectives of thesis

The introduction of this thesis work is going to be a comprehensive study of software-defined networking. At first motivation behind software-defined networking will be discussed. Gradually the thesis work will move down to the main concept, the difference between traditional networking, standardization of SDN, and the building blocks of a basic SDN architecture. A simple SDN network implementation using the OpenFlow protocol will conclude this work. This thesis work will have measurable milestones. These quantifiable small goals will measure the success of this thesis work. History and evaluation will be discussed. Architectural concepts will be explained. Various component of SDN architecture will be described. The operational procedure of SDN will be presented along the way. At last, a simple SDN network will be implemented using the OpenFlow protocol. The final result would be to measure the improvements on SDN network over a traditional networking system.

Methodology

The theoretical part of the thesis will be articulated by the knowledge building from the state-of-the-art literature review. Mathematical analysis of the thesis will also be done by studying different journal and articles in the relevant field. The methods and the technology that supports software-defined networking such as network virtualization and OpenFlow will be discussed in depth. The methods of OpenFlow protocol architecture will be examined in particular. The components of the SDN architecture such as data plane layer, control plane layer, and application layer and their methodologies will be formulated and explained. To observe the practical aspect of the thesis work, a small SDN network will be implemented. Network emulation will be used to reflect the basic architecture of a network traffic generation and reception with a full description of the network. In this network emulation, OpenFlow will be used as the standard communication interface between the control and forwarding layers of an SDN network. Open vSwitch (OVS) will also be used to interconnect different virtual machines within a host and virtual machines across the network. Open vSwitch is an OpenFlow-capable multi-layer virtual switch. To create a realistic virtual network with networking component, Mininet network emulation tool will be used. Iperf and social networking tool will be used to measure the throughput of the network.

The proposed extent of the thesis

40-50

Keywords

Software defined network (SDN), computer network, OpenFlow, Network function virtualization (NFV), Mininet

Recommended information sources

Ben Pfa, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Mart'sn Casado. The design and implementation of open vswitch. In Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, NSDI'15, pages 117{130, Berkeley, CA, USA, 2015. USENIX Association

Implementation of Software-Defined Networks Using Open-Source Environment. Petar ČISAR, Dragan ERLENVAJN, Sanja MARAVIĆ ČISAR. 2018. Supplement 1, 2018, Zagreb : Technical gazette, Portal of Croatian scientific and professional journals, 2018, Vol. 25. ISSN 1848-6339.

J. E. Gabeiras T. Braun, M. Diaz and T. Staub. End-to-end quality of service over heterogeneous networks. Springer, February 2008

kreutz2014softwaredefined, title={Software-Defined Networking: A Comprehensive Survey}, author={Diego Kreutz and Fernando M. V. Ramos and Paulo Verissimo and Christian Esteve Rothenberg and Siamak Azodolmolky and Steve Uhlig}, year={2014}, eprint={1406.0440}, archivePrefix={arXiv}, primaryClass={cs.NI}

PM, Rekha & .M, Dakshayini. (2015). A Study of Software Defined Networking with OpenFlow. International Journal of Computer Applications. 122. 5-12. 10.5120/21694-4798.

Shalimov, Alexander & Zuikov, Dmitry & Zimarina, Daria & Pashkov, Vasily & Smeliansky, Ruslan. (2013). Advanced study of SDN/OpenFlow controllers. ACM Proc. CEE-SECR. 10.1145/2556610.2556621.

Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices. Menth, Wolfgang Braun and Michael. 2014. 302-336, Tuebingen : Future Internet 2014, www.mdpi.com/journal/futureinternet, 2014, Vol. 6. ISSN 1999-5903.

Expected date of thesis defence

2020/21 SS – FEM

The Bachelor Thesis Supervisor

Ing. Tomáš Vokoun

Supervising department

Department of Information Technologies

Electronic approval: 29. 7. 2020

Ing. Jiří Vaněk, Ph.D.

Head of department

Electronic approval: 19. 10. 2020

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 01. 03. 2021

Declaration

I announce that I have worked on my bachelor's thesis entitled "Comprehensive evaluation of software-defined network deployment using the OpenFlow protocol" and have used only the references listed at the end of the thesis. I declare that this thesis does not breach any person's copyrights.

In Prague on 12.03.2021 ______________________________

Acknowledgment

I would like to express my deep-felt gratitude to my bachelor thesis supervisor, Ing. Tomáš Vokoun. I could successfully complete this work because of your continuous guidance and valuable suggestions.

I would also like to convey my heartfelt thanks to Goutam Kumar Saha and Md. Rafique Ahasan Chawdhery (Ph.D. Candidate), department of Agroecology and Crop Production, CZU, for encouraging and motivating me during this thesis work. Without your help and support, I could not finish this work.

My biggest thanks go to my parents for bringing me into this beautiful world. I would not be in this place without your teachings, blessings, and immeasurable love. You always believed in me and supported me in all situations in my life.

And finally, I must thank all my closest friends Md. Shahadat Hossain Sagor, Md Alamin, and Abdullah Al Mahmud for being there with me always and supporting me throughout my academic years.

Comprehensive assessment of software-defined network implementation using OpenFlow protocol

Abstract

In recent times, the increasing traffic demand which leads to a more complex network obsoleting traditional networking methods. To deal with this problem a new paradigm in networking called software-defined networking (SDN) has emerged. In this thesis work, a comprehensive overview of SDN has been presented along with its simple implementation by a well-known SDN protocol called OpenFlow. This research work describes the architecture and components of SDN and OpenFlow. During the work, it has been seen that how SDN can minimize the difficulty of handling large and distributed networks. A small SDN network has been implemented with the help of the Mininet network emulation tool using Open flow protocol, Open vSwitch, and Ryu controller. Several tests have been carried out to test the performance of SDN in comparison to the traditional network. In this implementation, it has been seen that routing convergence time in SDN network is less compare to traditional network as the topology size gets bigger. This also suggests that the SDN network converges more rapidly than the conventional network and the routing time for convergence is strongly affected by the changing topology size.

Keywords: Software-defined network (SDN), Computer network, OpenFlow, Network function virtualization (NFV), Mininet

Comprehensive assessment of software-defined network implementation using OpenFlow protocol

Abstrakt

Aktuálním problémem, souvisejícím se stale se zvyšujícími se požadavky na objem přenesených dat, je přechod k vice komplexním metodám řízení sítí a upozadění tradičním metod. Jedním z řešením tohoto problému je zavedení SDN (softwarově definovaných sítí). V této diplomové práci je představen komplexní přehled SDN spolu s běžně užívaným protokolem OpenFlow. Výzkumná část práce popisuje architekturu a komponenty SND, spolu s OpenFlow. V průběhu práce bylo ukázáno, jak může koncept SDN zjednodušit řízení a provoz rozsáhlých sítí na distribuční úrovni hierarchického modelu.

Pro účely praktického ověření, byla v v simulačním nátroji Mininet vytvořena virtuální síť se zavedeným OpenFlow protokolem a komponenty Open vSwitch a Ryu řadičem. Bylo realizováno několik experiment pro porovnání výkonnosti takové sítě s tradičním řešením. Výsledky ukazují, že síť na bázi SDN řešení konverguje u komplexních sítí k stabilnímu stavu znatelně rychleji, než tradiční řešení. Dalším zjištěním je, že doba potřebná ke stabilizaci vnitřního směrování je velmi výrazně ovlivněna změnou topologie sítě.

Klíčová slova: Softwarově definovaná sítí (SDN), Počítačová síť, OpenFlow, Virtualizace síťových funkcí (NFV), Mininet

Table of content

List of figures

List of tables

1. Introduction

The process of carrying and exchanging data from any communication endpoint to another by a shared medium in an information system is called computer networking. Often these communication endpoints are referred to as nodes. The definition of nodes depends on the type of network. A computer, switch, or router can be referred to as a node in computer networking. These nodes are connected by ethernet or optical cables and most recently by wireless mediums. Computer network not only consists of these nodes but also different infrastructures such as software and policies. Networking also involves the design, construction, and implementation. A computer network also requires regular operation and maintenance. To establish a successful connection between two nodes, a computer network relies on different standard protocols to perform many functionalities uniformly. The process may vary for different data types and different requirements of the user. These processes have no change based on the underlying hardware. (1).

The socio-economic development in the past few decades has changed people's lives in many aspects. One of the most important factors of the development is the computer network. A small computer network system known as LAN (local area network) is widely adopted in almost every household. LAN plays an important role in the daily operation of hospitals, educational institutions, and the development of overall people's lives through science and technology.

While establishing communication between network endpoints, the network should discover routes or data flowing paths. In a conventional networking environment, a network operator runs network protocols like OSPF (2), RIP (3) to obtain the optimum routes from point A to point B. Routing is a widely adopted option on a computer network. But the protocols used in the traditional network is often forcing the network operators to build their network on harsh and fixed options. In this way, they are also losing a transparent view of their network. The main reason for that is these routing protocols are not open sourced and written by the hardware vendors and implemented as classified software. The network operator cannot see inside the codes to dive deep into the functionalities to match with their custom routing requirements. These protocols often tailored by the hardware vendors such a way that it loses the interoperability between different vendors. Thus, the network operator gets binds to a particular supplier and they need to be dependent on suppliers for scalability (4).

In the past few decades, internet consumption has increased by many folds. Which led to a higher bandwidth backhaul network. The network needs to provide a path that can carry this huge amount of data. To enable this, the network becomes more complex and needs to be easily functional and manageable. Not only the complexity of the network grows but also the number of devices deployed increases as well. And to implement this complex network deployment, configuration and maintenance are becoming more and more cumbersome. The whole operation is also becoming time-consuming and error prone. To meet the user traffic demand, a network operator might also want to scale his network and launch a new network service. This also requires space and power to accommodate the newly added hardware. This is becoming increasingly difficult. To add a new device in the network involves capital investment as well. With the increased complexity it is also often difficult to find a network administrator to operate and maintain such a complex network. Another problem with these hardware-based network devices is that this device is made for a limited period of time and often reaches its end-of-life cycle. In this case, a network operator needs to go through the same process starting from procuring to deployment. In between, they also need to design, integrate, and made changes to the existing network architecture. This often brings them no extra profit worth of these much hustles. (5).

In these circumstances, the demand for a future network that is easy to configure, scalable, and simplifies network operation and maintenance has emerged. The increased demand for improvement in computer networking is driving researchers and scientists to develop improved methodologies to overcome the existing problems in a conventional network. Software-defined networking (SDN) is one of the innovations of future networking architecture.

2. Objectives and Methodology

Objectives

The main aim of this thesis is to show the improvements of the SDN network over a traditional networking system. History and evaluation will be discussed. Architectural concepts will be explained. Various components of SDN architecture will be described. The operational procedure of SDN will be presented along the way. At last, a simple SDN network will be implemented using the OpenFlow protocol.

The partial objectives of the thesis are:

1. Study of SDN, its architecture, and protocols.
2. Comparison with traditional network and its economic benefits.
3. Simulation of SDN network using custom topology.

Methodology

The theoretical part of the thesis will be articulated by the knowledge building from the state-of-the-art literature review. Mathematical analysis of the thesis will also be done by studying different journals and articles in the relevant field. The methods and the technology that supports software-defined networking such as network virtualization and OpenFlow will be discussed in depth. The methods of OpenFlow protocol architecture will be examined in particular. The components of the SDN architecture such as the data plane layer, control plane layer, and application layer and their methodologies will be formulated and explained. To observe the practical aspect of the thesis work, a small SDN network will be implemented. Network emulation will be used to reflect the basic architecture of a network traffic generation and reception with a full description of the network. While simulating the network, OpenFlow will act as the standard communication interface between the control and the forwarding layers of an SDN network. Open vSwitch (OVS) will also be used to interconnect different virtual machines within a host and virtual machines across the network. Open vSwitch is an OpenFlow-capable multi-layer virtual switch. To create a realistic virtual network with the networking component, the Mininet network emulation tool will be used. Iperf and social networking tools will be used to measure the throughput of the network.

3. Literature Review

Traditional network

In conventional networking, many devices transfer data while connected to each other. The decision-making process or the brain of the device i.e. network engine and the traffic forwarding parts are ties into the same box. A traditional networking device like switch and router has a control plane that decides upon the action taken on an arrived packet and the forwarding path takes the action.

3.1 Software-defined network- a new paradigm

In the early 2000s, as the traffic volumes increased and an increased focus on network dependability and productivity caused network operators to look for improved alternatives to some network management tasks such as control of the paths used to deliver traffic (commonly known as traffic engineering). In all respects, this implies that traditional routing rules for traffic control operations are simple and best. Software-Defined Networking (SDN), a new advanced method of network technology (6).

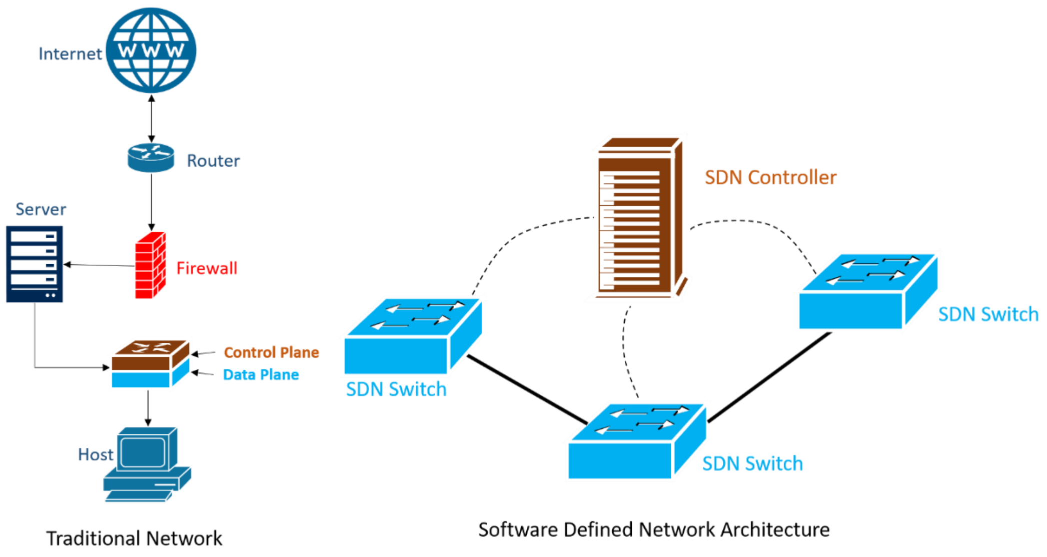


Figure 1: Traditional networking and SDN (7)

A special characteristic that makes SDN popular is its ability to create the network programmable. SDN divides the control plane from the data plane, which is normally joined in traditional network devices (6). Software-defined networking (SDN) has been discovered as a strong alternative solution.

Numerous problems of conventional networking can be solved by SDN. Challenges include increased network expansion, increased traffic demands, network troubleshooting, lack of programmability, and so on. This technology enables the operator to control, track and monitor network equipment using the software application and improves the reliability of the network (7).

Centralized Architecture

The concept of making a network programmable is the main driving force behind the innovation of software-defined networking. With the help of SDN, designing and managing networks has become more and more innovative. SDN can cater to today’s network requirements because SDN is dynamic, controllable, and easily adaptable to changes (8). The architecture of SDN has been described in detail next sections.

