TECHNISCHE UNIVERSITÄT CHEMNITZ

Faculty of Communication Networks

Department of Electrical Engineering and Information Technology

Research Project

Reinforcement Learning Approach for Adaptive Traffic Flow Rule Aggregation in Software Defined Networks

Syed Tasnimul Islam

Chemnitz, 30th January 2019

Supervisor: M. Sc. Trung Phan Van
Advisor: Prof. Dr.-Ing. Thomas Bauschert

Abstract

Software Defined Networking is an emerging technology that decouples control plane from the legacy networking hardware. By leveraging centralized network view and high speed Ternary Content Addressable Memory(TCAM) in forwarding plane a unique and efficient content filter technique is adopted in SDN. However, TCAM being a very high power-consuming component that SDN switches can store limited number of flow rules in the flow tables. This has drawn researchers to address the content filtering method which is "match field" to be dynamic instead of static. There has not been any effective solution to the problem where level of details of traffic flow and network stability are both achieved, recently.

In this research project, an adaptive traffic flow rule aggregation method is proposed that harnesses the recent advances in machine learning. A novel approach have been taken here integrating different modules that monitor and analyze overall traffic in the network, predict performance degradation of SDN switch and take decision to apply match field combination according to network state. This proposed method extends the recent work [28] for predicting switch degradation and for decision making purpose an advanced machine learning technique Reinforcement Learning(RL) is used. The RL-agent implements Q-Learning algorithm for decision making purpose and for taking action it adopts epsilon-greedy policy. The proposed novel application is implemented and evaluated under an SDN emulation environment and compared with the default SDN match field setup methods and also with the recent research [28] that has been integrated into this application in terms of total number of flows in the network and average number of packet_in messages to the controller. The results show that this novel approach yields significant SDN switch performance while providing detailed traffic flow information.

Contents

TCAM	**Ternary Content Addressable Memory**
SDN	**Software-Defined Networking**
OVS	**Open vSwitch**
STARs	**SofTware-defined Adaptive Routing**
ASIC	**Application-Specific Integrated Circuit**
ONOSs	**Open Network Operating System**
Bonesis	**Botnet Simulator**
FMSs	**Full Matching Scheme**
RL	**Reinforcement Learning**
MMOSs	**MAC address Match Only Scheme**
MDPs	**Markov Decision Process**
DATA	**Destination-aware Adaptive Traffic Flow Rule Aggregation**
DRQN	**Deep Recurrent Q-Network**
SVM	**Support Vector Machine**
TCP	**Transmission Control Protocol**
UDP	**User Datagram Protocol**
ICMP	**Internet Control Message Protocol**

1 Introduction

1.1 Motivation

Software-Defined Networking (SDN) is a novel technology that enables global network view and centralized flow control of the whole network. Unlike legacy network decision making responsibility for the whole network ; i.e. routing ,traffic engineering and network management applications are in the hand of a central device named **Controller**. To date[1] in most deployed SDN ,**Openflow** switch functions as a forwarding unit, which is only responsible for transmitting packets according to its function rules. This standard essentially gives engineers' access to flow tables[1] to define data flows through software designing rules to guide switches how to direct network traffic. Whenever a unique flow comes into a Openflow switch it sends a part of the packet header to match with existing rules to the Controller. In addition to packet headers, matches can also be performed against the ingress port and meta-data fields[1]. In Openflow forwarding decisions can be made taking into account from MAC-based forwarding in layer 2, IP-based routing in layer 3, and packet classification for layers 2–4 match fields. The Openflow switch stores the entries received from the controller in its flow tables and functions according to them. The flow table is stored in dedicated and very high speed memories called **Ternary Content-Addressable Memory (TCAM)**. But unfortunately, TCAM are expensive and have high power consumption [12] and can store a limited number of rules (typically 1500 to 3000 rules). The rule installation capacity is very much insufficient ,for example, recent measurement shows that data centers can have up to 10,000 concurrent flows per second per server rack[3]. The number of match fields used in SDN Controller to filter content is linearly proportional to number of new rules installed in TCAM. So in heavy real traffic scenario or in DDOS attack situations if more match fields are used then the probability of TCAM exhaustion[12] is high. So one simple solution could be just to use layer 2 match field that is destination MAC address only. But the number of match fields used directly impact the amount of unnecessary traffic in the network. Additionally in SDN the applications that will determine routing or that will be deployed to keep the network secure might need user node's different information i.e. VLAN id or packet's DSCP. These information are gathered from the match field values sent to the controller. The best solution of this dilemma would be to use variable match fields according to network state. But in current scenario most default matching strategies rely on static number of match fields[27]. In this paper a novel approach is taken to have a solution of this scenario.

1.2 Related Research Work

Many existing studies in the literature have addressed TCAM's limited rule space problem. Various researchers took different approach to mitigate this problem. For instance, the authors in [3] proposed an online routing scheme named **Software-defined Adaptive Routing (STAR)** that efficiently utilizes limited flow-table resources to maximize network performance. Authors in [4] formulated an optimization problem with the objective to maximize the number of flows in the network, constrained by the limited flow table space in SDN switches which is a great scheme from the service providers' point of view. But this does not solve the problem of saving the whole network when a sudden flood of unique packet is generated or unwanted traffic resides in the network in great number. Several other approaches were taken by different authors[9],[8],[2] keeping the limitation of TCAM space in mind but they did not provide any discussion on match fields. For example research was done [9] on flow table size reduction, flow based packet classification [8] by exploiting the temporal locality of network traffic. Only one existing study approached the method of having dynamic match field [28]. They proposed a in their paper, a practical extension for adaptive traffic flow matching in SDN-based networks in order to adapt the number of flow table entries in SDN switches according to the level of detail of traffic flow information. But the implementation of match field had only 2 degrees of freedom. One scheme was denoted as MAC Matching Only Scheme (MMOS) where only the destination MAC address is examined during the matching process and the other one was Full Matching Scheme (FMS) where in addition to MAC address the src/dst IPs and the src/dst ports were also added to the matching fields. But the full matching scheme does not give enough granularity of information needed for development of various applications.

In this research project, the general challenge of solving the problem of having dynamic match fields. For doing that a specific machine learning algorithm is used that is Reinforcement Learning. Continuous stream of network state is fed into Reinforcement learning's online learning algorithm Q-Learning [29] and it outputs best action based on network state. The action in this case is the number of match fields to be used according to current network state. However a module from the research work done in [27] is used in the implementation part of this paper as a reward function that is 2-Dimensional Support Vector Machine for prediction of switch state. For demonstration purpose of this project Open Network Operating System (ONOS) was used and for the limitation of simulation scenario only 9 match fields' combinations were used. But in real environment scenario all combination of match fields is possible with the algorithm proposed.

The rest of the report has been organized as follows:

- Chapter 2: Explains the basic process of an incoming packet handling in SDN and match field requirement of different applications.
- Chapter 3: Introduces machine learning in general and dives deep into Reinforcement Learning.
- Chapter 4: Explanation of proposed architecture, implemented algorithms and overall

work-flow .

- Chapter 5: Experimental setup and result analysis of the emulated system.
- Chapter 6: Direction of the Future Research and conclusion.

2 SDN Architechture

A software-defined networking (SDN) architecture defines how a networking and computing system can be built using a combination of open, software-based technologies and commodity networking hardware that separate the control plane and the data plane of the networking stack. Basic components of SDN are SDN Controller and OpenFlow Switch. An OpenFlow Switch which acts only as a forwarding device consists of one or more flow tables and a group table [26]. 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 Figure 2.1.

Whenever a packet is received by a SDN switch, it checks whether there is a matching forwarding rule. If so, it applies the rule to this packet. However, if no rule is applicable, i.e. it is a new unique packet, the switch sends packet_in message to the controller. The message PACKET-IN contains the incoming packet information, e.g.header, buffer id, in port, payload, etc Figure 2.2 (a). The amount of information of incoming packet that will be sent to the controller through the packet_in message depends on the number of match fields set by the network administrator in the controller's forwarding application i.e. ONOS forwarding application. According to Openflow-specification v1.3.0. an Openflow switch can support 40 match field types [26]. But it is not required to support all match field types, just those listed in the Table 2.1 .

2.1 PACKET-IN Message Processing with example

A brief overview of PACKET-IN message processing is shown in Figure 2.2 (a). For this example it can be assumed that the network administrator has selected the match field as "Match Destination MAC Only". It can also be assumed that the forwarding application used in this example scenario is "Reactive Forwarding". This means when a new flow comes into the switch, the OpenFlow agent on the switch, does a look-up in the flow tables, either in a search ASIC if in hardware or a software flow table in the case of a OpenFlow Switch . If no match for the flow is found, the switch creates an OFP PACKET packet and fires it off to the controller for instructions via PACKET-IN message. The controller decides what to do with the packet and sends back the flow installation rule to the switch via FLOW-MOD message. If the rule is a temporary one than in FLOW-MOD message a time out value is also added. After the time out value expires the switch removes the flow and informs the controller about it via FLOW-REMOVED message.In (Figure 2.3-Step1) host 1 sends a request to switch to send packet to host 2. We would assume there is no flow rule installed at the beginning in the flow table for this packet. As the match field is set as "Destination MAC Only" the switch will sent request for flow rules with MAC addresses of the destination

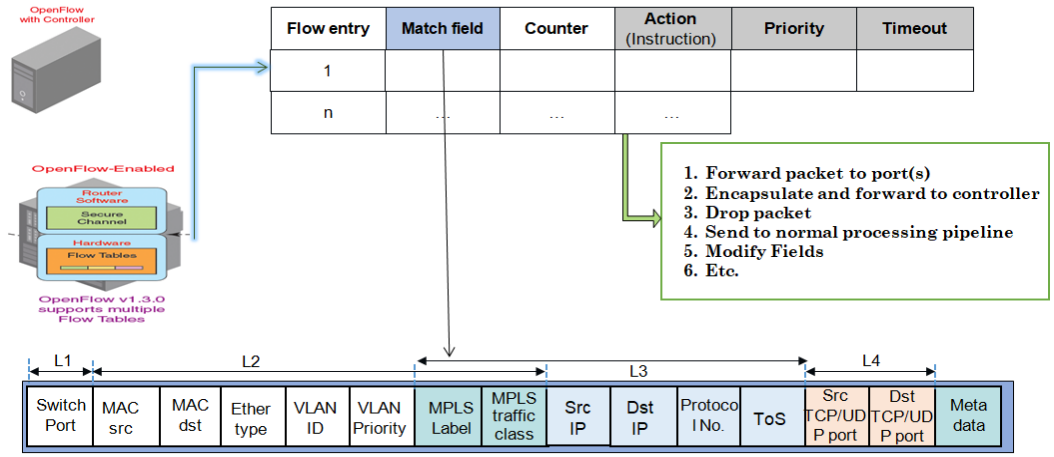


Figure 2.1: An overview of SDN and Openflow

and source host(Figure 2.3-Step1). Figure 2.2 (b) has a sample image of a PACKET-IN message when match field is set as destination MAC only and a 'ping' command is used.

