Q-DATA: Enhanced Traffic Flow Monitoring in Software-Defined Networks applying Q-learning

Trung V. Phan[†], Syed Tasnimul Islam[†], Tri Gia Nguyen[§] and Thomas Bauschert[†]
[†]Chair of Communication Networks, Technische Universität Chemnitz, 09126 Chemnitz, Germany
[§]Faculty of Information Technology, Duy Tan University, Danang 50206, Vietnam
Email: trung.phan-van | thomas.bauschert@etit.tu-chemnitz.de

***Abstract*—Software-Defined Networking (SDN) introduces a centralized network control and management by separating the data plane from the control plane which facilitates traffic flow monitoring, security analysis and policy formulation. However, it is challenging to choose a proper degree of traffic flow handling granularity while proactively protecting forwarding devices from getting overloaded. In this paper, we propose a novel traffic flow matching control framework called *Q*-DATA that applies reinforcement learning in order to enhance the traffic flow monitoring performance in SDN based networks and prevent traffic forwarding performance degradation. We first describe and analyse an SDN-based traffic flow matching control system that applies a reinforcement learning approach based on *Q*-learning algorithm in order to maximize the traffic flow granularity. It also considers the forwarding performance status of the SDN switches derived from a Support Vector Machine based algorithm. Next, we outline the *Q*-DATA framework that incorporates the optimal traffic flow matching policy derived from the traffic flow matching control system to efficiently provide the most detailed traffic flow information that other mechanisms require. Our novel approach is realized as a REST SDN application and evaluated in an SDN environment. Through comprehensive experiments, the results show that—compared to the default behavior of common SDN controllers and to our previous DATA mechanism—the new *Q*-DATA framework yields a remarkable improvement in terms of traffic forwarding performance degradation protection of SDN switches while still providing the most detailed traffic flow information on demand.**

***Index Terms*—Traffic Flow Monitoring, Reinforcement Learning, *Q*-learning algorithm, Network Statistics and Software-Defined Networking.**

I. Introduction

Software Defined Networking (SDN) is a new networking concept, which provides enormous capabilities for dynamic network traffic control and management [1]. By detaching the control plane from the data plane, it removes some restrictions of legacy networks. A centralized entity called SDN controller has a global network view that allows for a policy-based traffic management and a faster and more dynamic response to network state and traffic variations [1].

Although there are numerous available mechanisms for traffic analysis, traffic flow management and resilience [2] in SDN based networks, some significant challenges still remain to be addressed [3]. In particular, adapting the granularity of traffic forwarding while protecting forwarding devices from an overflow situation is one critical issue. As most current traffic management approaches rely on the default flow matching strategies of the available SDN controllers, it is difficult to perform traffic forwarding with variable granularity. For example, the *Open Network Operating System* (ONOS) [4] and *OpenDaylight* (ODL) [5] SDN controllers, by default, apply *Reactive Forwarding* based on layer 2 information, which uses the *MAC address* for flow matching only. Therefore, an incoming packet is matched to a flow entry by just using its layer 2 destination address. However, security and traffic monitoring mechanisms require traffic flow handling based on layer 3 and layer 4 information. A flow matching scheme that uses MAC and IP (and maybe also TCP/UDP) header fields requires much expensive TCAM memory for storing the respective flow rules [6] and in case the incoming traffic flow pattern is highly dynamic, this might lead to a significant degradation of the traffic forwarding performance in the data plane. Besides, the control plane might be affected because of a large number of *packet_in* messages [7].

In this paper, we propose a reinforcement learning based traffic flow matching control framework called *Q*-DATA, to enhance the performance of traffic flow monitoring in SDN based networks and proactively prevent flow-table overflow in SDN switches. We first describe a traffic flow matching control mechanism that applies a reinforcement learning based approach (*Q*-learning) for optimizing the traffic flow granularity in the data plane. It also considers the forwarding performance status of SDN switches derived by a Support Vector Machine algorithm. Next, we outline the *Q*-DATA framework that incorporates the optimal traffic flow matching policy derived from a *Q*-learning based Traffic Flow Matching Policy Creation module to efficiently provide detailed traffic flow information that other mechanisms, e.g., for traffic engineering, traffic monitoring, and intrusion detection, require. In particular, a Support Vector Machine algorithm is utilized to simultaneously analyse the current network traffic and predict the SDN switch performance degradation. Based on the prediction result the *Q*-learning based Traffic Flow Matching Policy Creation module issues an optimum action on changing the traffic flow matching scheme. Note that this proposal partially inherits[1] our previous work [8] which is explained later on.

[1]We leverage the use of the Support Vector Machine based performance degradation prediction mechanism and the traffic flow matching scheme change based on destination hosts from our previous study [8].

The paper is structured as follows. Section II provides related work and our previous study. Section III presents our approach for maximizing the level of traffic flow granularity based on Q-learning. Section IV explains the Q-DATA framework in detail. Our experiments and results are outlined in section V and section VI, respectively. Section VII provides a summary and outlines some ideas for future studies.

II. Related Work

A. Existing Methods for Flow Rule Control and Management in Software Defined Networks

Many studies already addressed issues related to flow rule installation and management in SDN switches - a topic that is of high interest in the SDN research community [9]–[16]. There exist several approaches for controlling TCAM utilization with the primary target of flow rule compression or aggregation.

The authors in [9] propose an online routing scheme that constrains flow-table resources in SDN switches. Similarly, in [10] the objective is to maximize the number of flow entries in the data plane considering the limited flow-table space in SDN switches. Nonetheless, these methods do not address the problem of protecting the network infrastructure when a sudden traffic increase is happening.

The studies in [11]–[13] deal with TCAM resource management. In particular, [11] outlines a solution for flow-table size reduction based on three criteria including Consistency[2], Absoluteness[3] and Accuracy[4]. An incoming packet classification approach is presented in [12] which exploits the temporal locality of network traffic to predict the flow of incoming packets. If the prediction is correct the forwarding latency and power consumption can be reduced trough avoiding the full flow-table lookup process in the TCAM. Rifai et al. introduce a framework called MINNIE [13] for flow-table compression using wildcard rules. Furthermore, the authors in [14] argue that for storing simple packet forwarding rules based on MAC addresses or VLAN IDs cheap SRAM memory is sufficient, while more complex matching rules (with more matching fields) might require the use of fast but expensive TCAM memory. Considering this, the amount of TCAM memory in SDN switches can be significantly reduced. The solution outlined in [15] applies the concept of flow rule aggregation by restructuring the matching fields. By that, the number of flow rules can be significantly reduced. Another approach for dynamic flow matching is proposed in [16] where a flow matching policy considering the DSCP values of different traffic types is applied.