SDN layers in OSI Model

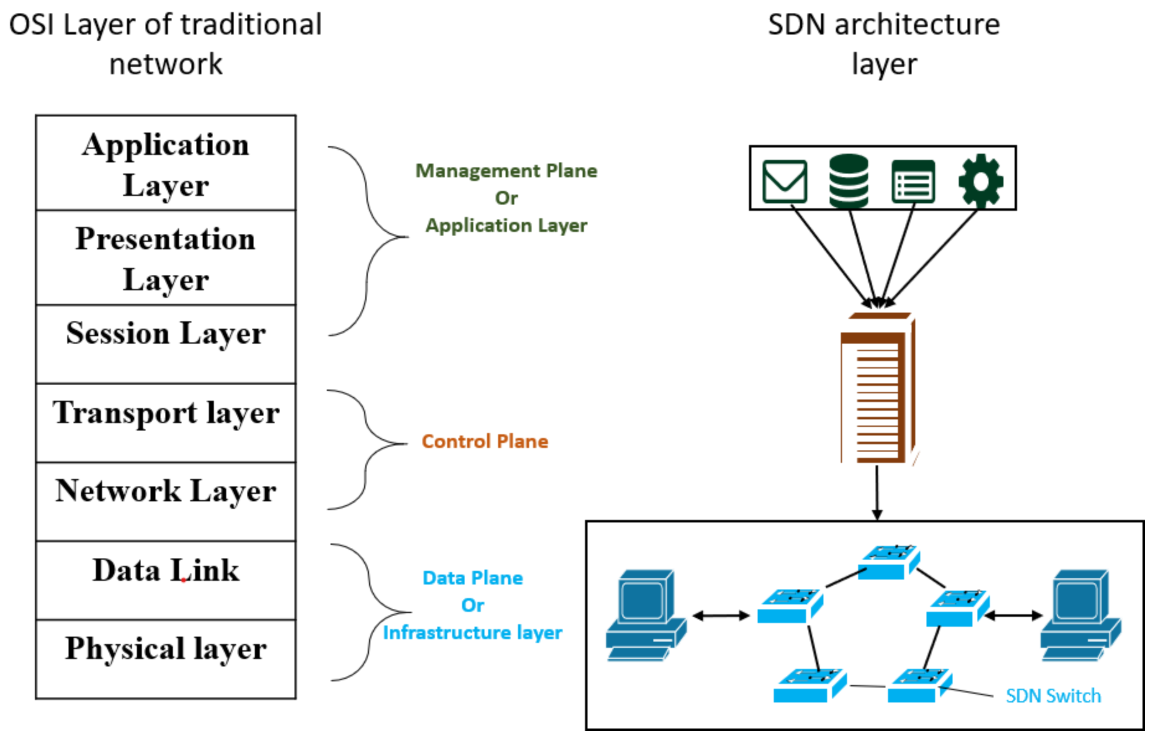


Figure 2: SDN layer relation with OSI model

In figure 2, the comparison of SDN layers with the traditional OSI model can be seen. The data link and physical layer work as the data plane or the infrastructure layer in the SDN model.

SDN control plane work equivalently to the transport layer and network layer in the OSI model. The top three layers in the OSI model works as the Application layer or the management layer of a Software-defined network. Though SDN is an application-based concept some might think this should lie in the application plane in the OSI model. But SDN is a network design with distinguishable different planes in its architecture.

Characteristics of SDN architecture

Following are some characteristics of SDN architecture:

Table 1: Advantages of SDN

Characteristics	**Advantage**
Programmability	Network control can be directly programmable as it is decoupled from the forwarding functions.
Agile	It enables the network administrator to dynamically adjust traffic flow on the network to meet the network demand. This process can also become intelligent by implementing machine learning algorithms on the application and controller layer.
Centrally controlled	In software-based controllers, the network features can be consolidated with a global network vision. This controller is considered to be a single logical switch to the program and policy engines.
Programmable configuration	Because the SDN programs do not depend on proprietary software, network administrators can write their programs to configure, manage, secure, and optimize network resources. Because the SDN programs are automated and dynamic this can be done very quickly as well.
Open-standard-based and vendor-neutral	SDN is developed and implemented by the network community and is initiated by many volunteer sources. SDN simplifies the architecture and function of the network by introducing Open Standards, as it is not vendor-specific equipment and protocols. (12)

Table 2: Disadvantages of SDN

Characteristics	Disadvantages
Centralized control	Without redundancy, the single controller is a single point of failure. If the controller becomes not functional, the whole network will be affected.
Single point of control	With the controller being the single point of control, all the updates happen from the controller hence it might be the bottleneck of the network. Pushing every small update to all the NEs from the controller might end up in huge overhead.

Components of SDN

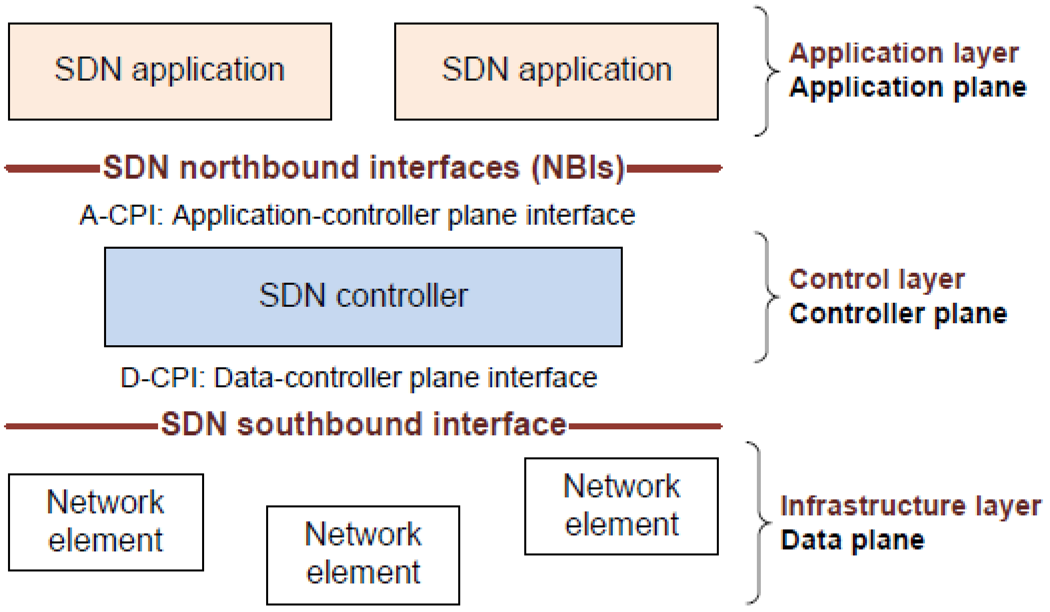


Figure 3: Provides an overview of the SDN model and its components (12)

Data Plane: Referred as the infrastructure layer in the ONF paper (Open Networking Foundation, 2014), this layer of the SDN architecture consists of a set of network elements. Usually, different infrastructure nodes which support SDN protocol lies in this layer. These nodes are responsible for transporting and processing Data packets. The instructions come from the SDN controller plane.

SDN controller passes all the decisions and actions to be taken by the Data-controller plane interface (D-CPI) to the infrastructure nodes. DCPI defines the way the control plane communicates and takes direction from the controller. In this process network elements in the data or forwarding plane makes the necessary changes to meet the user or networking requirements. Information exchanged between the controller and the data plane includes controlling information, routing policy, and resource configuration (9).

Data plane resources are reflections of the physical network elements and their capabilities. The data plane is simply a system of a set of nodes that has capabilities of traffic manipulation like consumption, produce, store, dropping or forwarding, etc. Links between the NEs interconnect them. NEs are the face of the network that connects clients and other network nodes. These are external data plane ports. It is also important to mention as per the advantage of SDN that one controller can control more than one data plane (9).

The data plane supports various models such as (10)

- Packet switching and forwarding models such as IPv4, IPv6, Ethernet, etc.
- Optical transport network, multi-protocol label switching, and other circuit switching models.
-Wireless communication models such as integration of wireless technology, characterization of wireless interfaces, their flows, and handover models.
-LTE support models and advanced packet core models (14).

Networking elements (Forwarding plane): The elements in the lowest layer are called networking elements or NEs. NEs are conventional hardware like switches or routers. These NEs need to support programmable interfaces like OpenFlow (see section 3.2). These devices can be hardware or can be software switches like Open vSwitch (see section 3.2) Hardware switches provide high performance with higher bandwidth and software switches provides support for flexibility in sudden changes in the network environments (6).

Southbound interface: With the help of this component SDN controllers communicate with network forwarding devices. Packet handling instruction, operating alarms, and notification of simple network management protocol (SNMP) are directed from the controller to the forwarding plane by these interfacing components. OpenFlow protocol also has a set of packet handling instructions that convey through the southbound interface. There is also another popular protocol is the Open vSwitch Database Management Protocol (OVSDB) (11).

Control Plane: The control plane is depicted by an SDN controller. It is a software that runs on a hardware/server This plane is also called the brain of the SDN architecture. This layer lies in the middle, above the data plane, or the infrastructure layer. The core responsibility of this plane is to program and manage the forwarding plane. It gets information about the packets from the forwarding plane and gives instructions to act on that particular packet. The instruction includes routing and processing of the packet. With the help of southbound interfaces, software controllers communicate with the data plane. There might be more than one software controller works in tandem or in a group to send control information to one or more cluster of NEs (12).
When there is only a single controller in a network it creates a single point of failure. This single point of failure threatens the stability of the whole network. If the controller becomes dysfunctional, the whole network ceases operation. To avoid this problem, a multiple controller network can be designed.

A network operator can design controller setup by following:

1. Centralized – A network setup can be with a single centralized controller which needs to have a global view of the network. This is a basic model. This way handling the network becomes easier. Centralized controller setup can be adapted to small to medium-sized networks.
2. Distributed- This type of setup usually involves multiple controllers in the network. Multiple controllers bring redundancy, resiliency, and enhanced output to the network. Each controller can run the entire network or a part of the network. These controllers communicate with each other after they have received a packet to form an end-to-end route out of their domain.

The most widely used is centralized architecture. Controllers using this kind are also designed with several threads as extremely concurrent devices. Except for data centres and similar large networks, their performance is therefore satisfactory when using a multi-core system. Application isolation can also increase resiliency to a certain degree with some of these controllers.

Network Operating System: The networking operating system is also called the SDN controller. It runs the core services to run the network. The key core services are topology, inventory and statistic service, and host tracker. The topological graph includes information on how the NEs are interconnected with each other.

By instructing switches to use link layer discovery protocol (LLDP) packets to discover the location and position of a particular node in the network, the network topology can be learned. Different specialized packets can be used to find out various information of the NEs on the network like the version of the OpenFlow running, capacity, and capabilities. A statistics service can also be run to gather information about the incoming and outgoing traffic in a particular interface, flow counters on the interfaces, and information about the flow table. By coordinating with the virtual machine platform, a host tracker can also locate any NE in the network by incepting the flowing traffic and by using IP or MAC addresses of the Hosts (6).

Application-controller plane interface (A-CPI): On the southbound part of the architecture, A-CPI enables the SDN network applications to hook into the SDN controller. One of the core responsibilities of this component is that to provide a simplified abstraction of the underlying network infrastructure to the upper layer. This often represents the bottom layers as a large switch to the SDN applications. For this application to successfully collaborate with the controller, there might be a need for native plugins running collocated with controllers. It uses a programming language-based application program interface (API). Using the directive API calls towards the controller, these applications can control network behavior and collect information.

SDN Application: SDN network application can control network behavior and implement network policies as well as execute various other network functionality. A network administrator can create network programs to meet the organization's demand and network requirements. SDN is providing a programmable abstraction to make this happen (6).

3.2 OpenFlow

OpenFlow is the communication protocol of SDN. This is considered to be the main invention that has led to the further developments of Software-defined networking. OpenFlow protocol has been standardized by the Open Networking Foundation (ONF) (9). This organization is operated by the users. The definition of OpenFlow is the first standard communication protocol of SDN. ONF has defined in OpenFlow how the interfacing and communication between the control and forwarding layer will work. Since ONF is an open-source project, it encourages the user to build various SDN applications that evolving SDN to its mature state.

Architecture of OpenFlow

In figure 4, the architecture of the OpenFlow network is shown. The main elements are OpenFlow supported switches and controller. The OpenFlow switch generalizes the Ethernet switch, which distinguishes the data layer from the control plane. This has been abstracted by the table of flows. Communication with the SDN controller is done via a secure channel (13).

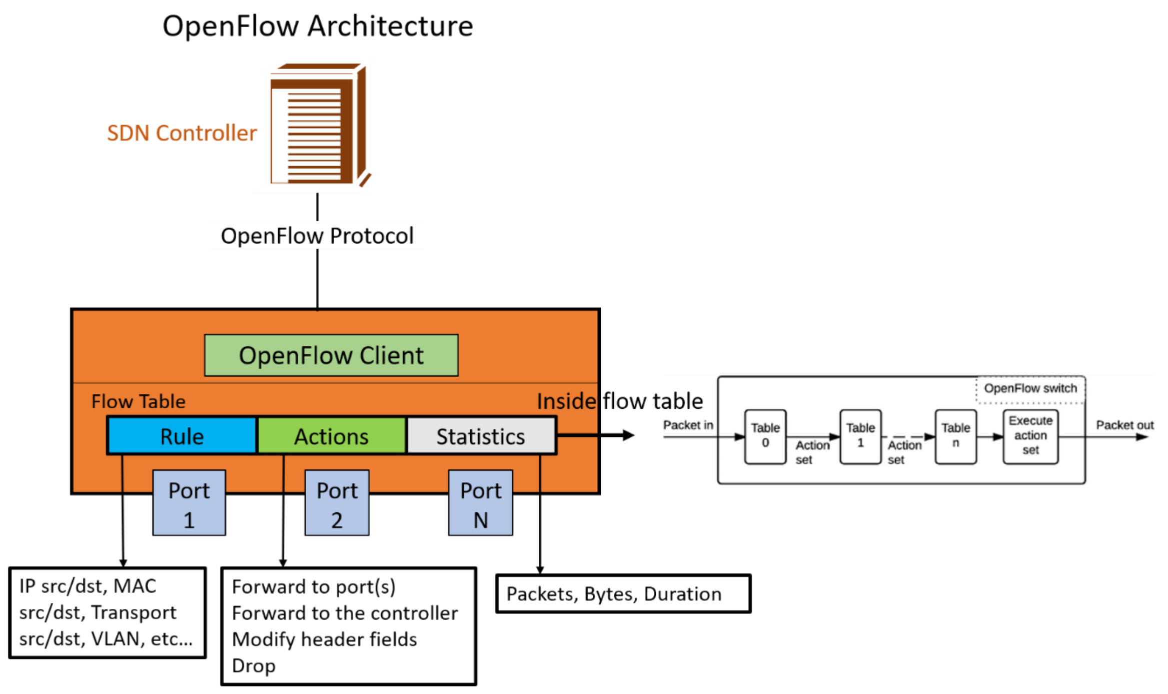


Figure 4: OpenFlow Architecture

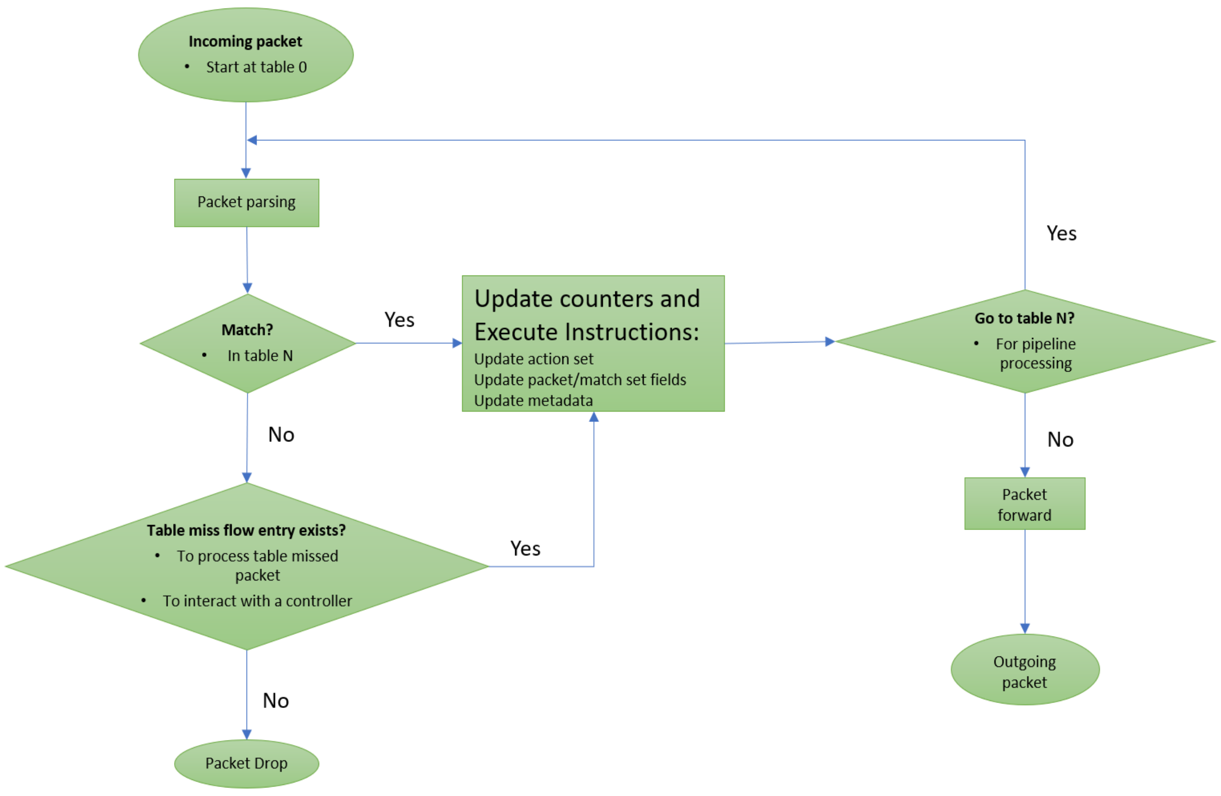
OpenFlow allows the data plane network elements to work as a simple network device that can only manipulate packets according to the instructions given by the controller. ForCES (Forwarding and Control Element Separation) is an interface standard that allows logic to be distributed across to single, multiple, or all of the network elements. OpenFlow protocol aims to have a centralized control plane (23).