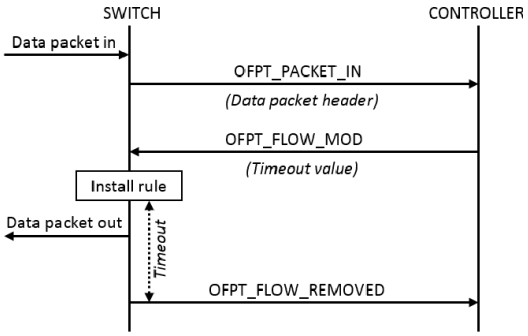
Table 2.1: Minimum number of match fields to be supported by a OpenFlow Switch.

Field	**Description**
`IN_PORT`	Ingress port. Physical or switch-defined logical port.
`ETH_DST` & `ETH_SRC`	Ethernet source and destination address. Can use arbitrary bitmask.
`ETH_TYPE`	Ethernet type of the OpenFlow packet payload.
`IP_PROTO`	IPv4 or IPv6 protocol number.
`IPV4_SRC` & `IPV4_SRC`	IPv4 source and destination address. Can use subnet mask.
`IPV6_SRC` & `IPV6_SRC`	IPv6 source and destination address. Can use subnet mask.
`TCP/UDP_SRC` & `TCP/UDP_DST`	TCP/UDP source and destination port.

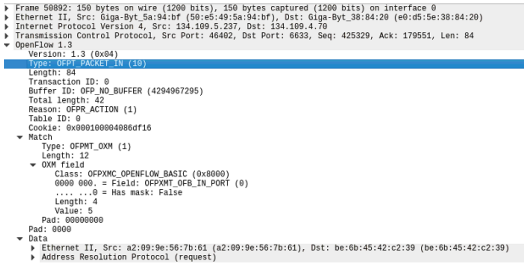
Controller will check its MAC table of hosts in the network and (Figure 2.3-Step4) sends a **PACKET-OUT** message to the switch to install the flow. In the mean time switch will install flow rule in its flow table (Figure 2.3-Step5).

2.2 Supported Match fields in some popular SDN Controllers

Nowadays, the most popular open source SDN Controllers are **ONOS**, **Floodlight**, **OpenDaylight** and **Ryu**. ONOS (Open Network Operating System) [31] is an Operating System (OS) designed to help network service providers build carrier-grade software-defined networks designed for high scalability, availability and performance. Although specifically



(a) OpenFlow messages when a unique packet arrives



(b) Example of a PACKET-IN message via Wire-shark

Figure 2.2

designed to address the needs of service providers, ONOS can also act as a software-defined networking (SDN) control plane for enterprise campus local area networks (LANs) and data center networks. ONOS features northbound and southbound application program interfaces (APIs) grounded in abstraction to prevent configuration and protocol lock-in for applications and devices, respectively.

The built in Reactive Forwarding application installs flows in response to every miss packet in that arrives at the controller. Using the REST API [30], system administrators can change the number of match fields for all switches of the network. Floodlight controller has a [15]convenient API called Wildcards and also another library named **OpenFlowJ-Loxigen** that allows to change match fields easily. Other popular platforms such as OpenDaylight, Ryu has their respective resourceful [16], [17] APIs for changing match fields in run time. OpenFlow v1.3 switches can support 40 match field types. The popular SDN platforms support a little varied number of match fields than this. ONOS supports 38 match fields out of 40.

2.3 Survey on SDN applications and match field requirements

Table 2.2 gives an overview of some popular SDN applications. Microsoft have a Lync [19] SDN Interface which is now called Skype for Business SDN Interface. And this allows developers to build applications and services that can monitor, isolate, and correct issues on the network that affect Skype for Business quality of experience. The idea here is rather than configuring quality of service manually for Lync Clients on network devices; classification and marking of traffic is dynamically and automatically done using Open Flow. This application requires source and destination IP address , mac addresss , application port , DSCP ,ECN , VLAN-ID, VLAN-PCP,IP-PROTO and some other match fields.
Another real world application that leverages SDN and OpenFlow is the HP Network Protector [20] which is available from the HP App Store. What this application does is it intercepts DNS queries and checks them against a database of malicious domain names

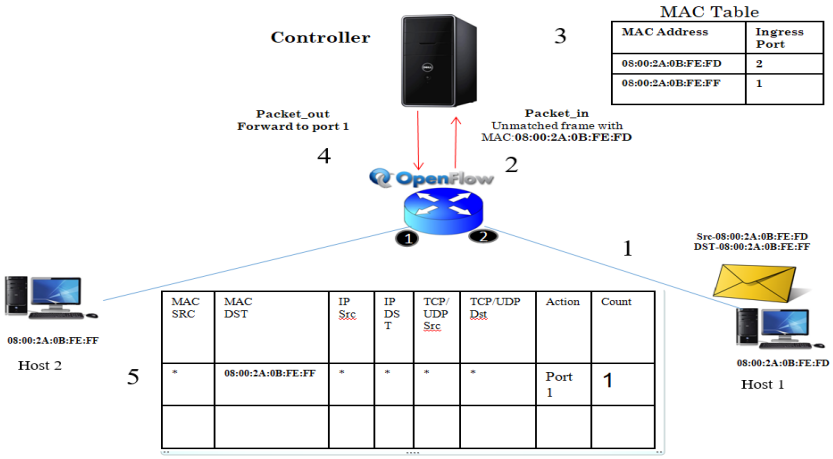


Figure 2.3: Example scenario of unique packet arrival in SDN

based on information from Tipping Point. So when a user tries to go to a malicious website, their traffic is intercepted because of an OpenFlow rule, is forwarded to Network Protector, and checked against the database. If the domain that they're trying to go to is malicious, the user either gets a non-existent domain reply from Network Protector, or they can be redirected to another server in the network. For flow matching this application requires IP-PROTO, TCP/UDP source and destination port, ETH-TYPE match fields.

An example application for allocating network resources based on an application's run-time requirements to accelerate Big-Data delivery [13] requires IN-PORT , TCP/UDP source and destination port, TCP/UDP source and destination IP address , TCP/UDP source and destination MAC address.

Table 2.2: Different popular SDN controllers and their supported match field.

Controller	Supported number of match fields	More Supported fields than OpenFlow	Less Supported fields than OpenFlow
ONOS	38	Null	MPLS-TC , PBB-ISID
FloodLight	40	PBB-UCA	PBB-ISID
OpenDayLight	40	Null	Null
Ryu	43	ACTSET-OUTPUT, TCP-FLAGS, PBB-UCA	Null

3 Reinforcement Learning

A brief discussion of Reinforcement Learning is done in this chapter that has been implemented in this research project. It is suggested to go through [29] for a detailed survey and rigorous derivations.

3.1 Definition

Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may act not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics—trial-and-error search and delayed reward—are the two most important distinguishing features of reinforcement learning.

3.2 Difference with other machine learning paradigms

Figure 3.1 gives overview illustration of all the machine learning algorithms. The basic three algorithms are **Supervised** Learning , **Unsupervised Learning** and **Reinforcement** Learning. Reinforcement learning is different from supervised learning, the kind of learning studied in most current research in the field of machine learning. Supervised learning is learning from a training set of labeled examples provided by a knowledgeable external supervisor. In interactive problems it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act. In uncharted territory, where one would expect learning to be most beneficial; an agent must be able to learn from its own experience. Reinforcement learning is also different from what machine learning researchers call unsupervised learning, which is typically about finding structure hidden in collections of unlabeled data. Although one might be tempted to think of reinforcement learning as a kind of unsupervised learning because it does not rely on examples of correct behavior, reinforcement learning is trying to maximize a reward signal instead of trying to find hidden structure.

