Mobile Networks and Applications
https://doi.org/10.1007/s11036-019-01402-8

Improved Flow Awareness Among Edge Nodes by Learning-Based Sampling in Software Defined Networks

Jun Deng[1] · He Cai[1] · Sheng Chen[1] · Jianji Ren[2] · Xiaofei Wang[1]

© Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract

To improve the specific quality of service, internal network management and security analysis in the future mobile network, accurate flow-awareness in the global network through packet sampling has been a viable solution. However, the current traffic measurement method with the five tuples cannot recognize the deep information of flows, and the Deep Packet Inspection (DPI) deployed at the gateways or access points is lack of traffic going through the internal nodes(e.g., base station, edge server). In this paper, by means of Deep Q-Network (DQN) and Software-Defined Networking (SDN) technique, we propose a flow-level sampling framework for edge devices in the Mobile Edge Computing (MEC) system. In the framework, an original learning-based sampling strategy considering the iterative influences of nodes is used for maximizing the long-term sampling accuracy of both mice and elephant flows. We present an approach to effectively collect traffic packets generated from base stations and edge servers in two steps: 1) adaptive node selection, and 2) dynamic sampling duration allocation by Deep Q-Learning. The results show that the approach can improve the sampling accuracy, especially for mice flows.

Keywords Edge computing · Deep Q-learning · SDN · Flow-awareness · Resource allocation

1 Introduction

With the increase of wireless access devices and traffic loads in the fifth Generation (5G) networks, many traffic flows will not pass through the backbone or gateway, but only exist among the base stations (BSs) and the edge servers, such as the flows in cooperative caching [1], content sharing [2] and intensive-computation offloading [3]. Therefore, global accurate flow awareness in Mobile Edge Computing (MEC) system is an urgent need for fine-grained quality of service guarantee [4, 5], security analysis, content awareness [6], internal network management and so on. These applications in MEC business model require more flows, with their deep information (e.g., security, application type, flow behavior) which can be brought by the payload of sampled packets. Especially for security of network edge devices, Disk Operating System (DOS) attacks could be packet-based with only a small amount packets and hide in short-life **mice flows** [7, 8], which makes themselves hard to capture.

It is challenging to deeply obtain flow information while sampling network-wise flows accurately under Deep Packet Inspection (DPI) technology [9] or statistics-based reports created by other measurements based on telemetry or sketch [10–13]. DPI is used to inspect packet payload at the gateways or access points, which is, however, hard to be deployed on all nodes [14]. This solution leads to a lack of the global detailed flow information. Meanwhile, statistics-based reports are short of payload and flow-level recognition of internal traffic. In addition, sFlow-based solutions can capture the packet with payload in a probability [15, 16], but the sampled packets of the same flow is non-consecutive,

✉ Xiaofei Wang
xiaofeiwang@tju.edu.cn

Jun Deng
jundeng@tju.edu.cn

He Cai
hecai@acm.org

Sheng Chen
chensheng@tju.edu.cn

Jianji Ren
renjianji@hpu.edu.cn

[1] Tianjin Key Laboratory of Advanced Networking, College of Intelligence and Computing, Tianjin University, Tianjin, China

[2] Henan Polytechnic University, Jiaozuo, China

Published online: 28 November 2019

which is useless for recognizing the flow-level deep information. Fortunately, Systematic Packet Sampling (SPS) that samples within a certain duration can capture consecutive packets [18, 19]. Combined with the SPS method, Software-Defined Networking (SDN) can achieve global and cyclical sampling, which is conducive to improving the flow-level sampling accuracy. Then the MEC concept along with SDN can enable more applications requiring computing and storage resources [20].

However, due to restrictions on the resources of MEC system, the selected sampling nodes should not be too many and the sampling duration cannot be assigned unlimitedly. The allocation of sampling duration is actually to allocate the bandwidth of collector reasonably for packet sampling brought by each sampling node. Hence, limited by the processing capacity of collector, the realization of global high-accuracy flow-level sampling is facing two challenges: 1) how to select the sampling nodes, 2) how to allocate the sampling duration.

Existing studies [18, 19, 21, 22] have proposed different methods to select sampling nodes or allocate resources, but they are limited to a certain extent. For example, study [18] selects K nodes with the most active flows currently passing through the node as the sampling nodes, and then assigns a equal sampling rate to the K nodes. However, the fixed K nodes do not guarantee to cover all the flows in the network, and the decision of sampling rate cannot change dynamically according to the current active flows covered by the nodes. In addition, study [21] selects sampling nodes based on a greedy strategy which can cover all the active flows, and it proposes an adaptive scheme to adjust the sampling rate considering the bandwidth resource. However, its algorithm results in a reduced accuracy.

