Efficient SDN-Based Traffic Monitoring in IoT Networks with Double Deep Q-Network

Tri Gia Nguyen[1(✉)], Trung V. Phan[2], Dinh Thai Hoang[3], Tu N. Nguyen[4], and Chakchai So-In[5]

[1] Faculty of Information Technology, Duy Tan University, Danang 50206, Vietnam
nguyengiatri@duytan.edu.vn
[2] Chair of Communication Networks, Technische Universität Chemnitz, 09126 Chemnitz, Germany
[3] School of Electrical and Data Engineering, University of Technology Sydney, Sydney, NSW 2007, Australia
[4] Department of Computer Science, Purdue University Fort Wayne, Fort Wayne, IN 46805, USA
[5] Department of Computer Science, Faculty of Science, Khon Kaen University, Khon Kaen 40002, Thailand

Abstract. In an Internet of Things (IoT) environment, network traffic monitoring tasks are intractable to achieve due to various IoT traffic types. Recently, the development of Software-Defined Networking (SDN) enables outstanding flexibility and scalability abilities in network control and management, thereby providing a potential approach to mitigate challenges in monitoring the IoT traffic. In this paper, we propose an IoT traffic monitoring approach that implements deep reinforcement learning technique to maximize the fine-grained monitoring capability, i.e., level of traffic statistics details, for several IoT traffic groups. Specifically, we first study a flow-rule matching control system constrained by different expected levels of statistics details and by the flow-table limit of the SDN-based gateway device. We then formulate our control optimization problem by employing the Markov decision process (MDP). Afterwards, we develop Double Deep Q-Network (DDQN) algorithm to quickly obtain the optimal flow-rule matching control policy. Through the extensive experiments, the obtained results verify that the proposed approach yields outstanding improvements in terms of the ability to simultaneously provide different required degrees of statistics details while protecting the gateway devices from being overflowed in comparisons with those of the conventional Q-learning method and the typical SDN flow rule setting.

This work was supported in part by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under Grant 102.01-2019.322, and the Thailand Science Research and Innovation (TSRI) and National Research Council of Thailand (NRCT) via the International Research Network Program (IRN61W0006).

S. Chellappan et al. (Eds.): CSoNet 2020, LNCS 12575, pp. 26–38, 2020.
https://doi.org/10.1007/978-3-030-66046-8_3

Keywords: Traffic monitoring · Internet of Things · Software-defined networking · Double Deep Q-Network · Deep reinforcement learning

1 Introduction

Internet of Things (IoT) culminates the interconnection of ubiquitous end-devices with unique digital identity, e.g., smart devices and industrial systems [11]. Nevertheless, it exhibits several challenges in the network control and management tasks, particularly for network traffic monitoring that can provide beneficial information to other crucial applications, e.g., traffic engineering and anomaly detection. Consequently, it is essential to acquire a solution that can dynamically afford a fine-grained traffic monitoring capability to aid applications with high demands about detailed traffic information in the IoT networks. This primary requirement is challenging to be realized due to the diversity of IoT traffic types in a ubiquitous context [11].

Recently, Software-Defined Networking (SDN) technology [8] has been drawing a notable breakthrough in Telco industries due to significant improvements concerning dynamics and flexibility in traffic control and management tasks. In particular, SDN defines a new design and management strategy for networking. The key innovation of this design is the separation of the control and data planes. The SDN controller (e.g., ONOS [9]) makes the control decision and the SDN data plane manages the data forwarding. The communication between two planes is performed through southbound APIs, e.g., OpenFlow [3]. Henceforward, with SDN's unique features, they enable the IoT networks with the ability to mitigate network control and management challenges [2]. Ultimately, it has become a new IoT research area that has been recently attracting the concentration of researchers [2,7].

1.1 Related Work

In order to avoid overflow problems in flow-tables of the SDN forwarding devices, Avant-Guard [12] framework is proposed, in which the control plane distributes intelligence, i.e., a connection migration module, into switches to identify hosts that unlikely finish TCP connections. Thereby, diminishing abnormal flow rules that may flood flow-tables in the case of saturation, e.g., TCP SYN flood attack. A reinforcement learning algorithm [6] is proposed with the primary purpose of obtaining the long-term intention of diminishing the monitoring overhead and of being aware of the overflow problem in the forwarding devices. Likewise, SDN-Mon framework [10] is proposed to improve the traffic monitoring granularity that promotes many networking applications, in which the monitoring logic is separated from the forwarding logic by modifying OpenvSwitch and the Lagopus software switch. Afterwards, additional monitoring/functional tables are appended to the end of the forwarding pipeline. Nevertheless, one of the main weaknesses of these proposals is that a comprehensive customisation effort is required in the data plane devices, which violates the origin of SDN design.

