République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique
Département Informatique

Mémoire de Licence

Filière: Informatique

Spécialité:ISIL

Thème

implementation of an intelligent traffic management protocol in SDN networks

Sujet Proposé par :
M (e) : Mme Doukha

Mme Bouziane :

Soutenu le :../../….

Présenté par :

- **MAZRI Said Nassim**
- **MERABTENE Nasreddine**

Devant le jury composé de:

M…………….. Président (e)
M……………… Membre

Binôme n° : 44 / 2021

Thanks

We would like to express our gratitude to our promoters Mrs.Bouziane and Mrs.Doukha. we thank them for having us supervised, guided, helped and advised us. And truly appreciate the help and support they've shown us over this long path.

We extend our sincere thanks to all the professors, speakers and all the people who by their words, their writings, their advice and their criticisms guided our reflections and agreed to meet with us and answer our questions during our research.

We thank our parents and families who supported and encouraged us. Finally, we thank our friends Ikram, Chakib, Faycal and Rania and all the other great people that I couldn't name who have always been there for us. Their unconditional support and encouragement has been a great help.

To all of these speakers, we present our thanks, respect and gratitude.

Contents

List of Figures

General introduction

The world is in a constant development in its all areas and categories, with the field of computer science growing exponentially, faster than any other given domain. The amount of information that is being treated, sent and received keeps increasing everyday.

Having information settling in one place is rare to happen (we are always in the need to send and receive text messages, photos, videos, controlling other machines remotely, saving important information on databases built in either local or online networks...) So basically, data transmission is needed in both our daily life use and professional fields.

Networking, the field in computer science that is responsible on data transmission, has seen sight on a newer and relatively better architecture than the traditional network architecture that were and are used since its appearance known as Software-Defined Networking (SDN)

The goal is certainly to provide a faster reliable networks with less probabilities of running into issues and overall bring a better experience to users. However, even with this evolution, the problem of network traffic management in SDN has even became a research subject due to its complexity. So finding a protocol or an algorithm that realises a routing that leads to achieve the best network performance optimizations is essential

Fortunately, with the existence of artificial intelligence that provides interesting and reliable results in multiple categories of computer science, it is not secret that machine learning methods can also be implemented in our problematic.

In the frame of our undergraduate project in the field of computer science, it has been asked from us to search and implement Artificial Intelligence routing protocol in the Software-Defined Networking.

In this memoir, we've implemented one of the best AI routing protocols in SDN. Starting with chapter 1 where we get a closer look to what an SDN is and what it is composed off and how it functions. And moving to the second part of the chapter where we give a brief definition of what artificial intelligence is with getting a closer look on the existing machine learning methods. Chapter 2 focuses mainly on routing protocols in SDN, giving examples of classical routing on SDN and others that use AI. And comparing protocols which will help us decide the best learning method to use, which will eventually be used in the implementation. Chapter 3 and 4 focus on the chosen solution, where we get a more detailed look on how the routing protocol works with the chosen AI method in the third chapter. Whereas the last chapter contains details on the tools that were used to realise the solution including the results and an evaluation.

We conclude this memoir by describing the main goals we have attended from this project.

Chapter 1

State of art

1 Overview on SDN

1.1 Introduction

With the progress and the development of communication technologies, network management is becoming more and more complex, which is leading to introduce a new paradigm in this field, where bandwidth allocation, routing, networking, scaling and configuration become easier and less expensive.
It is in this context that SDN technology was born, developed by *Open Networking Foundation* (ONF), created in 2011. This technology provides a high level of abstraction allowing to centrally manage and configure various network services and to see all an infrastructure as a single programmable entity.

1.2 Definition

SDN means *Software-Defined Networking*, a network that is defined by a software allowing network innovation, enabling dynamic and programmatically efficient network configuration to improve networking performance and monitoring.
This networking paradigm is mainly based on decoupling the network's control panel from the forwarding panel, which results in turning traditional network's complicated routing devices into simple switches whose job is to follow the policy that is depicted by an intelligent and programmable logically centralized controller, which is different from traditional networks where routing devices perform both forwarding and control functions.
Today, major networking vendors (Cisco , Google , ...) are releasing network infrastructure that support SDN, due to the promising opportunities that this model brings in managing this field, for its simplicity, programmability and elasticity.

1.3 Architecture

The SDN is composed of three main layers: application layer, control layer and infrastructure layer. The communication between these layers is realized via the control layer with the aid of Application Programming Interface (Southbound, Northbound)

1.3.1 The application layer

SDN applications are programs that communicate behaviors and needed resources with the SDN controller via application programming interfaces (APIs), and define the policy of the network's control logic as well as the routing policy.

1.3.2 The control layer

Located between the application and the infrastructure layer, it works on managing and configuring resources of the data layer via the APIs Southbound, this layer is mainly established with a centralized SDN controller that allows hosting the control's logic and administration of the network. The role of this layer lays in connecting the data to the hardware in the infrastructure layer via the southbound API and to the applications in the application layer via Northbound API.

1.3.3 Southbound interface

Application Programming Interface that allows the controller to interact with the switches/routers of the infrastructure layer. the most common used protocol as a southbound interface is OpenFlow.

1.3.4 Northbound interface

this interface is used to program the transmission elements by exploiting the abstraction of the network provided by the control plane, it represents the link between the applications and the SDN controller. The applications can tell the network what they need (data, storage, bandwidth...) and the network can deliver those resources, or communicate what it has. among those APIs, the most known would be the API REST, that calls HTTP queries (GET, POST...) to set a queries exchange.

1.3.5 East/West interface

it has been introduced to allow the communication between the controllers in a multi-controller architecture in order to synchronize the network states. this architecture is recently added to the SDN network.

1.3.6 The infrastructure layer

the layer that comes in the bottom, also can be referred to as (Data layer), contains all the transmission equipment (hardware) such as routers and switches. Its role relies in transporting the traffic generated according to the decisions made by the control layer.

1.4 SDN application areas

SDN networks have a wide variety of applications in network environments and they allow personalized control. They also simplify development and deployment of new network services and protocols. Here are some examples of SDN applications:

- **Data center and Cloud computing:** SDN is seen as one of the best and newest solutions, which allows to configure and easily manage the cloud and date centers.
- **Multimedia and QoS:** Internet architecture of nowadays relies on sending packets without achieving required performance when increasing traffic. Multimedia applications such as streaming video, video conferencing ... etc require a stable network resources and tolerate transmission errors .But in an SDN, different paths can be selected according to the flow rate for the various traffic flows.
- **Mobile wireless networks:** SDN is also being used in the domain of mobile wireless networks, specifically in 5G technology.

1.5 Benefits of SDN

1.5.1 Automatic configuration :

SDN allows network administrators to configure, administer, secure and optimize networks quickly through dynamic and automated SDN programs.

1.5.2 Extension capacity: scalability

With SDN, we acquire a dimension of network programmability. The latter easily changes the behavior of the network, adding new parts or new services without this resulting in very long delays

1.5.3 Flexibility

SDN is inseparable from the notions of flexibility. In fact, the services linked to the networks (priorities, routing, etc.) are grouped together in a controller, which oversees various equipment. Rules can thus be enacted to automate tasks. This is called the programmability of the network. The rules defined within the controller are thus transmitted to the various devices. It follows that the administrator can very easily automate, or even program your network.

1.6 Drawbacks of SDN

[1]

- The centralized controller is a potential point of attack and failure
- The Southborn Interface (SBI) is vulnerable to threats that could degrade the availability, performance and network integrity.
- the vulnerability of Openflow controllers and switches to denial of service attacks. [12]

1.7 The OpenFlow protocol

OpenFlow is a framework originally developed at Stanford University and currently under active standards development through the *Open Networking Foundation* (ONF). it is used by the controller to transmit instructions to the switches in the data layer that allows to program their data plan and retrieve data from the hardware equipment in order for the controller to have a global logical view of the physical network. Using the OpenFlow protocol, a controller not only is able to define transfer rules by sending flow inputs, but also learn network statistics, hardware details, ports, connectivity status and network topology of Ethernet switches.