OpenFlow protocol architecture primarily has three components.

- Data plane network consists of OpenFlow switches.
- Control Plane built up by OpenFlow controller.
- A secure channel that connects the OpenFlow switches with the controller. (14)

OpenFlow Switch

OpenFlow switch is consists of a flow table and group table. Flow table can be one or more. These tables are responsible for packet lookups and packet forwarding. The OpenFlow switch interacts with the controller through a protected channel. It can have one or more secure channels. In order to handle the OpenFlow switch, the controller uses the OpenFlow switch protocol. The controller can modify the flow data in the flow tables through the OpenFlow switch protocol. It can add, delete, or update an entry in the flow table. Depending on the flowing packets, the update can also happen reactively. Each flow table in a switch has several flow entries. The flow table also has a set of matching fields, counters, and instructions are given for each flow entry. If the packet matches the fields, the switches execute the instruction. It has group tables as well. The group table contains group entries. The set of action buckets for each group entry is based on a group type with unique semantics. Action on packets is done from one or more action buckets before being sent to the group (12) (8).



Incoming packet
• Start at table 0
Packet parsing
Yes
Match?
• In table N
Yes
Update counters and Execute Instructions:
Update action set
Update packet/match set fields
Update metadata
Go to table N?
• For pipeline processing
No
No
Packet forward
Table miss flow entry exists?
• To process table missed packet
• To interact with a controller
Yes
Outgoing packet
No
Packet Drop

Figure 5: Flowchart detailing packet flow through an OpenFlow switch (12)

OpenFlow Channel

This channel interface connects the OpenFlow switches to the SDN controller. A single controller can set up many connections to many switches. A switch also can have connections with multiple controllers. This multiple connection setups can ensure redundancy in the event of a network component failure. It also helps when the network element is overloaded. Firstly, the switch initiates the connection setup with a controller using the host and port setup configure in the switch. Once it establishes the connection with the controller, communication can happen both ways. The OpenFlow protocol has three communication types: control-to-switch, asynchronous, and symmetric; each with many subsets (9). Messages to the switches are initiated by the centralized controller. In this process, the controller sometimes gets a confirmation from the switch. Sometimes this confirmation is not needed. Asynchronous messages can be transmitted from a switch without a controller asking for it. Symmetric signals are sent from both directions without any request. Any time any side can initiate communication if needed (6).

Open vSwitch

This is an OpenFlow protocol-supported switch. This is a multilayer switch. It can operate in layers 2, 3, and 4. It is possible to connect virtual machines inside a host by this switch. It can also connect virtual machines across separate hosts through networks. It can also be used in dedicated switching hardware. This is an important component of an SDN solution. It supports multiple platforms including Linux and FreeBSD. An Open vSwitch can be set up locally or remotely (6).

OpenFlow Protocol

This is a network communication protocol. OpenFlow protocol allows network administration from a centralized location, SDN controller. This is the main protocol that connects the control plane and the Data plane. This protocol defines the control messages that pass to the OpenFlow switches through the secure channel from the controller. Both the controller and the switch must be able to recognize and produce the format of the messages. The format is well defined in the OpenFlow protocol.

OpenFlow protocol is part of the OpenFlow specification and it must be applied to the OpenFlow control plane as well as on the OpenFlow switch. This protocol defines the rules that will be followed by the switches. This rule will be followed if certain conditions are met when traffic flows through the network. For example, a matching condition could be an entry port, source or destination IP or MAC address, and or different security protocol specifications. These rules, which are also called flows, are listed on the flow table in the OpenFlow switch. As mentioned before, these flow entries can be created by both the controller and the switch. A controller can push a single flow entry to a single or multiple switches across the network. Sometimes if any packet arrives in the switch that has no entry on the flow table, the switch sends a special message with the packet details to the controller and the controller sends back the flow for that packet as a reply. This way the controller can control the behavior of the network. Many researchers think that OpenFlow protocol is the first standard protocol of SDN (18).

3.3 Network Function Virtualization (NFV)

Network function virtualization (NFV) is the process in which network services can be virtualized. Traditionally, network services run on proprietary hardware such as routers, switches, and firewalls, etc. The services can be packaged as a virtual machine (VM). The VMs can be then run on commodity servers instead of proprietary hardware.

The advantage of NFV is that it is not needed to have dedicated hardware for each network function. NFV solves the problem of scalability and agility of traditional networks. It can also deliver new network services and applications on demand. This process does not require additional hardware resources (15).

NFV architecture

To define standards for NFV implementation, this standard NFV architecture is proposed by the European Telecommunications Standards Institute (ETSI). The component of this architecture advocates better stability and interoperability.

NFV has the following components in its architecture:

- Virtualized network functions (VNFs): These software applications deliver network functionalities such as network configuration, Active directory and domain services, and file sharing, etc.
- Network functions virtualization infrastructure (NFVi): NFVis contains elements of virtualized infrastructure. The hardware infrastructure like compute engine, storage, and

networking components are needed to support the overlaying software applications. On top of it, a hypervisor like KVM or a container management platform like Docker is needed to run network applications.

- Management, automation, and network orchestration (MANO): To administer the NFVis and VNFs, the MANO framework is being used (19).

NFV Infrastructure (NFVI)

NFVI is a combination of both hardware and software. This environment host the VNFs. Commercial-off-the-shelf (COTS) acts as the physical resources. It provides computing hardware, storage, and network nodes and links that give VNFs all the resources to run. A virtualization layer separates the virtual resources from the physical resources. In a large data center, the virtual resources can work together with multiple VMs connected through virtual nodes and links. A virtual node also has switching and routing capabilities like the traditional networking hardware (5).

Software-defined networking (SDN) and NFV

Both the technologies are very similar and independent of each other. Both of these have a common functionality as virtualization. However, the differences among them are on functionalities and abstract tools. SDN creates an environment where network functionalities are separate, and control is centralized. Whereas NFV abstracts network functions from the hardware. SDN is supported by NFV with the infrastructure where SDN can run the network control software. By combining these two technologies one can create a network architecture that is more flexible, programmable, and used resource efficiently (15).

Spanning Tree Protocol (STP)

Spanning Tree Protocol is a Layer two protocol defined by IEEE 802.1D standard. A spanning tree is employed in bridged networks to dynamically verify the simplest path from a source to a destination while avoiding loops which may cause bridges to constantly forward constant frames. STP creates a hierarchical tree that spans the whole network, as well as all switches. Additionally, it determines all unnecessary routes and makes only 1 of them active at any given time. In case of any links fail STP allows the network to provide backup paths. The STP becomes an essential tool to avoid a broadcast storm. When a packet or frame circles around the network endlessly because of an unidentified destination, the switch suffers from a broadcast storm. STP ensures only 1 path exists between any 2 stations. The algorithm accomplishes this by sending special messages containing data (information) regarding the bridges. These messages permit the bridges to elect one bridge as the root of the spanning tree and additionally figure the shortest path from themselves to that bridge.

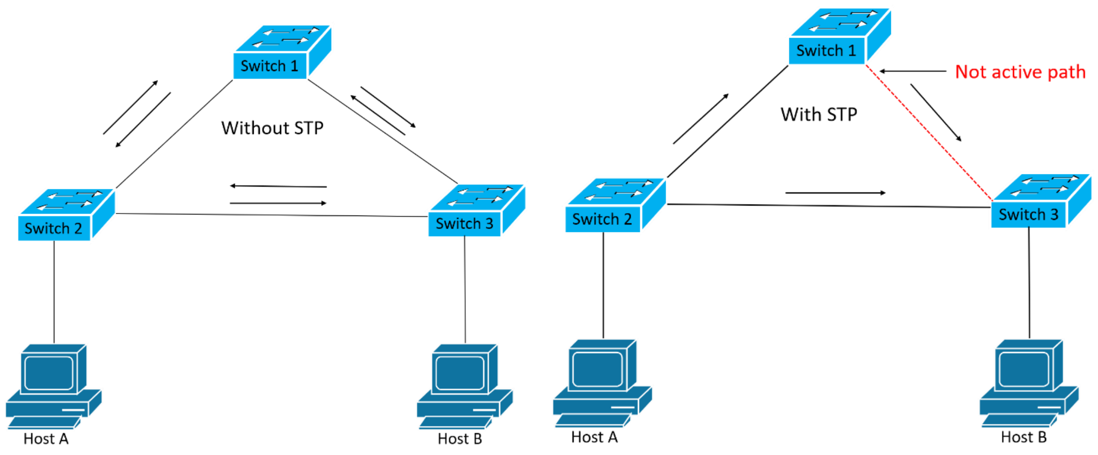


Figure 6: STP disables connection that creates a loop

In figure 6, there are two hosts A and B, and three switches S1, S2, and S3. When host A sends a transmit message to switch 1. Switch 1 cannot locate the destination address in its flow table, so it will forward the message to all connections except the one that the message originated from (S1). Switch 2 will accept the message and respond in the same way as Switch 1 and forward the message to Switch 3. As a result, switch 3 receives two copies of the original message and forwards each copy to host B and the other switches. Even if the transmitted message has now reached Host B, there are copies of the messages passing across the network that can produce even more copies because of the existing loop. This is considered a broadcast storm which has a huge negative effect on the output of the network.

Routing convergence during Link failover

Link failure or network disruption is getting more attention with the ever-increasing network in terms of size and complexity. In the traditional network environment, link failure has been taken care of by the traditional layer 2 and layer 3 switching and routing algorithms. The performance of the network depends on how fast the network converges to alternate paths in order to avoid data loss.

Many researchers have already presented the comparative analysis of the routing protocols in terms of network converging time (16) (17) (18). They have also presented how the network performance depends on this matter. In this paper (17) , the researchers have shown the comparison between OSPF routing protocol with OpenFlow protocol in a network failover scenario. They have shown routing convergence mechanism in both traditional routing algorithm OSPF and SDN protocol OpenFlow.

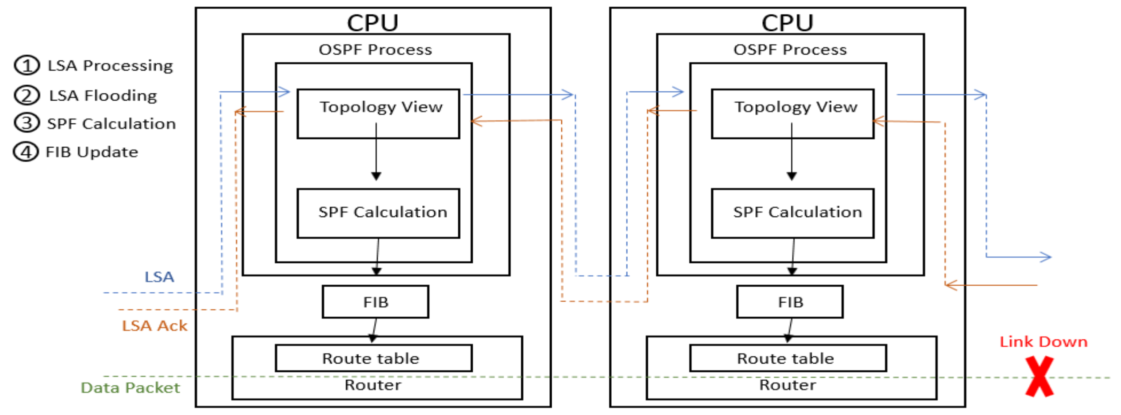


Figure 7: OSPF Convergence process (21)

In figure 7, the routing convergence process of OSPF can be seen. In this process the routing convergence time is calculated in the following way:

Time = *link down discovery time* + *LSA delivery time* + *SPF execution time* + *FIB update time* (17)

It can be seen from this equation that LSA delivery time and SPF execution which happens in every router in the network play an important role in the convergence time calculation. If the network size is larger this takes more time adding up to the overall convergence time.

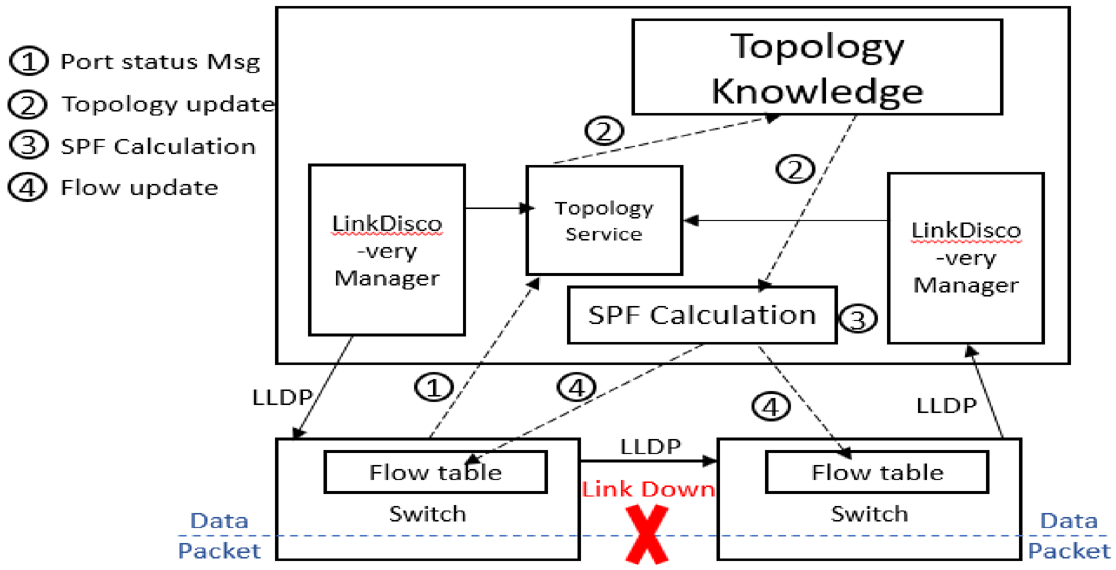


Figure 8: SDN convergence process (21)

For SDN, the reason behind faster convergence can be understood from figure 8. The network convergence time can be calculated from the following equation:

Time = link down discovery time + topology update time + SPF execute time + flow table update time (17)

This equation describes the time needed for the network to converge in case of link or node failure. It can be seen from the equation that the key factors while calculating the network convergence are topology update, SPF execution time, and flow update time. All of these are done by the centralized controller. These operation does not happen in the network-wide all elements. The network elements are also known as the SDN switches report the port status through LLDP and later the topology service updates the topology view of the network and does the SPF calculation. The flow table will also be updated accordingly. Later, the controller will push the updated flow table to the switches. Because most of the operations are happening in the controller thus convergence time in SDN become much faster than traditional network (17).