Distinctly, in this study, we investigate a win-win equilibrium[1] between different IoT traffic groups concerning their granularity degree demands as one of the crucial evaluation criteria. Because the monitoring operation should fairly observe the sufficient statistics details of all IoT traffic groups. Therefore, we are strongly motivated to develop an innovative approach in the SDN-based IoT networks to efficiently provide a traffic monitoring capability while protecting the SDN switches from being overflowed.

1.2 Our Proposal

In this paper, we propose a traffic monitoring approach that employs deep reinforcement learning technique to maximize the degree of traffic statistics details and to provide a win-win equilibrium of the monitoring capability for several traffic groups in the SDN-based IoT environment, as shown in Fig. 1. Specifically, we study an SDN-based flow matching control system constrained by the required granularity levels of traffic groups and the capacity of the SDN-based gateway. We then formulate the control optimization problem by applying the Markov decision process (MDP), and develop an advanced reinforcement learning technique, i.e., Double Deep Q-Network (DDQN) algorithm, to quickly obtain the optimal policy for large-scale systems. The experiment results confirm the effectiveness of our proposed approach in terms of providing a sustainable degree of IoT traffic granularity while avoiding the gateway overflow issue in comparison with other traditional methods.

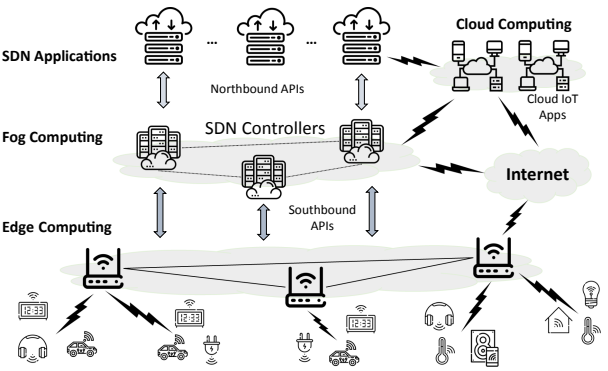


Fig. 1. SDN-based IoT networks.

[1] Different IoT traffic groups can meet their traffic granularity requirements at the same time.

2 System Model

2.1 Basic SDN Packet Forwarding Strategy

As presented in Fig. 2, the basic SDN packet forwarding in a gateway can be described as follows. Firstly, an incoming packet arrives at the gateway, i.e., step (1). Next, the searching process of a matched flow rule is performed by checking the packet header information to *match fields* in every flow rule, i.e., step (2). If the header information is suited to all *match fields* of a flow rule, this means that a matched flow is determined[2] and *Instructions* supervise the incoming packet in the matched flow rule, e.g., *forward* the packet to a particular gateway port (step (5)) or *drop* the packet. Contrarily, a *table-miss* event occurs for the entered packet, and the gateway creates a packet_in message and sends to its corresponding SDN controller for further guidance, i.e., step (3). Later, the SDN controller depends on its traffic forwarding plans and installs a new flow rule into the gateway with a selection of match fields and instructions, i.e., step (4). Then, a packet_out message is delivered out of the gateway, i.e., step (5), and the sent packet is the first packet of the new flow rule.

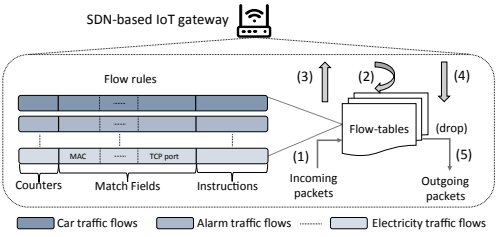


Fig. 2. Typical SDN packet forwarding logic.