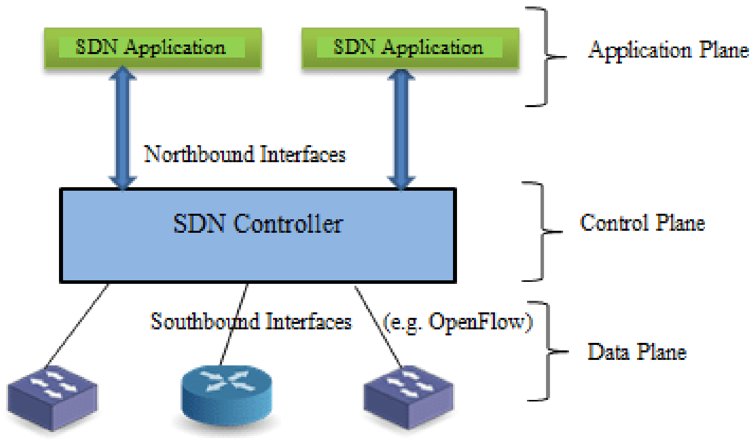

Figure 1.1: SDN Architecture. [5]

1.7.1 Components of an OpenFlow switch

An OpenFlow Switch consists of one or more flow tables and a group table, which perform packet look ups and forwarding, and an OpenFlow channel to an external controller. The switch communicates with the controller and the controller manages the switch via the OpenFlow protocol. Using the OpenFlow protocol, the controller can add, update, and delete flow entries in flow tables, both reactively (in response to packets) and proactively. Each flow table in the switch contains a set of flow entries; each flow entry consists of match fields, counters, and a set of instructions to apply to matching packets. (check Figure 1.2)

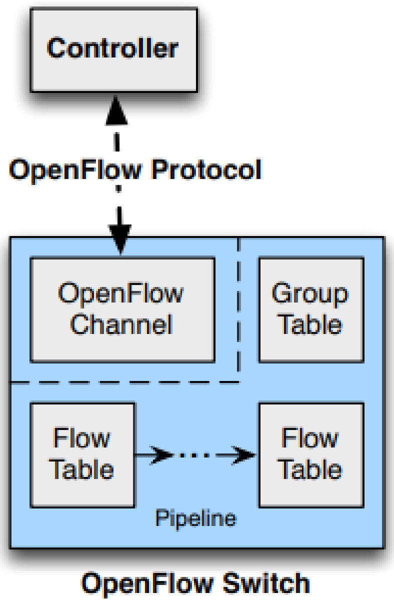

Figure 1.2: Main conception of an openFlow switch. [6]

1.7.2 Establishing a Switch-Controller Connection

It exist three cases of establishment of a connection between the switchOpenFlow and the controller

Case 01 : Connection established The switch sends a packet OFPT_HELLO with the supported OpenFlow version. The controller then checks the OpenFlow version and replies with a message OFPT_HELLO indicating the OpenFlow version that will be used. Therefore, the connection is established.

Case 02 : Connection failure As in the previous case, the switch sends an OFPT_HELLO packet with the protocol version used, the controller realizes that it does not support the OpenFlow version of the switch. Therefore, it returns an OFPT_ERROR packet indicating that it is a compatibility issue.

Case 03: Emergency mode As in the previous cases the switch sends an OFPT_HELLO packet to the controller, if it does not respond, it then goes into EMERGENCY_MODE emergency mode. The switch uses its default flow table, if however a packet does not match any record in the table it deletes it.

1.7.3 Behavior of an OpenFlow switch

When a switch receives a packet, it compares its header with the rules of its flow tables. If the mask configured in the header field of a rule matches the header of the packet, the rule with the highest priority is selected, then the switch updates its counter to this rule.If the flow entry corresponding to the packet does not exist in all the flow tables of the switch, in this case, it is a missing table. The entrance to this table in the flow table allows to specify the processing of this packet based on one of the instructions following instructions (depending on the basic configuration of the Openflow switch):

1. Remove the package
2. Forward the packet to another flow table.
3. Send the packet to the controller through the secure channel.

the flow entries in the flow tables are structured as follows :

- **Match fields:** This is the part the controller relies on to match packets, this consists of checking the input port or the packet header in order to apply an action.
- **Actions:** Used to modify the set of actions to be applied to the packet.
- **Stats:** It is possible to have a certain number of statistics which one could use to manage the entries in the flow tables, to then decide if a flow entry is active or not.

1.7.4 OpenFlow messages

The OpenFlow switch protocol supports three types of message: symmetric, asynchronous and command messages.

1. **Symmetric messages :** they are initiated by the switch or controller and sent without solicitation. For example, there are HELLO messages which are exchanged once that the secure channel has been established.
   - **Hello:** Hello messages are exchanged between the switch and the controller at the start of connection.
   - **Echo:** Echo request / reply messages can be sent from switch or controller, and return an echo reply. They serve mainly to check the quality of the controller-switch connection life.
   - **Error:** The switch uses error messages to report problems on the other side of the connection. They are mainly used by the switch to indicate the failure of a request initiated by the controller.
2. **Asynchronous messages:** initiated by the OpenFlow switch and are used to notify the controller of the passage of packets or the change in switch state.
   - **Packet-in:** For all packets which do not have a corresponding flow entry or in the event that the corresponding action is sending to the controller.
   - **Flow-Removed:** It informs the controller of the removal of a flow entry from a flow table.

- **Port-status:** It informs the controller of a modification on a port.
- **Role-status:** It informs the controller of a change in his role. When a new controller itself chooses the master, the switch is supposed to send role status messages to the old master controller.
- **Flow-monitor:** It informs the controller of a modification in a flow table.

3. **Command messages:** They are sent by the controller in order to collect information and configure switches through their flow tables. it exits three main types of control messages:

    - **Read-state:** They allow the controller to collect information on the plan data such as flow statistics and switch configuration with which it interacts.
    - **Modify-state :** These are the messages that make modification possible for the controller, creating and deleting entries (rules) in flow tables of the Openflow switch.
    - **Packet-out:** These messages come in response to Packet-in messages sent by the switch. They allow to show the switch to be aware on which output port the packet affected by the packet-in message must be routed.

1.8 conclusion

This first part of the chapter transcribes the documentation and research work that was carried out with the objective popularize SDN technology as much as possible, which remains subject to several interpretations. In particular, we have cited and explained the role of the controller within an SDN architecture as well as the operation of the Openflow Protocol.

2 Artificial intelligence

2.1 Introduction

In a daily basis, naturally individuals and so societies in general face problems that are in the search for answers. Same applies to in the computer science field, where multiple problems are being resolved using the classic techniques of problem solving algorithms, with given specific instructions to be followed by the machine. However, many issues were hard to solve and it is considered impossible to make instructions that leads to calculating the results, e.g. facial recognition, diagnosing cancer, driving a car... etc. In such cases, other problem solving techniques have to be taken : Machine learning.

2.2 Machine Learning

Machine learning(ML) is an application of artificial intelligence (AI) that provides the ability to automatically learn and improve from experience without being explicitly programmed, it focuses on the development of computer programs that can access data and use it to learn for themselves. It consists of letting the computer learn which calculations to perform, rather than providing it with those necessary instructions to follow. The implementation of machine learning is a strategic step and requires a lot of resources. Therefore, it is essential to understand what kind of task the Machine Learning algorithm is going to work upon. understanding the different types of perks and flavors they bring to the table is a necessity.

2.3 Types of Machine learning

Machine learning approaches are divided to three main categories as shown in the figure 1.5,

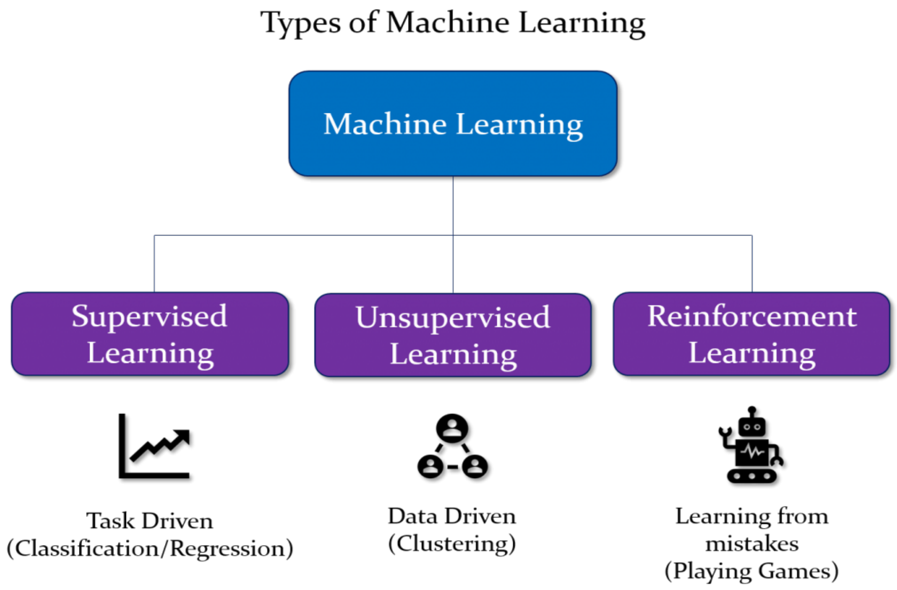
Types of Machine Learning
Machine Learning
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Task Driven (Classification/Regression)
Data Driven (Clustering)
Learning from mistakes (Playing Games)

Figure 1.3: Types of machine learning

2.3.1 Supervised learning

Supervised learning(SL) is one of the most basic types of machine learning, it consists of using labeled datasets to train machine learning algorithms, and to ensure the work of this method the data must be labeled accurately.
In this type, the algorithm is given a small training dataset to work with. This dataset is a part of a bigger dataset and it is used to give the algorithm a general view of the problem, the training dataset has the same characteristics as the final dataset and it provides the algorithm the required parameters to solve the problem, then the machine learning algorithm finds the relationship between the given parameters, basically creating a cause and effect relationship between the variables in the dataset, with the end of the training, the algorithm has a general idea of how the data work and the relation between the input and output.

