

ENHANCING PRIVATE NETWORK USING SOFTWARE DEFINED NETWORK

A MINI PROJECT REPORT

Submitted by

S. ANBARASU
(REG. NO: 19BIR003)

G. NAVEEN SELVAM
(REG. NO: 19BIR031)

N. TARUN
(REG. NO: 19BIR052)

*in partial fulfillment of the requirements
for the award of the degree of*

BACHELOR OF SCIENCE

IN

INFORMATION SYSTEMS

DEPARTMENT OF COMPUTER TECHNOLOGY - UG

KONGU ENGINEERING COLLEGE
(Autonomous)

PERUNDURAI ERODE – 638 060



November 2021

DEPARTMENT OF COMPUTER TECHNOLOGY- UG
KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI ERODE – 638060

November 2021

BONAFIDE CERTIFICATE

This is to certify that the mini project report titled **ENHANCING PRIVATE NETWORK USING SOFTWARE DEFINED NETWORK** is the bonafide record of work done by **S. ANBARASU** (REG. NO: 19BIR003), **G. NAVEEN SELVAM** (REG. NO: 19BIR031) and **N. TARUN** (REG. NO: 19BIR052) in partial fulfillment for the award of Degree of Bachelor of Science in **Information Systems** of Anna University Chennai during the year 2021-2022.

SUPERVISOR

HEAD OF THE DEPARTMENT

(Signature with seal)

Date:

Submitted for the end semester viva- voce examination held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

DEPARTMENT OF COMPUTER TECHNOLOGY- UG
KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI ERODE – 638060

November 2021

DECLARATION

We affirm that the project titled **ENHANCING PRIVATE NETWORK USING SOFTWARE DEFINED NETWORK** being submitted in partial fulfillment for the award of **B.Sc. Degree in Information Systems** is the original work carried out by us. It has not formed the part of any other project submitted for award of any degree, either in this or any other University.

S. ANBARASU(REG.NO:19BIR003)

G. NAVEEN SELVAM(REG.NO:19BIR031)

N. TARUN(REG.NO:19BIR052)

I certify that the declaration made above by the candidates is true to the best of my knowledge.

DATE :

Signature of the Supervisor
(Name & Designation)

ABSTRACT

Network management is challenging. To operate, maintain, and secure a communication network, network operators must grapple with low-level vendor-specific configuration to implement complex high-level network policies. In order to make networks easier to manage, many solutions to network management problems amount to stop-gap solutions because of the difficulty of changing the underlying infrastructure. The rigidity of the underlying infrastructure presents few possibilities for innovation or improvement, since network devices have generally been closed, proprietary, and vertically integrated. A new paradigm in networking, software defined networking (SDN), advocates separating the data plane and the control plane, making network switches in the data plane simple packet forwarding devices and leaving a logically centralized software program to control the behaviour of the entire network. SDN introduces new possibilities for network management and configuration methods.

In our proposed system, we identify problems with the current state-of-the-art network configuration and management mechanisms. It introduces mechanisms to improve various aspects of network management. The proposed project work focuses on three problems in network management: enabling frequent changes to network conditions and state, providing support for network configuration in a high-level language, and providing better visibility and control over tasks for performing network diagnosis and troubleshooting. The proposed system framework enables network operators to implement a wide range of network policies in a high-level policy language and easily determine sources of performance problems. The deployments in the private networks that show how SDN can improve common network management tasks are demonstrated using a virtual network simulator.

ACKNOWLEDGEMENT

We express our sincere thanks to our beloved Correspondent **Thiru.P.SACHITHANANDAN** and other philanthropic trust members of the Kongu Vellalar Institute of Technology Trust for having provided with necessary resources to complete this project.

We are always grateful to our beloved visionary Principal **V. BALUSAMY B.E., (Hons) M.Tech., Ph.D.**, and thank him for his motivation and moral support.

We express our deep sense of gratitude and profound thanks to **Dr.P.NATESAN M.E., Ph.D.**, Head of the Department, Computer Technology- UG for his invaluable commitment and guidance for this project.

We are in immense pleasure to express our hearty thanks to our beloved Project Coordinators **Dr. N.T. RENUKA DEVI M.C.A., M.Phil., Ph.D** and **Dr.K. SARASWATHI M.Sc., MPhil., Ph.D** and our guide **Dr. S. KALAISELVI M.E., Ph.D** for providing valuable guidance and constant support throughout the course of our project. We also thank the teaching, non-teaching staff members, fellow students and our parents who stood with us to complete our project successfully

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
	ABSTRACT	iv
	LIST OF FIGURES	vii
	LIST OF TABLES	viii
	LIST OF ABBREVIATIONS	ix
1	INTRODUCTION	1
	1.1 RESEARCH PROBLEM	2
	1.1 RESEARCH OBJECTIVES	2
2	LITERATURE REVIEW	3
3	RELATED WORKS	10
	3.1 OVERVIEW	10
	3.2 PLANES OF NETWORKING	10
	3.2.1 DATA PLANE	11
	3.2.2 CONTROL PLANE	13
	3.2.3 MANAGEMENT PLANE	13
	3.3 OPENFLOW (OF) PROTOCOL	13
	3.4 OPENFLOW ARCHITECTURE	14
	3.4.1 SOUTHBOUND	16
	3.4.2 NORTHBOUND	17
	3.4.3 CONTROLLER	18
4	PROPOSED WORK	20
	4.1 BUILD OF PLATFORM	20
	4.1.1 MININET	20
	4.1.2 RYU CONTROLLER	21
	4.2 DESIGN	22
	4.2.1 ALGORITHM DESCRIPTION	23
	4.2.2 NETWORK TOPOLOGY SPECIFICATION	23
	4.3 MODEL DESCRIPTION	24
	4.3.1 TOPOLOGY GENERATION	25

	4.3.2. COMMUNICATION BETWEEN MININET AND RYU CONTROLLER	26
	4.4 PERFORMANCE EVALUATION	26
	4.4.1 SHORTEST PATH COMPUTATION	26
	4.5 RESULT	31
5	CONCLUSION AND FUTURE WORK	32
	APPENDIX	33
	REFERENCES	38

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
3.1	Types of network planes	10
3.2	Open flow architecture	16
4.1	Ryu controller	22
4.2	Network Topology	24
4.3	Topology Generation	25
4.4	Running Ryu controller	26
4.5	TCP flow from h1 to s12	28
4.6	UDP flow from h1 to s12	29
4.7	Hop count and delay	31
A.5.1	Topology creation code	33
A.5.2	Link code	34
A.5.3	Arp handler and shortest path code	35
A.5.4	TCP traffic using iperf	36
A.5.5	UDP traffic using iperf	37

LIST OF ABBREVIATIONS

SDN	Software Defined Network
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
OSPF	Open Shortest Path First
MAC	Media Access Control
ARP	Address Resolution Protocol

CHAPTER 1

INTRODUCTION

Cloud computing today has become an essential and clients service that provides a platform to the user for accessing and accumulating required data. It also includes cloud storage where users can store their required data, which can be used in the future for retrieval as well as for analysis. With a close connection to the Internet of Things, the data routing and storage are controlled by the software-defined system and the cloud platform. On the other hand, for user feasibility, the cloud platform is also utilized in the form of remote access systems like Software as a Service, Platform as a Service, or Infrastructure as a Service where the user can avail of their required software and even hardware support. All these services are supported by the Software-Defined Network, which creates a Virtual Network. This helps for the faster exchange of data in a network with a high analytical performance that is desired for the data. So, the Internet of Things, Cloud Service, and Software Defined Network are, in turn, correlated for the utilization of remote data and to access it for the analytical purpose.

The purpose of establishing a Software Defined Network is to take control of the network elements like router and switches, which are responsible for data routing. Data is transferred from one node to another in the network to reach from the source to the destination following a particular path. Sometimes if the path is long or there is a physical presence of a passive device in the route, there are chances of some data loss. The traditional network of cloud computing contains highly dense servers and switches that are not controlled centrally. An overview of such a traditional network and SDN is shown below where data is coming from cloud computers and transferred to the user.

In the traditional cloud network, there is no specific control over the network device to generate a decision for the shortest route from the source to destination. Thus, the data is traversed through the network, and with the availability of the free path, it will be navigated to reach the destination. It means the data routing is executed without any

centrally controlled device. The main drawback of the traditional network is the low throughput so. There was a need to design a network that can be controlled centrally for the determination of the path through which the data will be navigated easily and without any loss. Thus, this is when Software Defined Network has emerged. Software- Defined Network is the latest technology in the field of cloud computing, which controls the Routers and Switches using the SDN Protocol. Software- Defined Network or SDN communicate with the routers and switches using its protocol, known as OpenFlow. In OpenFlow control, the routing mechanism contains multiple routing tables which are designed with multiple packets forwarding rule. So, at the time of packet transaction, the decision can be taken by the SDN Controller about the rule to be applied. With the application of such Rules in OpenFlow, different actions like forwarding of packet, modification of the content of the packet as well as the routing path and dropping of the packets are performed.

1.1 RESEARCH PROBLEM:

In this research, due to the several limitations of the traditional network, the SDN will be used for the design of the network to manage the resource of it and to take control for routing purposes. So, the ultimate objective is to find the shortest route from the source node to the destination node so that the performance of the SDN can be improved further. The proposed algorithm finds the shortest path in SDN networks to minimize the network trace while sending data packets from source to destination. The proposed algorithm improves the network performance which depends on the hop counts between the switches to transmit the data between two hosts.

1.2 RESEARCH OBJECTIVES:

The following research objectives are addressed to reduce the network traffic for packet transmission from the source node to the destination node. Further, to evaluate performance parameters and ensure Quality of Service (QoS) in SDN, the algorithms are implemented in the router control plane.

CHAPTER 2

LITERATURE REVIEW

Ahmed Dawud Alani, Et al., had published research Entitled Software defined networks challenges and future direction of research. Cloud computing are developing continuously, and companies are collecting their data in one data center so they can make benefit of the predictability, continuity and quality of service that provided by the cloud through using virtualization technologies. At the same time, the networks that provided energy efficiency and high security are highly demanded these days. Application providers and network operators are needing an efficient network solution to move with the speedily growing in the data and the high demand on the speed and effective services. SDN networks are arise as a proficient network technology that can be an appropriate with the constantly evolving future network functions and smart applications and at the same time reducing the cost of the of using these networks and services through make the software, hardware, and administration more simply to use and highly effective. Many research challenges appear to provide like this network solutions as how to reach a good Quality of Service (QoS), High data security, Network scalability and Best possible load balancing. Therefore, the goal of our paper is to analyze SDN NETWORK and give a view on the challenges that face its work.