To sum up, the determination of *match fields* in a flow rule represents a critical position in the SDN packet forwarding and flow matching in the gateway devices. In addition, the traffic statistics details collected by the control plane [8] are mainly defined by the number of match fields at the gateway. Accordingly, to adjust the traffic granularity degree, a flow matching control system controlling the match fields determination should be examined to obtain an efficient traffic monitoring capability in the SDN-based networks.

2.2 System Model

As aforementioned, it is essential to correctly and dynamically determine the right collection of match fields in traffic flows of various IoT devices. Therefore, we examine a flow matching control system operating as an SDN application

[2] For ease of comprehension, we study this simple matching strategy in this paper, more complex matching designs can be found in [3], e.g., multiple flow-tables searching.

that consists of a statistics collector, a control agent, a database, and a flow matching policy-maker. In which, the control agent controls the process of flow matching in flow-tables at an SDN-based IoT gateway i.

As stated in [1], from a macro scale, IoT traffic is realized as heterogeneous but group-specific from the representation of each local network. Accurately, IoT devices serving different applications can be attached in separate virtual local area networks (VLANs), which can be directed at the IoT networks' edge, i.e., the SDN-based IoT gateway. Hence, we consider $\mathcal{N}$ IoT traffic types traversing the gateway. For each traffic group, the required granularity for monitoring goals, i.e., the number of match fields in a flow rule, is denoted as Θ_n $(n \in \mathcal{N})$, where $0 < \Theta_n \leq \mathcal{M}_{max}$ and $\mathcal{M}_{max}$ is the maximum value of match fields in a flow rule. Originally, the control agent implements a policy to change the flow matching strategy of particular traffic groups into the gateway. Then, by relying on the collected statistics data, the control agent evaluates the effectiveness of the performed policy, i.e., the traffic granularity level of all traffic groups in the gateway, in comparison with the Θ_n value (*first* constraint) and the capacity of flow-tables of the gateway device (*second* constraint) indicated as $\mathcal{G}_{max}$. These processes are repeated to travel new policies and observations. Therefore, we can express the objective function of the control system as follows:

$$\max_{t} \sum_{n=1}^{\mathcal{N}} \theta_n, \quad \text{s.t.} \begin{cases} 0 < \Theta_n \leq \theta_n \leq \mathcal{M}_{max}, & \forall n \in \mathcal{N}, \\ 0 \leq \mathcal{F}_t < \mathcal{G}_{max}, \end{cases} \tag{1}$$

where θ_n is the actual granularity of the traffic group n and $\mathcal{F}_t$ is the total number of flow rules in the gateway at time step t.

3 Problem Formulation

To maximize the granularity degree of traffic while providing a win-win equilibrium for $\mathcal{N}$ traffic groups at the gateway, we adopt the Markov decision process (MDP) model [13] to present the control system operation as illustrated in Fig. 3. This framework allows the control system dynamically to perform optimal

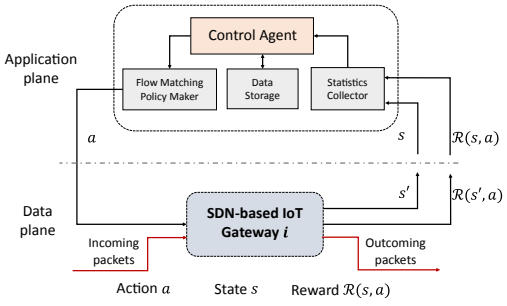


Fig. 3. Reinforcement learning based model.

actions based on its observations to maximize its average long-term reward. The MDP is defined by a tuple $< \mathscr{S}, \mathscr{A}, \mathscr{R} >$ where $\mathscr{S}$ represents the state space, $\mathscr{A}$ expresses the action space, and $\mathscr{R}$ signifies the immediate reward function.

3.1 State Space

As discussed previously, from the perspective of each local network, there are $\mathscr{N}$ traffic groups carrying traffic through the gateway [1]. Here, we aim to maximize the total traffic granularity degrees of all groups. Therefore, the state space of the gateway can be determined as follows:

$$\mathscr{S} \triangleq \{((\theta_1, f_1), ..., (\theta_n, f_n), ..., (\theta_{\mathscr{N}}, f_{\mathscr{N}}))\}, \tag{2}$$

where θ_n ($\theta_n \leq \mathscr{M}_{max}$) and f_n ($f_n \leq \mathscr{G}_{max}$) show the actual current granularity levels and the total number of flow rules of the traffic group n, respectively. The reason for choosing f_n is that the higher number of flow rules intimates the more match fields in a flow rule of the traffic group. Hence, a state of the system can be defined by $s = ((\theta_1, f_1), ..., (\theta_n, f_n), ..., (\theta_{\mathscr{N}}, f_{\mathscr{N}}))$.

