PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
Ministry of Higher Education and Scientific Research
University Mohammed Khider BISKRA
Faculty of The Exact Sciences, Natural and Life Sciences
Department of Computer Science

Report

Presented to obtain the diploma of academic master in Computer Science

Path: Network and Information and communications technology

An Adaptive Routing Approach for Software Defined Networks

By:

Guidad Aymen

Advisor:

Dr. Ayad Soheyb

Academic year: 2019/2020

Abstract

Software-Defined Networking is becoming more and more present in the network due to the complexity and difficulty to manage the traditional network. This new paradigm aims to separate the control plane and the data plane for more network programmability, serviceability, heterogeneity, and maintainability.

To improve the performance of data transmission in SDN, this project proposes an adaptive routing approach by taking advantage of the global network view of SDN, aimed at routing the data in the network with minimal costs based on machine learning technique (ANN).

We collect three network metrics from each transmission path. These metrics are bandwidth utilization, packet loss, latency. By using these three-load metrics, an Artificial Neural Network model is trained to predict the time spent (latency) for a different path and to choose one with low-latency as the data-flow transmission path. The contrast experiment results show that an adaptive routing approach proposed in this work can select a more rational transmission path for data-flow and can reduce network latency, especially when we have large network congestion.

Acknowledgements

In the name of Allah the most Merciful and Beneficent

All praises to Allah and His blessing for the completion of this project. First and foremost, I would like to sincerely thank my supervisor Dr. Ayad Soheyb for his guidance, understanding, patience and most importantly, he has provided positive encouragement and a warm spirit to finish this project. It has been a great pleasure and honor to have him as my supervisor. Also, I would like to thank my family and friends for providing me with unfailing support and encouragement throughout the process of researching and writing my project.

List of Abbreviations

SDN	software-defined networking
ANN	Artificial Neural Network
AI	Artificial Intelligence
ML	Machine learning
ONF	Open Networking Foundation
API	Application Programming Interface,
WAN	Wide Area Network
BYOD	Bring Your Own Device
VLAN	Virtual local area network
NPBs	Neutral Particle Beam
IPS	Intrusion Prevention Systems
IDS	Intrusion Detection Systems
DR	Disaster Recovery
NFV	Network functions virtualization
CPE	Customer Premises Equipment
POP	Points-Of-Presence
SP	Service Provider
OSPF	Open Shortest Path First
BGP	Border Gateway Protocol
TPU	Tensor Processing Unit
GPU	Graphics Processing Unit
RNN	Recurrent Neural Network
DPI	Deep Packet Inspection
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

HTTP	Hypertext Transfer Protocol
SPF	Shortest Path First
IP	Internet Protocol
KDN	Knowledge-defined Network
RT	Real-Time
REST API	Representational state transfer API
URL	Uniform Resource Locator
LLDP	Link Layer Discovery Protocol
MLP	Multilayer Perceptron
SSH	Secure Shell
CPU	Central Processing Unit
ARP	Address Resolution Protocol
MAC	Media Access Control
CLI	Command-Line Interface
SNMP	Simple Network Management Protocol
MIB	Management Information Base
SMI	Structure of Management Information
RTT	Round-Trip Time
IPX	Internetwork Packet Exchange
PPP	Point-to-Point Protocol
CRC	Cyclic Redundancy Check

Table of Contents

Table of Figures

Table of tables

General introduction

These days the usage of the network is growing at a faster pace, at the same time a lot of challenges are facing by the network administrator, to tackle the frequent network access by the users. The network infrastructure is growing rapidly to meet the business need, but it requires re-policing and reconfiguration of the network. But managing the underlying infrastructure becomes more complicated to handle the unprecedented network demand. The Software Defined Network (SDN), is the next-generation Internet technology, which not only solves the ossification of the Internet but also creates innovations and simplifies network management. The key idea behind SDN is a separation of the control plane from the data plane, as a result, devices in the data plane simple becomes the forwarding device and transfer all the decision-making activities in a centralized system called a controller.

With the recent advances in the field of Artificial Intelligence (AI), and especially the breakthroughs in Machine Learning (ML) and Deep Learning, we are experiencing more research interest in adopting AI as a tool for solving modern computer network problems. Computer networks are becoming increasingly complex and dynamic. AI is revolutionizing how complex computer networks and services are managed by making more informed decisions based on the vast amount of network data. AI techniques have shown great promises in many network and service management problems including cloud management, routing and traffic engineering, cybersecurity, etc. AI has also been a central component in cognitive networks and communication researches.

This project aims to propose and develop an adaptive routing approach based on Artificial Intelligence algorithms for SDN to optimize the management of network data centers according to the network behavior (Average Bandwidth, Packet loss, and latency).

The first chapter of this work opens with the explanation of the traditional network and its limitation, and the difference between data plane and control plane and further describes in detail the technology of SDN, this is followed by the principle of Open flow protocol. The chapter concludes SDN created an opportunity for solving the Traditional network's longstanding problems.

The second chapter presents machine learning and its approaches, and an overview of the neural network and its types. Also presents some research works done so far in the area of routing in SDN networks by using artificial neural networks.

The third chapter deals with the proposed system structure and explains each component in it.

The fourth chapter provides the tests and results of the proposed approach. Additionally, there is a presentation and explanation of the tools and the environment used in this work.

Chapter 1: SDN in General

1.1. Introduction

A new networking paradigm, Software-Defined Networking (SDN), promises to help to overcome the flexibility and scalability limitations of traditional networking by making use of network management centralization and by fostering automation using network programmability. Even though the idea of a programmable network is not new, SDN has recently become a hot topic in the networking community. Software-Defined Networking is already changing the way some organizations deploy and manage their networks. Microsoft, Amazon, Google, and Facebook among others, who run most of the internet traffic today, are early adopters of SDN and drivers of several SDN initiatives. They are part of the Open Networking Foundation, ONF, established in 2011 to standardize SDN architecture and protocols.

1.2. Traditional network

Conventionally, the networking worked to connecting hardware like routers and switches backed with a basic software program that used to define a configuration for all the connected devices in a network [1]. This can be observed from the following figure.

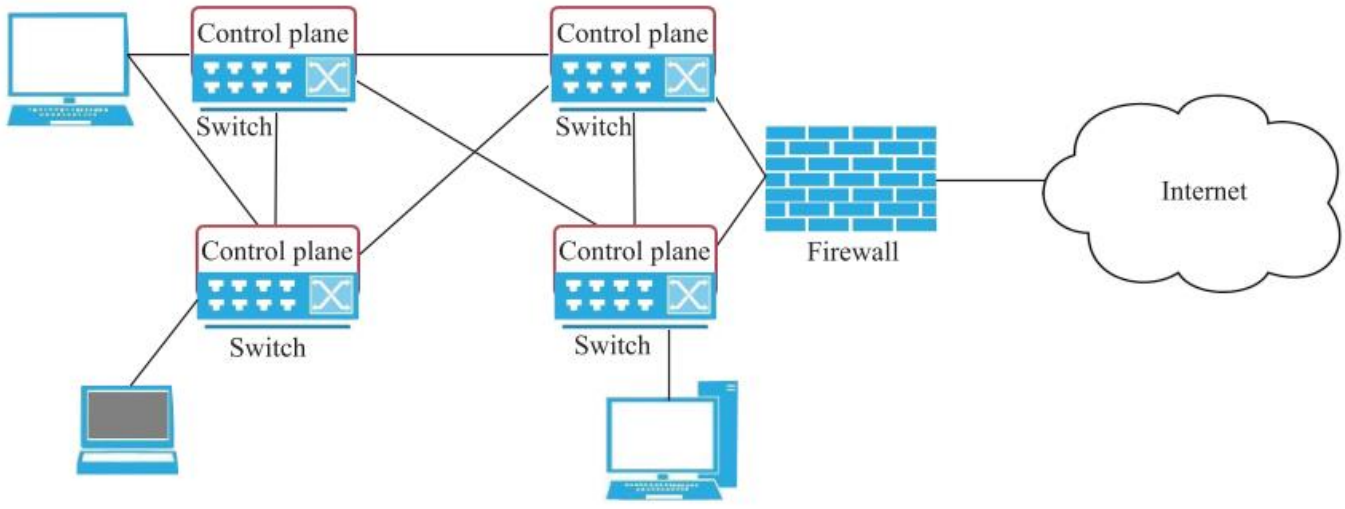

Figure 1: Traditional Network [2]

Even today traditional networks are used extensively worldwide. With some help of networking skills, a network administrator manages to architect the whole components of the network. Though the above figure looks simply, the overall size of the network goes on increasing usage and requirement over time. As the complexity of the network increases, it becomes difficult to add a program for every new device and to reconfigure the whole new system each time. This scenario becomes an extra overhead for network management.

Problems associated with traditional networking:

- Scalability
- Classification of data and routing traffic
- Time Consuming
- Multi-vendor environment requires a high level of expertise
- Decentralized network control

Thus, to address the problems associated with traditional networking, the concept of Software-defined networking was introduced.

1.3. Software-Defined Networking

The traditional computer networks are complex, difficult to manage, built from a large number of network devices. All of those drawbacks show the needed to find a new way to facilitate network evolution. In this context, it appeared the idea of "programmable network" [3]. Software-Defined Networking is one of the solutions developed. It separates the control plane (which decides how to handle the traffic) from the data plane (which forwards traffic according to decisions that the control plane makes). This characteristic expects a more simplify network management, but also enables innovation and evolution. But, SDN does not appear suddenly, it is a part of a long way of efforts to make the network more programmable.

1.3.1. History

Software-Defined Networking relies on past research on active networking and works on separating the control plane and the data plane, for example in the telephony networks, where the separation is used to simplify network management and the deployment of new services [4].

1.3.1.1. Active Networking

The first work which contributed to the current SDN is the active networking (between the mid-1990s and the early 2000s). It introduced programmable functions in the network to enable innovation. Two programming models have been proposed by the active networking community: the capsule model and the programmable router/switch model. The intellectual contribution of active networks to SDN are:

- programmable functions in the network to lower the barrier to innovation;
- network virtualization, and the ability to demultiplex to software programs based on packet headers.
- the vision of a unified architecture for middlebox orchestration.

1.3.1.2. Separating control and data planes

In the early 2000s, the idea to separate the control and data planes has been developed, and two innovations appeared: an open interface between the control and data planes, such as the ForCES (Forwarding and Control Element Separation), and logically centralized control of the network. Those two innovations have an intellectual contribution to SDN which is:

- logically centralized control using an open interface to the data plane.
- distributed state management.

1.3.1.3. OpenFlow

In the mid-2000s, a group of researchers of Stanford created OpenFlow switches [5]. To enable the creation of many new control applications, the design of controller platforms has quickly followed. The intellectual contributions are:

- generalizing network devices and functions.
- the vision of a network operating system.
- distributed state management techniques.

The term “SDN” has been first used to describe Stanford’s OpenFlow project, but now the definition is expanded to include a much wider array of technologies. All those innovations permitted the definition of a new paradigm for network architecture, called Software-Defined Networking, which refers to a network architecture where the forwarding state in the data plane is managed by a remote control plane decoupled from the former [6].

1.3.2. SDN Architecture

The basic elements of SDN include SDN devices, SDN controllers, and applications. The SDN devices contain components for deciding what to do with incoming traffic (frames or packets). The SDN controller programs the network devices and presents an abstraction of the underlying network infrastructure to the SDN applications. The controller allows an SDN application to define traffic flows and paths, in terms of common characteristics of packets, on the network devices to satisfy its needs and to respond to dynamic requirements by users and traffic/network conditions. The Open Networking Foundation defines a high-level architecture for SDN with three main layers as shown in Figure (2). [7]

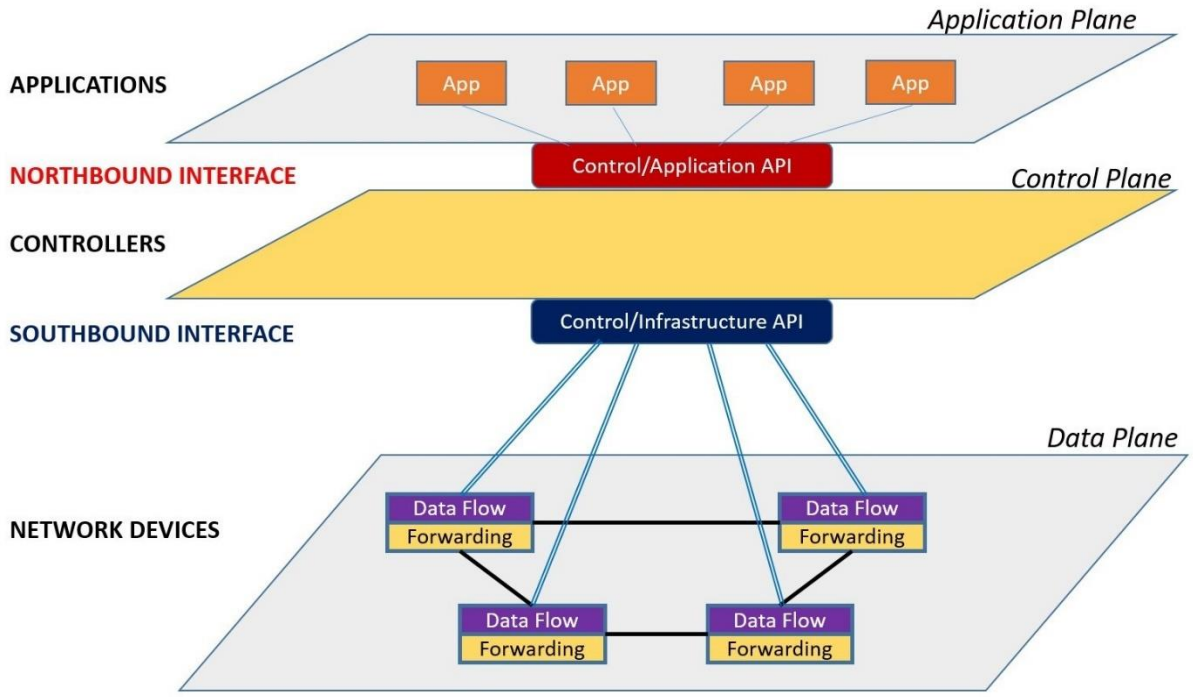

Figure 2. SDN Architecture [8]

1.3.2.1. Infrastructure Layer

This layer consists of SDN devices (both physical and virtual) that perform packet switching and forwarding. Specifically, an SDN device is composed of an application program interface (API) for communication with the controller, an abstraction layer, and a packet-processing component. The abstraction layer abstracts an SDN device as a set of flow tables. The packet processing function decides on actions to be taken, based on the results of the evaluating incoming packets relative to flow entries in the flow tables.

1.3.2.2. Control Layer

This layer provides the logically centralized control functionality that supervises the network forwarding behavior through an open interface. An SDN controller controls, through a southbound API, all SDN devices that make up the network infrastructure; and implements policy decisions such as routing, forwarding, load balancing, etc. It provides an abstract view of the entire network to the applications through a northbound interface.

1.3.2.3. Application Layer

his layer consists of end-user applications that utilize the SDN communications and network services [9]. Through the controller the applications can affect the behavior of the underlying infrastructure by configuring the flows to route packets through the best path between two endpoints, balancing traffic loads across multiple paths or destined to a set of endpoints, reacting to changes in network topology such as link failures and the addition of new devices and paths, or redirecting traffic for purposes of inspection, authentication, segregation, and similar security-related tasks.

1.3.3. Application of SDN

Recently, Google, Amazon, Facebook, Microsoft, and others have invested heavily in Software Defined Networking both in their data centers and their wide area networks (WAN), and many have published details about their homegrown SDN software and white box switch implementations.

In [10], they classified the Applications of SDN into the following domains.

- Data Centers
- Service Providers
- Campus Networks

1.3.3.1. Campus/Enterprise/Home Networks

There have been various use-cases within campus/enterprise networks. The applications such as Video Streaming and Collaboration, BYOD and Seamless Mobility, and Network Virtualization (Slicing/Traffic Isolation), Application-Aware Routing, are explored using SDN.

1.3.3.2. Data Center Networks

1.3.3.2.1. Network Virtualization

Network virtualization is one of the major applications of SDN in data-center networks. Two major scenarios of network virtualization, for which SDN is used in data-centers, are to realize multi-tenant Networks and stretched/extended networks. SDN is applied to dynamically create segregated topologically equivalent networks (multitenancy) across a data-center and to create location-agnostic networks, across racks or across data-centers, with VM mobility and dynamic reallocation of resources (stretched/extended networks). Some of the advantages of SDN in data-centers for network virtualization are better utilization of data-center resources, faster turnaround times, improved recovery times in disasters, overcome various limitations such as 4K of VLAN.

1.3.3.2.2. Tap Aggregation

Provide visibility and troubleshooting capabilities on any port in a multi-switch deployment without the use of numerous expensive network packet brokers (NPB). The advantages of using SDN are Dramatic savings and cost reduction, savings of 50-100K per 24 to 48 switches in the infrastructure. Less overhead in initial deployment, reducing the need to run extra cables from NPBs to every switch.

1.3.3.2.3. Energy Saving

A network-wide power manager that utilizes SDN to find the minimum-power network subset which satisfies current traffic conditions and turns off switches that are not needed. As a result, they show energy savings between 25-62% under varying traffic conditions. The Honeyguide approach to energy optimization in which it uses virtual machine migration to increase the number of machines and switches, that can be shutdown.

1.3.3.3. Service Provider and Transport Networks

SDN provides a fully programmatic Operator-Network interface, which allows it to address a wide variety of operator requirements without changing any of the lower-level aspects of the network. SDN achieves this flexibility by decoupling the control plane from the topology of the data plane so that the distribution model of the control plane need not mimic the distribution of the data plane. Below, we enlist a few use-cases related to service provider networks.

1.3.3.3.1. Dynamic WAN reroutes

SDN is used to provide dynamic yet authenticated programmable access to flow-level bypass using APIs to network switches and routers. The advantages of using SDN in this case: Savings of hundreds of thousands of dollars' unnecessary investment in 10Gbps or 100Gbps L4-7 firewalls, load-balancers, IPS/IDS that process unnecessary traffic.

1.3.3.3.2. Bandwidth on Demand

Enable programmatic controls on carrier links to request extra bandwidth when needed (e.g. DR, backups). The advantages of using SDN, in this case, are reduced operational expenses allowing self-service by customers and increased agility saving long periods of manual provisioning.

1.3.3.3.3. NFV and Virtual Edge

In combination with NFV initiatives, replace existing Customer Premises Equipment (CPE) at residences and businesses with lightweight versions, moving common functions and complex traffic handling into POP (points-of-presence) or SP datacenter. The advantages of using SDN are: the increased usable lifespan of on-premises equipment, improved troubleshooting, and flexibility to sell new services to business

1.4. SDN routing versus legacy routing

Legacy routing protocols such as OSPF and BGP have been developed very comprehensive, but its rigid complex system has been difficult to adapt to the fast-growing Internet. The emergence of the Software-Defined Network has brought hope for the solution of this problem. Benefit from the advantage of computation and fine-grained control for packets. However, is the performance of SDN routing better than legacy routing. [11]

- Routing in the SDN network has an advantage in a large-scale network topology.
- Routing convergence in legacy networks is much more influenced by link delay.
- Routing convergence time in the SDN network is less than legacy networks.

1.5. Open Flow

OpenFlow is the most popular SDN technology. It proposes to standardize the communication between the switches and the software-based controller [5]. Even if some people consider SDN and OpenFlow as synonyms, there are different. Indeed, SDN consists of decoupling the control and the data planes, while OpenFlow describes how a software controller and a switch should communicate in an SDN architecture [12].

1.5.1. Overview

An OpenFlow architecture consists of three basic concepts, as shown in figure (3).

- the network is built up by OpenFlow-compliant switches that compose the data plane.
- the control plane consists of one or more OpenFlow controllers.
- a secure control channel connects the switches with the control plane.

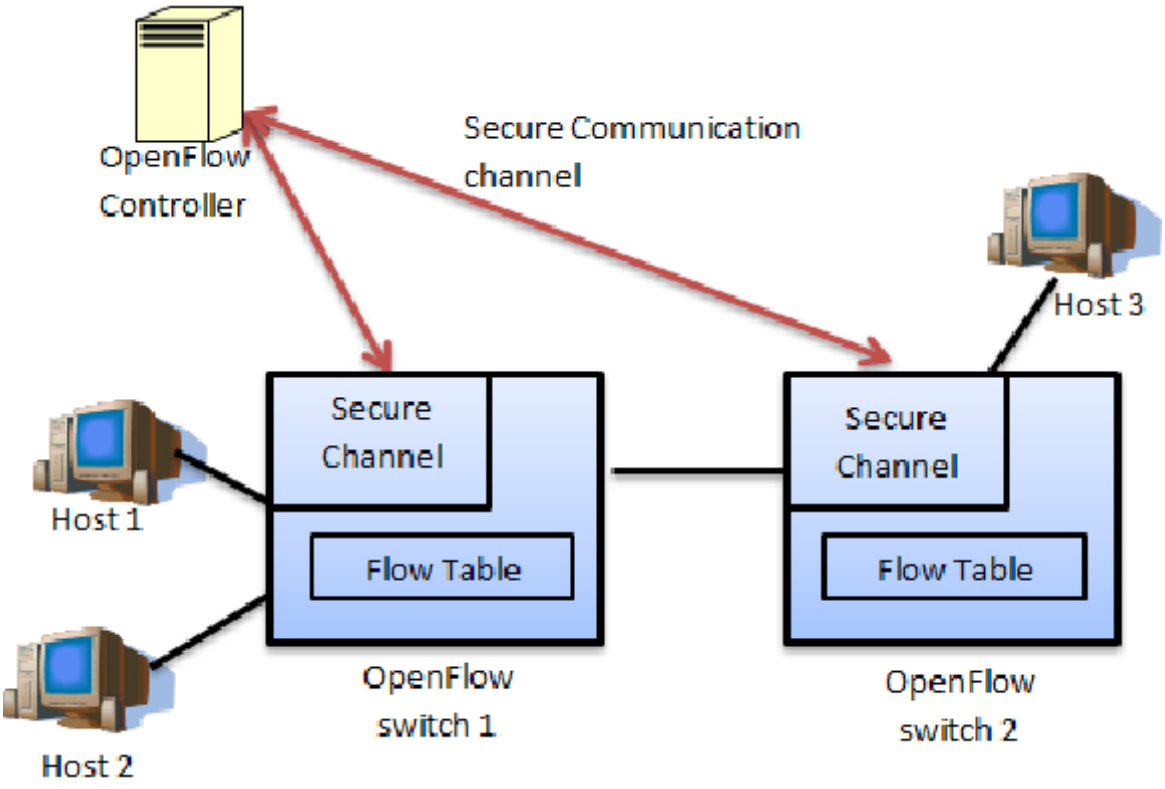

Figure 3. OpenFlow Components [13]

1.5.2. OpenFlow switches

OpenFlow switches consist of one or more flow table, a group table which perform packet lookups and forwarding, a meter table consists of meter entries, defining per-flow meters, one or more OpenFlow channel to an external controller, and port to forward flow entries. The components of an OpenFlow switch are illustrated in figure (4).