Recently, Deep Reinforcement Learning (DRL), which is the combination of Reinforcement Learning (RL) and deep learning, has been used commonly to obtain the resource allocation policy for solving the complexity problem in real wireless environment. Studies [23–26] show that DRL has great potential in handling automatic resource allocation based on the time-varying requirements. For example, study [23] formulates the resource allocation strategy, in which state value takes the computation capabilities of MEC servers and the cache capacities of BSs into consideration.

In this paper, we focus on adaptively selecting sampling nodes and dynamically setting sampling strategies for each sampling period, aiming to improve network-wise flow-level sampling accuracy with SPS.

For this purpose, based on DRL and SDN, we propose a framework considering the iterative influences of global nodes and the dynamic allocation of sampling duration. The influence which is decided by the current active flows covered by each node can be used to measure the importance of a node and determine the sampling nodes. The allocation is decided by the centrally controlled learning-based strategy, which can allocate appropriate sampling duration to nodes, and finally lead to a great advantage in capturing mice flows. Our main contributions are summarized as follows:

- we propose a learning-based flow-level sampling strategy in SPS, which can cover all the active flows in the network and guarantee data integrity for training process of artificial neural network.
- we model flow-level sampling and the resource allocation as a Markov Decision Process (MDP) and propose a framework based on SDN and Deep Q-Network (DQN) for centrally controlled learning-based sampling strategy among BSs and edge servers.
- We evaluate the method by a relatively realistic topology and a real trace. Finally the results show that our learning-based method can improve the sampling accuracy by 5.9% and improve the accuracy of mice flows by 8.1%.

The remainder of this paper is organized as follow: we provide an overview of previous related work in Section 2. The system model and problem formulation will be discussed in Section 3. Then the learning-based sampling strategy and the resource allocation algorithm will be discussed in Section 4 in detail along with experiments in Section 5. Finally, conclusions are given in Section 6.

2 Related work

There exist many studies for the improvement of sampling accuracy or the resource allocation. Study [18] uses flow betweenness centrality (FBC) to quantify the node's influence, and then assigns a sampling rate to the K most influential nodes under the constraint of collector capacity. However, dynamic resource allocation is not considered. Besides, study [19] formulates an optimization problem to find an appropriate sampling rate for each sampling node. But the decision of sampling rate is affected only by suspicious traffic, not all active flows. Furthermore, in [16, 22] a new model is proposed for maximizing the sampling accuracy of elephant flows. However, both of them lose sight of the sampling of mice flows. Actually, in a real network environment mentioned by the above studies, it is difficult to achieve an appropriate resource allocation considering the traffic dynamics and various constraints. Hence, many studies have devoted to optimizing resource allocation based on DRL nowadays. The authors in study [26] propose

a cache-enabled framework, in which a central scheduler is responsible for collecting the channel state information from each candidate, and then the deep Q-network will derive the optimal policy for candidate selection and automatically allocate the limited backhaul capacity for these candidates. Its centrally controlled architecture is universal, which is similar to ours as Fig. 1. Moreover, the authors in study [25] propose a scheme for making BSs perform ON/OFF state to reduce power consumption or enhance energy harvesting. This is analogous to deciding whether the node is a sampling node or not.

3 System model and problem formulation

In this section, we first introduce the details of the proposed flow-awareness architecture. Then, we discuss the sampling model and objective in the real MEC system. After that, the optimization and specific of MDP model are presented, and the problem formulation of maximizing the objective is given at last.

3.1 Flow-awareness architecture based on SDN in MEC

The flow-awareness architecture based on SDN in MEC is illustrated in Fig. 1, which contains four components: MEC nodes, SDN controller, neural network and collector.

MEC nodes In MEC environment, users receive services by sending requests to a base station with a fixed radio coverage, where each base station is linked to only a local edge server [27]. In our flow-awareness architecture, any base station and edge server can be selected as a sampling node, which can be co-controlled by SDN controller. Besides, the data flows passing through MEC nodes consist of the local and remote traffics. Under the learning-based sampling strategy, the packets of these flows will be transferred to the collector.

SDN controller SDN is an innovation, with which the MEC system can realize centralized and efficient control [20]. The SDN controller can obtain the information from the overall data plane, and can set different strategy flexibly through installing forwarding rules. By the pattern of flow entry and group entry, the controller can precisely manage any node to start or to stop sampling. In our architecture, by means of a learning-based strategy, the SDN controller gathers the flow informations (e.g. flow number, node id) generated from MEC nodes and sends them to the neural network, after which the controller can receive a feedback about sampling duration allocation.

