LEARNING-BASED ORCHESTRATOR FOR INTELLIGENT SOFTWARE-DEFINED NETWORKING CONTROLLERS

Imene Elloumi Zitouna

University of Tunis El Manar, École Nationale d'Ingénieurs de Tunis, Laboratoire Systems de Communications, Tunisia

ABSTRACT

This paper presents an overview of our learning-based orchestrator for intelligent Open vSwitch that we present this using Machine Learning in Software-Defined Networking technology. The first task consists of extracting relevant information from the Data flow generated from a SDN and using them to learn, to predict and to accurately identify the optimal destination OVS using Reinforcement Learning and Q-Learning Algorithm. The second task consists to select this using our hybrid orchestrator the optimal Intelligent SDN controllers with Supervised Learning. Therefore, we propose as a solution using Intelligent Software-Defined Networking controllers (SDN) frameworks, OpenFlow deployments and a new intelligent hybrid Orchestration for multi SDN controllers. After that, we feeded these feature to a Convolutional Neural Network model to separate the classes that we're working on. The result was very promising the model achieved an accuracy of 72.7% on a database of 16 classes. In any case, this paper sheds light to researchers looking for the trade-offs between SDN performance and IA customization

KEYWORDS

Open vSwitch OVS, Artificial Intelligence, Machine Learning, Supervided Learning, Reinforcement Learning, Hybrid Orckestrator, Openflow, QoS, QoE, Real time, User Behavior, User Engagements, Intelligent Software-Defined Networking ISDN.

1. INTRODUCTION

With the development of Software-Defined Anything, software is already penetrating into the Internet and even controlling the network. For example, a software developer can make a service/application at the application layer, and directly obtain the original data of the network device in the physical layer using southbound APIs [1]. Software technologies are already changing traditional network services that are typically provided by servers via application layer protocols into new network services that are provided from the network infrastructure (e.g., firewall service, QoS service). The network infrastructure also can provide services through open APIs, allowing network services to benefit operationally by enabling automated provisioning of network applications with different characteristics. This also helps to ensure that specific applications are getting the proper network resources that are dynamically allocated to meet service requirements (QoS, QoE, encryption, etc.) [1]. Due to these advances, many new service models are emerging in the field of networking (SDN/NFV/SD-WAN).

Artificial Intelligence (AI) and Machine Learning (ML) approaches have emerged in the networking domain with great promise. They can be clustered into AI/ML techniques for network engineering and management, network design for AI/ML applications, and system aspects. AI/ML techniques for network management, operations and automation improve the way we

DOI: 10.5121/ijsea.2020.11602

address networking today [10] [11] [12]. They support efficient, rapid, and trustworthy management operations. Machine learning (ML), is a key feature of future networks mainly the SDN paradigm.

Our work and interest in softwarization and network programmability fuels the need for improved network automation, including edge and fog environments. Moreover, network design and optimization for AI/ML applications address the complementary topic of supporting AI/ML-based systems through novel networking techniques. Including new architectures and performance models with great predictive ability in order to guarantee the continuity of QoS/QoE [2] [3].

In this paper, we elaborate an alternate method and enhancements provided. For the purpose of tackling the current difficulties, we propose an innovative solution based on the application of Artificial Intelligence and supervised and reinforcement Machine Learning techniques in SDN. Through SDN network virtualization, its layered architecture and its programmable interfaces, overall network control is therefore ensured by the logical centralization of the control function in the SDN controllers. Network flows are thus controlled using the OpenFlow protocol. In this work we present, a new reinforcement learning model in an SDN infrastructure. Which adds "learning" to the control function for better intelligent decision-making at the level of SDN controllers [7] [8].

These results will ultimately bring us a step closer to overcoming the problem of over exploitation of network resources, and how they can be deployed more efficiently.

One of the challenges that every machine learning algorithm is faced with is scalability and validity to large datasets. Our research devoted to applying, hybrid machine learning, which use an orchestrator and a set of intelligent controllers SDN combine supervised and reinforcement learning techniques, can improve the key performance metric as well as QoS of different real time applications and the QoE. Furthermore, prediction methods for behavior and engagements user combining classification techniques have the potential for creating more accurate results than the individual methods, particularly for large datasets.

The remainder of this paper is structured as follow: section II discusses a challenge, section III ISDN and orchestrator quality requirements, section IV describes our approach, section V presents design of reinforcement learning framework of ISDN, section VI describes the experimental results of supervised learning of the hybrid orchestrator, testbed, model context, dataset, data preparation, model selection and training, model creation and fine-tuning. Finally, section VII conclude the paper.

2. CHALLENGE

Adoption of the Service-Oriented Architecture principle in networking has enabled the Network-as-a-Service paradigm that is expected to play a crucial role. Network services significantly impact the performance of higher layer services/applications. The behavior of an end-to-end network service is the result of the combination of the individual network function behaviors as well as the behaviors of the network infrastructure composition mechanism.