Now the algorithm is deployed into the final dataset, from which it learns from in the same way as the training dataset. this proves that the supervised machine learning algorithm will continue to learn and improve even after being deployed and that by discovering new patterns and relationships on the new data.

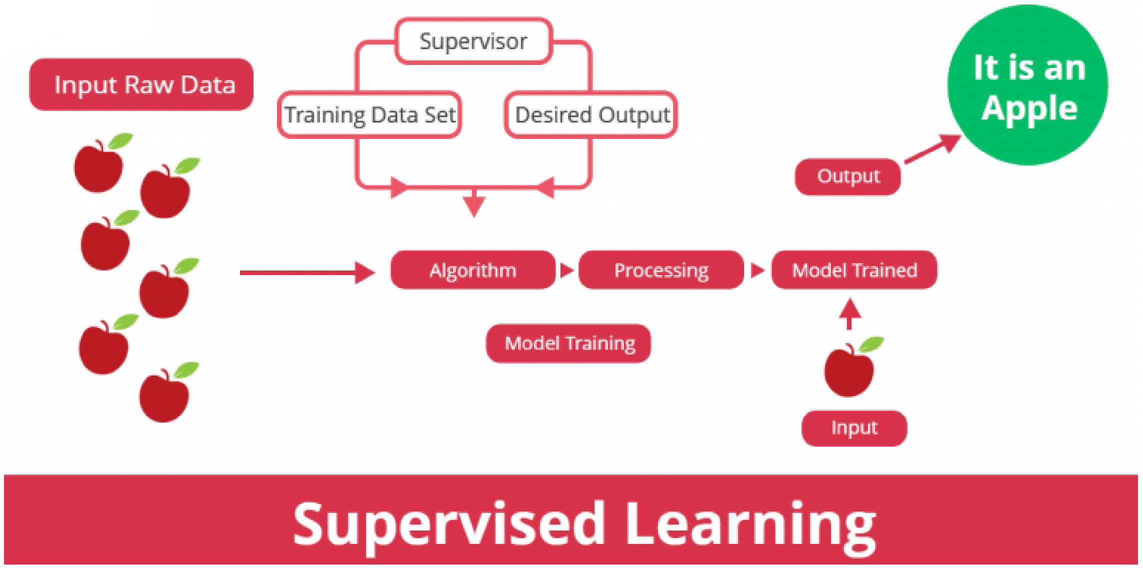

Figure 1.4: Supervised learning

Supervised learning algorithms We find many supervised learning algorithms we can list some of them:

1. K-nearest neighbor
2. Decision Tree (DT)
3. Random forest
4. Neural networks(we talk about the diffrenet NN

   (a) Random NN
   (b) Deep NN
   (c) Convolutional NN
   (d) Recurrent NN

5. Support vector machine (SVM)

6. Naive Bayes

7. Hidden Markov models

2.3.2 Unsupervised learning

Unsupervised learning uses machine learning algorithms to analyze and cluster unlabeled datasets. Unlike supervised learning the human labor is not needed to make the dataset machine-readable. which allows the program to work with much lager datasets.
In Supervised learning the labels let the program know the nature between two data points, however in unsupervised learning there are no labels to work off with. As a result, hidden patterns are created, relationships between data points are perceived in an abstract manner by the program, with no input required from a human being, and the creation of the hidden structures is what makes unsupervised learning algorithms flexible, instead of defining and setting the problem statement, the algorithm can adapt by changing dynamically its hidden patterns which offers more post-deployment improvement than supervised learning algorithms.

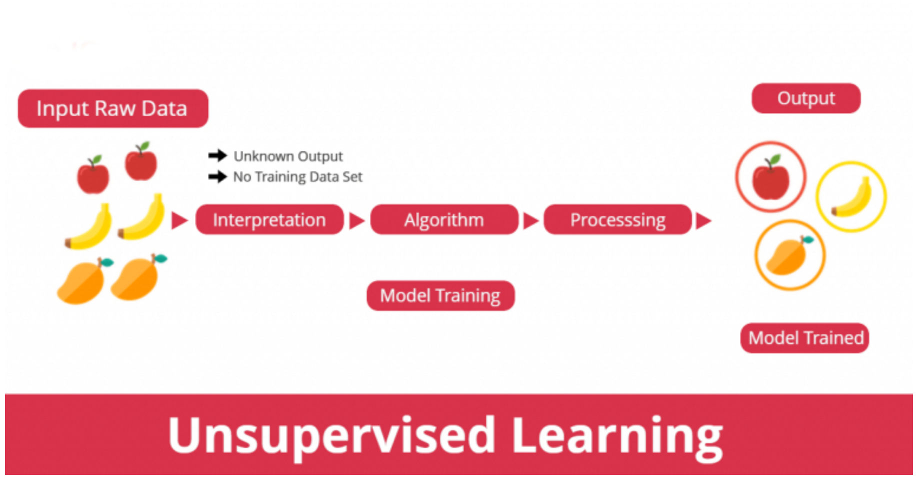

Figure 1.5: unsupervised machine learning example

2.3.3 Reinforcement learning

This form of learning is totally different from the other two. A decision-making agent observes an environment and decides according to its state which action has to be performed. In addition to the agent, this type of learning involves a set of states S and a set A of actions per state. When an action $a \in A$ is performed, the agent transits from a state to another. while executing an action in a state, the agent gets rewarded (the value of the reward could either be positive or negative).
The goal here is for the agent to maximize the reward.

Reinforcement learning differs from supervised and unsupervised learning in not needing labelled input/output pairs be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).[2]

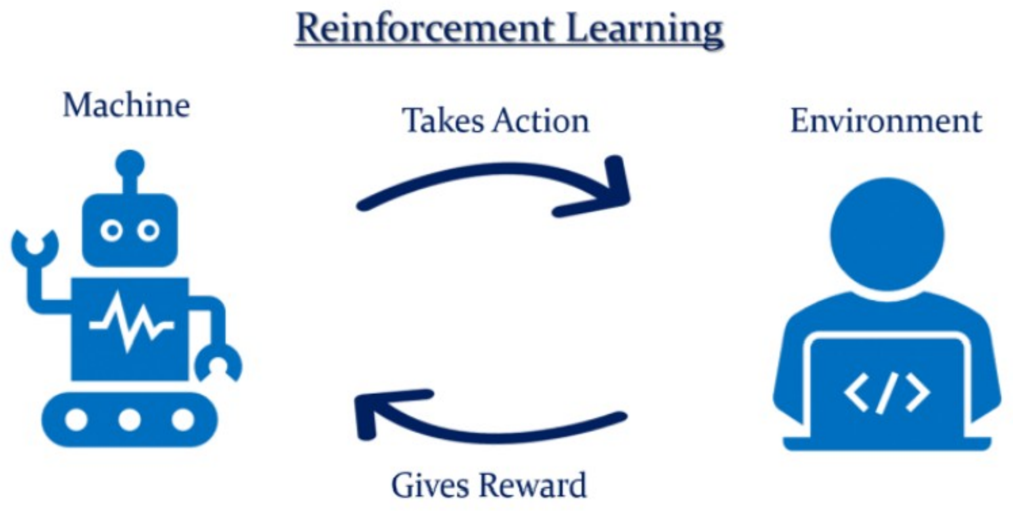

Figure 1.6: The machine keeps on taking actions based on each reward it gets. This process keeps on iterating until a desired level of learning is not reached

2.4 Conclusion

In this chapter, we had an overview on artificial intelligence and machine learning techniques, We've mentioned the three machine learning types and explained the basic functionalities of each type, which will be useful in the next chapters and in proving the reasons behind choosing the solution that we have implemented.