3.2 Action Space

Recall the objective function in Eq. (1), one of the essentials is to always hold all θ_n values greater or equal to their corresponding Θ_n ones. Thus, the control agent should quickly construct actions whenever θ_n does not meet the requirement. However, the principal goal is to maximize the total of actual traffic granularity; hence, the control agent should implement actions even in the case when all θ_n values meet the requirement. Furthermore, the strategy is to execute actions for only the traffic group holding the lowest θ_n value at a time step. If the flow matching strategy of all traffic groups changes at the same time, this could lead to a significant variation in the gateway's flow-tables and more latency due to table-miss events and packet_in messages.

$\mathscr{C} = \{c_1, c_2, ..., c_k\}$ expresses a list of all feasible match field combinations, e.g., $c_k =$ <*matchTcpUdpPorts*, *matchIpv4Address*,...> defined in *Reactive Forwarding* application of the ONOS SDN controller [9]. Therefore, the action space for changing the flow matching tactic in the gateway is determined by

$$\mathscr{A} \triangleq \{a : a \in \mathscr{C}\}. \tag{3}$$

It is noted that if the executed action is the same in comparison with the current one of the chosen traffic group, this indicates that the control agent goes to the sleep mode and waits for the next state observation.

3.3 Immediate Reward Function

Whenever, by executing an action, the total number of current flow entries $\mathscr{F}_t$ in the gateway reaches the limit $\mathscr{G}_{max}$, which may lead to either a degradation of the gateway forwarding performance or a packet_in flooding attack to the SDN controller, the control agent should be immensely punished for this taken action. Otherwise, the more match fields in a flow rule of a group, the higher the granularity level the group presents. Moreover, in case there exists a group getting $\theta_n < \Theta_n$, a little punishment[3] is applied for the immediate reward of the control agent. Consequently, we formulate the immediate reward function of the control agent as the total achieved granularity values abstracting the total punishments for groups not satisfying their requirements, which is described as follows:

$$\mathscr{R}(s,a) = \begin{cases} \sum_{n=1}^{\mathscr{N}} \theta_n, \text{ if } \theta_n \geq \Theta_n, \ \forall n \in \mathscr{N} \text{ and } \mathscr{F}_t < \mathscr{G}_{max}, \\ \sum_{n=1}^{\mathscr{N}} \theta_n - \sum_{n=1}^{\mathscr{N}} \Delta_{\mathscr{R}_n}, \text{ if } \theta_n < \Theta_n \text{ and } \mathscr{F}_t < \mathscr{G}_{max}, \\ -\mathscr{M}_{max}, \text{ if } \mathscr{F}_t = \mathscr{G}_{max}, \end{cases} \tag{4}$$

where $\mathscr{F}_t$ is the current total number of flow entries in the gateway, and $\mathscr{M}_{max}$ exhibits the maximum number of match fields in a flow rule.

3.4 Optimization Formulation

We formulate an optimization problem to obtain the optimal policy, denoted as $\pi^*(s)$, that maximizes the control system's average long-term reward, i.e., the traffic granularity level of all groups at the gateway. Hence, the optimization problem is described as follows:

$$\begin{aligned} \max_{\pi} \quad & \mathfrak{R}(\pi) = \sum_{t=1}^{\infty} \mathbb{E}(\mathscr{R}(s_t, \pi(s_t))) \\ \text{s.t.} \quad & \begin{cases} \mathscr{R} \in \mathbb{R}, \quad s_t \in \mathscr{S}, \quad \pi(s_t) \in \mathscr{A}, \\ 0 < \Theta_n \leq \theta_n \leq \mathscr{M}_{max}, \quad \forall n \in \mathscr{N}, \\ \mathscr{F}_t < \mathscr{G}_{max}, \end{cases} \end{aligned} \tag{5}$$

where $\mathfrak{R}(\pi)$ is the average reward of the control agent under the policy π and $\mathscr{R}(s_t, \pi(s_t))$ is the immediate reward function under the policy π at time step t.