However, this emerging network services are usually are usually compositions of multiple service components, APIs or network functions, based on new mechanisms, such as service mesh, internet of services, and service function chaining, running on the network layer and/or application layer. Meanwhile, machine learning (ML) has seen great success in solving problems

from various domains. In order to efficiently organize, manage, maintain and optimize networking systems, more intelligence needs to be deployed.

In this case, can we apply the machine learning algorithms in the realm of SDN, from the perspective of traffic classification, routing optimization, quality of service/quality of experience prediction and resource management ? How can SDN/ML brings us new chances to provide intelligence inside the networks ?

The logically centralized control, global view of the network, software-based traffic analysis, and dynamic updating of forwarding rules, of SDN make it easier to apply our solution of machine learning techniques in the SDN: "The Intelligent Software defined networking, ISDN".

It is believed that ML has high potential the aforementioned challenges in SDN-based networks, dynamic service provisioning and adaptive traffic control, as ML is a technology that can effectively extract the knowledge from data, and then accurately predict future resource requirements of each virtualized software-based appliance and future service demands of each user. Therefore, our research tends towards the proposal of a new hybrid orchestrator which learns the optimal ISDN controller in a multi-ISDN environment.

In another context, in the absence of a mechanism for evaluating the best SDN controller and best OVS in terms of response time in network congestion situations, centralization at the SDN controller level operated by OpenFlow increases these response times. Consequently, this will allow any attacker who has gained access to the administration network to act at any time on the configuration of the OVS.

It is also conceivable for an attacker to place himself in a man-in-the-middle in order to filter the commands of a legitimate administrator as well as the feedback of information from the OVS. Finally, listening to unencrypted or encrypted OpenFlow messages passively provides a great deal of information about the networks maintained by the controller.

Since the memory resources of the controller are likely to be greater than those of the OVS. This attack model presupposes the possibility of recording flows on the controller. There is a higher risk compared to the system master-slaves.

Therefore, without an intelligent prediction mechanism, it is simple for an opposing SDN controller to usurp the role of master, or to prevent a legitimate SDN controller who wishes to be master can configure the OVS, or at least have processing times that increase with the traffic rate.

On the other hand, we can measure even longer processing times for large data flows. The goal of our work is to be able to integrate intelligence into SDN controllers through the application of machine learning algorithms in terms of routing optimization, traffic classification and resource management.

3. ISDN AND ORCHESTRATOR QUALITY REQUIREMENTS

The QoS refers to technologies that manages network traffic in order to reduce packet loss, latency and jitter. They control and manage network resources by defining priorities for specific forms of information on the network. The QoE is a measurement that determines user behavior and user engagements and how satisfied the end user is network services [15]. Contrary to QoS, QoE takes considers the end-to-end connection and applications presently running over that network connection and how multimedia elements are meeting the end user’s demands.

- QoS metrics of OVS Flow table [2] [3] [4]:

  - PacketIn: this event is raised whenever the controller receives an OpenFlow packet-in message from the OVS. This indicates that a packet has arrived at the OVS port and the OVS either does not know how to handle this packet or the matching rule action implies sending the packet to the controller. This is where the controller performs the overhead calculation by incrementing the overhead when the OVS forwards a packet.

  - UpStream: this event is triggered in response to the establishment of a new control channel with a OVS.

  - DownStream: this event is fired when a connection to a OVS is terminated, either explicitly or because of OVS rebooting. We utilize this event handler to output the final value of measured control overhead.

  - TCAM: the OVS support the wire-rate access control list (ACL) and QoS feature with use of the ternary content addressable memory (TCAM). The enablement of ACLs and policies does not decrease the switching or routing performance of the OVS as long as the ACLs are fully loaded in the TCAM. If the TCAM is exhausted, the packets may be forwarded via the CPU path, which can decrease performance for those packets. Therefore, our learning can be done with the (QoS and ACL) TCAM percentage of the switches. This value can be used to determine the reward of the learning agent for the OVS 1.

  - Throughput: we aim to measure the maximum flow setup rate a controller can handle per period of time. Being aware of the amount of packet-in a controller can process ease the choice of SDN controllers appropriate to master network control charge.

  - Latency: represents the time consumed by a controller to process and reply an outstanding flow request from the OVS.

  - Scalability: represents the capacity of a controller to handle a large network with an increased number of connected hosts without degrading throughput and latency performance.

  - ICMP Round Trip Time (RTT): it is important to evaluate the influence of additional delay due to communications between OVS and SDN controller for the first ICMP Echo packet we target the selection of a controller that shows the lower delay.

  - TCP and UDP measurements: the purpose of TCP and UDP measurements is to estimate the impact of delay introduced by the communication between controller and OVS on the TCP transfer time and UDP packet losses. TCP and UDP evaluation are performed over multiple network topologies by Mininet. Single, linear, tree and data center network (DCN) topologies.

- User behavior [5] [6] [9]: is the tracking, collecting and assessing of user data and activities using monitoring systems. Our purpuse is to determine a baseline of normal activities specific to the organization and its individual users. They can also be used to identify deviations from normal. We propose to uses machine learning algorithms to assess these deviations of user behavior in near-real time.