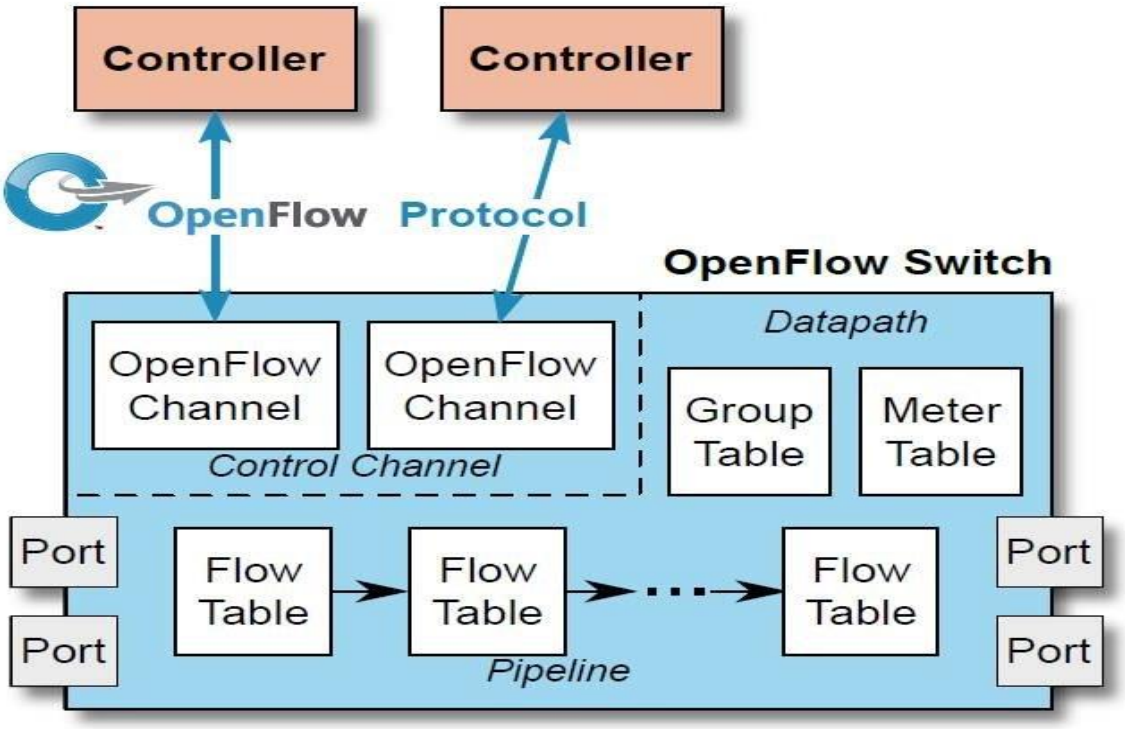

Figure 4. OpenFlow switch Components [14]

1.5.3. Flow tables

Each flow table in the switch contains a set of flow entries. In specification 1.0 [15], each of them consists of match fields, counters, and a set of instructions to apply to match packets as illustrated in figure (5). The header fields describe to which packet this entry is applicable. The counters are reserved for collecting statistics about flow. The actions specify how a packet of that flow is handled.

Header Fields	Counters	Actions

Figure 5. Flow table entry in OpenFlow 1.0 [14]

Other components have been added in the next specifications, in the figure (6), the header fields have been replaced by match fields which consist of the ingress port and packet headers, and optionally other pipeline fields such as metadata specified by a previous table. Priority is matching the precedence of the flow entry. Timeout is the maximum amount of time or idle time before a flow is expired by the switch. Cookies opaque data value chosen by the controller. And flags alter the way flow entries are managed.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Figure 6. Flow table entry in OpenFlow 1.5 [14]

1.5.4. Packet flow through an OpenSwitch

The figure (7) illustrates the packet processing in the OpenFlow pipeline. This processes in two stages, ingress processing and egress processing, which can be optional. The process always starts with the ingress processing at the first flow table. The packet is matched against the consecutive flow table from each of which the highest-priority matching flow table entry is selected. If a flow entry is found, the set of instructions of that flow entry executed. Otherwise, if there is a table miss, its instruction is executed, or the packet is dropped.

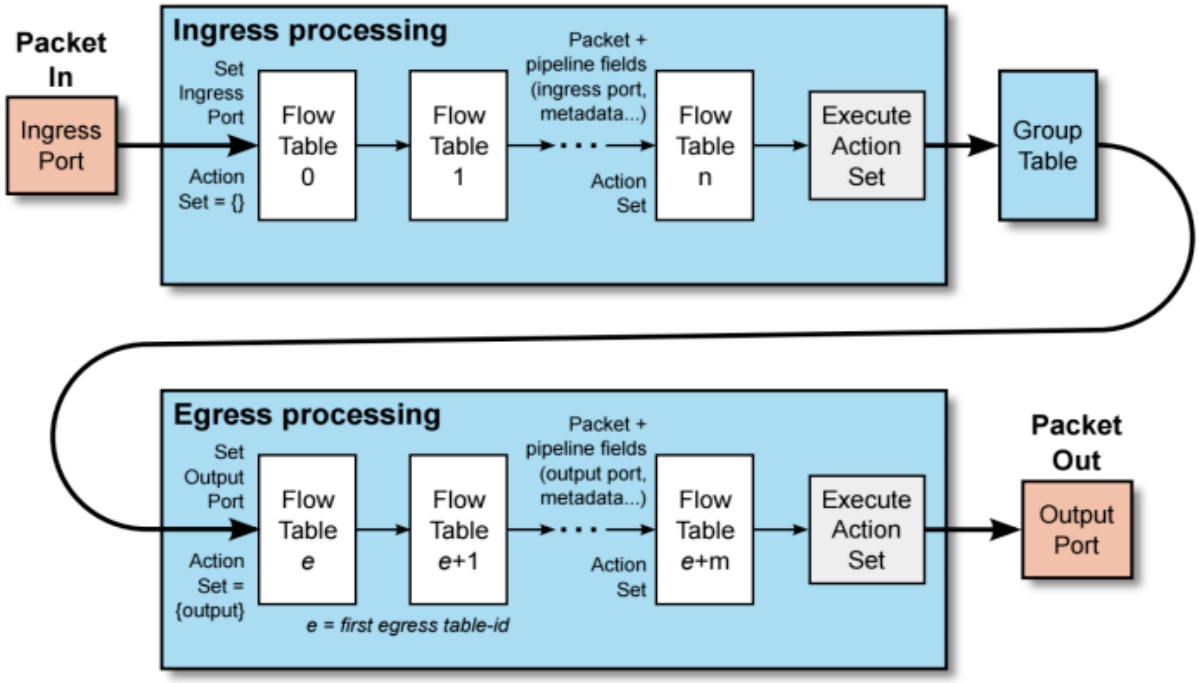

Figure 7. Packet flow through the processing pipeline [14]

Figure (8) summarizes the packet process through an OpenFlow switch. The process following by the packet is:

- the switch starts by performing a table lookup in the first flow table, and based on the pipeline processing, may perform table lookups in other flow tables.
- packet header fields are extracted and packet pipeline fields are retrieved.
- packet matches a flow entry if all the match fields of the flow entry are matching the corresponding header fields and pipeline fields from the packet

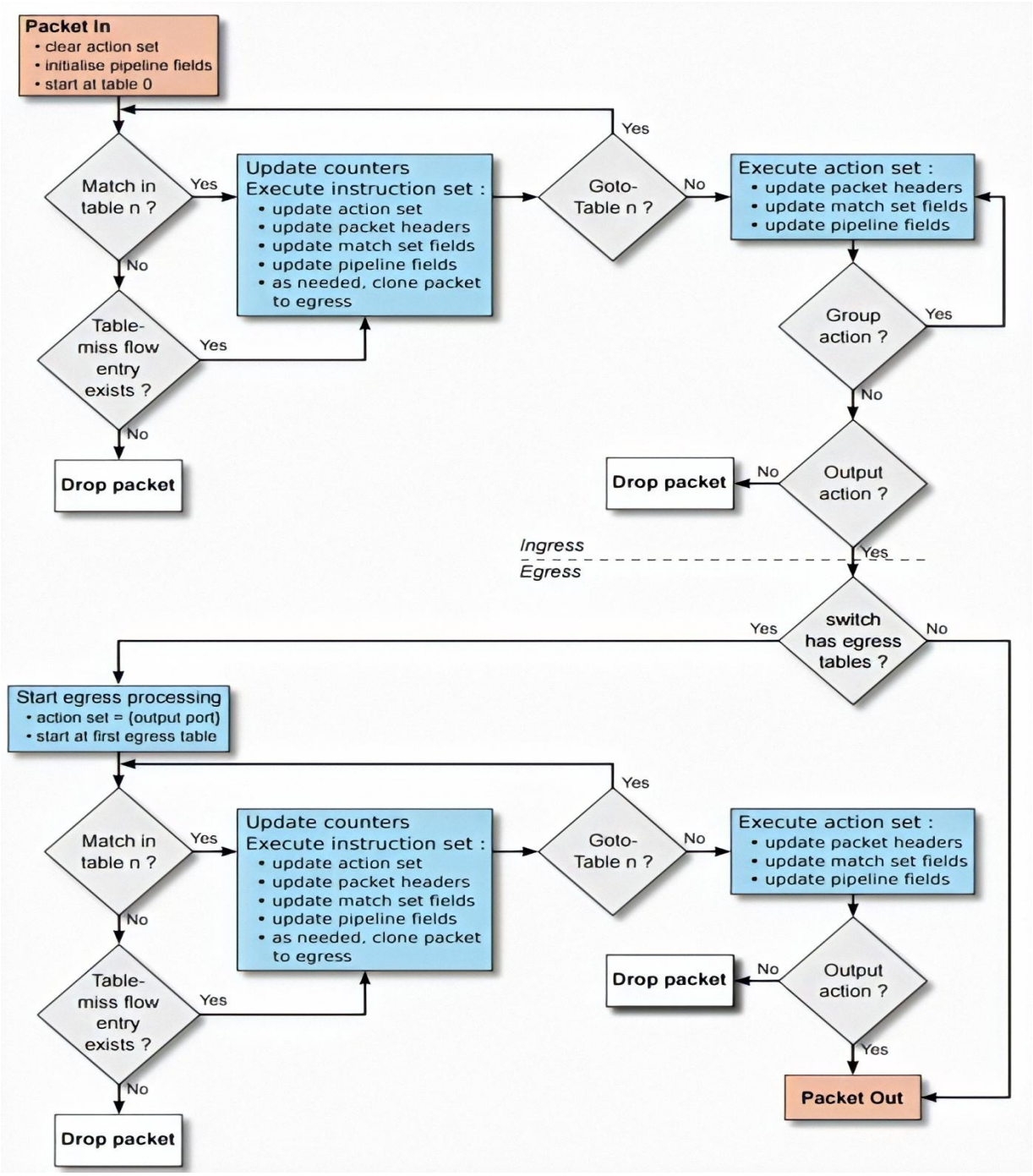

Figure 8. Simplified flowchart detailing packet flow through an OpenFlow switch [14]

1.5.5. OpenFlow messages

The controller configures and manages switches, receives events from them, and sends a packet out through the OpenFlow channel, an interface that connects OpenFlow switch and OpenFlow controller. The OpenFlow switch protocol supports three messages types.

- **Controller-to-switch**

Controller-to-switch messages are initiated by the controller and used directly to manage or inspect the state of the switch. A response from the switch may not be required. Those messages are:

- features: identity and basic capabilities of a switch.
- configuration: to set and query configuration parameters in the switch.
- modify-state: to manage state on the switch.
- read-state: to collect various information from the switch.
- packet-out: to send packets out of a specified port on the switch.
- barrier: they are request/reply messages use to ensure message dependencies have been met or to receive notifications for completed operation.
- role-request: to set the role of the OpenFlow channel, set the Controller ID, or query them.
- asynchronous-configuration: to set an additional filter on the asynchronous messages.

- **Asynchronous**

Asynchronous messages are sent from the switch without a controller soliciting. Those messages informing the controller are:

- packet-in: transfer the control of a packet.
- flow-removed: removal of a flow entry from a flow table.
- Port-status : change on a port.
- role-status: change of the role of the controller.
- controller-status: the status of OpenFlow channel changes.

- **Symmetric**

Symmetric messages are sent without any solicitation from the controller and switch. They are:

- Hello: messages exchanged between the switch and controller upon connection startup.
- Echo: request/reply messages to verify the liveness of a controller switch connection, and as well can be used to measure its latency or bandwidth.
- error: to notify a problem to the other side of the connection.
- experimenter: provide a standard way for OpenFlow switches to offer additional functionality within the OpenFlow message type space.

Figure (9) summarizes the main messages used to measure the delay between the switch and the controller.

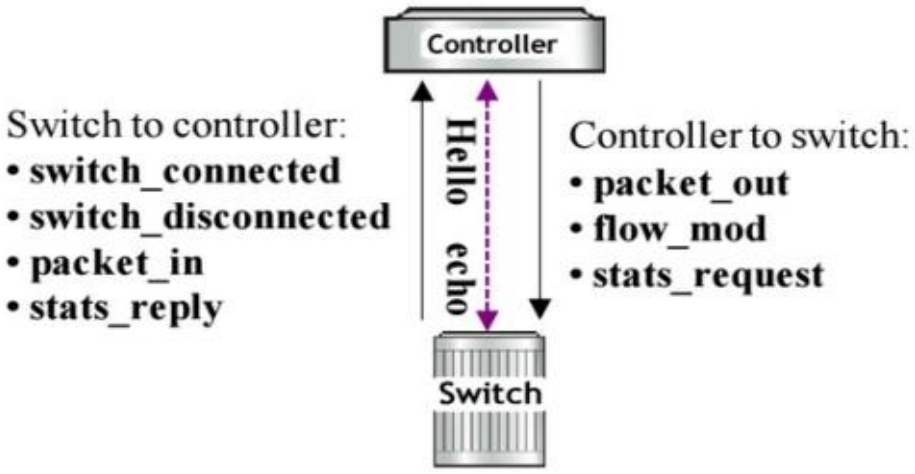

Figure 9. Types of OpenFlow messages [14]

1.6. Conclusion

In this chapter, we gave an overview of SDN architecture and compared it to traditional networks. We outlined its strength and challenges. We have also given a comparison of SDN routing and legacy routing, and we have given some applications of SDN. Then, we presented an overview on OpenFlow protocol and its architecture and components. In the next chapter, we present an overview on Machine learning and some related works done so far.

Chapter 2: Machine Learning and SDN

2.1. Introduction

SDN decouples the control plane and the data plane. The network resources in SDN are managed by a logically centralized controller, which acts as the Networking Operating System (NOS). The SDN controller can program the network dynamically. Furthermore, the centralized controller has a global view of the network by monitoring and collecting the real-time network state and configuration data, as well as packet and flow-granularity information. Applying machine learning techniques in SDN is suitable and efficient for the following reasons. First, recent advances in computing technologies such as Graphics Processing Unit (GPU) and Tensor Processing Unit (TPU) provide a good opportunity to apply promising machine learning techniques (e.g., deep neural networks) in the network field [16], [17]. Second, data is the key to the data-driven machine learning algorithms. The centralized SDN controller has a global network view and can collect various network data, which facilitate the applications of machine learning algorithms. Third, based on the real-time and historical network data, machine learning techniques can bring intelligence to the SDN controller by performing data analysis, network optimization, and automated provision of network services. Finally, the programmability of SDN enables the optimal network solutions (e.g., configuration and resource allocation) made by machine learning algorithms that can be executed on the network in real-time [18].

2.1. Machine learning

Machine Learning (ML) is a field of computer science, the objective of which is to study and develop algorithms that can “learn”. The “learning” concept refers to the ability of these algorithms to generalize different behaviors by only using information from training examples. In traditional software, the information needed to generalize this behavior is hard-coded in the program, whereas the code of ML algorithms defines the ability to learn, which can be used to generalize many behaviors. In other words, the main difference between traditional software and machine learning approaches is that, in machine learning algorithms, the output of the execution depends on the training phase of the software, therefore, the same algorithm can produce different outputs depending on the training data used.

Machine Learning techniques are used in a wide range of applications: image processing, voice recognition, search engines, intelligent personal assistants, self-driving cars, video games, However, there are few applications on the networking field.

The training is the process in which the model learns the best parameters to minimize the error. To improve the generalization capabilities of the model, usually, the dataset is divided into three sets: the training set, the validation set, and the test set. The training set and the validation set are used in the training phase: the training set is used by the learning algorithm to learn the best model parameters, whereas the validation set is used to explore different ML configuration parameters and to choose the optimal. Finally, the test dataset is used to give an independent metric of the performance of the model.

2.1.1. Machine Learning Approaches

2.1.1.1. Unsupervised learning

In this learning approach, the model is training by observing new data and extracting patterns in the data without being instructed on what they are. Opposed to supervised learning described below, the advantage of this approach is that the model can learn from data without supervision (as the name suggests). This means that there is no need for input data to be annotated, therefore it takes much less time and resources to deploy these models in practice.

The biggest hurdle of the supervised learning approach in real-world applications is to obtain appropriate data. Appropriate data in this context means, data that were somehow classified into different categories, which can be a very tedious and slow process. In some cases, the task itself prevents the usage of labeled data (i.e. labeled data are impossible to obtain or don't exist at all).

The majority of unsupervised learning algorithms belong to a group called clustering algorithms. These algorithms are centered on the idea to analyze geometric clustering of data in input space to determine their affiliation. This is achieved by the presupposition that data point clustering in input space is likely to exhibit similar properties. [19]

2.1.1.2. Supervised learning

The supervised learning approach is more commonly used. This approach requires training data in a specific format. Each instance has to have an assigned label. These labels provide supervision for the learning algorithm. The training process of supervised learning is based on the following principle. Firstly, the training data are fed into the model to produce a prediction of output. This prediction is compared to the assigned label of the training data to estimate model error. Based on this error the learning algorithm adjusts the model's parameters to reduce it. [19]

2.1.1.3. Reinforcement learning

In reinforcement learning, the machine learning algorithm interacts with an environment and must reach a certain goal. This could be to learn to play a game or to drive a vehicle in a simulation. The algorithm gets only information on how good or bad he has interacted with the environment. For example, in learning to play a game, could this information the winning or the losing of a game. [20]

2.1.2. Artificial Neural Networks

2.1.2.1. Structure

Artificial Neural Networks (ANNs) are an ML model inspired by the behavior of biological neurons and their interconnections. Each unit or neuron performs a simple action based on multiple input signals to produce a single output. The interconnection of many neurons is known as the network. Biologic neural networks do not follow any particular structure, but ANNs usually follow a layer-based structure in which the first layer is known as the input layer, the last layer is known as the output layer and the layers in between are known as a hidden layer. When there is more than one hidden layer, they are also referred to as Deep Learning.

Artificial Neural Networks (ANNs) were first proposed in the middle of the 20th century as a biologically inspired learning algorithm. In 1975, the backpropagation algorithm was first introduced, which made it possible to train multiple layers' networks [21]. However, the main revolution in the ANN field was in the 21st century, in which the advances in computing and GPU made it possible to train a complex network in a reasonable time.

Artificial Neural Networks (ANNs) can model complex non-linear systems, and for this reason, is one of the most used ML technique. However, the main trouble when using Artificial Neural Networks (ANNs) is the huge quantity of hyperparameters to choose from, to be able to learn efficiently. The parameters of the ANN are the values of the models that are used to compute the output, the hyper-parameters are the parameters that change the learning process and the fitting capabilities, but that is not part of the resulting model.

Mathematically, the operations performed in an ANN can be recursively defined by the equation (2.1), where i is the neuron:

$$y_i = f_i\left(\sum_{\forall j} W_{i,j}\ x_i + \ b_i\right) \tag{2.1}$$

Argument $(\sum_{\forall j} W_{i,j}\, x_i + b_i)$ of function f is often regarded as z. Therefore, the equation can be rewritten as

$$y_i = f_i(z) \tag{2.2}$$

Typical schema is shown in Figure (10), which depicts inputs, weights, bias, and activation function.

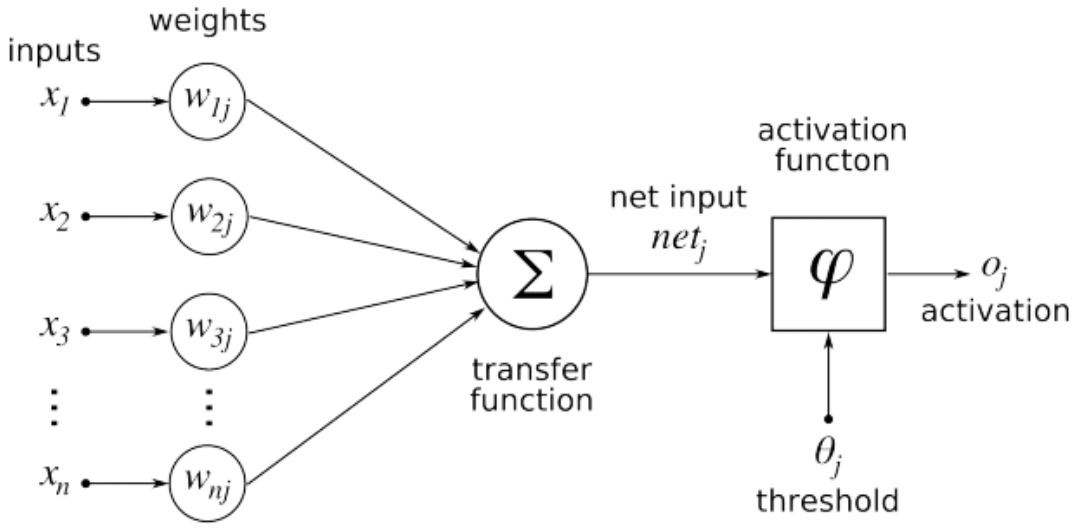

Figure 10. Model of the artificial neuron [22]

Inputs

Each neuron has multiple inputs x_i that are combined to execute some operation. Each input has designated weight assigned to it.