Chapter 2

Routing protocols in SDN

1 introduction

In order to find an optimal path between two given endpoints, we use standard routing protocols. With more and more data transmitted in data center networks, traffic exchanged among switches in data center networks has grown up rapidly, which may lead to the congestion problem and may further result in long delay and low throughput in data center networks.

Nowadays, solutions of multipath routing become funamental requirement to achieve an improvement in the overall throughput of networks. However, due to the dynamic and high volume of traffic, it is also challenging to control the congestion in an efficient way within the networks. It is urgent to deal with this congestion problem using an efficient path optimization algorithm such that the performance of data center networks can be improved. Software-Defined Networking becomes more and more popular to achieve those goals.

2 Different SDN Routing Protocols

2.1 Dynamic Load-Balanced Path Optimization Protocol [3]

The Dynamic Load-Balanced Path Optimization algorithm (DLPO) can be implemented on different SDN-based topologies. The protocol may change paths of flows during flow transmission in order to achieve load balancing among different links resolving the network congestion problem in the data center network. This algorithm consists of two stages [3] :

Path Initializing Stage: The choice of the temporary path according to the bandwidth available on each link, among all the possible paths between the source and destination. The path with the greatest available bandwidth will be chosen as a temporary path.

Dynamic Path Optimization Stage: The algorithm retrieves network statistics using the OpenFlow protocol. If the load of the links in the network is unbalanced $\partial(t) > \partial*$, the optimization algorithm will be triggered to balance link loads that consists of changing paths of flows during flow transmission using the equation below:

$$\partial(t) = \frac{1}{P} \sum_{i=T-P}^{T} \left(\frac{1}{N} \sum_{i=1}^{n} \left(\text{load}^{-}(t) - \text{load}_i(t) \right)^2 \right) \tag{2.1}$$

N : Number of links in the network.
$load^{-}(t)$:average load of all links at time t in a network.
$load_i(t)$: load of link i at time t.
T : current time.
$\partial(t)$: load balancing detection parameter.
P : Past time.

Multi-link DLPO: change the paths of the 10% of the most overloaded links to links with the highest available bandwidth.

If the first algorithm fails (the overload is still present), the second: DLPO single link will be triggered. It redirects the most important flow passing through a overloaded link to a path with maximum available free bandwidth.

2.2 CAMOR Protocol [10]

The Congestion Aware Multi-path Optimal Routing Solution for the support of the congestion problem. This proposed protocol combines multi-path routing and mechanisms adapted to the congestion problem in order to calculate the optimal path for any new data flow. The procedure is described below:

- Using Dijkstra algorithm to calculate the equal cost multiple paths between two endpoints (source and destination)
- Choose the optimal paths among all the selected Pi paths where the links are not not congested and the link speed meets the application speed requirement: calculate the congestion, load and flow available. For this four equations have been used.

$$L_i (i = 1, 2, \ldots n); \forall L_i \in P_n; \tag{2.2}$$

$$L_k = \sum_{i=1}^{n} (K_i) \tag{2.3}$$

$$L_l = \sum_{i=1}^{n} (l_i) \tag{2.4}$$

$$L_t^{P_i} = \frac{L_k - L_l}{L_i} \tag{2.5}$$

- Equation (2.2) defines the total number of links L_i belonging to a set P_i of shorter paths.

ML type	Advantages	Disadvantages
Supervised learning	Strong feature extraction capability from complicated traffic	Requiring a large labeled training data set
Unsupervised learning	Fast algorithm design and no need of a labeled training data set	Lack of accuracy in traffic feature extraction
Reinforcement learning	A direct mapping from traffic distribution to TE policies and no labeled data set requirement	An interaction environment is required

Table 2.1: A brief comparison between ML types routing advantages and disadvantages on SDN

- Equation (2.3) calculates the total capacity L_k of the links.
- Equation (2.4) calculates the total load L_i of the links.
- Equation (2.5) calculates the total available flow $L_t^{P_i}$.

3 Different SDN Routing Protocols using Artificial intelligence

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 section, we review some of the existing machine learning efforts to address issues in SDN.

In SDN, the flow table within switches is adjusted by the controller to manage the routing. For example, the controller can direct switches to reject flows or route it via a particular path. Ineffective routing choices can lead to network links being overloaded, and the end-to-end delivery delay increased, which affects the SDN's performance.

Most SDN routing algorithms still use the shortest path algorithm based on the number of hops to provide the optimized route. However, these algorithms do not consider the bandwidth, delay, resource utilization rate, time, and other factors. Besides, routing issues of optimizing can be called as a decision-making task. Thus, reinforcement learning is a practical approach.

As observed in Table 2.1, we can deduce that that the reinforcement learning type of ML types would be considered as the best methods for a routing protocol in SDN and that goes back to the fact that having a labeled data set is unnecessary which brings a huge advantage. Therefore, we'll be mentioning two ML approaches that are associated with reinforcement learning.

3.1 Deep Reinforcement Learning

DRL model is applied to optimize routing. The objective of the DRL model is to select the optimal routing paths for all source-destination pairs given the traffic matrix to minimize the network delay.

[11] proposed an RL agent that is an off-policy, actor-critic, deterministic policy gradient algorithm that interacts with its environment (i.e. the network) through the exchange of the three signals state, action and reward. Specifically, the state is represented by the Traffic Matrix (TM, being the bandwidth request between each source-destination pair), the action by a tuple of link-weights (that univocally determine the paths for all source-destination node pairs), and the reward is based on the mean network delay.

The agent tries to determine the optimal behavior policy π mapping from the space of states S to the space of actions $A(\pi : S \to A)$ that maximizes the expected reward $r \in \mathbb{R}$ (minimizes the network delay). It does so by iteratively improving its knowledge of the relationship between the three signals by means of two deep neural networks (one for the actor, one for the critic).

Once trained, the DRL agent can produce a near-optimal routing configuration in one single step. On the contrary, traditional optimization requires a large number of steps to produce a new configuration. This represents an important advantage for real-time control of the network.

DRL agents are model-free (they learn autonomously from experience the dynamics between state, action and reward) and can understand non-linear, complex, multi-dimensional systems, without making any simplifications. On the other hand, traditional optimization algorithms require an analytical model of the underlying system deriving from significant assumptions and simplifications.

3.2 Q-Learning

Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence ”model-free”), and it can handle problems with stochastic transitions and rewards without requiring adaptations. It can also be referred to as the classical tabular reinforcement learning, the tabular stores the state-action representations in a table referred to as a Q-table.

QR-SDN a classical tabular reinforcement learning approach that directly represents the routing paths of individual flows in its state-action space. Due to the direct representation of flow routes in the QR-SDN state-action space, QR-SDN is the first reinforcement learning SDN routing approach to enable multiple routing paths between a given source switch–destination switch pair while preserving the flow integrity. That is, in QR-SDN, packets of a given flow take the same routing path, while different flows with the same source-destination switch pair may take different routes.

The QR-SDN approach as proposed in [8] represents the actual flow routes for the Q-Learning reasoning. Therefore, the QR-SDN directly controls the routing, i.e. the reinforcement learning produces a specific routing path to be deployed for a given flow. This direct

Objective	Algorithm	State	Action	Reward
Optimize routing	Q-Learning	Link discovery and classification, Intensive learning training	Link weight (delay and bandwidth)	Conversational services, Streaming media service, Interactive service, Data class service
Controller synchronization	Q-Learning	Time slot counts	Synchronization Decision	Average of reduced path cost
Optimize routing	DRL	Service creation location.	The number of edge node routing requests	Edge node access status, Resource usage balance degree, Request data transmission delay

Table 2.2: Target and learning elements in routing [16]

flow routing avoid the splitting of any individual source-destination host flow and can exploit multiple routing paths for flows sharing the same source-destination switch pair. (more on that in the next chapter)

Topology-based routing approaches make data forwarding decisions based on link information among vehicles. In [14] and [15], fuzzy logic is employed to evaluate wireless links by considering link quality, the available bandwidth and vehicles' mobility. Then, the evaluation results are used by Q-learning algorithm to select the optimal route.

3.3 Conclusion

In this chapter, we had went through a small look on routing different SDN routing protocols including the ones using artificial intelligence, and we've deduced that the most efficient approaches are under the reinforcement learning algorithms, and have chosen to work with the Q-Learning due to its efficiency and the extra factors that it adds that consists on enabling multiple routing paths.