Neural network A Deep Neural Network (DNN) is based on the Multi-Layer Perceptron (MLP) structure, in which all adjustable parameters are jointly optimized by minimizing the loss function throughout the training process [17]. The

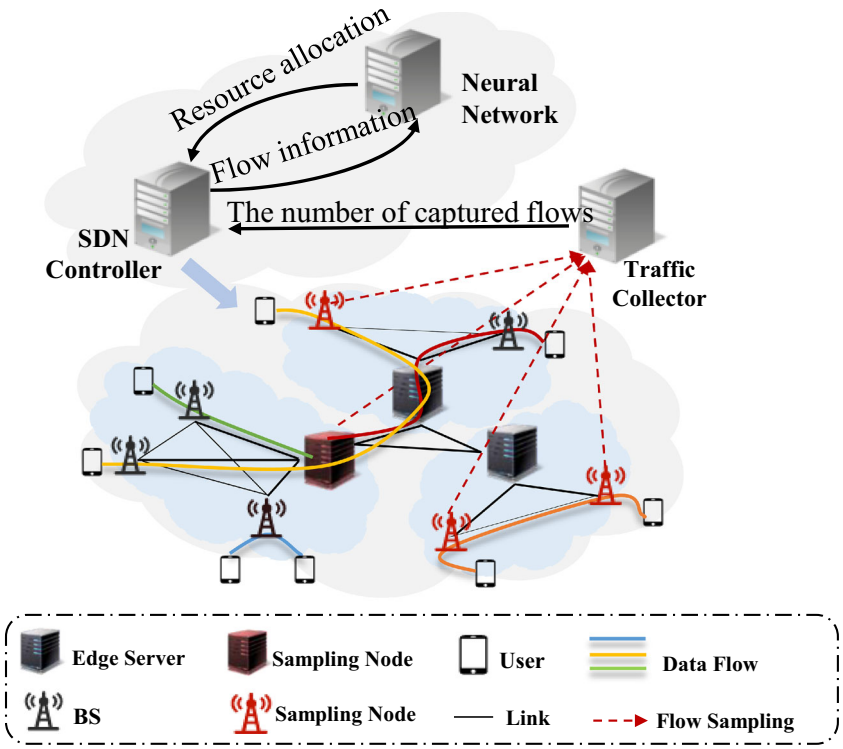


Fig. 1 Illustration of flow-awareness architecture in SDN

DRL algorithm can run in this neural network which is deployed in a single server shown in Fig. 1 and then solve complex problems in real-world environment. In our framework, the DNN system can receive the global flow informations from the controller, after which it will make an optimal policy decision to allocate sampling resources.

Collector A collector is just a traffic analyzer such as an intrusion detection system [18] or an application-aware traffic scheduler [28]. In Fig. 1, the traffic collector is responsible for collecting the sampled packets from the data plane and informing the controller of the number of captured flows.

3.2 Sampling model and objective

First, we quantify the influence and the cost of each node in the sampling period. And next, the objective is given in detail.

3.2.1 Concept of influence

We define a set $\mathbb{R}$, and $n = |\mathbb{R}|$ represents the number of MEC nodes. Suppose the set $F^c_{i,j}$ consists of the flows captured from the node R_i when its sampling duration is the whole j-th sampling period T, and F^t_j is total flows covered by all the nodes during T. So the influence I^j_i of R_i in the j-th period is quantified as

$$I^j_i = \frac{|F^c_{i,j}|}{|F^t_j|}, \tag{1}$$

which represents the strength of capturing flows from each node over T. Let $F^s_{i,j}$ include the sampled flows when R_i is allocated sampling duration $\widehat{t^j_i}$. $F^s_{i,j} = f(I^j_i, \widehat{t^j_i})$, where $f(I^j_i, \widehat{t^j_i})$ is a function to be formulated in the case of known flow distribution [30] and sampling duration. Hence, the non redundant flows sampled by global nodes F^s_j can be denoted as $F^s_j = \bigcup_{i=1}^n F^s_{i,j}$ within T, which is deduced as

$$F^s_j = \bigcup_{i=1}^{n} f(I^j_i, \widehat{t^j_i}). \tag{2}$$

3.2.2 Quantification of the cost

Cost w^j_i is the number of packets captured from R_i during $\widehat{t^j_i}$. v^j_i is the current packet speed (packets/T) during the j-th sampling period T. Thus w^j_i can be expressed as

$$w^j_i = \frac{v^j_i}{T/\widehat{t^j_i}}. \tag{3}$$

3.2.3 Objective

We formulate the influence-based sampling model in Eq. 4,with the constraints (5) and (6), where C is the capacity of collector (packets/T).

$$\max \bigcup_{i=1}^{n} f(I^j_i, \widehat{t^j_i}). \tag{4}$$

$$\text{s.t.} \quad \sum\nolimits_i^n w^j_i \cdot \widehat{t^j_i} \le C \cdot T, \tag{5}$$

$$0 \le \widehat{t^j_i} \le T. \tag{6}$$

The goal is to work out corresponding $\widehat{t^j_i}$ for each R_i to maximize the number of non redundant flows during j-th period under the constraint of C. Maximizing the value of Eq. 4 means capturing the most flows in every sampling period, which is equivalent to optimizing sampling accuracy.