Weights

Inputs of a neuron are weighted by parameters $W_{i,j}$ that are modified during the learning process. Each weight gives strength to each input into the neuron. The basic idea is that when the weight is small the particular input doesn't influence the output of the neuron very much. Its influence is large in the opposite case.

Bias

Another modifiable parameter is bias b_i that controls the influence of the neuron as a whole.

Activation Function

This is done with activation function $f_i(z)$. There are several different commonly used activation functions. Its usage depends on the type of network and also on the type of layer in which they operate. [19]

One of the oldest and historically most commonly used activation functions is the sigmoid function. It is defined by

$$f_i(z) = \frac{1}{1+e^{-z}} \tag{2.3}$$

Another activation function is the hyperbolic tangent. It is defined as

$$f_i(z) = \tanh(-z) \tag{2.4}$$

The hyperbolic tangent function is less common in feed-forward NN, but it is largely used in RNN.

Currently most frequently used activation function is Restricted Linear Unit (ReLU). It is very commonly used in both convolutional and fully connected layers. It is defined by

$$f_i(z) = \mathrm{Max}(0, z) \tag{2.5}$$

It has a drawback because it is not differentiable for $z = 0$, but it is not a problem in software implementation and one of its biggest advantages is that it can learn very quickly. All three activation functions are illustrated in Figure (11).

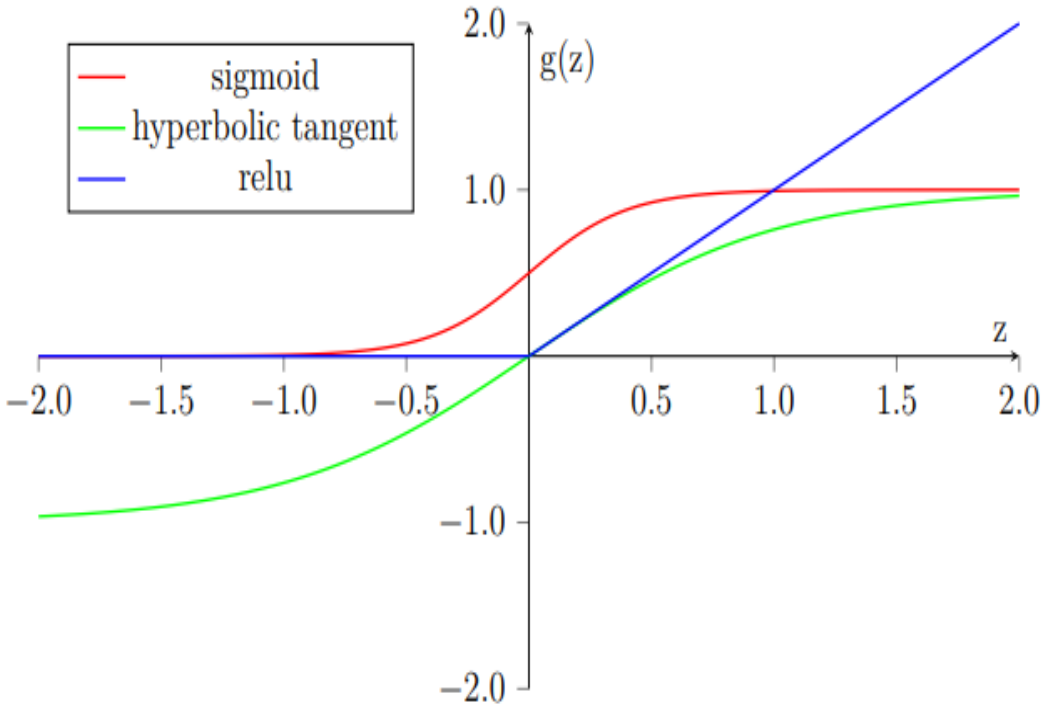

Figure 11. Activation Functions [19]

2.1.2.2. Type of Neural Network

2.1.2.2.1. Single-layer feed-forward network

A neural network in which the input layer of source nodes projects into an output layer of neurons but not vice-versa is known as a single feed-forward or acyclic network. In the single-layer network, ‘single-layer’ refers to the output layer of computation nodes as shown in Figure (12). [23]

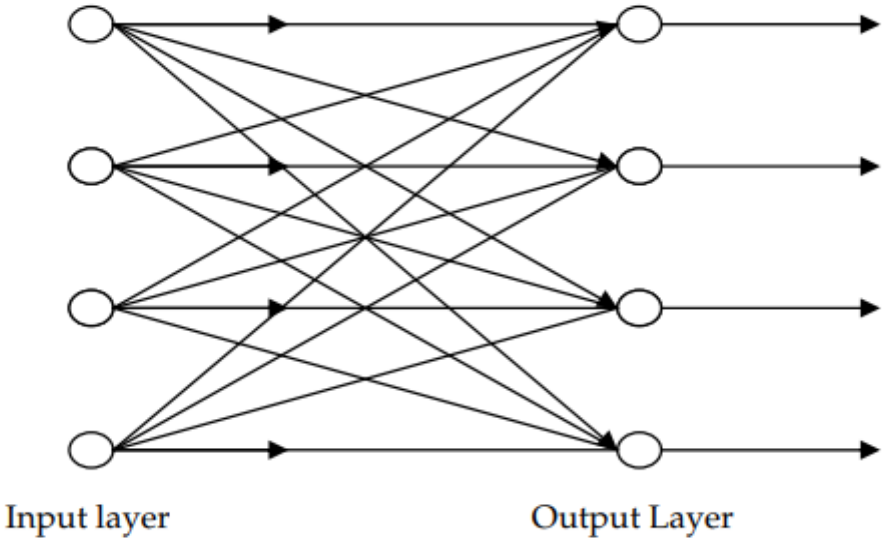

Figure 12. A Single layer feedforward network

2.1.2.2.2. Multilayer feed-forward network

This type of network (Figure 13) consists of one or more hidden layers, whose computation nodes are called hidden neurons or hidden units. The function of hidden neurons is to interact between the external input and network output in some useful manner and to extract higher-order statistics. The source nodes in the input layer of the network supply the input signal to neurons in the second layer (1st hidden layer). The output signals of the 2nd layer are used as inputs to the third layer and so on. The set of output signals of the neurons in the output layer of the network constitutes Figure 3: A Single layer feedforward network Input layer Output Layer the overall response of the network to the activation pattern supplied by source nodes in the input first layer. [23]

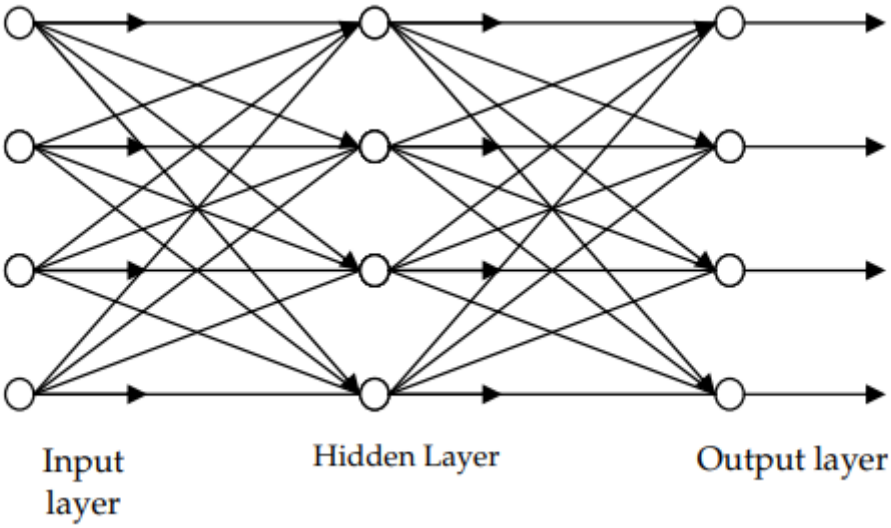

Figure 13. A multilayer feed-forward network

2.1.2.2.3. Recurrent Network

A feed-forward neural network having one or more hidden layers with at least one feedback loop is known as a recurrent network as shown in Figure (14). The feedback may be a self-feedback, i.e., where the output of a neuron is fed back to its input. Sometimes, feedback loops involve the use of unit delay elements, which results in nonlinear dynamic behavior, assuming that neural network contains nonlinear units. [23]

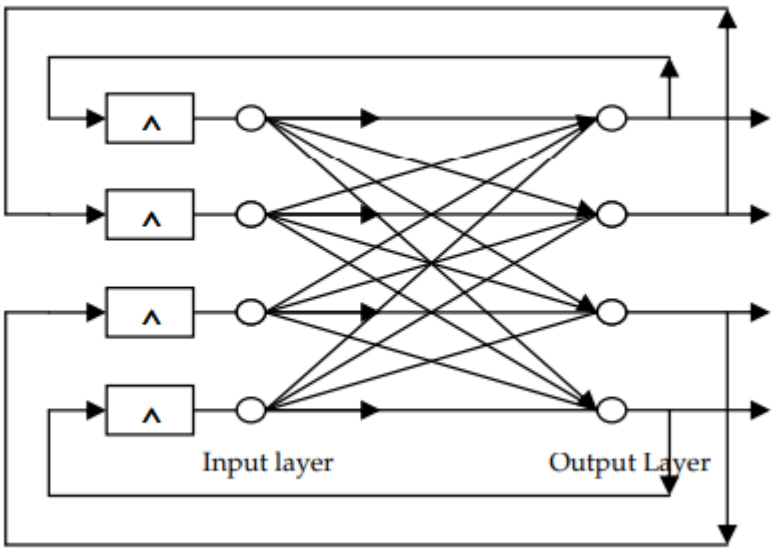

Figure 14. A recurrent network

2.2. Machine learning in SDN

The centralized SDN controller has a global network view, which makes the network easy to control and manage. Machine learning techniques can bring intelligence to the SDN controller by performing data analysis, network optimization, and automated provision of network services. In other words, the learning capability enables the SDN controller to autonomously learn to make optimal decisions to adapt to the network environments. In this subsection, we review some existing machine learning efforts to address issues in SDN, such as traffic classification and routing optimization.

2.2.1. Traffic Classification

Traffic classification is an important network function, which provides a way to perform fine-grained network management by identifying different traffic flow types. With the help of traffic classification, network operators can handle different services and allocate network resources more efficiently.

The widely-used traffic classification techniques include a port-based approach, Deep Packet Inspection (DPI), and machine learning [24] [25] [26]. The port-based approach uses TCP and UDP port numbers to determine applications. In the past, many applications used well-known ports such as TCP port 80 for HTTP protocol. Nowadays, most applications run on dynamic ports, which makes the port-based approach no longer effective.

DPI matches the payload of traffic flows with predefined patterns to identify the applications that traffic flows belong to. The patterns are defined by regular expressions. The DPI-based approach generally has high classification accuracy. However, it has some shortcomings. First, DPI can only recognize applications whose patterns are available. The exponential growth of applications makes the pattern update difficult and impractical. Second, DPI incurs high computational cost as all traffic flows need to be checked. Third, DPI cannot classify encrypted traffic on the Internet.

ML-based approaches can correctly recognize encrypted traffic and incur much lower computational cost than the DPI-based approach. Thus, ML-based approaches have been extensively studied. To do traffic classification, a large number of traffic flows are first collected, and then ML techniques are applied to extract knowledge from the collected traffic flows. In SDN, the controller has a global network view, which facilitates traffic collection and analysis. Thus, the ML-based approaches are generally implemented in the controller. Many studies have been done to classify traffic from different perspectives, such as elephant flow-aware [27], application-aware, and QoS-aware traffic classification [28].

2.2.2. Routing Optimization

Routing is a fundamental network function. In SDN, the controller can control the routing of traffic flows by modifying flow tables in switches. For example, the controller can guide switches to discard a traffic flow or route it through a specific path. Inefficient routing decisions can lead to the overloading of network links and increase the end-to-end transmission delay, which affects the overall performance of SDN. Thus, how to optimize the routing of traffic flows is an important research problem.

Shortest Path First (SPF) algorithm and heuristic algorithms [29] are two types of widely-used routing optimization approaches. SPF algorithm routes packets according to simple criteria such as hop-count or delay. Despite its simplicity, the SPF algorithm is a best-effort routing protocol and does not make the best use of network resources [30]. Heuristic algorithms (e.g., ant colony optimization algorithm) are another approach to solve the routing optimization problem. The high computational complexity is the main shortcoming of heuristic algorithms [30], [31].

In SDN, the controller is responsible for calculating the routing policy for each new flow. In this case, heuristic algorithms are not suitable because they increase the computational burden of the controller. Many studies have tried to solve the routing optimization problem using machine learning algorithms. Compared with heuristic algorithms, machine learning algorithms have some advantages. On one hand, once trained, machine learning algorithms can give the near-optimal routing solutions quickly. On the other hand, machine learning algorithms do not need an exact mathematical model of the underlying network. The routing optimization problem can be considered as a decision-making task. Thus, reinforcement learning is an effective approach. Supervised learning algorithms are also applied by many studies to optimize routing.

2.3. Network Performance Metrics

2.3.1. Throughput

throughput is the rate at which the sending process can deliver bits to the receiving process. Because other sessions will be sharing the bandwidth along the network path, and because these other sessions will be coming and going, the available throughput can fluctuate with time. [32]

2.3.2. Latency

In computer networking, latency is an expression of how much time it takes for a data packet to travel from one designated point to another. [33]

2.3.3. Delay

packet starts in a host (the source), passes through a series of routers, and ends its journey in another host (the destination). As a packet travels from one node (host or router) to the subsequent node (host or router) along this path, the packet suffers from several types of delays at each node along the path. The most important of these delays are the nodal processing delay, queuing delay, transmission delay, and propagation delay; together, these delays accumulate to give a total nodal delay. The performance of many Internet applications—such as search, Web browsing, e-mail, maps, instant messaging, and voice-over-IP—are greatly affected by network delays. [32]

Processing Delay

The time required to examine the packet's header and determine where to direct the packet is part of the processing delay. The processing delay can also include other factors, such as the time needed to check for bit-level errors.

Queuing Delay

At the queue, the packet experiences a queuing delay as it waits to be transmitted onto the link. The length of the queuing delay of a specific packet will depend on the number of earlier-arriving packets that are queued and waiting for transmission onto the link. If the queue is empty and no other packet is currently being transmitted, then our packet's queuing delay will be zero. On the other hand, if the traffic is heavy and many other packets are also waiting to be transmitted, the queuing delay will be long.

Transmission Delay

Assuming that packets are transmitted in a first-come-first-served manner, as is common in packet-switched networks, our packet can be transmitted only after all the packets that have arrived before it has been transmitted. Denote the length of the packet by L bits, and denote the transmission rate of the link from router A to router B by R bits/sec. For example, for a 10 Mbps Ethernet link, the rate is R = 10 Mbps; for a 100 Mbps Ethernet link, the rate is $R = 100$ Mbps. The transmission delay is L/R. This is the amount of time required to push (that is, transmit) all of the packet's bits into the link.

Propagation Delay

Once a bit is pushed into the link, it needs to propagate to the destination. The time required to propagate from the beginning of the link to the destination is the propagation delay.

2.3.4. Jitter

A crucial component of delay is the varying queuing delays that a packet experiences in the network's routers. Because of these varying delays, the time from when a packet is generated at the source until it is received at the receiver can fluctuate from packet to packet. This phenomenon is called jitter. [32]

2.3.5. Packet loss

Each packet switch has multiple links attached to it. For each attached link, the packet switch has an output buffer (also called an output queue), which stores packets that the router is about to send into that link. The output buffers play a key role in packet switching. If an arriving packet needs to be transmitted onto a link but finds the link busy with the transmission of another packet, the arriving packet must wait in the output buffer. Thus, in addition to the store-and-forward delays, packets suffer output buffer queuing delays. These delays are variable and depend on the level of congestion in the network. Since the amount of buffer space is finite, an arriving packet may find that the buffer is completely full with other packets waiting for transmission. In this case, packet loss will occur—either the arriving packet or one of the already-queued packets will be dropped. [32]

2.3.6. Bandwidth utilization

Bandwidth is the maximum data transmission rate possible on a network. Bandwidth utilization is an important factor to improve network performance refers to how much bandwidth is currently being used on a network.

2.4. Related work

With the recent advances in the field of Artificial Intelligence, we are experiencing more research interest in adopting AI as a tool for solving modern computer network problems, especially in SDN.

A load balance (Adaptive) solution scheme proposed in [34]. By taking advantage of the global network view of SDN, four load features are collected from each transmission path. These features are bandwidth utilization ratio, packet loss rate, transmission latency, and transmission hops. By using these four load features, an Artificial Neural Network model is trained to predict the integrated load of each path.

Some operations in this method could have been included in the same neural network model, for example when we have N paths from source to destination, their neural network will run N times to predict a load of each path separately, then take the lowest load path as a selected transmission path, while it was possible to perform these operations only once.

KDN (Knowledge-defined) approach based on a load balancing method proposed in [35] using performance metrics (bandwidth and latency) of each path of the network. Applying an Artificial Neural Network (ANN) to model the system behavior, this model takes bandwidth of all links existing in the topology as inputs, and outputs are paths latency, after the training their model can predict paths latency from given bandwidth links.

Since this model takes all the links as inputs, this means that when multiple requests arrive at the same time to find the transmission path, their model gives us the same result for different connections because the inputs are similar.

Implementation of a neural network inside the OpenFlow switch as an internal controller proposed in [36]. this NN model takes packet loss and available bandwidth as inputs, after that it can predict the transmission path. this method eliminates one of the aims of the SDN controller since switch can route alone, also doesn't take into account the number of transmission hops.

The following table represents a comparison between the neural network models used in our related works, in each one we refer to the input and output features used in it.

	Inputs	Outputs
Research on Load Balance Method in SDN [34]	bandwidth utilization ratio, packet loss rate, transmission latency, and transmission hops for the path.	Integrated load path.

A Load Balancing Method Based on Artificial Neural Networks for Knowledge-defined Data Center Networking [35]	the available bandwidth of all existing links.	paths latency.
Implementation of Neural Switch using OpenFlow as Load Balancing Method in Data Center [36]	available bandwidth, packet loss.	Transmission path

Table 1. related work's neural network model

Our proposed NN model takes the advantages of [37] (multiple metrics as inputs because using different metrics may cause different accuracy and rationality) and [38] (paths latency as outputs, so we don't need a self-learning approach because we have a metric by which we can figure out the ideal path). we collect 3 metrics, which referred to bandwidth utilization, packet loss, latency. this model can predict paths latency by the inputs (bandwidth utilization and packet loss for the links that make up these output paths) to choose the least loaded path. we will explain it in the next chapter in detail.

2.5. Conclusion

In this chapter, we presented a review on machine learning and its approaches, and we gave an overview of the neural network and its types. We have also given reviews on research works done so far in the area of routing in SDN networks by using artificial neural networks. In the next chapter, we present our proposed approach for adaptive routing in SDN environments.

Chapter 3: Adaptive Routing Approach For Software Defined Networks

3.1. Introduction

Path redundancy technology in SDN effectively provides the robustness and stability for the network. But how to evenly distribute traffic among multiple paths has become an urgent problem for SDN researchers. In this work, we have proposed and implemented an adaptive routing for SDN using AI. Adaptive routing is a method that will establish an optimal routing path for data packets depending on the real-time status of the network. With the help of SDN architecture, the implementation of adaptive routing is much easier. Due to the openness of SDN, the determination of optimal routing paths and directing packets through these optimal paths is possible. Adaptive routing can be implemented according to many parameters such as bandwidth, packet loss, latency, and other networking constraints to provide an optimal path. Adaptive routing can help the network to achieve desirable QoS parameters.

3.2. General Architecture

The general architecture of our proposed approach is illustrated in figure (15), where we see that it consists of three principal components which are:

The centralized controller: it manages the network topology by linking devices between them, also enables the network to be intelligently and centrally controlled, or programmed using software applications.

The network topology: is a physical description of the total resources in a network, this topology contains switches and hosts that are connected to each other and managed by the controller.

Decision-making based Machine learning module: Requested by the centralized controller to smartly route network traffic.

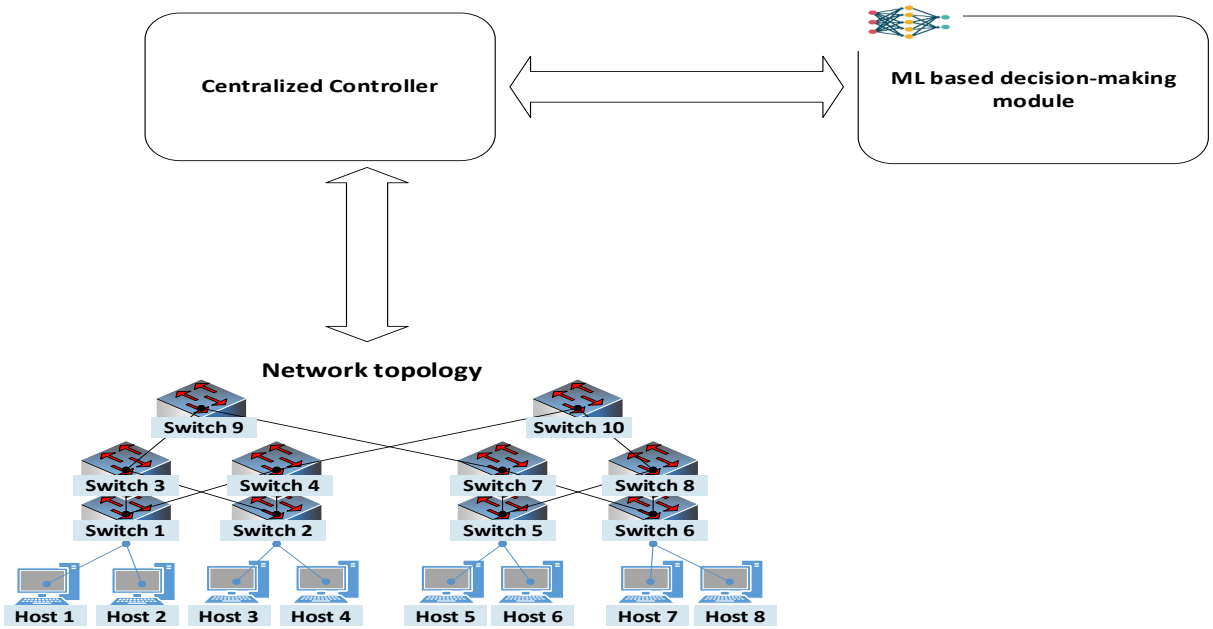

Figure 15. General Architecture

3.3. Detailed Architecture

The figure (16) shows the detailed architecture of our proposed adaptive routing approach in SDN environment using Machine Learning technics.