[3] Difference between the gained θ_n and required Θ_n values is calculated by $\Delta_{\mathscr{R}_n} = \Theta_n - \theta_n$, which is the punishment of the group n.

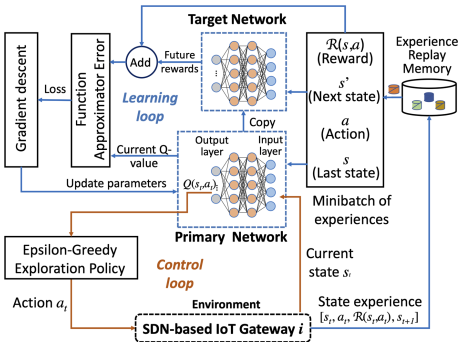


Fig. 4. Double Deep Q-Network based model.

4 Efficient SDN-Based IoT Traffic Monitoring with Double Deep Q-Network

The considering MDP model is with a large number of states due to the combinations between $\mathcal{N}$ traffic groups, as represented in Eq. (2). In this section, we develop the DDQN algorithm [14] to quickly obtain the optimal policy for the control agent. Specifically, the DDQN algorithm is developed to improve the performance of the Deep Q-Network (DQN) algorithm [5] that employs a deep neural network as a nonlinear function approximator to find the approximated values of $Q^*(b, a)$. The original intention of the DDQN is to determine an action according to an online or primary neural network Q_{net} (primary Q-network), and it uses a target neural network $\hat{Q}_{net}$ (target Q-network) to estimate the target Q-value of the executed action, as shown in Fig. 4. In our approach, an experience replay mechanism and a target Q-network are implemented as follows:

Experience Replay Mechanism: We deploy a replay memory pool, $\mathscr{E}$, to store the control agent's experience at each time step, $e_t = [s_t, a_t, \mathscr{R}(s_t, a_t), s_{t+1}]$, over many episodes, and $\mathscr{E} = \{e_1, ..., e_t\}$. The collected samples of experiences from $\mathscr{E}$ are picked up at random to execute the neural network updates. This approach allows achieving a high data usage efficiency since each experience is trained many times by the neural networks. Additionally, the randomizing technique reduces the correlations between the observed samples and therefore lessening the variance of the neural network updates.

Target Q-Network: During the learning period, the Q-values will be corrected resulting in changes of the value calculations if the shifting set of values is applied for refurbishing the primary Q-network Q_{net}, and this destabilizes the learning

algorithm. Hence, to improve the stability of the algorithm with neural networks, we employ a separate network $\hat{Q}_{net}$, called target Q-network, for generating the target Q-values in the update process of the primary Q-network Q_{net}. More precisely, after every C time steps, we clone the Q_{net} and replace the $\hat{Q}_{net}$ by the cloned Q_{net}, then the renewed $\hat{Q}_{net}$ is used for the following C steps to the Q_{net}. This modification addresses divergence or oscillations, thereby stabilising the learning algorithm.

The details of the DDQN algorithm for the control system are explained in Algorithm 1. Specifically, the learning process contains many episodes, and in each episode, the control agent conducts an action according to the ϵ-greedy policy, and it then observes a new state and determines an immediate reward. Next, an experience is stored in the experience replay memory $\mathscr{E}$ for the training process at the next episodes. During the learning process, the control agent acquires a random minibatch of experience to update the primary Q-networks by minimising the following lost function [14].

$$\begin{aligned} L_{\phi}(\theta_{\phi}) = \mathbb{E}_{(s,a,\mathscr{R}(s,a),s')\sim U(\mathscr{M})}[\mathscr{R}(s,a) \\ + \gamma \hat{Q}_{net}(s^{'}, \arg\max_{a'} Q_{net}(s^{'}, a^{'}; \theta); \theta^{-}) - Q_{net}(s, a; \theta_{\phi})]^2, \end{aligned} \tag{6}$$

where γ indicates the discount factor, θ_{ϕ} are parameters of the primary network Q_{net} at episode ϕ, and θ^{-} are the parameters of the target network $\hat{Q}_{net}$.