Chapter 3

Study of the chosen solution

1 introduction

Since SDN grants the ability to control the behaviour of the network and that's by allowing software applications to dynamically program the network devices.
In this chapter, we're going to discuss about the steps taken which are:

- Setting up the environment.
- Reinforced Learning Agent implementation.
- Sdn controller implementation (Rerouting + latency monitoring).

2 In-depth view of the conception

Before implementing the Reinforcement learning Agent we have to set the environment (we get the data from the network) to let the agent observe it, this environment consists of 3 parts:the state ,the Actions , finally the reward given for each action taken.

2.1 Setting up the state,action and reward design

2.1.1 State-space design

The network G($\mathcal{V},\mathcal{E}$) with $\mathcal{V}$ as set of Vertices that are connected by the set of edges $\mathcal{E}$ in this study we focus on flows that transmit data from a single sender to a single receiver,and for the data transmission, we take as an example the transport control protocol(TCP) flow, the flow is being transmitted from a sender(source host) s_f to a receiver(destination host) d_f and we refer to it by f, and finally we get the set of all flows that were transmitted $\mathcal{F}$.
Each flow f transfers a prescribed traffic rate R^f from a source host s_f to the network, and we note P_{s_f,d_f} the path between a source host to a destination host which is a sequence of vertices $P = (v_1, v_2, ..., v_n)$ that is included in the set of all possible paths $P \in \mathcal{P}_{s_f,d_f} = (P_{s_f,d_f,1}, P_{s_f,d_f,2}, ...)$ connecting the source and destination hosts, and $\mathcal{P}_{s_f,d_f}$ can be determined by a graph search algorithm such as Depth-First Search (DFS) Algorithms.
The reinforcement learning agent, which operates inside the SDN controller observes the

environment (the network) by measuring the chosen key performance indicators in our case it's the Latency, and this observation consists of the environment's state S_t from the set of states $\mathcal{S} = (S^1, ..., S^n)$ and a reward $R \in \mathbb{R}$. Finally we define the state table that contains the chosen path for each flow and we note:

$$S_t = \begin{cases} f_{s_1,d_1} : & P_{s_1,d_1,t} \\ \vdots \\ f_{s_i,d_i} : & P_{s_i,d_i,t}. \end{cases} \tag{3.1}$$

the state space is basically a key-based dictionary where the flow is the key and the path taken is the value of that key.

2.1.2 Action-space design

the selection of an action $A_t \in \mathcal{A}$ is done according to the current state S_t and its reward R_t, the set of action $\mathcal{A} = A_1, A_2, ...$ is constructed by the set of possible paths $\mathcal{P}$,including the current path for a flow f where $\mathcal{A} = \mathcal{P}_{s_f,d_f}$ so one of the paths is selected either to replace or keep the current value, which means an action basically changes the value(current path) of a key (a flow) in the key-based dictionary (state space).
The action A applied to a single flow is described as

$$A_t = \{f_{s_1,d_1} : P_{s_1,d_1,t} \Rightarrow P_{s_1,d_1,t+1}\} \tag{3.2}$$

There is also another type of action other than *OneFlow*(3.2) that allows to update multiple flows and we refer to it as *Direct Change* and it consists on updating all the flows at a step of time

$$A_t = \begin{cases} \{f_{s_1,d_1} : P_{s_1,d_1,t} \Rightarrow P_{s_1,d_1,t+1}\} \\ \vdots \\ \{f_{s_i,d_i} : P_{s_i,d_i,t} \Rightarrow P_{s_i,d_i,t+1}\} \end{cases} \tag{3.3}$$

but we won't be using the *Direct flow* action mode. as we can see in the equations 3.2 and 3.3 the action A_t has a major impact on the size of the possible action size $|\mathcal{A}|$ which leaves the *Direct flow* approach at disadvantage ,because in this approach the action space scales with the product of the number of the possible paths for the flows where on the other side the *One flow* approach scales with the sum of the numbers of possible paths.

2.1.3 Reward design

We use the Reward R_{t+1} to measure the performance of an Action A_t when solving the flow routing problem, in this study the evaluation is based on the latency which means the lower the latency the higher the reward, also network congestion has also a possibility to happen which generally increases greatly the latency which lowers the reward so the agent automatically tries to avoid that.
The reward R_t consists of the sum of latencies L_f along with the current paths P_{s_f,d_f} of the flows $f \in \mathcal{F}$ and we apply the root square to weigh the outliers relatively heavily

$$R_t = -\sqrt{\frac{\sum_{\forall f \in \mathcal{F}} L_f^2}{|\mathcal{F}|}} \tag{3.4}$$

and since the reinforcement learning agent tries to maximize the reward and so we put the minus, after all a higher latency lowers the quality of service.

2.2 Reinforcement learning agent implementation

the Agent has two parts , the learning part and the exploration method and that's what we are discussing in this section:

2.2.1 Q-Learning

the Q-value $Q\left(S_t, A_t\right) \in (-\infty, 0)$ is a quality measure of a taken action A_t in a state S_t at a time t, Q-learning is based on an iterative update rule as shown in the equation:

$$\underbrace{Q\left(St, At\right)}_{\text{new Q-value}} \leftarrow (1-\alpha) \underbrace{Q\left(St, At\right)}_{\text{old Q-value}} + \alpha(R_{t+1} + \gamma \quad \underbrace{\max Q\left(S_{t+1}, a\right)}_{\text{expected value of future state}} \tag{3.5}$$

with:

- α : learning rate.
- γ : discount factor.
- R_{t+1} : observed reward.

and so α is the learning rate that determines how quickly the agent adopts and uses the new learned Q-Values, the discount factor γ expresses how the future expected rewards will be used and taken. The future expected rewards ($\max_{a \in \mathcal{A}} Q\left(S_{t+1}, a\right)$) is basically how much the reward we can get if the highest valued action a is taken.
Each state-action pair and its corresponding Q-value has to get saved in a tabular data structure and here we used nested dictionaries where the states S are the keys, and the values are also dictionaries with the actions as a keys and the Q-values as values of the actions , as shown in (3.6).

$$\text{Q-Table } = \begin{cases} S^1 : \begin{cases} A^1 : Q\left(S^1, A^1\right) \\ \vdots \\ A^j : Q\left(S^1, A^j\right) \end{cases} \\ \vdots \\ S^i : \begin{cases} A^1 : Q\left(S^i, A^1\right) \\ \vdots \\ A^j : Q\left(S^i, A^j\right). \end{cases} \end{cases} \tag{3.6}$$

and since the selection of the action is done according to the highest Q-value , we initialized the Q-value in the Q-table to $-\infty$ and we use at first a random routing action.

2.2.2 Exploration modes

Reinforced learning needs exploration strategies to test out new action-state combination. Therefore multiple strategies are possible:

- Epsilon greedy($\epsilon - greedy$).
- *Softmax*.
- *Upper Confidence Bound* (UCB).

where:
The $\epsilon - greedy$ strategy consists on choosing the action with the highest Q-value

$$A_t \doteq \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q_t(s, a), \tag{3.7}$$

where $\epsilon \in [0, 1]$ is the probability of choosing a random action, which helps to explore new states but lowers the exploited reward.
The *Softmax* strategy converts the Q-values into selection probabilities for each action of the states following the equation: 3.8

$$\Pr \{A_t = a\} = \frac{\exp[-1/(Q(s,a) \cdot \tau)]}{\sum_{b \in \mathcal{A}} \exp[-1/(Q(s,b) \cdot \tau)]} \tag{3.8}$$

this strategy is controlled by the *temperature* θ, a low θ favors exploitation which means the action with highest Q-value is chosen more often, in contrast a high *temperature* pushes the agent to explore more possibilities to acquire new knowledge.
the last strategy *Upper Confidence Bound* (UCB) focuses on reducing exploration over time and that by continuously reducing ϵ and the *temperature*(θ), it follows the same concept as *annealing* which allows to move smoothly from exploration to exploitation, it basically controls the exploration by counting how many times the action has been chosen so a bonus b^+ is added to the equation of the $\epsilon - greedy$ 3.9. Therefore the actions are now chosen based not only on their Q-value but also with their potential.

$$A_t \doteq \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left(Q_t(s, a) + cb^+ \right). \tag{3.9}$$

That bonus b^+ is decreased over time following the formula $b^+ = \sqrt{\ln N(s)/N(s,a)}$. where $N(s)$ represents the number of times a state has been selected, and $N(s,a)$ represents the selection counter of the combination action-state, the degree of exploration can be adjusted with the parameter $c > 0$, by which a higher c gives the bonus b^+ more strength which leads to more exploration.