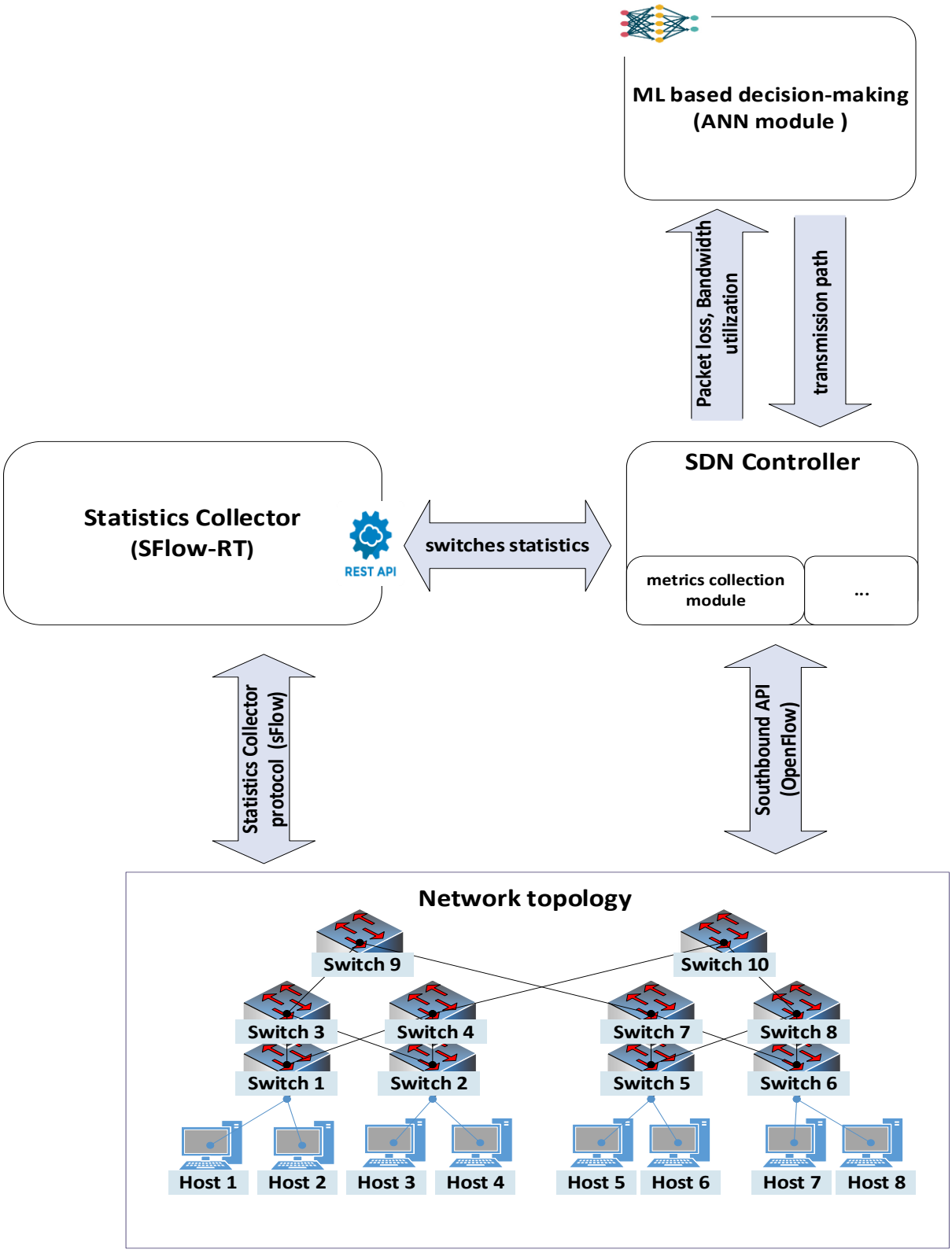

Figure 16. Detailed Architecture

3.3.1. Network Topology

In our work, we have studied and proposed an adaptive routing-based ML approach for Fat-Tree topology which is the most implemented topology in nowadays data centers, the processing elements are interconnected by a tree structure, in which the IP cores are at the leaves of the tree, and the interior nodes are switches. An advantage of a tree structure is that communication distances are short for local communication patterns. Moreover, the fat-tree is a tree structure with redundant interconnections on its branches; the number of interconnections increases as the root is reached. The purpose is to increase the bandwidth at higher levels, where it is most needed. Because it is not feasible to provide a channel between every pair of nodes, the network channels are shared among the IP nodes. [39]

Figure (17) depicts a Fat-Tree topology which is built with k-port switches have k pods, each of which contains two layers of $(k/2)$ switches. It consists of $(k/2)^2$ core switches, $(k^2/2)$ aggregation and edge switches respectively and supports $(k^3/4)$ servers in total. However, the wiring complexity is $O(n^3)$ which is a serious challenge. [40]

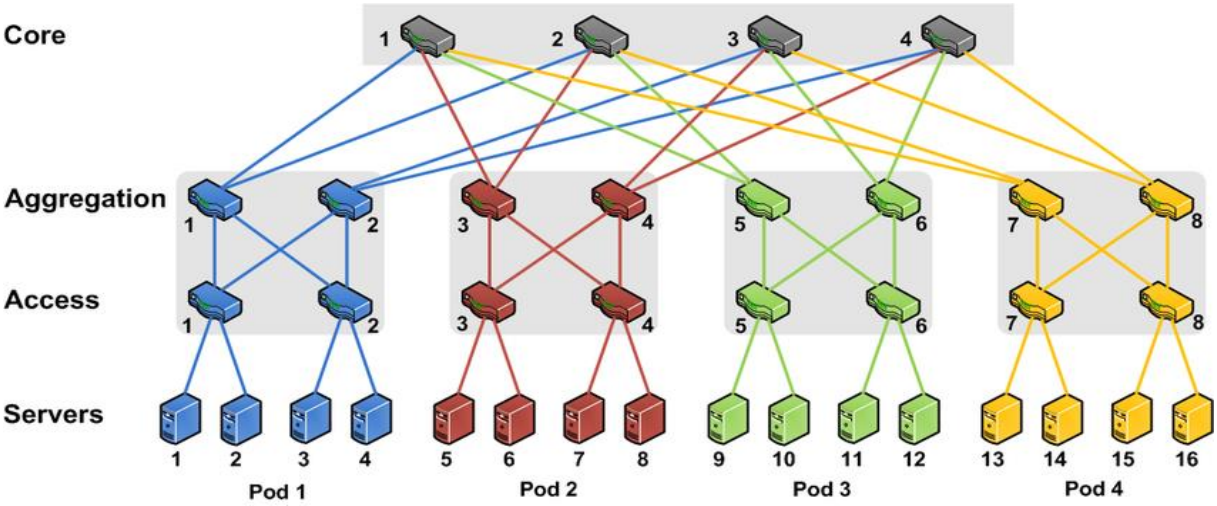

Figure 17. A simple 3-level Fat-Tree topology. [41]

3.3.2. Statistics Collector

The statistics collector module gathers statistics from switch's interfaces where interface counters (bits, packet …) are collected from switches and sent across the network to the Collector.

Statistics received are analyzed to convert into actionable metrics or summary statistics, and then the real-time results are stored.

statistics accessible through REST API, any language that supports HTTP request messages can be used to retrieve statistics from the collector.

3.3.3. SDN Controller

SDN Controller is the main component of the architecture where it fully manages the network topology, implements algorithms that collect network statistics (Bandwidth usage and packet loss, latency) according to specific metrics and uses our proposed ML module to direct network traffic smartly.

3.3.3.1. Metrics Collection

The metrics collection module runs in the SDN controller to collect three real-time metrics namely, bandwidth utilization and packet loss, latency.

A. Bandwidth utilization

One of the basic tasks in monitoring network traffic is to accurately track the utilization of links in your network. The statistic collector used to obtain the link load condition between switches.

The utilization of links on a particular interface are defined by the equations (3.1) and (3.2):

$$\boldsymbol{ifoututilization} = \frac{(ifoutoctets \times 8)}{(ifspeed \times interval)} \times 100 \qquad (3.1)$$

Where

$ifoututilization$: Acronym for interface output utilization refer to the percentage of maximum outbound bandwidth that is being currently used. [42]

$ifoutoctets$: the total number of octets sent from the interface.

$ifspeed$: speed of the interface.

$interval$: time interval.

$$\boldsymbol{ifinutilization} = \frac{(ifintoctets \times 8)}{(ifspeed \times interval)} \times 100 \qquad (3.2)$$

where

$ifinutilization$: Acronym for interface input utilization refer to the percentage of maximum inbound bandwidth that is being currently used. [42]

$ifinoctets$: the total number of inbound octets.

$ifspeed$: speed of the interface.

$interval$: time interval.

Now, each direction on the link has its own bandwidth, so both numbers are needed. If we want to summarize the link bandwidth utilization, we could take the maximum of the two utilizations. The following algorithm (Algorithm 1) describes how to measure link utilization.

Algorithm: Bandwidth utilization algorithm

Input srcifinutilization, srcifoututilization, dstifinutilization, dstifoututilization.
Output BW

srcMaxUtilization = Max (srcifinutilization, srcifoututilization);
dstMaxUtilization = Max (dstifinutilization, dstifoututilization);
BW = Max (srcMaxUtilization, dstMaxUtilization);

Algorithm 1. Bandwidth utilization algorithm

The following table (Table 2) represents a description of the notations used in the previous algorithm.

Notations	Description
Srcifinutilization	Source Port Receive bandwidth
Srcifoututilization	Source Port Send bandwidth
dstifinutilization	Destination Port Receive bandwidth
dstifoututilization	Destination Port Send bandwidth
srcMaxUtilization	Source Port Max bandwidth
dstMaxUtilization	Destination Port Max bandwidth
BW	The Link bandwidth

Table 2. Notation and Variables

B. Packet loss

Packet loss (PLoss) always occurs during the packet processing period in a data transmission device. If switches in the network are too busy to process the incoming packets, the packets may be dropped by switches. The packet loss indicates the busy condition of the switch. The SDN controller can collect a cumulative number of transmitted packets Tx and a cumulative number of received packets Rx at corresponding OpenFlow switch ports. Thus, the packet loss can be calculated by Equation (3.3):

$$\boldsymbol{PLoss} = \frac{Tx - Rx}{Tx} \tag{3.3}$$

Now, each direction on the link has its own packet loss. If we want to summarize the link packet loss, we could take the maximum of the two directions.

C. Latency

According to the OpenFlow specification, the SDN controller detects the current network topology actively using Link Layer Discovery Protocol (LLDP) and maintains global topology information. During the process of link discovery, the controller sends LLDP packets as Packet-out messages to all switches in the network. When the SDN switch receives an LLDP packet from the controller, it sends the packet to all other switches connected to it directly. When a switch receives an LLDP packet from another switch, it sends the LLDP packet to the controller as a Packet-in message for help, because there is no corresponding forwarding rule in the switch's Flow Table. After the SDN controller receives Packet-in messages with LLDP, it can analyze which switches are connected directly to each other, and construct the global topology. [43]

Floodlight controller [44] provides a method using the existing LLDP protocol for measuring latency where latency is computed by injecting timestamps into LLDP packets and sending these LLDP packets as packet-outs to each switch in the network. When the controller receives one of its own LLDPs on a neighboring switch and processes that LLDP as a packet-in, it will examine the timestamp in the LLDP and subtract it from the current time. This elapsed time, minus the control plane latency of the origin switch (calculated by OpenFlow echo request/reply), minus the control plane latency of the switch that sent the packet-in (calculated by OpenFlow echo request/reply), yields the latency of the link, steps illustrated in the figure (18). [45]

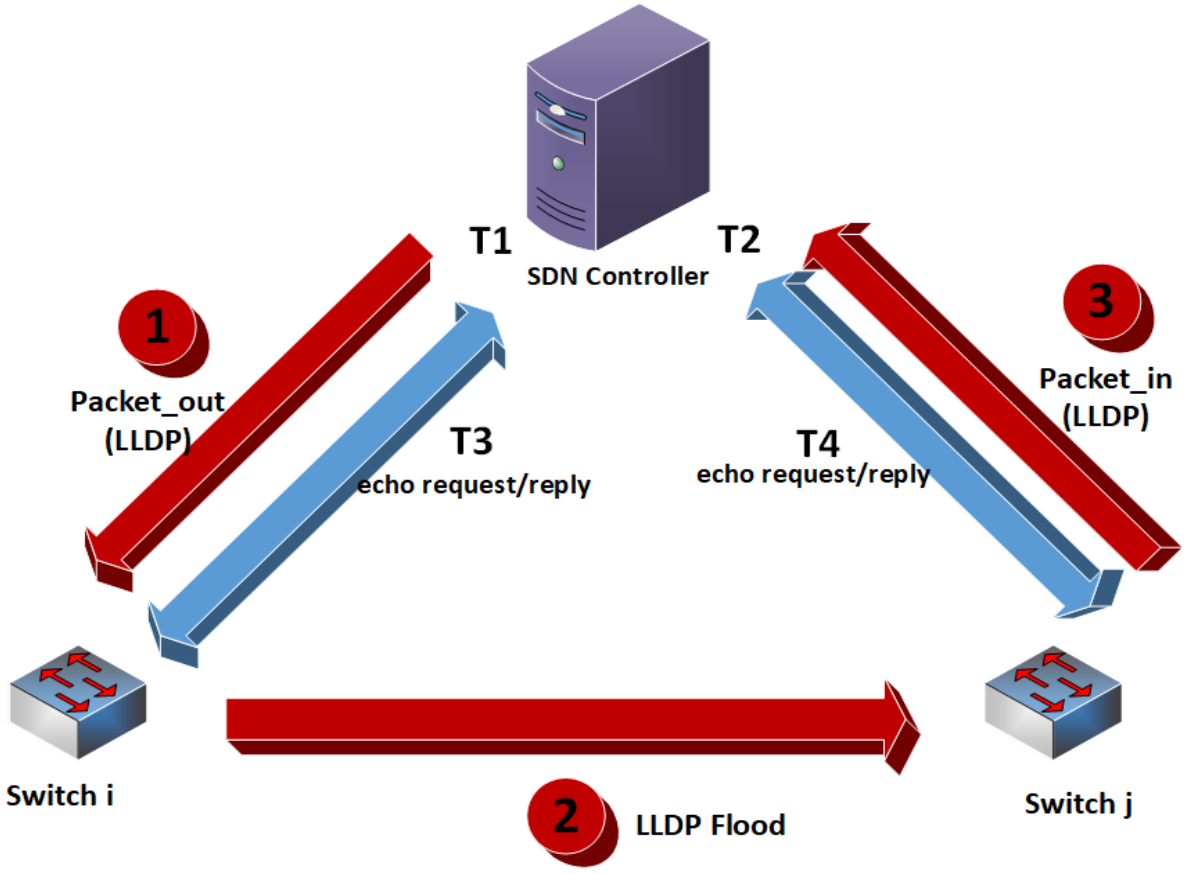

Figure 18. An illustration of how floodlight controller measures the link latency

$latency_l$: the time spent by the link l.

$$total_{time} = T1 - T2$$

$$latency_l = total_{time} - \left(\frac{T3 + T4}{2}\right) \quad (3.4)$$

$$where \ \boldsymbol{l} \ is \ the \ link \ between \ switch \ \boldsymbol{i} \ and \ \boldsymbol{j}$$

For several links L_1, L_2 … L_n with its transmission latency $latency_1$, $latency_2$ … $latency_n$, respectively, then the total latency of this path is as follows:

$$P\_latency = \frac{1}{n} \times \sum_{l=1}^{n} latency_l \quad (3.5)$$

$where \ P\_latency$ means path latency.

3.3.4. ML based decision-making module

The machine learning module used as a decision-maker is an artificial neural network module. This NN module implements the Multilayer Perceptron Network MLP (also called as multilayer feedforward networks, that was explained in the previous chapter) to find the low-latency path between a source node and destination node in real-time, this module using 3 features (bandwidth utilization, packet loss, latency). Because using different methods or different features may cause different accuracy and rationality, it is very important to choose an accurate and rational machine learning method.

we know that the path is a set of links. Now, when we have multiple paths and we want to know which path is less time-consuming. NN module takes load (bandwidth utilization, packet loss) of the links that make up these paths as inputs, then the NN module can predict the time spent in each path.

in our NN architecture, layers are defined as follows:

Input layer

bandwidth utilization and packet loss are taken as input features where:

- $[BW1 - BW_n]$: are bandwidth utilization of the links that make up the output layer paths.
- $[PLoss1 - PLoss_n]$: are packet loss of the links that make up the output layer paths.

  Where **n** is the number of links that make up the output layer paths.

Hidden layer

There are many rule-of-thumb methods for determining the correct number of neurons to use in the hidden layers, such as the following [35]

- The number of neurons should be between the size of the input layer and the size of the output layer.
- The number of neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of neurons should be less than twice the size of the input layer
- The number of neurons is determined by the following equation [34]

$$N = \sqrt{m + n} + a$$

Where N denotes as the number of neurons in the hidden layer, m denotes as the number of neurons in the input layer, n denotes as the number of neurons in the output layer and a is the constant between 1 and 10.

Output layer

Fat-Tree topology has three layers. The core layer is the busiest layer, where all the communication between the PODs passes on it. If we assume that we took all possible paths between each two hosts, this gives us about a thousand possible paths (the higher the number k) which makes the process difficult. For this reason, we chose the number of core switches as the number of paths. NN module's output layer gives the latency of the path depending on input features.

- $[P\_latency_1 - P\_latency_m]$: paths latency.

  Where $\boldsymbol{m}$ is the number of core switches in Fat-Tree topology.

Consequently, the structure of the MLP can be depicted in Figure (19), where the activation function used is the popular ReLU activation function.

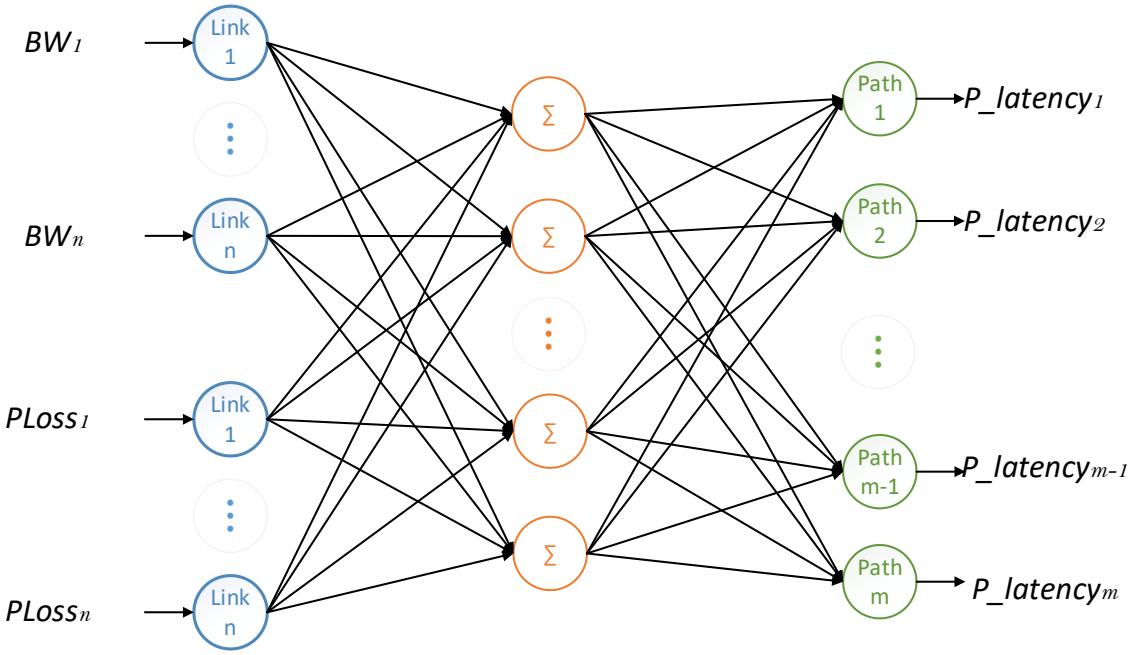

Figure 19. Neural Network Architecture

3.4. The flowchart of the proposed approach

The proposed adaptive routing algorithm is as follows:

1. When a new data-flow transmitted into the SDN domain, OpenFlow switches process the matching between packet head information and flow-tables. If the flow-table matches the packet head information, this data-flow will be transmitted by the Action field in the flow-table. And if there is no flow-table to match this packet, OpenFlow switches will transmit this packet's head information to the SDN controller to decide the transmission path.

2. When finding only one path for data transmission, the SDN controller will create new flow-tables and allocate them to OpenFlow switches to active data transmission

3. When finding multiple paths for data transmission, the SDN controller will transmit bandwidth utilization and packet loss for the links that make up these paths to the ANN module.

4. The ANN module processes the metrics and chooses the low-latency path and sent to the SDN controller.

5. The SDN controller receives the chosen path from the ANN module

We also illustrated our approach using a flowchart as shown in figure (20).