- User engagements [5] [6]: is a process comprised of four distinct stages: 1) point of engagement, 2) period of sustained engagement, 3) disengagement, and 4) reengagement. The period of Engagement Attributes are: Aesthetic and Sensory Appeal, Attention, Awareness, Control, Interactivity, Novelty, Challenge, Feedback, Interest and Positive Affect. Also we propose to uses machine learning algorithms for specification the new SLA of user engagement

4. APPROACH OF LEARNING ORCHESTRATOR

Our intention would be to design an SDN architecture composed of multiple controllers where the choice of the suitable and optimal controller and OVS to be implemented in a specific domain depends on the controller performance and efficiency to respond to QoS and QoE requirement imposed by the service application with machine learning environment.

We extract relevant information from the Data flow generated from a SDN (PacketIn, UpStream, DownStream, TCAM, etc.) and using them to learning, to predicting and to accurately identify the optimal destination OVS using Reinforcement Learning. Also we extract relevant information from the Data flow generated from the optimal Intelligent Software-Defined Networking controllers (CPU Throughput, CPU Latency, Scalability, ICMP Round Trip Time (RTT), TCP and UDP measurements, etc) with Supervised Learning run in the Hybrid Orchestrator (Figure 1). Our performance evaluation considers two well-known centralized controllers and multi OVS. The evaluation in terms of throughput, latency, and scalability is performed with Cbench. With Mininet and Iperf we evaluate TCP, UDP and ICMP flow over several network topologies. The main purpose is to predict and to select the ISDN controller that exhibits the highest throughput, scalability, lowest latency, delay, packet loss, TCP transfer time and most rich in features and also to predict the optimal OVS with lowest TCAM percentages. This value can be used to determine the reward of the learning agent for the session.

We extend this hybrid orchestrator by adding the code to install the selected forwarding North Rules (R1(RRD), R2(SRD), R3(RTPD), etc) created by the learning agent prior to handling any outcoming traffic.

At the beginning of each emulation experiment, traffic flows are generated into the network topology, the state of the flow table in the OVS, and the state of the ISDN controllers is observed. To test the controller's performance in terms of latency, OVS sends an asynchronous message toward controller and waits for a reply. Hence, the number of acknowledgments gathered in a test period of time is used to calculate the average latency. For the throughput measurements, each OVS transfers a burst of packet-in without waiting for a response to estimate maximum flow requests a controller can handle per second. Emulated hosts have access to Linux file system commands. Like "Iperf". Therefore, network design can easily move to the real system with minor deployment [5].