2.3 SDN Controller implementation

The implementation of the centralized SDN controller was done using the *Ryu* framework. The role of the controller is to perform three tasks simultaneously: the reinforcement learning (Learning agent), latency monitoring and the rerouting.

2.3.1 Rerouting

As one of the advantages of SDN is the ability to modify the routing paths of the traffic flows while running without causing any packets losses or down time. So, we need to update the flow routes, so we used a Rerouting algorithm. [8]

Algorithm 1 (Re)routing Algorithm

```
Input: $OLDPATH$, $NEWPATH$, $FLOWID$
/* New switches on the path */
$flowAddList \leftarrow []$ /* Modify routes of switches on the path */
$flowModList \leftarrow []$ for $index, switch$ in $ENUMERATEnewPath$ do
    if $switch$ in $oldPath$ then
        $oldIndex \leftarrow GetIndexoldPath, switch$ if $oldPath[oldIndex - 1] == newPath[index - 1]$ then
            continue
        else
            if $newPath[oldIndex - 1]$ not in $flowAddList$ then
                $flowModList \leftarrow flowModList$+newPath[oldIndex - 1]
            end
        end
    else
        $flowAddList \leftarrow flowAddList+switch$ if $newPath[oldIndex - 1]$ not in $flowAddList$ then
            $flowModList \leftarrow flowModList$+newPath[oldIndex - 1]
        end
    end
end
/* Add forwarding rule for flow to new switches */
for $switch$ in $flowAddList$ do
    $nextSwitch \leftarrow newPath[GETINDEXNewPath, switch + 1]$
    $AddFlowSWITCHswitch, flowID, nextswitch$
end
/* Modify forwarding rule for flow of persistent switches */
for $switch$ in $REVERSEDflowModList$ do
    $nextSwitch \leftarrow newPath[GETINDEXNewPath, switch + 1]$
    $ModFlowSWITCHswitch, flowID, nextswitch$
end
$flowDelList \leftarrow SETDIFFoldPath, newPath$ /* Delete forwarding rule for flow
    of old switches */
for $switch$ in $flowDelList$ do
    $DELFLOWSWITCHswitch, flowID$
end
```

the goal here is to modify the forwarding rules without interrupting the existing traffic flows. The approach is to iterate over a new path, which is a list suggested by the rein-

forcement learning agent, and compare the new path with the old one. If the new path and the old path share the same path (switches in the correct sequence), then nothing will change. If there is a new switch, it will be added to *flowAddList* and its predecessor switch will be added to *flowModList*. In addition, if the old and new switches don't share the same predecessor switch, then the forwarding rule has to be modified, and finally the predecessor switch is added to *flowModList*. And to avoid any useless connection, the forwarding rules are first added to new switches then it is changed in the existing ones, but starting from the destination to the source to avoid gaps in the routing path.
These modifications of the forwarding rules rely on the information gathered from the topology and the relations between the ports and the switches. And these informations are gathered by the Latency monitoring thread and used to determine the outgoing port to reach the succeeding switch on the new path. The manipulation of forwarding rule is implemented with Openflow's *FlowMod* command.
And Finally, the unused switches on the old path have to get removed. In order to do that, we first compare the old path with the new path to build and store the relative of the old path in the *flowDelList*. Eventually, each forwarding rule per switch in the *flowDelList* is deleted.

2.3.2 Latency monitoring

This part is responsible for the gathering of link latency statistics. The used approach creates artificial Ethernet (probing) packets [7]. The payload contains a timestamp and the datapath ID from the source switch(sending switch). The packets are padded with zeros to reach the minimum Ethernet packet size of 64Bytes. These packets are broadcasted every second to neighboring switches. The advantage of this approach is that the latency that is being measured is the same as the latency perceived by the flows. The packet probes experience the packet processing, queuing, transmission, and propagation delays on the one-way path from the source host to the destination host. The dropped probe packets are ignored, only the packets that arrived to its destination are considered in the latency measurement. The latency measurement is done every second, on the other hand the learning step is done every two seconds to allow the latency monitoring and the learning process to operate asynchronously. And by measuring the latency twice as frequently as the learning steps, we ensure that the measured values are up to date. The 2 seconds duration of one learning step is defined as step time T_{step}.
We know that packet reordering may occur when transitioning from an old routing to a new routing configuration, and that lowers the performance of the network [13]. To avoid that, the routing configuration changes should be minimized, and basically to achieve that minimization, we need to minimize the time till convergence since it reflects the routing configuration changes required by a learning approach.

Chapter 4

Implementation and Evaluation of the work

1 Introduction

In this chapter, we going to discuss about the hardware and software used to implement the program, the working method and then its evaluation

2 Working environment

In this part,we will include the technologies used for the implementation, with the programming language, libraries, and frameworks

2.1 Programming language

The programming language used is Python, which is an interpreted and high-level programming language. It was chosen for many reasons:

– It runs on multiples operating systems(Windows, linux..)

– It supports procedural and object-oriented programming at the same time.

– Ryu framework and Mininet both supports python

– It contains a huge number of useful libraries

– Dynamic declaration of variables without specifying the types

2.2 Libraries

Among several python libraries, we will cite some important ones that we used in the program:

– Numpy: a python library that is used to handle multidimensional arrays and matrices(creation, multiplication, transpose, ...)

– Json: a python library which allows decoding and encoding of Json files

– Mathplotlib: a comprehensive library for creating static, animated, and interactive visualizations in Python,

2.3 Framework and emulator

For this project, we used two main tools which are Ryu framework, and Mininet emulator where:

– Ryu is a component-based software defined networking framework [9], that provides software components with well defined API's which makes it easy for developers to create new network management and control applications. Python is supported by Ryu and it includes multiple packages that were used in the program

– Mininet is a network emulator that creates a network of hosts, switches, controllers and virtual links[4].

2.4 Hardware specifications

The testing and evaluation were carried out on one PC where we have set a virtual machine with a Linux operating system
the specification are as follows:

PC	CPU	RAM	GPU	Operating system
ASUS ROG GL551VW DS51	I5-6300HQ	8	Nvidia GTX 960M	Windows 10 pro

For the virtual machine:

- Linux version : Ubuntu 18.04
- RAM given : 4GO
- Python version: 3.8

2.5 IDE (Pycharm)

PyCharm is an integrated development environment used in computer programming, specifically for python programming. It is a cross-platform IDE supported by windows, linux and macOS.

3 Solution implementation

The program is split up to two main part which are the implementation of our testing topology using Mininet and the implementation of the controller with the learning agent included

3.1 Mininet topology

We split this part to two sections where in the first one we will discuss about the methods we used to create the topology,and in the second one we talk about how we generated the flow.

3.1.1 Adding SDN components

we got 4 main components to add

- the SDN controller $C0$
- the switches S_n
- the hosts H_n
- the links : which are divided to two parts:
  - Switch-Switch links which have latency and bandwidth as parameters
  - Host-switch links with negligible latency

3.1.2 Flow generation

To generate the flow we used *Iperf*, which is an open source software written in C. It runs on a variety of platforms, including Linux, Windows, and Mac. It is a widely used tool for measuring and tuning network performance. It can produce performance metrics for any network.
Since in our study we focus on single source to single receiver connection we used *Iperf* to create a flow between two single hosts

3.2 Controller thread

This part is split up to three parts: the Learning module, the Controller and the configuration module where we specified the used configuration and parameters.

3.2.1 Configuration thread

In this file, we initialized multiple parameters which are:

- learning rate α (With value of 0.8)
- discount factor γ (With value of 0.8)
- the exploration probability ϵ (varries for each step of the evaluation)
- the temperature θ for the SOFTMAX approach
- the exploration degree for the UCB approach

and we also specified the exploration strategy and have set the path where to save to results.

3.2.2 Learning agent

The learning agent is implemented as a single module and it is called in the remote controller. The agent parameters is taken from the config file.

3.2.3 Controller

The controller was already implemented by Rischke [8], which includes the routing protocol and the modified remote controller

4 Evaluation

In this section, we will evaluate the study with three approaches. The first we set a static epsilon and we compare the exploration modes in the same topology. The second approach is to fix the exploration mode and modify the exploration probability (ϵ). Finally, we test the scalability of our program with a fixed exploration strategy.