The economical aspect of SDN

Software-defined networking provides great opportunities for performance enhancement and costs savings. The cost of day-to-day activities can be drastically decreased with improved visibility and versatility to reassign resources on requests. A high potential for dynamically allocating network functions over network nodes compared with conventional networks. However, it is also much time-consuming to test and experiment and deploy and is incompatible with business needs. Current networks at this stage are too costly and too difficult to handle.

- Complete operation visibility to speed up the diagnosis and restoration.
- Increased automation eliminates configuration errors.
- Use orchestration to reconfigure services without interference dynamically. (24)

4. Implementation

The following methods and tools have been used to implement the test scenarios described below:

Mininet: Mininet is a stable network simulation platform for reliable testing. It was required a robust network simulation framework to build a secure/reliable testing environment. Mininet is a pioneering method in the field of network simulation. Mininet is a network emulation tool that allows the creation of a realistic virtual network with real networking components. Mininet operates with standard Linux network applications and OpenFlow support is available for Mininet switches. In this thesis work, Mininet will be used to develop the network. In the end, traffic testing will also be carried out to test the SDN functionality. I created this topology without adding unnecessary complexity which fits the needs of the project. Such topologies can be described in various methods. For my setup, I have used the Mininet libraries for Python to describe them.

The start of Mininet (in Ubuntu) is as easy as downloading from the repository via command line and running $sudo mn in the terminal. Various other parameters can also be included like the number of switches, hosts, topology types, etc. with instructions. But in the end, coding in a Python script is simpler.

OpenFlow protocol: OpenFlow is the basic interface for communication between the control and forwarding layers in the SDN architectures. When the traffic requirement comes to the controller, the control layer software decides how to route the traffic to meet the requirement. OpenFlow protocol conveys the information to the transport layer (e.g. switch) devices. OpenFlow protocol is a standardized protocol to interact and manipulate the devices in the forwarding plane.

Ryu: Ryu is a highly modular, small SDN controller written in Python. The core of Ryu is smaller than other controllers. Ryu offers a well-defined API with software components that facilitates the development of new network management and control applications. Ryu makes these operations simple for developers. To manage network devices Ryu supports different protocols like OpenFlow, Netconf, and Off-config. This controller is highly recommended after the analysis based on Analytic Hierarchy Process (AHP). For the project setup, I have chosen the Ryu controller for being the one recommended by OpenFlow developers.

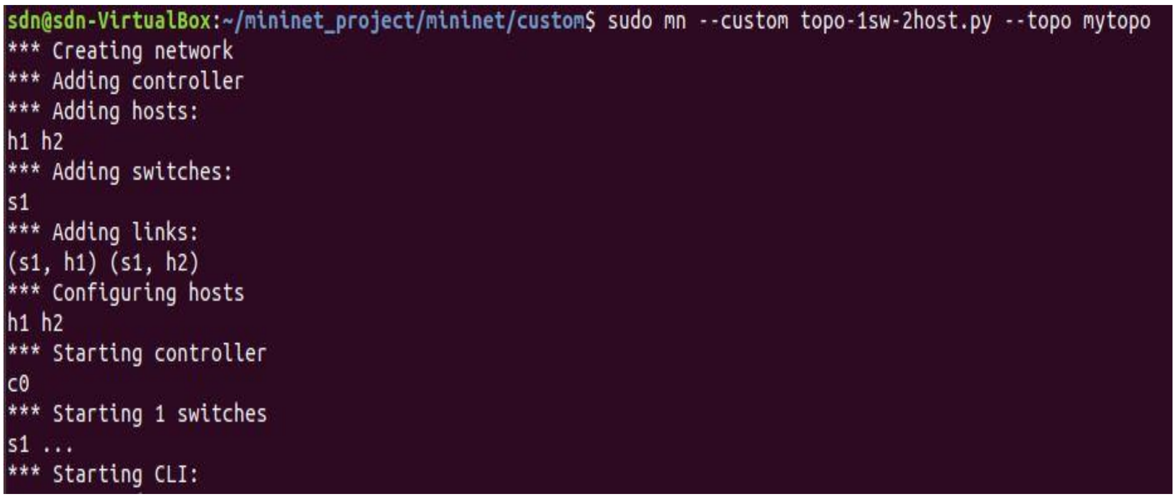
Its position in the project would be to serve as a controller, but I won't create any application on it because it would surpass the goal of the thesis.

Open vSwitch: Open vSwitch (OVS) is an OpenFlow-capable multi-layer virtual switch. It can be used to interconnect different virtual machines within a host and virtual machines across the network. Since it supports various conventional switching features, depending on the size of our network, Open vSwitch will be used during the emulation.

Iperf: Iperf is a networking tool to measure the throughput of the network. The transmitting protocol (TCP) traffic can be initialized as well as user datagram protocol (UDP). These network utility tools will be used to carry out the testing.

Simple Topology

In the first experiment (Fig.9), a minimum topology is built and deployed with Mininet. The aim is to check the basic functionality and the efficiency of all the components in SDN. This topology consists of only one switch with two hosts.



```
sdn@sdn-VirtualBox:~/mininet_project/mininet/custom$ sudo mn --custom topo-1sw-2host.py --topo mytopo
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

Figure 9: Network deployment with mininet (simple topology)

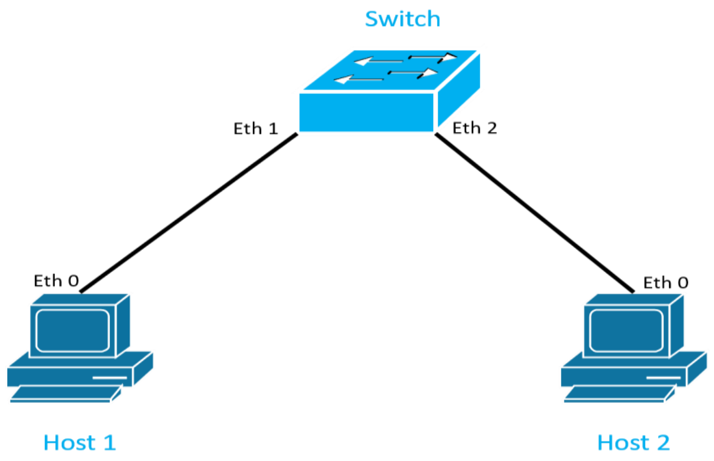
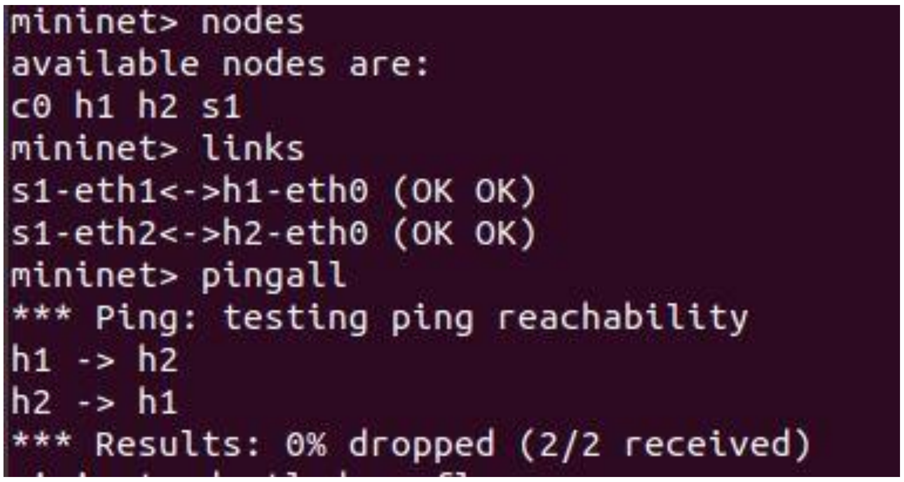


Figure 10: Simple network diagram

In Figures 9 and 10, it can be seen that a simple network has been created in Mininet by executing the command line "sudo mn –custome topo-1sw-2host.py –topo mytopo". Mininet adds hosts and a switch in the network and connects the hosts with the switch in eth1 and eth2 port. It uses the controller which supports OpenFlow protocol and already present in the Mininet. Mininet also configures the host with IP addresses and MAC addresses on their respective eth0 port. It also virtually connects the hosts with the switch. In short, Mininet simply creates a network topology, adds switch S1, adds hosts h1 and h2, adds links between h1 and s1, h2 and s1. After that, Mininet configures all hosts, launches the controller, and switches.



```
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> links
s1-eth1<->h1-eth0 (OK OK)
s1-eth2<->h2-eth0 (OK OK)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

Figure 11: Network details

Figure 11 shows the nodes in the network along with the links between them. It specifically shows the particular ports of the switch and host that are being connected to each other.

When the ping test among the devices had been carried out, it was successful. It proves that the controller is also working as expected and forwards the flow table in the switches. This mechanism allows the hosts to be reachable to one another. In an appendix (1), the output of "dpctl dump-flows" is attached. In the figure in the appendix (1), the output of a flow table view can be seen. In this view, the flow entries can be seen. These flow entries have been pushed by the controller to the switch. These entries ensure that the hosts are reachable to each other. It confirms the functionality of the controller.

The figure in appendix (2) shows the hosts h1-eth0 and h2-eth0 and loopback (lo) interfaces. IP address and netmask are assigned to all hosts.

Network Topology with 4 switches and 2 hosts + OpenFlow +Ryu

Now a simple topology with 4 switches and 2 hosts is made using Mininet, OpenFlow protocol, and Ryu controller. switches and hosts are connected in a ring topology.

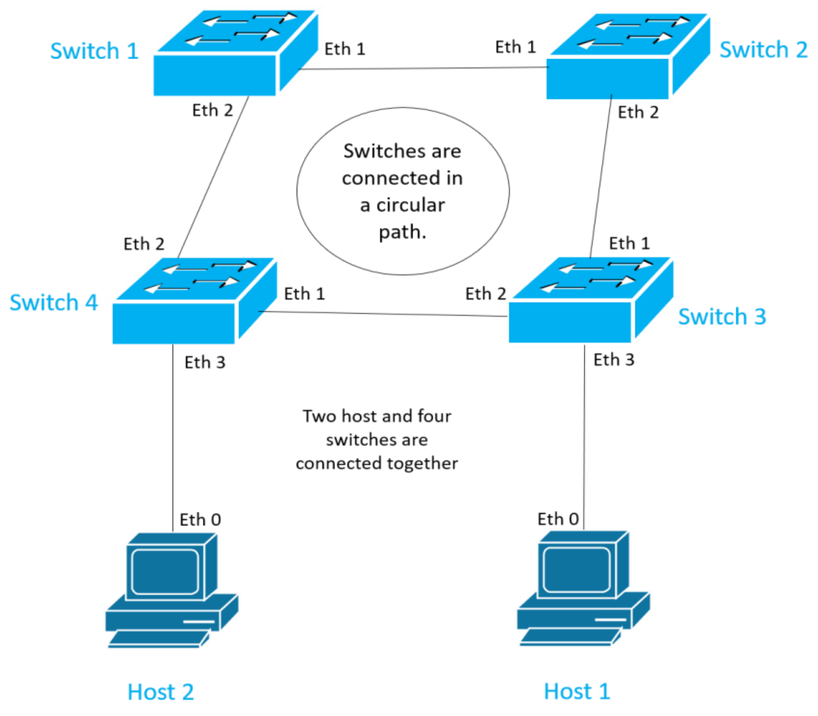
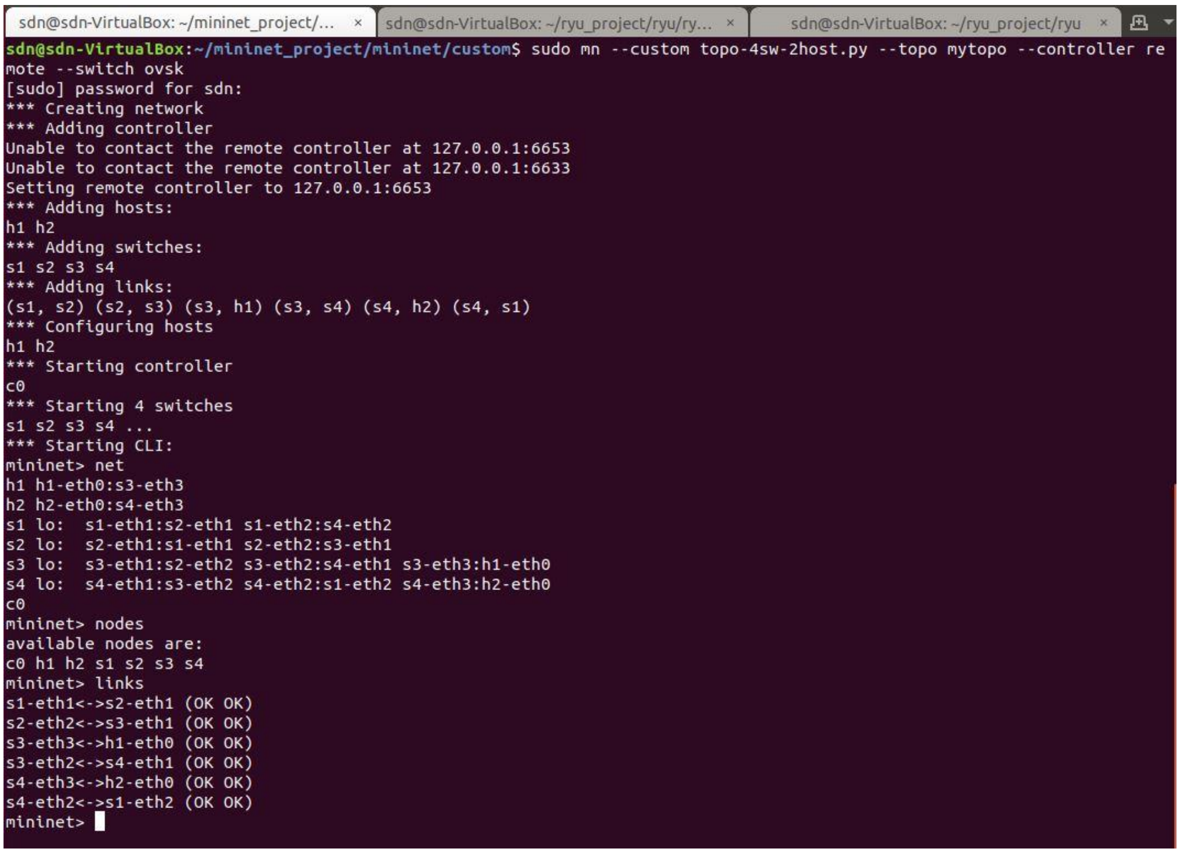


Figure 12: Network topology diagram


sdn@sdn-VirtualBox: ~/mininet_project/...
sdn@sdn-VirtualBox: ~/ryu_project/ryu/ry...
sdn@sdn-VirtualBox: ~/ryu_project/ryu

```
sdn@sdn-VirtualBox:~/mininet_project/mininet/custom$ sudo mn --custom topo-4sw-2host.py --topo mytopo --controller re
mote --switch ovsk
[sudo] password for sdn:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(s1, s2) (s2, s3) (s3, h1) (s3, s4) (s4, h2) (s4, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet> net
h1 h1-eth0:s3-eth3
h2 h2-eth0:s4-eth3
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s4-eth2
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1
s3 lo:  s3-eth1:s2-eth2 s3-eth2:s4-eth1 s3-eth3:h1-eth0
s4 lo:  s4-eth1:s3-eth2 s4-eth2:s1-eth2 s4-eth3:h2-eth0
c0
mininet> nodes
available nodes are:
c0 h1 h2 s1 s2 s3 s4
mininet> links
s1-eth1<->s2-eth1 (OK OK)
s2-eth2<->s3-eth1 (OK OK)
s3-eth3<->h1-eth0 (OK OK)
s3-eth2<->s4-eth1 (OK OK)
s4-eth3<->h2-eth0 (OK OK)
s4-eth2<->s1-eth2 (OK OK)
mininet> 
```

Figure 13: Ring network details

In figure 13, a simple network has been created in Mininet with SDN remote controller by executing the command line “sudo mn –customer topo-4sw-2host.py –topo mytopo –controller remote –switch ovsk”. Mininet adds 2 hosts and 4 switches in the network and connects the hosts with the switch in eth0, eth1, eth2, and eth3 port. It uses the extended Ryu controller which supports OpenFlow protocol and was added. Mininet also configures the host with IP addresses and MAC addresses on their respective eth0 eth1 eth2 and eth3 port. It also virtually connects the hosts with the switch. In short, Mininet simply creates a network topology, adds switch S1, adds hosts h1 and h2, adds links between h1 and s1, h2 and s1. After that, Mininet also configures all hosts, launches the controller as well as the 4 switches. It shows the nodes in the network along with the links between them. Mininet does not have a GUI interface. Ryu SDN Controllers topology viewer is used to show the network topologies.