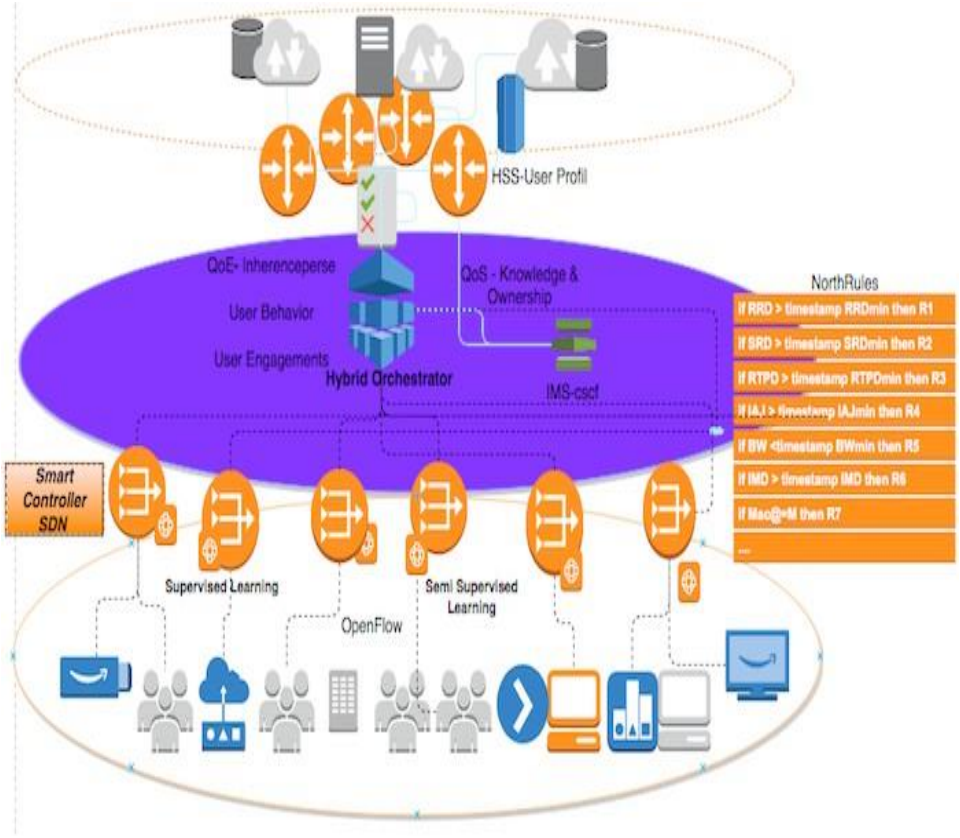

Figure 1. Hybrid Learning Orchestrator for multi-ISDN proposed

5. DESIGN OF REINFORCEMENT LEARNING FRAMEWORK OF ISDN

In our approach, the configuration of rules can be defined by two decisive parameters, flow match frequency and flow recentness. Therefore, the RL algorithm can be modeled to obtain the network configuration that minimizes the long-term control plane overhead [7]. This approach is modeled using a Markov Decision Process (MDP), which can be represented as the tuple <S, A, P, R>, where S = {s1, s2, . . . , si } is the state-space, A = {a1, a2, . . . , aj } is the action-space, P defines the transition probability from state i to state j, and R defines the reward associated with various actions a ∈ A. The goal of this MDP is to develop a policy $\pi : S \rightarrow A$ that maximizes the cumulative rewards obtained in the long term [4][6]. In the problem that we address in this research, we can know all the forwarding rules that need to be processed for all tra c ows transported on the network. For our problem, the state-space, action-space, and reward function are de ned as follows:

In our approach, we utilize the following RL algorithms:

5.1. Reinforcement Learning Framework of ISDN.

In our design, the Q-Learning algorithm is utilized in our emulation experiments. The decision to utilize the QLearning algorithm is motivated by the fact that it does not need a model for the environment, and it updates the Q-values at each time step based on the estimation of action-value function. This iterative learning process is conducted in discrete time steps in which the agent interacts with the environment. At each step, the agent interacts with the environment. At each step, the agent selects an action at ∈ A in the given state st.The agent makes its own

decisions to choose an action based on the action selection policy with the goal of maximizing the expected reward. The notation Q(s, a) denotes the average Q-value of an action a at state s. While the immediate reward is collected, Q(s, a) can be further refined as:

$$Q(st,at)= r + Q(st,at) - \varepsilon Q(st,at)+ \gamma \max Q(st+1,at+1)$$

where ε ($0 < \varepsilon \leq 1$) is the learning rate that determines to what degree the newly obtained information overrides the old one and γ ($0 \leq \gamma \leq 1$) is the reward factor that defines the importance of future rewards and guarantees convergence of the accumulated reward. In our experiments, we use a discount factor of 0.85 as these values are commonly used in practice.

ALGORITHM : Q-Learning Algorithm

Preconditions:
Initialize Q-table random
TCAM percentage Initialize
state s_t
Initialize goal ε
Procedure:
01: improvement $_{measures}$ = 0
02: repeat
03: for (step = 0; step < learning_iteration;
step++) 04: Get action a_t from s_t using β
05: Get parameter from s_t using β
06: $\beta = \beta - \beta$*(step / learning_iteration)
07: Take action a_t on the parameter and receive reward r, control overhead C
08: Sample new state s_t +1 after applied action a_t
09: Update $Q_t \leftarrow r + Q_t - \varepsilon Q_t + \gamma * maxQ_t+1$
10: Update the corresponding parameter of Q_t
11: improvement $_{measures}$ = get_improvement $_{measures}$
($best_t$ ++, $best_t$ - -) 12: end for
***13:* until improvement $_{measures}$ > ε**

The Q-Learning algorithm can have a chance to explore the action space in the process of searching for a better action that produces a better reward. β is used as the action selection policy in which the action with the highest Q- value is always picked while the probability of picking some other action at random is small.

If the action is chosen to be performed due to its Q-value, the associated parameter value will be updated as well. ε represents the terminal objective of the algorithm. Where this algorithm keeps track of the chosen parameter corresponding to the Q-value of each action.

This value interval can be divided into four different states as follows:

- State 1: If the value of the TCAM belongs to [0..30]: it is a weak state
- State 2: If the value of the TCAM belongs to [30..50]: it is an ideal and appreciated state.
- State 3: If the value of the TCAM belongs to [50..80]: it is a critical state.
- State 4: If the value of the TCAM belongs to [80..90]: it is a mediocre state

To facilitate the understanding of the different actions of the agent, we can first represent the states as follows:

Table 1. The different states available for each controller

State 1	State 2
State 4	State 4

The agent's actions are modelled as follows:

Action 1 (go up): transition from state 4 to state 1 or from state 3 to state 2 Action 1 (go up): transition from state 4 to state 1 or from state 3 to state 2 Action 3 (turn right): transition from state 1 to state 2 or from state 4 to state 3 Action 4 (turn left): transition from state 2 to state 1 or from state 3 to state 4

6. EXPERIMENTAL RESULTS

6.1. Testbed Setup

The testbed consists of a single machine with (Intel Core i5-7200U CPU @ 3.1 GHz), 8 GB of memory available. The system runs Ubuntu 16.04 LTS-64 bit with VMware Workstation Pro version 14.1.1 installed. Controllers, Cbench and Mininet are installed in separate VMs the allocated memory is respectively (4GB, 2GB, 2GB).