Nitheesh Murugan Kaliyamurthy, Et al., Had presented a research article on Software-Defined Networking: Software-defined networking is an evolving network architecture beheading the traditional network architecture focusing its disadvantages on a limited perspective. A couple of decades before, programming and networking were viewed as different domains which today with the lights of SDN bridging themselves together. I am to overcome the existing challenges faced by the networking domain and an attempt to propose cost-efficient effective and feasible solutions. Changes to the existing network architecture are inevitable considering the volume of connected devices and the data being held together. SDN introduces a decoupled architecture and brings customization within the network making it easy to configure, manage, and troubleshoot.

'Are paper focuses on the evolving network architecture, the software-defined networking. Unlike a generic view on the evolving network, which makes work as a review, this work addresses various perspectives of the architecture leaving it an intermediate work in between the review of the literature and implementation, contributing towards factors like the design, programmability, security, security behaviors, and security lapses. 'Is paper also analyses various weak points of the architecture and evolves the attack vectors in each plane leaving a conclusion to further progress towards identifying the impacts of the attacks and proposing mitigation strategies.

Network cost optimization-based capacitated controller deployment for SDN [3] was presented by Rong Chai, Xicheng Yang, Chun ling Du, Qianbin Chen. Here, as a novel network paradigm, software-defined networking (SDN) is capable of simplifying network management and offering flexible support to various user services. To meet the rapidly increasing transmission demands of SDN switches, the controller deployment strategy in an SDN scenario should be designed. In this paper, we investigate the capacitated controller deployment problem for SDN. Consider the signaling transmission and processing performance of switches and address the worst-case performance, we define network response time (NRT) as the maximum control plane response time of switches. Then aiming to achieve the tradeoff between NRT and the cost of controllers, we introduce the concept of network cost which is defined as the weighted sum of NRT and controller cost. The capacitated controller deployment problem is formulated as a constrained network cost minimization problem. To solve the optimization problem, we propose a two-stage heuristic algorithm, which first tackles the controller deployment subproblem under the unlimited capacity constraint, and then solves controller-type matching subproblem. Specifically, during the first stage, a minimum eccentricity-based controller deployment algorithm is designed to determine the number and location of controllers as well as the association strategy between controllers and switches. During the second stage, a greedy method-based controller-type matching strategy is proposed to determine the types of deployed controllers. Extensive simulations are performed, and the results certify the effectiveness of the proposed algorithm.

Hybrid SDN evolution: A comprehensive survey of the state-of-the-art was presented by Sajad Khorsandroo ^a, Adrián Gallego Sánchez ^b, Ali Saman Tosun ^c, JM Arco ^d, Roberto Doriguzzi-Corin ^e. Software-Defined Networking (SDN) is an evolutionary networking paradigm which has been adopted by large network and cloud providers, among which are Tech Giants. However, embracing a new and futuristic paradigm as an alternative to well-established and mature legacy networking paradigm requires a lot of time along with considerable financial resources and technical expertise. Consequently, many enterprises cannot afford it. A compromise solution then is a hybrid networking environment (a.k.a. Hybrid SDN (hSDN)) in which SDN functionalities are leveraged while existing traditional network infrastructures are acknowledged. Recently, hSDN has been seen as a viable networking solution for a diverse range of businesses and organizations. Accordingly, the body of literature on hSDN research has improved remarkably. On this account, we present this paper as a comprehensive state-of-the-art survey which expands upon hSDN from many different perspectives.

Hamza Mokhtar Et al., proposed a work on Multiple-level threshold load balancing in distributed SDN controllers. It focuses on the distributed Software-Defined Networking (SDN) approach that has been adopted to address the scalability issue linked to the use of a single SDN controller, particularly in large-scale networks. However, this method gives rise to a new challenge of uneven load balance on the distributed SDN controller. Several studies tried to solve this issue, but they have focused only on balancing the load when an overload occurs and have not achieved the load balance continuously. In this paper, we propose the multiple threshold load balance (MTLB) switch migration scheme to achieve continuous load balancing among the controllers. This approach classifies the load into several gradual levels that represents the basis for the switch migration when the load of a controller differs from others, which lead the threshold value to be dynamically adjusted. Moreover, our scheme reduces the overhead of unwanted update operations among the controllers by adopting the controller load status as an indicator for disseminating the load information. We implement our scheme using the Floodlight controller and Mininet emulator. The experimental results show that our scheme has better comprehensive

performance than the other schemes in terms of response time, control overhead, and throughput rate.

Linking handover delay to load balancing in SDN-based heterogeneous Networks is proposed by Modhawi Alotaibi and Amiya Nayak. This proposes the Software-Defined Networking (SDN) paradigm provides the ability to handle mobility more efficiently due to its programmability and fine granularity. However, in this emerging setting, the handover procedure still suffers delay due to exchanging and processing handover signaling messages. In this paper, we study the relevancy between an SDN controller's load and handover delay. We show that an over-loading state can prolong handover delay, so as a countermeasure, reaching that state is mitigated by applying a load balancing mechanism. Our primary metric is the controller's response time, as it directly affects the completion of any mobility-related procedure. We propose a load balancing management framework that deploys two concepts: network heterogeneity and context-aware vertical mobility. Our proposal is composed of three main aspects. First, we identify candidate users based on their context information. Second, we reduce the frequency of load dissemination between multiple controllers, and hence, reducing processing and communication overhead. Third, after the candidate users are determined, we optimize the decision problem on the selection among heterogeneous candidate networks. Through simulation, our framework has shown as much drop as a 28% drop-in response time compared to previous proposals.

Wei-Che Chine Et al., explained the about an SDN-SFC-based service-oriented load balancing for the IoT applications. Nowadays, with the rapid advance of network technology, the miniaturization of terminal equipment, and the trend of constant reduction of costs, the topic of IoT turns increasingly popular. As IoT becomes common in daily life, one can collect information at any time and in any environment, such as temperature, humidity, and PM2.5. Based on the analysis of such data, we can learn the degree of environmental comfort of a city, which is beneficial for the development of a smart city and other applications. However, the use of many terminal equipment is likely to trigger huge demands of bandwidth, followed by the increase in the time of data

transmission. To address this issue, this study proposed a service-oriented SDN-SFC load balance mechanism. It considered and classified the type and priority of service required by each terminal device. Then, it adopted the heuristic algorithm to plan the transmission paths among SFCs to reduce the load of each SF and improve the overall network performance. The simulation results indicate that the method proposed by this study can shorten the time of data transmission and achieve load balance.

Yingying Guo Et al., proposed a work on Routing optimization with path cardinality constraints in a hybrid SDN. The emergence of Software Defined Networking (SDN) increases the flexibility of routing and provides an efficient approach to balance network flows. Due to the economic and technical challenges in transiting to a full SDN-enabled network, a hybrid SDN, with a partial deployment of SDN switches in a traditional network, has been a prevailing network architecture. For a hybrid SDN, the routing flexibility is influenced by the expensive and limited Ternary Content Addressable Memory (TCAM) resource, where SDN switches store the flow entries dispatched from SDN controllers. Considering the limited TCAM resource, to reduce the abundant flow entries in TCAM, we propose to impose the constraints on the number of routing paths (i.e., path cardinality constraints) when optimize flows routing. To solve this problem, in this paper, we first formulate the routing optimization problem with the path cardinality constraints in a hybrid SDN as a Mixed Integer Non-Linear Programming (MINLP) problem. Then, we propose an incremental deployment method for obtaining a hybrid SDN and an H-permissible Paths Routing Scheme (HPRS) to effectively route traffic flows under the path cardinality constraints. After that, the theoretic analysis is given to prove that the approximation ratio of HPRS is $O(\log L)$. Finally, through extensive experiments, we demonstrate that our proposed algorithm HPRS can efficiently reduce flow entries and approximates optimal routing under different network topologies

Huang Fei Song Et al., Entitled the work on Fast and Consistent Network Reconfiguration with low latency for SDN. Software-defined network (SDN) is one of the important technologies to achieve the customization of network services. The key to realize a highly adaptive SDN, which can respond to the changing demands or recover

after network failure in a short period of time, is to update the configuration effectively. However, the inconsistent configuration update may lead to transient and incorrect network behaviors and long reconfiguration time between current configuration and target configuration may degrade undesired network performance. Most of the existing works to reduce the network reconfiguration time focus on the consistent update between current configuration and target configuration and ignore the impact of different target configurations on network reconfiguration time. Therefore, reducing the network reconfiguration time needs to consider not only the consistent update between current configuration and target configuration, but also the impact of different target configurations. In this paper, we first analyze the impact of different target configuration of the control plane on the update operation in the forwarding rules of the data plane.. Subsequently, we propose the consistent scheduling update (CSU) algorithm to solve the consistency conversion from the current configuration to the target configuration. Finally, experimental results demonstrate that our algorithms can reduce the network reconfiguration time up to 39% compared with previous SDN consistent update methods while keeping the similar consistency.

Resilient backup controller placement in distributed SDN under critical targeted attacks proposed by Eusebia Calle, David Martínez. Today, telecommunication networks are crucial infrastructures, as has, for example, been demonstrated by the COVID-19 crisis. Thus, protecting such infrastructures, including software-defined based networks (SDN), is of the utmost importance for network providers to assure society has constant access to reliable services. Targeted attacks on SDN can seriously affect their connectivity and thus service continuity. Such an attack, launched on network nodes, divides the network into disjoint components and, since the number of SDN controllers is limited, results in isolating a significant proportion of nodes from the (surviving) controllers, causing major disruptions in service availability. In this paper, we present an optimization approach which can be used by the SDN network operator to properly locate the controllers by considering predictable sets of critical targeted attacks on network topology. The proposed approach includes an algorithm for predicting, based on appropriately defined attack effectiveness measures, the sets of most dangerous attacks.

Such sets are then used as input data for controller placement optimization, which is performed by means of mixed-integer programming methods. In the optimization, the impact of the considered attacks is measured by a novel network availability measure. To minimize the consequences of attacks we consider additional backup controllers. Finally, we present results of a numerical study based on the introduced approach that illustrate the effectiveness of our approach.

