“© 2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.”

Federated Deep Reinforcement Learning for Traffic Monitoring in SDN-Based IoT Networks

Tri Gia Nguyen, *Senior Member, IEEE*, Trung V. Phan, *Member, IEEE*, Dinh Thai Hoang, *Member, IEEE*, Tu N. Nguyen, *Senior Member, IEEE*, and Chakchai So-In, *Senior Member, IEEE*

***Abstract*—This paper proposes a novel traffic monitoring framework, namely, DeepMonitor, for SDN-based IoT networks to provide fine-grained traffic analysis capability for different IoT traffic types at the network edges. Specifically, we first develop an intelligent flow rule match-field control system, called DeepMonitor agent, for SDN-based IoT edge nodes, taking different granularity-level requirements and their maximum flow-table capacity into consideration. We then formulate the control optimization problem for each edge node employing the Markov decision process (MDP). Next, we develop a double deep Q-network (DDQN) algorithm to quickly achieve the optimal flow rule match-field policy. Moreover, we propose a federated DDQN-based traffic monitoring mechanism to significantly improve the learning performance of the edge nodes. The results obtained through extensive emulations show that by applying the DeepMonitor, the flow-table overflow problem at the edge nodes can be completely bypassed. The average number of match-fields in a flow rule achieved by DeepMonitor is increased by approximately 37% (for medium and diverse granularity-level requirements) and 41.9% (for high granularity-level requirement) compared to that of an existing solution, i.e., FlowStat. Finally, by adopting DeepMonitor, the DDoS attack detection performance of an intrusion detection system can be enhanced by up to 22.83% compared with that of FlowStat.**

***Index Terms*—Traffic Monitoring, Internet of Things, Markov Decision Process, Software-Defined Networking, Deep Reinforcement Learning, Federated Learning.**

I. Introduction

The Internet of Things (IoT) is an emerging technology that aims to connect billions of smart devices in a heterogeneous fashion worldwide [2]. As a result, the IoT is expected to lead to the interconnection of ubiquitous terminal devices in numerous emerging applications [3], e.g., home automation, health care, smart cities, and the automotive industry. However, the nature of such open large-scale communication introduces new security vulnerabilities [4] that have recently attracted users with criminal intentions, e.g., cyberattacks on IoT devices increased by 300% in 2019 according to [5]. In many cases, attackers do not directly target the IoT devices but leverage them to attack other victims, e.g., a Mirai botnet [6] utilized malware-infected IoT devices to launch a distributed denial-of-service attack to a remote website. As a result, IoT device insecurity has become a top concern and top driver of cyberattack traffic on the Internet in 2019 [7]. Therefore, due to IoT systems' heterogeneity and scalability [4], which expose a wide-range attack surface, IoT security issues [8], e.g., authentication, encryption, access control, network, and application security, should be taken into consideration.

A. Motivations

According to [4], [8], cyberattack detection is considered one of the most challenging tasks to effectively defend against IoT cybercrimes due to the difficulties in extracting and analyzing digital information in IoT networks [9], especially at the network edges, which are close to IoT devices [10]. In particular, heterogeneous and large-scale IoT systems have many different types of IoT terminals, e.g., sensors, monitors and alarms, which simultaneously generate various kinds of traffic to the network edge nodes, constituting a highly dynamic traffic load into IoT networks. In addition, conventional intrusion detection systems (IDS) for IoT networks, i.e., signature-based and anomaly-based IDSs, are ineffective because of IoT devices' heterogeneous nature [11]. Consequently, it has become challenging to collect traffic information from IoT networks and then extract and analyze the gathered data [4], [8]. Therefore, it is crucial to acquire a traffic measurement mechanism that can dynamically provide a fine-grained traffic analysis capability to assist security applications in effectively detecting anomalies in IoT networks.

Recently, Software-Defined Networking (SDN) has been introduced as a breakthrough technology in Telco industries with outstanding advantages with respect to dynamics and flexibility in network control and management [12]. SDN introduces a new networking design and management approach that separates the control and data planes. Specifically, the SDN controller has a global view over the network and makes decisions while the SDN switches handle data forwarding. Alternatively, communication between two planes is accomplished via southbound interfaces, e.g., the OpenFlow [13] protocol. The OpenFlow protocol enables an SDN switch to efficiently perform data forwarding using flow-tables, where flow rules are dynamically added/removed/modified by control

Tri Gia Nguyen is with FPT University, Danang 50509, Vietnam. E-mail: tri@ieee.org

Trung V. Phan is with Technische Universität Chemnitz, Chair of Communication Networks, 09126 Chemnitz, Germany. E-mail: trung.phan-van@etit.tu-chemnitz.de

Dinh Thai Hoang is with the School of Electrical and Data Engineering, University of Technology Sydney, Australia. E-mail: hoang.dinh@uts.edu.au

Tu N. Nguyen is with the Department of Computer Science, Kennesaw State University, Marietta, GA 30060, USA. E-mail: tu.nguyen@kennesaw.edu

Chakchai So-In is with Applied Network Technology Laboratory, Department of Computer Science, Faculty of Science, Khon Kaen University, Khon Kaen 40002, Thailand. E-mail: chakso@kku.ac.th

Corresponding author: Chakchai So-In (chakso@kku.ac.th).

A part of this study was presented at 9th International Conference on Computational Data and Social Networks (CSoNet), Dallas, TX, USA, December 11–13, 2020 [1].

messages from the SDN control plane. Notably, a flow rule is defined as a stream of packets having the same matching criteria in their match-fields, e.g., MAC addresses and IP addresses, and a flow rule is constructed by a combination of counters, match-fields[1], and instructions [13] (as illustrated in Fig. 1). By adopting the SDN architecture, a wide range of network control and management tasks, e.g., network monitoring [14], have been achieved.

According to [15], the integration of SDN with IoT systems has enabled new features and abilities to fulfill the requirements of IoT in an efficient, scalable, seamless, and cost-effective manner, which conventional network technologies fail to achieve. In particular, SDN integration is considered at four IoT networking aspects, i.e., edge, access, core, and data center. First, the limitations in IoT data aggregation, flow monitoring, and admission control can be solved appropriately for SDN-based edge networking. Next, the requirements at the access networking in information access, wireless access networking, and rule-caching are effectively fulfilled with the adoption of SDN. For core networking, the SDN improves the security of networks, the performance of packet classification and routing. Moreover, the SDN can dynamically perform traffic engineering tasks and online resource sharing for virtual machines at data centers where cloud-based IoT applications operate. Finally, the SDN is considered a prominent solution for resolving latency-related issues, e.g., congestion control in smart factories [16], quality of experience control for 3D video streaming [17], and multichannel reassignment tasks in IoT networks [18].

One of the essential purposes of integrating SDN into IoT networks is to collect traffic flow information [11], [15], e.g., flow rule statistics that can be used for security analysis [14], from the data plane devices. For instance, the authors in [19] developed an adaptive statistics collection algorithm, namely, PayLess, that renders detailed traffic information at runtime without requiring substantial control overhead by controlling the number of flow rule statistic messages between the control plane and the data plane. In [20], a flow rule assignment mechanism, namely, FlowStat, was introduced to select a specific SDN switch in the data plane to install an exact-match flow rule (which contains all possible match-fields) to obtain detailed statistics for a particular flow rule. Meanwhile, flow rules at switches that are not chosen carry only the source and destination IP address in the match-fields to circumvent the overflow problem, thereby enabling higher data plane forwarding performance while still providing accurate flow rule information. In OpenMeasure [21], the authors introduced an efficient flow rule measurement solution in SDN that performs adaptive measurements by applying an online learning algorithm and is constrained by the available ternary content addressable memory (TCAM). Moreover, the authors proposed two lightweight heuristic algorithms for flow rule assignments to increase flow measurement accuracy. As a result, the switches' flow rules are dynamically updated, and the most informative flows are measured at runtime.

[1]A flow rule can contain a large number of match-fields (e.g., more than 40 fields [13]), which are determined by the control plane.

Moreover, with the aim of achieving significant network control performance by utilizing the advantages of SDN in IoT networks, several SDN-based traffic flow aggregation[2] solutions [23]–[25] were proposed to efficiently control traffic flows for different traffic engineering tasks in the data plane. For example, considering the QoS provision ability, the authors in [23] introduced an admission control scheme with flow aggregation in SDN-based networks. Specifically, network calculus is utilized to optimize the admission control process by selecting individual flows that are then merged into a single aggregate flow. The amount of bandwidth and buffer at an the SDN switch is dynamically determined by a proposed analytical technique to meet traffic flow QoS requirements. The obtained results show that the proposed approach enhances bandwidth and buffer utilization in the SDN switches. Similarly, a scheme, called FlowMan [24] was proposed to ensure high QoS metrics, i.e., high throughput and low delay, in heterogeneous flow management in an SDN-based IoT environment using a generalized Nash bargaining game theory. Precisely, the first part of the FlowMan scheme is to select the optimal incoming flow rate using the bisection approach and then to aggregate these flows, while the second part performs the optimal mappings of traffic flows to the appropriate switches. FlowMan can effectively reduce network delay and increase network throughput in the presence of heterogeneous flows. In addition, Mauboubi *et al.* [25] proposed an intelligent SDN-based traffic (de)aggregation solution that divides the TCAM of a switch into two parts. The first part optimally aggregates incoming flows, and the second part deaggregates important flows into more informative flows. As a result, this approach minimizes the overhead of analyzing obtained per-flow statistics at the control plane.

The aforementioned studies focus on either methods [19]–[21] to install and measure informative flows, e.g., exact-match flow rules that contain all possible match-fields, or flow aggregation techniques [23]–[25] to reduce the number of flow rules by using fewer match-fields in a flow rule. However, the existing approaches [19]–[21], [23]–[25] are ineffective to implement in heterogeneous and large-scale IoT networks due to highly dynamic traffic behavior. For example, by aggregating individual flows into a few flows, the network visibility is significantly decreased, making the control plane unable to track and monitor network traffic flows for security or forensic analysis. Furthermore, the TCAM of an SDN switch can quickly reach its limit when placing many exact-match flow rules in flow-tables at a time, leading to a flow-table overflow problem that degrades the forwarding performance of the switch [22]. Thus, a dynamic solution that can simultaneously provide significant traffic analysis capability while avoiding the overflow problem's flow-tables in SDN-based IoT networks is urgently needed.

Recently, deep reinforcement learning (DRL) techniques [26], [27] have emerged as a highly effective tool to address many dynamic optimization problems in the areas of communications and networking [28], e.g., jamming attack defense

[2]Flow aggregation is to merge individual flows into one flow by first removing all related flows and then installing a new flow with a general match-field formation [22] for all removed flows.

in wireless networks [29], radio frequency energy harvesting optimization [30], dynamic routing in SDN-based IoT networks [31], efficient Deep Transfer Learning [32], and risk assessment for private sensor IoT devices [33]. In the context of SDN-based network traffic control, the study that is most closely related to this paper is our recent work [34]. In [34], we proposed an approach that utilizes a supervised learning algorithm to anticipate the flow-table overflow problem. Then, relying on the predicted overflow result, a DRL algorithm makes decisions to maximize the average number of match-fields in a flow rule in the SDN data plane. However, one of the main drawbacks of this solution is that the supervised algorithm is more likely to make incorrect predictions in a heterogeneous and dynamic traffic environment. Moreover, the supervised algorithm is not updated to adapt its prediction performance to the current network traffic behavior at runtime. As a result, the DRL algorithm has difficulty performing the correct policies based on inaccurate inputs, leading to a very slow convergence in the learning process. In this paper, the DRL technique controls the match-fields in flow rules with the flow-table overflow issue as one of the main constraints. If the flow-tables of an SDN switch reach the flow rule limit, then the switch buffer quickly becomes full. Afterward, all incoming packets are directly forwarded to the control plane [13], making the SDN controller become a bottleneck and possibly suspended, e.g., saturation attack targeting the SDN control plane [35]. To address this issue, in this work, we propose to severely punish the DRL-based control agent in its reinforcement learning process whenever the flow-tables of the switch reach the maximum flow rule capacity, allowing the learning algorithm to proactively avoid the flow-table overflow. More importantly, in this work, we consider the issue of collaborative machine learning of complex models among distributed agents, which was not discussed and addressed in [34]. Then, we propose a federated deep reinforcement learning algorithm to improve participating agents' learning performance in a decentralized manner.