The mentioned solutions only focus on flow-table size reduction and on enhancing the data plane forwarding performance. Contrary to our solution they do not consider the possibility of adaptively changing the traffic flow matching scheme depending on the current network state and the level of detail of traffic flow information that other mechanisms, for e.g., traffic engineering and monitoring, require.

B. Destination-aware Adaptive Traffic Flow Rule Aggregation

In our previous work, we proposed a destination-aware adaptive traffic flow rule aggregation solution named DATA [8] for adapting the number of flow entries in SDN switches according to the level of detail of traffic flow information that other mechanisms require and at the same time preventing SDN switch performance degradation.

We analyzed common SDN flow matching strategies of the ONOS [4] and ODL [5] SDN controllers and their implications. We denoted the MAC Matching Only Scheme as *MMOS* strategy and the Full Matching Scheme as *FMS* strategy. Using MMOS the ability to track and monitor network traffic for security or forensic analysis is limited, whereas applying the FMS strategy can result in significant degradation of the forwarding performance or even to an SDN switch outage in case the maximum number of flow entries is reached. To solve this problem, we applied a 2-dimensional Support Vector Machine (SVM) algorithm [17] to anticipate the switch performance degradation well before it occurs and to trigger the flow matching scheme change in time. After analyzing the SDN switch performance and if a potential forwarding performance degradation is figured out, the Analyzer applies Algorithm 3 (see appendix section) to find some destination hosts whose associated flows are most critical regarding the forwarding performance of the SDN switch, i.e., have the most flow entries. Afterwards the Analyzer co-operates with the built-in forwarding application of the SDN controller to conduct the traffic flow matching scheme changes[5] for these destination hosts. These actions can be either to change to MMOS if a sudden increase or an overflow of the flow-table space in an SDN switch is expected or to return to FMS in case there is no overflow risk (see Algorithm 4 in appendix section).

Our DATA approach outperforms legacy flow rule matching schemes in terms of the number of flow entries in the SDN switches, the average $packet\_in$ rate in the SDN control plane and the number of errors and exceptions.

Although, the DATA method has many advantages in comparison to legacy approaches, some issues still should be addressed for further improvement - e.g., the limited number of only two flow matching schemes (MMOS and FMS) and the lack of feedback about the impact of the respective flow matching scheme on the network performance. Therefore, in this paper, we propose a novel traffic flow matching control mechanism that can flexibly switch between many different flow matching schemes based on the current network state. The novel scheme provides a much higher level of detail of traffic flow information even in case of high traffic load, while

[2]All the flows must be allotted with the same actions after the reduction.

[3]All the manually added rules must be executed in the highest priority.

[4]The statistics data must be accurate all the time.

[5]In order to perform a traffic flow matching scheme change for a destination host, the built-in forwarding application firstly deletes all flow entries related to the destination host in the switch, then it installs a flow entry with a new match field combination in the switch.

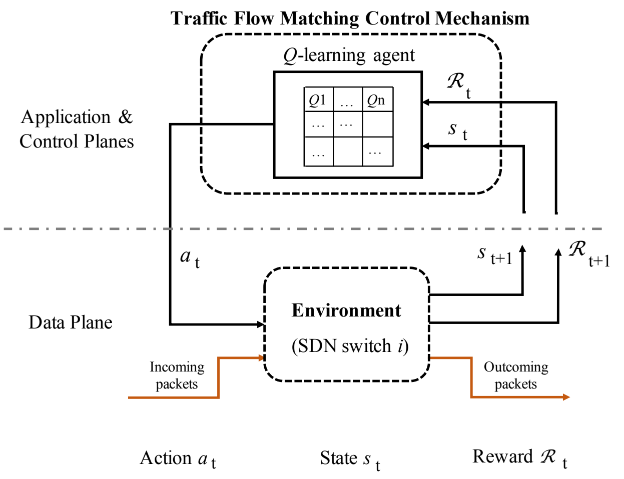


Fig. 1. Reinforcement learning-based model of a traffic flow matching control system in SDN based networks

effectively preventing flow-table overflow and degradation of the data plane forwarding performance.

III. Maximizing the traffic flow granularity applying a Q-learning algorithm

Fig. 1 shows a traffic flow matching control mechanism based on reinforcement learning. The traffic flow matching control mechanism is realized as an SDN application and the environment is represented by the devices in the data plane, i.e., the SDN switches. In the following, we have a look at a single SDN switch i representing the environment and investigate the traffic flow matching control mechanism. We assume that a state s_t of the SDN switch i at a time t is represented by a tuple including the total number of current flow entries (f_i) and the number of flow entry changes (Δf_i) between two consecutive observations; the *long-term* goal of the control system is to maximize the traffic flow granularity in state s_t of SDN switch i while protecting the switch from forwarding performance degradation.

Regarding the system operation, in a given state s_t the control mechanism initiates an action a_t to change the traffic flow matching scheme in SDN switch i. Afterwards, a new state s_{t+1} is observed and a reward $\mathcal{R}_t$ is calculated as soon as the change of the traffic flow matching scheme is executed. Then the next action a_{t+1} is applied to the environment in order to achieve the long-term goal. The traffic flow matching control mechanism based on reinforcement learning operates via agent-environment interaction and can be modeled as a Markov Decision Process (MDP) [18]. In the following, the MDP model is outlined in detail.

1) State Space: The state space of SDN switch i is defined as follows:

$$\mathcal{S}_i \triangleq \{(f_i, \Delta f_i) : 0 < f_i \leq f_{cap_i}; -f_{cap_i} \leq \Delta f_i \leq f_{cap_i}\}, \quad (1)$$

where f_i is the current total number of flow entries in switch i, Δf_i is the number of flow entry changes between two consecutive observations and f_{cap_i} is the maximum number of flow entries in switch i. The state of SDN switch i is defined as tuple $s = (f_i, \Delta f_i) \in \mathcal{S}_i$. In our previous study [8] we already discussed the reasons for choosing the tuple $(f_i,\Delta f_i)$ as the representative for the state of an SDN switch. The reasons can be summarized as follows: the effort for flow entry searching and matching in an SDN switch is proportional to the number of matching fields and an SDN switch has a maximum capacity (f_{cap}) for storing the flow entries. Accordingly, the change of the number of flow entries indicates the control plane load (wrt. *of_mod* and *of_removed* messages sent between SDN controller and switch) affecting both the SDN switch and the controller performance.