CHAPTER 3

RELATED WORK

3.1 OVERVIEW:

These days the usage of networks has increased a lot. The pandemic has made the rise in network areas. As the usage increased workload also increased, the servers lost the capable speed, server crash took place, bandwidth required rate increased, many technical issues were caused. These all took place because of the low-level configuration devices which cannot cope up with high-level network policies. To overcome this several techniques are available, in this project to solve by using open flow protocol and This gives rise to the new paradigm of software define networks, the possibility of enabling high-level policies to interpret with the low-level configuration device.

3.2 PLANES OF NETWORKING:

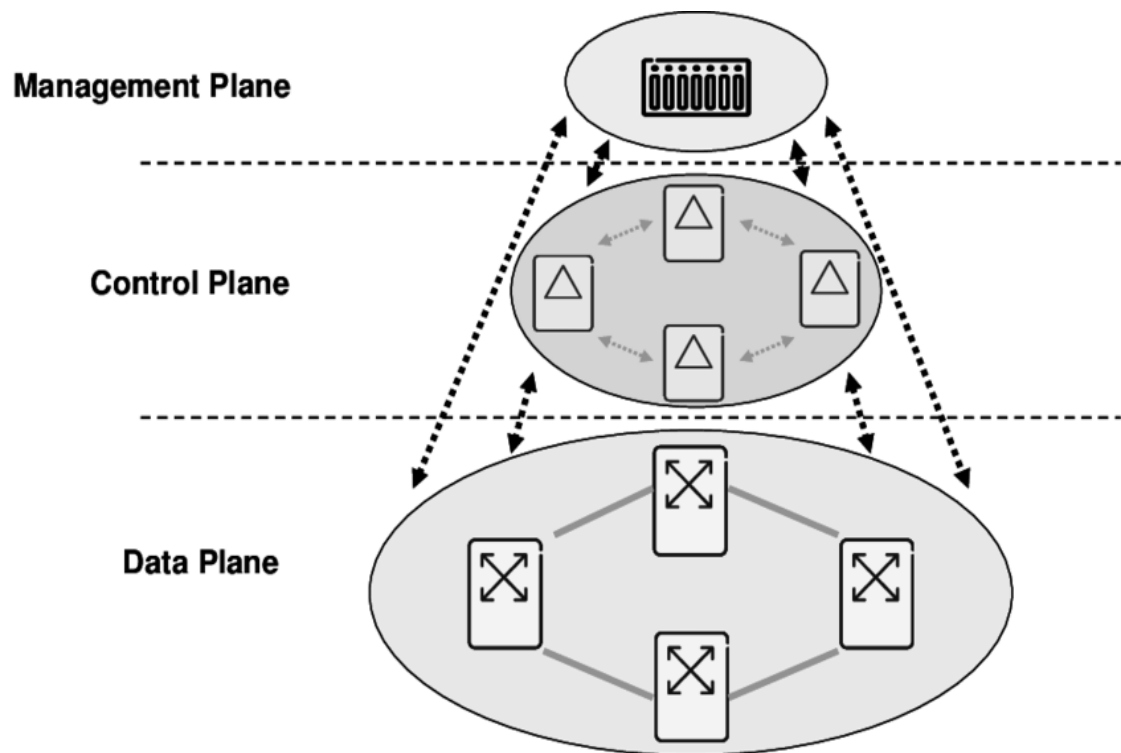


Figure 3.1 Types of network planes

3.2.1 DATA PLANE:

A data plane, or forwarding plane, is the part of a router that examines an incoming data packet and sends it to the correct output destination on the network. A data packet header contains information about where the packet came from and where it needs to go, which the data plane uses to direct network traffic. The data plane accesses a table, sometimes called a router table, which retrieves previously listed IP addresses to which a packet can be sent. Sending the packet to the right location is called forwarding. The table will also contain instructions to drop a data packet if it doesn't meet required specifications. These specifications are based on the router's configuration and what traffic it can permit on the network.

Some routers have multiple-forwarding capabilities, which allow them to process more packets at a time. Some routers, upon receiving an invalid or forbidden packet, are also configured to send a message to the sender alerting them that their request failed. But some routers have security features that force a data plane to drop packets without sending any notification.

This protects the destination IP address from malicious traffic and keeps the sender from knowing more details about the request than they should. A data plane differs from a control plane, which manages forwarding paths in an entire network—in other words, managing the entire routing process. A data plane is restricted to one router and manages packet forwarding only for it. Forwarding is considered part of Layer 3 of the OSI model, a seven-layer conceptual model that helps clarify how network connections are processed and completed. Layer 3, known as the Network Layer, manages all the routing and data transmission within a network. IP addresses are also part of the Network Layer.

3.2.2 CONTROL PLANE:

In network routing, the control plane is the part of the router architecture that is concerned with drawing the network topology, or the information in a routing table that defines what to do with incoming packets. Control plane functions, such as

participating in routing protocols, run in the architectural control element. In most cases, the routing table contains a list of destination addresses and the outgoing interface(s) associated with each. Control plane logic also can identify certain packets to be discarded, as well as preferential treatment of certain packets for which a high quality of service is defined by such mechanisms as differentiated services. Depending on the specific router implementation, there may be a separate forwarding information base that is populated by the control plane, but used by the high-speed forwarding plane to look up packets and decide how to handle them.

In computing, the control plane is the part of the software that configures and shuts down the data plane. By contrast, the data plane is the part of the software that processes the data requests. The data plane is also sometimes referred to as the forwarding plane. The distinction has proven useful in the networking field where it originated, as it separates the concerns: the data plane is optimized for speed of processing, and for simplicity and regularity. The control plane is optimized for customizability, handling policies, handling exceptional situations, and in general facilitating and simplifying the data plane processing.

The conceptual separation of the data plane from the control plane has been done for years. An early example is Unix, where the basic file operations are open, close for the control plane and read write for the data plane. In contrast to the control plane, which determines how packets should be forwarded, the data plane actually forwards the packets. The data plane is also called the forwarding plane. Think of the control plane as being like the stoplights that operate at the intersections of a city. Meanwhile, the data plane (or the forwarding plane) is more like the cars that drive on the roads, stop at the intersections, and obey the stoplights.

3.2.3 MANAGEMENT PLANE:

The management plane typically handles high-level network management and operations including network monitoring and customer billing. The telecommunication FCAPS (Fault, Configuration, Accounting, Performance, Security) model describes the functions of network management. The data plane carries the user data,

and employs interfaces to network transport equipment such as transponders, optical amplifiers, and optical and electronic switches and switch ports.

The control plane operates between these two layers. The role of the control plane is evolving, but generally, the control plane handles automated network provisioning, some fault recovery, and other administrative control functions like topology management and liveness verification. As shown in Figure 16.4 illustrations of example control plane architectures, the control plane data communications can be provided in-band or out-of-band. The control plane data communications topology can also be isomorphic to the communications data plane, or not, as the use case and technology demand.

The management plane addresses router configuration and collection of various statistics, such as packet throughput, on a link. Router configuration refers to configuration of a router in a network by assigning an IP address, identifying links to its adjacent routers, invoking one or more routing protocols for operational use, and so on. Statistics collection may be done, for example, through a protocol known as the Simple Network Management Protocol (SNMP). The management plane of a router is closely associated with network operations. The management plane may also include human actions to configure and support networks. Unlike a traditional switch where the management plane is integrated into the hardware, the VSM is deployed as a VM running the NX-OS as the OS. It is installed using either an ISO file or an open virtualization format (OVF) template.

3.3 OPENFLOW (OF) PROTOCOL:

OpenFlow is an open standard for a communications protocol that enables the control plane to break off and interact with the forwarding plane of multiple devices from some central point, decoupling roles for higher functionality and programmability. One of the key concepts to understanding SDN is the separation of control plane and data plane. Typically, a network is comprised of many routers and switches, each exchanging table information to build topologies. Each of these network devices has their own individualized control plane for brain-like functions such as route or MAC learning. Each network device also has its own data plane for forwarding packets. The challenge is each device has its own perspective of the network, and the only way you can view that

perspective is by connecting to that device via a CLI and issue commands or configurations. The same is applicable for other devices like firewalls, load balancers, not just routers, and switches.

A control plane is the brain of the operations. It typically runs in software and builds the necessary tables for forwarding, such as the RIB or MAC tables. This table is typically sent as a copy down to the forwarding plane and installed in hardware to allow for high throughput forwarding of traffic. These two planes (and usually an additional management plane for things like SSH, SNMP, etc.) are traditionally running on every single network device in a network. The Open Networking Foundation (ONF), a user-led organization dedicated to promotion and adoption of software-defined networking (SDN), manages the OpenFlow standard.

ONF defines OpenFlow as the first standard communications interface defined between the control and forwarding layers of an SDN architecture. OpenFlow allows direct access to and manipulation of the forwarding plane of network devices such as switches and routers, both physical and virtual (hypervisor-based). It is the absence of an open interface to the forwarding plane that has led to the characterization of today's networking devices as monolithic, closed, and mainframe-like. A protocol like OpenFlow is needed to move network control out of proprietary network switches and into control software that's open source and locally managed.

3.4 OPENFLOW ARCHITECTURE:

SDN provides separation between the control plane (controller) and data plane (switch) functions of networks using a protocol that modifies forwarding tables in network switches. This makes it possible to optimize networks on the fly and quickly respond to changes in network usage without the need for manually reconfiguring existing infrastructure or purchasing new hardware. SDN separates the control of network devices from the data they transport, and the switching software from the actual network hardware. It also provides an entity, the controller, that has a comprehensive view of the entire network and its status, and with which switches (network resources) and applications (network consumers) can communicate in real-time. The controller makes it possible for networks to

interact with applications and efficiently reconfigure themselves at need, allowing them to implement multiple logical network topologies on a single common network fabric.

OpenFlow is a multivendor standard defined by the Open Networking Foundation (ONF) for implementing SDN in networking equipment. The OpenFlow protocol defines the interface between an OpenFlow Controller and an OpenFlow switch, see Figure 1 below. The OpenFlow protocol allows the OpenFlow Controller to instruct the OpenFlow switch on how to handle incoming data packets. The OpenFlow instructions transmitted from an OpenFlow Controller to an OpenFlow switch are structured as “flows”. Each individual flow contains packet match fields, flow priority, various counters, packet processing instructions, flow timeouts and a cookie.