B. Our Proposal and Contributions

In this paper, we propose a novel traffic monitoring framework, namely DeepMonitor, developed from the federated deep reinforcement learning approach to maximize the traffic flow statistic details (or *the traffic granularity degree* hereafter) for different traffic groups (e.g., sensors, monitors and alarms) at the edge layer of an SDN-based IoT network. In particular, we first develop a flow rule match-field control system, called a DeepMonitor agent, at each SDN-based IoT edge node constrained by its maximum flow-table capacity and granularity level requirements of traffic types. To address the dynamics of the flow rule match-field control system, we develop a Markov decision process (MDP) framework. This framework enables the DeepMonitor agent to make decisions in a real-time manner based on its current status, i.e., the traffic granularity degree of all traffic groups derived from the edge device. Then, we develop a deep reinforcement learning algorithm, i.e., DDQN, that enables the DeepMonitor agent to quickly reach the optimal policy without demanding further information from the environment in advance. Concerning the IoT system's heterogeneity and scalability, there is a tremendous amount of traffic data generated from various IoT devices; hence, each DeepMonitor agent at an edge node may not be able to learn all information. Furthermore, sensitive data can be endangered, e.g., industrial deployments, if the data set is shared among DDQN agents. Additionally, significant effort is required to transfer shared data sets for training among agents considering communication costs. Therefore, we develop a highly effective federated deep reinforcement learning technique to improve the learning performance of the DDQN algorithm at DeepMonitor agents, which can significantly reduce the training time and quickly achieve the optimal policy for the whole system.

As a proof-of-concept, we evaluate the DeepMonitor framework in an SDN emulation environment, and the results show that the proposed framework can achieve remarkable improvements in terms of not only providing a fine-grained monitoring capability for all IoT traffic groups but also protecting edge devices from the overflow problem. In particular, the average number of match-fields in a flow rule is increased[3] by approximately 37% (for medium and diverse required traffic granularity settings) and by 41.9% (for high required traffic granularity setting) compared to that of the FlowStat solution [20]. This improvement significantly outperforms that of the typical flow matching strategy of ONOS (Open Network Operating System) [36]. Moreover, the federated DDQN solution can reduce the learning loss by up to 66% and improve the learning rate by up to 40% compared to those of the centralized mechanism. Finally, we evaluate the monitoring capability of the DeepMonitor framework when a DDoS attack occurs in the network. The emulation results obtained by employing the federated DDQN algorithm show that the DeepMonitor framework can effectively support the intrusion detection system to improve the attack detection performance by 22.83% compared to that of FlowStat.

The major contributions of this paper can be summarized as follows:

- To maximize the traffic granularity degree for all IoT traffic groups while protecting the SDN-based IoT edge nodes from the flow-table overflow problem, we develop a flow rule match-field control system for each edge node and then formulate its objective function utilizing the MDP model.
- To quickly obtain the optimal policy of the considered MDP system without requiring information about traffic behavior of IoT devices in advance, we develop a DDQN-based flow rule match-field control algorithm. The proposed algorithm can learn the match-field determination process by interacting with the flow-tables of SDN-based edge devices, thereby maximizing the average long-term traffic granularity degree of all traffic groups and providing the maximum traffic granularity degree.
- To improve the learning performance of the DDQN algorithm at DeepMonitor agents, we propose a federated

[3]Note that the more match-fields a flow rule contains, the more specific information that flow rule can provide [34].

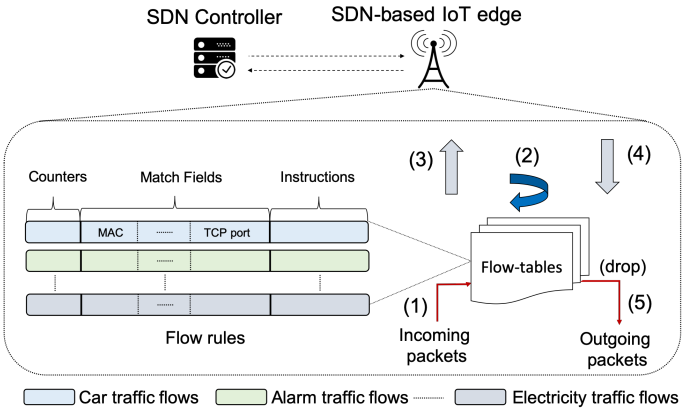


Fig. 1. Typical packet forwarding logic in an SDN-based IoT network.

deep reinforcement learning-based IoT traffic monitoring mechanism to reduce the DDQN training time while quickly obtaining the optimal policy.

- Finally, extensive experiments are conducted to demonstrate the efficiency and capability of the DeepMonitor framework compared to other solutions in a ubiquitous IoT environment. The obtained results show that by applying the optimal flow rule match-field control policy, the traffic granularity degree is significantly enhanced while completely avoiding the flow-table overflow, leading to a remarkable DDoS attack detection performance.

This paper is organized as follows. Section II presents the DeepMonitor framework and describes the flow rule match-field control system at the edges in detail. Then, Section III formulates the control optimization problem, and Section IV-B introduces the DDQN algorithm. Next, Section V provides details about the proposed federated deep reinforcement learning-based solution. Finally, a performance evaluation is presented in Section VI, and conclusions are outlined in Section VII.

II. DeepMonitor Framework

In this section, we first study how IoT traffic is delivered with the synthesis of SDN; then, the main attributes of the DeepMonitor framework are discussed in detail.

A. Typical SDN Packet Forwarding Process

As exhibited in Fig. 1, the typical packet forwarding process at an SDN-based IoT edge device can be briefly explained as follows:

- Step 1: An incoming packet arrives at the SDN-based IoT edge device.
- Step 2: The matched flow rule search process in flow-tables is accomplished by comparing the packet header information[4] to *match-fields* in every flow rule. If the header information is matched to all *match-fields* of a flow rule, then a matched flow rule has been found[5], and the process proceeds to Step 5.
- Step 3: If there is no matched flow rule, a *table-miss* event occurs for the arrived packet, and the edge forms a packet_in message and sends the message to its corresponding SDN controller for further supervision.
- Step 4: The SDN controller performs its traffic forwarding schemes and installs a new flow rule into the edge with a suitable selection of match-fields and instructions.
- Step 5: The incoming packet is supervised by *Instructions* in the matched flow rule, e.g., *forward* the packet to a specific edge port or *drop* the packet.

In summary, in a flow rule, the *match-field* determination is critical to the SDN packet forwarding and flow rule matching at the edge devices. Precisely, the packet forwarding can be performed only through a course of actions, either (1)(2)(5) or (1)(2)(*drop*) or (1)(2)(3)(4)(5), as presented in Fig. 1. For example, an incoming packet that does not match any flow rules in the flow-tables of an SDN switch and follows all five steps results in an extra delay in the packet forwarding between users/clients. Therefore, flow rule match-field control problems should be carefully considered [20], [37]. In addition, the traffic granularity or the traffic flow statistic details collected by the control plane [38] are principally determined by the total number of match-fields that a flow rule contains at the edge device, and the more match-fields a flow rule contains, the more detailed flow information or the higher degree of traffic granularity it provides [34]. Therefore, to adjust the traffic granularity degree, the determination of the match-fields in a flow rule should be properly controlled. In other words, a flow rule match-field control system should be implemented to achieve a dynamic traffic granularity level and to obtain efficient traffic monitoring capability in SDN-based IoT networks.

B. DeepMonitor Framework

As stated in [39], SDN can be deployed at different networking levels, i.e., edge (or access), core, and cloud networks, thereby covering IoT traffic control and management from devices up to cloud services. Therefore, we propose a traffic monitoring framework, namely, DeepMonitor, consisting of four main components, i.e., SDN-based IoT edge nodes, DeepMonitor agents, SDN controllers, and an aggregation server, that are located in different levels of SDN-based IoT infrastructures, as shown in Fig. 2. In particular, DeepMonitor agents are integrated with the SDN-based IoT edges that are connected to the SDN control plane via southbound interfaces. By means of this approach, the SDN controllers can implement traffic control and management policies at the edge nodes. In our approach, to perform a change of the flow rule match-field strategy at an edge node, the integrated DeepMonitor agent sends a request to the associated SDN controller, and a control message is then sent to the edge node to implement the policy at its flow-tables. In addition, for the federated learning

[4]The header of a packet regularly includes information, e.g., MAC, IP addresses and TCP/UDP port numbers.

[5]For ease of comprehension, we study this simple matching strategy in this paper; more complex matching methods can be found in [13], e.g., multiple flow-tables searching.

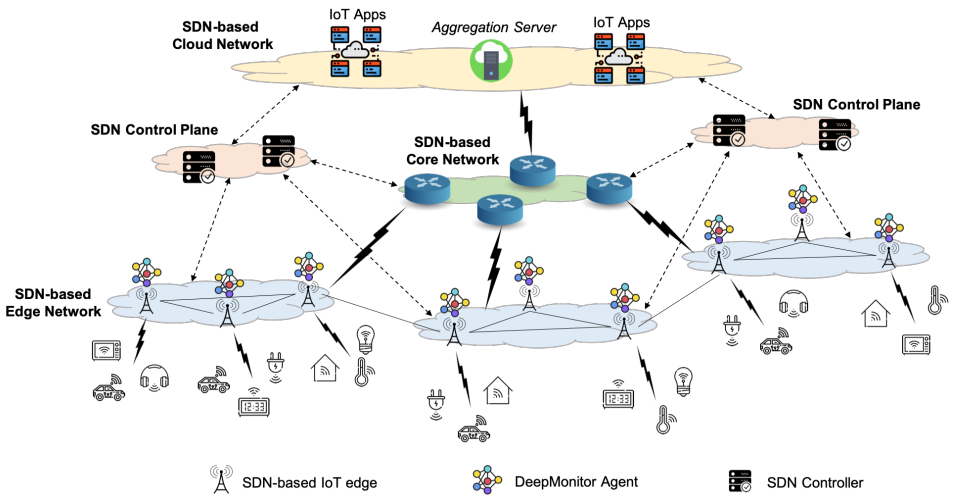


Fig. 2. DeepMonitor framework in SDN-based IoT networks.

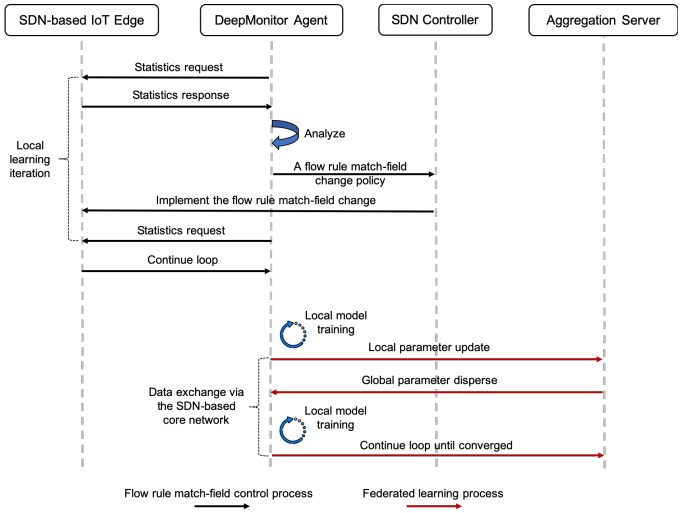


Fig. 3. Overall workflow of the DeepMonitor framework.

process, the data exchange between DeepMonitor agents and the aggregation server is transmitted over the SDN-based core network. Notably, the federated learning data exchange is handled as normal traffic forwarding in the SDN-based core and cloud networks, e.g., IoT devices send and receive data from their cloud-based applications.

With respect to the operation of the DeepMonitor framework, as shown in Fig. 3, there are two separate processes:

- *Flow rule match-field control at the edge*: This process is performed at an SDN-based IoT edge supervised by a DeepMonitor agent and an associated SDN controller. Specifically, IoT traffic statistics from the edge are collected and analyzed by the DeepMonitor agent. Afterwards, the agent forms a flow rule match-field change policy and sends it to the SDN controller to implement the policy at the edge device's flow-tables. The same process is repeated after a certain period, referred to as a local learning iteration in the remainder of the paper, to evaluate the policy's effectiveness and to identify the optimal policy. This process is referred to as reinforcement learning, and in this study, we develop the DDQN algorithm to learn the flow rule match-field control.
- *Federated deep reinforcement learning for agents*: To improve the learning performance of the DDQN algorithm at DeepMonitor agents, a federated learning algorithm is developed. Specifically, the DDQN algorithm at each agent is first trained by its collected data set for a predetermined time period due to the online learning process. Then, for the next federated learning round, the agent uploads its local parameter update to the aggregation server and receives the global parameter disperse. This learning loop is repeated until the DDQN algorithm converges. More details are provided in Section V.