4.1 Comparison between exploration strategies

For the network topology we have chosen the six switches and 8 hosts topology as shown in the figure(4.1)

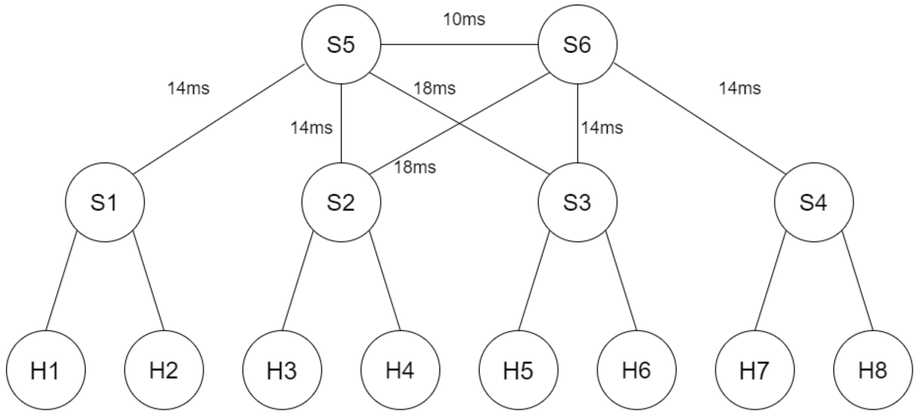

Figure 4.1: Six switches 8 hosts topology with link latency

and we fixed the exploration probability ($\epsilon = 0.2$), and the result are as follows:

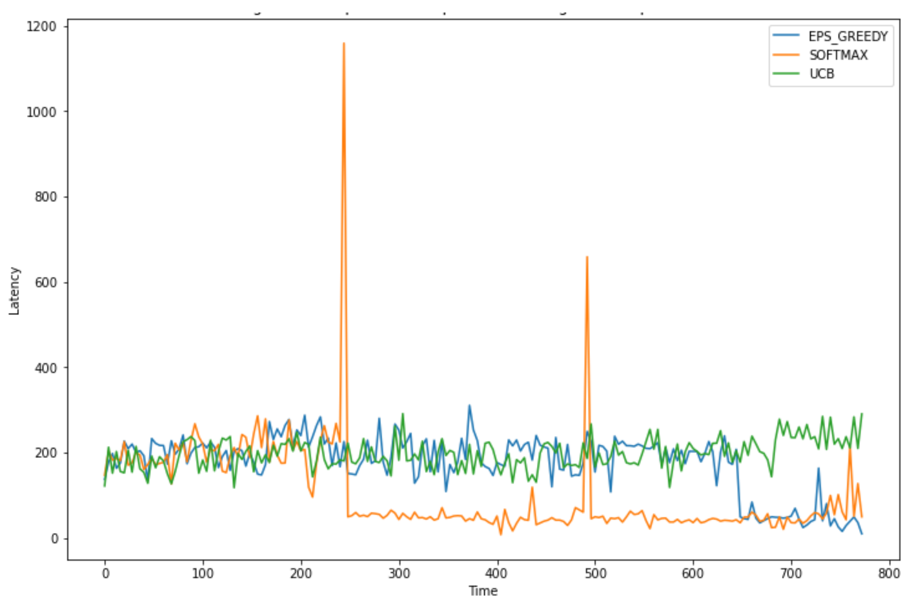

Figure 4.2: Average latency comparison between strategies with $\epsilon = 0.2$

for the $SOFTMAX$ strategy, the temperature θ was set to 0.05 and the exploration degree was set to 30 (for UCB strategy).
as the graph shows, the three strategies are similar at the beginning, but the $SOFTMAX$ approach gets better result as the steps augment, for the latency spike it is due to the random action that the agent chose which resulted in a longer route with a higher latency. As we advance further in time, the $\epsilon-greedy$ strategy results converges to even lower latency due to the exploitation of the previous results.

4.2 Exploration and Exploitation comparison

In this Part we used the previous topology(figure 4.1) combined with the $SOFTMAX$ strategy with different exploration probability values(0.5 , 0.4 , 0.2) to alter between the exploration and exploitation, and there are the results

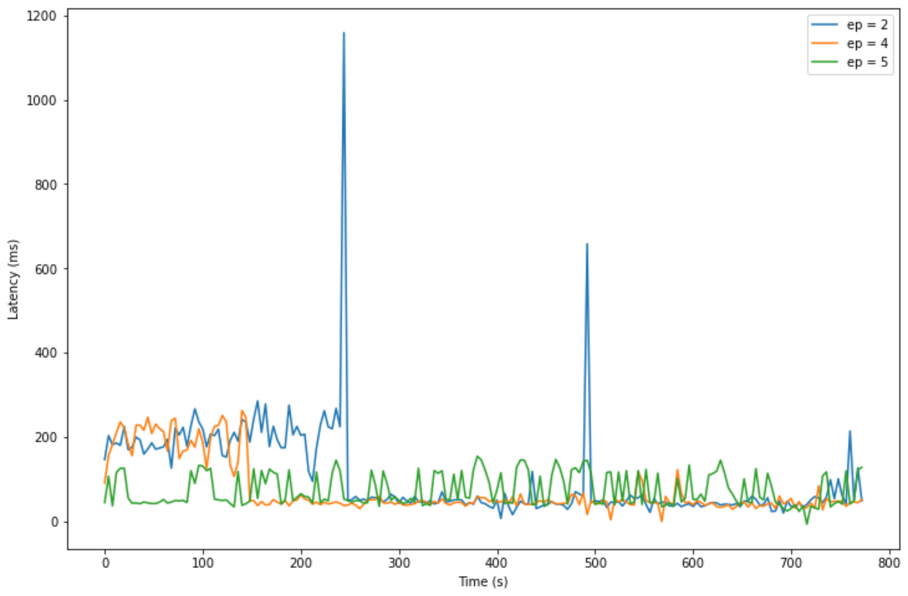

Figure 4.3: Exploration vs Exploitation

4.3 Scalability testing

For this test we created 3 different topologies with scalability variation as shown in the figures(4switches-6hosts(fig4.4),6switches-8hosts(fig4.1),8switches-10hosts(fig4.5)), the 6switches-8hosts topology has been modified and a static bandwidth($bd = 3Mb/s$) was added to links.

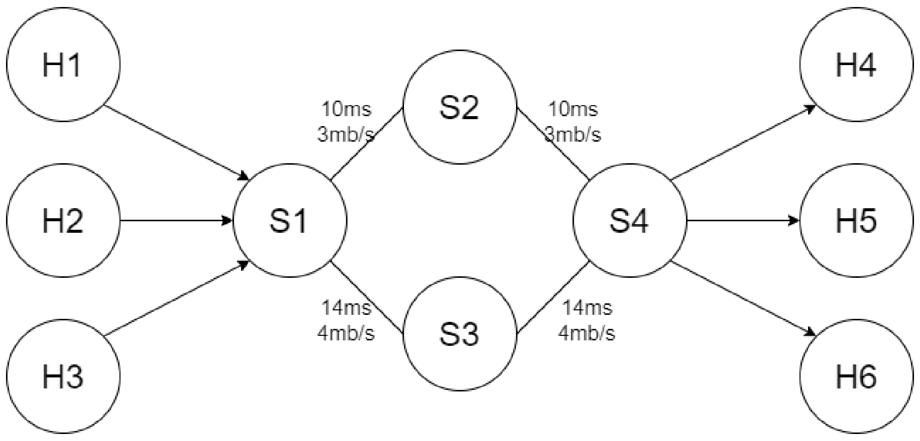

Figure 4.4: Four switches six hosts topology

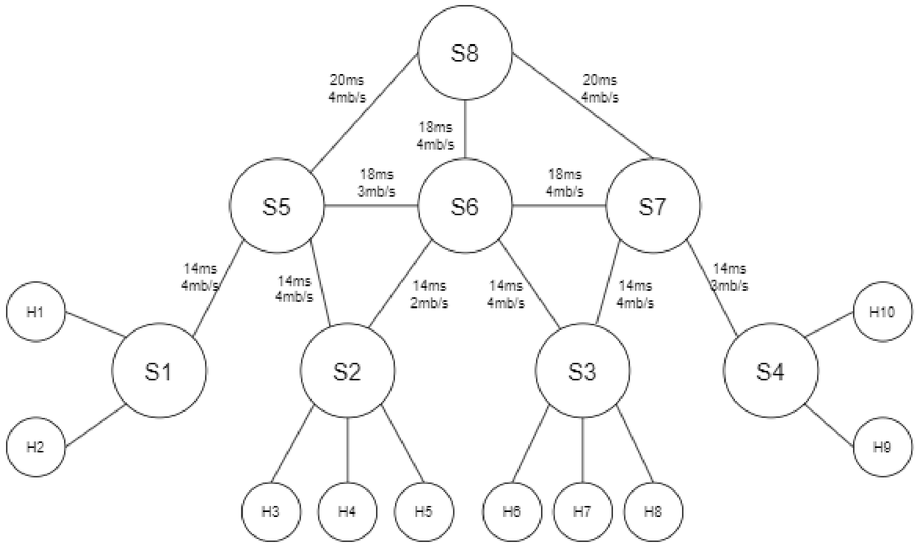

Figure 4.5: Eight switches ten hosts topology

First, we compared only two topologies :