The flows are organized in tables. An incoming packet may be processed by flows in multiple “pipelined” tables before exiting on an egress port. The OpenFlow protocol standard is evolving quickly with release 1.6 as the current revision at the time of this blog being published. The OpenFlow controller maintains the OpenFlow protocol communications channels to the OpenFlow switches, maintains a local state graph of the OpenFlow switches and exposes a northbound API to the OpenFlow applications. The northbound API may be viewed as an abstraction of the network and allows the OpenFlow applications to read the state of the network and to instruct the network to perform various tasks.

A real world OpenFlow capable network may consist of only OpenFlow switches or a mixture of OpenFlow switches and traditional switches and routers. The latter network type is called an overlay network. Some OpenFlow applications will require only partial deployment of OpenFlow switches whereas others require a network consisting of only OpenFlow switches.

Most of the SDN network fabrics and applications we will discuss in this blog can be introduced as overlays, and multiple applications can be introduced in a tagged fashion, with new ones building on the foundations laid by the previous applications.

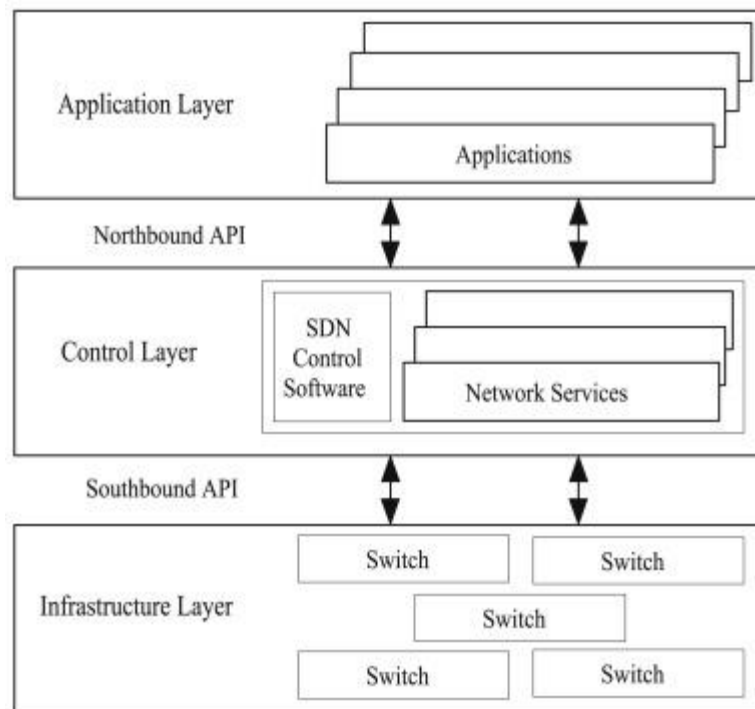


Figure 3.2: OpenFlow Architecture

3.4.2 SOUTHBOUND:

The Southbound Interface is a collection of drivers that handles communication to all data-plane elements in the network. OpenFlow Driver. The OpenFlow is an open communications interface between control plane and the forwarding layers of a network. A southbound interface (SBI) is a component's lower-level interface layer. It is directly connected to that lower layer's northbound interface. It breaks down the concepts into smaller technical details that are specifically geared toward a lower layer component within the architecture.

In software-defined networking (SDN), the southbound interface serves as the OpenFlow or alternative protocol specification. It allows a network component to communicate with a lower-level component. The main objective of a southbound interface is to provide communication and management between the network's SDN

controller, nodes, physical/virtual switches and routers. It allows the router to discover the network topology, define network flow and implement several requests relayed from northbound application programming interfaces (API).

The management of network nodes is handled by the Network Management System (NMS) allowed by the southbound interface. The southbound integration is supported by the following interfaces:

- Simple Network Management Protocol (SNMP)
- Command Line Interface (CLI)
- File Transfer Protocol (FTP) or SSH File Transfer Protocol (SFTP)
- Telnet (TN) or Secure Shell (SSH)

For the southbound interface, it uses common protocols like Telnet, SSH and SNMP to communicate with your hardware. The control / data plane remains in your switches, routers, and other devices. APIC-EM is not an SDN controller that replaces the control plane

3.4.3 NORTHBOUND:

A northbound interface of a component is an interface that allows the component to communicate with a higher-level component, using the latter component's southbound interface. The northbound interface conceptualizes the lower-level details (e.g., data or functions) used by, or in, the component, allowing the component to interface with higher level layers. In architectural overviews, the northbound interface is normally drawn at the top of the component it is defined in; hence the name northbound interface. A southbound interface decomposes concepts in the technical details, mostly specific to a single component of the architecture. Southbound interfaces are drawn at the bottom of an architectural overview. A northbound interface is typically an output-only interface (as opposed to one that accepts user input) found in carrier-grade network and telecommunications network elements. The languages or protocols commonly used include SNMP and TL1. For example, a device that is capable of sending out syslog messages but that is not configurable by the user is said to implement a northbound interface. Other

examples include SMASH, IPMI, WSMAN, and SOAP. northbound flow can be thought of as going upward, while southbound flow can be thought of as going downward.

In architectural diagrams, northbound interfaces are drawn at the top of the applicable component, while southbound interfaces are drawn at the bottom of the component. While the terms northbound and southbound can apply to almost any type of network or computer system, in recent years they have been used increasingly in conjunction with application program interfaces (APIs) in software-defined networking (SDN). In an enterprise data centre, functions of northbound APIs include management solutions for automation and orchestration, and the sharing of actionable data between systems. Functions of southbound APIs include communication with the switch fabric, network virtualization protocols, or the integration of a distributed computing network. In NMS Northbound Interface (NBI) - is a InfiMONITOR's programming interface for integration with the higher-level NMS (Network Management System). Thus, NBI allows to perform integration to the existing infrastructure. Integration with InfiMONITOR is performed through software.

3.4.4 CONTROLLER

A Controller is an application in a software-defined networking (SDN) architecture that manages flow control for improved network management and application performance. The SDN controller platform typically runs on a server and uses protocols to tell switches where to send packets. SDN controller direct traffic according to forwarding policies that a network operator puts in place, thereby minimizing manual configurations for individual network devices. By taking the control plane off of the network hardware and running it instead as software, the centralized controller facilitates automated network management and makes it easier to integrate and administer business applications. In effect, the SDN controller serves as a sort of operating system (OS) for the network. The controller is the core of a software-defined network. It resides between network devices at one end of the network and applications at the other end. Any communication between applications and network devices must go through the controller.

The controller communicates with applications -- such as firewalls or load balancers -- via northbound interfaces. The Open Networking Foundation (ONF) created a working group in 2013 focused specifically on northbound APIs and their development. The industry never settled on a standardized set, however, largely because application requirements vary so widely. The controller talks with individual network devices using a southbound interface, traditionally one like the OpenFlow protocol. These southbound protocols allow the controller to configure network devices and choose the optimal network path for application traffic. OpenFlow was created by ONF in 2011. In addition to centralizing and simplifying the control of enterprise network management, SDN offers the following succinct advantages: Traffic programmability. Greater agility. Capacity to generate policy-driven network supervision. Its greatest advantage is allowing the creation of a framework to support more data-intensive applications like big data and virtualization.

Traditionally, SDN controllers are used in data centre networks. As SDN technology evolved, however, the WAN became a compelling use case, driving the growth of software-defined WAN (SD-WAN) technology. An SD-WAN controller performs many of the same duties as an SDN controller, following policy configurations to direct WAN traffic over the most efficient route. The SD-WAN market has fewer notable open-source options than SDN, as most SD-WAN controllers typically come tied together with the vendor's proprietary SD-WAN platform.

CHAPTER 4

PROPOSED WORK

4.1 BUILD OF PLATFORM:

In this project, the algorithm is implemented in the virtual environment that is created by the Oracle Virtual Box. In the Virtual Box, Ubuntu 20.04 is installed for creating the operating environment for the simulation. The simulation of the network is done using a Mininet simulator, which is capable of creating the network environment and the relevant simulation within the scope of the virtual environment. The framework is designed where network topology is created to find the shortest path between the source node and the destination node. The framework is segregated into different layers of the network, and in each layer, there are some network components present which define the data routing partially. The bottom layer contains the data plane layer in which the Mininet simulation environment is installed. So, all the network creation will be done here. The upper section or layer contains the Network Control Layer, where the Ryu Controller is installed where the routing control is done in this layer. So, below are the description of the required simulation components and the environments.

4.1.1 MININET:

In a virtual environment to simulate a large network, Mininet is the open-source network simulator for Software Defined Network. The primary reason to use the Mininet is that it supports OpenFlow Protocol, which is essential for the network configuration and computation for Software Defined Network. It also provides an inexpensive platform for developing, testing, and creating custom topologies in the network. The remarkable features of Mininet are

- Mininet is not sensitive to a specific programming language. So, any programming language can be used for the creation of the network topology within the scope of the environment.
- Mininet can virtualize the network topology in the virtual system of the host, and when the topology executes, the relevant source code will not be modified. So, the network topology can be created easily within the environment.
- The network that is created within the Mininet simulator is done in real-time, and so the real-time network topology can be created and so the simulation can be done in real-time.
- Mininet is feasible to add huge numbers of hosts and nodes into the network, and thus, the dense network can be created in real-time.
- The network that is designed using Mininet can act as the real network. It means the behaviour of the designed network supports the features of the real network devices.
- Mininet is Open Source, and the use of the Mininet in the virtual box does not draw any credential and, thus, easy to use for this design.

4.1.2 RYU CONTROLLER:

Ryu controller is basically the SDN Controller, which works as the main controlling device on the SDN. It makes control over the switches and the devices.

Additionally, it controls the flow table so that the packet can be routed throughout the network using the Switch Control. The approach of using Ryu Controller in this project is that Ryu Controller can be easily programmed with Python language and is supported by the OpenFlow protocol, which is essential for the design and packet routing operations. below represents how the RYU controller acts as a mediator between the application plane and data plane in SDN.