When it comes to Machine learning, there are four main steps that need to be followed which are:

1. Looking at the big picture
2. Getting The Data
3. Preparing the Data for Machine Learning algorithms
4. Selecting and training, and fine-tuning a model The machine learning libraries used are :

1. TensorFlow : is well known for its flexible architecture which allows for the deployment of computation across a variety of platforms like CPUs or GPUs on a desktop, a server, or a mobile device.
2. Keras : is a python-based Deep Learning. It works in a different way than other Deep Learning frameworks. Keras does not support low-level computation like TensorFlow which is why it uses a tool named Backend.
3. Scikit-learn : is a Machine Learning Library. It includes a wide array of classification, regression, and clustering algorithms. It was conceived in a way to interoperate with other mathematical python libraries like Scipy, and NumPy.
4. Jupyter Notebook : is a Web application allowing the creation and sharing of doc- uments containg live code, visualizations, and narrative text.
5. Anaconda : is a Python and R programming languages distribution which conveniently installs them additionally to other packages commonly used in Data science and scientific computing.

We have succeded in training a model which is to be implemented on an SDN Controller. its task is intelligent control and routing. We will go through these steps one by one while explicitly explaining the choices we have made throughout the development process [13] [14].

6.2. Model Context

For this model, we used Mininet, a network emulator to generate a Network OVS, controllers, hosts, and links. in figure .2. The OVS support Openflow protocol for high flexibility when it comes to customized routing and SDN. Figure 3, 4 and 5 presents the iteration of our algorithm.

6.3. Q-Learning Algoritm Outputs

Starting from the database containing the ternary memory consumption rate of a OVS as a function of time, we applied the Q-learning algorithm and we have it trained several times so that it converges towards an optimal result and informs us the best OVS in terms of QoS. At each iteration, the agent stores the corresponding values (Q-Values) resulting from Bellman's equation in a dynamic array named Q-table. Since all of the study and testing is done on a database of seven OVS, such a procedure allows us to get two OVS in order to decide which of these two is the most efficient.

The different rates of memory consumption existing in our database are quite large. The figure. 2 below shows part of the database imported into the working environment.

The three previous cases show us different iterations of the algorithm, which ends up deciding in the Xth iteration, with X is the length of the database, that the OVS R2 is optimal and we can also retain it.

6.4. The Dataset

In order to train a model, we need a Dataset. In our case, we used the emulated network on Mininet to generated Data flows and recuperate them using Wireshark, a free opensource packet sniffer. A sample of the recuperated data is shown in Table II.

```
Entrée [47]: data
```

Out[47]:

	time	tcam
0	1	9.960
1	2	19.921
2	3	14.843
3	4	17.968
4	5	25.000
5	6	39.843
6	7	29.882
7	8	36.914
8	9	10.937
9	10	39.843
10	11	29.882
11	12	36.914
12	13	10.937
13	14	39.843
14	15	29.882
15	16	36.914
16	17	10.937

Figure 2. Percentage of ternary memory consumption as a function of time

```
('etat de R1', 1)
('etat de R2', 3)
('action sur R1:', 1)
('action sur R2:', 0)
('etat futur de R1', 4)
('Q_table de ROUTER 1', [[0.0, 0.0, 0.0, 0.0], [0.0, 8.1, 10.0, 0.0], [0.0, 0.0, 0.0, 10.350000000000001], [18.1,
0.0, 0.0, 0.0], [9.0, 0.0, 0.0, 0.0]])
('etat futur de R2', 2)
('Q_table de ROUTER 2', [[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 18.990000000000002, 0.0, 0.0], [30.0310
00000000002, 0.0, 0.0, 15.876000000000001], [0.0, 0.0, 20.286, 0.0]])
('etat de R1', 4)
('etat de R2', 2)
('action sur R1:', 2)
('action sur R2:', 3)
('etat futur de R1', 3)
('Q_table de ROUTER 1', [[0.0, 0.0, 0.0, 0.0], [0.0, 8.1, 10.0, 0.0], [0.0, 0.0, 0.0, 10.350000000000001], [18.1,
0.0, 0.0, 0.0], [9.0, 0.0, 16.290000000000003, 0.0]])
('etat futur de R2', 1)
('Q_table de ROUTER 2', [[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 18.990000000000002, 0.0, 0.0], [30.0310
00000000002, 0.0, 0.0, 15.876000000000001], [0.0, 0.0, 20.286, 0.0]])
R1 est le commutateur choisi
--------------------------------------------------------------------
```