A fundamental innovation in [5] was to freeze the parameters of the $\hat{Q}_{net}$ for a fixed number of time steps C while updating the Q_{net} by gradient descent, which strengthens the stability of the algorithm. Furthermore, we hence differentiate the loss function in Eq. (6) concerning the weight parameters of Q_{net} and $\hat{Q}_{net}$, and then we can obtain the gradient update as follows:

$$\begin{aligned} \nabla_{\theta_{\phi}} L_{\phi}(\theta_{\phi}) = \mathbb{E}_{(s,a,\mathscr{R}(s,a),s')}[(\mathscr{R}(s,a) + \gamma \hat{Q}_{net}(s^{'}, \arg\max_{a'} Q_{net}(s^{'}, a^{'}; \theta); \theta^{-}) \\ - Q_{net}(s, a; \theta_{\phi}))\nabla_{\theta_{\phi}} Q_{net}(s, a; \theta_{\phi})]. \end{aligned} \tag{7}$$

Rather than calculating the full expectations in the gradient in Eq. (7), the loss function in Eq. (6) can be minimized by the gradient descent algorithm, which is the essential engine of most deep learning algorithms.

It is noted that the target network parameters θ^{-} are used by the primary network parameters θ_{ϕ} for every C time steps and are settled fixed between different updates. In addition, the learning process of the DDQN algorithm is conducted by updating the neural network parameters utilizing prior experiences in an online manner.

Algorithm 1. Efficient IoT traffic granularity acquisition with Double Deep Q-Network

Initialize replay memory $\mathscr{E}$ with a buffer size $\mathbb{N}$.
Initialize the primary Q-network Q_{net} with arbitrary weights θ.
Initialize the target Q-network $\hat{Q}_{net}$ with arbitrary weights $\theta^{-} = \theta$, C (the target network replacement frequency), and T (terminal step in an episode).
for episode $\phi \in \{1, 2, ..., \phi_{max}\}$ **do**
  for $t \in \{1, 2, ..., T\}$ **do**
    Select a random action a_t with probability ϵ, otherwise select $a_t = \arg\max_{a \in \mathscr{A}} Q^{*}(s_t, a_t; \theta)$.
    Perform action a_t at the gateway and observe a new state s_{t+1} and calculate an immediate reward $\mathscr{R}(s_t, a_t)$.
    Store experience $e_t = (s_t, a_t, \mathscr{R}(s_t, a_t), s_{t+1})$ in $\mathscr{E}$ and replace s_t by s_{t+1}.
    Sample random minibatch of experience $(s_j, a_j, \mathscr{R}(s_j, a_j), s_{j+1})$ from $\mathscr{E}$ as $y_j = \mathscr{R}(s_j, a_j) + \gamma \hat{Q}_{net}\left(s_{j+1}, \arg\max_{a_{j+1} \in \mathscr{A}} Q_{net}(s_{j+1}, a_{j+1}; \theta); \theta^{-}\right)$.
    Execute a gradient descent step on $\|y_j - Q_{net}(s_j, a_j; \theta)\|^2$ with respect to the θ.
    Replace $\hat{Q}_{net} \leftarrow Q_{net}$ every C time steps.
    Go to next episode if $t = T$.
  end for
end for

5 Performance Evaluation

5.1 Experiment Setup

To evaluate the proposed approach, we emulate an SDN-based IoT network following the architecture shown in Fig. 1, which consists of 6 OvS (Open vSwitch) running as SDN-based IoT gateway devices and several contained-based hosts (24 hosts/OvS), and it is under control by an ONOS controller (v.1.3) [9]. The emulation operates on the machine with Intel(R) Core(TM) i7-7700 computer with clock speed 3.60 GHz, 64 GB RAM, and NVIDIA GeForce GTX 1080 Ti. Initially, two out of six OvS gateways OvS1 and OvS2 are chosen for the supervision of associated control Agent1 and Agent2, respectively, residing in another machine. To show the improvements of the deep reinforcement learning algorithm, we implement the same setup but apply the Q-learning algorithm [13] to solve our optimization problem.