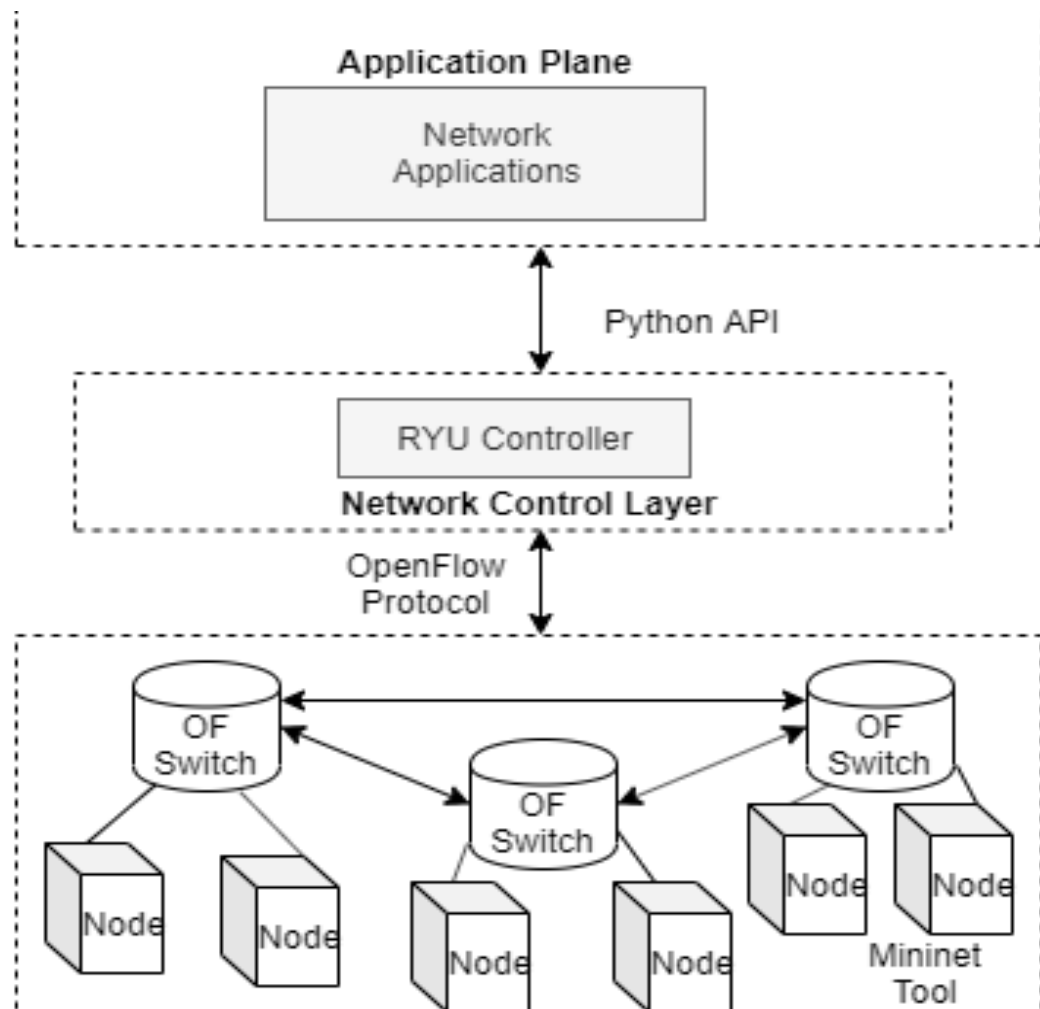


Figure 4.1: Ryu Controller

4.2 DESIGN:

This section explains about the algorithmic flow and the network topology design used in the proposed project.

4.2.1 ALGORITHM DESCRIPTION:

The aim of the routing algorithm is to identify all available network paths in the topology and discover the best and the shortest path with the least cost in order to route the large ow. All the ow will be pushed to the switches and forwarded accordingly. The algorithm is calculating the transmitted and received bytes on the switch ports and further calculating the computational cost of all the paths to get the minimum cost to reach the destination.

The Arp Handler class in the controller is used to initialize the information of network topology. The create interior links function gets all the links of the source and the destination ports. The packet in handler function checks if the packet contains all the packet header information. If it is present, the algorithm is executed, and if not, the packet is dropped. When the MAC address of source node and destination node match all the packet header, information is passed to install path function. Finally, the packet is sent to the destination node, and the Datapath is printed.

4.2.2 NETWORK TOPOLOGY SPECIFICATIONS:

Mininet can be used to create the network topology using the node, edges, and the host that are discussed in the previous section. While the creation of the network topology, the Ryu controller will be acting as the interface controller, and thus, the

packet routing can be taken place in that network design. In this project, Fat-tree network topology is designed with the following specifications:

- 15 Hosts
- 16 Switches
- RYU Controller

As network throughput and fault tolerance are the main objectives of data networks. thus, Fat-tree topology is used in SDN Open ow protocol. In this topology, all the switches are interconnected to each other, forming a three-layer architecture of switches: core switches, aggregation switches, and edge switches.

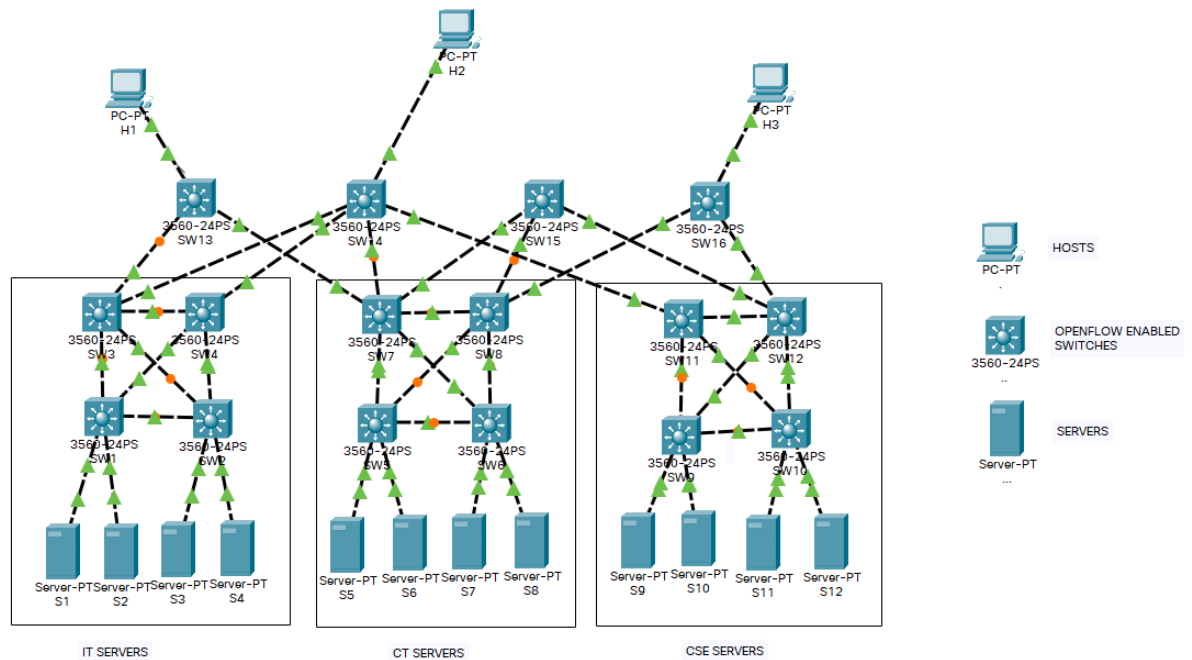


Figure 4.2 Network Topology

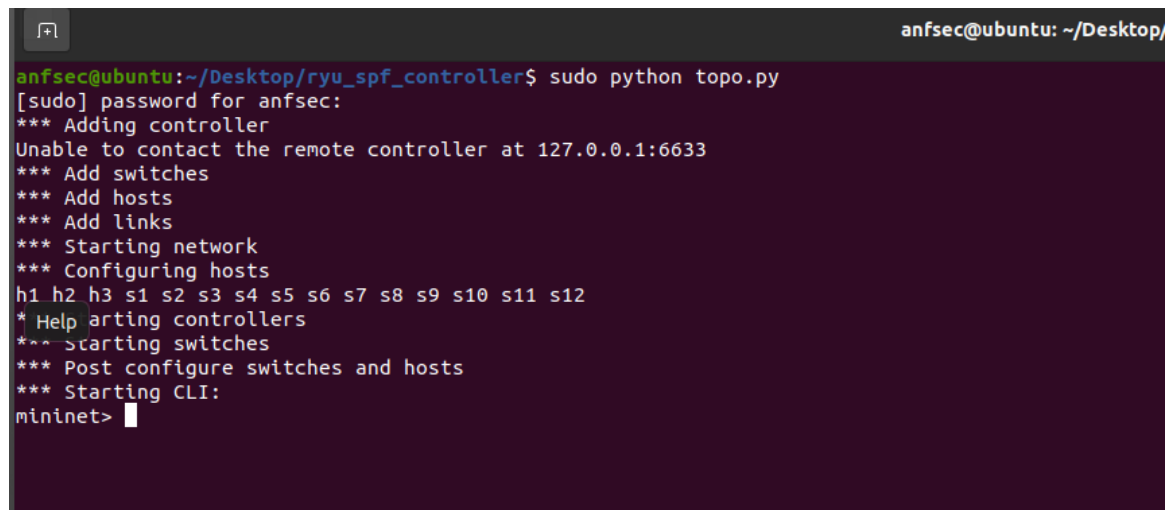
The Ryu manager helps to load the shortest path algorithm files in the RYU controller. It observes links that are formed between hosts and switches and get the shortest path for hop count and delay.

4.3 MODEL DESCRIPTION:

This section explains the steps taken to implement the proposed shortest path algorithm. To achieve efficient utilization of network resources, the Mininet tool is used for network topology generation that interacts with the RYU controller for sending messages to the destination node. The hosts and switches are controlled by the Controller with the help of flow tables. In order to choose a suitable path, the algorithm suggested, and network traffic is monitored by the Controller.