In short, Reinforcement learning is the problem of getting an agent to act in the world so as to maximize its rewards. The agent receives a reward, which depends on the action and the state. The goal is to find a function, called a policy, which specifies which action to take in each state, so as to maximize some function (e.g., the mean or expected discounted

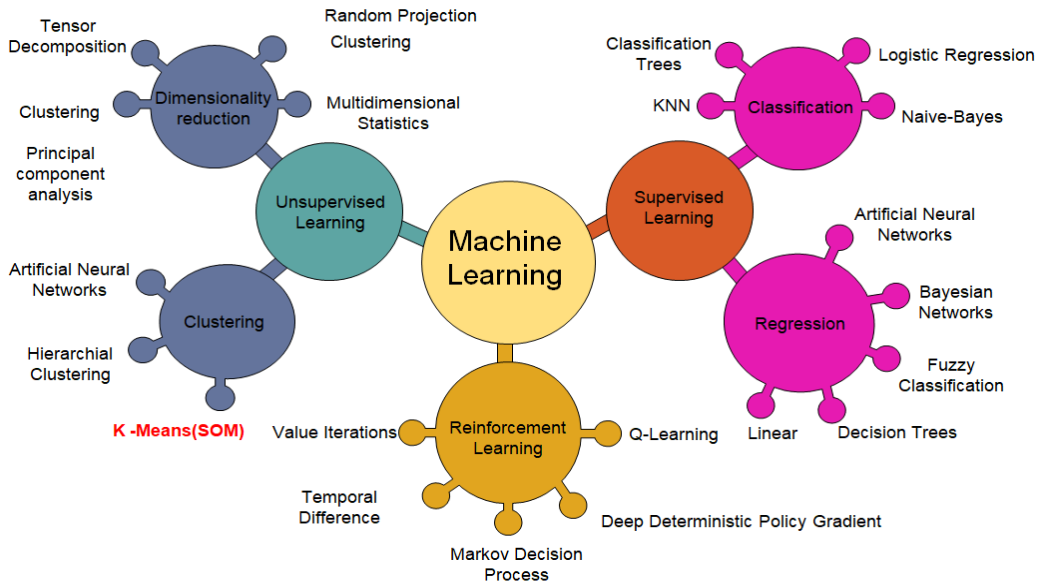


Figure 3.1: Overview of all machine learning algorithms

sum) of the sequence of rewards. Consider the general setting shown in Figure 3.2 where an agent interacts with an environment. At each time step t, the agent observes state S_t , and is asked to choose an action a_t. Following the action, the state of the environment transits to S_{t+1} and the agent receives reward r_t.

3.3 Markov Decision Process(MDP)

The Markov decision process, better known as MDP, is an approach in reinforcement learning to take decisions in a grid world environment. A grid world environment consists of states in the form of grids. The MDP tries to capture a world in the form of a grid by dividing it into states, actions, models/transition models, and rewards. The solution to an MDP is called a policy and the objective is to find the optimal policy for that MDP task. A stochastic process has the Markov property if the conditional probability distribution of future states of the process depends only upon the present state [32] as shown in Figure 3.3 . An MDP provides a mathematical framework for modeling decision-making situations where outcomes are partly random and partly under the control of the decision maker.

MDP is defined as the collection of the following:

- **States** : S
- **Actions** : A

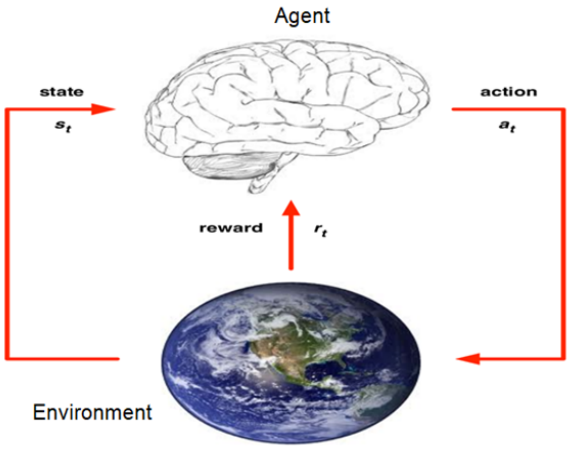


Figure 3.2: RL-agent's interaction with environment

- **Transition model** : T(s,a,s')
- **Rewards** : R(s,a)

3.3.1 State

The State set [21] is a set of different states, represented as s, which constitute the environment. States are the feature representation of the data obtained from the environment. State spaces can be either discrete or continuous.
The grid world shown in figure 3.4 can be considered as having 12 discrete states, where the green-colored grid is the goal state, red is the state to avoid, and black is a wall that a robot would bounce back from if it hit it head on.

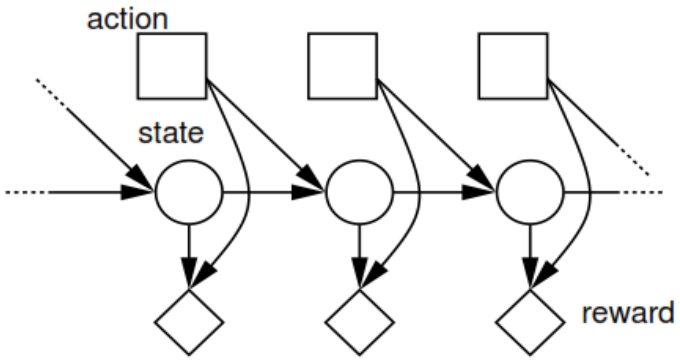


Figure 3.3: A Markov decision process. Circles indicate visible variables, and squares indicate actions.

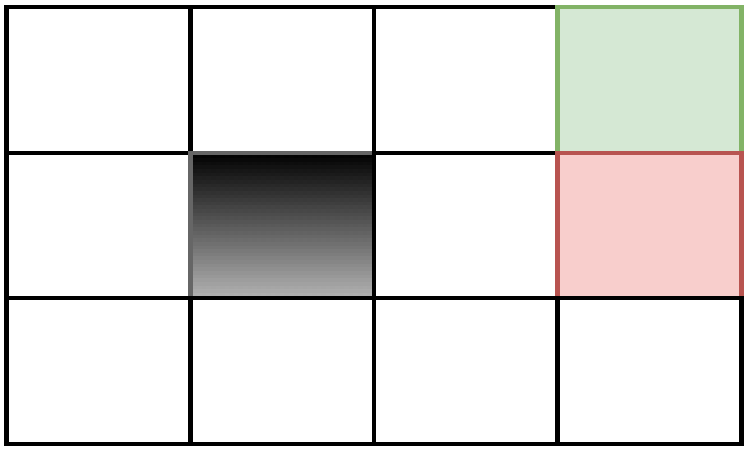


Figure 3.4: State Space having 12 discrete states and 4 discrete actions.

3.3.2 Action

Actions are the tasks an agent can perform or execute in a particular state. In other words, actions are sets of things an agent is allowed to do in the given environment. Like states, actions can also be either discrete or continuous. In the following grid world example there are 12 discrete states and 4 discrete actions (**UP**, **DOWN**, **RIGHT**, and **LEFT**)

Exploration vs Exploitation dilemma

Inspired by both control theory and behaviorist psychology [22], RL aims to obtain the optimal policy pi* under circumstances with unknown and partially random dynamics. Since RL does not have explicit knowledge over whether it has come close to its goal, it needs the balance between exploring new potential actions and exploiting the already learnt experience. This is the Exploration vs Exploitation dilemma in Reinforcement Learning.

Epsilon-Greedy Algorithm

There are several solution of this dilemma but the method taken in this research project implementation is Epsilon-Greedy Algorithm(Figure 3.5) .

The greedy algorithm in reinforcement learning always selects the action with highest estimated action value. Its a complete exploitation algorithm, which does not care for exploration. Well it can be a smart approach if the action value has been successfully estimated to the expected action value, like if the true distribution is known, just select the best actions. But when the estimation is unsure then the epsilon comes to the rescue.
The epsilon in the greedy algorithm adds exploration to the mix. So counter to previous logic of always selecting the best action, as per the estimated action value, now few times (with epsilon probability) select a random action for the sake of exploration and the remaining times behave as the original greedy algorithm and select the best known action.

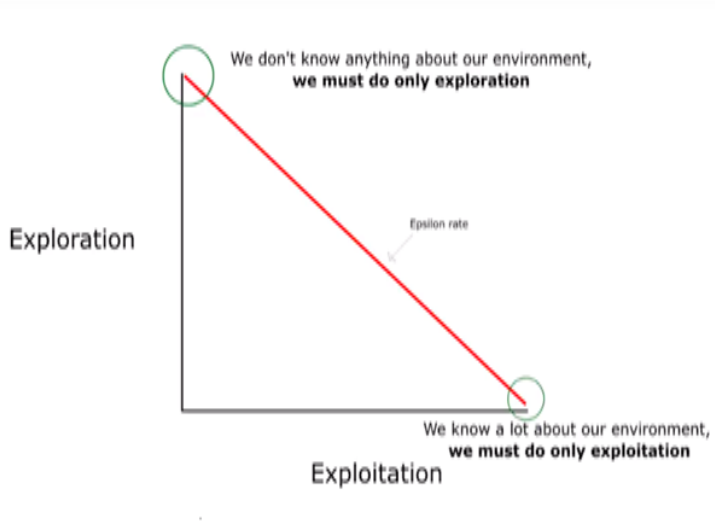


Figure 3.5: Exploration vs exploitation dilemma.

- p = ∈ ; action selection is random
- p = 1 - ∈ ; action selection is greedy

0-epsilon greedy algorithm will always select the best known action and 1-epsilon greedy algorithm will always select the actions at random.

3.3.3 Reward Function

The goal of solving an MDP is to find a policy which maximizes the total expected reward received over the course of the task. The expected discounted return for a policy (pi) is defined as the sum of discounted rewards that is expected when following policy (pi) :

$$(R^t)_\pi = (r^t + \gamma r^{t+1} + \gamma r^{t+2} + ...)_\pi = (\sum_{k=0}^{\infty} \gamma^k r^{t+k})_\pi \tag{3.3.3.1}$$

where t is the current time, and p(s, a) is the probability of selecting action a in state s. It is to be noted ,the discounting is required to ensure that the sum of infinite (discounted) rewards is finite, so that the quantity to be optimized is well-stated; the discount factor $\gamma \in [0, 1]$. It also models the agent behavior when the agent prefers immediate rewards than rewards that are potentially received far away in the future. Now the value varies from 0 to 1, if value is near 0 immediate reward is given preference and if value goes near 1, importance of future rewards increase until at 1 it is considered equal to immediate rewards.

learning under the assumption that all the states have the same next state distribution. It has been shown [23] by researchers that the asymptotic convergence rate of Q-learning has exponential dependency on 1 -γ, i.e., the rate of convergence is of
$f0^{\sim}(1/t^{1-\gamma})\ \ for\ \ \gamma \geq 1/2$.