However, the complexity for solving the problem is $O(2^n)$ considering whether a node is assigned sampling duration or not. Also, $f(I^j_i, \widehat{t^j_i})$ in Eq. 4 is usually difficult to formulated because of the imprecision and instability of predicting flow distribution in a real network environment [30]. Some studies try to model the packet arrival [31] or the flow arrival [32] as a Poisson process. Nonetheless, the lacking of **prior knowledge** (exact packet arrival or flow arrival) makes the solution not completely practical. As a result, before the sampling begins, the number of captured flows cannot be obtained without the above prior knowledge, which makes F^s_j impossible to quantify.

Optionally, the current active flows covered and the current packet speed are available, which can be used to quantify the influence and cost of the corresponding node. In this paper, we find a model-free reinforcement learning method to replace the traditional solution considering the state including the current flow numbers.

3.3 Specifics of MDP model

Before introducing problem formulation, we describe a brief review of reinforcement learning and Q-learning, and then the specifics of Markov Decision Process are presented according to the current available knowledge in our flow-awareness framework.

3.3.1 Reinforcement learning and Q-learning

Reinforcement learning can be extremely effective in handling the complex real-world situations, where an agent can gain reward and automatically learn to behave by interacting with the external environment and constantly trying to act. As one of the widely-used algorithm, Q-Learning is powerful in resource allocation because of its model-free and off-policy characteristics. In our flow-awareness framework, based on the structure of MDP, the flow-level sampling problem can be modeled as three components: state space, action space and reward function.

3.3.2 State space

We regard the whole of network nodes in the sampling environment as an agent. The sampling process is periodic, and state changes periodically with decision epoch. Therefore, for the agent in sampling environment, the j-th sampling period is equivalent to decision epoch. Let $q_i^j = |F_{i,j}^q|$ be the number of the flows currently covered by any node R_i, and each node is assigned a ratio of sampling duration r_i^j. Under the constraints of node's packet speed and collector's processing capacity, r_i^j is used to calculate the unit sampling duration, and then the sampling duration $\widehat{t_i^j}$ of each node is also calculated. As shown below, during the j-th period, the packet speed of R_i is v_i^j. So the unit sampling duration $\widehat{t^j}$ can be expressed as

$$\widehat{t^j} = \frac{\mathrm{C} \cdot T}{\sum_i^n \mathrm{r}_i^j \cdot \mathrm{v}_i^j}, \quad i, j \in \mathbb{Z}, \tag{7}$$

then,

$$\widehat{t_i^j} = r_i^j \cdot \widehat{t^j}. \tag{8}$$

Based on the above computing method, the state vector should include the vector of flow number q^j and the vector of sampling duration ratio r^j. Therefore, the observed state vector s_i during decision epoch j can be denoted as

$$\mathrm{s}_j = [q^j, \ r^j] \in S, \tag{9}$$

where S is the state space, and $q^j = [q_1^j, q_2^j, ..., q_n^j]$, $r^j = [r_1^j, r_2^j, ..., r_n^j]$, $q_i^j, r_i^j \in \mathbb{Z}$.

3.3.3 Action space

The goal of one action is to allocate resources to one of the nodes in one of the three ways : increase, decrease and keep unchanged. When the agent is in the state s_j, it will decide which node to execute the action $\phi(s_j)$, and then r_i^j of the node will be changed. So the action space A is expressed as

$$A = [a_j^1, \ a_j^{-1}, \ a_j^0], \tag{10}$$

where a_j^1, a_j^{-1}, a_j^0 are defined as follow:

1) The action vector $a_j^1 = [a_{j,1}^1, \ a_{j,2}^1, ..., \ a_{j,n}^1]$, where any element $a_{j,i}^1 = 1$ represents increasing r_i^j by one.
2) The action vector $a_j^{-1} = [a_{j,1}^{-1}, a_{j,2}^{-1}, ..., a_{j,n}^{-1}]$, where any element $a_{j,i}^{-1} = -1$ represents decreasing r_i^j by one.
3) The action vector $a_j^0 = [0]$ means that any r_i^j remains unchanged.

3.3.4 Reward function

In order to perfectly reflect the approval of the agents actions, the reward function needs to take into account the objectives of flow-awareness framework [33]. As Eq. 4 shows, the goal is to maximize the sampling accuracy of the whole sampling process by capturing the most flows in each sampling period. In order to achieve the goal through learning-based method, the above variable F_j^s and F_j^t are used again to calculate sampling accuracy ρ within the j-th period as follow:

$$\rho = \frac{|F_j^s|}{|F_j^t|}, \tag{11}$$

where F_j^s and F_j^t can be get at the end of each sampling period. Then, in our flow-awareness architecture, reward function $R(s_j, \phi(s_j))$ is defined as

$$R(s_j, \phi(s_j)) = e^{\rho}, \tag{12}$$

when the agent takes the action $\phi(s_j)$ upon the state s_j. The natural exponential function in Eq. 12 is monotonically increasing, which can guide the objective of maximizing the sampling accuracy.