Figure 3. 1st iteration of the algorithm

```
('etat de R1', 2)
('etat de R2', 1)
('action sur R1:', 1)
('action sur R2:', 2)
('etat futur de R1', 3)
('Q table de ROUTER 1', [[0.0, 0.0, 0.0, 0.0], [0.0, 20.604172500000004, 65.01315011461001, 0.0], [0.0, 43.5546145
76625006, 0.0, 52.313011343775021, [41.650731475, 0.0, 0.0, 0.0], [21.934653750000003, 0.0, 16.290000000000003, 0.
0]])
('etat futur de R2', 2)
('Q table de ROUTER 2', [[0.0, 0.0, 0.0, 0.0], [0.0, 50.764501452150014, 73.52440426664651, 0.0], [0.0, 70.5826714
0738502, 0.0, 21.361158000000003], [69.03226399165001, 0.0, 0.0, 49.99649995215002], [48.15280030693502, 0.0, 52.9
0566661350002, 0.0]])
R2 est le commutateur choisi
--------------------------------------------------------------------------
```

Figure 4. 5th iteration of the algorithm

```
('etat de R1', 4)
('etat de R2', 4)
('action sur R1:', 0)
('action sur R2:', 0)
('etat futur de R1', 1)
('Q table de ROUTER 1', [[0.0, 0.0, 0.0, 0.0], [0.0, 136179126.7210223, 113352876.44597864, 0.0], [0.0, 80663333.2
9481906, 0.0, 129271262.56640089], [121768242.50498992, 0.0, 0.0, 125349086.88219044], [140504429.49266958, 0.0, 1
30665431.65608722, 0.0]])
('etat futur de R2', 1)
('Q table de ROUTER 2', [[0.0, 0.0, 0.0, 0.0], [0.0, 175966527.3483686, 131091434.93210757, 0.0], [0.0, 128257221.
3069882, 0.0, 144939674.09418386], [133604782.00057118, 0.0, 0.0, 151316697.2787004], [183602667.32877815, 0.0, 15
7909394.01029158, 0.0]])
R2 est le commutateur choisi
--------------------------------------------------------------------------
```

Figure 5. Xth iteration of the algorithm

6.5. Data preparation

To prepare our Data, we first start by eliminating useless information from it. As is shown in the Table II., our Dataset contains many columns that our model's has no use for, which is the descriptive information in the last column. Another constraint when it comes to exploiting datasets is that Machine Learning algorithms can't handle String input, for that we had to transform Source IP address, Destination IP address and Protocol columns into integers. This process can be easily done via simple Python functions. The following Table.I illustrates the prepared Dataset.

Following this step, we proceed to dividing our Data into two parts: a Training set and a validation set.

Table 2. A Glimpse of the Raw Dataset

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.134.178	192.168.134.254	DHCP	342	DHCP Requ
2	0.000062	192.168.134.254	192.168.134.178	DHCP	342	DHCP ACK
3	5.234208	Vmware_31:3e:a2	Vmware_f7:af:62	ARP	60	Who has 1
4	5.234233	Vmware_f7:af:62	Vmware_31:3e:a2	ARP	60	192.168.1
5	25.764849	192.168.134.134	192.168.134.178	TCP	74	42830 → 6
6	25.765292	192.168.134.178	192.168.134.134	TCP	74	6653 → 42
7	25.765334	192.168.134.134	192.168.134.178	TCP	66	42830 → 6
8	25.765808	192.168.134.134	192.168.134.178	TCP	66	42830 → 6
9	25.768471	192.168.134.178	192.168.134.134	OpenFlow	82	Type: OFP
10	25.768522	192.168.134.134	192.168.134.178	TCP	54	42830 → 6
11	25.768868	192.168.134.178	192.168.134.134	TCP	66	6653 → 42
12	25.768891	192.168.134.134	192.168.134.178	TCP	54	42830 → 6
13	27.457449	192.168.134.134	192.168.134.2	DNS	76	Standard
14	27.457607	192.168.134.134	192.168.134.2	DNS	76	Standard
15	27.473012	192.168.134.2	192.168.134.134	DNS	108	Standard
16	27.553225	192.168.134.2	192.168.134.134	DNS	137	Standard
17	27.811764	192.168.134.134	192.168.134.2	DNS	76	Standard
18	27.812379	192.168.134.134	192.168.134.2	DNS	76	Standard
19	27.853755	192.168.134.2	192.168.134.134	DNS	108	Standard
20	27.922989	192.168.134.2	192.168.134.134	DNS	137	Standard

Table 3. A Glimpse of the Processed Dataset

Out[15]:

	No.	Time	Source	Destination	Protocol	Length
0	1	0.000000	0	0	0	342
1	2	0.000062	1	1	0	342
2	3	5.234208	2	2	1	60
3	4	5.234233	3	3	1	60
4	5	25.764849	4	1	2	74
5	6	25.765292	0	4	2	74
6	7	25.765334	4	1	2	66
7	8	25.765808	4	1	2	66
8	9	25.768471	0	4	3	82
9	10	25.768522	4	1	2	54
10	11	25.768868	0	4	2	66

6.6. Model Selection and Training

As our Data is labeled, it is natural to choose a Supervised Learning al- gorithm, all the more so due to Supervised learning being customizable and frequently used in problems relating to self-healing networks. This is particularly the case for the OpenDayLight project, a modular open platform for the customization and the automatization of networks of all sizes. It is a Project that took birth out the SDN movement, clear focusing on Network programmability.