3.4 Solving MDP with Q-Learning

Until now the blocks that create an MDP problem has been discussed. The policy is the solution to the MDP problem.

$$\textbf{Policy : } \pi(\mathbf{s}) \rightarrow \mathbf{a}$$

The policy is a function that takes the state as an input and outputs the action to be taken. A policy tells the learning agent what action to take for each possible state.
Q-learning is one of the reinforcement learning techniques used to find out the optimal action-selection policy based upon Markov decision process [23] . It's a model-free learning algorithm [1] .The Q-learning method uses a Q-table (figure 3.6) to map the state-action pairs. Q-table represents a table of values for every state and actions possible within the environment. The Q-table is updated according to the following function that follows Bellman's Equation:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)) \tag{3.4.1}$$

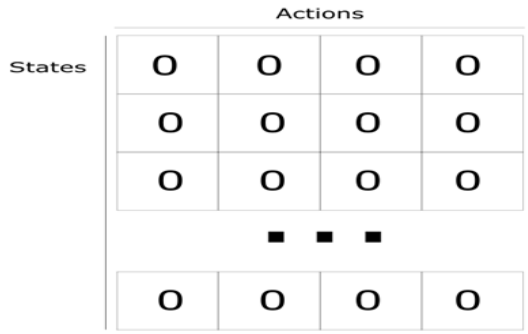


Figure 3.6: Q-table formed of state-action pair

Additionally it is to be noted that the learning rate α can take values between 0 to 1, where value of one means an agent tries to learn everything (exploration by ignoring prior knowledge) & value of zero means an agent learns nothing (exploitation of prior knowledge).There is interesting relationship between the convergence rate and the learning rate used in Q-learning [22]. For a polynomial learning rate, one which is $1/t^w$ at time t where w $\in$ (1/2,1) it has been shown that the convergence rate is polynomial in 1/(1 -γ), where γ is the discount factor. In contrast it is also shown that for a linear learning rate,

one which is 1=t at time t, the convergence rate has an exponential dependence on 1/(1 -γ).

Algorithm 1:Q-Learning algorithm

Require:

- **States** S = 1...n
- **Actions** A = 1...n
- **Reward Function** : R
- **Learning Rate** $\alpha \in [0, 1)$
- **Discount Factor** $\gamma \in [0, 1]$

Procedure: **Initialize Q-table arbitrarily for all state-action pairs[3.6]**:
while *Q is not converged* **do**
  Start at state S_t;
  Calculate policy π according to Q and epsilon-greedy strategy;
  Apply action;
  Get reward;
  Get new state S_{t+1};
  Update Equation 3.4.1;
  Set $S_t = S_{t+1}$;
end

4 Proposed Architecture

Fig. 4.1 provides an overview of the extended SDN control plane. It comprises the default SDN control plane with a built-in forwarding application and the proposed novel application. Detailed information about the relevant components is provided below.

4.1 Built-in Forwarding Application

Most of the common SDN controllers [31], [17], [16], [15] come with a built-in forwarding application which creates flow rules for which are fed into the SDN switches. In this the novel application communicates with the controller via RESTful API [30].

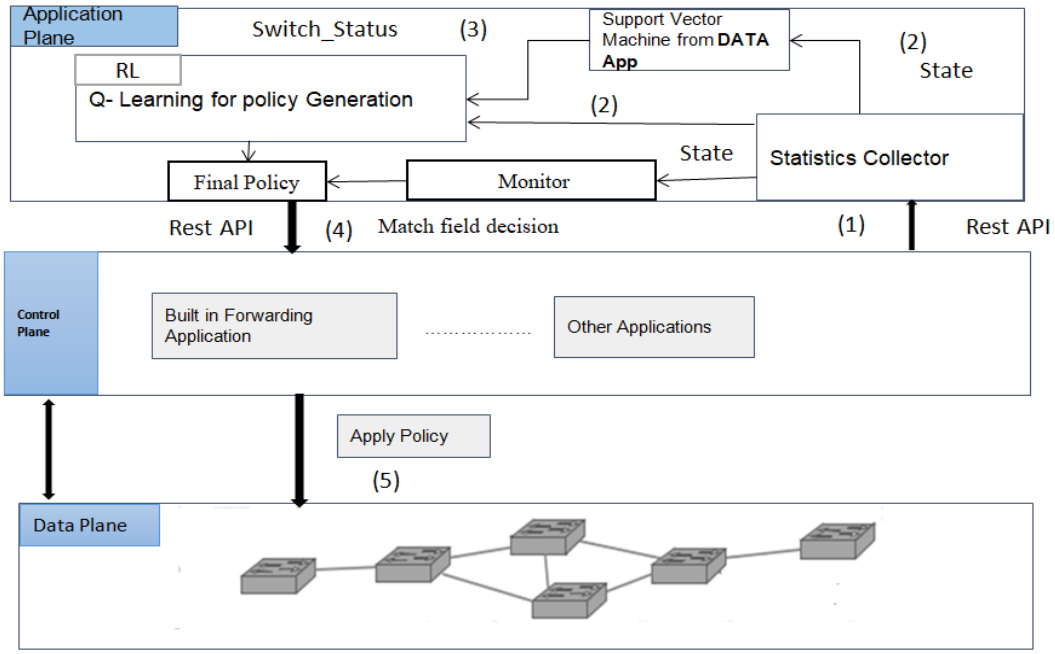


Figure 4.1: Proposed Architecture

4.2 Proposed Application

The proposed application consists of the following three main components: The Statistics Collector , Monitor , Support Vector Machine Engine, Reinforcement Learning Application.

4.2.1 Statistics Collector

The Statistics Collector periodically gets information about traffic flows traversing the SDN switches from the SDN controller via a RESTful API. It sends the collected statistical information to the Support Vector Machine Engine

Algorithm 2: Overflow Monitor to take instant decision on critical network state

input : $f = \sum_{c=1}^{k} f_c$: total number of current flow entries in switch
TotalIP=$\sum_{c=1}^{k} IP_c, f_{threshold}$
output: H= : Match field Combination

```
while True do
    if f > f_threshold then
        if TotalIP > f_threshold/2 then
            H = MMOS;
            Enable MMOS monitor;
        else
            H = Combination of Layer2 and Layer 3;
            Enable RL-agent handover;
        end
    else
        continue;
    end
    return H;
end
```

which has been adopted from the DATA App [23] and to the Reinforcement Learning Application. Both the input of the SVM Engine and the RL Application is a tuple (f,Δf); the first element of which is total number of flow entries at current time and the second one is the difference between two consecutive total number of flow entries.

Monitor

The main goal of this research project is to have as much as details of the incoming packet as possible according to current network state. For this reason in addition to Reinforcement Learning to generate policy of number of match fields at run time this "Monitor" module which is actually a sub-module of Statistics Collector comes into place. It has two parts "Over Flow Detector" and "MMOS Monitor". The "Over Flow Detector" works according to (Algorithm 2). If total number of flows is over switch flow capacity threshold which is 3000 it checks if total number of source IP addresses in all the flows at current time is greater

than 1500. If it is then flow matching scheme is immediately changed to "Destination MAC address only". The idea behind this is that, as Statistics Collector continuously collects flow data within short time interval it can take immediate effect when large unnecessary traffic flow occurs into the network that might cause saturation attack scenario.
If total number of IP addresses is less than 1500 then instead of taking match field decision as "Destination MAC address only" which has only layer 2 information in addition layer 3 match field is kept. The reason behind the number 1500 is that if overflow occurs in the network but for example total number of source ip addresses of incoming packets is 1400 then we might have 1400*2 = 2800 flows in total in network after taking the match field decision as layer-3. In that case we can have immediate remedy of Overflow problem and also will be able to keep more information about the incoming packet.
The "MMOS Monitor" is enabled only if the "Over Flow Detector" sets match field as "Destination MAC address only". The main purpose of this sub-module is to set the matching scheme as Full Matching Scheme as soon as possible according to (Algorithm 3). It checks the packet rate in all links. Whether it is incoming or outgoing. If the total packet rate is less than 300 , suppose 280 and if idle_timeout value is 10 then the multiplication of these both is 2800 which is less than the threshold value of total number of flows. In this scenario it is safe to turn back to full matching scheme.This is what the Algorithm 3 does.

Algorithm 3: MMOS Monitor Algorithm for changing back to FMS

```
input  : TotalR_pkt = Σ_{c=1}^{k} R_pkt:total packet rate of all links in switch
         Switch_Status, f_threshold
output: H= : Match field Combination
while MMOS Monitor Enabled do
    if Switch_Status = 'Good' then
        f_extra = idle_timeout * TotalR_pkt;
        if f_extra + f_new < f_threshold then
            H = FMS;
            Disable MMOS Monitor;
        else
            continue;
        end
    else
        continue;
    end
    return H;
end
```

It is to be noted that both in module "OverFlow Monitor" and "MMOS Monitor" just before changing from Full matching Scheme to Destination MAC address only scheme or

from “MMOS” to “FMS” the monitor module instructs the built-in forwarding application through RESTful API to send of_mod messages to remove all flwo entries from the switch.