5.2 Parameter Setting

For Q-learning algorithm, the learning rate α and the discount factor γ are empirically set at 0.6. For the DDQN algorithm, we apply parameters based on the common settings for designing neural networks [5], i.e., two-fully connected hidden layers are used together with input and output layers (as shown in Fig. 4), the size of the hidden layers is 128, the size of the output layer is 10 (which indicates 10 flow matching strategies), the mini-batch size is 32, the replay memory $\mathscr{E}$ holds a size $\mathbb{N}$ of 10,000, the target network replacement frequency C is 100

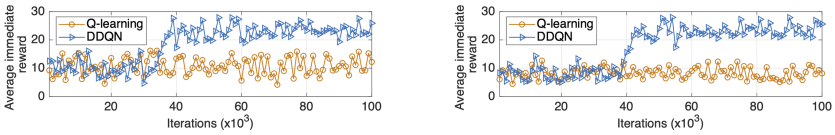
iterations, and the discount factor γ is 0.6. In the learning process of two algorithms, ϵ-greedy algorithm is employed with the initial value of 1.0 and its final value of 0.1 [5], the duration of an iteration is 5 s, and the maximum iterations in an episode T is 100. We let the capacity of flow-tables of the gateway device $\mathscr{G}_{max} = 3000$. Note that we perform 10 different flow matching tactics and 11 primary match fields provided by the ONOS controller [9], that indicates $|\mathscr{C}| = 10$, and $\mathscr{M}_{max} = 11$.

As discussed previously in Sect. 2, we can generalize three common traffic groups, i.e., sensor traffic, monitor traffic, and alarm traffic. *Sensor traffic*: IoT sensor devices generate traffic in a particular period with a low number of packets per flow. *Monitor traffic*: identify by a small number of flows but a significant number of packets per flow. *Alarm traffic*: we assume this traffic group contains a moderate amount regarding both the number of flows and packets per flow. Besides, Hping3 tool [4] is utilized to randomly generate TCP/UDP traffic flows between hosts.

From the above classification, it indicates that $\mathscr{N} = \{sensor, monitor, alarm\}$, and experiments are performed with two settings of required granularity levels as follows: *Diverse*: $\Theta_{sensor} = 3$, $\Theta_{monitor} = 6$, $\Theta_{alarm} = 9$; and *High*: $\Theta_{sensor} = \Theta_{monitor} = \Theta_{alarm} = 9$.

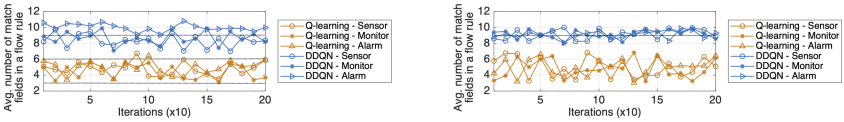
5.3 Results

Convergence Rate of Reinforcement Learning Algorithms. As illustrated in Fig. 5, the average reward value is acquired after every 1,000 iterations during the training period of the Agent1 that controls the gateway OvS1. The convergence rate of the DDQN algorithm is considerably higher than that of the Q-learning algorithm. In particular, the DDQN algorithm requires around 40,000 iterations to achieve the significant average value, i.e., approximately 25.0, for the total actual granularity in both the *Diverse* and *High* settings. This is because the higher required degree of granularity regularly demands actions that give a higher number of match fields in a flow rule, resulting in the higher probability of overflowing the gateway (i.e., more flow rules installed) and in the more critical penalty to the control agent.



(a) Diverse required granularity level (b) High required granularity level

Fig. 5. Average reward derived from Agent1 during the training.

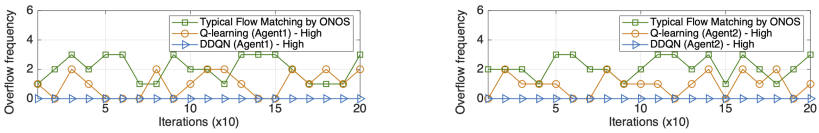


(a) Diverse required granularity level (b) High required granularity level

Fig. 6. Average number of match fields in a flow rule derived from OvS1 during the testing.

Reliable and Win-Win Equilibrium Traffic Granularity. As demonstrated in Fig. 6), one can see that all traffic groups applying the DDQN based solution outperform those utilizing the Q-learning algorithm in terms of the average number of match fields in a flow rule during the testing phase. Accurately, in the case of *Diverse* scenario (Fig. 6 (a)), the traffic groups under the supervision of the DDQN algorithm usually account for a substantial number of match fields in a flow rule that varies from 7.0 to 11.0 ($\mathscr{M}_{max}$), and these degrees all satisfy the prerequisites. For the *High* setting (Fig. 6 (b)), due to a very high precondition of granularity ($\Theta = 9.0$) and a highly dynamic traffic behaviour, the DDQN based control agent cannot satisfy that the requirements all the time. However, in overall, it keeps a significant total degree of the gained granularity.