III. System Model and Problem Formulation

In this section, we first study details of a flow rule match-field control system for each SDN-based IoT edge node utilizing the MDP framework. The optimization problem is then discussed and formulated.

A. System Model

As discussed previously, it is crucial to accurately and dynamically determine the correct collection of match-fields in traffic flows of various IoT devices to acquire efficient traffic monitoring capability in SDN-based IoT networks. Accordingly, we develop a flow rule match-field control system, referred to as DeepMonitor agent, residing in an edge device and consisting of a statistics collector, a control agent, a database, and a flow rule match-field policy maker, as shown in Fig. 4. The control agent supervises the operation of flow rule match-field determination in flow-tables at an SDN-based IoT edge i. Notably, we design the control mechanism for single edge node i. Hence, if there are many IoT edges that need to be supervised, it is feasible to initiate DeepMonitor agents at these edges to obtain sufficient traffic granularity at all monitoring points.

TABLE I
IMPORTANT NOTATION

Notation	*Definition*
$\mathcal{N}$	Number of IoT traffic types passing through the edge i
Θ_n	Required granularity level or number of match-fields in a flow rule
$\mathcal{M}_{max}$	Maximum number of match-fields in a flow rule
$\mathcal{G}_{max}$	Maximum number of flow rules in the flow-tables of an edge device
θ_n	Actual current granularity of the particular traffic group n
w_n	Importance of the particular traffic group n
$\mathcal{F}_t$	Total number of flow rules in the edge at time step t
f_n	Total number of flow rules of a particular traffic group n
$\mathcal{S}, \mathcal{A}, \mathcal{R}$	State space, action space, and immediate reward function
s_t, a_t	A state and an action taken by the control agent at time step t
$\mathcal{X}$	A collection of all match-field combinations
$\Delta_{\mathcal{R}_n}$	Difference between the gained and required values of a particular traffic group n
γ	Discount factor
Q_{net}, $\hat{Q}_{net}$	Primary Q-network and target Q-network
$\mathcal{E}$, N	Replay memory pool and memory buffer size
e_t	An experience in $\mathcal{E}$
ϕ, ϕ_{max}	An episode and maximum number of episodes in the training phase
θ_ϕ, θ_ϕ^-	Weight parameters of Q_{net} and $\hat{Q}_{net}$ at episode ϕ
C	Number of time steps after which the target network is replaced by the primary network
T	Number of time steps in an episode
m	Number of participating DeepMonitor agents
$\mathcal{T}$	Number of federated learning rounds
$\mathcal{D}$	Set of local data sets
$\mathbf{w}_G^0$	Initial global parameters
η	Learning rate of the federated learning
$\mathbf{w}_i^{\tau+1}$, $\mathbf{w}_G^{\tau+1}$	Local update and global update at the learning round i

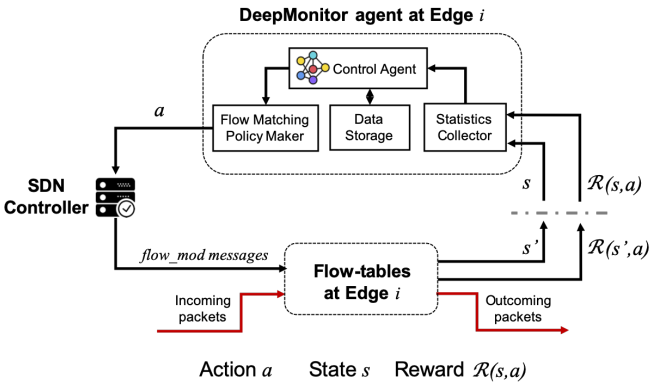


Fig. 4. Reinforcement learning-based model of DeepMonitor agent.

According to a recent survey [40], IoT traffic is heterogeneous from a macro perspective but group-specific from the local network perspective. Concretely, IoT devices serving different applications can be individually connected in separate virtual local area networks (VLANs), which can be supervised at the edge of the IoT networks, i.e., the SDN-based IoT edge i, because VLAN_ID is one of the match-fields in a flow rule. Generally, we first assume that there exist $\mathcal{N}$ IoT traffic types passing through edge i. For each traffic group, the required granularity for monitoring purposes, i.e., the number of match-fields in a flow rule, is denoted by Θ_n ($n \in \mathcal{N}$), where $0 < \Theta_n \leq \mathcal{M}_{max}$ and $\mathcal{M}_{max}$ is the maximum number of match-fields in a flow rule. Specifically, the control agent must maintain at least a certain amount of information, i.e., a significant number of match-fields in a flow rule, to ensure the operation of the network monitoring application. Initially, the control agent implements a policy to move the flow rule match-field strategy of specific traffic groups into the edge via the SDN controller. To perform a flow rule match-field scheme change, the SDN controller sends $flow\_mod$ messages to the edge to delete all existing flow rules related to the traffic group and to install flow rules with the new match-field combination in the flow-tables of the edge. Afterwards, on the basis of the collected statistical data, the control agent estimates the effectiveness of the executed policy, i.e., the traffic granularity degree of all traffic groups in the edge, in comparison to the Θ_n value (first constraint in (1)) and the limit of flow-tables of the edge device (second constraint in (1)), denoted as $\mathcal{G}_{max}$. These processes are repeated to explore new policies and observations. Accordingly, we can formulate the objective function of the control system as follows:

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

where θ_n and w_n are the actual current granularity and the importance of the particular traffic group n, respectively; and $\mathcal{F}_t$ represents the total number of flow rules in the edge at time step t. Specifically, the value of θ_n is calculated as

$$\theta_n = \frac{\sum_{f=1}^{f_n} \mathcal{M}_f}{f_n}, \tag{2}$$

where f_n ($f_n \leq \mathcal{G}_{max}$) is the total number of flow rules of traffic group n at the flow-tables of the edge and $\mathcal{M}_f$ is the number of match-fields in a flow rule f.

Theoretically, by trying to perform more changes in the match-fields of flow rules at the edge's flow-tables, the control

agent might get closer to its objectives. Nevertheless, it becomes challenging to discover an optimal flow rule match-field policy in SDN-based IoT networks due to the heterogeneity and scalability [15], [41].

B. Problem Formulation

To maximize the granularity degree of the traffic of all $\mathcal{N}$ traffic groups at the SDN-based IoT edge, we develop an MDP framework [42] with the DeepMonitor system, as exhibited in Fig. 4. This framework enables the DeepMonitor system to dynamically implement optimal actions based on its observations to maximize its average long-term reward. The MDP is defined by a tuple $<\mathcal{S},\mathcal{A},\mathcal{R}>$, where $\mathcal{S}$ signifies the state space, $\mathcal{A}$ expresses the action space, and $\mathcal{R}$ represents the immediate reward function.

State Space: As aforementioned, from the perspective of each local network, there exist $\mathcal{N}$ traffic groups transferring traffic through the edge [40]. In this paper, we aim to maximize the total traffic granularity levels of all groups. Then, the state space of the edge can be defined as follows:

$$\mathcal{S} \triangleq \{((\theta_1, f_1), ..., (\theta_n, f_n), ..., (\theta_\mathcal{N}, f_\mathcal{N}))\}, \tag{3}$$

where θ_n ($\theta_n \leq \mathcal{M}_{max}$) and f_n ($f_n \leq \mathcal{G}_{max}$) represent the actual current granularity levels and the total number of flow rules of the traffic group n, respectively. The reason for selecting f_n is that a higher number of flow rules entails a greater number of match-fields in a flow rule of the traffic group. Thus, the state of the system is $s = ((\theta_1, f_1), ..., (\theta_n, f_n), ..., (\theta_\mathcal{N}, f_\mathcal{N}))$.

Action Space: Recall the objective function in (1); one of the prerequisites is to always keep all θ_n values greater than or equal to their corresponding Θ_n values. Thus, the control agent should quickly take actions whenever θ_n does not satisfy this requirement. However, the main objective is to maximize the total actual traffic granularity; hence, the control agent should perform actions even in the case that all θ_n values meet the prerequisites. Moreover, the strategy is to implement actions for only the traffic group with the lowest θ_n value at a time step because changing the flow rule match-field strategies of all groups simultaneously can lead to a significant variation in the flow-tables of the edge and more latency for communications due to table-miss events by the new match-field combinations that generate packet_in messages to the control plane [35].

$\mathcal{X} = \{x_1, x_2, ..., x_k\}$ depicts a collection of all match-field combinations, e.g., x_k = <"*matchTcpUdpPorts*", "*matchIpv4Address*",...> defined in the *Reactive Forwarding* application of the ONOS [36]. Therefore, the action space for changing the flow rule match-field tactic in the edge is determined by

$$\mathcal{A} \triangleq \{a : a \in \mathcal{X}\}. \tag{4}$$

Note that if the chosen action is the same as the current action of the selected traffic group, the control agent goes into sleep mode and waits for the next state observation.

Immediate Reward Function: Whenever, by performing an action, the total number of flow rules $\mathcal{F}_t$ in the edge reaches the limit $\mathcal{G}_{max}$, which can lead to either forwarding performance degradation of the edge [22] or a packet_in flood to the SDN controller [35], the control agent should be immensely punished. By contrast, the more match-fields in a flow rule of a group, the higher the granularity degree the group provides. In addition, if there exists a group gaining $\theta_n < \Theta_n$, a small punishment is applied to penalize the immediate reward of the control agent; the punishment is calculated as the difference between the gained θ_n and required Θ_n values, $\Delta_{\mathcal{R}_n} = \Theta_n - \theta_n$, for group n. Accordingly, we establish the immediate reward function for the control agent as the total gained granularity values minus the total punishments for groups not meeting their requirements, which is represented as follows:

$$\mathcal{R}(s,a) = \begin{cases} \sum_{n=1}^{\mathcal{N}} w_n \theta_n, \text{ if } \theta_n \geq \Theta_n, \ \forall n \in \mathcal{N} \text{ and } \mathcal{F}_t < \mathcal{G}_{max}, \\ \sum_{n=1}^{\mathcal{N}} w_n (\theta_n - \Delta_{\mathcal{R}_n}), \text{ if } \theta_n < \Theta_n \text{ and } \mathcal{F}_t < \mathcal{G}_{max}, \\ -\mathcal{M}_{max}, \text{if } \mathcal{F}_t = \mathcal{G}_{max}, \end{cases} \tag{5}$$

where $\mathcal{F}_t$ is the current total number of flow rules in the edge and $\mathcal{M}_{max}$ is the maximum number of match-fields in a flow rule.

C. Optimization Formulation

We formulate an optimization problem to achieve the optimal policy, referred to as $\pi^*(s)$, that maximizes the control system's average long-term reward, i.e., the traffic granularity degree of all groups at the edge. Thus, the optimization problem is defined as

$$\begin{aligned} \max_{\pi} \quad & \mathfrak{R}(\pi) = \sum_{t=1}^{\infty} \mathbb{E}(\mathcal{R}(s_t, \pi(s_t))), \\ \text{s.t.} \quad & \begin{cases} \mathcal{R} \in \mathbb{R}, \quad s_t \in \mathcal{S}, \quad \pi(s_t) \in \mathcal{A}, \\ 0 < \Theta_n \leq \theta_n \leq \mathcal{M}_{max}, \quad \forall n \in \mathcal{N}, \\ \mathcal{F}_t < \mathcal{G}_{max}, \\ \sum_{n=1}^{\mathcal{N}} w_n = 1.0, \end{cases} \end{aligned} \tag{6}$$

where $\mathfrak{R}(\pi)$ and $\mathcal{R}(s_t, \pi(s_t))$ represent the average reward function and the immediate reward function obtained by the control agent under policy π at time step t, respectively.

IV. Efficient Traffic Monitoring with Double Deep Q-Network at The Edge Layer in SDN-based IoT Networks

In this section, we first illustrate the application of the Q-learning algorithm to solve the optimization problem in (6). However, due to a prolonged learning convergence performance of the Q-learning, we then introduce a deep reinforcement learning technique to obtain the optimal policy quickly.