4.3.1 TOPOLOGY GENERATION:

The controller in SDN stores all the information of network topology. And the proposed algorithm is used to find the shortest path for the available network topology. below represents that the network is established between the host and switches.



```

anfsec@ubuntu: ~/Desktop/ryu_spf_controller$ sudo python topo.py
[sudo] password for anfsec:
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
* Help starting controllers
*** Starting switches
*** Post configure switches and hosts
*** Starting CLI:
mininet>

```

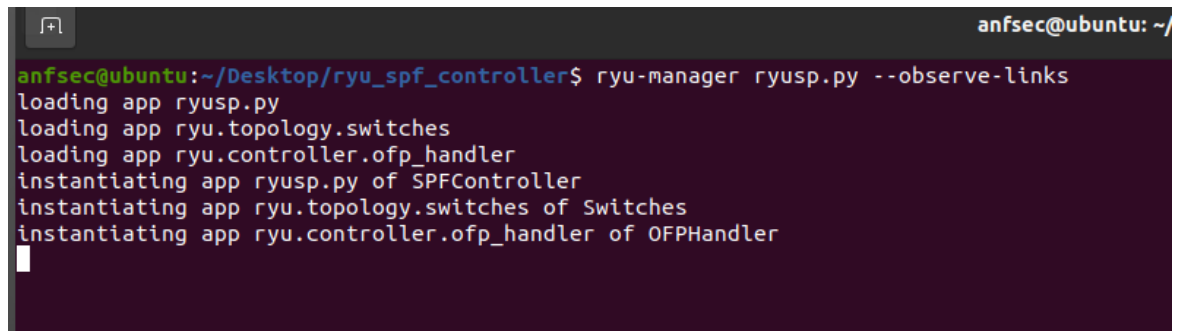
Figure 4.3 Topology Generation

To check the connectivity, the ping all command in the Mininet displays the connectivity between every host in the system and tests whether every host is active and reachable to each other. If the hosts are active, it shows 0% dropped, and all the packets are received.

Sometimes ping all command may take about 30 minutes or more to complete the connection between the hosts. The links between the hosts and switches are set to 0.20 Mbit, 0.10 Mbit, and 0.05 Mbit bandwidths for the transmission of data packets.

4.3.2 COMMUNICATION BETWEEN MININET AND RYU CONTROLLER:

To create a connection between network topology and controller, the switches communicate with the controller using the default port 6633. Each switch in the topology is assigned a unique port for keeping track of packages sent. In this project, 16 OpenFlow switches are connected to 15 hosts. Out of these 16 switches, 4 switches from S13 are the core switches then, 8 switches from S1 to S12 are the aggregation switches, and the last 8 switches are edges switches which are connected to 8 hosts in the network. To calculate the shortest path for the fat-tree topology, the controller script is executed.



```

anfsec@ubuntu: ~/Desktop/ryu_spf_controller$ ryu-manager ryusp.py --observe-links
loading app ryusp.py
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app ryusp.py of SPFController
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler

```

Figure 4.4 Running Ryu Controller

4.4 PERFORMANCE EVALUATION:

This section presents the outcome for the packet transmission between the source node to the destination node. The performance parameters considered for the packet simulations are Hop count, Delay (weight), Path bandwidth (throughput), Packets transferred, and Time.

4.4.1 SHORTEST PATH COMPUTATION:

In SDN, the shortest path is determined based on Open shortest Path First (OSPF), which communicates with the OpenFlow controller and routers in the network. The packet information is stored in ARP Handler in the controller, which is based on the Address Resolution Protocol (ARP). As the controller operates in two modes, i.e., proactive mode and reactive mode. In the proactive mode, the controller gets the information from switches, and in the reactive mode, the switches forward the ARP request to the controller and compute the shortest path.

(i) Comparison of TCP and UDP Measurement:

In a network, the performance of packet transmission from source to destination is dependent on the bandwidth of data and the average data rate of packets transferred. In general, the difference between TCP and UDP bandwidth is that the amount of data generated in a network is calculated by TCP, while in UDP, the rate of data to be transmitted needs to be defined. Represents the throughput for a specified time interval. The graph is plotted using gnu plot tool where X-axis is set to "Time (sec)", and Y-axis is set to "Throughput (Gbps)". Thus, it can be stated that for TCP, the bandwidth keeps on fluctuating, and the highest bandwidth measured was around 6.4 Gbps at 0 to 15 sec time intervals. While for the UDP, the highest bandwidth was 10.5 Gbps at 0 to 15 sec time intervals.

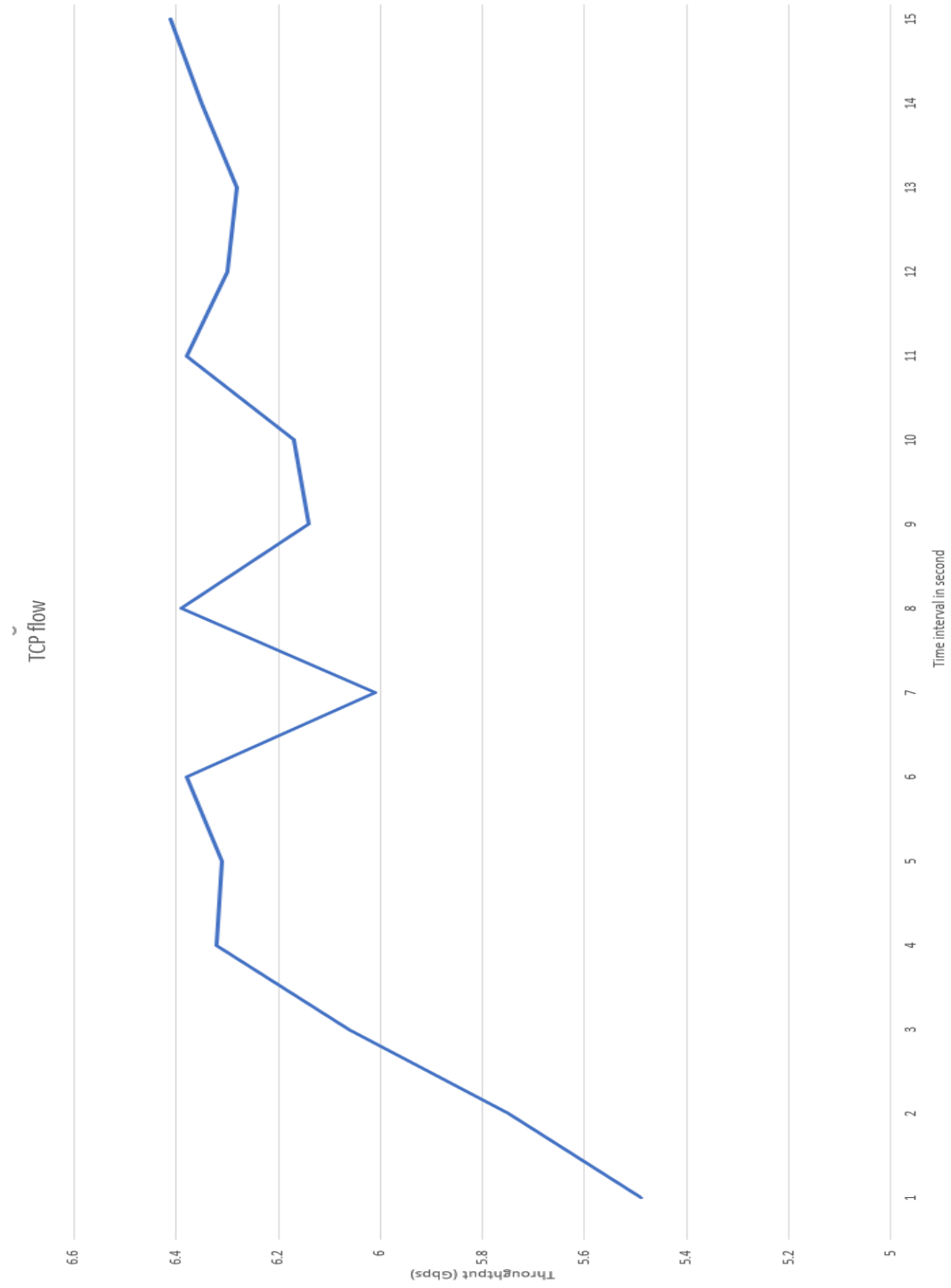


Figure 4.5 TCP flow from h1 to s12

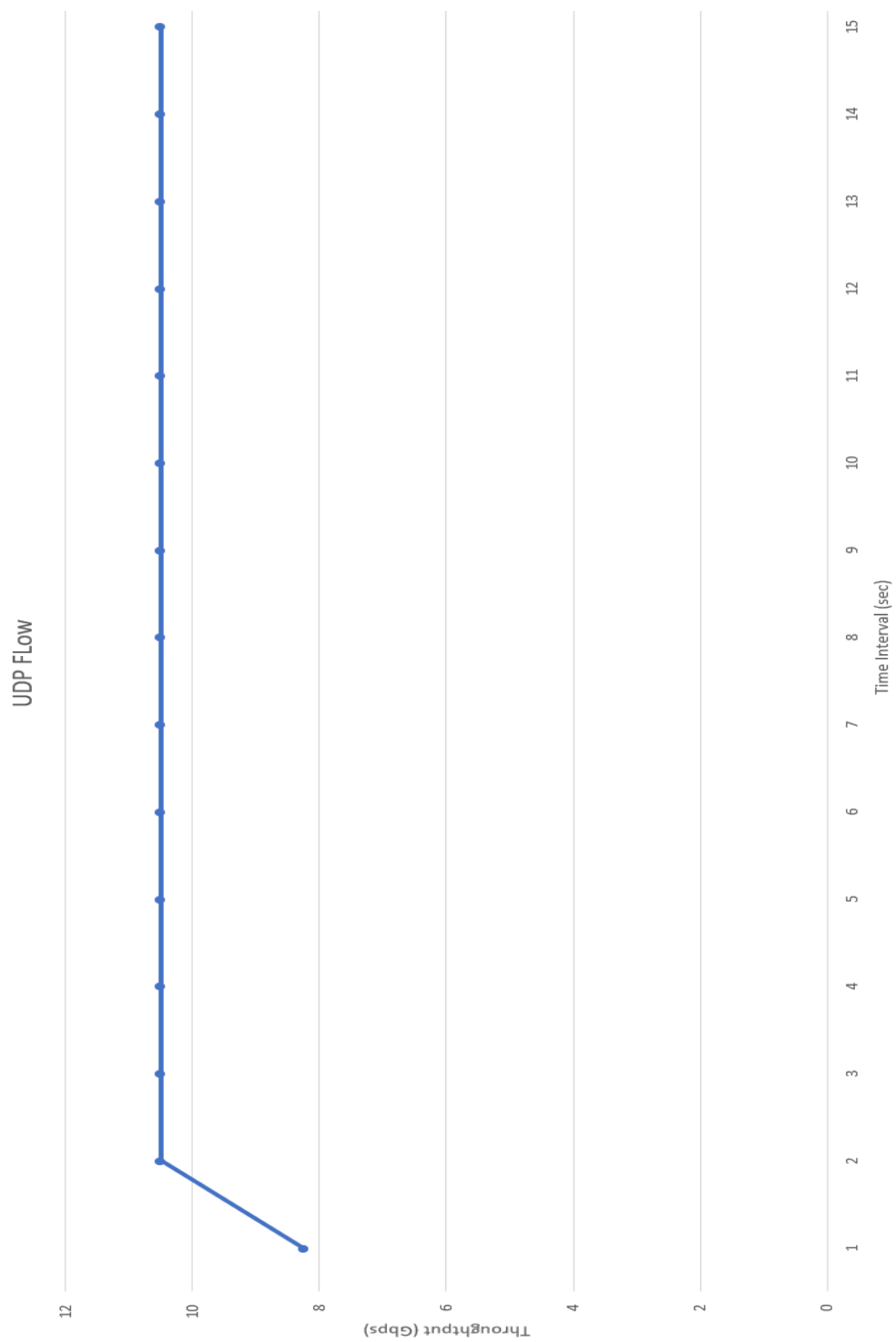


Figure 4.6 UDP flow from h1 to s12

(ii) Comparison of Hop Count and Delay:

In a network, the hop count is dependent on the number of switches that a packet transmits between the source and destination. It is the distance measured in the network based on the number of networking devices (switches). In general, if the hop count is less, it cannot be stated that the packet transmission between the source and destination will be faster. Also, if the hop count is high, it might transfer packet faster via different paths. The delay is the weight of the links in the network. In this project, the delay for all the paths in the network is calculated, and originally, the delay with the least weight is the shortest path. Figure. 16 represents the comparison of hop count and delay of the network for the transmission of the packets from source to destination. The hop count with "3" are all grouped, and similarly, hop count with "5" are grouped together. The graph also represents the delay for every shortest path calculated.

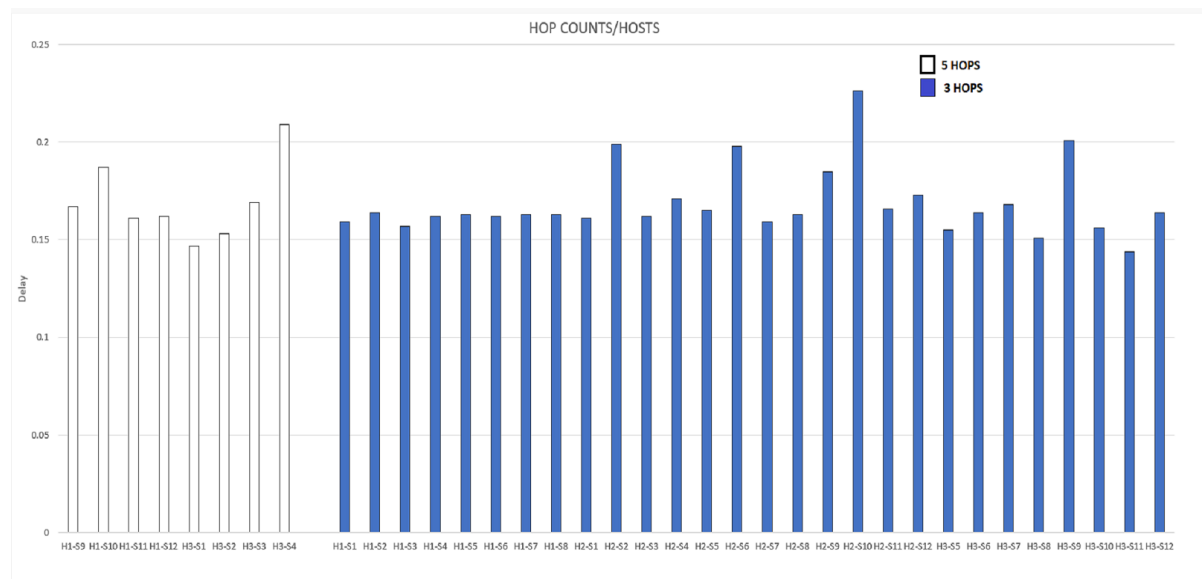


Figure 4.7 Hop count and delay

4.5 RESULT:

The bandwidth in the network is measured using the iperf tool. The controller in SDN uses RYU handlers and decorators for sending OpenFlow messages between the nodes. With the iperf tool, the TCP throughput along with UDP throughput and data loss is measured by sending and receiving TCP and UDP packets between pair of hosts. Also, the time taken by both TCP and UDP is calculated for data packets sent and received.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Today SDN has changed the traditional network into a more flexible and programmable platform that creates and supports virtualization of large networks. In SDN, though separation of data plane and control plan has improved the network performance, but due to network traffic in the control plane, the bandwidth(throughput) is reduced. Thus, in this project, a virtual simulation environment was created where path selection is made for packet transfer from the source node to the destination node. The proposed algorithm implemented for fat-tree network topology brings to the insights that the shortest path with least hop count and delay improves the network throughput.

In the future, this project can be enhanced using Layered Shortest Path Algorithm for different custom network topologies and different controllers. Also, this project is implemented on local machines considering a small network of 8 hosts and 20 switches so, in future Wide Area Network can be considered.

APPENDIX

SAMPLE CODE:

```
def myNetwork():
    net = Mininet( topo=None,
                  build=False,
                  ipBase='10.0.0.0/8', autoStaticArp=False)
    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                        controller=RemoteController,
                        ip= '127.0.0.1',
                        port=6633)
    info( '*** Add switches\n' )
    sw1 = net.addSwitch('sw1', cls=OVSKernelSwitch)
    sw2 = net.addSwitch('sw2', cls=OVSKernelSwitch)
    sw3 = net.addSwitch('sw3', cls=OVSKernelSwitch)
    sw4 = net.addSwitch('sw4', cls=OVSKernelSwitch)
    sw5 = net.addSwitch('sw5', cls=OVSKernelSwitch)
    sw6 = net.addSwitch('sw6', cls=OVSKernelSwitch)
    sw7 = net.addSwitch('sw7', cls=OVSKernelSwitch)
    sw8 = net.addSwitch('sw8', cls=OVSKernelSwitch)
    sw9 = net.addSwitch('sw9', cls=OVSKernelSwitch)
    sw10 = net.addSwitch('sw10', cls=OVSKernelSwitch)
    sw11 = net.addSwitch('sw11', cls=OVSKernelSwitch)
    sw12 = net.addSwitch('sw12', cls=OVSKernelSwitch)
    sw13 = net.addSwitch('sw13', cls=OVSKernelSwitch)
    sw14 = net.addSwitch('sw14', cls=OVSKernelSwitch)
    sw15 = net.addSwitch('sw15', cls=OVSKernelSwitch)
    sw16 = net.addSwitch('sw16', cls=OVSKernelSwitch)
    info( '*** Add hosts\n' )
    h1 = net.addHost('h1', cls=Host, ip='10.1.0.1', mac='00:00:00:00:00:01', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.1.0.2', mac='00:00:00:00:00:02', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='10.1.0.3', mac='00:00:00:00:00:03', defaultRoute=None)

    s1 = net.addHost('s1', cls=Host, ip='10.0.0.1', mac='00:00:00:00:00:11', defaultRoute=None)
    s2 = net.addHost('s2', cls=Host, ip='10.0.0.2', mac='00:00:00:00:00:12', defaultRoute=None)
    s3 = net.addHost('s3', cls=Host, ip='10.0.0.3', mac='00:00:00:00:00:13', defaultRoute=None)
    s4 = net.addHost('s4', cls=Host, ip='10.0.0.4', mac='00:00:00:00:00:14', defaultRoute=None)

    s5 = net.addHost('s5', cls=Host, ip='10.0.1.1', mac='00:00:00:00:00:21', defaultRoute=None)
    s6 = net.addHost('s6', cls=Host, ip='10.0.1.2', mac='00:00:00:00:00:22', defaultRoute=None)
    s7 = net.addHost('s7', cls=Host, ip='10.0.1.3', mac='00:00:00:00:00:23', defaultRoute=None)
    s8 = net.addHost('s8', cls=Host, ip='10.0.1.4', mac='00:00:00:00:00:24', defaultRoute=None)

    s9 = net.addHost('s9', cls=Host, ip='10.0.2.1', mac='00:00:00:00:00:31', defaultRoute=None)
    s10 = net.addHost('s10', cls=Host, ip='10.0.2.2', mac='00:00:00:00:00:32', defaultRoute=None)
    s11 = net.addHost('s11', cls=Host, ip='10.0.2.3', mac='00:00:00:00:00:33', defaultRoute=None)
    s12 = net.addHost('s12', cls=Host, ip='10.0.2.4', mac='00:00:00:00:00:34', defaultRoute=None)z
```

Figure A.5.1 Topology creation code

```

net.addLink(sw3, sw13)
net.addLink(sw3, sw14)
net.addLink(sw4, sw14)
net.addLink(sw7, sw13)
net.addLink(sw7, sw14)
net.addLink(sw7, sw15)
net.addLink(sw8, sw15)
net.addLink(sw8, sw16)
net.addLink(sw11, sw14)
net.addLink(sw12, sw15)
net.addLink(sw12, sw16)

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

info( '*** Starting switches\n')
net.get('sw1').start([c0])
net.get('sw2').start([c0])
net.get('sw3').start([c0])
net.get('sw4').start([c0])
net.get('sw5').start([c0])
net.get('sw6').start([c0])
net.get('sw7').start([c0])
net.get('sw8').start([c0])
net.get('sw9').start([c0])
net.get('sw10').start([c0])
net.get('sw11').start([c0])
net.get('sw12').start([c0])
net.get('sw13').start([c0])
net.get('sw14').start([c0])
net.get('sw15').start([c0])
net.get('sw16').start([c0])

info( '*** Post configure switches and hosts\n')

CLI(net)
net.stop()

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