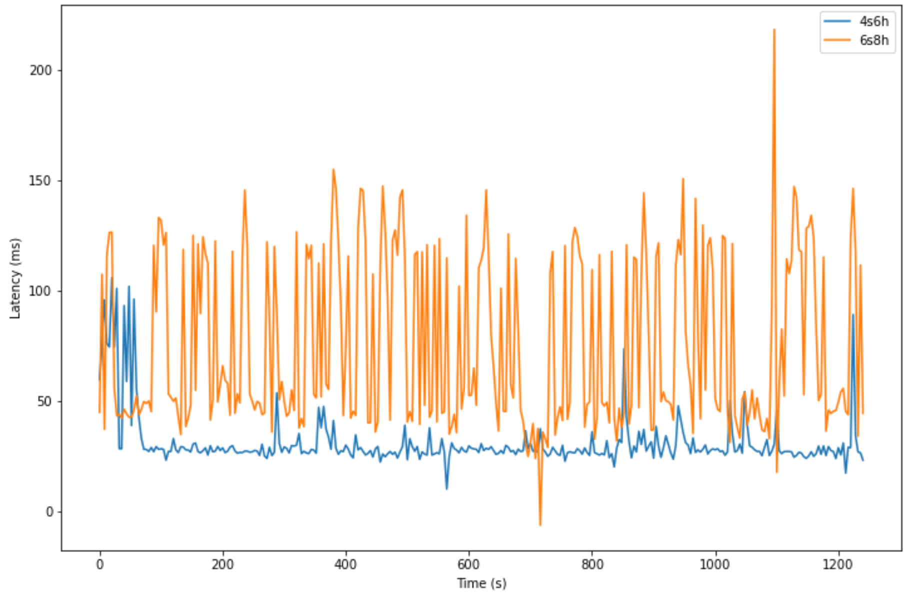

Figure 4.6: 4 switches 6 hosts and 6 switches and 8 hosts comparison

This comparison shows the scalability problem in q-learning ,which shows that the larger the topology, the more the protocol works less efficiently, and we can confirm it with last comparison where we compared all three topologies, and the results are:

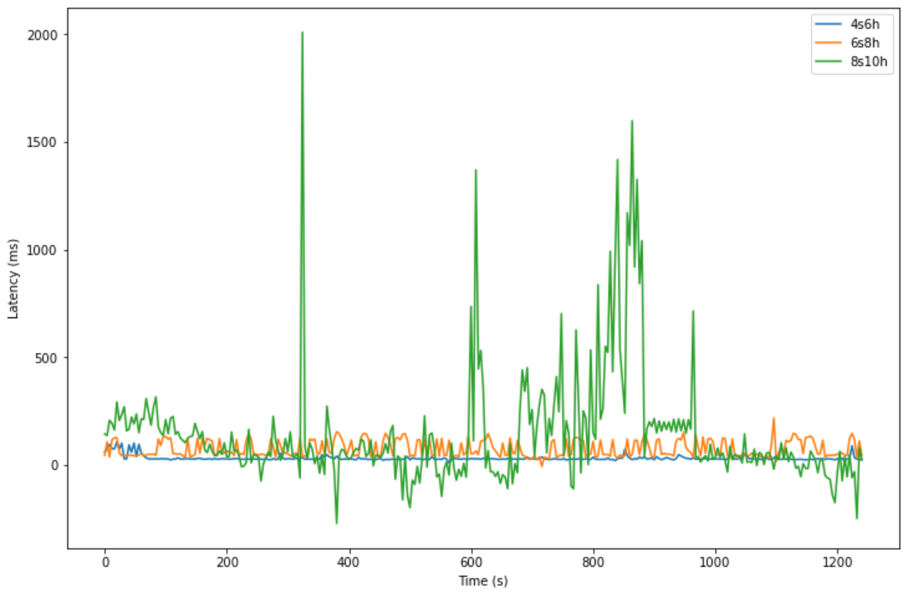

Figure 4.7: All existing topologies comparison

As a conclusion the flow routing using Q-learning is highly effective in achieving low latencies and preserving the flow integrity in small networks, which isn't the case in large networks.

General Conclusion

In this project, we went through multiple chapters, starting from the state of art, where we introduced and defined the SDN and the artificial intelligence. Then we mentioned some previous works and protocols with and without AI implemented in SDN. We also brought up the conception of the program, finally, the implementation and the evaluation of our study.

The goal of this study was to implement a routing protocol using artificial intelligence in SDN, while trying to solve routing problems such as latency and packet losses. The main objective was to achieve lower latency, but we have faced many problems while implementing the program, Not only the program was implemented on a virutal machine but also it is favorable to test in far more powerful hardware for longer periods.

Even though, multiple simulations and tests were taken. And the results were promising, that shows how effective the Q-learning in small SDN topologies is, where the average latency have taken the values $[5 - 10ms]$, which is a good latency in my opinion.

The program can still be improved and optimized in future studies, and we suggest the usage of neural networks in the agent for a better path selection which results to even lower latency.

Finally, this study helped us understand how the SDN works and we are looking forward to create our own routing protocol using AI in the future.

References

[1] HAMENNICHE Amira ACHAIBOU Radia Amina. "Gestion des traffic dans les Réseaux SDNs". In: 2020.

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement learning: A survey". In: *Journal of Artificial Intelligence Research* 4 (1996), pp. 237–285.

[3] Yuan-Liang Lan, Kuochen Wang, and Yi-Huai Hsu. "Dynamic load-balanced path optimization in SDN-based data center networks". In: *2016 10th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*. IEEE. 2016, pp. 1–6.

[4] *Mininet*. URL: `http://mininet.org/`. (accessed: 09.07.2021).

[5] Sangeeta Mittal. "Performance Evaluation of Openflow SDN Controllers". In: Mar. 2018, pp. 913–923. ISBN: 978-3-319-76347-7. DOI: `10.1007/978-3-319-76348-4_87`.

[6] Amitava Mukherjee et al. "Fault Tracking Framework for Software-Defined Networking (SDN)". In: Feb. 2017. ISBN: 9781522520245. DOI: `10.4018/978-1-5225-2023-8.ch011`.

[7] Kevin Phemius and Mathieu Bouet. "Monitoring latency with openflow". In: *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*. IEEE. 2013, pp. 122–125.

[8] Justus Rischke et al. "Qr-sdn: towards reinforcement learning states, actions, and rewards for direct flow routing in software-defined networks". In: *IEEE Access* 8 (2020), pp. 174773–174791.

[9] *RYU Sdn*. URL: `https://ryu-sdn.org/`. (accessed: 09.07.2021).

[10] Syed Asif Raza Shah et al. "CAMOR: congestion aware multipath optimal routing solution by using software-defined networking". In: *2017 International Conference on Platform Technology and Service (PlatCon)*. IEEE. 2017, pp. 1–6.

[11] Giorgio Stampa et al. "A deep-reinforcement learning approach for software-defined networking routing optimization". In: *arXiv preprint arXiv:1709.07080* (2017).

[12] M. Tury. "Les risques d'OpenFlow et du SDN. 2015". In: (2015).

[13] Yi Wang, Guohan Lu, and Xing Li. "A study of Internet packet reordering". In: *International Conference on Information Networking*. Springer. 2004, pp. 350–359.

[14] Celimuge Wu, Satoshi Ohzahata, and Toshihiko Kato. "Flexible, portable, and practicable solution for routing in VANETs: A fuzzy constraint Q-learning approach". In: *IEEE Transactions on Vehicular Technology* 62.9 (2013), pp. 4251–4263.

[15] Celimuge Wu, Satoshi Ohzahata, and Toshihiko Kato. “Routing in vanets: A fuzzy constraint q-learning approach”. In: *2012 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2012, pp. 195–200.

[16] Junfeng Xie et al. “A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges”. In: *IEEE Communications Surveys Tutorials* 21.1 (2019), pp. 393–430. DOI: 10.1109/COMST.2018.2866942.