2) Action Space: $\mathcal{F} = \{g_1, g_2, ..., g_m\}$ denotes a list of all feasible match field combinations, e.g., g_m = <"*matchTcpUdpPorts*", "*matchIpv4Address*", "*matchVlanId*",...> in case of the ONOS controller [4]. The action space for changing the traffic flow matching scheme in the SDN switch i is defined by

$$\mathcal{A}_i \triangleq \{a : a \in \mathcal{F}\}, \quad (2)$$

where a represents a traffic flow matching scheme change related to a destination host (as discussed in section II-B) in SDN switch i.

3) Immediate Reward Function: On the one hand, whenever, through executing an action, the total number of current flow entries in the SDN switch i reaches the limit f_{cap_i} (which then leads to a performance degradation), the traffic flow matching control system should not get any reward for this action. On the other hand, the more matching fields a flow entry contains, the more detailed information is available for that flow. Hence, we determine the immediate reward as the average number of matching fields of all flow entries in the SDN switch i:

$$\mathcal{R}_i(s,a) = \begin{cases} \frac{\sum_{x=1}^{f_i} \Theta_x}{f_i}, & 0 < f_i < f_{cap_i}, \\ 0, & f_i = f_{cap_i}, \end{cases} \quad (3)$$

where f_i is the current total number of flow entries in the switch i, Θ_x is an integer number representing the number of enabled match fields in flow entry x.

4) Optimization Formulation: We define an optimization problem to acquire the optimal policy applicable in state s, denoted by $\pi^*(s)$, that maximizes the long-term reward, i.e., the traffic flow granularity in the SDN switch i while protecting it from forwarding performance degradation. In particular, in state s, the agent issues an optimal action a to get close to or reach the long-term reward. The MDP under consideration is finite and the state space $\mathcal{S}_i$ contains at maximum $2f_{cap_i}^2$ states. The optimization problem is formulated as follows:

$$\max_{\pi} \quad \{\Re(\pi)_i = \sum_{t=1}^{2f_{cap_i}^2} \mathbb{E}(\mathcal{R}_i(s_t, \pi(s_t))) : \mathcal{R}_i \in \mathbb{R}; s_t \in \mathcal{S}_i; \pi(s_t) \in \mathcal{A}_i\} \quad (4)$$
$$\text{subject to} \quad SVM(s_t) = Good, \forall s_t \in \mathcal{S}_i,$$

where $\Re(\pi)_i$ is the cumulative reward for SDN switch i under a policy π, $\mathcal{R}_i(s_t,\pi(s_t))$ is the immediate reward associated with policy π for a switch i at iteration t, and SVM(s_t) is the result of the Support Vector Machine algorithm predicting the forwarding performance of the SDN switch. A "*Good*" result of the SVM algorithm means that the switch can handle the current number of flow entries without any forwarding performance problems.

In order to solve the optimization problem, we apply the Q-learning algorithm [18] which uses a Q-table to represent all possible state-action pairs within the environment as shown in Fig. 1. The Q-learning agent can learn from its own decisions at each iteration, and the algorithm will converge to the optimal policy π^* after a certain number of iterations [18]. The expected return of state s under policy π is denoted as $\vartheta_\pi(s) : \mathcal{S}_i \longrightarrow \mathbb{R}$. It is expressed as follows:

$$\vartheta_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{2{f_{cap}}_i^2} \gamma \mathcal{R}_i(s_t,a_t)|s_t = s\right] = \mathbb{E}_\pi[\mathcal{R}_i(s_t,a_t) + \gamma\vartheta_\pi(s_{t+1})|s_t = s], \forall s \in \mathcal{S}_i, \tag{5}$$

where $\gamma \in [0, 1)$ is a discount factor that indicates the importance of the long-term reward [18]. The optimal policy π^* in state s represents an action a that yields the maximum value of the expected return $\vartheta_*(s)$:

$$\vartheta_*(s) = \max_a \{\mathbb{E}_\pi[\mathcal{R}_i(s_t,a_t) + \gamma\vartheta_\pi(s_{t+1})|s_t = s]\}, \forall s \in \mathcal{S}_i. \tag{6}$$

Thus, for all state-action (s,a) pairs, the optimal Q-functions are

$$Q_*(s,a) \triangleq \mathcal{R}_i(s_t,a_t) + \gamma\mathbb{E}_\pi[\vartheta_\pi(s_{t+1})], \forall s \in \mathcal{S}_i. \tag{7}$$

Hence $\vartheta_*(s)$ can be expressed as $\vartheta_*(s) = \max_a \{Q_*(s,a)\}$. By conducting different actions a to the environment the optimal Q-function value, i.e., $Q_*(s,a)$, for all state-action (s,a) pairs is figured out. In particular, the Q-function is updated at each iteration as follows:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha[\mathcal{R}_i(s_t,a_t) + \gamma \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t)], \tag{8}$$

where $s_t \in \mathcal{S}_i$, $a_t \in \mathcal{A}_i$. $Q_t(s_t,a_t)$ is the Q-value for a state-action pair (s_t,a_t), $\mathcal{R}_i(s_t,a_t)$ is the immediate reward for the SDN switch i at an iteration t, $\gamma \in [0, 1]$ is the discount factor and $\alpha \in [0,1]$ is the learning rate. Moreover, to mitigate the exploration and exploitation dilemma that has direct impact on the convergence rate of any learning algorithms, the ϵ-greedy algorithm [18] is applied. Instead of always taking the best action according to the network state, the Q-learning agent will take some random actions, where the probability of a random decision is determined by the value of epsilon, ϵ. In its learning phase, the Q-learning agent first of all arbitrarily initializes the Q-table for all state-action pairs and afterwards updates it by using Equation 8. Accordingly, the agent acquires a trained or converged Q-table.