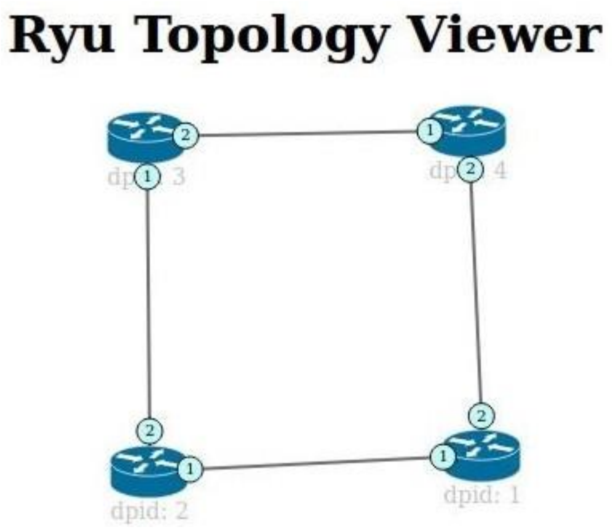


Figure 14: Ryu controller topology viewer

The performance of the Ryu controller is assessed by two scenarios. The first scenario does not use the Ryu Controller STP settings, and the second scenario uses STP settings. The Packet Internet Grouper (PING) test from host 2 with IP address 10.0.0.2/8 to host 1 with IP address 10.0.0.1/8 is used to verify the link on the network. PING runs echo request packets to the target host by sending the Internet Control Message Protocol (ICMP) message and wait for the ICMP response.

Without STP

Based on the first scenario test, figure 15 shows the result of the PING test where STP is not enabled in the network.

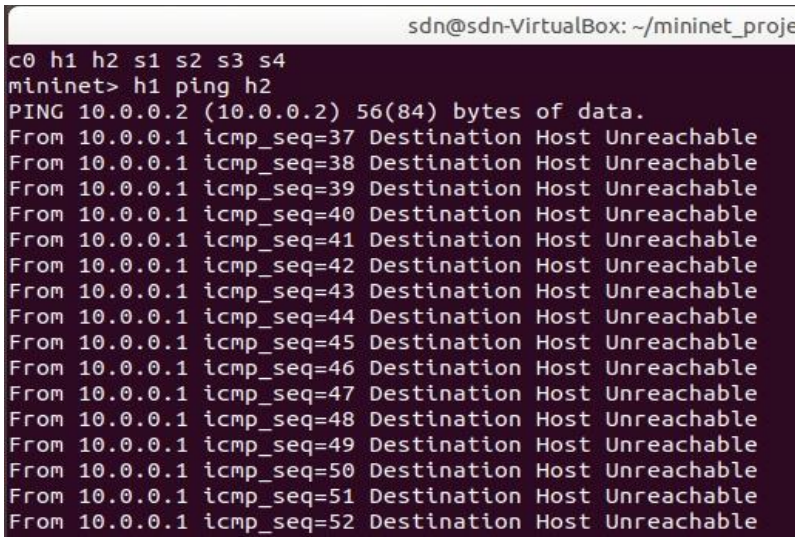
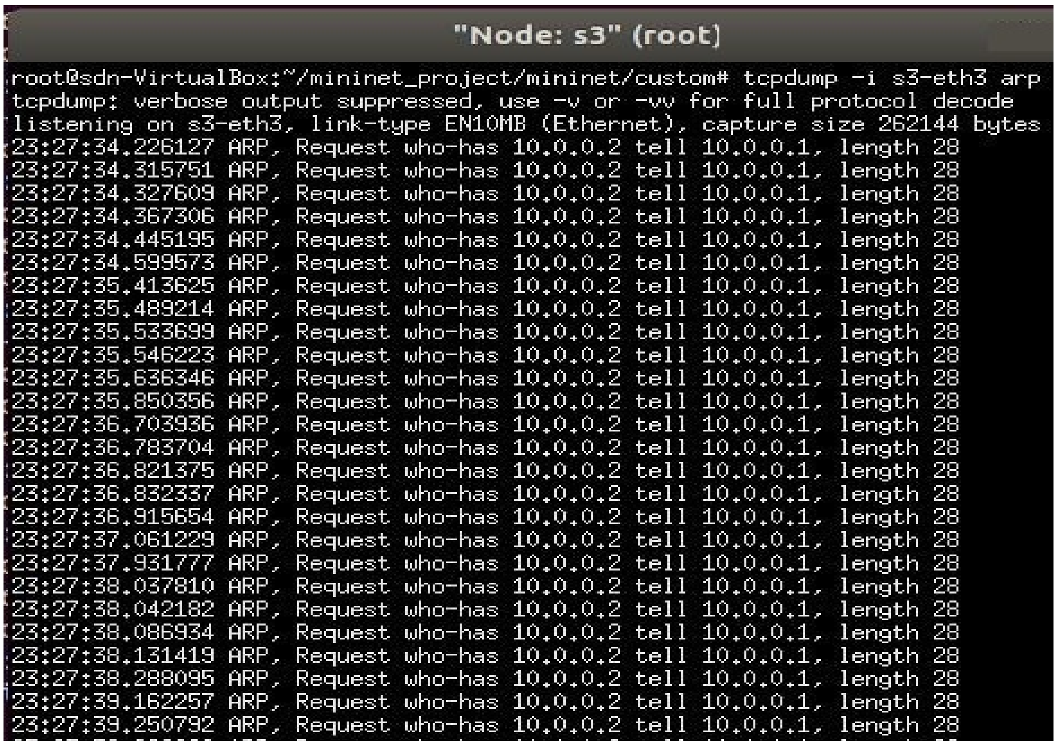

```
sdn@sdn-VirtualBox: ~/mininet_proje
c0 h1 h2 s1 s2 s3 s4
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=37 Destination Host Unreachable
From 10.0.0.1 icmp_seq=38 Destination Host Unreachable
From 10.0.0.1 icmp_seq=39 Destination Host Unreachable
From 10.0.0.1 icmp_seq=40 Destination Host Unreachable
From 10.0.0.1 icmp_seq=41 Destination Host Unreachable
From 10.0.0.1 icmp_seq=42 Destination Host Unreachable
From 10.0.0.1 icmp_seq=43 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44 Destination Host Unreachable
From 10.0.0.1 icmp_seq=45 Destination Host Unreachable
From 10.0.0.1 icmp_seq=46 Destination Host Unreachable
From 10.0.0.1 icmp_seq=47 Destination Host Unreachable
From 10.0.0.1 icmp_seq=48 Destination Host Unreachable
From 10.0.0.1 icmp_seq=49 Destination Host Unreachable
From 10.0.0.1 icmp_seq=50 Destination Host Unreachable
From 10.0.0.1 icmp_seq=51 Destination Host Unreachable
From 10.0.0.1 icmp_seq=52 Destination Host Unreachable
```

Figure 15: Ping test without STP in the network

The ICMP packet is being sent over to host 2 (10.0.0.2/8) from host 1 (10.0.0.1/8). As I can see in figure 15 that the packet is not being received by host 2 because the packet is traveling on the network in an endless loop. The result of the tcpdump of port eth3 in switch3 can be seen in figure 16. Host 1 is connected with this port. This tcpdump function sends arp requests to get the physical address of the corresponding destination IP address of host 2 (10.0.0.2/8). Without STP, the arp requests are getting dropped without reply. There are 4 switches and 2 network segments in this topology. Each segment H1-S3 and H2-S4 has 1 switch and 1 host. When host 1 transmits data to host 2, switch 3 first sends data to segment H2-S4 and it fails because there are no arp entries. S4 also sends this message to S1 and S2. Likewise, all switches keep sending and resending data. This creates a broadcast flooding in the network. In an appendix (3) the output of "dpctl dump-flows" can be seen. The command shows the output of the actual forwarding table flow entries. No port is blocked that can allow the loop to break.

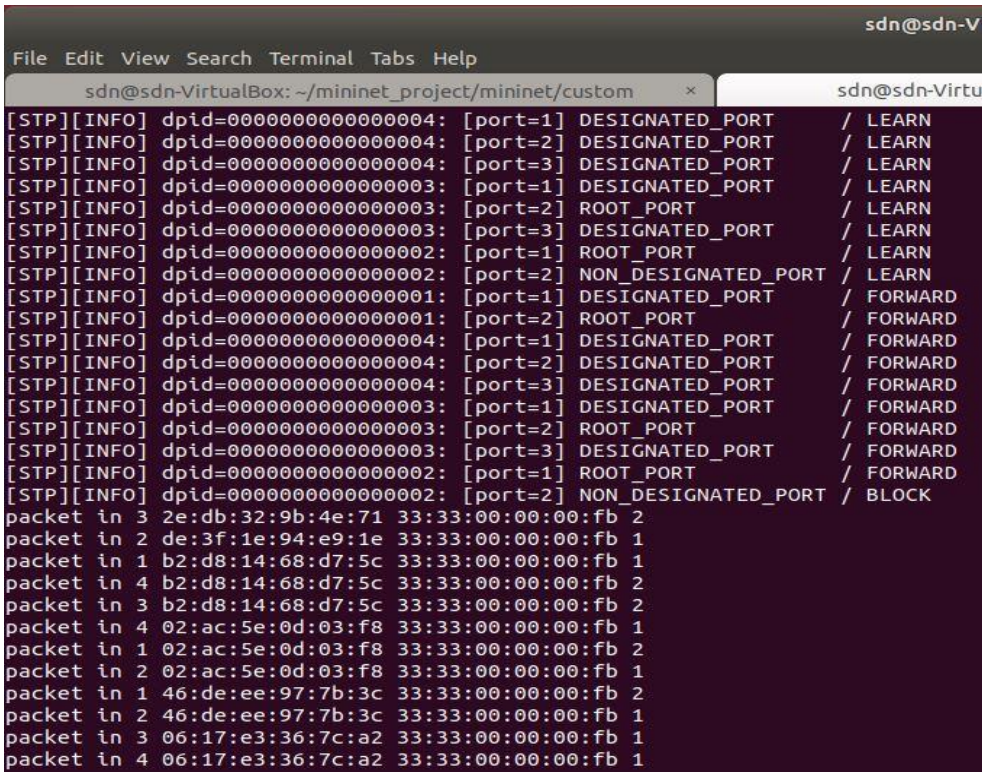


```
"Node: s3" (root)
root@sdn-VirtualBox:~/mininet_project/mininet/custom# tcpdump -i s3-eth3 arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s3-eth3, link-type EN10MB (Ethernet), capture size 262144 bytes
23:27:34.226127 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:34.315751 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:34.327609 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:34.367306 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:34.445195 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:34.599573 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:35.413625 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:35.489214 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:35.533699 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:35.546223 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:35.636346 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:35.850356 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:36.703936 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:36.783704 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:36.821375 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:36.832337 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:36.915654 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:37.061229 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:37.931777 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:38.037810 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:38.042182 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:38.086934 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:38.131419 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:38.288095 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:39.162257 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
23:27:39.250792 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
```

Figure 16: tcpdump on s3-eth3

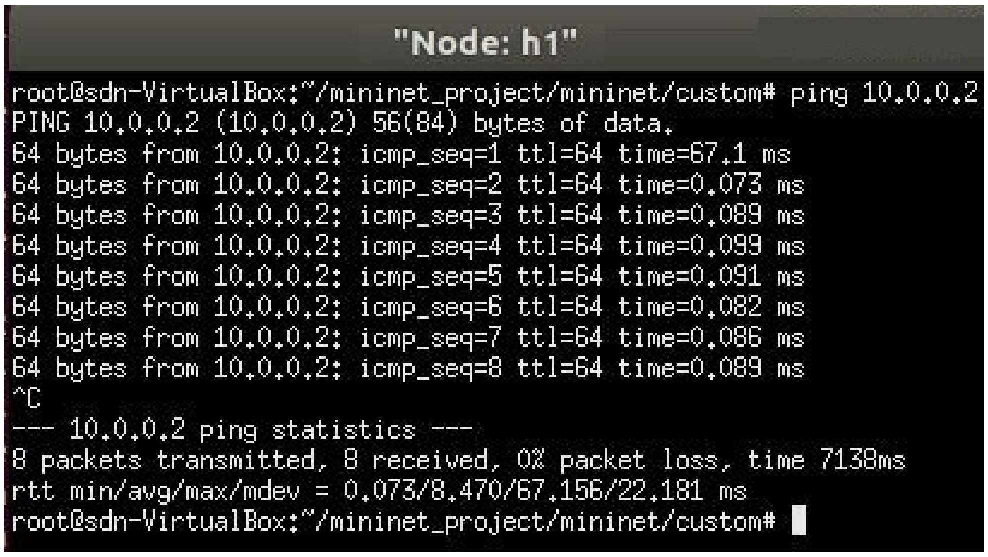
With STP

When the STP session initializes in the controller, the switch ports start from the BLOCK state. Later the port status changes to LISTEN to state and LEARN state. When the port stays in the BLOCK state it will not receive packets from the attached network segment and will discard packets. In LISTEN state the port gets ready to learn the network but still discards packets. When the port goes to LEARN state, it starts learning the MAC addresses by processing the incoming frames. It gradually updates the MAC address table. During the FORWARD state, the port works as expected. It keeps on receiving and forwarding packets. Different states of STP are shown in figure 17. This figure is captured from the Ryu controller console.


sdn@sdn-V
File Edit View Search Terminal Tabs Help
sdn@sdn-VirtualBox: ~/mininet_project/mininet/custom
sdn@sdn-Virtu

```
[STP][INFO] dpid=0000000000000004: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000004: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000004: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=2] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=1] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000002: [port=2] NON_DESIGNATED_PORT / LEARN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=2] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000004: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000004: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000004: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=2] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=1] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000002: [port=2] NON_DESIGNATED_PORT / BLOCK
packet in 3 2e:db:32:9b:4e:71 33:33:00:00:00:fb 2
packet in 2 de:3f:1e:94:e9:1e 33:33:00:00:00:fb 1
packet in 1 b2:d8:14:68:d7:5c 33:33:00:00:00:fb 1
packet in 4 b2:d8:14:68:d7:5c 33:33:00:00:00:fb 2
packet in 3 b2:d8:14:68:d7:5c 33:33:00:00:00:fb 2
packet in 4 02:ac:5e:0d:03:f8 33:33:00:00:00:fb 1
packet in 1 02:ac:5e:0d:03:f8 33:33:00:00:00:fb 2
packet in 2 02:ac:5e:0d:03:f8 33:33:00:00:00:fb 1
packet in 1 46:de:ee:97:7b:3c 33:33:00:00:00:fb 2
packet in 2 46:de:ee:97:7b:3c 33:33:00:00:00:fb 1
packet in 3 06:17:e3:36:7c:a2 33:33:00:00:00:fb 1
packet in 4 06:17:e3:36:7c:a2 33:33:00:00:00:fb 1
```

Figure 17: STP states in RYU controller



```
root@sdn-VirtualBox:~/mininet_project/mininet/custom# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=67.1 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.099 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.086 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.089 ms
^C
--- 10.0.0.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7138ms
rtt min/avg/max/mdev = 0.073/8.470/67.156/22.181 ms
root@sdn-VirtualBox:~/mininet_project/mininet/custom# 
```

Figure 18: Ping test

When the ping test among the hosts had been carried out, it was successful. It proves that the controller is also working as expected and forwards the flow table in the switches. This mechanism allows the hosts to be reachable to one another.
In an appendix (5), the output of the flow table can be seen. In this figure, we can see the highlighted flow entry where the action for the incoming packets on port “s3-eth1” has been set as “drop”. This flow entry enables the ring to work without the broadcast storm.