3.4 Problem formulation

In a continuous sampling process, the allocation of resources will have an impact on future sampling results. Thus, the agent needs to consider not only the immediate rewards, but also the future rewards. Based on Eq. 12, taking into account the lone-term rewards, the expectation of maximizing objective throughout the whole sampling process upon the initial state $s_1 = s$ and an infinite time horizon is defined as

$$V(s) = E\left[\lim_{J \to \infty} \sum_{j=1}^{J} (\gamma)^{j-1} R(s_j, \phi(s_j)) | s_1 = s\right]. \tag{13}$$

In Eq. 13, $\gamma \in [0, 1)$ is the discount factor, which represents the preference for proximate rewards and make the reward show an exponential decay.

That is, the agent is expected to find an optimal control policy ϕ^* to get the optimal value function at the state s as follow:

$$V^*(s) = \max_{\phi} V(s). \tag{14}$$

The control policy ϕ^* follows the Bellmans optimality equation as Eq. 15, where $s_j = s$ is the current state over the epoch j, and s_{j+1}=s' is the next state after taking the optimal action $\phi(s_j) = \varphi$.

$$V^*(s) = \max_{\varphi} \left\{ R(s,\varphi) + \gamma \cdot \sum_{s'} \Pr\{s'|s,\varphi\} \cdot V^*(s') \right\}. \tag{15}$$

Besides, $R(s,\varphi)$ is the immediate reward and $\Pr\{s'|s,\varphi\}$ is the transition probability from s to s.

However, the traditional solutions to Eq. 15 need complete information or knowledge of the sampling process. Therefore, Q-Learning is introduced to overcome the lack of the future reward R and the probability Pr. In Q-Learning, a state-value function, namely Q-function, is defined as the right-hand side of Eq. 15, then we can get the maximum Q-function as follow:

$$Q(s,\varphi) = R(s,\varphi) + \gamma \cdot \sum_{s'} \Pr\{s'|s,\varphi\} \cdot V^*(s'). \tag{16}$$

After that, the optimal state function $V^*(s)$ can be denoted as

$$V^*(s) = \max_{\varphi} Q(s,\varphi). \tag{17}$$

Then Eq. 16 can be optimized as Eq. 18 by incorporating Eq. 17.

$$Q(s,\varphi) = R(s,\varphi) + \gamma \cdot \sum_{s'} \Pr\{s'|s,\varphi\} \cdot \max_{\varphi'} Q(s',\varphi'). \tag{18}$$

Finally, under the known state s, the current action φ and the immediate reward R, the agent will produce an optimal estimate of the future control policy by finding the appropriate Q-function after a continuous learning process. Thus, the Q-function is updated to an iterative form as

$$\begin{aligned} Q^{j+1}(s,\varphi) = {} & Q^j(s,\varphi) + \alpha \cdot [R(s,\varphi) \\ & + \gamma \max_{\varphi'} Q^j(s',\varphi') - Q^j(s,\varphi)], \end{aligned} \tag{19}$$

where $\alpha \in [0,1]$ is the learning rate. The Eq. 19 expresses that after the agent performs the current optimal action $\phi(s_j)$, the state s_j turns to the state s_{j+1} and the corresponding reward is fed back. Then, with a proper α, the objective Q-function will approach the convergence faster.

Q-Learning uses a Q-Table store the Q value of each state-action pair. However, in the real sampling environment, the state space and action space are huge. Thus, the approach based on Q-Learning will lead to a high-dimensional Q-Table, with which the learning process will be extremely slow. Here, a deep reinforcement learning method is used to solve this problem.

4 Sampling strategy based on DRL

In this section, we formulate the dynamic sampling duration allocation problem as a deep reinforcement learning process. Lastly, the description of iterative influence for adaptive node selection is given.

4.1 Double deep Q-learning for resource allocation

During the application of deep Q-learning, the state-action Q-function is replaced by a deep neural network, called Q-network. It has the possibility to generalize unseen states and narrow the state space, which is beneficial for accelerating the learning process. In the Q-network, the value in the Q-Table is replaced by multi-layer weight θ_j which will update after any policy decision j. Thus, the Q-function is changed as

$$Q(s,\varphi) = Q((s,\varphi);\theta_j). \tag{20}$$

In addition, at each training step, the corresponding state transition $\widehat{T}_j = (s_j, \phi(s_j), R(s_j,\phi(s_j)), s_{j+1})$ will be stored in an experience pool with a pre-defined capacity $\widehat{M}$. After the samples in the pool have accumulated to a certain number, the agent will sample the M batches (past experience) randomly from the pool for the reflective learning, which is what we called experience replay. These samples should be multidimensional and diverse, which can be guaranteed by an initial large exploration probability ε. Actually, the value of ε is gradually reduced from the initial ε_1 to the final ϵ.