In summary, the Q-learning agent generates the optimal policy $\pi^*(s)$ for a state s representing an action a that needs to be taken to maximize the value of the $Q_*(s,a)$ function, i.e., $\pi^*(s) = \arg\max_a Q_*(s,a)$. Algorithm 1 provides implementation details of the Q-learning algorithm.

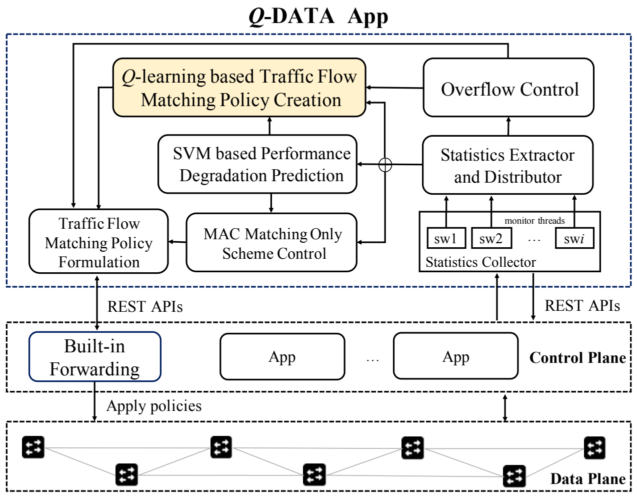

Q-DATA App
Q-learning based Traffic Flow Matching Policy Creation
Overflow Control
SVM based Performance Degradation Prediction
Statistics Extractor and Distributor
Traffic Flow Matching Policy Formulation
MAC Matching Only Scheme Control
monitor threads
sw1
sw2
...
sw*i*
Statistics Collector
REST APIs
REST APIs
Built-in Forwarding
App
...
App
Control Plane
Apply policies
Data Plane

Fig. 2. Q-DATA framework architecture

Algorithm 1 Optimal traffic flow matching policy creation with Q-learning algorithm

```
1: Inputs: $\mathcal{F}$; for a state-action pair $(s,a)$ $\forall s \in \mathcal{S}_i$, $a \in \mathcal{A}_i$,
   initialize a $Q$-table entry arbitrarily; initialize values of $\alpha$,
   $\gamma$, and $\epsilon$, respectively.
2: loop
3:     Current state $s_t$.
4:     Execute action $a_t$ according to an exploratory policy ($\epsilon$).
5:     Obtain a new state $s_{t+1}$ and an immediate reward $\mathcal{R}_i$.
6:     Update the $Q$-table entry for $Q(s_t,a_t)$ using Equation 8.
7:     Update $s_t \longleftarrow s_{t+1}$.
8: end loop
9: Outputs $\pi^*(s) = \arg\max_a Q_*(s,a)$.
```

IV. Q-DATA FRAMEWORK

In this section, the design and operation of the Q-DATA framework for enhanced traffic flow monitoring and proactively prevention of forwarding performance degradation in SDN based networks is outlined.

A. *Q-DATA Framework Architecture*

Fig. 2 shows the Q-DATA framework architecture consisting of the Built-in Forwarding application located in the control plane and the REST Q-DATA application residing in the SDN application plane.

1) Built-in forwarding application: Most of the well-known SDN controllers [4], [5] provide basic forwarding functionality by running a built-in forwarding application to create flow rules which are then downloaded to the SDN switches. We propose to add a REST API interface to the built-in forwarding application to securely communicate with the Q-DATA

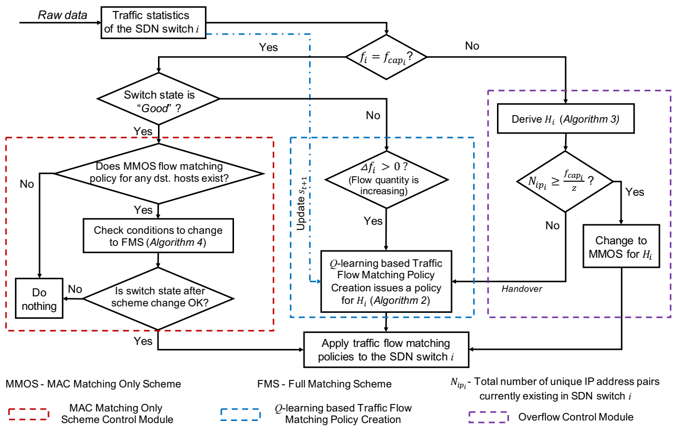

Raw data
Traffic statistics of the SDN switch i
$f_i = f_{cap_i}$?
Yes
No
Switch state is "Good" ?
No
Yes
Does MMOS flow matching policy for any dst. hosts exist?
No
Yes
Check conditions to change to FMS (Algorithm 4)
Do nothing
No
Is switch state after scheme change OK?
Yes
Update s_{t+1}
$\Delta f_i > 0$? (Flow quantity is increasing)
Yes
Q-learning based Traffic Flow Matching Policy Creation issues a policy for H_i (Algorithm 2)
Derive H_i (Algorithm 3)
$N_{ip_i} \geq \frac{f_{cap_i}}{z}$?
Yes
No
Change to MMOS for H_i
Handover
Apply traffic flow matching policies to the SDN switch i
MMOS - MAC Matching Only Scheme
FMS - Full Matching Scheme
N_{ip_i} - Total number of unique IP address pairs currently existing in SDN switch i
MAC Matching Only Scheme Control Module
Q-learning based Traffic Flow Matching Policy Creation
Overflow Control Module

Fig. 3. Operational workflow of the Q-DATA framework

application. Initially, the Q-DATA App instructs the built-in forwarding application to apply the Full Matching Scheme (FMS) strategy.