Topology with 4 switches and 2 hosts + OpenFlow + Ryu with connection failure between two nodes

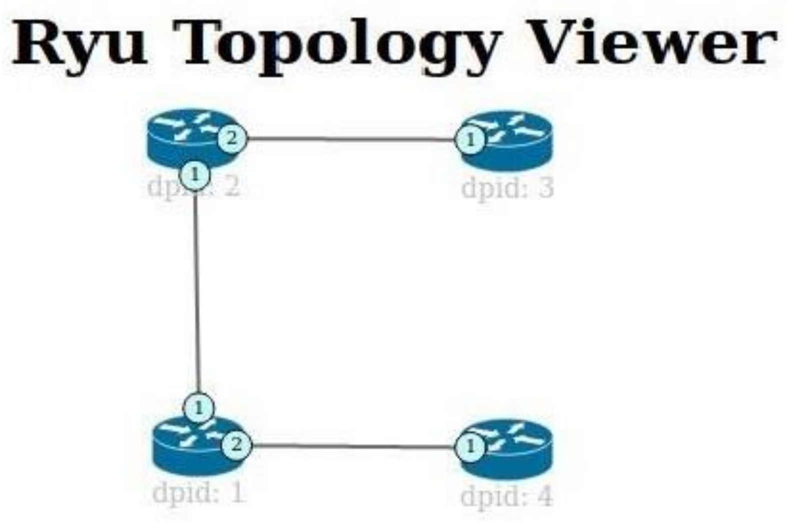


Figure 19: Topology view when s3 and s4 is not connected

Figure 19 shows the network topology connection between s3 and s4 is not established. The flow table, in the appendix (4) shows the updated values that have been pushed to the switches after the controller has adapted to the changes in the network automatically.

Topology with 9 switches and 2 hosts + OpenFlow + Ryu + STP - Link failover test with STP

In this test, a link failover scenario has been observed in a switching network. I have used a topology with 9 switches and 2 hosts. A switching template called “simple_switch_stp_13” was used from the RYU controller application library. This SDN application has the ability to run switching on the topology. Having STP enabled in the application ensures loop breaking and failover as well. I have created the topology such as one host is connected to eth-4 port of switch 1 and another host is connected to eth-4 port of switch 9.

There exist 3 paths for a packet traveling in this network. Path 1 is via switch 2 and switch 3. The second path is switch 7 and switch 8. The third path has switch 4,5 and 6. Since there is no shortest path algorithm running in the controller, it has randomly chosen path 3 to transfer packets.

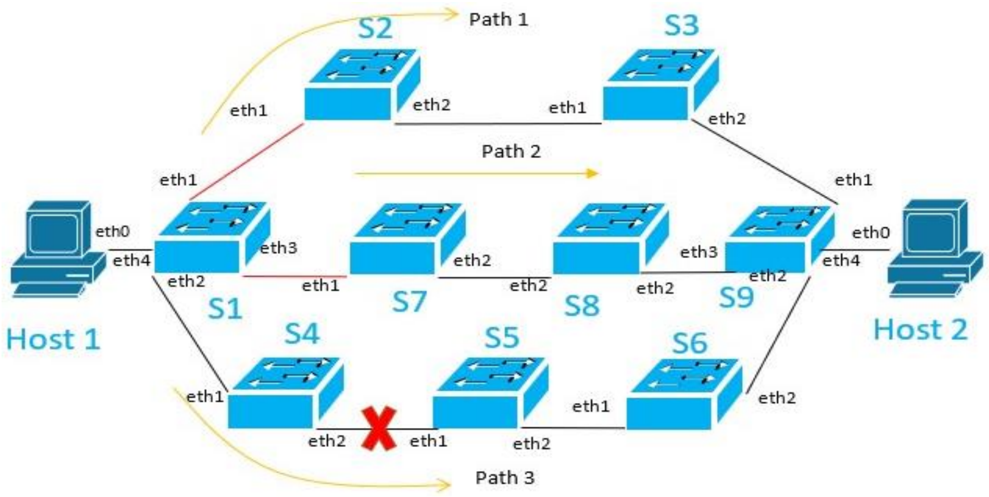
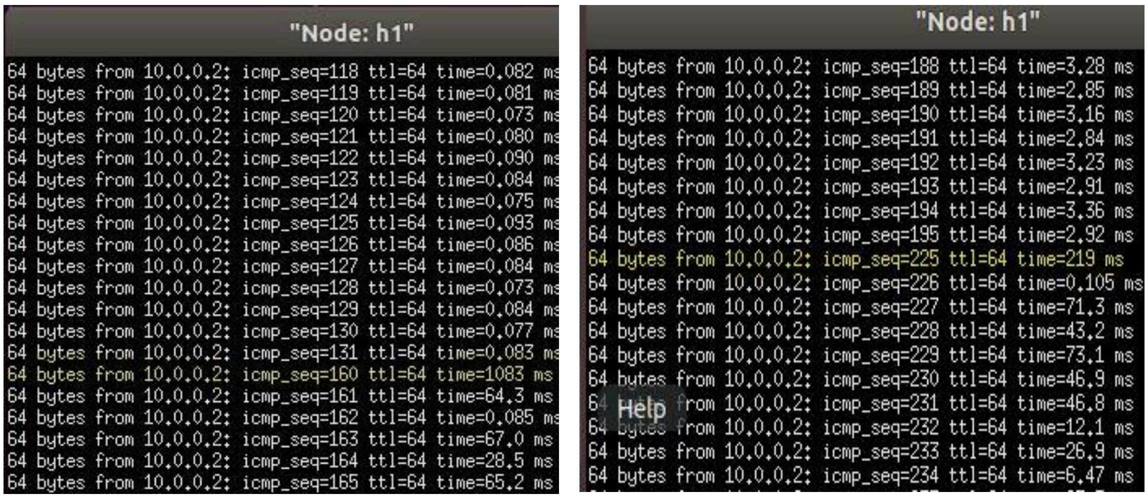


Figure 20: Fail-over test

In figure 20, it can be seen that STP has disabled eth3 and eth1 ports in Switch 1 to break the loop in the topology. Since the controller has chosen path 3 for the traffic flow, I have disconnected the link between switch 4 and 5 to simulate a failover test. After the link has been disconnected, the controller has initiated the calculation to choose the alternate path and after a certain delay, the traffic was established again. A detailed discussion on the result follows in the next chapter. The updated flow tables are also attached in appendix (5).



```
64 bytes from 10.0.0.2: icmp_seq=118 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=119 ttl=64 time=0.081 ms
64 bytes from 10.0.0.2: icmp_seq=120 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=121 ttl=64 time=0.080 ms
64 bytes from 10.0.0.2: icmp_seq=122 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=123 ttl=64 time=0.084 ms
64 bytes from 10.0.0.2: icmp_seq=124 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=125 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=126 ttl=64 time=0.086 ms
64 bytes from 10.0.0.2: icmp_seq=127 ttl=64 time=0.084 ms
64 bytes from 10.0.0.2: icmp_seq=128 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=129 ttl=64 time=0.084 ms
64 bytes from 10.0.0.2: icmp_seq=130 ttl=64 time=0.077 ms
64 bytes from 10.0.0.2: icmp_seq=131 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=160 ttl=64 time=1083 ms
64 bytes from 10.0.0.2: icmp_seq=161 ttl=64 time=64.3 ms
64 bytes from 10.0.0.2: icmp_seq=162 ttl=64 time=0.085 ms
64 bytes from 10.0.0.2: icmp_seq=163 ttl=64 time=67.0 ms
64 bytes from 10.0.0.2: icmp_seq=164 ttl=64 time=28.5 ms
64 bytes from 10.0.0.2: icmp_seq=165 ttl=64 time=65.2 ms
```

```
64 bytes from 10.0.0.2: icmp_seq=188 ttl=64 time=3.28 ms
64 bytes from 10.0.0.2: icmp_seq=189 ttl=64 time=2.85 ms
64 bytes from 10.0.0.2: icmp_seq=190 ttl=64 time=3.16 ms
64 bytes from 10.0.0.2: icmp_seq=191 ttl=64 time=2.84 ms
64 bytes from 10.0.0.2: icmp_seq=192 ttl=64 time=3.23 ms
64 bytes from 10.0.0.2: icmp_seq=193 ttl=64 time=2.91 ms
64 bytes from 10.0.0.2: icmp_seq=194 ttl=64 time=3.36 ms
64 bytes from 10.0.0.2: icmp_seq=195 ttl=64 time=2.92 ms
64 bytes from 10.0.0.2: icmp_seq=225 ttl=64 time=219 ms
64 bytes from 10.0.0.2: icmp_seq=226 ttl=64 time=0.105 ms
64 bytes from 10.0.0.2: icmp_seq=227 ttl=64 time=71.3 ms
64 bytes from 10.0.0.2: icmp_seq=228 ttl=64 time=43.2 ms
64 bytes from 10.0.0.2: icmp_seq=229 ttl=64 time=73.1 ms
64 bytes from 10.0.0.2: icmp_seq=230 ttl=64 time=46.9 ms
6[illegible] from 10.0.0.2: icmp_seq=231 ttl=64 time=46.8 ms
64 bytes from 10.0.0.2: icmp_seq=232 ttl=64 time=12.1 ms
64 bytes from 10.0.0.2: icmp_seq=233 ttl=64 time=26.9 ms
64 bytes from 10.0.0.2: icmp_seq=234 ttl=64 time=6.47 ms
```

Figure 21: Switching time in two tests

5. Result and Discussion

In this chapter, the results of the tests will be discussed and that will lead us to the overall result of this thesis work.

Test 1

At first, I have simulated SDN with a simple topology. A switch is connected with two hosts and the Mininet in-built controller is pushing the flows on the switch to make hosts reachable to each other.

Result: Host 1 can ping Host 2.

Discussion: This test shows the centralized operation principle of SDN.

Test 2

This test was performed with a ring topology. First, the network is not having STP running in it. And then STP in the ring is enabled.

Result: When STP is not running in the network the hosts are not able to ping each other and with STP they are reachable.

Discussion: This test shows that the traditional networking protocols are also valid for SDN and these protocols can be programmed to be pushed from the centralized controller to the network elements.

Test 3

In this test, I have shown that SDN can adapt to the network changes and take intelligent decisions by itself to ensure network reliability. After the link between two switches is being disconnected hence the primary path between the hosts is not available but the controller can update its flow table accordingly and the hosts are reachable to each other.

Result: Host 1 can ping Host 2.

Discussion: This test shows the main advantage of the SDN over the traditional network. By the result of this test, it can be seen that the SDN controller has detected the fault itself and taken the necessary step by itself (updating the flow table) to ensure network reliability.

Test 4

In this test, the network failover scenario in a Layer 2 switching network has been observed. The traffic flow was established again through the second link after a certain delay. The network takes some time to converge to the alternate path after the primary path is disconnected.

Result: The ping from host 1 to host 2 is resumed after a certain delay.

Discussion: In the traditional network with L2 packet switching, we know STP can take up to 50s (IEEE 802.1Q-2014) to switch the network traffic to the alternate path. The restoration time causes massive packet loss in the network. The PING test between two hosts is already added to the implementation part. In the first attempt, the controller took 1083ms to restore the traffic in an alternate path. While on the second attempt, the network shows a much-improved result with only 219ms to converge.

Moreover, this also points out the fact that the network convergence in the legacy network also depends on the network size. The layer 3 routing protocol like BGP and OSPF needs to update their neighbours with link failure information. This information propagated through the whole network and eventually, it takes more time to cover all the elements in the network when the network size is large. The larger the network the more time it takes to converge. But in SDN, there is less dependency on the network size as the network fail event is reported by the switch to the controller, and the controller updates the flow table alone with the pre-existing knowledge of the network and pushed these updates to the switches. This process is much faster than the old procedure.

Summary

The test results can be summarized in below table which will help us to understand how these results lead us to achieve the main objective of this thesis.

Table 3: Improvements on SDN

Test	Improvements on SDN over traditional network		
	Legacy network	**SDN**	**Comments**
1	De-centralized operation	Centralized operation	This feature lets SDN open to programmable network infrastructure.
2	Networks protocols run in every network element and are often vendor depended and need interoperability.	Network protocols are programmable and are controlled from a centralized location hence easier operation.	Network services can be created and run on the SDN controller as a network application and have no vendor dependency and fully compatible.
3	Redundancy needs to be specified by allocating resources beforehand with a revert mechanism.	Fault detection and restoration are fully automated. SDN controller can works intelligently on fault occasions.	A fully protected network can be programmed in SDN that will rely on intelligent and smart fault detection.
4	Network convergence time is more in all legacy switching and routing protocols.	SDN provides faster network convergence with a centralized point for updating the network change push to the forwarding devices.	While in the legacy system, network convergence depends on updating all the network elements using failure information propagation, in SDN the failure information gets updated in the controller only. The controller chooses the alternate path updated all the switches which makes SDN much faster.

Although these improvements can be seen from the test results, the same improvements have been seen in the literature review as well.

1. With SDN, network provisioning becomes easier and simple. A network operator can provision network elements from a centralized location without having physical access to the edge devices. This way a network administrator can save valuable time and money while building a network infrastructure.
2. Since the edge devices are not intelligent and cheap users can save money on traditional costly network equipment.
3. As the network controller is fully programmable, it can be hosted anywhere from a physical server to a virtual environment. This gives more freedom to the small network to grow on demand.
4. In case of a link failure scenario, SDN can converge much faster than legacy routing and switching protocols.

The above-mentioned advantages and the result presented from the test show that this thesis has achieved the objectives as promised in the beginning.