A. Q-Learning Based Flow Rule Match-Field Control

In this section, we first propose the Q-learning algorithm to solve the optimization problem in (6). In particular, the Q-learning algorithm [42] utilizes a Q-table to store all state-action pairs in the considered environment. The Q-learning control agent is able to learn from its decisions at each time step or iteration, and the learning algorithm can converge to the optimal control policy $\pi^*(s)$ after a certain number of time steps [42]. Specifically, the expected return for the control

Algorithm 1 Q-Learning-Based Flow Rule Match-Field Control Algorithm

Inputs: For a state-action pair (s,a) $\forall s \in \mathcal{S}$, $a \in \mathcal{A}$, arbitrarily initialize a Q-table entry; initialize values of α, γ, ϵ, and T (terminal condition in an episode).
for episode $\phi \in \{1, 2, ..., \phi_{max}\}$ **do**
  for $t \in \{1, 2, ..., T\}$ **do**
    Given current state s_t.
    Perform action a_t according to the ϵ policy.
    Obtain a new state s_{t+1} and calculate an immediate reward $\mathcal{R}$.
    Update the Q-table entry for $Q(s_t, a_t)$ by

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha[\mathcal{R}(s_t, a_t) + \gamma \max_a Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)], \quad (7)$$

    Update $s_t \longleftarrow s_{t+1}$.
    Go to the next episode if $t = T$.
  end for
end for
Outputs $\pi^*(s) = \arg\max_a Q^*(s, a)$.

agent of state s under policy π is represented as $\vartheta_\pi(s) : \mathcal{S} \longrightarrow \mathbb{R}$, which is determined as follows:

$$\begin{aligned} \vartheta_\pi(s) &= \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma \mathcal{R}(s_t, a_t) | s_t = s \right] \\ &= \mathbb{E}_\pi \left[\mathcal{R}(s_t, a_t) + \gamma \vartheta_\pi(s_{t+1}) | s_t = s \right], \forall s \in \mathcal{S}, \end{aligned} \quad (8)$$

where $\gamma \in [0, 1]$ is a discount factor that represents the importance of the long-term reward [42]. To find the optimal policy $\pi^*(s)$, the optimal action at a given state can be obtained using the following optimal value function:

$$\vartheta^*(s) = \max_a \{\mathbb{E}_\pi \left[\mathcal{R}(s_t, a_t) + \gamma \vartheta_\pi(s_{t+1}) | s_t = s \right]\}, \forall s \in \mathcal{S}. \quad (9)$$

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

$$Q^*(s, a) \triangleq \mathcal{R}(s_t, a_t) + \gamma \mathbb{E}_\pi \left[\vartheta_\pi(s_{t+1}) \right], \forall s \in \mathcal{S}. \quad (10)$$

Then, the optimal value function $\vartheta^*(s)$ can be referred to as $\vartheta^*(s) = \max_a \{Q^*(s, a)\}$. By performing different actions a on the controlled environment, the optimal Q-function value for all state-action (s,a) pairs can be achieved. Concretely, the Q-function is updated in each iteration by

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha[\mathcal{R}(s_t, a_t) + \gamma \max_a Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)], \quad (11)$$

where $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$, $Q_t(s_t, a_t)$ is the Q-value for a state-action pair (s_t, a_t), $\mathcal{R}(s_t, a_t)$ represents the immediate reward derived from the gateway at time step t, $\gamma \in [0, 1]$ is the discount factor and $\alpha \in [0,1]$ is the learning rate. Furthermore, to relieve the exploration and exploitation that have an immediate influence on the convergence rate of any learning algorithm, the ϵ-greedy mechanism [42] is employed. In the learning period, the Q-learning control agent initializes a Q-table to store all state-action pairs and subsequently updates it by applying (11). Accordingly, the agent can obtain a trained Q-table.

Algorithm 1 presents the implementation details of the Q-learning based flow rule match-field control algorithm. According to [43], the Q-learning algorithm can obtain the optimal policy in a sufficient period of learning in the case that the state space and action space are well-bounded. However, the MDP model under consideration has billions of states based on combinations among $\mathcal{N}$ traffic groups, as illustrated in (3), leading to a very slow convergence rate [43]. To address this issue, in the following, we develop a DDQN-based flow rule match-field control algorithm to obtain the optimal policy for the control agent quickly, thereby efficiently producing the maximum granularity level for IoT traffic monitoring tasks.

B. Double Deep Q-Network-Based Flow Rule Match-Field Control

In this section, we propose the DDQN algorithm [27] to solve the slow convergence problem discussed earlier. Specifically, the DDQN algorithm utilizes a deep neural network as a nonlinear function approximator. In the DDQN algorithm, an action is chosen according to a primary neural network Q_{net}, and a target neural network $\hat{Q}_{net}$ is applied to evaluate the target Q-value of the implemented action, as exhibited in Fig. 5. According to the outcomes in [26], the control agent performance can vary significantly whenever a nonlinear function approximator is applied because a small variation in Q-values can immensely influence the policy in the learning period. Accordingly, in the DeepMonitor control system, the data distribution and the relationships between the current and the target Q-values can be diversified. To address this weakness, we implement an experience replay scheme and a target Q-network.

Experience replay mechanism: We define a replay memory pool $\mathcal{E}$ to store the control agent's experience at each time step, $e_t = [s_t, a_t, \mathcal{R}(s_t, a_t), s_{t+1}]$, over many episodes, and $\mathcal{E} = \{e_1, ..., e_t\}$. Through the learning process, the collected samples of experience from $\mathcal{E}$ or a minibatch of experiences $U(M)$ is selected at random to update the neural networks. This strategy can achieve high data efficiency because the neural networks can be trained several times by the same experience. Furthermore, the randomizing method eliminates the correlations between the observed samples, thereby reducing the variance of the neural network updates.

Target Q-network: The Q-values are corrected, leading to uncertainties in the value calculations during the learning period, which destabilizes the learning algorithm. Hereafter, to improve the stability of the learning algorithm with neural networks, a separate network $\hat{Q}_{net}$ (target Q-network) is applied to produce the target Q-values for the primary Q-network Q_{net}. More accurately, Q_{net} is cloned, and $\hat{Q}_{net}$ is replaced by the cloned Q_{net} after C time steps. This modification addresses divergence, thereby stabilizing the learning algorithm.

Algorithm 2 presents the details of the learning DDQN algorithm applied for the DeepMonitor control agent. In particular, for an episode, the control agent takes an action following the ϵ-greedy policy and then observes a new state and calculates

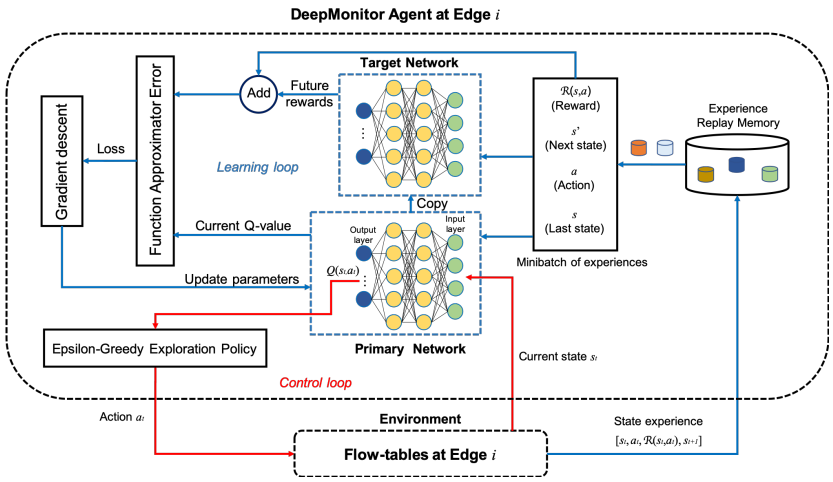


Fig. 5. Double deep Q-network-based model of a DeepMonitor agent at the edge.

Algorithm 2 Double Deep Q-Network-Based Flow Rule Match-Field Control Algorithm

Initialize replay memory $\mathcal{E}$ with buffer size 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 is the target Q-network replacement frequency, and T is the 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, (with probability $1-\epsilon$) select $a_t = \arg\max_{a \in \mathcal{A}} Q^*(s_t, a_t; \theta)$.
    Perform action a_t at the edge, observe a new state s_{t+1} and calculate an immediate reward $\mathcal{R}(s_t, a_t)$.
    Store experience $e_t = (s_t, a_t, \mathcal{R}(s_t, a_t), s_{t+1})$ in $\mathcal{E}$ and replace s_t by s_{t+1}.
    Sample random minibatch of experience $(s_j, a_j, \mathcal{R}(s_j, a_j), s_{j+1})$ from $\mathcal{E}$, and for all $e_j \in U(M)$, calculate $y_j = \mathcal{R}(s_j, a_j) + \gamma \hat{Q}_{net}\left(s_{j+1}, \arg\max_{a_{j+1} \in \mathcal{A}} Q_{net}(s_{j+1}, a_{j+1}; \theta_\phi); \theta_\phi^-\right)$.
    Execute a gradient descent step on $\left(y_j - Q_{net}(s_j, a_j; \theta_\phi)\right)^2$ with respect to θ_ϕ.
    Replace $\hat{Q}_{net} \leftarrow Q_{net}$ every C time steps.
    Go to next episode if $t = T$.
  end for
end for

an immediate reward. Then, the experience is stored in the $\mathcal{E}$ for training in the next episodes. During the learning process, the control agent takes a minibatch of experience at random to update the Q_{net} by minimizing the following loss function.

$$L_\phi(\theta_\phi) = \mathbb{E}_{e_j \in U(M)}[(\mathcal{R}(s_j, a_j) + \gamma \hat{Q}_{net}(s_{j+1}, \arg\max_{a_{j+1}} Q_{net}(s_{j+1}, a_{j+1}; \theta_\phi); \theta_\phi^-) - Q_{net}(s_j, a_j; \theta_\phi))^2], \quad (12)$$

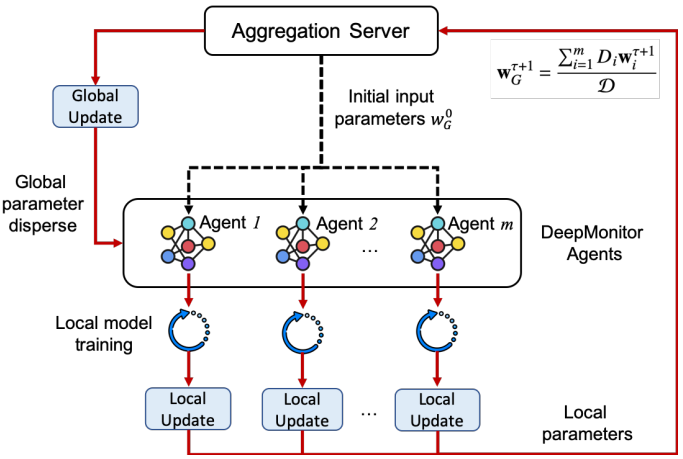


Fig. 6. Overall workflow of the federated deep reinforcement learning solution.

where γ is the discount factor, θ_ϕ is the weight parameters of Q_{net}, and θ_ϕ^- is the weight parameters of $\hat{Q}_{net}$.

A primary innovation in [26] is to freeze the parameters of $\hat{Q}_{net}$ for C time steps while updating Q_{net} via stochastic gradient descent, which enhances the stability of the algorithm. Likewise, we differentiate the loss function in (12) with respect to the weight parameters of Q_{net} and $\hat{Q}_{net}$; then, we can derive the gradient update as follows:

$$\nabla_{\theta_\phi} L_\phi(\theta_\phi) = \mathbb{E}_{e_j \in U(M)}[(\mathcal{R}(s_j, a_j) + \gamma \hat{Q}_{net}(s_{j+1}, \arg\max_{a_{j+1}} Q_{net}(s_{j+1}, a_{j+1}; \theta_\phi); \theta_\phi^-) - Q_{net}(s_j, a_j; \theta_\phi))\nabla_{\theta_\phi} Q_{net}(s_j, a_j; \theta_\phi)]. \quad (13)$$

Instead of calculating the expectation in (13), the loss function in (12) can be minimized by the stochastic gradient descent algorithm [44]. Furthermore, the learning of the DDQN algorithm continues by updating the neural network parameters using prior experience in an online manner.