2) Q-DATA App: In Q-DATA a Statistics Collector periodically gets raw information about all traffic flows traversing the SDN switches from the SDN controller via the REST APIs [4], [5]. The collected statistical data of the SDN switch i is forwarded to a Statistics Extractor and Distributor for extracting and distributing flow statistics information to other modules, i.e., the SVM based Performance Degradation Prediction module, the MAC Matching Only Scheme Control module, the Overflow Control module and the Q-learning based Traffic Flow Matching Policy Creation module. The SVM based Performance Degradation Prediction module is designed to anticipate the performance degradation of the SDN switch i well before it occurs [8] and to provide the prediction result to the Q-learning based Traffic Flow Matching Policy Creation module and the MAC Matching Only Scheme Control module. The Overflow Control module acts as an immediate reaction mechanism against a flow-table overflow situation, e.g., in case the network is under a Denial-of-Service attack. The MAC Matching Only Scheme Control module monitors and checks conditions for a traffic flow matching scheme change to FMS in the SDN switch i. The Q-learning based Traffic Flow Matching Policy Creation module relies as discussed above on a converged Q-table to choose the most appropriate traffic flow matching scheme for a given state of the SDN switch i. Finally, the Traffic Flow Matching Policy Formulation module formulates policies received from the Overflow Control, the MAC Matching Only Scheme Control and the Q-learning based Traffic Flow Matching Policy Creation modules and sends them to the Built-in Forwarding application for implementation in the SDN switch i.

B. Operational Workflow

Initially, the Statistics Collector sends a request to the SDN controller to ask for network topology information. Then, it launches a monitor thread *swi* for each SDN switch i - see Fig. 2. In regular time intervals (observation period), the monitor thread *swi* gathers raw traffic flow statistics from the SDN switch i and passes them to the Statistics Extractor and Distributor where the tuple (f_i,Δf_i)—the total number of current flows and the flow number changes—is determined. Afterwards, this data is forwarded to the Overflow Control module, the SVM based Performance Degradation Prediction module and the MAC Matching Only Scheme Control module.

Fig. 3 shows the detailed operational workflow of the Q-DATA framework. Firstly the Statistics Extractor and Distributor module compares f_i to f_{cap_i} and if the current total number of flow entries in a switch i reaches its upper limit, then it is denoted as an overflow situation. In this case, the Overflow Control module has to find an appropriate traffic flow matching change policy for some destination hosts H_i (derived from Algorithm 3) which have most flow entries in switch i, so that the overflow situation is mitigated. We suppose that a source-destination node pair (having a unique pair of IP addresses), that transfers traffic through switch i, puts z flow entries ($z \geq 1.0$) on average in the switch i (e.g., either request or response flows). N_{ip_i} denotes the total number of unique IP address pairs in the flow-table of the switch i. In case of a non-saturation attack the number of hosts (represented by their IP addresses[6]) sending traffic through the SDN switch i is usually much less than the maximum number flow entries f_{cap_i}. Thus, if $N_{ip_i} \geq \frac{f_{cap_i}}{z}$, there are some destination hosts H_i serving a lot of incoming requests from other hosts or being under saturation attacks (e.g., Denial-of-Service attacks). Hence, it is reasonable to match incoming traffic flows related to these destination hosts using only MAC address information. This avoids a sudden overflow situation of switch i. Otherwise, the Overflow Control module handovers to the Q-learning based Traffic Flow Matching Policy Creation module to issue an optimal traffic flow matching policy for the H_i destination hosts via Algorithm 2.

In case the current total number of flow entries f_i is less than the switch's upper limit f_{cap_i}, the SVM based Performance Degradation Prediction module checks for a potential performance degradation of the SDN switch i based on the tuple (f_i,Δf_i), and forwards its prediction result to the MAC Matching Only Scheme Control module and the Q-learning based Traffic Flow Matching Policy Creation module. If the switch state is predicted as *Good*, the MAC Matching Only Scheme Control module checks whether there exists a MMOS flow matching policy for any of the destination hosts. If a MMOS flow matching policy is found, then Algorithm 4 is applied to check the conditions for a change to the FMS strategy. In case a possible performance degradation is detected for switch i, and if the total number of flow entries is increasing (Δf_i>0), then the Q-learning based Traffic Flow Matching Policy Creation module executes Algorithm 2 to apply the most appropriate traffic flow matching policy for destination hosts (derived from Algorithm 3) in the switch i.

[6]Initially the Built-in Forwarding application applies the FMS scheme, hence IP address information is available before an overflow problem appears in the switch i.

Algorithm 2 Q-learning based traffic flow matching policy creation for the SDN switch i

```
Input: A tuple $(f_i, \Delta f_i)$ at iteration $t$; $\mathcal{R}_i$; $H_i$.
begin
    Utilize a converged $Q$-table from Algorithm 1.
    $s_t \leftarrow (f_i, \Delta f_i)$.
    Drive optimal action $a_t = \pi^*(s_t)$ from the $Q$-table.
    Apply $a_t$ for $H_i$ derived from Algorithm 3.
    Update $s_t \leftarrow s_{t+1}$ {see Fig. 3}.
    Get reward $\mathcal{R}_i$ using Equation 3.
    Update $Q$-table using Equation 8.
end
```

V. Experiments

A. Example SDN Network Scenario

In order to evaluate the performance of the Q-DATA framework, we leverage the MaxiNet framework [19] to emulate a simple SDN based network consisting of 3 Web servers (S1-S3) (using Apache Web server images) and 5 hosts (H1-H5) which are all connected to a single SDN switch (implemented as OpenvSwitch). The emulated SDN network runs within one Linux machine and is controlled by a remote ONOS SDN controller running on another physical machine. For ease of deployment, we place both the Q-DATA App and the ONOS SDN controller on the same Linux machine.

B. Training Q-learning and SVM Algorithms

Initially, for training the Q-learning agent we use the Hping3 tool [20] installed in hosts (H1-H5) to randomly generate traffic between hosts and Web servers. The Q-learning agent depends on the collected data for making decisions about changing the traffic flow matching scheme, and for updating its Q-table accordingly. In particular, we set the ϵ value to 0.8 in order to have 80% of random actions in a set of 9 match field combinations, and the state observation time is set to 10.0 seconds. For training the SVM algorithm, we apply the same traffic generation strategy as for the Q-learning agent training phase and initially apply the FMS scheme. Afterwards, we monitor any errors or exceptions indicating that the switch cannot handle new flow requests, and set $sign$ = -1 as a label for the associated tuple $(f_i,\Delta f_i)$. Otherwise, we set $sign$ = +1. These labelled samples are then used for training the SVM algorithm.

We observe that the switch starts getting overflowed or cannot handle new flow rules if the current total number of flows is around 3000 (f_{cap_i}) [6], [8]. Setting the *idle_timeout* value (after which the flow entries are removed) to 10 seconds, the safety threshold for the packet rate the switch can handle is 300 packets per second assuming that each packet belongs to a different traffic flow rule (worst case assumption). Therefore, for traffic generation, we apply three levels: low load (R1=100), medium load (R2=200) and high load (R3=300).