6. Conclusion

The main aim of this thesis is to show the improvements of the SDN network over a traditional networking system. This paper describes software-defined networking on a detailed technological basis (mechanism and functionalities) as well as the differences between the SDN and traditional network. The fundamental aspects of SDN architecture and OpenFlow protocol have been discussed. It has been shown that with the network change, the SDN can learn the changes by itself and reflect the changes in the flow table hence maintaining the integrity and quality of the network. The controller is fully programmable, and the network operators can program the actions on the flow tables on basis of their network requirements.

Virtualization makes do more with less a reality as resources such as compute, storage, and networking are right sized to the application workloads. On the other hand, it is very difficult to determine whether the invention will have a certain market impact (e.g. on increasing efficiency or reducing costs). The relentless growth in users and expectations nevertheless means that the operators have to reconsider the use of existing network technology to remain competitive and profitable. As the literature review shows, the distinction between the control plane and the data plane provides great advantages, such as ease of management, improved features, complex implementation, and economic conditions of virtual networks. SDN also ensured traffic recovery within a very short amount of time compares to the traditional network which requires full resource allocation for the same.

A small-scale network has been implemented to demonstrate the idea of SDN in the latter half of this thesis work. It has been shown that with the network change, the controller can adapt to that and updates its flow table automatically and broadcast to the network switches. A network failover test has been carried out and SDN shows much-improved results over the traditional network. It can switch to the alternative path with the help of STP within a short period of time.

7. References

1. ROUSE, M. *Search Networking tech target: Tech Target*. 10. October 2006. https://searchnetworking.techtarget.com/definition/networking. (Accessed date: 06. November 2020.).

2. MOY, J. *OSPF Version 2*. RFC 2328. ietf.org. [Online] April 1998. https://tools.ietf.org/html/rfc2178 (Accessed date: 06. November 2020).

3. HEDRIK, C. *Routing Information Protocol. Network Working Group*. Rutgers University, June 1988. https://tools.ietf.org/html/rfc1058 (Accessed date: 06. November 2020).

4. KATTA, Naga Praveen Kumar. *Building efficient and reliable software-defined networks* (Dissertation). Princeton university. Princeton: Princeton University, 2016. https://www.cs.princeton.edu/~jrex/thesis/naga-katta-thesis.pdf (Accessed date: 06. November 2020).

5. MIJUMBI, R., Serrat, J., Gorricho, J., Bouten, N., Turck, F, D. and Boutaba, R. *Network Function Virtualization: State-of-the-art and Research Challenges*. 2015. DOI: 10.1109/COMST.2015.2477041

6. SARKA, T. K. *SDN testbed-based evolution of flow processing-aware controller placement*. Master's Thesis. Supervisor: Prof. Dr. Holger Karl University of Paderborn, 2017.

7. TAHA, A. *Software-Defined Networking and its Security. Master's Thesis*. Aalto: Aalto University, School of Electrical Engineering, 2014.

8. FEAMSTER, N., Rexford, J. and Zegura, E. *The Road to SDN: An Intellectual History of Programmable Networks*. ACM SIGCOMM Computer Communication Review, 2014. DOI: https://doi.org/10.1145/2602204.2602219.

9. Open Networking Foundation. SDN definition. [Online] https://opennetworking.org/sdn-definition/. (Accessed date: 06. November 2020).

10. SHIN, M., Nam, K., and Kim, H. *Software-defined networking (SDN): A reference architecture and open APIs,* 2012. DOI: 10.1109/ICTC.2012.6386859.

11. PFAFF, B. Davie, Ed. VMware. *The Open vSwitch Database Management Protocol. IETF*. Available from: https://tools.ietf.org/html/rfc7047 ISSN: 2070-1721

12. BRAUN, W. and Menth, M. *Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices*, 2014. DOI:10.3390/fi60203022014.

13. CISAR, P**.,** Erlenvajn, D. and Čisar, M, S. *Implementation of Software-Defined Networks Using Open-Source Environment*. 2018, ISSN 1848-6339. DOI: https://doi.org/10.17559/TV-20160928094756

14. REKHA, P. M. and Dakshayini, M. *A Study of Software Defined Networking with OpenFlow*, 2015. DOI: 10.5120/21694-4798

15. ETSI (European Telecommunications Standards Institute), *Group Specification: Network Functions Virtualisation (NFV)* [Online] V1.2.1.; Architectural Framework. 2014. Available from: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf. (Accessed date: 06. November 2020).

16. GOPI, D., Cheng. S. and Huck, R. *Comparative analysis of SDN and conventional networks using routing protocols* [Online]. International Conference on Computer, Information and Telecommunication Systems (CITS) July 2017, Available from: https://www.researchgate.net/publication/319868083_Comparative_analysis_of_SDN_and_conventional_networks_using_routing_protocols. (Accessed date: 28. November 2020). DOI: 10.1109/CITS.2017.8035305

17. ZHANG, H. and Yan, J. *Performance of SDN Routing in Comparison with Legacy Routing Protocols*, [Online]. International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2015. Available from: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7307865. (Accessed date: 06. November 2020).

18. ALJUNAID, H. A. M., Warip, M. B. N. M., Ahmad, R. B., Anuar, S.M., Ibrahim, Z., Khairunizam, W., Razlan, M. Z. and Bakar, A. S. *Software Defined Networks Security: Link Failure Analysis in SDN* [Online]. IOP Conference Series: Materials Science and Engineering, 2019. Available from:

https://www.researchgate.net/publication/334097438_Software_Defined_Networks_Security_Link_Failure_Analysis_in_SDN. (Accessed date: 28. November 2020). DOI: 10.1088/1757-899X/557/1/012039.

19. OLADUNJOYE, O. *Software-defined networking-The emerging paradigm to computer networking*, 2017. Available from: https://core.ac.uk/download/pdf/84793161.pdf. (Accessed date: 28. November 2020).

20. KREUTZ, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky S. and Uhlig, S. *Software-Defined Networking: A Comprehensive Survey*, [Online]. In Proceedings of the IEEE, Jan 2015.Available from: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6994333. (Accessed date: 06. November 2020). DOI: 10.1109/JPROC.2014.2371999.

21. NADEAU, D, T. and Gray, K. *SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies [Online]*. Sebastopol: O'Reilly Media, Inc.2013. Available from: https://books.google.cz/books/about/SDN_Software_Defined_Networks.html?id=Bc1qAAAAQBAJ&printsec=frontcover&source=kp_read_button&redir_esc=y#v=onepage&q&f=false. (Accessed date: 06. November 2020). ISBN: 9781449342302

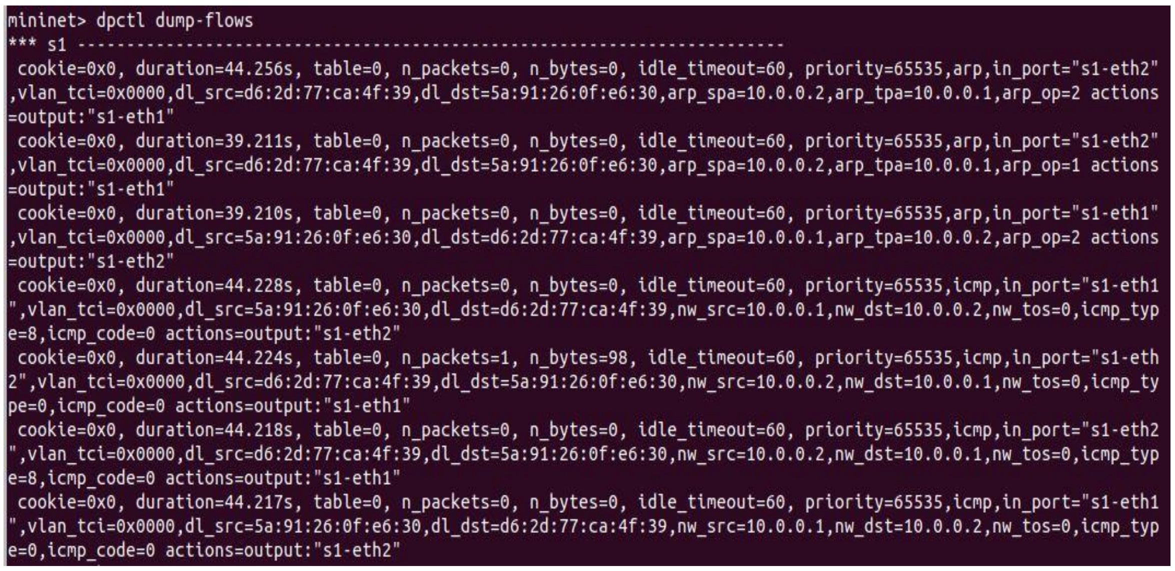
22. SÁNCHEZ, L. F. *SDN-sniff: monitor multi-tenant traffic in virtualized network infrastructure*, 2018. Available from: https://upcommons.upc.edu/bitstream/handle/2117/117982/Bachelor_Thesis_Luis_Fern%C3%A1ndez.pdf. (Accessed date: 06. November 2020).

23. LARA, A. Using Software-Defined Networking to Improve Campus, Transport and Future Internet Architectures (Dissertation). Supervision of Professor Byrav Ramamurthy. Lincoln, Nebraska: December 2015. Available from: https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1109&context=computerscidiss. (Accessed date: 06. November 2020).

24. KARAKUS, M. and Durresi, Arjan. Economic Viability of Software Defined Networking (SDN),2018. DOI: https://doi.org/10.1016/j.comnet.2018.02.015. (Accessed date: 06. November 2020).

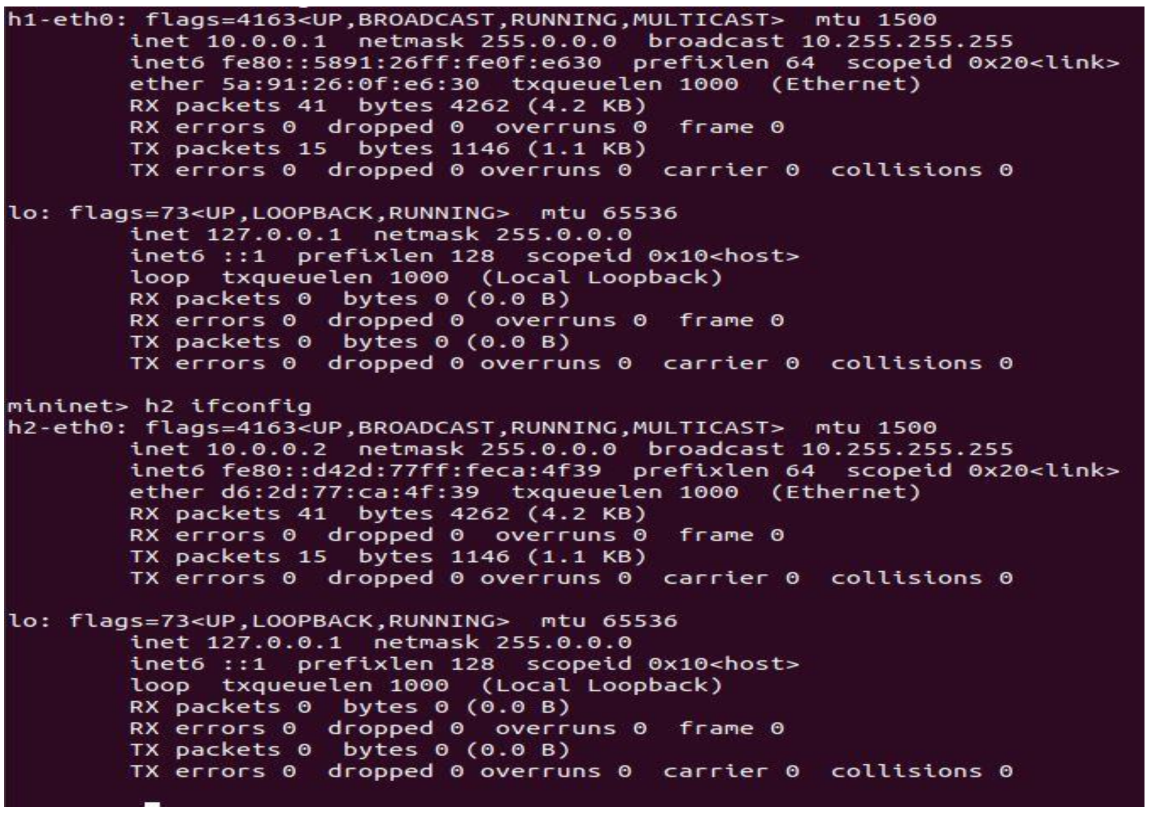
8. Appendix

(1) Output of flow table view



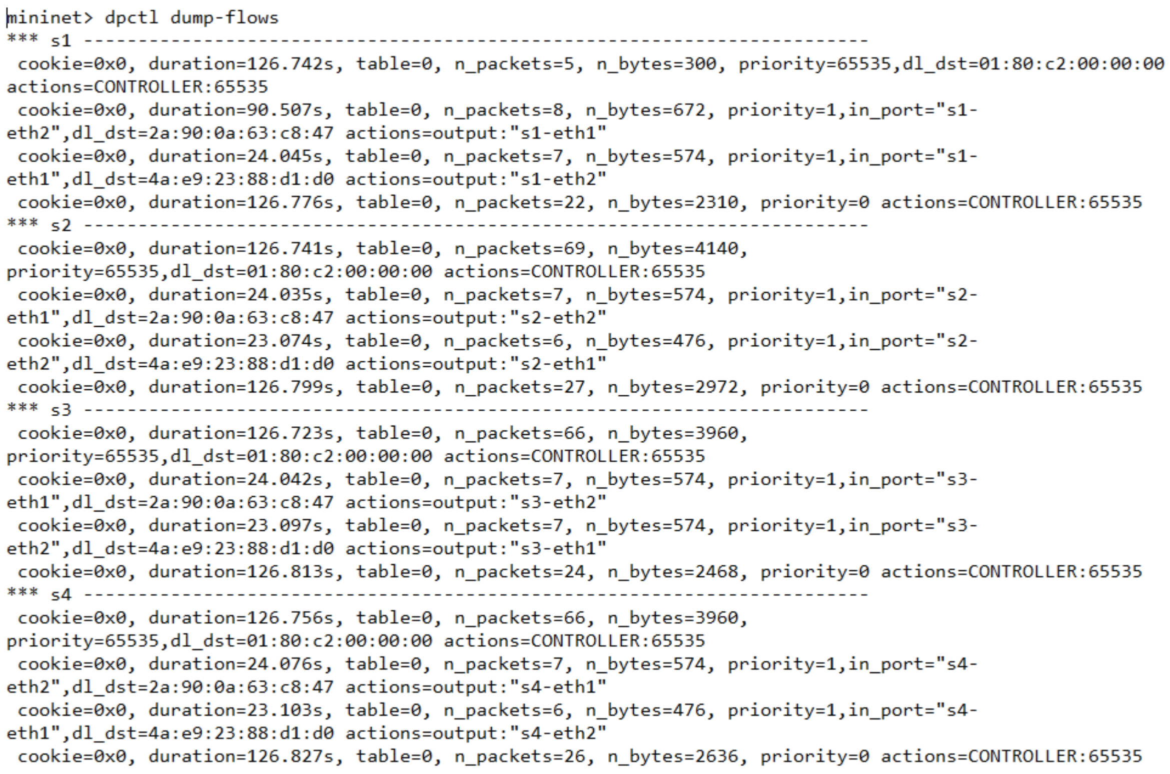
```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=44.256s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535,arp,in_port="s1-eth2"
,vlan_tci=0x0000,dl_src=d6:2d:77:ca:4f:39,dl_dst=5a:91:26:0f:e6:30,arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=2 actions
=output:"s1-eth1"
 cookie=0x0, duration=39.211s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535,arp,in_port="s1-eth2"
,vlan_tci=0x0000,dl_src=d6:2d:77:ca:4f:39,dl_dst=5a:91:26:0f:e6:30,arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=1 actions
=output:"s1-eth1"
 cookie=0x0, duration=39.210s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535,arp,in_port="s1-eth1"
,vlan_tci=0x0000,dl_src=5a:91:26:0f:e6:30,dl_dst=d6:2d:77:ca:4f:39,arp_spa=10.0.0.1,arp_tpa=10.0.0.2,arp_op=2 actions
=output:"s1-eth2"
 cookie=0x0, duration=44.228s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535,icmp,in_port="s1-eth1
",vlan_tci=0x0000,dl_src=5a:91:26:0f:e6:30,dl_dst=d6:2d:77:ca:4f:39,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_typ
e=8,icmp_code=0 actions=output:"s1-eth2"
 cookie=0x0, duration=44.224s, table=0, n_packets=1, n_bytes=98, idle_timeout=60, priority=65535,icmp,in_port="s1-eth
2",vlan_tci=0x0000,dl_src=d6:2d:77:ca:4f:39,dl_dst=5a:91:26:0f:e6:30,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_ty
pe=0,icmp_code=0 actions=output:"s1-eth1"
 cookie=0x0, duration=44.218s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535,icmp,in_port="s1-eth2
",vlan_tci=0x0000,dl_src=d6:2d:77:ca:4f:39,dl_dst=5a:91:26:0f:e6:30,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_typ
e=8,icmp_code=0 actions=output:"s1-eth1"
 cookie=0x0, duration=44.217s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535,icmp,in_port="s1-eth1
",vlan_tci=0x0000,dl_src=5a:91:26:0f:e6:30,dl_dst=d6:2d:77:ca:4f:39,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_typ
e=0,icmp_code=0 actions=output:"s1-eth2"
```