V. Federated Deep Reinforcement Learning-Based IoT Traffic Monitoring

As stated in [26], [27], DDQN training from scratch usually requires a large amount of data and considerable computing resources to find the optimal policy efficiently. In addition, sensitive data can be endangered, e.g., industrial deployments, if the data set is shared among DDQN agents, and significant effort is required to transfer shared data sets among agents due to the communication costs [45]. Furthermore, if there is a newly deployed edge in the IoT network, it can significantly benefit from knowledge transferred from other edge nodes. Therefore, we propose a federated deep reinforcement learning solution to enhance the learning performance of the DDQN algorithm at DeepMonitor agents. Specifically, federated agents collaboratively learn a shared predictive model without sharing the training data set through many learning rounds.

A. Federated Deep Reinforcement Learning-Based IoT Traffic Monitoring Algorithm

The workflow of the federated deep reinforcement learning is shown in Fig. 6. In each learning round, the aggregation server sends a global model (global update) to all DeepMonitor agents, and in the first learning round, the initial global parameters w_G^0 of the DDQN algorithm, e.g., the number of epochs of parallel training and neural network weights (θ^- and θ from Algorithm 2), are dispersed together. When each DeepMonitor agent receives the updated global model, it trains this model (local model training) based on its local obtained experience. After a period of time, the agent uploads the new trained model (local update) to the aggregation server and then obtains an updated global model for the next learning round. This process is repeated until the global loss function from the local updates is minimized [45].

Algorithm 3 Federated Deep Reinforcement Learning-Based IoT Traffic Monitoring Algorithm

1: m is the number of participating DeepMonitor agents.
2: $\mathcal{T}$ is the number of federated learning rounds.
3: $\mathbf{w}_G^0$ is the initial global parameters, and the learning rate is η.
4: Aggregation Server disperses $\mathbf{w}_G^0$ to m agents.
5: **for** $\tau \in \{1,2,...,\mathcal{T}\}$ **do**
6: **for** $i \in \{1,2,...,m\}$ **do**
7: Agent i computes its local update.
8: Set $\mathbf{w}_i^{\tau+1} = \mathbf{w}_i^{\tau} - \eta \nabla \mathcal{L}_i(\mathbf{w}_i^{\tau})$.
9: Send $\mathbf{w}_i^{\tau+1}$ to Aggregation Server.
10: **end for**
11: Aggregation Server receives $\mathbf{w}_i^{\tau+1}$ from each agent i.
12: Update global model using

$$\mathbf{w}_G^{\tau+1} = \frac{\sum_{i=1}^{m} D_i \mathbf{w}_i^{\tau+1}}{\mathcal{D}}. \tag{14}$$

13: Disperse $\mathbf{w}_G^{\tau+1}$ to all agents.
14: **end for**

The proposed federated deep reinforcement learning scheme for traffic monitoring is formally described as follows. We consider a traffic monitoring system consisting of an aggregation server and m DeepMonitor agents with a set of local data sets $\mathcal{D} = \{D_1, D_2, ..., D_i, ..., D_m\}$. Each D_i contains a set of experiences e_t (see Experience replay mechanism in Section IV-B). For a DeepMonitor agent i in round τ in the federated learning process, the aim is to find the objective parameter $\mathbf{w}_i^{\tau}$ that minimizes the loss function

$$\mathcal{L}_i(\mathbf{w}_i^{\tau}) = L_i(\theta_i), \tag{15}$$

where $L_i(\theta_i)$ is the loss function derived from (12), θ_i is the current weight parameters of the Q_{net}, and the local problem at agent i is defined as

$$\mathbf{w}_i^{\tau} = \arg\min_{\mathbf{w}_i^{\tau}} \mathcal{L}_i\left(\mathbf{w}_i^{\tau}\right). \tag{16}$$

Then, the global loss function is formulated as

$$\mathcal{L}(\mathbf{w}_G^{\tau}) = \frac{\sum_{i=1}^{m} D_i \mathcal{L}_i(\mathbf{w}_i^{\tau})}{\mathcal{D}}, \tag{17}$$

where $\mathcal{D} = \sum_{i=1}^{m} D_i$ and the learning problem is now to find

$$\mathbf{w}^* = \arg\min_{\mathbf{w}_G^{\tau}} \mathcal{L}(\mathbf{w}_G^{\tau}). \tag{18}$$

Due to the complexity of the local problem in (16), we adopt a gradient descent algorithm [46] to obtain the solution. Specifically, for round $\tau > 0$, each DeepMonitor agent i calculates the parameters $\mathbf{w}_i^{\tau+1}$ using the update rule as follows

$$\mathbf{w}_i^{\tau+1} = \mathbf{w}_i^{\tau} - \eta \nabla \mathcal{L}_i(\mathbf{w}_i^{\tau}), \tag{19}$$

where $\eta \geq 0$ represents the learning rate. Then, the global parameters $\mathbf{w}_G^{\tau+1}$ are updated at the aggregation server as

$$\mathbf{w}_G^{\tau+1} = \frac{\sum_{i=1}^{m} D_i \mathbf{w}_i^{\tau+1}}{\mathcal{D}}. \tag{20}$$

The updated global parameters $\mathbf{w}_G^{\tau+1}$ are used for the next round of training at the DeepMonitor agents. Algorithm 3 provides details of the proposed federated deep reinforcement learning-based solution used in the DeepMonitor framework.

B. Convergence Analysis

To evaluate the convergence performance of Algorithm 3, we can make the following assumptions considering the loss function, which is similar to [47].

Assumption 1. *For all participants i:*
1) $\mathcal{L}_i(\boldsymbol{w})$ *is convex;*
2) $\mathcal{L}_i(\boldsymbol{w})$ *is L-smooth; that is,* $\mathcal{L}_i(\boldsymbol{w}') \leq \mathcal{L}_i(\boldsymbol{w}) + \nabla \mathcal{L}_i(\boldsymbol{w}) \cdot (\boldsymbol{w}' - \boldsymbol{w}) + \frac{L}{2} \| \boldsymbol{w}' - \boldsymbol{w} \|^2$.

According to [47], Assumption 1 assures the utility of linear regression and the rules of updates in federated learning; hence, we have the following lemma.

Lemma 1. $\mathcal{L}(\boldsymbol{w})$ *is convex and L-smooth.*

Proof. Straightforwardly from Assumption 1, in line with the definition of convex, $\mathcal{L}(\mathbf{w})$ is the finite-sum formation of $\mathcal{L}_i(\mathbf{w})$, and it is a triangle inequality. □

Theorem 1. *Given that* $\mathcal{L}(\boldsymbol{w})$ *is L-smooth and* μ*-strong, let* $\mu_\tau = 1/L$ *and* $\boldsymbol{w}_* = \arg\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$*, where* $\boldsymbol{w}_*$ *is the optimal solution; therefore, we have*

$$\| \boldsymbol{w}_\tau - \boldsymbol{w}_* \| \leq \left(1 - \frac{\mu}{L}\right)^\tau \| \boldsymbol{w}_1 - \boldsymbol{w}_* \|, \tag{21}$$

where $O(\varpi) = \frac{L}{\mu}\log(\frac{\|\boldsymbol{w}_1 - \boldsymbol{w}_*\|}{\varpi})$ *is referred to as the gradient dispersion, which represents how the parameters* $\boldsymbol{w}_G$ *are dispersed to each participant.*

Proof. Please see Appendix A. □

We can derive the convergence in expectation as follows

$$\left[\mathcal{L}(\mathbf{w}_\tau) - \mathcal{L}(\mathbf{w}_*)\right] \leq \varpi^\tau \left[\Delta^\tau\left(\mathcal{L}(\mathbf{w}_*)\right)\right]. \tag{22}$$

Therefore, $\mathcal{L}(\mathbf{w})$ is proved to be bounded as $\Delta^\tau\left(\mathcal{L}(\mathbf{w}_*)\right) = \mathcal{L}(\mathbf{w}_1) - \mathcal{L}(\mathbf{w}_*)$. In other words, our proposed federated learning approach can converge after a certain number of learning rounds.

C. Computational Complexity Analysis

In this study, we implement a deep reinforcement learning algorithm, i.e., DDQN, that applies deep neural networks to solve our optimization problem at each SDN-based IoT edge device. Nevertheless, substantial CPU and GPU resources are generally needed to operate deep learning-based algorithms, especially for complex problems with many constraints that usually require multiple hidden neural network layers. Therefore, the computational complexity should be considered to guarantee the practical deployment of the proposed algorithms. Specifically, the computational complexity of Algorithm 1 is bounded by $O\left(\phi_{max}T(|\mathcal{S}|^2 \times |\mathcal{A}|)\right)$. According to [48], the computational complexity of a deep neural network at each time step in the training process is calculated as $O(S_{in}N_l + \sum_{l=1}^{\zeta-1} N_l N_{l+1})$, where ζ, S_{in} and N_l represent the number of training layers, the input layer size, and the number of neurons in the l-th layer, respectively. Moreover, for T time steps in each training episode and ϕ_{max} episodes until convergence, the total computational complexity of a deep neural network in the training phase is expressed as $O(\phi_{max}T(S_{in}N_l + \sum_{l=1}^{\zeta-1} N_l N_{l+1}))$. In addition, with a replay memory buffer size N, the computational complexity of a classic DDQN algorithm is formulated as $O(2|\mathcal{S}|^2 \times |\mathcal{A}| + \log_2 \mathrm{N})$ [27], [48]. Thus, the total computational complexity of Algorithm 2 is $O\left(\phi_{max}T(S_{in}N_l + \sum_{l=1}^{\zeta-1} N_l N_{l+1}) + 2|\mathcal{S}|^2 \times |\mathcal{A}| + \log_2 \mathrm{N}\right)$. For each learning round in Algorithm 3, an agent i computes its local update on a data set D_i with computational complexity $O\left(|D_i|(S_{in}N_l + \sum_{l=1}^{\zeta-1} N_l N_{l+1}) + 2|D_i|^2 + \log_2 \mathrm{N}\right)$. Hence, the total computational complexity of Algorithm 3 is $O\left(\mathcal{T}(m|\mathbf{w}_i|(|D_i|(S_{in}N_l + \sum_{l=1}^{\zeta-1} N_l N_{l+1}) + 2|D_i|^2 + \log_2 \mathrm{N}) + |\mathbf{w}_G|)\right)$.

Recently, the authors in [49] proposed a novel method, namely, network compression, which can reduce the CPU and storage requirements of neural networks by a factor of 10. In addition, the obtained results show that the deep learning algorithms can be efficiently deployed on standard platforms from Intel, Qualcomm, and NVidia, without requiring special hardware or software. In this work, Algorithm 2 and Algorithm 3 follow common settings for deep neural networks [26], [27], as discussed previously; therefore, our proposed DeepMonitor framework can easily be implemented in practice.

VI. Performance Evaluation

A. Experimental Setup

In this work, a federated deep reinforcement learning scheme for traffic monitoring is developed to improve the learning performance of DeepMonitor agents in a decentralized manner. Therefore, for the DeepMonitor framework evaluation, we utilize MaxiNet to emulate a distributed SDN-based IoT network consisting of three hierarchical enterprises. Each network contains six access OvSes (Open vSwitch) running as SDN-based IoT edge devices and several container-based hosts (24 hosts/OvS) as IoT devices, and it is under the control of an ONOS SDN controller (v.1.3) [36]. The emulation runs on three machines with Intel(R) Core(TM) i7-7700 CPUs with a clock speed of 3.60 GHz, 64 GB RAM memory, and NVIDIA GeForce GTX 1080 Ti. The aggregation server is on a separate physical machine connected to three enterprise networks.

B. Parameter Setting

Concerning the configurations of the DDQN algorithm, we apply the common settings of neural networks [26]. In particular, two fully connected hidden layers are implemented with input and output layers (see Fig. 5), the size of the output layer is 10 (which indicates 10 flow rule match-field strategies), and the activation function is rectified linear unit (ReLU). The mini-batch size is 32, the replay memory $\mathcal{E}$ holds a size N of 10000, and the target Q-network replacement frequency C is 1000 iterations. In the learning process of the Q-learning and DDQN algorithms, the ϵ-greedy algorithm is employed with an initial value of 1.0 and final value of 0.1 [26], [50], the duration of a local learning iteration t is 10 seconds, and the maximum number of iterations in an episode T is 100. For the configurations of the federated DDQN, the number of participants m is 18, the learning rate η is 0.6, and the number of federated learning rounds $\mathcal{T}$ is 200. Furthermore, we conduct experiments utilizing TensorFlow Federated [51] to evaluate the DeepMonitor system under various parameter settings, as mentioned above.