In our model, a more reliable algorithm Double Deep Q-Network (Double DQN) [34] is used for training to achieve appropriate resource allocation, in which two different Q-network, known as main network $Q(s,\varphi;\theta_j)$ and target network $\widehat{Q}(s,\varphi;\widehat{\theta}_j)$, are responsible for selecting action and evaluating action respectively. The double mode can address the issue of over-estimation Q-value in DQN, and then accelerate convergence velocity of iteration.

At the training process, the network Q will approximate gradually to the network $\widehat{Q}$ by the mini-batch Stochastic Gradient Descent (SGD) algorithm with the objective of minimizing the loss function as follow:

$$L(\theta_j) = E_{(s,\varphi,R(s,\varphi),s')\in \mathrm{M}_j} \left[\left(\widehat{y}_j - Q(s,\varphi;\theta_j) \right)^2 \right], \tag{21}$$

where $\widehat{y}_j = R(s,\varphi) + \gamma \cdot \widehat{Q}(s', \arg\max_{\varphi'} Q(s',\varphi';\theta_j);\widehat{\theta}_j)$ is the target value. The mini-batch SGD algorithm is stable with the less training samples and lower training error, which can accelerate the learning speed [35]. Then, in each

training step, the weight θ in the main network Q will be updated with the partial derivative $\Delta\theta_j = \frac{\partial L(\theta_j)}{\partial \theta_j}$ after running SGD. The update process of weight is denoted as

$$\theta_{j+1} = \theta_j + \alpha \cdot \Delta\theta_j. \tag{22}$$

Algorithm 1 Double DQN-based sampling resource allocation algorithm.

Initialization:

Initialize action space A and initialize experience replay memory $\widehat{M}$ and mini batch M.

Initialize some other super parameter learning rate α, replacing period κ, final exploration probability ϵ and discount factor γ.

Initialize main network Q with random weights θ.

Initialize target network $\widehat{Q}$ with $\widehat{\theta} = \theta$.

Iteration:

The learning system receives the flow number q^1 from SDN controller, then combines q^1 and the initial ratio of resource allocation r^1 into the first observed state s_1.
for episode $j = 1$ **to** J **do**
  Generate a random number g.
  if $g \leq \varepsilon_j$ **then**
    randomly choose an action φ_j,
  else
    choose the action $\varphi_j = \arg\max_{\phi} Q(s_j, \phi(s_j); \theta_j)$.
  end if
  Execute action φ_j, gain the reward $R(s_j, \varphi_j)$, and the new observation s_{j+1}.
  Construct the state-action transition $\widehat{T}_j = (s_j, \varphi_j, R(s_j, \varphi_j), s_{j+1})$,and store $\widehat{T}_j$ in $\widehat{M}_j$.
  Sample mini batch M_j of transitions from $\widehat{M}_j$ randomly after the experience pool amasses several samples.
  Based on Double DQN, select next optimal action $\varphi^* = \arg\max_{\varphi'} Q(s', \varphi'; \theta_j)$ according to the $s' \in M_j$ upon Q.
  Evaluate φ^* and calculate the target value $\widehat{y_j}$ under the immediate reward $R(s, \varphi)$ upon $\widehat{Q}$.

$$\widehat{y_j} = R(s, \varphi) + \gamma \cdot \widehat{Q}(s', \varphi^*; \widehat{\theta_j})$$

  Gain the loss function $L(\theta_j)$ by calculating the expectation of $(\widehat{y_j} - Q(s, \varphi; \theta_j))^2$.
  Update θ_j by partial derivative $\Delta\theta_j$ with the objective of minimizing loss.

$$\theta_{j+1} = \theta_j + \alpha \cdot \Delta\theta_j.$$

  Update ε_j by reduction $\Delta\varepsilon$ until achieving the final ϵ.

$$\varepsilon_{j+1} = \varepsilon_j - \Delta\varepsilon_j.$$

  Every κ steps update $\widehat{\theta}_j = \theta_j$.
end for

Commonly, in order to reduce the correlation between Q value and $\widehat{Q}$ value and further improve the stability of learning process, the weight $\widehat{\theta}$ in the target network $\widehat{Q}$ only updates after a fixed training times κ, and κ represents the period of replacing $\widehat{Q}$ with Q. Algorithm 1 shows the details of training process based on Double DQN, and we also refine this process with Fig. 2. By means of Algorithm 1, the SDN controller can periodically calculate the sampling duration for each sampling node, and then dynamically set corresponding sampling strategy for each node.