C. Experiment Setup

We conduct several experiments with different flow matching strategies: MMOS only, FMS only, the novel Q-DATA framework (with ϵ = 0.0, ϵ = 0.2, ϵ = 0.8) and the DATA scheme [8]. The built-in forwarding application of the ONOS SDN controller applies *Reactive Forwarding*.

In order to show the performance enhancement in traffic flow monitoring in SDN based networks with the Q-DATA framework, we implement a SOM-based IDS application (Self Organizing Map algorithm [21]) to detect abnormal traffic on top of the ONOS controller. We consider some common attacks, which can make the SDN switch become overflowed, comprising TCP SYN flood [22], Port scanning [22], Low and Slow Denial-of-Service [6]. The attack traffic is stemmed from hosts and it is directed to Web servers in our setup.

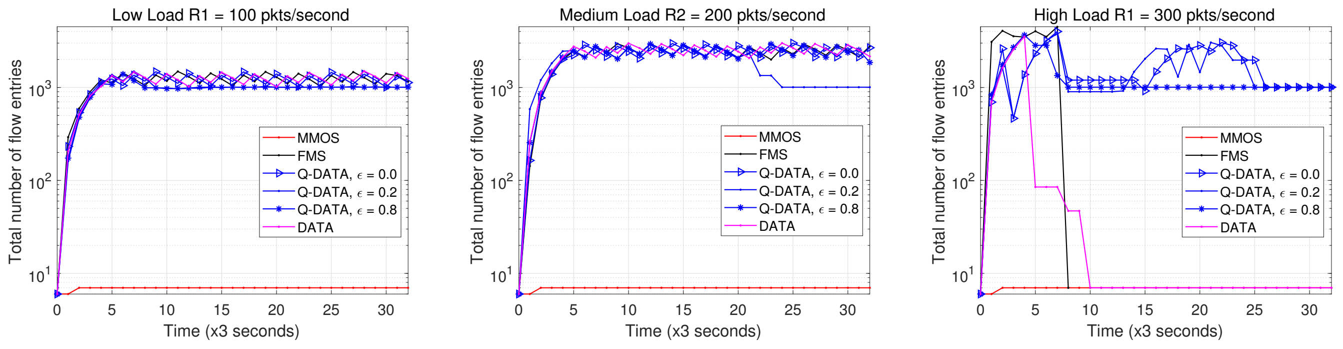
For the performance analysis, traffic from the 5 hosts towards the 3 servers is generated randomly with three different load levels (R1, R2, R3). During the experiments we trace the total number of flow entries in the SDN switch, the average number of *packet_in* messages per second to the ONOS controller, errors and exceptions in the ONOS controller, the frequency of traffic flow matching policy changes, the CPU utilization of the controller machine and the attack detection performance of the SOM-based IDS.

VI. Results

A. Network related Performance Results

1) Total number of traffic flow entries in the SDN switch: As can be seen in Fig. 4, the MMOS scheme accounts for a very low amount of traffic flow entries in all scenarios. For the low and medium load cases, FMS, Q-DATA (ϵ = 0.0, ϵ = 0.2 and ϵ = 0.8) and DATA are supposed to have the same amount of traffic flow entries in the switch since the total number of flow entries is always below the critical level (f_{cap_i}). Note, that there are some minor variations for Q-DATA with ϵ = 0.2 and ϵ = 0.8 because the Q-learning based Traffic Flow Matching Policy Creation module is allowed to take random actions that leads to a *Good* state of the SDN switch with a high immediate reward value (average number of match fields of a flow entry) and to no further flow entry changes in the remaining time.

In the high load scenario, the FMS scheme leads to errors and exceptions after a short period of time causing a massive reduction in the number of flow entries because the SDN switch and the ONOS controller suspend their operation. In case of DATA, after reaching the switch's flow-table entry upper limit (f_{cap_i}), the flow matching scheme is changed to MMOS for some destination hosts leading to a very small amount of flow entries in the remaining time. In contrast, the Q-DATA framework maintains a significant number of flow rules by applying appropriate traffic flow matching policies, e.g., a layer 2 & layer 3 matching scheme which provides a higher traffic flow matching granularity and avoids the performance degradation of the switch. Besides, the Q-learning based Traffic Flow Matching Policy Creation module



(a) (b) (c)

Fig. 4. Total number of flow entries in the SDN switch for three different traffic loads: (a) Low Load $R1$=100, (b) Medium Load $R2$=200 and (c) High Load $R3$=300

depends on future states, i.e., s_{t+1}, and tries to maximize the traffic flow matching granularity by changing to other schemes which provide more traffic flow information details. Therefore we observe some changes in the number of flow entries during our experiments.

2) Average packet_in message rate to the ONOS controller: Fig. 5 (a) illustrates the average number of *packet_in* messages per second arriving at the Built-in Forwarding application. Contrary to the FMS and DATA schemes, for all traffic loads, the Q-DATA framework with the optimal traffic flow matching policy ($\epsilon = 0.0$) allows the ONOS controller to process an acceptable *packet_in* rate. This significantly reduces the workload of the Built-in Forwarding application because of less new flow installation queries. The results for the Q-DATA scheme with $\epsilon = 0.2$ and $\epsilon = 0.8$ are expected to be better for a longer experiment duration.

3) Errors and exceptions: Another key criterion for the performance evaluation of the Q-DATA solution is the time until an error or exception (observed by the ONOS terminal) occurs due to a degraded SDN switch. Our measurements show that the FMS scheme causes *disconnected* channels errors and *FlowRuleManager* exceptions in the ONOS controller after 7 to 10 seconds since the high traffic load is generated. For the other traffic load cases, no errors and exceptions are observed.

4) Frequency of changing flow matching policy: We record the total number of traffic flow matching scheme changes of the proposed Q-DATA framework and the DATA scheme. As shown in Fig. 5 (b), the DATA scheme tries to keep the SDN switch in a *Good* state as long as possible—therefore no changes in the traffic flow matching scheme occur for low and medium load scenarios, but some changes happen in the high load case (i.e., a change from FMS to MMOS). Contrary, Q-DATA performs some changes depending on newly incoming traffic flows in the switch. In particular, in the high load case, Q-DATA with $\epsilon = 0.0$ performs several flow matching scheme changes, e.g., between layer 2 & layer 3 matching and FMS, to provide more traffic flow information details while guaranteeing that the SDN switch forwarding performance does not degrade.