Similar to [22], the number of flow rules that an OvS can handle in an emulation environment is set to 3000, and if there are more incoming packets that request new flow rule installations, e.g., a DoS/DDoS attack, while the buffer is fully occupied, the OvS's operation can become suspended after a short time period. Hence, we set $\mathcal{G}_{max} = 3000$. Note that we perform experiments with 10 different flow rule match-field combinations and 11 standard match-fields provided by the ONOS controller [36], so $|\mathcal{X}| = 10$ and $\mathcal{M}_{max} = 11$.

Although various IoT traffic groups exist, we previously discussed the traffic groups from the perspective of local networks [40] in Section III; thus, we can generalize three common traffic groups:

- *Sensor traffic*: IoT sensor devices produce traffic in a certain period with a low number of packets per flow.
- *Monitor traffic*: This traffic group is regularly recognized as real-time IoT applications, which can be distinguished by a small number of flows and a considerable number of packets per flow.

- *Alarm traffic*: This traffic group is not plainly defined because alarm IoT devices generate traffic flows only when an unusual event occurs. Consequently, we assume that this traffic group has moderate values of both the number of flows and packets per flow.

Moreover, Hping3 tool [52] is installed at the container-based hosts to randomly generate TCP/UDP traffic flows according to the characteristics of these traffic groups, thereby generating a highly dynamic traffic load in the emulation.

The above classification indicates that $\mathcal{N} = \{sensor, monitor, alarm\}$, and experiments are performed with three settings of required granularity levels, as follows:

- *Medium*: $\Theta_{sensor} = \Theta_{monitor} = \Theta_{alarm} = 5.0$.
- *Diverse*: $\Theta_{sensor} = 3.0$, $\Theta_{monitor} = 6.0$, $\Theta_{alarm} = 9.0$.
- *High*: $\Theta_{sensor} = \Theta_{monitor} = \Theta_{alarm} = 9.0$.

In addition, the importance of individual traffic groups is set equally, i.e., $w_{sensor} = w_{monitor} = w_{alarm} = 1/3$.

C. Benchmark Solutions

Regarding the provision of traffic monitoring capability in SDN-based IoT networks, the DeepMonitor framework's performance is evaluated by comparison with a recent solution, i.e., FlowStat [20], and the typical flow matching by ONOS [36]. In FlowStat, the authors present a flow-rule placement solution to select an SDN switch in the traffic flow path to place an exact-match flow rule that contains all available match-fields. Meanwhile, the source IP address and destination IP address are included in the match-fields of the flow rules at switches that are not selected. Accordingly, by observing at an SDN switch, the FlowStat forms two types of match-field combinations in flow rules, i.e., exact-match and (source IP address, destination IP address). For ease of understanding, we assume that the SDN controller randomly selects and installs these two match-field combinations in the match-fields of the flow rules in an SDN-based IoT edge. The default flow matching strategy provided by ONOS contains six match-fields in a flow rule, i.e., source and destination MAC addresses, source and destination IP addresses, and source and destination layer-4 ports. In contrast to these existing approaches, each DeepMonitor agent considers the current state of an SDN-based IoT edge on the basis of six parameters (as shown in (3)) with our settings above, and it can flexibly switch between 10 match-field combinations to maximize the traffic granularity while avoiding the overflow at the flow-tables of the edge.

D. Performance Analysis

1) Convergence of reinforcement learning algorithms: We first present the learning process of reinforcement learning algorithms at a particular DeepMonitor agent (Agent1) applying the standalone[6] Q-learning, the standalone DDQN, and our proposed federated DDQN. As illustrated in Fig. 7, the average immediate reward value for every 1,000 iterations is calculated during the training period of Agent1 that supervises edge OvS1. Clearly, the convergence performance of the federated DDQN is better than that of the standalone DDQN and significantly better than that of the standalone Q-learning algorithm. In particular, with the *Medium* setting, the federated DDQN algorithm requires only 30,000 iterations to obtain the average reward value of 8.0; meanwhile, it requires approximately 120,000 iterations to achieve the same performance in the case of the standalone DDQN. Moreover, the federated DDQN algorithm needs only around 40,000 and 50,000 iterations to achieve that average value in the *Diverse* and *High* settings, respectively. A higher required degree of traffic granularity demands actions that provide a higher number of match-fields in a flow rule. As a result, this scenario leads to a higher probability of overflowing the edge (i.e., more flow rules installed) and to a severer punishment for the control agent w.r.t. the immediate reward. Both the federated and the standalone DDQN algorithms converge and achieve a similar average reward at the end of training.

On the contrary, as shown in Fig. 7, the standalone Q-learning algorithm could not achieve the optimal policy compared to that obtained by the federated and the standalone DDQN algorithms for all settings. The MDP model under consideration has billions of states (as can be observed in (3)); thus, after two hundred thousand iterations, the Q-learning algorithm cannot attain results as good as those obtained by the DDQN algorithms. Especially in the case of the *High* setting (see Fig. 7 (c)), the Q-learning-based control agent is more likely to overflow the edge (the maximum number of flow rules is reached but the edge remains operational) to achieve the higher required granularity level, producing a lower range of the average immediate reward value compared with that in the *Medium* and *Diverse* settings.

In summary, by applying the federated DDQN algorithm, a DeepMonitor agent can quickly obtain the optimal flow rule match-field control policy to provide the maximum traffic granularity degree for all traffic groups, in comparison with the standalone DDQN and Q-learning.

2) Federated deep reinforcement learning performance: Next, we evaluate the learning performance of the DDQN algorithm at DeepMonitor agents by comparing the proposed federated learning-based solution with the traditional approach, i.e., a centralized DDQN (the algorithm is trained by data sets collected from all agents). Fig. 8 (a) shows the average value of the loss function in (12) obtained for every 1,000 iterations (or a federated learning round) with the high required granularity setting. Specifically, the federated DDQN solution reduces the learning loss by approximately 66% on average in the first 70 learning rounds and then achieves the same loss values as the centralized DDQN approach. In terms of the learning convergence, Fig. 8 (b) shows the average reward obtained from the learning algorithm by two different learning approaches. In particular, the federated DDQN algorithm can reduce the number of iterations required to obtain the optimal policy by 20,000 iterations (approximately 40%) compared to that of the centralized DDQN. This improvement results from the individual training with local data sets at agents and the global model update in the federated learning-based method. In other words, this enhancement indicates that our proposed federated learning solution in the DeepMonitor framework can

[6]The learning algorithm is trained with data sets collected by the Agent1.

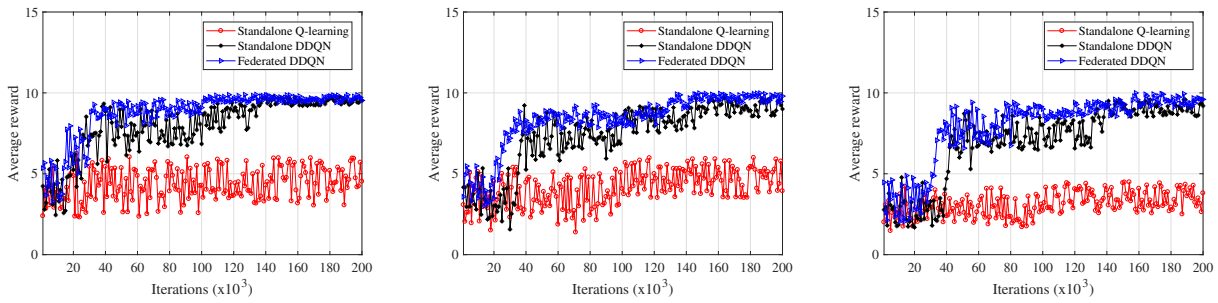


Fig. 7. Average reward of DeepMonitor Agent1 during the training phase. (a) Medium, (b) Diverse and (c) High required granularity levels.

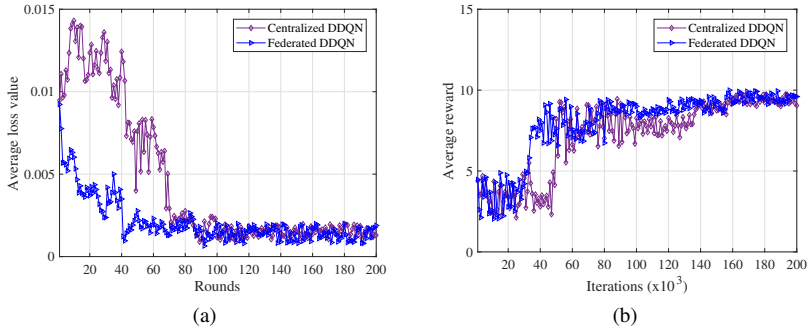


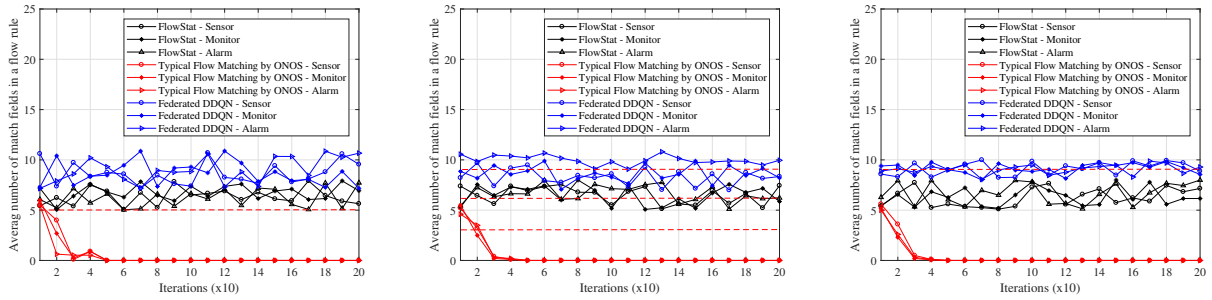
Fig. 8. (a) Average loss value and (b) average immediate reward derived from the learning algorithm by two different learning approaches.

efficiently improve the learning performance of the DDQN algorithm at the agents considering the learning stability and convergence with much lower learning loss.

3) Reliable provision performance of traffic granularity: Next, we evaluate our proposed solution by considering the degree of traffic granularity between three traffic groups in the testing phase at DeepMonitor Agent1 (the federated DDQN algorithm already converges). As demonstrated in Fig. 9 (a), (b) and (c), all traffic groups applying the federated DDQN-based solution outperform those of FlowStat and of typical flow matching by ONOS considering the average number of match-fields in a flow rule (extracted every 10 iterations). Specifically, for the *Medium* and *Diverse* settings, the federated DDQN-based solution allows flow rules in the flow-tables of the edge to carry a considerable number of match-fields that is approximately 37% higher than that of FlowStat, and these levels fully satisfy the prerequisites (red dashed lines). This is because the federated DDQN-based control agent is likely to change between many flow rule match-field combinations, which can provide a higher number of match-fields while preventing the flow-table overflow problem. Meanwhile, the FlowStat mechanism performs changes in only two flow rule match-field schemes, as mentioned earlier, leading to a lower average number match-fields in a flow rule during the testing.

For the *High* case, due to a very high precondition of granularity (Θ=9.0), the federated DDQN-based control agent tries to maximize the actual granularity for all traffic groups equally. Therefore, it cannot guarantee that the requirements are satisfied all the time because of highly dynamic traffic behavior, but overall, it maintains a significant total degree of the gained granularity, which is increased by roughly 41.9% compared with that obtained by FlowStat. For the typical flow matching by ONOS, the flow-tables of the OvS become overflowed after a few iterations due to the highly dynamic incoming traffic, thereby suspending the operation of the OvS and being unable to provide the traffic flow statistics measurement. To sum up, the DeepMonitor outperforms FlowStat and typical flow matching by ONOS in providing fine-grained traffic monitoring capability for all IoT traffic groups.