4.2.2 Support Vector Machine Engine

The task of the 2-dimensional SVM Engine is to predict the switch performance degradation.The pre-trained SVM Engine from DATA App comprises of two data groups, one is Safe-Zone and the other one is Overloaded-Zone. Whenever a new data sample is fed to the SVM Engine it is checked to which of the two data groups it belongs. If the state lies in Safe-Zone then status of the switch is predicted as 'Good' and if it lies in Overloaded-Zone then the status is 'Bad'.Switch status is monitored by the SVM engine and is sent to the modules that require the info in different time intervals. The reason for choosing the tuple (f, Δf), for evaluating the forwarding performance of a OpenFlow switch are as follows: Control plane performance is widely variable, and it depends on flow table occupancy [14]. The effort for flow searching and matching within a SDN switch is proportional to the number and matching fields of flow entries [28].Moreover, a SDN switch has a maximum capacity for storing the flow entries [2].The difference between number of flows in a switch also directly effects number of PACKET-IN messages sent to the Controller.

4.2.3 Reinforcement Learning

The Reinforcement Learning application applies the Q-Learning algorithm to solve the problem of match field selection.

State Definition, S

In the case of this research project, state is a tuple (f,Δf) consisting of total number of flows in current time and difference between two consecutive flows. It is a continuous state-space. The algorithm that Q-Learning follows will be described in chapter 5 but in short it can be mentioned earlier that Reinforcement Learning has to take an action only when the network state is bad and below flow threshold that is f= 3000. So it can be said that range of value of both f and Δf will be between 0 to 3000.

Action Definition, A

The RL-agent can take 9 actions which is comprised by 9 dictionaries [24] showed in figure 4.2. The dictionaries are kept in a json [25] file . Suppose the RL agent takes action number 4 the json [25] file is parsed and line number 4 from figure 4.2 is selected and passed to the RESTful API of ONOS controller for changing match field. The main reason behind the structure of selecting combination of match fields like this is that in this research project the implementation is done only on ONOS controller and for applying match field change in run time ONOS REST API is used. Fig-4.3 has all the fields that ONOS forwarding application accepts to reconfigure. It is suggested to keep in mind that though only 5 fields are actively changed here; ONOS controller supports 39 match fields in total [26]

1. {"matchTcpUdpPorts": "True","matchVlanId": "True", "matchIcmpFields": "True", "matchIpv4Dscp": "True", "matchIpv4Address": "True","matchDstMacOnly": "false"}
2. {"matchTcpUdpPorts": "false","matchVlanId": "false", "matchIcmpFields": "True", "matchIpv4Dscp": "false", "matchIpv4Address": "True","matchDstMacOnly": "false"}
3. {"matchTcpUdpPorts": "false","matchVlanId": "True", "matchIcmpFields": "True", "matchIpv4Dscp": "false", "matchIpv4Address": "True","matchDstMacOnly": "false"}
4. {"matchTcpUdpPorts": "false","matchVlanId": "false", "matchIcmpFields": "True", "matchIpv4Dscp": "True", "matchIpv4Address": "True","matchDstMacOnly": "false"}
5. {"matchTcpUdpPorts": "false","matchVlanId": "True", "matchIcmpFields": "True", "matchIpv4Dscp": "True", "matchIpv4Address": "True","matchDstMacOnly": "false"}
6. {"matchIcmpFields": "false","matchTcpUdpPorts": "True", "matchVlanId": "false", "matchIpv4Dscp": "false", "matchIpv4Address": "True","matchDstMacOnly": "false"}
7. {"matchIcmpFields": "false","matchTcpUdpPorts": "True", "matchVlanId": "True", "matchIpv4Dscp": "false", "matchIpv4Address": "True","matchDstMacOnly": "false"}
8. {"matchIcmpFields": "false","matchTcpUdpPorts": "True", "matchVlanId": "false", "matchIpv4Dscp": "True", "matchIpv4Address": "True","matchDstMacOnly": "false"}
9. {"matchIcmpFields": "false","matchTcpUdpPorts": "True", "matchVlanId": "True", "matchIpv4Dscp": "True", "matchIpv4Address": "True","matchDstMacOnly": "false"}

Figure 4.2: 9 Match field combinations that RL-agent can choose as an action

and by activating or deactivating any match field ONOS forwarding application and ONOS core automatically activates or deactivates several other match fields. Additionally some Booleans here are actually a combination of several match fields itself. As for example, the Boolean "matchTcpUdpPorts" activates ore deactivates 4 match fields TCP_Source, TCP_Destination, UDP_Source , UDP_Destination. Again, according to Openflow Switch Specification there are different pre-requisite match fields for using some match fields. Such as "matchIpv4Dscp" field has pre-requisite ETH_TYPE=0x0800 or ETH _TYPE=0x86dd but the match field "ETH_TYPE " itself is not possible to be activated by the RESTful API for this reason in the background the forwarding application activates it. For the limitation of emulation environment all match fields related to "IPV6" were omitted.

Reward Function, R

When Statistics collector gets flow information of a switch through RESTful API it gets the match field criteria that has been configured for that device. In figure 4.4 flow information for 2 match field actions taken from figure 4.2 can be seen. The left image of figure 4.4 is the action 1 of figure 4.2 and the right image is the action 2 of figure 4.2. With this comparison we can easily see that changing those boolean values from figure 4.2 increases or decreases number of match field criteria fields. The more match field criteria the more detailed information of incoming packet which is our goal. It is to be noted that as both the images of figure 4.4 were the flow information of a ping command between two hosts for that reason there is no TCP/UDP port information available in the left image.
The reward for Q-Learning for this implementation is determined as the average number of

```
"org.onosproject.fwd.ReactiveForwarding": {
    "matchIpv6Address": "false",
    "matchIpv6FlowLabel": "false",
    "ignoreIPv4Multicast": "false",
    "matchIcmpFields": "true",
    "flowTimeout": "10",
    "matchTcpUdpPorts": "true",
    "recordMetrics": "true",
    "matchDstMacOnly": "false",
    "matchIpv4Address": "true",
    "matchVlanId": "false",
    "matchIpv4Dscp": "false",
    "ipv6Forwarding": "false",
    "packetOutOfppTable": "false",
    "packetOutOnly": "false",
    "flowPriority": "10"
  }
```


Figure 4.3: Fields supported by RESTful API of ONOS to choose as match field

criteria of all flows at current time for the switch. The RL-agent always understands the best action according to the reward value. The RL-agent only knows the integar value of the action it takes. For example action no.1 or action no.5. But it does not know what the action really means. But in this scenario it is expected of the RL-agent to take the action which has most match field combinations. Evaluating the images in Figure 4.4 it can be said that if the agent takes an action that has more match field combinations the average value of criteria of all flows will be higher and if it has less combination than the average value will be lower. So this reward function will always prompt the RL-agent to take the action having most match field combinations according to the network condition.

Q-Learning implementation

The decision of changing of match field combination is done by RL-agent following algorithm 4. The main purpose it to change match field combination only when the switch status is considered as Bad by the SVM engine, flows under threshold and flows are in increasing condition. Because if we change match field combination frequently then it also has negative effect on the network infrastructure and condition. As the standard value of idle_timeout is 10 seconds , the match field change duration is kept 10 seconds if condition one is fulfilled. Idle_timeout is a period of time set in a flow entry. if there is no more incoming packets that matches to the flow entry since last matched packet, then the flow will be removed after idle_timeout seconds. The condition 2 is a special case scenario. Though it is not desirable to change match field combination if the switch status is good but when algorithm 2 after monitoring overflow sets match field combination as layer 3 , to regain more information

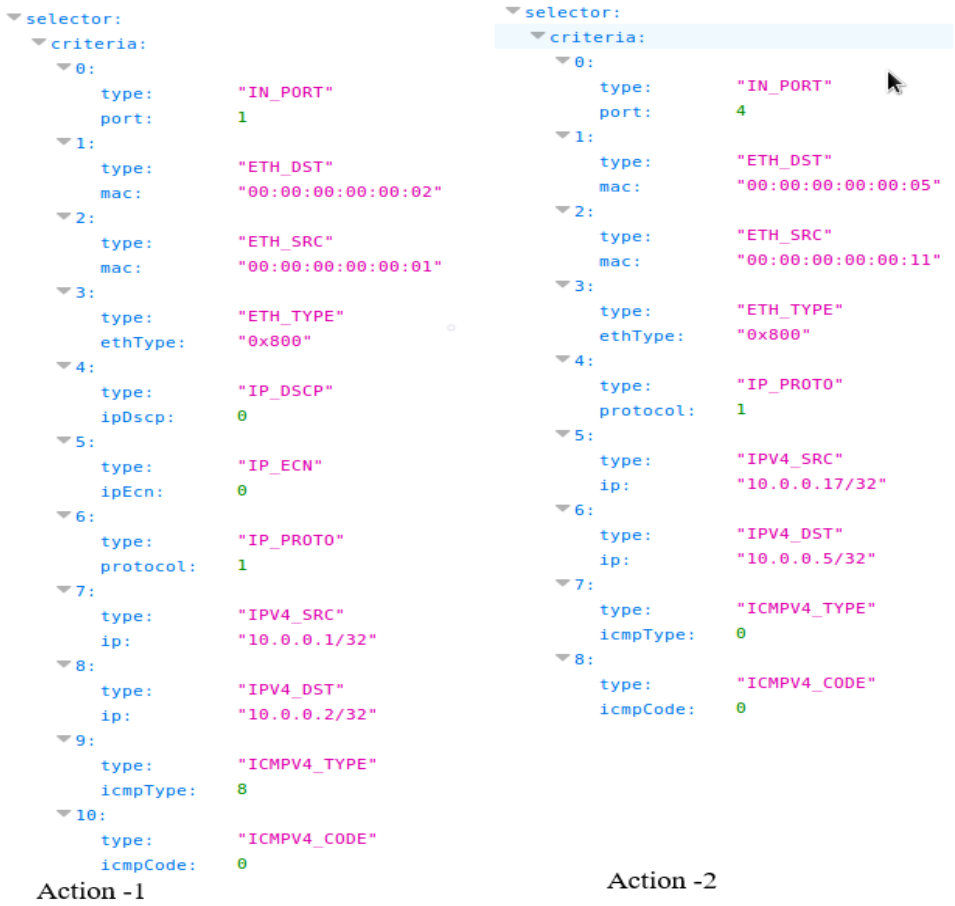


Figure 4.4: Example of two match field combinations after action taken by RL-agent

of the incoming packet the decision making authorization of match field combination is handed over to the Reinforcement Learning agent. For taking an action RL-agent follows epsilon-greedy policy. The value of epsilon will depend on what the experimenter's goal is. But it is suggested to let the RL-agent to take more random action in training period and in implementation period also set the value of epsilon in a way so that it can still do some exploration.