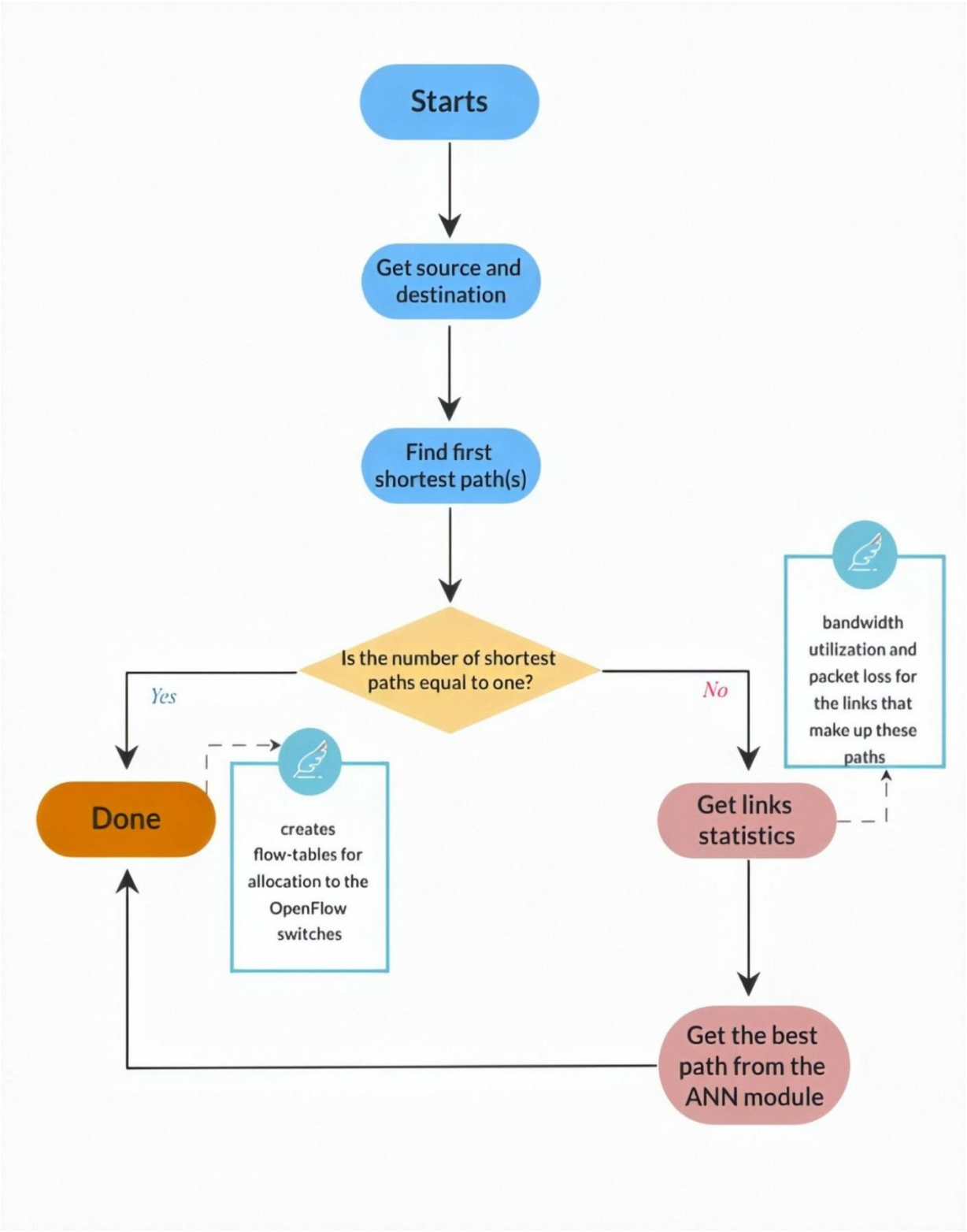

Figure 20. The flowchart of the proposed approach

3.5. Routing algorithm based on the low-latency path

To confirm the effectiveness of the proposed ANN-based adaptive routing, we propose a routing algorithm in an SDN network based on the low-latency path. Since the two proposed routing methods depend on the low-latency path, this enables us to see how effective ANN-based routing is by comparing them together. This algorithm is run by the controller and deals with the Packet_in packet sent by the OpenFlow switch. Table (4) shows the steps of the algorithm, the first step is to create a graph corresponding to the topology, then we compute the first $(k/2)^2$ paths of all switches using Yen's algorithm [46], then we find the low-latency path between every two switches in one direction, and finally, we store the selected path in cache table, steps 2, 3, and 4 repeated every 10s. When a switch needs a path, the controller gets it from the cache table that the algorithm fills.

Algorithm: Routing algorithm

Input links latency, source node S and target node T
/* k is the POD number in Fat-Tree topology & $(k/2)^2$ is the Core number */

Step 1) Computing first $(k/2)^2$ available path according to transmission source node S and Target node T.

Step 2) Computing first $(k/2)^2$ paths.

Step 3) Calculate the first low-latency path.

Step 4) Store the Selected transmission path in the cache table with corresponding S and T.

Algorithm 3. Proposed Routing Algorithm

Figure (21) shows an example of the algorithm process. When Host 1 has a flow transmission request to Host 2, the request will be processed by following steps: Host 1 transmit flow to Switch A, there are no available forwarding rules so Switch A sends a Packet-In message to Controller. When the Controller received the transmission request from Host 1 to Host 2, the Controller obtains three available paths according to the network topology and then selects the low-latency path. After that, the Controller sends a Packet-Out to Switch A, B, C, D, and E to allocate a flow so that the packets will be transmitted to Host 2 then back to Host 1.

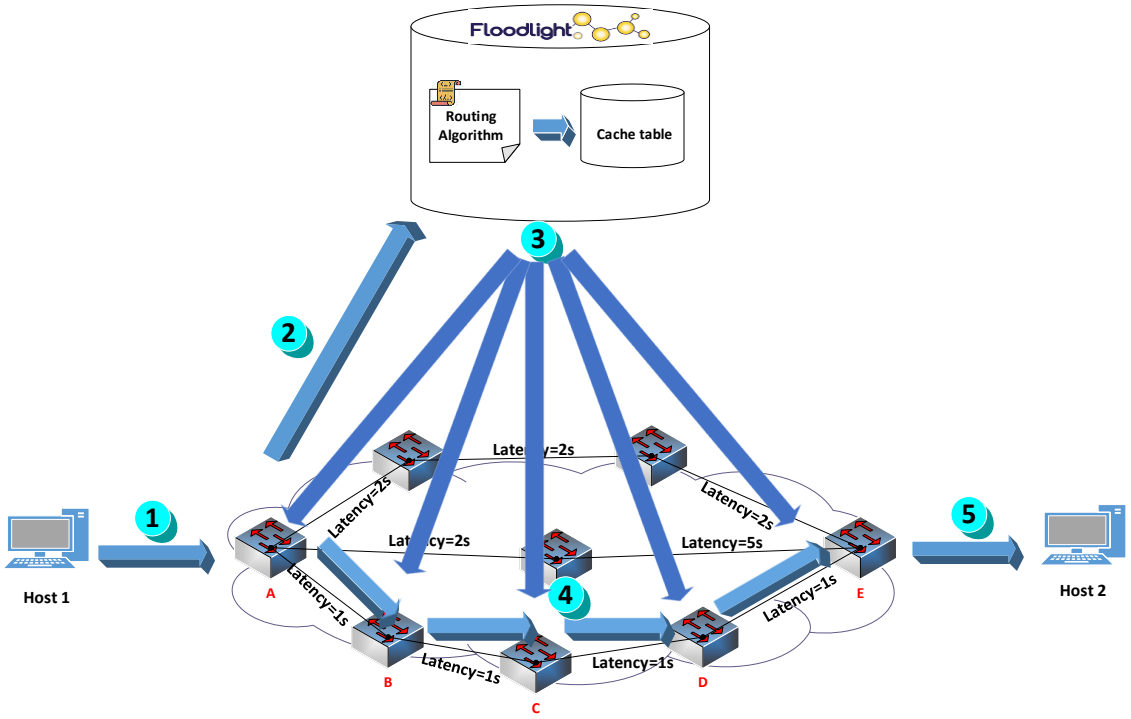

Figure 21. An example of the algorithm process

3.6. Conclusion

In this chapter, we have presented our adaptive routing-based ML approach, we explained each component in it and the relationship between them. After that, we have presented a calculated low-latency routing algorithm in order to see the effectiveness of the proposed ANN-based adaptive routing.

Chapter 4: Experimental Results

4.1. Introduction

In order to confirm the effectiveness of the proposed approach, we will do a case study where we create a scenario simulates reality as possible in order to build a dataset, this dataset uses to training an ANN module on different hidden layers until we get accurate results, and then we see the effectiveness of the proposed adaptive routing is by comparing it with low-latency routing.

4.2. Software Tools

4.2.1. Mininet software

The Mininet program is a network emulator. It can simultaneously perform a set of terminals, routers, Ethernet switches, and respective links to a single Linux Kernel (core). It uses virtualization technology to enable a single system to be simulated as a complete network using the same core system. Each virtual terminal in Mininet works like a real terminal. Moreover, it enables secure connection (type SSH) in the terminal, executes any program (provided that it is installed on the Linux system). The running programs can send packets between the terminals as well as recognizes the link between the interfaces as type Ethernet. While sending of packets is carried out by given connection speed and the required delay. The packets are processed by devices that operate as routers (Ethernet switches, routers) [47].

Briefly, in Mininet, terminals, routers, switches, controllers, and connections are created using software and not hardware. It is possible to create a Mininet network similar to a real network based on hardware, or the creation of a hardware network similar to that of Mininet, which performs the same binary code and applications on each platform.

4.2.1.1. Topologies in Mininet

Mininet supports the creation of customizable topologies. With the creation of the corresponding Python code, it is possible to create flexible topology which can be configured based on the information included in the code and can be reused in multiple experiments [47]. For example, the following illustrates a network topology consisting of a specified number of users (hosts) and associated with a switch.

```
1. #!/usr/bin/python
2. from mininet.topo import Topo
3. from mininet.net import Mininet
4. from mininet.util import dumpNodeConnections
5. from mininet.log import setLogLevel
6. class SingleSwitchTopo(Topo):
```

```
7.  "Single switch connected to n hosts."
8.  def __init__(self, n=2, **opts):
9.  # Initialize topology and default options
10.  Topo.__init__(self, **opts)switch = self.addSwitch('s1')
11. # Python's range(N) generates 0..N-1
12. for h in range(n):
13. host = self.addHost('h%s' % (h + 1))
14. self.addLink(host, switch)
15. def simpleTest():
16. "Create and test a simple network"
17.  topo = SingleSwitchTopo(n=4)
18.  net = Mininet(topo)
19.  net.start()
20.  print "Dumping host connections"
21.  dumpNodeConnections(net.hosts)
22.  print "Testing network connectivity "
23.  net.pingAll()
24.  net.stop()
25. if __name__ == '__main__':
26. # Tell mininet to print useful information
27.  setLogLevel('info')
28. simpleTest()
```

Classes and functions used are explained further:

- Topo: The base class used in topologies Mininet
  - addSwitch (): adds a switch in the topology and returns the name of the switch.
  - addHost (): adds terminal in topology and returns the name.
  - addLink (): adds a two-way connection in topology. (Connections to Mininet both ways unless otherwise stated).
- Mininet: main class for the creation and management of the network
  - start (): Enables the operation of the network.
  - pingAll (): check the connectivity of the terminal by performing successive ping requests between nodes.
  - stop (): Terminates the network operation.
  - net.hosts: Returns the name of all hosts.
  - dumpNodeConnections nodes (): dumps connections to / from a set of nodes.

4.2.1.2. Setting performance parameters

Besides the basic functions of networking, Mininet provides configurable performance and isolation of certain characteristics, through CPULimitedHost and TCLink classes.

```
1.  #!/usr/bin/python
2.  from mininet.topo import Topo
3.  from mininet.net import Mininet
4.  from mininet.node import CPULimitedHost
5.  from mininet.link import TCLink
6.  from mininet.util import dumpNodeConnections
7.  from mininet.log import setLogLevel
8.  class SingleSwitchTopo(Topo):
9.  "Single switch connected to n hosts."
10. def __init__(self, n=2, **opts):
11. Topo.__init__(self, **opts)
12. switch = self.addSwitch('s1')
13. for h in range(n):
14. # Each host gets 50%/n of system CPU
15. host = self.addHost('h%s' % (h + 1), cpu=.5/n)
16. # 10 Mbps, 5ms delay, 10% loss, 1000 packet queue
17. self.addLink(host,switch,bw=10, delay='5ms', loss=10, max_queue_size=1000, use_htb=True)
18. def perfTest():
19. "Create network and run simple performance test"
20.  topo = SingleSwitchTopo(n=4)
21.  net = Mininet(topo=topo,
22.  Host=CPULimitedHost, link=TCLink)
23.  net.start()
24.  print "Dumping host connections"
25.  dumpNodeConnections(net.hosts)
26.  print "Testing network connectivity "
27.  net.pingAll()
28. print "Testing bandwidth between h1 and h4"
29.  h1, h4 = net.get('h1', 'h4')
30.  net.iperf((h1, h4))
31.  net.stop()
32.  if __name__ == '__main__':
33.  setLogLevel('info')
34. perfTest()
```

The most important methods and parameters used are explained further:

- self.addHost (name, cpu): With the use of this command allows the definition of the percentage of total system CPU that will use the virtual user.
- self.addLink (node1, node2, bw = 10, delay = '5ms', max_queue_size = 1000, loss = 10, use_htb = True): Creates a bidirectional connection between two nodes with specific characteristics such as capacity, delay, packet loss tolerance, with a maximum queue size of 100 packets. The parameter bw is expressed in Mb / s, while the delay is followed by the corresponding unit time (s, ms, us). Antitheta loos the parameter is expressed in percentage.

4.2.1.3. Run programs in virtual terminals

Running programs in terminals are the most memorable event during the execution of the experiments, so further commands can be supported from the usual pingAll () and iperf () type commands. This process is supported by the Mininet software. Each terminal in Mininet is a bash shell type associated with one or more network interfaces thus support running bash type commands. For this reason, to communicate with each terminal is mainly used method type CMD [47].

For example: (execute a command from a host and imprinting effect through method cmd).

```
1. h1 = net.get('h1')
2. result = h1.cmd('ifconfig')
3. print result
```

4.2.1.4. Configuration methods of hosts

Hosts in Mininet provide several processes that contribute to the ease of network configuration [47].

- IP (): Returns the IP address of the terminal as a specific interface.
- MAC (): Returns the MAC address of the terminal as a specific interface.
- setARP (): Creates a static ARP entry in the ARP cache of the terminal.
- setIP (): Settings specific IP address for a terminal interface.
- setMAC (): Settings specific IP address for a terminal interface.

For example:

```
print "Host", h1.name, "has IP address", h1.IP(), "and MAC address", h1.MAC()
```

4.2.1.5. Mininet CLI

The Mininet includes Command Line Interface that can operate on a network. It provides a variety of useful commands, and the ability to display xterm window for execution on individual nodes of a network [47].

```
1. from mininet.topo import SingleSwitchTopo
2. from mininet.net import Mininet
3. from mininet.cli import CLI
4. net = Mininet(SingleSwitchTopo(2))
5. net.start()
6. CLI(net)
7. net.stop()
```

Using command-line helps to debug the network, displays the network topology (using the command net), checks the connectivity (with pingall command), and sends commands to all terminals independently.

```
*** Starting CLI:
mininet> net
c0
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (0/2 lost)
mininet> h1 ip link show
746: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
749: h1-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP qlen 1000
link/ether d6:13:2d:6f:98:95 brd ff:ff:ff:ff:ff:ff
```

4.2.2. Floodlight controller

Floodlight Controller is based on Java. The controller comprises an autonomous collection of modules which perform the main functions of Floodlight as OpenFlow controller and applications are developed to overlap (on-top) the base unit controller (REST applications) or to operate together with the controller (Module applications), as shown in the following figure (22) [44].

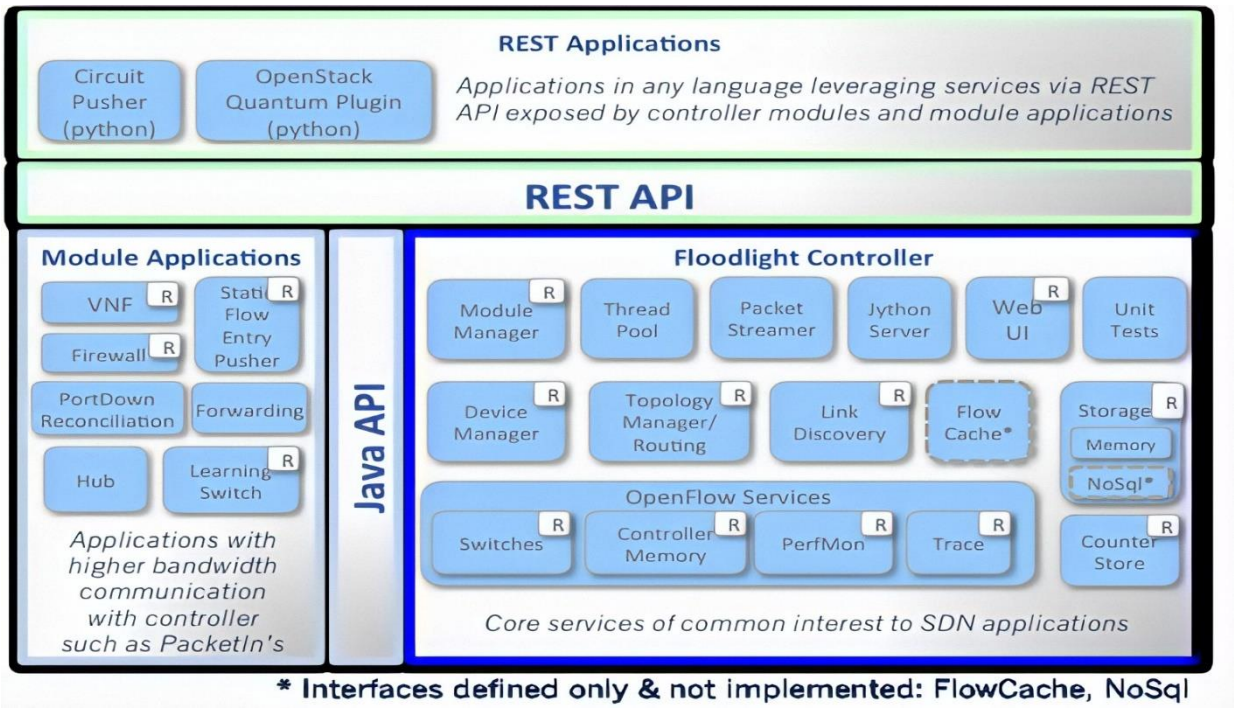

Figure 22.Floodlight architecture [48].

As shown above, Floodlight consists of functional topology and link modules (Topology Manager, Link Discovery), control devices and units (Device Manager, Module Manager), OpenFlow services (OpenFlow Services), unit storage and handling (Storage, Counter Store, Flow Cache, Packet Streamer, Thread Pool).

Applications that use Floodlight as an underlying layer are developed as independent Java modules and use the programming REST interface of the controller (REST API application programming interface). Through its REST API applications, Floodlight communicates with the controller and can use the network for any function. Also, applications can be developed and adjusted to the level of the Floodlight controller, enough to develop and implement the controller modules as Java (Java modules). This way, although more demanding and difficult than the use of the REST API's, has significant benefits as regards the execution speed of the application and greater communication bandwidth offer with Floodlight, since the coupling of application controller becomes more directly to the use of Java API.

4.2.3. sFlow-RT

sFlow-RT is a real-time and scalable platform and a traffic collector to provide real-time visibility to SDN. It is sampling technology to analyze and interpret the flows depending on time.

The sFlow system consists of two basic components:

- sFlow agent – a program that, after initiation on a network node, sends the packet samples and data traffic counters to the component sFlow collector, which later processes this information, and
- sFlow collector – an application that handles sFlow datagrams sent by the sFlow agent and displays (graphically or through REST API) the analyzed parameters.

Figure (23) gives an overview of the network with basic sFlow components and illustrates how the technology works. sFlow has two very important functions are polling and sample rate. The polling function defines the temporal sampling of counters. This is the time interval, expressed in seconds, through which a network device with running sFlow agent counts packets. For example, if that interval is 90, a network device will count all packets that passed through the interface in the last 90 seconds. The sample rate is the function that defines the sampling of data flow. That is the number that defines how many packets, counted during a polling interval, will be forwarded to the sFlow agent. If that number is defined as 1/10, it means that every tenth packet counted during polling will be forwarded to the sFlow agent [49].