4) Flow-table overflow avoidance at the edges: As aforementioned, the flow-table overflow avoidance ability is considered an essential criterion of the DeepMonitor framework. Thus, we observe the flow-table overflow events caused by three different solutions during the testing phase under the *High* setting. Through the ONOS terminal, an error message, i.e., $flow\_mod\_failed$ [13], stemming from any of the SDN-based IoT edges indicates a table-full event at the edge. The obtained results show that no overflow problems are observed at all edges when applying the federated DDQN-based control agent because the federated DDQN algorithm can effectively find the optimal flow rule match-field policy depending on the current state of an edge concerning the actual traffic granularity of different traffic groups. Similarly, by randomly changing between the two flow rule match-field policies mentioned earlier, FlowStat can avoid flow-table overflows at all edges. Meanwhile, the default behavior of the ONOS controller makes the edges become overflowed after a few iterations if the incoming packet rate is 300



(a) Medium (b) Diverse (c) High

Fig. 9. Average number of match-fields in a flow rule of DeepMonitor Agent1 during the testing phase with different required granularity levels.

unique packets/second for more than 10 seconds, and the edge operation is suspended if this packet rate continues for a longer time (see Fig. 9). Consequently, the federated DDQN-based solution can completely circumvent the flow-table overflow at the SDN-based IoT edges and is superior to the standard flow matching by ONOS.

5) Impact on network performance: As discussed earlier in Section III, changes in the flow rule match-field strategies can have an additional impact on the network controller and forwarding performance, i.e., a higher number of packet_in messages and significant packet forwarding latency; hence, we observe these two metrics to evaluate the impact on the network performance produced by different mechanisms with different settings of Θ.

Fig. 10 presents the average number of packet_in messages arriving at the ONOS controller for three different required granularity levels, i.e., Medium, Diverse and High, extracted every 10 iterations from the OvS1 during the testing period. Concretely, for all three Θ settings, the federated DDQN-based control agent can maintain an acceptable packet_in rate that is slightly higher than that in FlowStat; however, it maximizes the actual traffic granularity levels of three groups (illustrated in Fig. 9). The reason is that the federated DDQN algorithm implements the optimal policy that meets the Θ requirements and simultaneously maintains a fair number of packet_in messages arriving at the ONOS controller, thereby obtaining a rational number of installed flow rules at the flow-tables of OvS1. By contrast, in the case of the typical flow matching by ONOS, there are no packet_in messages sent to the control plane after a few iterations due to the overflow of the flow-tables at the OvS1, as discussed earlier, which then suspends the switch's operation, leading to a massive decrease in the number of packet_in messages for all Θ settings.

With respect to the average latency of packets between hosts, the results are measured for hosts within OvS1 and hosts between OvS1 and OvS18. As shown in Fig. 11 (a), the federated DDQN and the FlowStat solutions obtain a similar end-to-end delay between hosts in both scenarios. Specifically, for FlowStat, the change between only two flow rule match-field strategies, i.e., exact-match and (source IP address and destination IP address), reduces the communication time between hosts, on average, because the match-field combination containing only source IP and destination IP addresses does not require substantial time to perform the matching process, leading to less delay for incoming packets. As discussed earlier, the federated DDQN-based control agent maximizes all traffic groups' actual granularity by putting more match-fields in a flow rule, thereby causing a lower delay than that of FlowStat. Notably, the typical ONOS flow rule match-field approach causes the flow-table overflow problem and OvS operation suspension; therefore, the latency between hosts cannot be correctly measured.

In conclusion, the DeepMonitor framework can provide good network performance in terms of the packet_in rate at the SDN control plane and the communication latency between hosts/IoT devices.

6) Case study on DDoS attack detection: To show the improvements in the traffic analysis capability provided by the DeepMonitor framework, we evaluate the anomaly detection performance of the IDS application based on Self-Organizing Map[7] [53] for a common DDoS attack, i.e., TCP SYN flood[8]. To evaluate the attack detection performance, we employ the following fitness function:

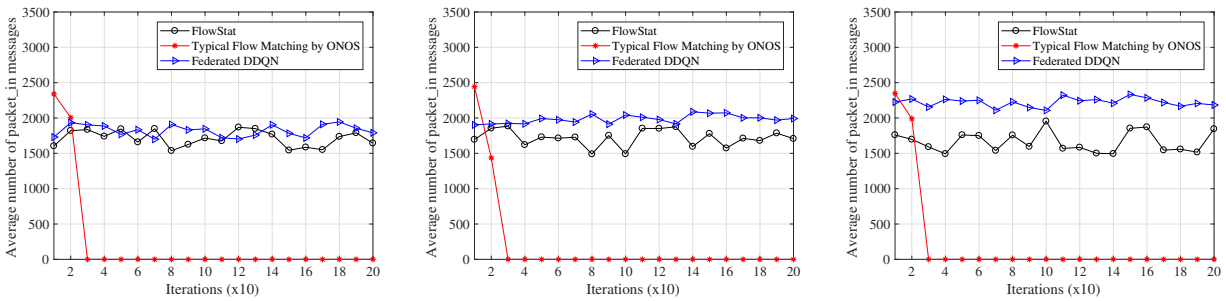
$$F_{DDoS} = W_{D_r} D_r + W_{A_c} A_c + W_{F_a} e^{-F_a}, \tag{23}$$

where D_r denotes the detection rate, A_c is the accuracy, F_a is the false alarm rate and W_{D_r}, W_{A_c} and W_{F_a} represent weight values that are equally set to 1/3. Note that a detection system generally aims to minimize the false alarm rate; thus, it is represented by e^{-F_a}, as shown in (23).

In case the network is under a DDoS attack, the flow-tables of the SDN-based IoT edge will be flooded very quickly if the controller applies a flow rule match-field scheme with a high number of match-fields [22], eventually leading to a packet_in flooding attack to the controller [35] and likely suspension of the control plane operation, i.e., to overload the *Reactive Forwarding* application in the ONOS controller [36]. Similarly, in our experiments with a centralized SDN controller for each network, if no DDoS mitigation solutions

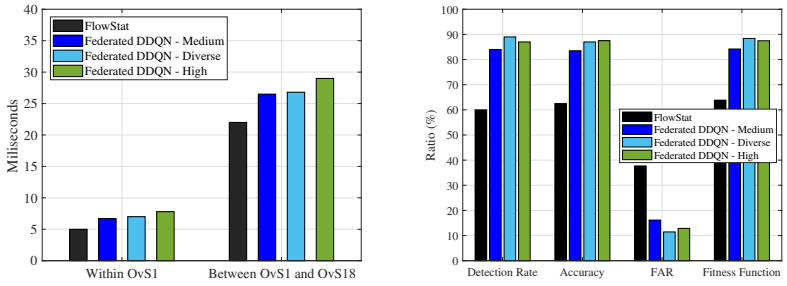
[7]The IDS application utilizes the Self-Organizing Map algorithm, which recognizes the traffic behavior patterns by means of a 6-tuple, as in [53], namely, the average number of packets per flow, average number of bytes per flow, average duration per flow, percentage of pair-flows, growth of single-flows and growth of different ports.

[8]Attackers attempt to send TCP segments as fast as possible with many spoofed source IP addresses and TCP ports to the Web servers, resulting in a huge amount of new flow rules in the SDN-based IoT edge in a short period.



(a) Medium (b) Diverse (c) High

Fig. 10. Average number of packet_in messages arriving at the ONOS controller from Agent1 with different required granularity levels.



(a) (b)

Fig. 11. (a) Latency between hosts in the data plane. (b) TCP SYN DDoS attack detection results.

are deployed, the ONOS controller starts receiving errors and exceptions [22], [35] promptly due to the enormous number of arriving packet_in messages.

Because the operation of the ONOS controller is suspended quickly if the typical flow matching by ONOS is applied, making the IDS application unable to collect traffic information from the SDN controller (via REST APIs [36]) and to detect attacks, no anomaly detection results are obtained. By contrast, as shown in Fig. 11 (b), the FlowStat and the federated DDQN-based solutions, frequently change between different flow rule match-field strategies according to the current network situation and can provide more detailed information, i.e., network traffic flow statistics. Additionally, the two solutions can protect the SDN controller from a packet_in flooding attack, thereby enabling the IDS application to recognize the attack presence. As discussed previously, because of the higher degree of granularity or statistical information, the federated DDQN-based solutions facilitate the IDS application to achieve DDoS attack detection performance that is 22.83%, on average, (derived from (23)) better than that in the case of FlowStat, as demonstrated in Fig. 11. In conclusion, by employing the DeepMonitor framework, the network control plane can provide efficient traffic monitoring capability to other applications, e.g., cyberattack detection, in SDN-based IoT networks.

VII. Conclusion

In this paper, we developed the DeepMonitor framework as an efficient traffic monitoring solution in SDN-based IoT networks, employing the advances of SDN and federated deep reinforcement learning. In particular, we introduced the DDQN-based flow rule match-field control algorithm for supervising a particular IoT edge to achieve efficient traffic monitoring performance, which is constrained by the required granularity levels of different traffic groups and by the maximum flow-table capacity of the edge device. Then, we proposed the federated deep reinforcement learning-based IoT traffic monitoring mechanism to improve the learning performance of the above DDQN algorithm. The results obtained from extensive experiments demonstrated that the proposed monitoring framework can provide a reliable traffic granularity level for all groups and significantly mitigate the flow-table overflow issue at the SDN-based IoT edges. This performance is increased by approximately 37% (for medium and diverse required granularity settings) and by 41.9% (for high required granularity setting) in comparison with that of FlowStat and far exceeds that achieved by the typical ONOS flow matching. In addition, the federated DDQN solution can reduce the learning loss by up to 66% and reduce the number of learning iterations required to achieve the optimal policy by 40% compared to the centralized mechanism in the case of the high granularity requirement. Moreover, for a case study on DDoS attack detection, the achieved results show that by applying the federated DDQN algorithm, the DeepMonitor framework can assist the IDS application to significantly improve the overall attack detection performance by 22.83% compared with the FlowStat solution.

APPENDIX A
PROOF OF THEOREM 1

First, we prove the μ-strong convexity of $\mathcal{L}(\mathbf{w})$. Given $\forall \mathbf{w}, \mathbf{w}^{'} \in \mathbb{R}, \beta \in [0,1]$, and we assume that $x := \mathbf{w} + \mathbf{w}^{'}$, $y := \beta\mathbf{w} + (1+\beta)\mathbf{w}^{'}$, and $\exists \beta_1$, $\beta_2 \in (0,1)$. According to the Taylor formula, we have

$$\mathcal{L}(\mathbf{w}) = \mathcal{L}(y) + \nabla\mathcal{L}(y)(\mathbf{w}-y) + \frac{1}{2}(\mathbf{w}-y)\nabla^2\mathcal{L}(\varrho_1)(\mathbf{w}-y), \tag{24}$$

and

$$\mathcal{L}(\mathbf{w}^{'}) = \mathcal{L}(y) + \nabla\mathcal{L}(y)(\mathbf{w}^{'}-y) + \frac{1}{2}(\mathbf{w}^{'}-y)\nabla^2\mathcal{L}(\varrho_2)(\mathbf{w}^{'}-y), \tag{25}$$

where $\varrho_1 := y + \beta_1(\mathbf{w}-y)$ and $\varrho_2 := y + \beta_2(\mathbf{w}^{'}-y)$. Then, we can incorporate the two above equations as follows:

$$\begin{aligned}\beta\mathcal{L}(\mathbf{w}) + (1-\beta)\mathcal{L}(\mathbf{w}^{'}) = \mathcal{L}(y) + \frac{1}{2}\beta(1-\beta)(\mathbf{w}-\mathbf{w}^{'})^2 \\ \times\left[(1-\beta)\nabla^2\mathcal{L}(\varrho_1) + \beta\nabla^2\mathcal{L}(\varrho_2)\right].\end{aligned} \tag{26}$$

Recall the definition of strong convexity; there is a constant μ^* satisfying

$$\frac{1}{2}(\mathbf{w}-\mathbf{w}^{'})\left[(1-\beta)\nabla^2\mathcal{L}(\varrho_1) + \beta\nabla^2\mathcal{L}(\varrho_2)\right] \geq \mu^* \parallel \mathbf{w}-\mathbf{w}^{'} \parallel . \tag{27}$$

Then, we can rewrite (26) as follows:

$$\begin{aligned}\beta\mathcal{L}(\mathbf{w}) + (1-\beta)\mathcal{L}(\mathbf{w}^{'}) &= \mathcal{L}(y) + \frac{1}{2}\beta(1-\beta)(\mathbf{w}-\mathbf{w}^{'})^2 \\ &\times\left[(1-\beta)\nabla^2\mathcal{L}(\varrho_1) + \beta\nabla^2\mathcal{L}(\varrho_2)\right] \\ &\geq \mathcal{L}(y) + \mu^*\beta(1-\beta) \parallel \mathbf{w}-\mathbf{w}^{'} \parallel^2,\end{aligned} \tag{28}$$

where $\mu^* = \frac{\mu}{2}$. Then, inserting $y := \beta\mathbf{w} + (1+\beta)\mathbf{w}^{'}$ into (28) proves the μ-strong convexity of $\mathcal{L}(\mathbf{w})$.