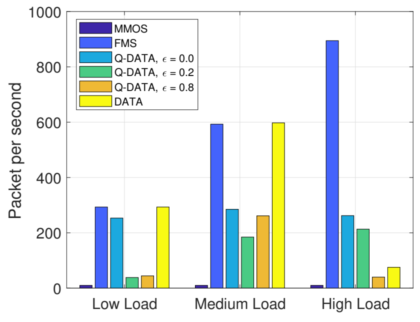
TABLE I
ANOMALY DETECTION PERFORMANCE OF DIFFERENT TRAFFIC FLOW MATCHING SOLUTIONS AND TRAFFIC LOADS

TCP SYN flood attack detection performance (%)						
	MMOS	FMS	Q-DATA $\epsilon = 0.0$	Q-DATA $\epsilon = 0.2$	Q-DATA $\epsilon = 0.8$	DATA
$R1$	0.0	98.02	97.05	96.04	96.53	97.52
$R2$	0.0	97.64	96.53	97.37	96.06	98.05
$R3$	0.0	0.0	85.20	81.20	82.00	0.0
Port scanning attack detection performance (%)						
	MMOS	FMS	Q-DATA $\epsilon = 0.0$	Q-DATA $\epsilon = 0.2$	Q-DATA $\epsilon = 0.8$	DATA
$R1$	0.0	96.53	97.45	96.00	96.33	97.30
$R2$	0.0	98.22	96.43	97.01	96.62	97.22
$R3$	0.0	0.0	84.34	84.10	82.56	0.0
Low and Slow DoS attack detection performance (%)						
	MMOS	FMS	Q-DATA $\epsilon = 0.0$	Q-DATA $\epsilon = 0.2$	Q-DATA $\epsilon = 0.8$	DATA
$R1$	0.0	96.32	96.25	95.70	97.12	95.78
$R2$	0.0	96.34	97.36	96.54	96.21	95.92
$R3$	0.0	0.0	88.24	85.32	85.45	0.0

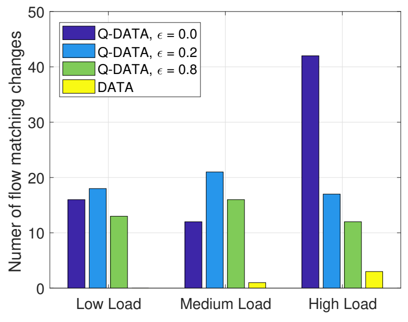
5) Computational overhead: Fig. 5 (c) shows measurements of the CPU utilization of the controller machine. It can be seen that the three Q-DATA scheme variants ($\epsilon = 0.0$, $\epsilon = 0.2$, $\epsilon = 0.8$) consume more CPU resources for all traffic loads. This is due to the fact that the Q-DATA App actively monitors and analyzes the network traffic, especially in the case of high traffic load. It tries to maximize the traffic flow matching granularity and to avoid any performance degradation of the switch. Nonetheless, this computational overhead is acceptable considering the benefits of the Q-DATA Scheme.

B. Anomaly Detection Performance Results

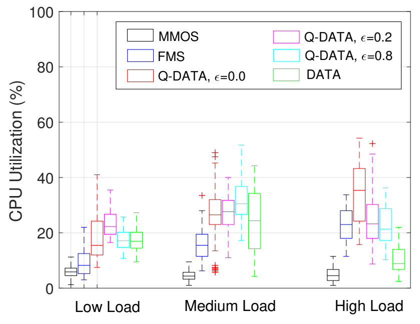
In order to show the enhancement of the traffic flow monitoring capability provided by the Q-DATA framework, we evaluate the anomaly detection performance of the SOM-based IDS application for three attack types, i.e., TCP SYN



(a)



(b)



(c)

Fig. 5. (a) Average *packet_in* rate (pkts/s) to the ONOS controller for different traffic flow matching schemes and traffic loads, (b) Total number of traffic flow matching scheme changes (during 500 seconds experiment duration) for different loads, (c) Average CPU utilization of the controller machine (during 500 seconds experiment duration)

flood[7], Port scanning[8] and Low and Slow Denial-of-Service[9]. For the evaluation we apply the following fitness function:

$$F_{anomaly} = W_{D_r} D_r + W_{A_c} A_c + W_{F_a} e^{-F_a}, \tag{9}$$

where D_r represents the Detection rate, A_c Accuracy, F_a False alarm rate and W_{D_r}, W_{A_c} and W_{F_a} are weight values which are equally set to 1/3 in our evaluation.

As shown in Table I, no alert is raised in case of the MMOS scheme for all attack types and traffic loads because the traffic towards the Web servers is grouped into flow entries in the switch that makes it for the IDS impossible to detect any attacks. In the low and medium load scenarios, for the FMS, *Q*-DATA ($\epsilon = 0.0$, $\epsilon = 0.2$, $\epsilon = 0.8$) and DATA schemes, three attacks are detected by the IDS with similar levels of attack detection performance.

In the high traffic load case, for FMS, the operation of the SDN controller and the switch are suspended. This makes the IDS application unable to gather traffic information from the SDN controller and to detect the attacks. For the DATA scheme the SDN switch stays operational, however traffic flows targeting to the servers are aggregated to some MMOS flows in the SDN switch. Hence there is no chance[10] to recognize malicious traffic flows towards the Web servers for all three attack types. Contrary, *Q*-DATA, by frequently changing between different flow matching schemes, provides more detailed traffic flow information and enables the IDS application to recognize the attack presence. However, because of the variation of statistics information caused by the traffic flow matching scheme change in the switch, the attack detection performance in case of high traffic load is lower than for low and medium traffic loads.

[7]Attackers try to send as fast as possible TCP segments with different spoofed source IP addresses and TCP ports to the Web servers leading to a large number of new flow entries in the SDN switch in a short time period.

[8]Attackers try to send as many as possible TCP segments with different destination ports to the Web servers and wait for response packets.

[9]Attackers periodically send requests as slow as possible with little resources and try to keep all installed flow entries in the SDN switch alive as long as possible, which renders the victim inaccessible.