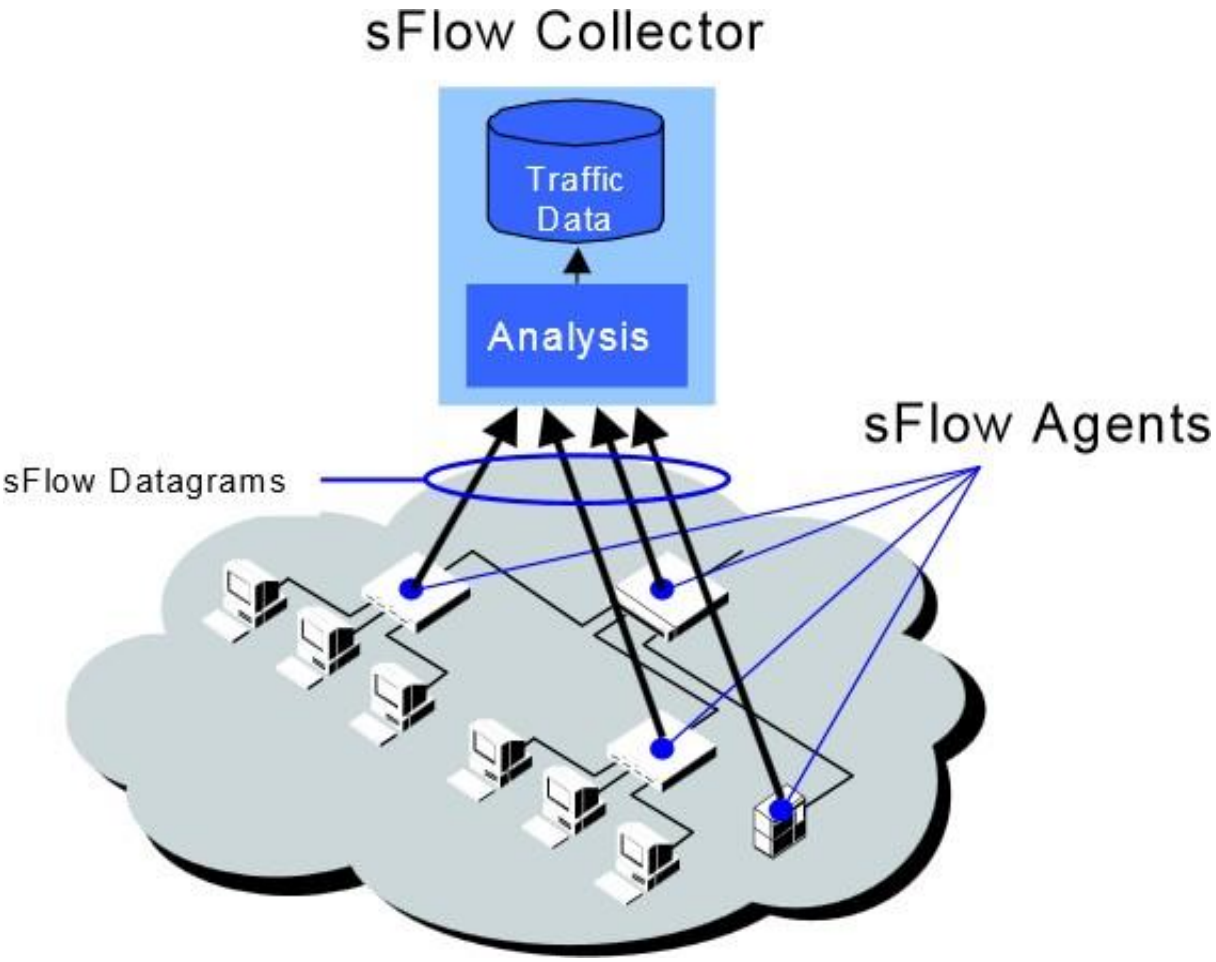

Figure 23. The schema of the sFlow agent and collector network [49]

4.2.3.1. sFlow-RT traffic counters

Traffic counters are defined for each interface of each node on which collecting information about that interface is wanted. The majority of sFlow-RT interface counters (Table 5) use SNMP-names and properties from the SNMP MIB table [50]. The objects in the sFlow-RT are compliant to the SNMP SMI (Structure of Management Information) [51].

Object	Description	Unit
ifinoctets	The total number of octets received on the interface	Octets
ifindiscards	The number of inbound frames (Ethernet) or packets (IP, IPX, AppleTalk) which were chosen to be discarded, even though no errors had been detected (e.g. discarded because buffer space was full)	
ifoututilization	The number that represents link utilization for outgoing traffic of interface	%
ifspeed	Evaluation of interface bandwidth. Interface whose bandwidth does not vary or assessment cannot be made, this entry should contain the nominal value of the bandwidth	bit/s

ifinpkts	The number of received packets on the interface	
ifoutdiscards	The number of outgoing frames (Ethernet) or packets (IP, IPX, AppleTalk) which were chosen to be discarded, even though no errors had been detected (e.g. discarded because buffer space was full)	
ifinutilization	The number that represents link utilization for incoming traffic on interface	%
ifindex	The number that uniquely identifies interface, must be greater than zero	
ifinucastpkts	The number of unicast packets delivered to higher-level protocols (e.g. packets delivered to PPP (*point-to-point*) protocol)	
iftype	Type of interface (e.g. ethernet Csmacd)	
ifinmulticastpkts	The number of incoming multicast packets	
ifouterrors	The number of packets that could not be transmitted because of errors	
ifinerrors	The number of inbound packets that contain errors and cannot be delivered to higher-level protocol (e.g., packets with incorrect CRC (Cyclic Redundancy Check))	
ifoutoctets	The total number of octets sent out of interface	Octets
ifoutpkts	The total number of packets sent out of interface	
ifoutucastpkts	The total number of packets that will be delivered from higher-level protocols and sent to unicast address, including those that will be discarded on lower layer and those that will not be send because of Errors	

Table 3. sFlow-RT Traffic counters [42]

4.2.4. Keras

The implementation of the proposed module is constructed based on using the Python programming language. Python has a large number of scientific libraries for data processing and machine learning approaches. In this work, we used a Keras library, Keras is a deep-learning framework for python that provides high-level building blocks for developing almost any kind of deep-learning model in a much more convenient way than to build it all from scratch. Keras also allows the same code to be run on both the CPU and GPU. Keras does however not handle low-level operations, such as tensor manipulation and differentiation. So, what it does instead is to rely on well-optimized tensor libraries (like TensorFlow) to handle that part, functioning as a backend engine of Keras. As of now, the Keras team recommends using TensorFlow as its backend [52].

4.3. Case Study

4.3.1. Experimental Environment

We used Mininet as the network emulator and Floodlight as the SDN controller to run experiments on multipath topologies to evaluate the performance of the proposed adaptive routing method. The emulated data-center topology is shown in Figure (24), Fat-Tree with 9 Core switches, 18 Aggregation switches, 18 Edge switches, 54 Hosts.

In the real network environment, network traffic presents strong randomness and uncertainty. So, in this work, the experiment is designed to simulate the network environment in reality as possible. Traffic will be transmitted randomly among hosts, also the time spent on traffic is random.

The final purpose of the experiment is to build a dataset for training NN module. During the traffic transmission, we collect metrics (bandwidth utilization, packet loss, and latency). The routing algorithm used for this experiment is the proposed low-latency routing algorithm.

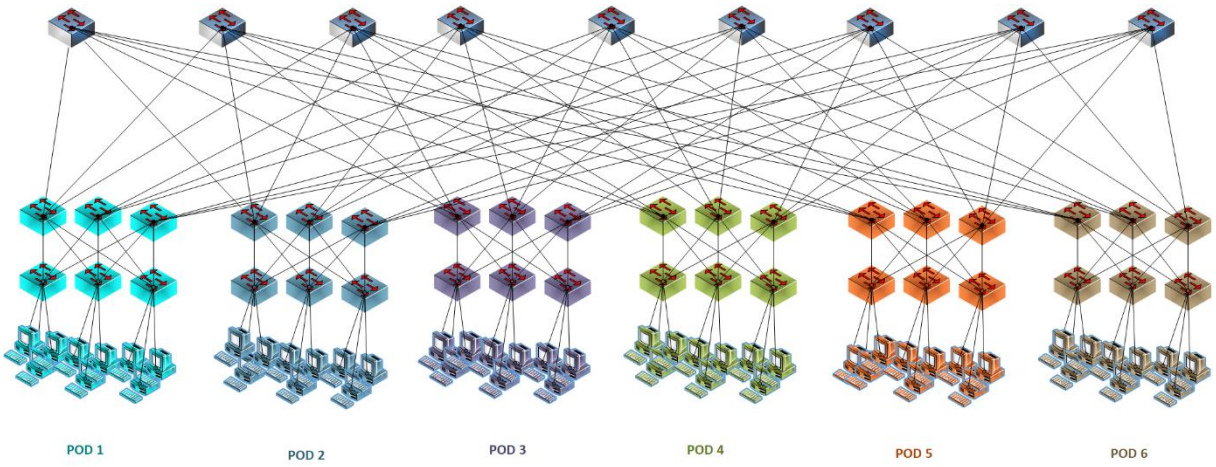

Figure 24. Fat-Tree Topology (k = 6)

4.3.2. Experimental Results Analyze

4.3.2.1. Training Results of neural network module

Before utilizing the MLP to indicate the low-latency path, it is indispensable to train the neural network with a large number of datasets to achieve the least error Artificial Neural Network. We use Mininet to emulate the SDN topology as shown in Figure (24). Hosts in the topology will transmit traffic randomly to other hosts to simulate the network traffic, also the time spent on traffic is random, while the SDN controller utilizes OpenFlow and sFlow

protocol to record the network metrics (bandwidth utilization, packet loss, and latency). We collect metrics in 100 sec to generate the dataset to train the MLP.

Each path between two hosts present in two different pods has 4 links, and we have 9 core switches in our topology. So, the input layer has 72 features, 36 (4 multiply 9) for bandwidth utilization, and 36 for packet loss. Every 10 sec we calculate the latency of paths between all the hosts except the hosts in the same pod, through our topology we will get 270 samples, we use these paths in the dataset to train the NN module. At the end of the scenario, we get 2700 samples. The following timeline (Fig.25) summarizes what we have said.

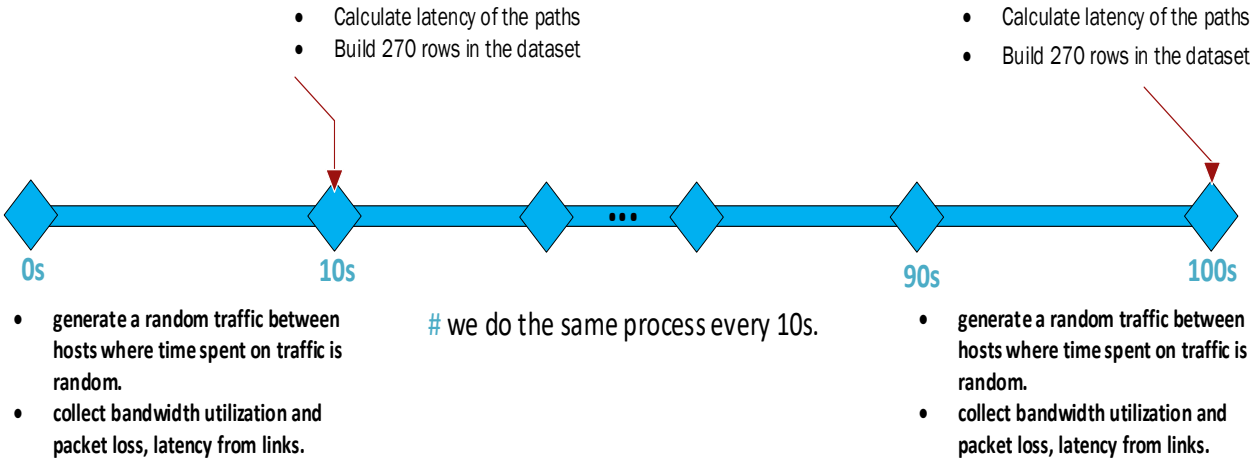

Figure 25. Scenario timeline for building the dataset

The variables in this data set are as follows (Table 4), average bandwidth and packet loss of the links as inputs and path's latency as outputs.

Variable Name	**Type**	**Description**
BW*i*	Numeric	average bandwidth of the link ***i***
PLoss*i*	Numeric	packet loss bandwidth of the link ***i***
P_Latency*i*	Numeric	latency of the path ***i***

Table 4. Notations and variables in this data set

Since, the input layer has 72 features, 36 for bandwidth utilization, and 36 for packet loss and every 10 sec we calculate the latency of paths between all the hosts except the hosts in the same pod. So, the structure of the dataset will be as follows (Table 5).

rows	Time	INPUTS							OUTPUTS		
270 records per 10s	Each 10 until 100 sec	BW_1	…	BW_{36}		$PLoss_1$	…	$PLoss_{36}$	$P\_Latency_1$	…	$P\_Latency_9$

Table 5. dataset structure

As depicted in the previous chapter on the ANN module, the number of neurons in the hidden layer cannot be easily determined. In the procedure of training, we train the MLP with a value of 18, 48, and 140 as the number of neurons in the hidden layer. And we can get the results of these different neural networks to evaluate the performance of these 3 neural networks.

Figure (26) shows the error in the three MLPs with different neuron numbers in the hidden layer. As shown in Figure (26) when the number of neurons in the hidden layer is too small (namely 18), the error of the trained MLP is relatively the largest. Along with the increase in the size of datasets, all the error results of MLP are decreasing. When the number of neurons comes into 140, the error results are relatively least and keep steady. Take all this into consideration, MLP with 140 neurons in the hidden layer is selected, meaning this type of MLP is the fittest one for SDN network case.

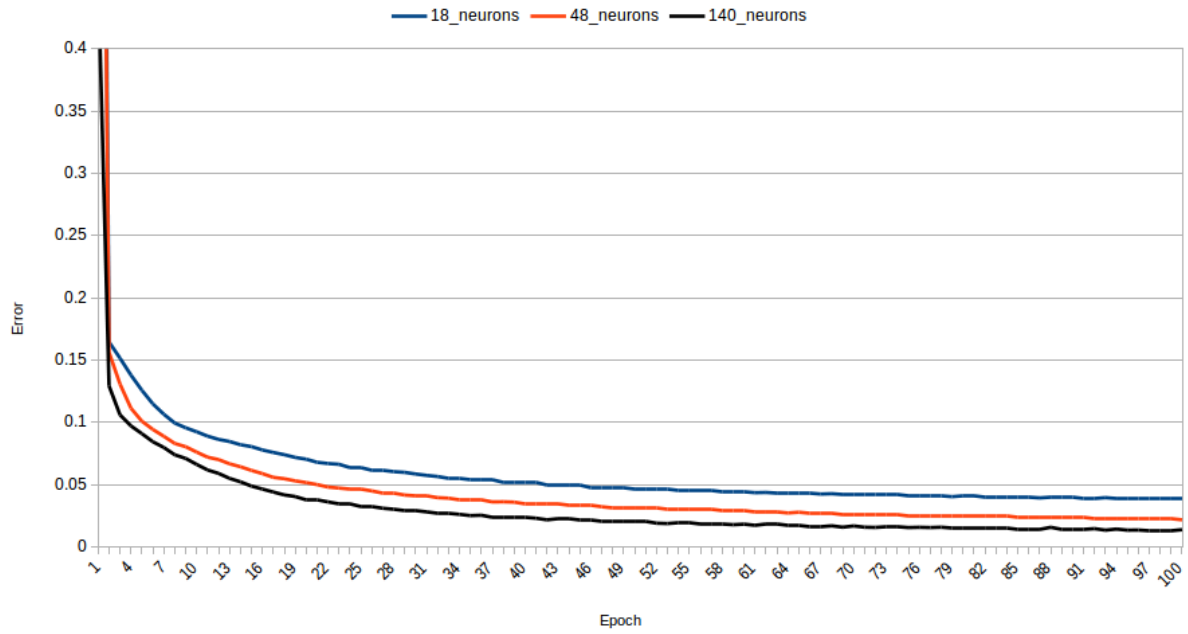

Figure 26. Error for Different number of neurons in the hidden layer of MLP

Figure (27) shows the accuracy in the three MLPs with different neuron numbers in the hidden layer. When the number of neurons comes into 140, the results are more accurate and keep steady, MLP with 140 neurons in the hidden layer is selected, meaning this type of MLP is the fittest one for SDN network case.

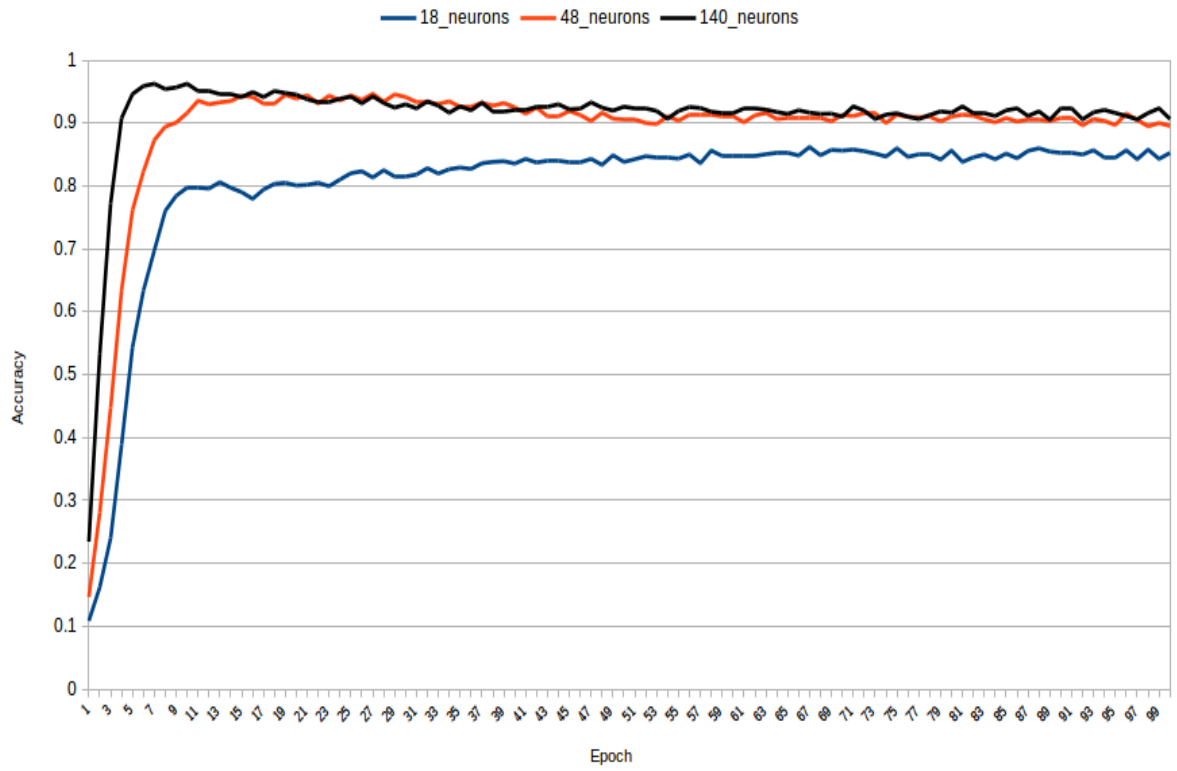

Figure 27. Accuracy for Different number of neurons in the hidden layer of MLP

4.3.2.2. Experimental Results of low-latency Routing based on SDN

In this section, we apply the simulation experiment platform of Mininet and floodlight to confirm the effectiveness of the proposed algorithm and compare it with single-path (first shortest path) transmission methods. The same scenario was launched in two methods, in which traffic is transmitted between hosts present in different pods, also the time spent on traffic is random, and we deliberately have a large traffic to make the congestion in the network. We calculate RTT time during each communication and compare results between the two algorithms. The experiment takes 100 sec. (the RTT is how long it takes for a request sent from a source to a destination, and for the response to get back to the original source. Basically, the latency in each direction).

In order to know the efficiency of the proposed algorithm, two different topologies were tried (Fat-Tree 4-array and 6-array). We examine the performance of the round-trip time in Fat-Tree with 4-array, as shown in Figure (28). Compared with single-path, low-latency routing, on the whole, is smooth and can reduce the time delay.

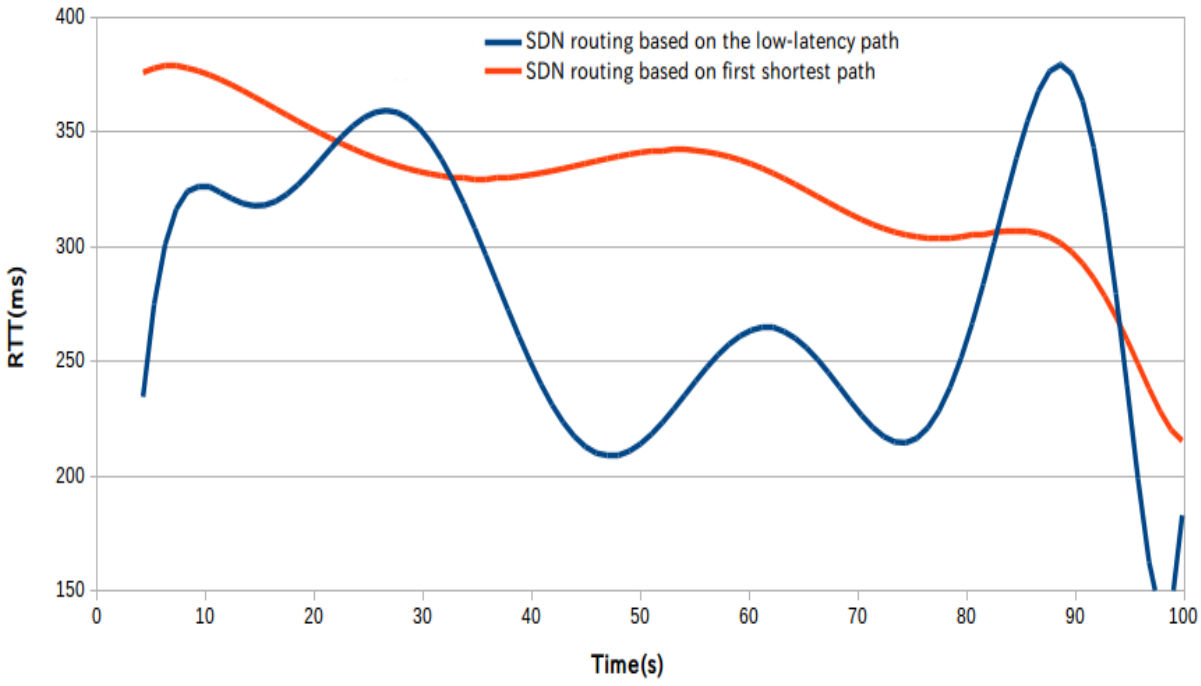

Figure 28. RTT comparison between the two algorithms (4-array Fat-Tree topology experiment)

Now, we examine the performance of the round-trip time in Fat-Tree with 6-array, as shown in Figure (29). Compared with single-path, low-latency routing, on the whole, is smooth and can reduce the time delay. Also, we have noticed that with large topology and large network congestion, the proposed algorithm is much better.