4.2 Adaptive node selection

Before getting the flow numbers of nodes, we should determine the sampling nodes. Based on the analysis above, we propose an iterative computing algorithm for selecting nodes, which can periodically interact with Algorithm 1. The goal of this iteration method is to select the nodes that exactly cover all the real-time flows, while reducing the sampling nodes.

Algorithm 2 Nodes selection by iterative influence.

Initialization:

Initialize the set $\mathbb{R}^s$ of selected nodes, $\mathbb{R}^s = \varnothing$.

Initialize the map $\mathbb{M}$ for storing the each result pair: (Node, Flow Number).

Iteration:

SDN controller will select sampling nodes until sampling process ends. During each sampling period(or episode) j,

for selection round $k = 1$ **to** n **do**
  Initialize the maximum number of flows $max = -1$
  for each node $R_i \in \mathbb{R} - \mathbb{R}^s$ **do**
    Subtract the redundant flows covered by selected nodes in previous rounds.
    $\mathbb{F}^q_{i,j} = F^q_{i,j} - \bigcup_{l=1}^{k-1} \widetilde{F}^q_l$. //if $k = 1$, $\mathbb{F}^q_{i,j} = F^q_{i,j}$
    if $|\mathbb{F}^q_{i,j}| > max$ **then**
      $max = |\mathbb{F}^q_{i,j}|$
      $R^s_k = R_i$
    end if
  end for
  if $max > 0$ **then**
    Select R^s_k as sampling node and put R^s_k into $\mathbb{R}^s$
    Put map (R^s_k, max) into $\mathbb{M}$
  else
    Break and end selecting nodes.
  end if
  Store the maximum number in each round, $\widetilde{F}^q_k = max$.
end for
Send $\mathbb{M}$ to deep neural network.

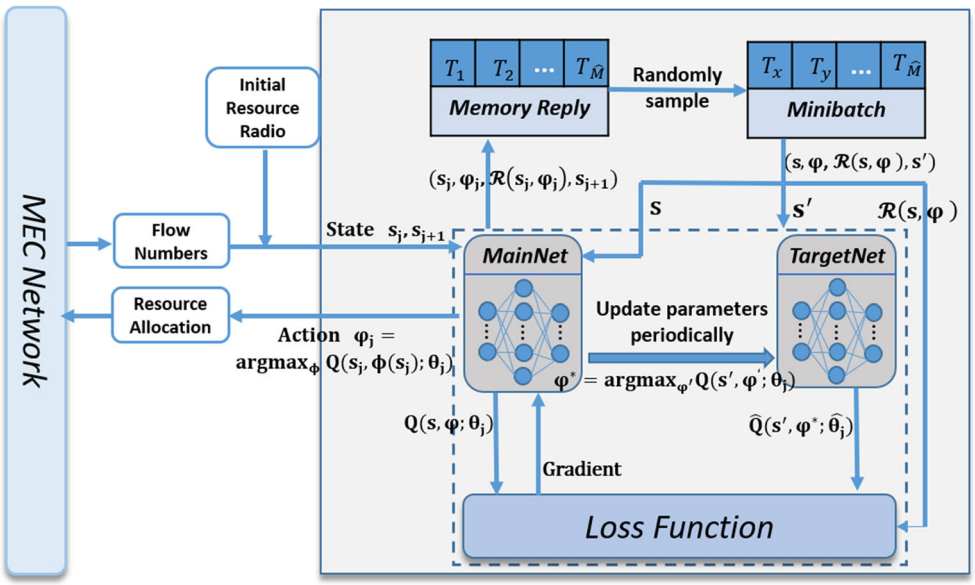


Fig. 2 Training process of the double DQN-based agent

Algorithm 2 describes the iterative selection process which occurs in each sampling period j. In the first iteration round, the node with the highest current influence, that is the node covering the maximum number of current flows, is selected as the sampling node. At the subsequent iterations, the flows covered by the selected nodes are no longer considered when the remaining nodes (defined as $\mathbb{R} - \mathbb{R}^s$) calculate the number of flows $|\mathbb{F}^q_{i,j}|$ covered by their own. Suppose R_i is selected in the k-th round, and let $\widetilde{F}^q_k$ be the set including the flows covered by R_i except that covered by the nodes selected in the previous rounds. Thus, in the $(k+1)$-th round, the iterative flow number of each remaining node $\mathbb{F}^q_{i,j} = F^q_{i,j} - \bigcup_{l=1}^{k} \widetilde{F}^q_l$, where $\bigcup_{l=1}^{k} \widetilde{F}^q_l$ includes all the flows covered by all the selected nodes of the previous k rounds. The iterative selection rounds will not stop until $max = 0$. Therefore, the final flow numbers of nodes after iterations are only measured from the non redundant flows, which can be formulated as $\mathbb{F}^q_{i,j} \bigcap \mathbb{F}^q_{i',j} = \varnothing$ for any two sampling nodes.