4.3 Operational Work flow

Initially the Statistics Collector sends a get request to the SDN controller for network information(step (1) in Fig. 4.1). In regular time intervals(observation period) it also counts the total number of flows and measure the flow number changes in order to send the tuple (f, Δf) to the SVM Engine , Monitor and the RL agent (step (2) in Fig. 4.1).As illustrated in Fig 4.5 , in case the SVM engine predicts the switch status good it sends

Algorithm 4: Q-Learning implementation

Require:

- **States** S = $(f_1, \Delta f_1)$.....$(f_n, \Delta f_n)$
- **Actions** A = 1...9
- **Reward Function** : R
- **Learning Rate** $\alpha \in = 0.6$
- **Discount Factor** $\gamma \in = 0.7$

Procedure: **Initialize Q-table or use a pre-trained Q-table**:
while *Q is not converged* **do**
  Start at state S_t;
  Calculate policy π according to Q and epsilon-greedy strategy;
  if *Switch_Status = "Bad"* && $f < 3000$ && $\Delta f > 0$ **then**
    Apply action;
  else
    continue;
  end
  if *Switch_Status ="Good"* && $f < 3000$ && *RL-agent handover enabled* **then**
    Apply action;
  else
    continue;
  end
  Get reward;
  Get new state S_{t+1};
  Update Equation 3.4.1;
  Set $S_t = S_{t+1}$;
end

Switch Status as 'Good' to the RL-agent and in case switch status bad it sends information as 'Bad' to the RL-agent. In parallel the RL-agent initializes the Q-table. When the training period goes on, the Q-table is initialized arbitrarily for all state-action pairs. In every iteration when the Q-table is updated it is automatically saved in a file for later use. For Q-learning policy iteration while training the value of epsilon of epsilon-greedy policy is set as 0.8 so that 80% time it takes random action and 20% time it takes the best action from the updated Q-table. On the other hand when a pre-trained Q-table is used then the value of epsilon can be set to 0.2 so that 80% time it takes the best action but still 20% time it can explore and take random action or it can be set as 0 so that all the time RL-agent takes the best action. In every iteration decided action by the RL-agent is applied via the RESTful

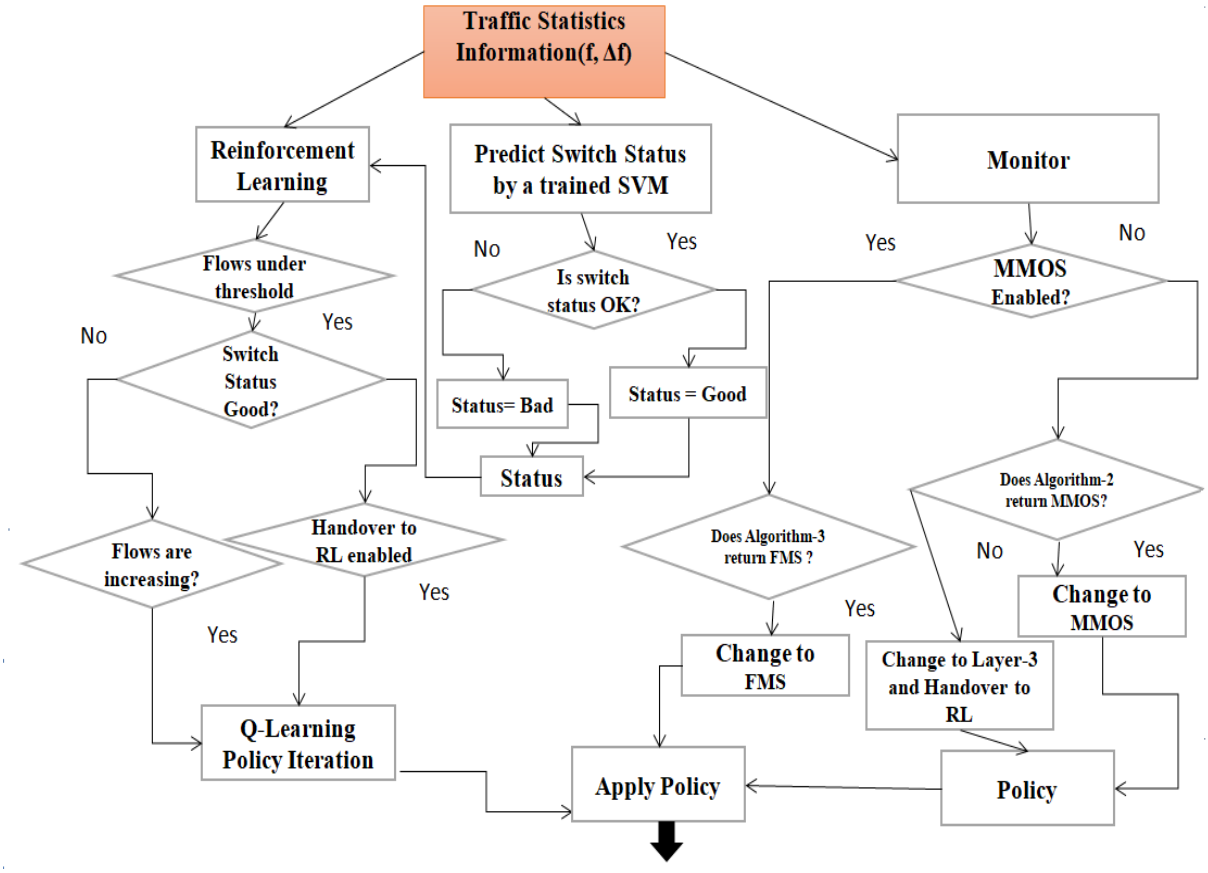


Figure 4.5: Work flow of the proposed architecture

API (step (4) in Fig. 4.1). It is also to be noted that the condition that whether RL-agent will take any action or not will follow algorithm 4. Monitor also follows the algorithm 2 and algorithm 3 to take any match field action.

5 Performance Evaluation

5.1 Scenario Setup

The MaxiNet framework [33] is applied to emulate SDN network comprising 4 hosts and 1 server (see Fig 5.1). The emulated topology of SDN network runs within one Linux PC and is controlled by a remote ONOS SDN controller running on another Linux PC (see Fig 5.2). The ONOS SDN controller , RL-App and SVM Engine all are placed in the same Linux PC.The hosts are running within Linux containers using custom made docker [34] image "randompacketgenerator" and the server run using Apache Web server image. Whenever the topology(figure 5.2) is generated by MaxiNet 3 hosts automatically start to generate random packet emulating normal behavior of real network. For generating random traffic in the "randompacketgenerator" docker image hping3 [35] tool is used. Random packet generated by the hosts in the network was used to train the Q-table.

It has been observed that a SDN switch starts getting overloaded and can not handle new flow rules after the capacity 3000 flow rules is filled up [28]. The safety threshold of packet rate R is 300 packets per second assuming each packet is unique. Accordingly, for evaluation purpose traffic generation is divided into three levels: low (R=100), medium (R=200) and high (R=300).

For analyzing performance several experiments were carried out with 3 flow matching strategies: Match Destination MAC only, Full matching scheme (FMS) , DATA Application [28], and the Reinforcement Learning Application with dynamic match fields. For performance analysis traffic was generated toward the server with different load levels R=(100,200,300). For evaluation purpose packet generation 'DDoS Botnet Simulator'(Bonesi) [36] tool is used and for that purpose custom docker image "Zombie" is used . During the experiment total number of flow entries in the SDN network were traced and the average number of packet_in messages per second to the ONOS controller was extracted. In performance analysis period Q-table is initialized from the per-trained Q-table that is saved as a text file.

5.1.1 Packet Generators

5.1.2 Random Packet Generator

Random packet is generated using hping3 tool. It is a command-line oriented TCP/IP packet assembler/analyzer. The interface is inspired to the ping(8) unix command, but hping isn't only able to send ICMP echo requests. It supports TCP, UDP, ICMP and RAW-IP protocols, has a traceroute mode, the ability to send files between a covered channel, and many other features.

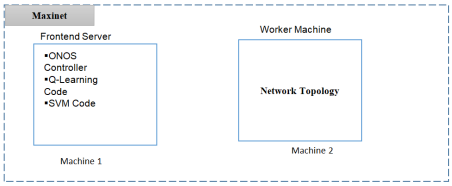


Figure 5.1: Experimental Setup

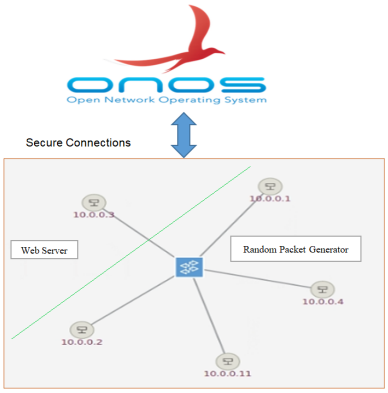


Figure 5.2: Topology

```
$ hping3 IP Proto –S –V –p 80 –i uRate –c To–Packets –d Size
```

In this command 5 variables are changed to generate random packet.