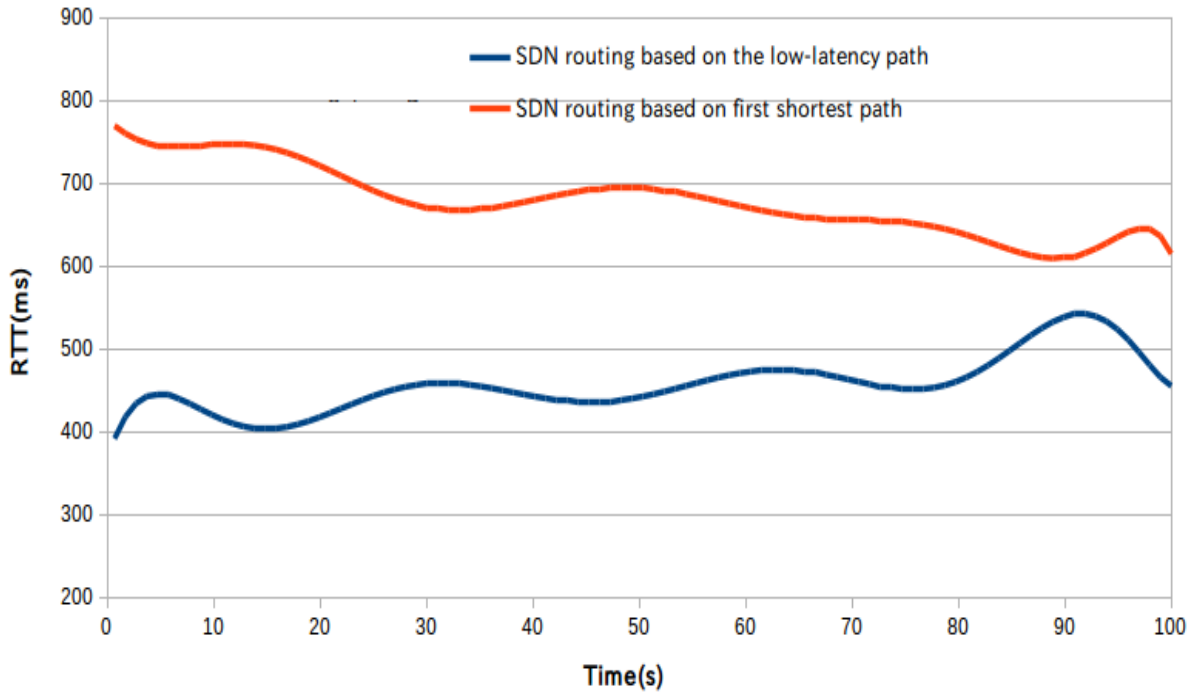

Figure 29. RTT comparison between the two algorithms (6-array Fat-Tree topology experiment)

4.3.2.3. Experimental Results of Adaptive Routing based on ANN

In order to evaluate an adaptive routing approach based on ANN, we have used one other routing method strategy as a comparison. it's the proposed low-latency routing because it performs the same ANN routing approach (they both choose to route on the low-latency path), and this is good for testing the effectiveness of ANN. In the 100 sec of the experiment, traffic is transmitted between hosts present in different pods, and the time spent on traffic is random, and we deliberately have large traffic to make the congestion in the network. During the experiment, the round-trip time and the packet loss will be calculated in each communication, and we compare them in both methods. The experimental results are shown in Figure (30) and Figure (31).

We first examine the performance of the round-trip time, as shown in Figure (30). Compared with low-latency, and single-path routing methods, ANN-based routing, on the whole, is smooth and can reduce the time delay. This is due to ANN-based routing doesn't take time to calculate the ideal path. When a request arrives in order to choose the best path among the paths, ANN predicts directly the transmission path. Contrary to low-latency routing that needs to process each path alone, this is what caused the ANN-based routing response to be faster and reduce the time delay over time. As for the First Shortest Path, RTT is larger because it chooses the same path throughout the experiment. We have noticed that with large topology and large network congestion, the proposed ANN-based routing is much better.

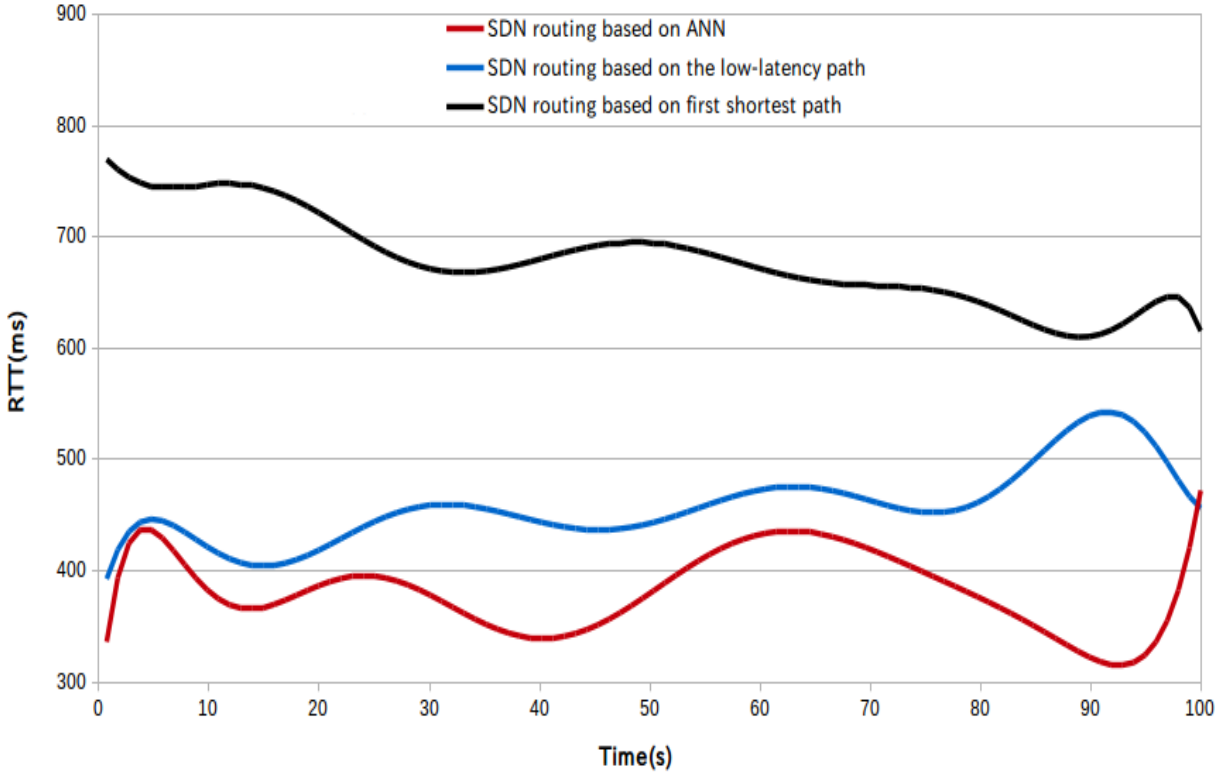

Figure 30. RTT when using the proposed ANN-based routing compared to low-latency and First shortest path routing methods

The packet loss was tested in the same test environment, as shown in Figure (31). As can be seen from the graph, the fluctuation of the packet loss is relatively smooth and less than 0.2% when using the ANN-based routing method compared to low-latency routing method, and less than 3% when compared to First Shortest Path routing method because it makes full use of idle paths, which reduces the adverse effects of congestion caused by a single link.

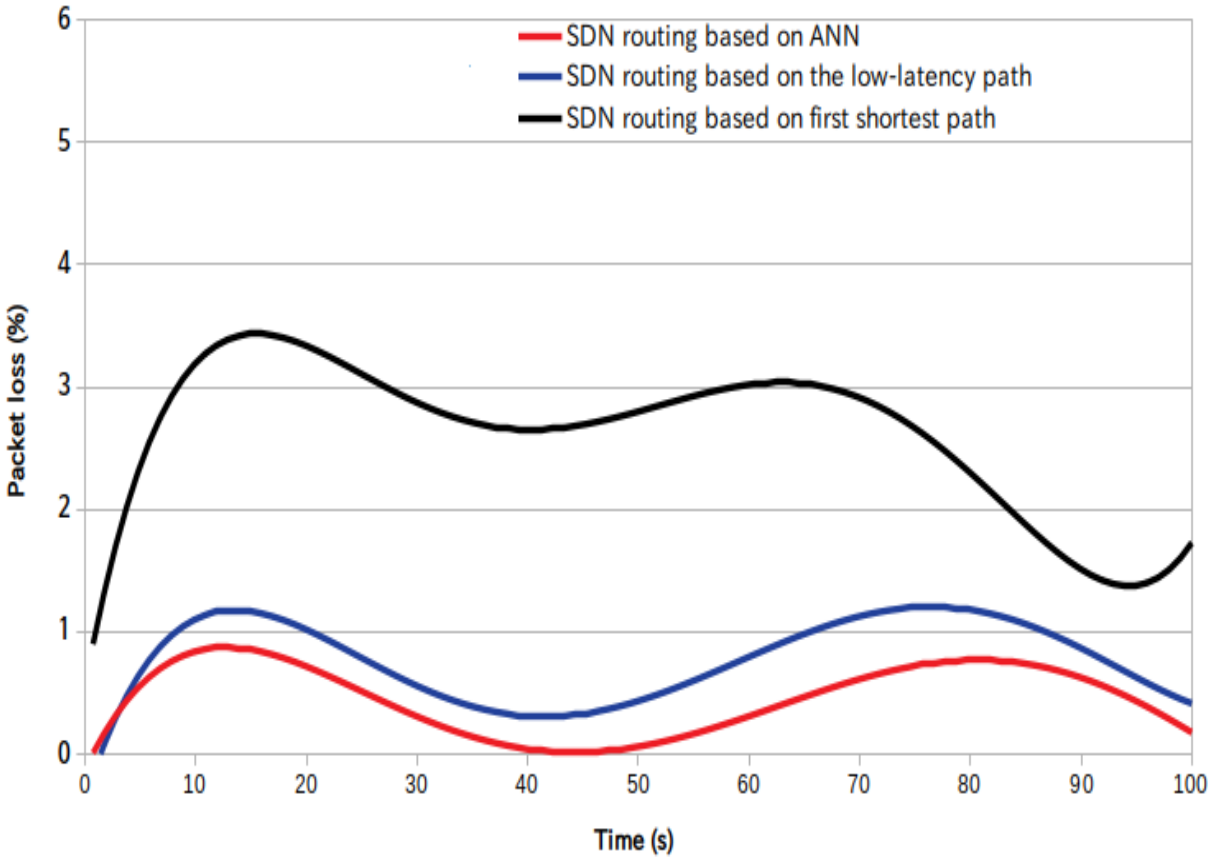

Figure 31. Packet Loss when using the proposed ANN-based routing compared to low-latency and First shortest path routing methods

4.4. Conclusion

In this chapter, we have presented the tests and results of our proposed approach. Additionally, there is a presentation and explanation of the tools and the environment used in this work. We have concluded that our proposed approach can improve the routing process, especially when we have large network congestion.

General Conclusion

Software-defined network is still a new architecture to create and manage computer networks, which are still being developed and tested. Over time protocols that support it (like OpenFlow) become more mature and reliable, and more controllers are developed in different programming languages providing more and more features to create centralized, functional, and reliable computer networks.

The main purpose of this project was to provides an adaptive routing approach for SDN using neural networks. The first main step has been to discover and learn more about SDN and OpenFlow protocol. And the second one was to see How artificial neural networks work. And the third one, confirm the effectiveness of this approach by comparing it in two other methods.

The main purpose of this project was to provide an adaptive routing approach for SDN using neural networks as a Machine Learning approach. The first main step has been to discover and learn more about SDN and OpenFlow protocol, and the second one was to see how artificial neural networks work. Finally, confirm the effectiveness of this approach by comparing it in two other methods.

As the same with traditional networks, path redundancy technology in SDN effectively provides the robustness and stability for the network. But how to evenly distribute traffic among multiple paths has become an urgent problem for SDN researchers. Taking advantage of the global network view of SDN, this project proposes an adaptive routing solution scheme based on real-time path load conditions. This method collects bandwidth utilization, packet loss rate, latency, and integrate them by Artificial Neural Network to indicate the path's time spent. We utilize Mininet and Floodlight controller to simulate the SDN network (on this project is Fat-Tree with 6-array). an adaptive routing approach proposed in this project is applied in the simulation experiment. Compared with low-latency and First shortest path routing methods, the experimental results show that the approach proposes in this project provides better network performance and can reduce network latency, especially when we have large network congestion.

During this study, we concluded that adopting ML in networks gives a great help, especially with the presence of SDN. This study proved this, where we used ML to improve the routing process and we got good results.

Bibliography

[1] B. Samaresh , M. Sudip, R. Sanku Kumar and S. O. Mohammad, "Software-Defined WSN Management System for IoT Applications," *IEEE,* 2016.

[2] M. R. Abbasi, A. Guleria and a. M. S. Devi, "Traffic Engineering," *Journal of Telecommunications and Information Technology,* 2016.

[3] B. N. Astuto, M. Mendonça, X. N. Nguyen, K. Obraczka and T. Turletti, "A survey of software-defined networking : Past, present and futur of programmable netorks," *IEEE,* p. 16, 2014.

[4] Nick Feamster, Jennifer Rexford and Ellen Zegura, "The road to sdn : an intellectual history of programmable networks," *Queue - Large-Scale,* 2013.

[5] Nick McKeown, Guru Parulkar, Tom Anderson, Larry Peterson, Hari Balakrishnan, Jennifer Rexford, Scott Shenker and Jonathan Turner, "Openflow : Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review,* 2008.

[6] Diego Kreutz, Fernando M. V. Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky and Steve Uhlig, "Softwaredefined networking : a comprehensive survey," 2014.

[7] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow and Guru Parulkar, "ONOS: Towards an Open, Distributed SDN OS," *HotSDN'14, , Chicago, IL, USA.,* 22 August 2014.

[8] D. B. Hoang, "telsoc," [Online]. Available: https://telsoc.org/journal/ajtde-v3-n4/a28. [Accessed 08 04 2020].

[9] Paul Göransson and Chuck Black, Software Defined Networks - A Comprehensive Approach, Morgan Kaufmann, 2014.

[10] S. K. N. Rao, "SDN AND ITS USE-CASES- NV AND NFV," *NEC Technologies India Limited,* 2014.

[11] H. Zhang and J. Yan, "Performance of SDN Routing in comparison with legacy routing protocols," *IEEE computer society,* 2015.

[12] Adrian Lara, Anisha Kolasani and Byrav Ramamurthy, "Network innovation using openflow, a survey," *IEEE communications surveys,* 2014.

[13] A. Lara, A. Kolasani and B. and Ramamurthy, "Network Innovation using OpenFlow: A Survey," *IEEE Communications Surveys & Tutorials,* 2014.

[14] "Open Networking Foundation," [Online]. Available: https://www.opennetworking.org/. [Accessed 2020].

[15] "Open Network Fundation," [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/.

[16] M. Wang, Y. Cui, X. Wang, S. Xiao and a. J. Jiang, "Machine learning," *IEEE Network,* vol. 32, pp. 92-99, March 2018.

[17] M. Usama and al, "Unsupervised machine learning for networking: Techniques, applications and research challenges," *IEEE Access,* pp. 65579-65615, 2017.

[18] G. Xu, Y. Mu and a. J. Liu, "Inclusion of artificial intelligence in communication networks and service," *ITU Journal: ICT Discoveries,* pp. 1-6, Oct 2017.

[19] O. Zapletal, "Image Recognition by Convolutional Neural Networks," [Online]. Available: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=146227. [Accessed 11 May 2020].

[20] C. Muenker, "Using Convolutional Neural Networks to distinguish vehicle pose and vehicle class," [Online]. Available: https://www.ke.tu-darmstadt.de/lehre/arbeiten/master/2016/Muenker_Christoph.pdf. [Accessed 8 May 2020].

[21] D. W. Patterson, "Artificial neural networks : theory and applications.," p. 477, 1996.

[22] S. TANMANA, "durofy," [Online]. Available: https://durofy.com/machine-learning-introduction-to-the-artificial-neural-network. [Accessed 07 May 2020].

[23] "Shodhganga Mirror," [Online]. Available: https://shodhganga.inflibnet.ac.in/bitstream/10603/48/6/chaper%204_c%20b%20bangal.pdf. [Accessed 9 May 2020].

[24] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares and a. H. S., "Machine learning in software defined networks: Data collection and traffic classification," *IEEE ICNP,* pp. 1-5, 2016.

[25] Z. A. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt and a. G. Noubir, "Application-awareness in SDN," *ACM SIGCOMM,* pp. 487-788, 2013.

[26] P. Wang, S. C. Lin and a. M. Luo, "A framework for QoS-aware traffic classification using semi-supervised machine learning in SDNs," *IEEE SCC,* pp. 460-465, june 2016.

[27] X. Peng, Q. Wenyu, Q. Heng, X. Yujie and L. Zhiyang, "An efficient elephant flow detection with cost-sensitive in SDN," *IEEE INISCom,* pp. 24-28, March 2015.

[28] L. Yunchun and L. Jingxuan, "MultiClassifier: A combination of DPI and ML for application-layer classification in SDN," *International Conference on Systems and Informatics (ICSAI 2014),* pp. 682-686, 2014.

[29] R. Hajlaoui, H. Guyennet and a. T. Moulahi, "A survey on heuristic-based routing methods in vehicular ad-hoc network: Technical challenges and future trends," *IEEE Sensors Journal,,* vol. 16, pp. 6782-6792, Sept 2016.

[30] L. Yanjun, L. Xiaobo and a. Y. Osamu, "Traffic engineering framework with machine learning based meta-layer in software-defined networks," *IEEE ICNIDC,* pp. 121-125, Sept 2014.

[31] A. Azzouni, R. Boutaba and a. G. Pujolle, "NeuRoute: Predictive dynamic routing for software-defined networks," *arXiv preprint arXiv,* 2017.

[32] Kurose and J. F. Ross, Computer Networking A Top-Down Approach, 7 ed., 2017, pp. 63-72.

[33] R. Margaret, "latency definition," [Online]. Available: https://whatis.techtarget.com/definition/latency. [Accessed 11 May 2020].

[34] C. Chen-xiao and X. Ya-bin, "Research on Load Balance Method in SDN," *International Journal of Grid and Distributed Computing,* vol. 9, pp. 25-36, 2016.

[35] A. M. R. Ruelas and C. E. Rothenberg, "A Load Balancing Method based on Artificial Neural Networks for Knowledge-defined Data Center Networking," *IFIP Latin American Networking Conference,* Oct 2018.

[36] A. M. R. Ruelas and C. E. Rothenberg, "Implementation of Neural Switch using OpenFlow as Load Balancing Method in Data Center".

[37] X. Y.-b. Cui Chen-xiao, "Research on Load Balance Method in SDN," *International Journal of Grid and Distributed Computing,* vol. 9, pp. 25-36, 2016.

[38] C. E. R. Alex M. R. Ruelas, "A Load Balancing Method based on Artificial Neural Networks for Knowledge-defined Data Center Networking," *IFIP Latin American Networking Conference ,* Oct 2018.

[39] A. M. Sllame and A. H. Abdelkader, "A COMPARATIVE STUDY BETWEEN FAT TREE AND MESH NETWORK-ONCHIP INTERCONNECTION ARCHITECTURES," *EUROSIS-ETI,* 2014.

[40] T. WANG, Z. SU, Y. XIA and M. HAMDI, "Rethinking the Data Center Networking: Architecture, Network Protocols,and Resource Sharing," *IEEE ACCESS,* vol. 2, pp. 1482-1483, December 30, 2014.

[41] H. Vardhan, N. Thomas, S.-R. Ryu, B. Banerjee and a. Prakash, "Wireless Data Center with Millimeter Wave Network," *IEEE Global Telecommunications Conference GLOBECOM,* 2010.

[42] "sFlow-RT Metrics," 07 March 2020. [Online]. Available: https://sflow-rt.com/metrics.php.

[43] Z. SHU, J. WAN, J. LIN, S. WANG, D. LI, S. RHO and A. C. YANG, "Traffic Engineering in Software-Defined Networking: Measurement and Management," *IEEE ACCESS,* vol. 4, June 21, 2016.

[44] "project floodlight," 05 March 2020. [Online]. Available: http://www.projectfloodlight.org/floodlight/.

[45] R. Izard, "Links' utilization, Packet loss, and latency," [Online]. Available: https://groups.io/g/floodlight/topic/24923842#2. [Accessed 15 May 2020].

[46] "wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Yen%27s_algorithm. [Accessed 05 May

2020].

[47] "Mininet," [Online]. Available: http://mininet.org/.

[48] "Floodlight atlassian," 05 March 2020. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343549/Architecture.

[49] "Traffic Monitoring using sFlow," March 2020. [Online]. Available: http://www.sflow.org/sFlowOverview.pdf.

[50] "Structure of Management Information Version 2," 4 MARCH 2020. [Online]. Available: http://tools.ietf.org/html/rfc2578.

[51] "Management Information Base for Network Management of TCP/IP-based internets: MIB- II," 4 MARCH 2020. [Online]. Available: https://www.ietf.org/rfc/rfc1213.txt.

[52] N. Jonsson, "Ways to use Machine Learning approaches for software development," *UMEA UNIVERSITET,* 2018.

[53] "ONF Open Network Foundation," [Online]. Available: https://www.opennetworking.org/.