(2) Host details



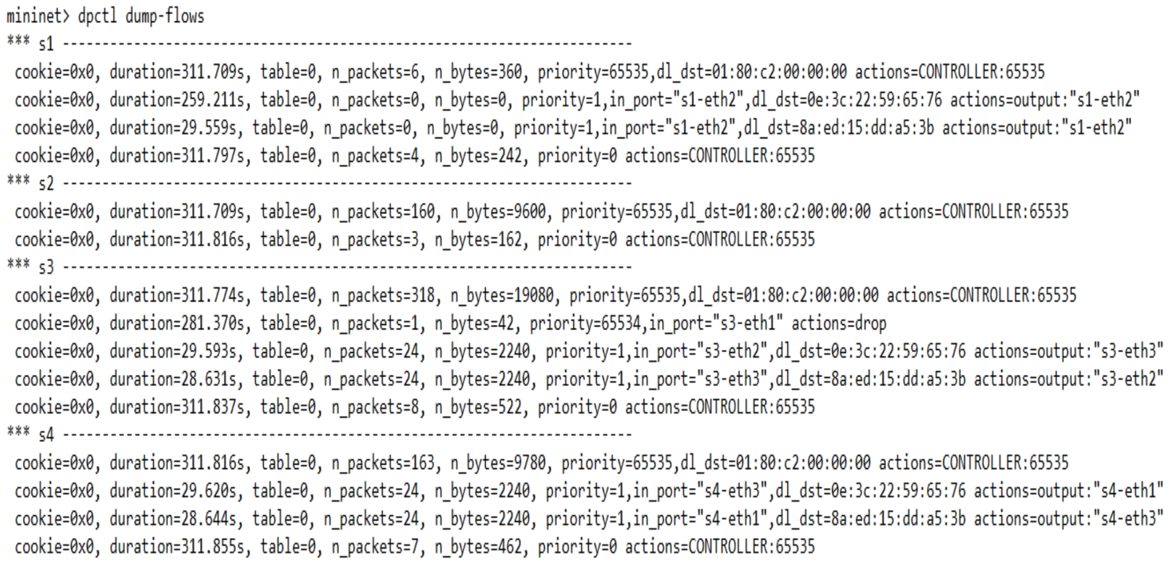
```
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::5891:26ff:fe0f:e630  prefixlen 64  scopeid 0x20<link>
        ether 5a:91:26:0f:e6:30  txqueuelen 1000  (Ethernet)
        RX packets 41  bytes 4262 (4.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 15  bytes 1146 (1.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet> h2 ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.2  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::d42d:77ff:feca:4f39  prefixlen 64  scopeid 0x20<link>
        ether d6:2d:77:ca:4f:39  txqueuelen 1000  (Ethernet)
        RX packets 41  bytes 4262 (4.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 15  bytes 1146 (1.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

(3) Flow table when STP is not enabled in the ring



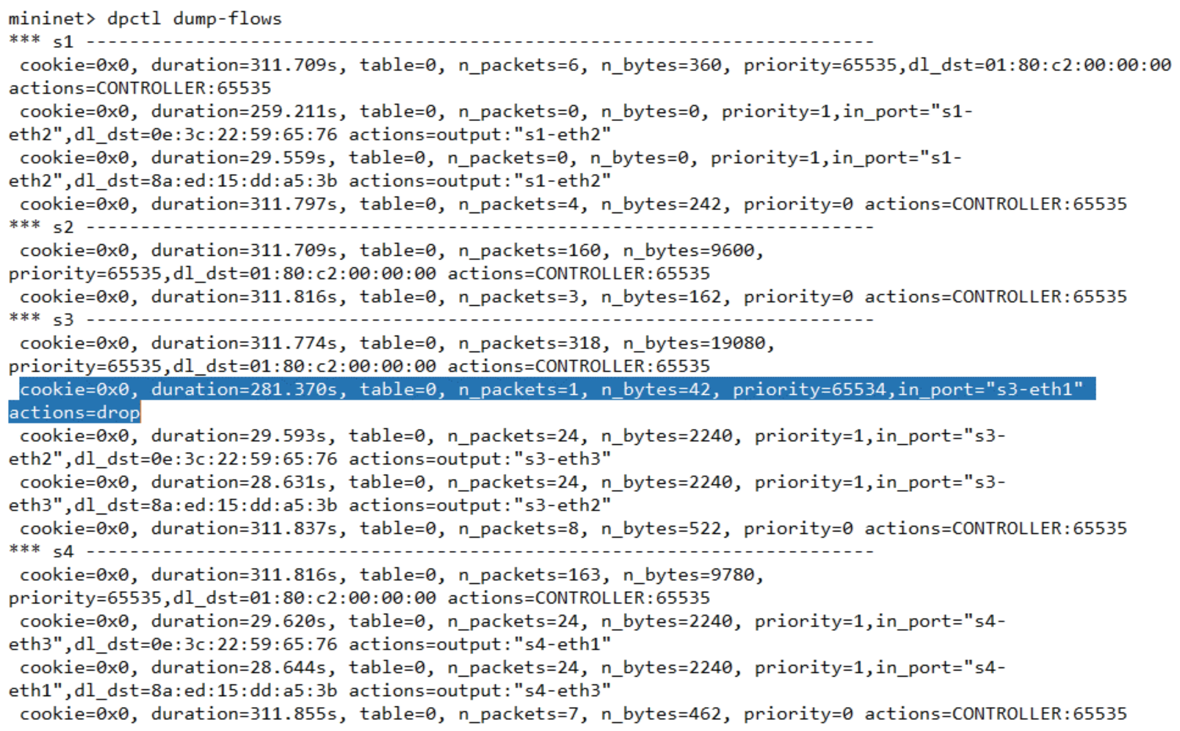
```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=126.742s, table=0, n_packets=5, n_bytes=300, priority=65535,dl_dst=01:80:c2:00:00:00
actions=CONTROLLER:65535
 cookie=0x0, duration=90.507s, table=0, n_packets=8, n_bytes=672, priority=1,in_port="s1-
eth2",dl_dst=2a:90:0a:63:c8:47 actions=output:"s1-eth1"
 cookie=0x0, duration=24.045s, table=0, n_packets=7, n_bytes=574, priority=1,in_port="s1-
eth1",dl_dst=4a:e9:23:88:d1:d0 actions=output:"s1-eth2"
 cookie=0x0, duration=126.776s, table=0, n_packets=22, n_bytes=2310, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=126.741s, table=0, n_packets=69, n_bytes=4140,
priority=65535,dl_dst=01:80:c2:00:00:00 actions=CONTROLLER:65535
 cookie=0x0, duration=24.035s, table=0, n_packets=7, n_bytes=574, priority=1,in_port="s2-
eth1",dl_dst=2a:90:0a:63:c8:47 actions=output:"s2-eth2"
 cookie=0x0, duration=23.074s, table=0, n_packets=6, n_bytes=476, priority=1,in_port="s2-
eth2",dl_dst=4a:e9:23:88:d1:d0 actions=output:"s2-eth1"
 cookie=0x0, duration=126.799s, table=0, n_packets=27, n_bytes=2972, priority=0 actions=CONTROLLER:65535
*** s3 ------------------------------------------------------------------------
 cookie=0x0, duration=126.723s, table=0, n_packets=66, n_bytes=3960,
priority=65535,dl_dst=01:80:c2:00:00:00 actions=CONTROLLER:65535
 cookie=0x0, duration=24.042s, table=0, n_packets=7, n_bytes=574, priority=1,in_port="s3-
eth1",dl_dst=2a:90:0a:63:c8:47 actions=output:"s3-eth2"
 cookie=0x0, duration=23.097s, table=0, n_packets=7, n_bytes=574, priority=1,in_port="s3-
eth2",dl_dst=4a:e9:23:88:d1:d0 actions=output:"s3-eth1"
 cookie=0x0, duration=126.813s, table=0, n_packets=24, n_bytes=2468, priority=0 actions=CONTROLLER:65535
*** s4 ------------------------------------------------------------------------
 cookie=0x0, duration=126.756s, table=0, n_packets=66, n_bytes=3960,
priority=65535,dl_dst=01:80:c2:00:00:00 actions=CONTROLLER:65535
 cookie=0x0, duration=24.076s, table=0, n_packets=7, n_bytes=574, priority=1,in_port="s4-
eth2",dl_dst=2a:90:0a:63:c8:47 actions=output:"s4-eth1"
 cookie=0x0, duration=23.103s, table=0, n_packets=6, n_bytes=476, priority=1,in_port="s4-
eth1",dl_dst=4a:e9:23:88:d1:d0 actions=output:"s4-eth2"
 cookie=0x0, duration=126.827s, table=0, n_packets=26, n_bytes=2636, priority=0 actions=CONTROLLER:65535
```

(4) Flow table after the network has learned the changes and update the table



```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=311.709s, table=0, n_packets=6, n_bytes=360, priority=65535,dl_dst=01:80:c2:00:00:00 actions=CONTROLLER:65535
 cookie=0x0, duration=259.211s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s1-eth2",dl_dst=0e:3c:22:59:65:76 actions=output:"s1-eth2"
 cookie=0x0, duration=29.559s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s1-eth2",dl_dst=8a:ed:15:dd:a5:3b actions=output:"s1-eth2"
 cookie=0x0, duration=311.797s, table=0, n_packets=4, n_bytes=242, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=311.709s, table=0, n_packets=160, n_bytes=9600, priority=65535,dl_dst=01:80:c2:00:00:00 actions=CONTROLLER:65535
 cookie=0x0, duration=311.816s, table=0, n_packets=3, n_bytes=162, priority=0 actions=CONTROLLER:65535
*** s3 ------------------------------------------------------------------------
 cookie=0x0, duration=311.774s, table=0, n_packets=318, n_bytes=19080, priority=65535,dl_dst=01:80:c2:00:00:00 actions=CONTROLLER:65535
 cookie=0x0, duration=281.370s, table=0, n_packets=1, n_bytes=42, priority=65534,in_port="s3-eth1" actions=drop
 cookie=0x0, duration=29.593s, table=0, n_packets=24, n_bytes=2240, priority=1,in_port="s3-eth2",dl_dst=0e:3c:22:59:65:76 actions=output:"s3-eth3"
 cookie=0x0, duration=28.631s, table=0, n_packets=24, n_bytes=2240, priority=1,in_port="s3-eth3",dl_dst=8a:ed:15:dd:a5:3b actions=output:"s3-eth2"
 cookie=0x0, duration=311.837s, table=0, n_packets=8, n_bytes=522, priority=0 actions=CONTROLLER:65535
*** s4 ------------------------------------------------------------------------
 cookie=0x0, duration=311.816s, table=0, n_packets=163, n_bytes=9780, priority=65535,dl_dst=01:80:c2:00:00:00 actions=CONTROLLER:65535
 cookie=0x0, duration=29.620s, table=0, n_packets=24, n_bytes=2240, priority=1,in_port="s4-eth3",dl_dst=0e:3c:22:59:65:76 actions=output:"s4-eth1"
 cookie=0x0, duration=28.644s, table=0, n_packets=24, n_bytes=2240, priority=1,in_port="s4-eth1",dl_dst=8a:ed:15:dd:a5:3b actions=output:"s4-eth3"
 cookie=0x0, duration=311.855s, table=0, n_packets=7, n_bytes=462, priority=0 actions=CONTROLLER:65535
```

(5) Flow table when STP is enabled in the ring



```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=311.709s, table=0, n_packets=6, n_bytes=360, priority=65535,dl_dst=01:80:c2:00:00:00
actions=CONTROLLER:65535
 cookie=0x0, duration=259.211s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s1-
eth2",dl_dst=0e:3c:22:59:65:76 actions=output:"s1-eth2"
 cookie=0x0, duration=29.559s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s1-
eth2",dl_dst=8a:ed:15:dd:a5:3b actions=output:"s1-eth2"
 cookie=0x0, duration=311.797s, table=0, n_packets=4, n_bytes=242, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=311.709s, table=0, n_packets=160, n_bytes=9600,
priority=65535,dl_dst=01:80:c2:00:00:00 actions=CONTROLLER:65535
 cookie=0x0, duration=311.816s, table=0, n_packets=3, n_bytes=162, priority=0 actions=CONTROLLER:65535
*** s3 ------------------------------------------------------------------------
 cookie=0x0, duration=311.774s, table=0, n_packets=318, n_bytes=19080,
priority=65535,dl_dst=01:80:c2:00:00:00 actions=CONTROLLER:65535
 cookie=0x0, duration=281.370s, table=0, n_packets=1, n_bytes=42, priority=65534,in_port="s3-eth1"
actions=drop
 cookie=0x0, duration=29.593s, table=0, n_packets=24, n_bytes=2240, priority=1,in_port="s3-
eth2",dl_dst=0e:3c:22:59:65:76 actions=output:"s3-eth3"
 cookie=0x0, duration=28.631s, table=0, n_packets=24, n_bytes=2240, priority=1,in_port="s3-
eth3",dl_dst=8a:ed:15:dd:a5:3b actions=output:"s3-eth2"
 cookie=0x0, duration=311.837s, table=0, n_packets=8, n_bytes=522, priority=0 actions=CONTROLLER:65535
*** s4 ------------------------------------------------------------------------
 cookie=0x0, duration=311.816s, table=0, n_packets=163, n_bytes=9780,
priority=65535,dl_dst=01:80:c2:00:00:00 actions=CONTROLLER:65535
 cookie=0x0, duration=29.620s, table=0, n_packets=24, n_bytes=2240, priority=1,in_port="s4-
eth3",dl_dst=0e:3c:22:59:65:76 actions=output:"s4-eth1"
 cookie=0x0, duration=28.644s, table=0, n_packets=24, n_bytes=2240, priority=1,in_port="s4-
eth1",dl_dst=8a:ed:15:dd:a5:3b actions=output:"s4-eth3"
 cookie=0x0, duration=311.855s, table=0, n_packets=7, n_bytes=462, priority=0 actions=CONTROLLER:65535
```