- IP - "Here target IP address to where the packet is sent is placed".
- Proto - "Here which protocol packet is to be generated is placed. i.e TCP ,ICMP , UDP".
- Rate - "Packet rate is chosen randomly in each iteration in a python code from 10 packet per second to 300 packet per second".
- To-Packets - "It is the total number of packets send in each iteration. It is also chosen randomly from 2000 to 20000".
- Size - "It is packet body size in byte. It is also chosen randomly from 64 to 1048".

The 3 hosts generate TCP, UDP and ICMP packet within random time interval 4 seconds to 19 seconds.

5.1.3 Botnet Simulator

BoNeSi generates ICMP, UDP and TCP (HTTP) flooding attacks from a defined botnet size (different IP addresses). BoNeSi is highly configurable and rates, data volume, source IP addresses, URLs and other parameters can be configured. The main reason of using Bonesi in evaluation purpose is that it has an additional functionality to fix number of IP addresses to be used as source IP address while generating traffic. This control is required to check whether [Algorithm 2] works perfectly. An example bonesi command is:

```
$ bonesi –i file –p tcp –r Rate –c 10000 –d t4–eth0 –u IP:PORT
```

The parameters that are changed in here in different evaluation purpose.

- IP:PORT - “Here target IP address and port to where the packet is sent is placed”.
- Rate - “Packet rate is chosen 100,200,300 for different evaluation purpose”
- Size - “Text file name where the list of IP addresses reside. The default file name is 50k-bots.Additionally for evaluation purpose 1k-bots were created where list of 1000 IP addresses were kept”.

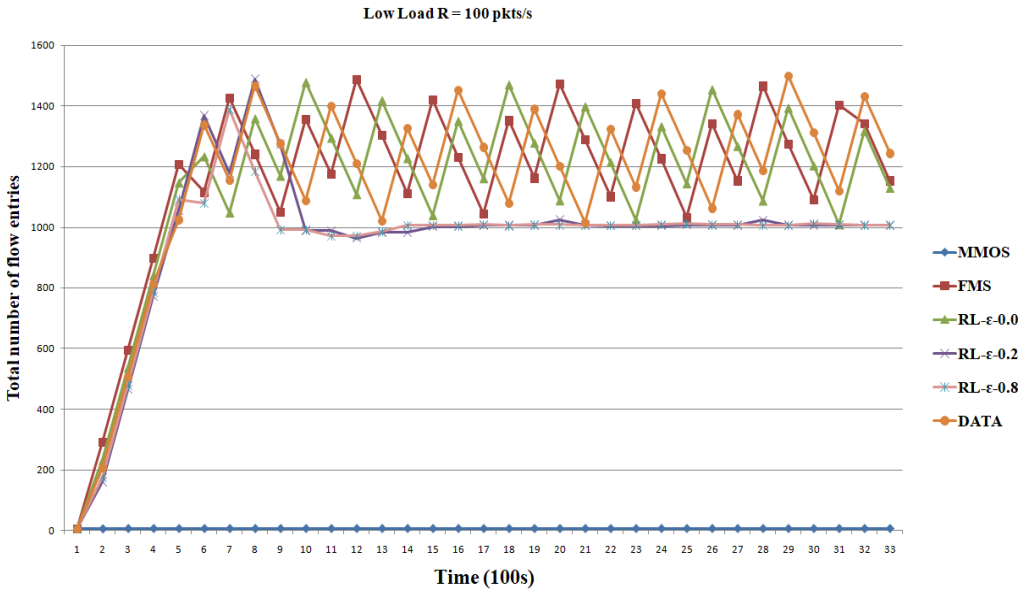


Figure 5.3: Total number of flow entries in low load scenario

5.2 Result Analysis

For evaluation purpose in addition to comparing with MMOS,FMS and DATA APP different epsilon-values of Q-Learning algorithm were also tested. This comparison might help to choose which value of epsilon is compatible to which network condition or purpose. Data for both total number of flows and total packet_in messages were taken every 3 seconds. So in figure 5.3 , 5.4 and 5.5 single time unit is 3 seconds.

5.2.1 Total number of flow entries in the SDN Network

Figure 5.3 , 5.4 and 5.5 shows that in all kind of load MMOS scheme has very low amount of flow entries.In low load and medium load scenario FMS, DATA, RL-0.0 are supposed to have the same behaviour as the total number of flows are below critical level and RL-agent is configured to take the best action which is full matching scheme in this case. In low load

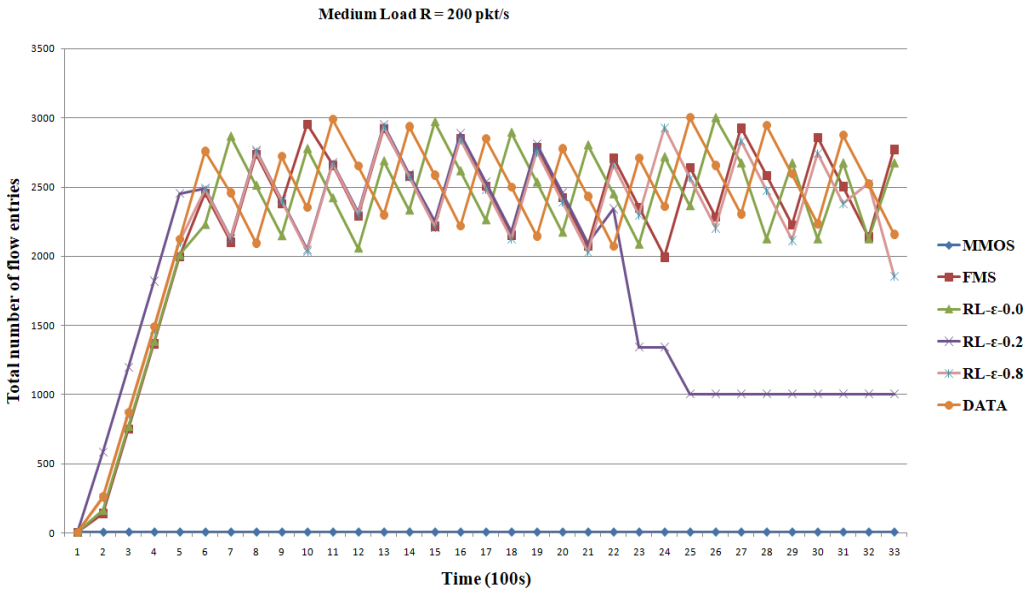


Figure 5.4: Total number of flow entries in medium load scenario

epsilon value 0.2 and 0.8 scenario after 24 seconds the RL-agent took random action and that random action caused constant SVM output "Good". As incoming packet rate was constant according to Algorithm 4 RL-agent did not take any action to change match field and so total number of flows was constant. This was supposed to be the same in case of medium load but in the figure 5.4 for only epsilon value 0.2 the same result is visible but not in the case of epsilon value 0.8. As we have discussed in [section 3.3.2] in epsilon value 0.8 scenario the RL-agent is supposed to take 80% random and 20% best action. As there is no time parameter of randomness it can be assumed that throughout the total 100 second time frame it was executing that 20% best action which is full matching scheme. To discuss about the result of high load from figure 5.5 the time frame can be divided into 3 parts. First one is , 3 seconds to 27 seconds.When FMS is set switch being overloaded could not install any flow rules and showed error and dropped all the packets. According to its algorithm the DATA app reaching threshold changed matching scheme from FMS to MMOS. As in bonesi 1000 IP addresses were set monitoring overflow RL-agent set the matching scheme upto layer-3 follwing algorithm 2. The second time slot is 27 seconds to 48 seconds. In this slot following algorithm -3 RL-agents started taking action according to respective epsilon values. In time slot 48 seconds to 81 seconds while taking respective actions RL-agents again encountered overflow and went back to layer-3 match field combination.

5.2.2 Average packet_in message rate to the ONOS controller

In Table 5.1 the average number of packets per second (packet_in rate) arriving at the built-in forwarding application requesting new flow rules is shown. It is very clear that in contrast

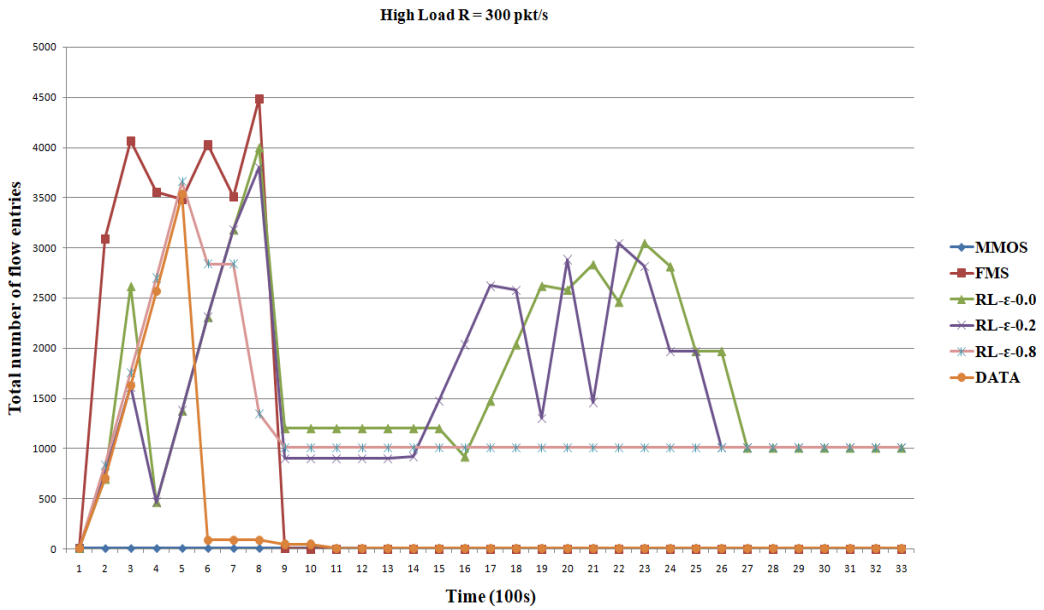


Figure 5.5: Total number of flow entries in high load scenario

with FMS or DATA App in RL-0 where only the best action is taken has an acceptable packet_in rate. The result for RL-0.2 and RL-0.8 are supposed to be better also but it can be seen only when data for longer period of time is observed.