[10]Nevertheless, for a larger scale network that comprises several switches (like the enterprise network in our previous study [8]), the SOM-based IDS is expected to achieve a good attack detection performance as some switches carry attack traffic flows and still stay operational (i.e., are in a *Good* forwarding performance state).

VII. Conclusion

In this paper, we present a traffic flow matching control framework based on reinforcement learning called *Q*-DATA which improves traffic flow monitoring in SDN based networks and proactively prevents performance degradation of SDN switches. We conduct a comprehensive performance analysis of the *Q*-DATA framework. Our results show that—compared to the default behavior of common SDN controllers and to our previous DATA scheme—the new *Q*-DATA framework by applying always the optimal traffic flow matching policy yields remarkable performance benefits. In our future work, we intend to focus on an optimized integration of traffic flow matching control and traffic anomaly detection.

VIII. Acknowledgments

This work has been performed in the framework of the Celtic-Plus project SENDATE Secure-DCI, funded by the German BMBF (ID 16KIS0481).

References

[1] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys Tutorials*, vol. 16, pp. 1617–1634, Third 2014.

[2] Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho, and C. Yang, "Traffic engineering in software-defined networking: Measurement and management," *IEEE Access*, vol. 4, pp. 3246–3256, 2016.

[3] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "Research challenges for traffic engineering in software defined networks," *IEEE Network*, vol. 30, pp. 52–58, May 2016.

[4] ONOS, "Description of the onos controller." www.onosproject.org, May 2019.

[5] OpenDaylight, "Description of the opendaylight controller." www.opendaylight.org, May 2019.

[6] T. A. Pascoal, Y. G. Dantas, I. E. Fonseca, and V. Nigam, "Slow tcam exhaustion ddos attack," in *IFIP International Conference on ICT Systems Security and Privacy Protection*, pp. 17–31, Springer, 2017.

[7] H. Wang, L. Xu, and G. Gu, "Floodguard: A dos attack prevention extension in software-defined networks," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 239–250, June 2015.

[8] T. V. Phan, M. Hajizadeh, N. Tuan Khai, and T. Bauschert, "Destination-aware adaptive traffic flow rule aggregation in software-defined networks," in *2019 International Conference on Networked Systems (NetSys)*, March 2019.

[9] 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. Softwarization and Caching in NGN.

[10] X. Jia, Q. Li, Y. Jiang, Z. Guo, and J. Sun, "A low overhead flow-holding algorithm in software-defined networks," *Computer Networks*, vol. 124, pp. 170 – 180, 2017.

[11] B. Leng, L. Huang, X. Wang, H. Xu, and Y. Zhang, "A mechanism for reducing flow tables in software defined network," in *2015 IEEE International Conference on Communications (ICC)*, pp. 5302–5307, June 2015.

[12] 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, pp. 1007–1020, June 2014.

[13] M. Rifai, N. Huin, C. Caillouet, F. Giroire, J. Moulierac, D. L. Pacheco, and G. Urvoy-Keller, "Minnie: An sdn world with few compressed forwarding rules," *Computer Networks*, vol. 121, pp. 185 – 207, 2017.

[14] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "Past: Scalable ethernet for data centers," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, (New York, NY, USA), pp. 49–60, ACM, 2012.

[15] S. Luo, H. Yu, and L. M. Li, "Fast incremental flow table aggregation in sdn," in *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–8, Aug 2014.

[16] A. Mimidis, C. Caba, and J. Soler, "Dynamic aggregation of traffic flows in sdn: Applied to backhaul networks," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pp. 136–140, June 2016.

[17] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods*. New York, NY, USA: Cambridge University Press, 2000.

[18] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to reinforcement learning*, vol. 135. MIT press Cambridge, 1998.

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

[20] hping3, "Description of the hping3 tool." www.hping.org, May 2019.

[21] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *IEEE Local Computer Network Conference*, pp. 408–415, Oct 2010.

[22] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Communications Surveys Tutorials*, vol. 18, pp. 602–622, Firstquarter 2016.

APPENDIX A
ALGORITHMS FROM OUR PREVIOUS WORK [8]

Algorithm 3 Identification of the destination hosts whose flows are most critical to the performance of SDN switch i

```
Input: $S_i = \{(host_1, flow_1), (host_2, flow_2), ..., (host_\chi, flow_\chi)\}$: set of destination hosts and respective number of flow entries associated with these hosts in switch $i$; $f_i = \sum_{c=1}^{\chi} flow_c$: total number of current flow entries in switch $i$; $index = 1$: first index. Output: $H_i = \{\}$: set of destination hosts.
begin
Sort $S_i$ in descending order of the current flow $flow_c$ (from highest to lowest numbers).
loop
    $H_i.append(S_i[index])$
    $f_{remaining} = 1 + \sum_{c=index+1}^{\chi} flow_c$ {One MMOS flow entry is installed in switch $i$}.
    $\Delta f_i = f_i - f_{remaining}$ {Delete $\Delta f$ flow entries in switch $i$}.
    $sign = SVM(f_{remaining}, \Delta f_i)$
    if $sign = +1$ then
        break {Switch $i$ can handle $f_{remaining}$ entries}.
    else
        $index = index + 1$ {Switch $i$ cannot handle $f_{remaining}$ entries}.
    end if
end loop
return $H_i$.
```

Algorithm 4 Identification of the MMOS flows/destination hosts related to SDN switch i for which changing back to FMS is feasible

```
Input: $S_i = \{(host_1, R_{pkt_1}), (host_2, R_{pkt_2}), ..., (host_\chi, R_{pkt_\chi})\}$: set of destination hosts and respective packet rate of MMOS flows associated with these hosts in switch $i$; ($f_i$, $f_{cap_i}$): total number of current flow entries and maximum number of flow entries in switch $i$; $f_{extra}$: number of flow entries that might be added in switch $i$. Output: $H_i = \{\}$: set of destination hosts.
begin
for $index = 1$; $index \leq z$; $index$++ do
    $f_{extra} = idle\_timeout * R_{pkt_{index}}$ {Worst case assumption: each packet is associated with a new entry in switch $i$}.
    if $(f_{extra} + f_i) < f_{cap_i}$ then
        $H_i$.append[$h_{index}$].
    else
        continue
    end if
end for
return $H_i$.
```