In the following section, we introduce the libraries used from realization of our model.

6.7. Model Creation and Fine-tuning

When it comes to Creating and Fine-tuning Convolutional Neural Network models comes down to optimizing the Hyperparameters of the model (e.g. Number de filters, size of each du filter, activation function of each layer, etc.). There are many ways to do this. One can try to fiddle with them manually until a great combination is found, which can be quite time-consuming, or he can try to use pre-installed Scikit-learn search methods like the Randomized search.

In our case, and considering the fact that our Data is not complexe, we used the simplest fine-tuning method which is Grid search. the resulting optimized CNN structure had the following architecture in figure .6.

The following is a break-out of the models layers:

- The Input Layer: this layer contains our input values which we want to predict and train the model with, where each neuron in that layer contains a value took from the sample. In our case, our dataset contains 6 columns, thus the 6 neurons at the top layer.
- The Output Layer: is the layer will give us an information about the sample we gave to our neural network. In our case, it would be predicting the IP destination address. As our Data flow simulation engaged 16 IP addresses, the number of neurons in this layer is 16.
- The Hidden Layer: The hidden layers can be composed of one or many sublayers whose numbers and hyperparameters where determined by the Grid Search. In our case it contains two layers.

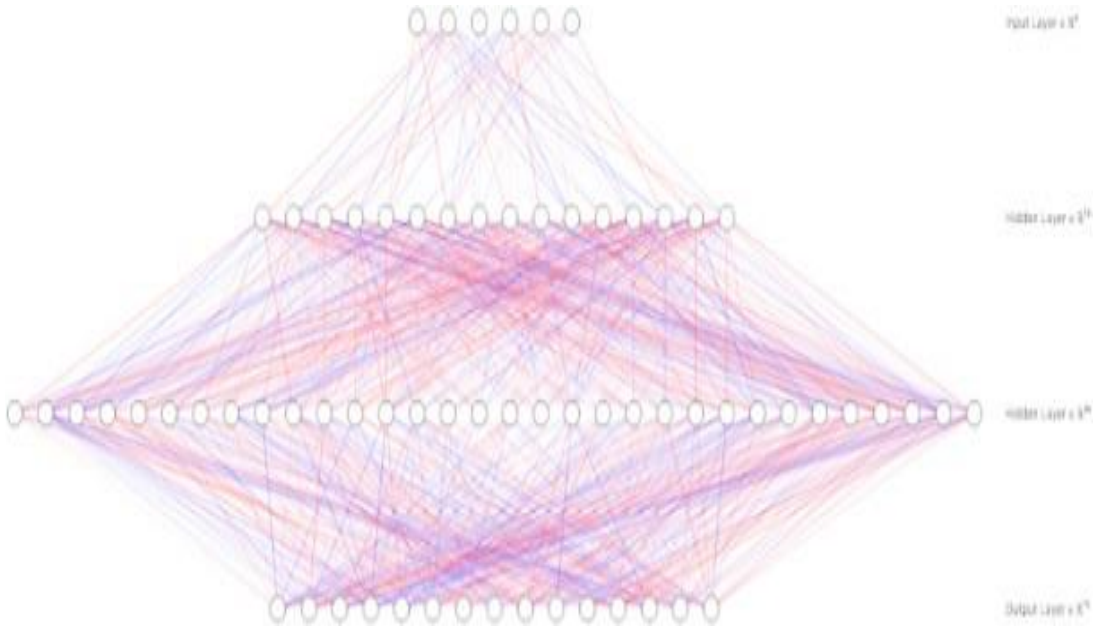

Figure 6. Model Architecture

At this point, we have built our model and pre-processed our Data. All that is left is to compile the model, train it, and save it using the previously defined functions compile(), model.fit(), and model.save() respectively.

The evaluation of our model using the validation Data gives an accuracy ratio of 72.7%.

6.8. Problems and limitations

When it comes to Machine Learning, the main tasks are the selection of the algorithm and the Data, and latter is what we encountered.

- Insufficient Quantity of Data is a Major problem when it comes to training a model. Even for very simple problems like ours, we need thousands of examples. For more complex problems like sound recognition or image classification, we would need millions of examples [16].
- Irrelevant Features: As they say: garbage in, garbage out. Our model will only learn correctly if the training Data contains many relevant features and not too many insignificant ones. These problems, more often than not, lead data scientists to over-fitting their Data to the training set, Optimizing the hyperparameters in search for a better accuracy ratio at the training without taking into account the global, dynamic context in which the mode l will be deployed, which usually leads to bad overall performance after deployment.

CONCLUSIONS AND FUTURE WORKS

Supervised machine learning methods, such as neural network (NN), convolutional neural network (CNN), and recurrent neural network (RNN) can apply to prediction and classification. Furthermore, reinforcement learning methods, are tools for generative networks and discriminative networks.