Table 5.1: Average packet_in rate (pkt s/s) to the SDN controller for different traffic flow rule aggregation schemes and traffic loads R

Packet Rate	MMOS	FMS	DATA	RL-0	RL-0.2	RL-0.8
100 R	0.03125	293.333	293.3611	253.4839	38.4764	44.425943
200 R	0.03125	592.9815	579.5682	285.0172	184.7286	261.5205
300 R	0.03125	894.7551	75.28571	262.2264	213.3373	39.9619

6 Conclusion and Future Work

The goal of this research project has been to keep enough incoming traffic information according to network state. For meeting the requirement a combination of supervised learning and reinforcement learning technique have been used. After result analysis it is clear that the proposed methodology out performed two legacy match field combination system MMOS and FMS. It also outperformed the destination aware adaptive traffic application in controlling total number of flows, average packet_in message and also in degree of freedom at determining combination of match fields. The proposed architecture of this research project can immediately take action in overflow condition to protect the network infrastructure. Additionally, it also monitors periodically after being overflown whether network is good enough for installing full matching scheme. Standard idle_timeout value is 10 seconds and RL-agent also takes actions in 10 seconds time interval when it is required. Moreover, when decision of match field changing is taken by RL-agent no of_mod message is sent to the switch to remove all flows. In this way putting extra load on the infrastructure is avoided. There are still some limitation in the proposed application. As Q-Learning is an online learning algorithm and it works best in finite state-action space to get a in real network scenario it will take a lot of iterations for Q-table to converge. Currently RL-agent thinks of the best action which has the most average flow criteria. But in scenario where repeatedly high rate unique incoming packet occur Algorithm 4 is not the best solution then. The main reason of it is that the base of Q-Learning is Markov Decision process which only has memory of present and future state. It does not have memory of past states and decision. For overcoming this problem in future Deep Recurrent Q-Network(DRQN) can be used. This gives RL-agent memory of its past states and decisions. Furthermore, in real network the environment is partially observable. For this reason for fastest convergence of q-table DRQN can be applied. In this research project in the implementation step only emulated network had only one switch and the determined match field combination was deployed in the whole network. Therefore, in future research reinforcement learning can be implemented to deploy match field combination to individual switches according to that switch's status and also overall network status. In addition, in performance analysis incoming packet from legacy internet and enterprise network can be also included. Moreover, the monitor module could be extended two act as an intrusion detection application and decision of Reinforcement Learning agent can also vary in those special scenario.

Bibliography

[1] WenjuanLi , Weizhi Meng, Lam For Kwok, *"A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures"*, *Journal of Network and Computer Applications*, Volume 68, Pages 126-139,June 2016

[2] Myriana Rifai, Nicolas Huin ,Christelle Caillouet, Frederic Giroire, Joanna Moulierac , Dino Lopez Pacheco , Guillaume Urvoy-Keller, *"Minnie: An SDN world with few compressed forwarding rules"*, *Computer Networks*, vol. 121, pp. 185–207, 2017.

[3] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid, and H. J. Chao, *"STAR: Preventing flow-table overflow in software-defined networks"*, *Computer Networks*, vol. 125, pp. 15–25, 2017.

[4] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid, and H. J. Chao, *"A low overhead flow-holding algorithm in software-defined networks"*, *Computer Networks*, vol. 125, pp. 15–25, 2017.

[5] S. Q. Zhang, Q. Zhang, A. Tizghadam, B. Park, H. Bannazadeh, R. Boutaba, and A. Leon-Garcia, *"TCAM space-efficient routing in a software defined network"*, *Computer Networks*, vol. 125, pp. 26–40, 2017.

[6] Yiheng Su , Ting Peng , Xiaoxun Zhong ,Lianming Zhang, *"Matching model of flow table for networked big data"*, *Computing Research Repository (CoRR)* , vol. abs/1712.09158 , 2017 .

[7] M. Latah and L. Toker, *"Artificial intelligence enabled software-defined networking: a comprehensive overview"*, *IET Networks*, 2018.

[8] P. T. Congdon, P. Mohapatra, M. Farrens, and V. Akella, *"Simultaneously Reducing Latency and Power Consumption in OpenFlow Switches"*, *IEEE/ACM Transactions on Networking*, vol. 22, no. 3, pp. 1007–1020, 2014.

[9] B. Leng, L. Huang, X. Wang, H. Xu, and Y. Zhang, *"A mechanism for reducing flow tables in software defined network"*, *2015 IEEE International Conference on Communications (ICC)*, 2015.

[10] S. Bhowmik, M. A. Tariq, A. Balogh, and K. Rothermel, *"Addressing TCAM Limitations of Software-Defined Networks for Content-Based Routing"*, *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems - DEBS 17*, 2017

[11] N. Kang, Z. Liu, J. Rexford, and D. Walker, *"Optimizing the 'one big switch' abstraction in software-defined networks"*, *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies - CoNEXT 13*, 2013

[12] T. A. Pascoal, Y. G. Dantas, I. E. Fonseca, and V. Nigam, *"Slow TCAM Exhaustion DDoS Attack"*, *ICT Systems Security and Privacy Protection IFIP Advances in Information and Communication Technology*, pp. 17–31, 2017.

[13] 13. L. W. Cheng S. Y. Wang , *"Application-aware SDN routing for big data networking"*, *IEEE Glob. Commun. Canf GLOBECOM*, 2015.

[14] M. Kuźniar, P. Perešíni, D. Kostić, and M. Canini, *"Methodology, measurement and analysis of flow table update characteristics in hardware openflow switches"*, *Computer Networks.*, vol. 136, pp. 22–36, 2018.

[15] *"Floodlight Controller"*, `https://floodlight.atlassian.net`

[16] *"OpenDaylight Controller"*, `https://wiki.opendaylight.org`

[17] *"Ryu Controller"*, `https://ryu.readthedocs.io`

[18] D. Kreutz, F. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, *"Software-defined networking: A comprehensive survey"*, *Proc. IEEE*, vol. 103, no. 1, pp. 14-76, Jan. 2015.

[19] Microsoft, *"Skype for business"*, `https://docs.microsoft.com/en-us/skype-sdk/sdn/articles/skype-for-business-sdn-interface`

[20] Hewlett-Packard, *"HP Network Protector"*, `https://support.hpe.com/hpsc/doc/public/display?docId=emr_na-c04626978`

[21] Sayon Dutta, *"Reinforcement Learning with TensorFlow"*, ISBN 139781788835725 , April 2018.

[22] R. Li, Z. Zhao, Q. Sun, C.-L. I, C. Yang, X. Chen, M. Zhao, and H. Zhang, *"Deep Reinforcement Learning for Resource Management in Network Slicing"*, *IEEE Access*, vol. 6, pp. 74429–74441, 2018.

[23] Even-Dar, Eyal and Mansour, Yishay, *"Learning Rates for Q-learning"*, *J. Mach. Learn. Res.*, December 2004.

[24] Guido van Rossum, *"Dictionary in Python"*, `https://docs.python.org/3/tutorial/datastructures.html?highlight=dictionary`

[25] JavaScript Object Notation, *"Explaination of JSON file format"* `https://en.wikipedia.org/wiki/JSON`

[26] OpenFlow, *"OpenFlow Switch Specification"*, Version 1.3.0 , June 25,2012. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[27] T.V.Phan , Mehrdad Hajizadeh, Nguyn Tun Khi, T. Bauschert, *"eATM: A Practical Extension for Adaptive Traffic Matching in Software Defined Networks"*, Unpublished.

[28] T.V.Phan , Mehrdad Hajizadeh, Nguyn Tun Khi, T. Bauschert, *"Destination-aware Adaptive Traffic Flow Rule Aggregation in Software-Defined Networks"*, 3rd Workshop on Software-Defined Networking and Network Function Virtualization for Flexible Network Management, Munich, Germany Accepted , 2019 .

[29] Sutton, Richard S., and Andrew G. Barto, *"Reinforcement Learning an Introduction"*, *A Bradford Book*, 1998.

[30] *"Description of ONOS Controller's RESTful API"*, https://wiki.onosproject.org/display/ONOS/Appendix+B%3A+REST+API

[31] *"Description of ONOS Controller"*, www.onosproject.org

[32] R. Bellman, *"Dynamic programming and stochastic control processes"*, *Information and Control*, vol. 1, no. 3, pp. 228–239, 1958.

[33] P. Wette, M. Draxler, and A. Schwabe, *"MaxiNet: Distributed emulation of software-defined networks"*, *2014 IFIP Networking Conference*, pp. 1–9, June 2014.

[34] K. Jangla, *"Docker Images"*, pp. 55–76, 2018.

[35] *"hping3(8) - Linux man page"*, https://linux.die.net/man/8/hping3.

[36] *"Botnet Simulator"*, https://github.com/Markus-Go/bonesi.