Applying the μ-strong convexity of $\mathcal{L}(\mathbf{w})$, we have

$$\nabla\mathcal{L}(\mathbf{w})(\mathbf{w}-\mathbf{w}_*) \geq \mathcal{L}(\mathbf{w}) - \mathcal{L}(\mathbf{w}_*) + \frac{\mu}{2} \parallel \mathbf{w}-\mathbf{w}_* \parallel^2 . \tag{29}$$

Then, we can obtain the following:

$$\begin{aligned}\parallel \mathbf{w}_{\tau+1}-\mathbf{w}_* \parallel^2 &= \parallel \mathbf{w}_\tau - \eta\nabla\mathcal{L}(\mathbf{w}_\tau) - \mathbf{w}_* \parallel^2 \\ &= \parallel \mathbf{w}_\tau - \mathbf{w}_* \parallel^2 - 2\eta\nabla\mathcal{L}(\mathbf{w}_\tau)(\mathbf{w}_\tau - \mathbf{w}_*) \\ &+ \eta^2 \parallel \nabla\mathcal{L}(\mathbf{w}_\tau) \parallel^2 \\ &\leq \parallel \mathbf{w}_\tau - \mathbf{w}_* \parallel^2 - 2\eta\Big(\mathcal{L}(\mathbf{w}) - \mathcal{L}(\mathbf{w}_*) \\ &+ \frac{\mu}{2} \parallel \mathbf{w}_\tau - \mathbf{w}_* \parallel^2\Big) + \eta^2 \parallel \nabla\mathcal{L}(\mathbf{w}_\tau) \parallel^2 .\end{aligned} \tag{30}$$

By smoothing $\mathcal{L}(\mathbf{w})$, we can obtain the gradient bound as follows:

$$\begin{aligned}\mathcal{L}(\mathbf{w}_*) \leq \mathcal{L}\left(\mathbf{w} - \frac{1}{L}\nabla\mathcal{L}(\mathbf{w})\right) \leq \mathcal{L}(\mathbf{w}) - \parallel \nabla\mathcal{L}(\mathbf{w}) \parallel^2 \\ + \frac{1}{2L} \parallel \nabla\mathcal{L}(\mathbf{w}) \parallel^2 \leq \mathcal{L}(\mathbf{w}) - \frac{1}{2L} \parallel \nabla\mathcal{L}(\mathbf{w}) \parallel^2 .\end{aligned} \tag{31}$$

Finally, by incorporating (30), (31) is reconstructed as

$$\begin{aligned}\parallel \mathbf{w}_{\tau+1}-\mathbf{w}_* \parallel^2 &= \parallel \mathbf{w}_\tau - \eta\nabla\mathcal{L}(\mathbf{w}_\tau) - \mathbf{w}_* \parallel^2 \\ &\leq \parallel \mathbf{w}_\tau - \mathbf{w}_* \parallel^2 - \eta\mu \parallel \mathbf{w}_\tau - \mathbf{w}_* \parallel^2 \\ &+ 2\eta(\eta L - 1)(\mathcal{L}(\mathbf{w}) - \mathcal{L}(\mathbf{w}_*)) \\ &\leq \left(1 - \frac{\mu}{L}\right) \parallel \mathbf{w}_\tau - \mathbf{w}_* \parallel^2 \leq \left(1 - \frac{\mu}{L}\right) \parallel \Delta^*\mathbf{w} \parallel^2 .\end{aligned} \tag{32}$$

The proof of Theorem 1 is completed.

ACKNOWLEDGMENT

This research was supported by a grant from the National Science and Technology Development Agency (NSTDA), the Coordinating Center for Thai Government Science and Technology Scholarship Students (CSTS), Khon Kaen University, Thailand Science Research and Innovation (TSRI), and National Research Council of Thailand (NRCT) via International Research Network Program (IRN61W0006).

REFERENCES

[1] T. G. Nguyen, T. V. Phan, D. T. Hoang, T. N. Nguyen, and C. So-In, "Efficient sdn-based traffic monitoring in iot networks with double deep q-network," in *International Conference on Computational Data and Social Networks*, pp. 26–38, Springer, 2020.

[2] T. Qiu, N. Chen, K. Li, M. Atiquzzaman, and W. Zhao, "How can heterogeneous internet of things build our future: A survey," *IEEE Communications Surveys Tutorials*, vol. 20, pp. 2011–2027, Thirdquarter 2018.

[3] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, vol. 4, pp. 1125–1142, Oct. 2017.

[4] I. Butun, P. Österberg, and H. Song, "Security of the internet of things: Vulnerabilities, attacks, and countermeasures," *IEEE Communications Surveys Tutorials*, vol. 22, no. 1, pp. 616–644, Firstquarter 2020.

[5] Z. Doffman, "Cyberattacks on iot devices surge 300% in 2019, 'measured in billions', report claims." https://www.forbes.com, Oct. 2020.

[6] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, pp. 80–84, Jul. 2017.

[7] M. Michael, "Attack landscape h1 2019: Iot, smb traffic abound." https://blog.f-secure.com, Oct. 2020.

[8] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in internet-of-things," *IEEE Internet of Things Journal*, vol. 4, pp. 1250–1258, Oct. 2017.

[9] M. Stoyanova, Y. Nikoloudakis, S. Panagiotakis, E. Pallis, and E. K. Markakis, "A survey on the internet of things (iot) forensics: Challenges, approaches, and open issues," *IEEE Communications Surveys Tutorials*, vol. 22, no. 2, pp. 1191–1221, Secondquarter 2020.

[10] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge computing in industrial internet of things: Architecture, advances and challenges," *IEEE Communications Surveys Tutorials*, pp. 1–1, Fourthquarter 2020.

[11] A. Molina Zarca, J. B. Bernabe, R. Trapero, D. Rivera, J. Villalobos, A. Skarmeta, S. Bianchi, A. Zafeiropoulos, and P. Gouvas, "Security management architecture for nfv/sdn-aware iot systems," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8005–8020, Oct. 2019.

[12] B. A. A. Nunes, M. Mendonca, X. 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, Thirdquarter 2014.

[13] O. N. Foundation, "Openflow switch specification version 1.5.1." https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, May 2021.

[14] P. Tsai, C. Tsai, C. Hsu, and C. Yang, "Network monitoring in software-defined networking: A review," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3958–3969, Dec. 2018.

[15] S. Bera, S. Misra, and A. V. Vasilakos, "Software-defined networking for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, Dec. 2017.

[16] V. Balasubramanian, M. Aloqaily, and M. Reisslein, "An sdn architecture for time sensitive industrial iot," *Computer Networks*, vol. 186, p. 107739, 2021.

[17] P. Zhou, Y. Xie, B. Niu, L. Pu, Z. Xu, H. Jiang, and H. Huang, "Qoe-aware 3d video streaming via deep reinforcement learning in software defined networking enabled mobile edge computing," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2020.

[18] T. Wu, P. Zhou, B. Wang, A. Li, X. Tang, Z. Xu, K. Chen, and X. Ding, "Joint traffic control and multi-channel reassignment for core backbone network in sdn-iot: A multi-agent deep reinforcement learning approach," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2020.

[19] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–9, May 2014.

[20] S. Bera, S. Misra, and A. Jamalipour, "Flowstat: Adaptive flow-rule placement for per-flow statistics in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 530–539, Mar. 2019.

[21] C. Liu, A. Malboubi, and C. Chuah, "Openmeasure: Adaptive flow measurement and inference with online learning in sdn," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 47–52, Apr. 2016.

[22] T. V. Phan, S. T. Islam, T. G. Nguyen, and T. Bauschert, "Q-data: Enhanced traffic flow monitoring in software-defined networks applying q-learning," in *2019 15th International Conference on Network and Service Management (CNSM)*, pp. 1–9, Dec. 2019.

[23] J. Huang, Y. He, Q. Duan, Q. Yang, and W. Wang, "Admission control with flow aggregation for qos provisioning in software-defined network," in *2014 IEEE Global Communications Conference*, pp. 1182–1186, Dec. 2014.

[24] A. Mondal and S. Misra, "Flowman: Qos-aware dynamic data flow management in software-defined networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, pp. 1366–1373, Jul. 2020.

[25] M. Malboubi, L. Wang, C. Chuah, and P. Sharma, "Intelligent sdn based traffic (de)aggregation and measurement paradigm (istamp)," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pp. 934–942, May 2014.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[27] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, p. 2094–2100, AAAI Press, Feb. 2016.

[28] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3133–3174, Fourthquarter 2019.

[29] N. Van Huynh, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, ""jam me if you can:" defeating jammer with deep dueling neural network architecture and ambient backscattering augmented communications," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 11, pp. 2603–2620, Nov. 2019.

[30] N. Van Huynh, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, D. Niyato, and P. Wang, "Optimal and low-complexity dynamic spectrum access for rf-powered ambient backscatter system with online reinforcement learning," *IEEE Transactions on Communications*, vol. 67, no. 8, pp. 5736–5752, Aug. 2019.

[31] Z. Ding, L. Shen, H. Chen, F. Yan, and N. Ansari, "Energy-efficient relay-selection-based dynamic routing algorithm for iot-oriented software-defined wsns," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 9050–9065, 2020.

[32] T. V. Phan, S. Sultana, T. G. Nguyen, and T. Bauschert, "Q-transfer: A novel framework for efficient deep transfer learning in networking," in *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 146–151, 2020.

[33] Y. Huang, X. Guan, H. Chen, Y. Liang, S. Yuan, and T. Ohtsuki, "Risk assessment of private information inference for motion sensor embedded iot devices," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 3, pp. 265–275, 2020.

[34] T. V. Phan, T. G. Nguyen, N. N. Dao, T. T. Huong, N. H. Thanh, and T. Bauschert, "Deepguard: Efficient anomaly detection in sdn with fine-grained traffic flow monitoring," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1349–1362, Sept. 2020.

[35] 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, Jun. 2015.

[36] ONOS, "Description of the onos controller." www.onosproject.org, May 2021.

[37] X. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Rules placement problem in openflow networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, pp. 1273–1286, 2016.

[38] B. A. A. Nunes, M. Mendonca, X. 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, no. 3, pp. 1617–1634, Thirdquarter 2014.

[39] I. Farris, T. Taleb, Y. Khettab, and J. Song, "A survey on emerging sdn and nfv security mechanisms for iot systems," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 812–837, Firstquarter 2019.

[40] F. Alam, R. Mehmood, I. Katib, N. N. Albogami, and A. Albeshri, "Data fusion and iot for smart ubiquitous environments: A survey," *IEEE Access*, vol. 5, pp. 9533–9554, Apr. 2017.

[41] 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.

[42] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[43] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning.," in *ICLR*, Jan. 2016.

[44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[45] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y. C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys Tutorials*, vol. 22, no. 3, pp. 2031–2063, Thirdquarter 2020.

[46] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NIPS Workshop on Private Multi-Party Machine Learning*, Oct. 2016.

[47] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.

[48] H. Yang, Z. Xiong, J. Zhao, D. Niyato, L. Xiao, and Q. Wu, "Deep reinforcement learning based intelligent reflecting surface for secure wireless communications," *IEEE Transactions on Wireless Communications*, Sept. 2020.

[49] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, (Cambridge, MA, USA), p. 1135–1143, MIT Press, Dec. 2015.

[50] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, May 1992.

[51] TensorFlow, "Tensorflow federated: Machine learning on decentralized data." https://www.tensorflow.org/federated, Oct. 2020.

[52] Hping3, "Description of the hping3 tool." www.hping.org, May 2021.

[53] 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.