These methods contribute substantially to improving prediction and classification in relevant applications, but there remain issues and limitations that require further attention from the research community.

The testing process of Machine Learning on SDN controllers has long been considered critical for ISPs around the world. Fortunately, our trained-model confirmed the viability and the potential of the combination of the two aforementioned technologies.

For our case, we were implementing a supervised learning algorithm for intelligent routing and a reinforcement learning algorithm for to choice the optimal controller. This hybrid learning solution is being improved constantly as it self adjusts itself to all the customers.

In this paper, we trained a supervised-learning CNN to predictively routing data frames while arguing the choices taken in the process. We additionally presented the used frameworks followed by the experimental results. We expend the New Supervised Machine Learning Methods to the business layer and try to implemented it in an hybrid orchestrator which will have to efficiently choose which controllers ISDN to use in real-time to maximize performance indicators with Novel Reinforcement Learning Method where an agent has to take real-time decisions which is true for our problem. Despite the fact that the machine learning-based approach performs well in our experiments, further development is needed to expand the capabilities of our emulation scope. The rest of this work will be the application of Q-learning algorithm that we proposed. Currently, implementing the user behavior and the user engagements in the learning labels and model training more thoroughly is the focus of our ongoing research.

REFERENCES

[1] J. Wagner. Why Performance Matters. Accessed: Mar. 22, 2019. [Online]. Available: https://developers.google.com/web/fundamentals/performance/ why-performance-matters/

[2] M. Seufert, P. Casas, N. Wehner, L. Gang, and K. Li, “Streambased machine learning for real-time QoE analysis of encrypted video streaming traffic,” in Proc. 22nd Conf. Innov. Clouds, Internet Netw.Workshops, Feb. 2019, pp. 76–81.

[3] V. Krishnamoorthi, N. Carlsson, E. Halepovic, and E. Petajan, “BUFFEST: Predicting buffer conditions and real-time requirements of HTTP (S) adaptive streaming clients,” in Proc. 8th ACM Multimedia Syst. Conf., Jun. 2017, pp. 76–87.

[4] M. Lu, W. Deng, and Y. Shi. 2018. TF-idletimeout: Improving e ciency of TCAM in SDN by dynamically adjusting ow entry lifecycle. In Proceedings of the 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, Budapest, 2681–2686.

[5] M. H. Mazhar and Z. Shafiq, “Real-time video quality of experience monitoring for HTTPS and QUIC,” in Proc. IEEE Conf. Comput. Commun. (INFOCOM), Apr. 2018, pp. 1331–1339.

[6] W. Robitza, “Towards behavior-oriented quality of experience assessment for online video services,” in Proc. Int. Conf. Interact. Experiences TV Online Video, Jun. 2017, pp. 123–127.

[7] Tam N. Nguyen, 2018 2nd Cyber Security in Networking Conference (CSNet), The Challenges in SDN/ML Based Network Security : A Survey. 4 Apr 2018.

[8] Ting Y, Al-Fuquaha A, Shuaib K, M. Sallabi F, Qadir J, 2018, November. SDN Flow Entry Management Using Reinforcement Learning, In 2018 Transactions on Autonomous and Adaptive Systems (ACM).

[9] Seufert. M, Wassermann. S and Casas. P, IEEE COMMUNICATIONS LETTERS,. Considering User Behavior in the Quality of Experience Cycle: Towards Proactive QoE-Aware Traffic Management, VOL. 23, NO. 7, JULY 2019.

[10] S. C. Lin, I. F. Akyildiz, P. Wang, and M. Luo. 2016. QoS-aware adaptive routing in multi-layer hierarchical software de ned networks: A reinforcement learning approach. In Proceedings of the 2016 IEEE International Conference on Services Computing (SCC). IEEE, San Francisco, CA, 25–33.

[11] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari. 2017. Deep reinforcement learning for dynamic multichannel access. In Proceedings of the International Conference on Computing, Networking and Communications (ICNC). IEEE, Silicon Valley, 257–265.

[12] S. C. Lin, I. F. Akyildiz, P. Wang, and M. Luo. 2016. QoS-aware adaptive routing in multi-layer hierarchical softwaredefined networks: A reinforcement learning approach. In Proceedings of the 2016 IEEE International Conference on Services Computing (SCC). IEEE, San Francisco, CA, 25–33.

[13] POX Controller, [Online]. Available: https://github.com/noxrepo/pox. February 2020.

[14] Google DeepMind. [Online]. Available: https://deepmind.com/ . February 2020.

[15] S.Garba, Metric-Based Framework for Testing & Evaluation of Service-Oriented System, Intenational Journal of Software Engineering & Application (IJSEA) , Vol.10, No.3, May 2019.

[16] L.Zeheng and H.LiGuo, TRACING REQUIREMENTS AS A PROBLEM OF MACHINE LEARNING, International Journal of Software Engineering & Applications (IJSEA), Vol.9, No.4, July 2018.