ANNEX A: Scripts created in this project

A.1. Script to create Network Topology (Fat-Tree with 6-array)

```
1.  # called sFlow-RT to install agents in the OpenFlow switches
2.  execfile('sflow-rt/extras/sflow.py')
3.
4.  CoreSwitchList = []
5.  AggSwitchList = []
6.  EdgeSwitchList = []
7.  HostList = []
8.
9.
10. class FatTree( Topo ):
11.
12.     # create a Fat-Tree Topology
13.     def __init__( self, k):
14.         " Create Fat Tree topo."
15.         POD = k
16.         pod = POD
17.         end = pod / 2
18.         iCoreLayerSwitch = (k / 2) ** 2
19.         iAggLayerSwitch = k * (k / 2)
20.         iEdgeLayerSwitch = k * (k / 2)
21.         iHost = iEdgeLayerSwitch * (k / 2)
22.
23.         # Init Topo
24.         Topo.__init__(self)
25.         SCount = 0
26.         for x in range(1, pod * (pod / 2) + 1):
27.           PREFIX = "s"
28.           EdgeSwitchList.append(self.addSwitch(PREFIX + str(x)))
29.           SCount = SCount + 1
30.           print("ESwitch[", SCount, "]")
31.
32.         for x in range(SCount + 1, SCount + pod * (pod / 2) + 1):
33.           PREFIX = "s"
34.           AggSwitchList.append(self.addSwitch(PREFIX + str(x)))
35.           SCount = SCount + 1
36.           print("ASwitch[", SCount, "]")
37.
38.         for x in range(SCount + 1, SCount + ((pod / 2) ** 2) + 1):
39.           PREFIX = "s"
40.           CoreSwitchList.append(self.addSwitch(PREFIX + str(x)))
41.           SCount = SCount + 1
42.           print("CSwitch[", SCount, "]")
43.
44.
45.         count = 0
46.         digit2 = 0
47.         digit3 = 0
48.
49.         for a in range(0, pod):
50.           for b in range(0, pod / 2):
51.              for c in range(2, 2 + (pod / 2)):
52.                 count = count + 1
53.                 digit2 = count / 100
54.                 digit3 = count / 10000
55.                 PREFIX = "h"
56.                 print("host ip:", "10." + str(a) + "." + str(b) + "." + str(c))
57.                 print("host mac:", "00:00:00:" + str(digit3 % 100).zfill(2) + ":" +
    str(digit2 % 100).zfill(2) + ":" + str(
58.                     count % 100).zfill(2))
```

```
                HostList.append(self.addHost(PREFIX + str(count), ip="10." + str(a) + "." + str(b) + "." + str(c),
                                             mac="00:00:00:" + str(digit3 % 100).zfill(2) + ":" + str(
                                             digit2 % 100).zfill(2) + ":" + str(count % 100).zfill(2)))

        for x in range(0, iEdgeLayerSwitch):
          for y in range(0, end):
             self.addLink(EdgeSwitchList[x], HostList[end * x + y], bw=10)


        print("iAggLayerSwitch=", iAggLayerSwitch)
        for x in range(0, iAggLayerSwitch):
         if(varr == 7 ): varr = 4
         for y in range(0, end):
          self.addLink(AggSwitchList[x], EdgeSwitchList[end * (x / end) + y], bw=10)


        for x in range(0, iAggLayerSwitch, end):
          for y in range(0, end):
             for z in range(0, end):
              self.addLink(CoreSwitchList[y * end + z], AggSwitchList[x + y], bw=10)


def Dataset_load():
    print(os.system("java -jar code/link_utilization.jar "))
def Dataset_loss():
    print(os.system("java -jar code/packet_loss.jar "))
def Dataset_latency():
    print(os.system("java -jar code/latency.jar "))


def ping_pod1(h1,h2,h3,h4,h5,h6):

    # Run PING Between hosts
    time.sleep(random.randint(0,5))
    h1.cmd('ping  -s65000 -w 20 %s' % h2.IP())
    time.sleep(random.randint(0,5))
    h1.cmd('ping  -s65000 -w 15 %s' % h3.IP())
    time.sleep(random.randint(0,5))
    h1.cmd('ping  -s65000 -w 20 %s' % h4.IP())
    time.sleep(random.randint(0,5))
    h1.cmd('ping  -s65000 -w 15 %s' % h5.IP())
           time.sleep(random.randint(0,5))
           h1.cmd('ping  -s65000 -w 20 %s' % h6.IP())

       def ping_pod2(h1,h2,h3,h4,h5,h6):

           # Run PING Between hosts
           time.sleep(random.randint(0,5))
           h3.cmd('ping  -s65000 -w 20 %s' % h4.IP())
           time.sleep(random.randint(0,5))
           h2.cmd('ping  -s65000 -w 15 %s' % h5.IP())
           time.sleep(random.randint(0,5))
           h2.cmd('ping  -s65000 -w 20 %s' % h6.IP())
           time.sleep(random.randint(0,5))
           h2.cmd('ping  -s65000 -w 15 %s' % h4.IP())
           time.sleep(random.randint(0,5))
           h2.cmd('ping  -s65000 -w 20 %s' % h3.IP())

       def ping_pod3(f,h1,h2,h3,h4,h5,h6):

           # Run PING Between hosts
```

```
            time.sleep(random.randint(0,5))
            h5.cmd('ping  -s65000 -w 20 %s' % h6.IP())
            time.sleep(random.randint(0,5))
            h4.cmd('ping  -s65000 -w 15 %s' % h6.IP())
            time.sleep(random.randint(0,5))
            h3.cmd('ping  -s65000 -w 20 %s' % h5.IP())
            time.sleep(random.randint(0,5))
            h3.cmd('ping  -s65000 -w 15 %s' % h6.IP())
            time.sleep(random.randint(0,5))
            h4.cmd('ping  -s65000 -w 20 %s' % h5.IP())


        def topology():
            "Create and fat-tree with 6-array"
            topo = FatTree(k=6)
            # build networ with ovsswitch
            net = Mininet(topo=topo,host=CPULimitedHost, link=TCLink, controller=None
    ,switch=partial(OVSSwitch))
            # connect to controller
            c1 = net.addController('c1', controller=RemoteController, ip='127.0.0.1',
     port=6653)


            print("*** Starting network")
            thrds = []
            net.start()

            for i in range(0,9):
              # we first run the script to collect metrics (bandwidth usage, packet l
    oss, latency)
              if i == 0:
                thread = threading.Thread(target=Dataset_load, args=()); thrds.append
    (thread); thread.start();
                thread = threading.Thread(target=Dataset_latency, args=()); thrds.app
    end(thread);  thread.start();
                thread = threading.Thread(target=Dataset_loss, args=()); thrds.append
    (thread); thread.start();

                # build thread Between hosts for runing PING traffic
                thread = threading.Thread(target=ping_pod1, args=(writer,net.hosts[i]
    ,net.hosts[i+9],net.hosts[i+18],net.hosts[i+27],net.hosts[i+36],net.hosts[i+45],))
                thrds.append(thread)
                thread.start()

                thread = threading.Thread(target=ping_pod2, args=(writer,net.hosts[i]
    ,net.hosts[i+9],net.hosts[i+18],net.hosts[i+27],net.hosts[i+36],net.hosts[i+45],))
                thrds.append(thread)
                thread.start()

                thread = threading.Thread(target=ping_pod3, args=(writer,net.hosts[i]
    ,net.hosts[i+9],net.hosts[i+18],net.hosts[i+27],net.hosts[i+36],net.hosts[i+45],))
                thrds.append(thread)
                thread.start()

            # to prevent bugs from threads (synchron)
            for j in thrds:
                j.join()

            CLI(net)
            net.stop()

        if __name__ == '__main__':
            # setLogLevel( 'info' )
            topology()
```

A.2. Script to Collect Bandwidth Utilization Metric.

```
1.  public class linkUtilization {
2.
3.      static JSONParser parser ;
4.      static URL oracle ; // URL to Parse
5.      static URLConnection yc ;
6.      static BufferedReader in ;
7.      static String inputLine;  static StringBuilder s;
8.
9.      public linkUtilization() {
10.         super();
11.     }
12.
13.     public static void main(final String[] array) throws IOException, JSONException,
     InterruptedException, ParseException {
14.
15.         try (final PrintWriter printWriter = new PrintWriter(new File("/home/code/Da
    tabase/link_utilization.csv"))) {
16.             final StringBuilder sb = new StringBuilder();
17.             sb.append("Time,Link,%Link_utilization\n");
18.             int n = 10;
19.             while (true) {
20.                 Thread.sleep(10 * 1000);
21.                 final JSONObject jsonFromUrl = readJsonFromUrl("http://localhost:800
    8/app/link-metrics/scripts/metrics.js/metric/json");
22.
23.                 final JSONObject OutUtilization = (JSONObject)jsonFromUrl.get("ifout
    utilization");
24.                 final JSONObject InUtilization = (JSONObject)jsonFromUrl.get("ifinut
    ilization");
25.
26.
27.                 for (final Map.Entry<String, String> entry : hashMap.entrySet()) {
28.
29.                     // bandwidth  utilization in and out
30.                     float utilization_BW_0 = 0.0f;
31.                     float utilization_BW_1 = 0.0f;
32.
33.
34.                     /********************************** LINK UTILIZATON *********
    *********************************************/
35.
36.                     // get switch id from links file
37.                     final String[] port = entry.getValue().toString().split("<-
    >");
38.
39.
40.                     /***** get max from in and out utilization from the first direct
    ion *******/
41.
42.                     if (OutUtilization.has(port[0]) && InUtilization.has(port[0])) {

43.                         utilization_BW_0 = Max(Float.parseFloat(OutUtilization.get(p
    ort[0]).toString()), Float.parseFloat(InUtilization.get(port[0]).toString()));
44.                     }
45.                     else if (OutUtilization.has(port[0])) {
46.                         utilization_BW_0 = Float.parseFloat(OutUtilization.get(port[
    0]).toString());
47.                     }
48.                     else if (InUtilization.has(port[0])) {
49.                         utilization_BW_0 = Float.parseFloat(InUtilization.get(port[0
    ]).toString());
```

```
50.                         }
51.
52.                     /***** get max from in and out utilization from the second direc
    tion ******/
53.                     if (OutUtilization.has(port[1]) && InUtilization.has(port[1])) {

54.                         utilization_BW_1 = Max(Float.parseFloat(OutUtilization.get(p
    ort[1]).toString()), Float.parseFloat(InUtilization.get(port[1]).toString()));
55.                         }
56.                     else if (OutUtilization.has(port[1])) {
57.                         utilization_BW_1 = Float.parseFloat(OutUtilization.get(port[
    1]).toString());
58.                         }
59.                     else if (InUtilization.has(port[1])) {
60.                         utilization_BW_1 = Float.parseFloat(InUtilization.get(port[1
    ]).toString());
61.                         }
62.
63.
64.                     /********************************** RESULTS ***************
    ************************************/
65.
66.                     // link utilization & packet loss summary
67.                     sb.append(n + "," + (Object)entry.getKey() + "," + String.format
    ("%.2f", Max(utilization_BW_0, utilization_BW_1)) +"\n");
68.                   }
69.                    n += 10;
70.                    if (n == 110) {
71.                     break;
72.                     }
73.             }
74.             printWriter.write(sb.toString());
75.             System.out.println("done!");
76.         }
77.         catch (FileNotFoundException ex2) {
78.             System.out.println(ex2.getMessage());
79.         }
80.     }
81.
82. }
```

A.3. Script to Collect Packet loss Metric.

```
1.  public class packetLossTx_Rx {
2.
3.
4.      static HashMap<String, String> packetLoss = new HashMap<String, String>();
5.
6.
7.      public static void main(final String[] array) throws IOException, JSONException,
     InterruptedException, ParseException {
8.
9.          try (final PrintWriter printWriter = new PrintWriter(new File("/home/code/Da
    tabase/Packet loss.csv"))) {
10.             final StringBuilder sb = new StringBuilder();
11.             sb.append("Time,Link,%Packet loss\n");
12.             int n = 10;
13.             while (true) {
14.
15.                 Thread.sleep(10 * 1000);
16.                 int link_num = 1 ;
17.                 for (final Map.Entry<String, String> entry : packetLoss.entrySet())
    {
18.                     String src = entry.getKey().split("&")[0];
19.                     String dst = entry.getKey().split("&")[1];
20.
21.                     //System.out.println("src "+src+" dst "+dst);
22.                     int getTxsrc = getTx(src); int getTxdst = getTx(dst);
23.                     int getRxsrc = getRx(src); int getRxdst = getRx(dst);
24.                     int Src_TX =  getTxsrc -
     Integer.parseInt((entry.getValue().split("&")[0]).split("-")[0]);
25.                     int Src_RX =  getRxsrc -
     Integer.parseInt((entry.getValue().split("&")[0]).split("-")[1]);
26.
27.                     int Dst_TX =  getTxdst -
     Integer.parseInt((entry.getValue().split("&")[1]).split("-")[0]);
28.                     int Dst_RX =  getRxdst -
     Integer.parseInt((entry.getValue().split("&")[1]).split("-")[1]);
29.
30.                     float PL ;
31.                     if( (Src_TX - Dst_RX) > (Dst_TX - Src_RX) ) { PL =  (Src_TX -
     Dst_RX);  PL /= Src_TX; }
32.                     else { PL =  (Dst_TX - Src_RX); PL /= Dst_TX; }
33.
34.                     if(PL < 0 ) PL = 0 ;
35.                     sb.append(n + "," + "L"+link_num + "," + String.format("%.2f", (
    PL  * 100 ) ) +"\n");
36.                     link_num++;
37.                     packetLoss.put(src+"&"+dst, +getTxsrc+"-
    "+getRxsrc+"&"+getTxdst+"-"+getRxdst);
38.
39.                   }
40.
41.               n += 10;
42.               if (n == 110) break;
43.
44.             }
45.             printWriter.write(sb.toString());
46.             System.out.println("done!");
47.         }
48.         catch (FileNotFoundException ex2) {
49.             System.out.println(ex2.getMessage());
50.         }
51.     }
52.
53.
```

```
54.
55.    public static int getTx(String Switch){
56.
57.        int Tx = 0;
58.        try{
59.
60.         String[] command = {"/bin/sh","-c","ifconfig "+Switch};
61.         Process ifconfig = Runtime.getRuntime().exec(command);
62.             BufferedReader r = new BufferedReader(new InputStreamReader(ifconfig.get
    InputStream()));
63.             StringBuilder sber = new StringBuilder();
64.             String s ;
65.             while (( s = r.readLine()) != null) sber.append(s);
66.             r.close();
67.             Pattern TX = Pattern.compile("TX packets:(\\d+)");
68.             Matcher matcherTx = TX.matcher(sber.toString());
69.             while (matcherTx.find())
70.               Tx = Integer.parseInt(matcherTx.group(1));
71.             return Tx;
72.               }catch (IOException e) {e.printStackTrace();}
73.
74.     return Tx;
75.
76.    }
77.
78.
79.
80.    public static int getRx(String Switch){
81.
82.        int Rx = 0;
83.        try{
84.
85.         String[] command = {"/bin/sh","-c","ifconfig "+Switch};
86.         Process ifconfig = Runtime.getRuntime().exec(command);
87.             BufferedReader r = new BufferedReader(new InputStreamReader(ifconfig.get
    InputStream()));
88.             StringBuilder sber = new StringBuilder();
89.             String s ;
90.             while (( s = r.readLine()) != null) sber.append(s);
91.             r.close();
92.             Pattern RX = Pattern.compile("RX packets:(\\d+)");
93.             Matcher matcherRx = RX.matcher(sber.toString());
94.             while (matcherRx.find())
95.               Rx = Integer.parseInt(matcherRx.group(1));
96.             return Rx;
97.           } catch (IOException e) {e.printStackTrace();}
98.
99.     return Rx;
100.          }
101.
102.       }
```

A.4. Script to Collect Latency Metric.

```
1.  public class latency {
2.
3.      static JSONParser parser ;
4.      static URL oracle ; // URL to Parse
5.      static URLConnection yc ;
6.      static BufferedReader in ;
7.      static String inputLine;  static StringBuilder s;
8.      static float[] latency = new float[108];
9.      static String[] links = new String[108];
10.
11.     public latency() {
12.         super();
13.     }
14.
15.     public static void main(final String[] array) throws IOException, JSONException,
     InterruptedException, ParseException {
16.
17.         for (int i = 0; i < latency.length; ++i) latency[i] = 0.0f;
18.         try (final PrintWriter printWriter = new PrintWriter(new File("/home/code/Da
    tabase/latency.csv"))) {
19.             final StringBuilder sb = new StringBuilder();
20.             sb.append("Time,Link,latency(ms)\n");
21.             int n = 1;
22.             while (true) {
23.                 Thread.sleep(1 * 1000);
24.
25.                 JSONObject link;
26.                 parser = new JSONParser();
27.                 oracle = new URL("http://127.0.0.1:8080/wm/topology/links/json"); //
     URL to Parse
28.                 yc = oracle.openConnection();
29.                 in = new BufferedReader(new InputStreamReader(yc.getInputStream()));

30.                  s = new StringBuilder();
31.                while ((inputLine = in.readLine()) != null)
32.                        s.append(inputLine);
33.                link = new JSONObject("{ \"links\":"+s.toString()+"}");
34.                JSONArray jArray = link.getJSONArray("links");
35.                int indx = 0;
36.                for (final Map.Entry<String, String> entry : hashMap.entrySet()) {
37.
38.                    final String[] port = entry.getValue().toString().split("<->");
39.
40.
41.
42.                  /************************************ LATENCY ******************
    ************************************/
43.                  String src = (port[0].toString().split("-"))[0] ;
44.                  String des = (port[1].toString().split("-"))[0];
45.                  String pid_src = null , pid_des = null ;
46.                  JSONObject jb = null;
47.
48.                  for (int i = 0; i < jArray.length(); i++) {
49.                      jb = jArray.getJSONObject(i);
50.                      if(pid_src.equals(jb.getString("src-
    switch")) && pid_des.equals(jb.getString("dst-switch")))
51.                        { latency[indx] += Float.parseFloat( jb.get("latency").toStr
    ing());
52.                          links[indx] = entry.getKey().toString();
53.                          break;}
54.                      else if(pid_des.equals(jb.getString("src-
    switch")) && pid_src.equals(jb.getString("dst-switch")))
```

```
                            {latency[indx] += Float.parseFloat( jb.get("latency").toStr
ing());
                              links[indx] = entry.getKey().toString();
                              break;}

                    }

                 indx++;
                }
                if (n % 10 == 0) {
                    for (int m = 0; m < MyDataSets.latency.latency.length; ++m) {
                        sb.append(String.valueOf(n) + "," + links[m] + "," + String.f
ormat("%.2f",latency[m] / 10) + "\n");
                        latency[m] = 0.0f;
                        links[m] = "";
                    }
                }
                 n +=1;
                 if (n == 101) break;


            }
            printWriter.write(sb.toString());
            System.out.println("done!");
        }
        catch (FileNotFoundException ex2) {
            System.out.println(ex2.getMessage());
        }
    }


    public static JSONObject readJsonFromUrl(final String s) throws IOException, JSO
NException {
        final InputStream openStream = new URL(s).openStream();
        return new JSONObject(readAll(new BufferedReader(new InputStreamReader(openS
tream, Charset.forName("UTF-8")))));

    }

    private static String readAll(final Reader reader) throws IOException {
        final StringBuilder sb = new StringBuilder();
        int read;
        while ((read = reader.read()) != -1) {
            sb.append((char)read);
        }
        return sb.toString();
    }
}
```

A.5. Script for routing traffic in the low-latency path in SDN.

```
1.  public class lowLatencyRouting {
2.
3.      public static LinkDiscoveryManager ld = null;
4.      public static TopologyManager tm = null;
5.      public static IOFSwitchService switchService;
6.      static ArrayList<String> PATHs ;
7.      static HashMap<String, List<Path>> mapOfPaths = new HashMap<String, List<Path>>(
    );
8.      static HashMap<String, List<Route>> AllPaths = new HashMap<String, List<Route>>(
    );
9.      public static Map<String, String> PathLatency = Collections.synchronizedMap(new
    HashMap<String, String>());
10.     static OFMessageDamper messageDamper;
11.     HashMap<String, String> mapOfpatH = new HashMap<String, String>();
12.     static int var = 0, muasure=0;
13.     static Boolean pol;
14.     private final static Object lock = new Object();
15.
16.
17.     public lowLatencyRouting() {
18.          Map<Link, LinkInfo> links = new HashMap<Link, LinkInfo>();
19.
20.
21.         while (true) {
22.
23.             if (ld != null) {
24.
25.                 links.clear();
26.                 links.putAll(ld.getLinks());
27.
28.             HashMap<String, String> mapOfpath = new HashMap<String, String>();
29.             HashMap<String, String> map = new HashMap<String, String>();
30.             List<String> PATH = new ArrayList<String>();
31.                     ArrayList<Edge> edge = new ArrayList<Edge>();
32.                 for (Link link: links.keySet()) {
33.                     if(var == 0){
34.                         edge.add(new Edge(link.getSrc().toString(),link.getDst().toS
    tring(),1));
35.                         if(!mapOfpatH.containsKey(link.getDst().toString()+" "+link.
    getSrc().toString()))
36.                             mapOfpatH.put(link.getSrc().toString()+" "+link.getDst(
    ).toString(),link.getSrc().toString()+"-"+link.getSrcPort()+" "+link.getDst()+"-
    "+link.getDstPort().toString());
37.                         }
38.
39.                     mapOfpath.put(link.getSrc().toString()+" "+link.getDst().toStrin
    g(),link.getSrc().toString()+"-"+link.getSrcPort()+" "+link.getDst().toString()+"-
    "+link.getDstPort()+"::"+link.getLatency().getBigInteger());
40.                      }
41.
42.
43.                 if(var == 0){
44.                       messageDamper = new OFMessageDamper(1000,EnumSet.of(OFType.FLO
    W_MOD),250);
45.                       Graph graph = new Graph(edge);
46.                       TopologyInstance ti = tm.getCurrentInstance(true);
47.                       Set<DatapathId> switches = ti.getSwitches();
48.                       DatapathId[] IDs = new DatapathId[switches.size()];
49.                       switches.toArray(IDs);
50.                       List<Path> paths;
51.                       for (int i = 0; i < IDs.length; i++) {
52.                             for (int j = i+1; j < IDs.length; j++) {
```

```
                                paths = new DefaultKShortestPathFinder().findShortestPaths(IDs[i].toString(),IDs[j].toString(), graph, 4);
                                mapOfPaths.put(IDs[i].toString()+" "+IDs[j].toString(), paths) ;
                            }}
                        if(links.size() == TOTAL_NUM_LINKS && switches.size() == NUM_SWs) var++;
                       }

                   for (final Entry<String, List<Path>> entry : mapOfPaths.entrySet()) {
                       int latency , min = 0  , val = 0;
                         for (Path path : entry.getValue()){
                         latency  = 0;
                         PATHs = new ArrayList<String>();
                         for (int l = 0; l < path.getNodeList().size()-1; l++) {
                             String SrcDst = mapOfpath.get(path.getNodeList().get(l)+" "+path.getNodeList().get(l+1));
                             if(SrcDst != null && !"".equals(SrcDst) ){
                                 PATHs.add(SrcDst.split("::")[0]);
                                 latency += Integer.parseInt(SrcDst.split("::")[1]);

                                 if(val != 0 && min < latency ) break;
                             }

                           }
                         if(val == 0) {min = latency; PATH = PATHs; }
                         else if(latency < min ){min = latency; PATH = PATHs;}
                         val++;

                         }
                         map.put(entry.getKey().split(" ")[0]+" "+entry.getKey().split(" ")[1],PATH.toString().replace("[","").replace("]","").replace(", "," "));

                    }

                   synchronized (RoutingBasedOnLatency.map) {
                                        // return latency from each path to take the low one.
                       RoutingBasedOnLatency.map.clear(); RoutingBasedOnLatency.map.putAll(map);
                    }
            }


            try {
                Thread.sleep(10000);
            } catch (InterruptedException e) {}
        }

}

}
```