```

Figure A.5.2 Link code

```

def ipv4_routing(self,msg,src,dst,datapath,dpid,ofproto,parser,in_port):
    if dst in self.net:
        print("%s -> %s" % (src,dst))
        print("nodes:")
        print(self.net.nodes)
        print("edges:")
        print(self.net.edges)
        try:
            path=nx.shortest_path(self.net,src,dst)
            # install the path
            next_index=path.index(dpid)+1
            current_dpid=dpid
            current_dp=datapath
            path_len=len(path)

            return_out_port = None
            while next_index < path_len:
                if current_dp is None:
                    continue

                next_dpid=path[next_index]
                out_port=self.net[current_dpid][next_dpid]['port']
                if current_dpid == dpid:
                    return_out_port = out_port

                self.add_flow(
                    datapath=current_dp,
                    dst=dst,
                    out_port=out_port
                )

                next_index += 1
                current_dpid=next_dpid
                if current_dpid in self.dpid_to_datapath:
                    current_dp=self.dpid_to_datapath[current_dpid]
                else:
                    current_dp=None

                out_port = return_out_port
            except nx.NetworkXNoPath:
                print("No path")
                return

        else:
            out_port = ofproto.OFPP_FLOOD

    actions = [datapath.ofproto.parser.OFPActionOutput(out_port)]

def arp_handler(self,pkt,src,dst,datapath,dpid,ofproto,parser,in_port):
    arp_hdr = pkt.get_protocol(arp.arp)
    if not arp_hdr:
        return

    arp_src_ip = arp_hdr.src_ip
    arp_dst_ip = arp_hdr.dst_ip
    eth_src = src
    eth_dst = dst

    self.arp_table[arp_src_ip] = eth_src
    self.ip_to_datapath[arp_src_ip] = datapath

    # print(" ARP: %s (%s) -> %s (%s)" % (arp_src_ip, src, arp_dst_ip, dst))

    hwtype = arp_hdr.hwtype
    proto = arp_hdr.proto
    hlen = arp_hdr.hlen
    plen = arp_hdr.plen

    if arp_hdr.opcode == arp.ARP_REQUEST:
        # request
        # lookup the arp_table
        if arp_dst_ip in self.arp_table:
            actions = [parser.OFPActionOutput(in_port)]
            ARP_Reply = packet.Packet()
            eth_dst = self.arp_table[arp_dst_ip]
            # reply
            ARP_Reply.add_protocol(ethernet.ethernet(
                ethertype=0x0806,
                dst=eth_src,
                src=eth_dst))
            ARP_Reply.add_protocol(arp.arp(
                opcode=arp.ARP_REPLY,
                src_mac=eth_dst,
                src_ip=arp_dst_ip,
                dst_mac=eth_src,
                dst_ip=arp_src_ip))

            ARP_Reply.serialize()
            # send back
            out = parser.OFPPacketOut(
                datapath=datapath,
                buffer_id=ofproto.OFP_NO_BUFFER,
                in_port=ofproto.OFPP_CONTROLLER,
                actions=actions, data=ARP_Reply.data)
            datapath.send_msg(out)

```

Figure A.5.3 Arp handler and shortest path code

OUTPUT SCREENSHOT:

```

"Node: s12"
TX packets 44 bytes 2808 (2,8 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ubuntu:/home/anfsec/Desktop/sdn/ryu_spf# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85,3 KByte (default)
-----
[ 70] local 10.0.2.4 port 5001 connected with 10.1.0.1 port 36240
[ ID] Interval      Transfer      Bandwidth
[ 70] 0.0-15.0 sec  9.74 GBytes  5.57 Gbits/sec
^Croot@ubuntu:/home/anfsec/Desktop/sdn/ryu_spf# iperf -s -p 5001 -i 1 >output
^Croot@ubuntu:/home/anfsec/Desktop/sdn/ryu_spf# cat output
-----
Server listening on TCP port 5001
TCP window size: 85,3 KByte (default)
-----
[ 70] local 10.0.2.4 port 5001 connected with 10.1.0.1 port 36244
[ ID] Interval      Transfer      Bandwidth
[ 70] 0.0- 1.0 sec   654 MBytes   5.49 Gbits/sec
[ 70] 1.0- 2.0 sec   685 MBytes   5.75 Gbits/sec
[ 70] 2.0- 3.0 sec   722 MBytes   6.06 Gbits/sec
[ 70] 3.0- 4.0 sec   753 MBytes   6.32 Gbits/sec
[ 70] 4.0- 5.0 sec   753 MBytes   6.31 Gbits/sec
[ 70] 5.0- 6.0 sec   761 MBytes   6.38 Gbits/sec
[ 70] 6.0- 7.0 sec   716 MBytes   6.01 Gbits/sec
[ 70] 7.0- 8.0 sec   762 MBytes   6.39 Gbits/sec
[ 70] 8.0- 9.0 sec   732 MBytes   6.14 Gbits/sec
[ 70] 9.0-10.0 sec   735 MBytes   6.17 Gbits/sec
[ 70] 10.0-11.0 sec   761 MBytes   6.38 Gbits/sec
[ 70] 11.0-12.0 sec   751 MBytes   6.30 Gbits/sec
[ 70] 12.0-13.0 sec   749 MBytes   6.28 Gbits/sec
[ 70] 13.0-14.0 sec   757 MBytes   6.35 Gbits/sec
[ 70] 14.0-15.0 sec   764 MBytes   6.41 Gbits/sec
[ 70] 0.0-15.0 sec  10.8 GBytes  6.18 Gbits/sec
root@ubuntu:/home/anfsec/Desktop/sdn/ryu_spf# cat output | grep sec | head -15
[ 70] 0.0-15.0 sec  10.8 GBytes  6.18 Gbits/sec

```

Figure A.5.4 TCP traffic using iperf

```

root@ubuntu:/home/anfsec/Desktop/sdn/ryu_spf# iperf -c 10.0.2.4 -u -b 10M -t 15 -p 5001
-----
Client connecting to 10.0.2.4, UDP port 5001
Sending 1470 byte datagrams, IPG target: 1121.52 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[ 69] local 10.1.0.1 port 40107 connected with 10.0.2.4 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 69] 0.0-15.0 sec  18.8 MBytes  10.5 Mbits/sec
[ 69] Sent 13375 datagrams
[ 69] Server Report:
[ 69] 0.0-15.0 sec  18.8 MBytes  10.5 Mbits/sec    0.009 ms    0/13375 (0%)
root@ubuntu:/home/anfsec/Desktop/sdn/ryu_spf# █
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 69] local 10.0.2.4 port 5001 connected with 10.1.0.1 port 40107
[ ID] Interval      Transfer      Bandwidth      Jitter    Lost/Total Datagrams
[ 69] 0.0- 1.0 sec  1.25 MBytes  10.5 Mbits/sec  0.008 ms    0/ 895 (0%)
[ 69] 1.0- 2.0 sec  1.25 MBytes  10.5 Mbits/sec  0.015 ms    0/ 892 (0%)
[ 69] 2.0- 3.0 sec  1.25 MBytes  10.5 Mbits/sec  0.012 ms    0/ 891 (0%)
[ 69] 3.0- 4.0 sec  1.25 MBytes  10.5 Mbits/sec  0.014 ms    0/ 892 (0%)
[ 69] 4.0- 5.0 sec  1.25 MBytes  10.5 Mbits/sec  0.008 ms    0/ 891 (0%)
[ 69] 5.0- 6.0 sec  1.25 MBytes  10.5 Mbits/sec  0.013 ms    0/ 892 (0%)
[ 69] 6.0- 7.0 sec  1.25 MBytes  10.5 Mbits/sec  0.010 ms    0/ 892 (0%)
[ 69] 7.0- 8.0 sec  1.25 MBytes  10.5 Mbits/sec  0.005 ms    0/ 891 (0%)
[ 69] 8.0- 9.0 sec  1.25 MBytes  10.5 Mbits/sec  0.012 ms    0/ 892 (0%)
[ 69] 9.0-10.0 sec  1.25 MBytes  10.5 Mbits/sec  0.012 ms    0/ 892 (0%)
[ 69] 10.0-11.0 sec 1.25 MBytes  10.5 Mbits/sec  0.010 ms    0/ 891 (0%)
[ 69] 11.0-12.0 sec 1.25 MBytes  10.5 Mbits/sec  0.020 ms    0/ 892 (0%)
[ 69] 12.0-13.0 sec 1.25 MBytes  10.5 Mbits/sec  0.011 ms    0/ 892 (0%)
[ 69] 13.0-14.0 sec 1.25 MBytes  10.5 Mbits/sec  0.007 ms    0/ 891 (0%)
[ 69] 0.0-15.0 sec  18.8 MBytes  10.5 Mbits/sec  0.009 ms    0/13375 (0%)
root@ubuntu:/home/anfsec/Desktop/sdn/ryu_spf# █

```

Figure A.5.5 UDP traffic using iperf

REFERENCE

1. Ahmed Dawud Alani, Et al., 2019. Software defined networks challenges and future direction of research. *International Journal of Research*, 2019.
2. Nitheesh Murugan Kaliyamurthy, Swapnesh Taterh, Suresh Shanmugasundaram, Ankit Saxena, Omar Cheikhrouhou, and Hadda Ben Elhaida 2021. Software-Defined Networking: An Evolving Network Architecture Programmability and Security Perspective. *Security and Communication Networks*.
3. Rong Chai, Xizheng Yang, Chunling Du, Qianbin Chen 2021. Network cost optimization-based capacitated controller deployment for SDN. *Communication and Information Engineering*
4. Sajad Khorsandroo, Adrián Gallego Sánchez, Ali Saman Tosun, 2021. Hybrid SDN evolution: A comprehensive survey of the state-of-the-art. *NETCOM Research Group*.
5. Hamza Mokhtar, Xiaoqiang Di, Ying Zhou, Alzubair Hassan, Shafiu Musa 2021. Multiple-level threshold load balancing in distributed SDN controllers. *Software Research Centre*,
6. Modhawi Alotaibi, Amiya Nayak 2021. Linking handover delay to load balancing in SDN-based heterogeneous networks. *Computer Science and Engineering*.
7. Wei-Che Chien, Chin-Feng Lai a, Hsin-Hung Cho, Han-Chieh Chao 2018, An SDN-SFC-based service-oriented load balancing for the IoT applications. In *Department of Computer Science and Information Engineering*.
8. Yingya Guo, Huan Lu, Xia Yin, Jianping Wue, 2021 Routing optimization with path cardinality constraints in a hybrid SDN. *Institute for Network Sciences and Cyberspace*,
9. Huangfei Song, Songtao Guo, Pan Li, Guiyan Liu 2021, FCNR: Fast and Consistent Network Reconfiguration with low latency for SDN. *Laboratory of Dependable Service Computing in Cyber Physical Society*
10. Eusebi Calle, David Martínez a, Mariusz Mycek, MichałPióro, D.M., 2020. Resilient backup controller placement in distributed SDN under critical targeted attacks.