Data Plane Overflow Avoidance. Next, we measure the overflow frequency caused by different mechanisms during the testing phase with the *High* setting at gateways including OvS1 and OvS2, and reinforcement learning based solutions are compared with the typical ONOS flow setting that uses the MAC address, IP address, and port number. Results presented in Fig. 7 show that no overflow events are observed at two gateways in the DDQN based solution. Accordingly, the DDQN outperforms the regular ONOS flow matching and the Q-learning based mechanism in terms of the ability of data overflow avoidance.



(a) (b)

Fig. 7. Overflow frequency derived from (a) OvS1, (b) OvS2.

6 Conclusion

In this paper, we have developed the efficient IoT traffic monitoring solution employing the advances of SDN and deep reinforcement learning technique. Specifically, we first have introduced the MDP-based flow matching control system

supervising a particular SDN-based IoT gateway. Next, the DDQN algorithm has been developed to maximize the average long-term traffic granularity level of all traffic groups while avoiding the overflow problem. Results obtained from extensive experiments have confirmed that the proposed monitoring approach using the DDQN algorithm can not only provide a reliable and win-win equilibrium traffic granularity level for all groups, but also completely avoid the data plane overflow issue. To the best of our knowledge, this is the first monitoring system in IoT networks, which can efficiently provide a network traffic monitoring capability in an equilibratory manner for different IoT traffic types, and the proposed approach can be applied to various IoT networks with the SDN integration.

References

1. Alam, F., Mehmood, R., Katib, I., Albogami, N.N., Albeshri, A.: Data fusion and IoT for smart ubiquitous environments: a survey. IEEE Access **5**, 9533–9554 (2017). https://doi.org/10.1109/ACCESS.2017.2697839
2. Bera, S., Misra, S., Vasilakos, A.V.: Software-defined networking for internet of things: a survey. IEEE Internet Things J. **4**(6), 1994–2008 (2017). https://doi.org/10.1109/JIOT.2017.2746186
3. Open Networking Foundation: Openflow switch specification version 1.5.1 (2020)
4. Hping3: Description of the hping3 tool, October 2020. www.hping.org
5. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
6. Mu, T.Y., Al-Fuqaha, A., Shuaib, K., Sallabi, F.M., Qadir, J.: SDN flow entry management using reinforcement learning. ACM Trans. Auton. Adapt. Syst. **13**(2), 11:1–11:23 (2018). https://doi.org/10.1145/3281032
7. Nguyen, T.G., Phan, T.V., Nguyen, B.T., So-In, C., Baig, Z.A., Sanguanpong, S.: SeArch: a collaborative and intelligent NIDS architecture for SDN-based cloud IoT networks. IEEE Access **7**, 107678–107694 (2019). https://doi.org/10.1109/ACCESS.2019.2932438
8. Nunes, B.A.A., Mendonca, M., Nguyen, X., Obraczka, K., Turletti, T.: A survey of software-defined networking: Past, present, and future of programmable networks. IEEE Commun. Surv. Tutor. **16**(3), 1617–1634 (Third Quarter 2014). https://doi.org/10.1109/SURV.2014.012214.00180
9. ONOS: Description of the ONOS controller, October 2020
10. Phan, X.T., Fukuda, K.: SDN-Mon: fine-grained traffic monitoring framework in software-defined networks. J. Inf. Process. **25**, 182–190 (2017). https://doi.org/10.2197/ipsjjip.25.182
11. Qiu, T., Chen, N., Li, K., Atiquzzaman, M., Zhao, W.: How can heterogeneous internet of things build our future: a survey. IEEE Commun. Surv. Tutor. **20**(3), 2011–2027 (2018). https://doi.org/10.1109/COMST.2018.2803740
12. Shin, S., Yegneswaran, V., Porras, P., Gu, G.: Avant-guard: scalable and vigilant switch flow management in software-defined networks. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 413–424. ACM, NY, USA (2013). https://doi.org/10.1145/2508859.2516684
13. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning, 1st edn. MIT Press, Cambridge (1998)
14. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)