Table 1 Parameter setting

Symbol	Range	Description
L_1	128	The neurons in first layer
L_2	64	The neurons in second layer
$\widehat{M}$	2000	Experience pool capacity
M	128	Mini-batch size
γ	0.9	Discount factor
ϵ	0.1	Final exploration probability
α	0.05	Learning rate
κ	250	The period of replacing target $\widehat{Q}$
T	1	Sampling period

The iterative method achieves the full coverage of flows with as few nodes as possible, and it also removes the interference to the training state caused by the duplicate flow among nodes. In conclusion, the two-step approach constituted by Algorithm 1 and Algorithm 2 implements an adaptive and intelligent sampling strategy, which can deal with the complex problem in the real-world traffic environment.

5 Trace-driven evaluation and results

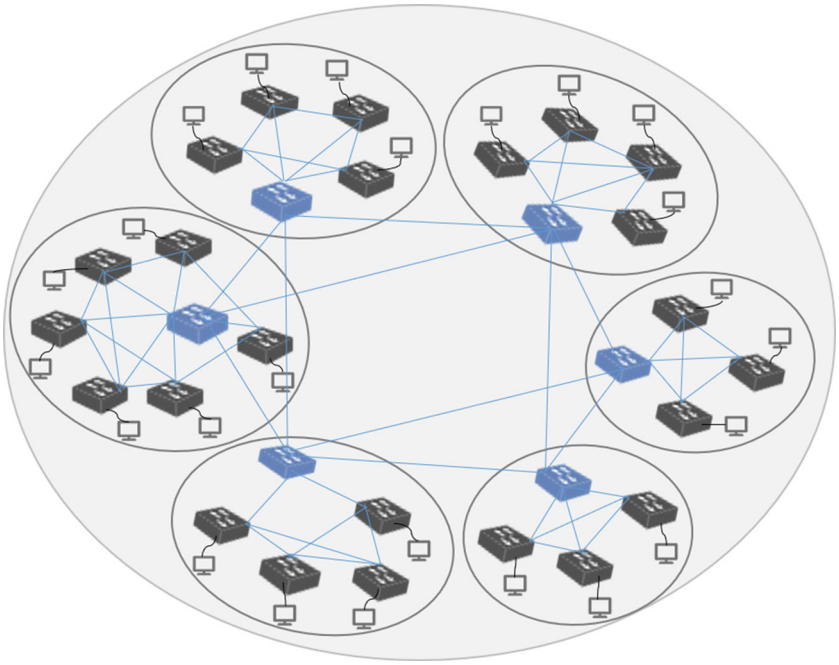
5.1 Experiment settings

To verify the performance of learning-based method, we built a experimental bed based on Floodlight controller, Tensorflow, OpenVswitch and Mininet. The test bed contains 4 Dell servers, and each server has a 20-core CPU and runs Ubuntu 16.06.2 LTS. One of the servers runs the Floodlight that is mainly responsible for the statistics of flow information. Another one performs the Double DQN-based algorithm with TensorFlow and the collector runs on another different ones. With respect to the parameter settings on the neural network, the two fully-connected hidden layers with the first-layer 128 neurons and the second-layer 64 neurons are used to serve as the evaluation and target $\widehat{Q}$ network. Table 1 gives the remaining parameter setting.

The network topology we deployed on the remaining servers comes from the open dataset provided by Shanghai Telecom[1] [36]. In order to simulate data packet transmission, virtual switches are used to replace base stations and

[1] http://sguangwang.com/TelecomDataset.html/

Fig. 3 Experimental topology



edge servers. This is equivalent to deploying a switch upon each base station or edge server. As shown in Fig. 3, a grey node is regarded as a base station and a blue one is an edge server, and a host is mounted to each base station node. In the experiments, we let $T = 1$ second. The real trace collected by "the WIDE Project"[2] in 2018/09/27 is used. We reserve the TCP and UDP flows, and select up to 13000 flows from the trace for different experiments. The radio of elephant flows to mice flows obeys the 2-8 principle [7].

5.2 Results and analysis

We implement four algorithms for selecting nodes.

1) Top-k based on Flow Betweenness Centrality (FBC) [18, 29]: Select the fixed K sampling nodes covering the most current active flows.
2) Top-k based on Betweenness Centrality (BC): Select the K sampling nodes with the highest structural influence in a topology [29].
3) Random-k: Select K nodes randomly.
4) Adaptive selection based on the Extend FBC (EFBC): Select the sampling nodes by Algorithm 2.

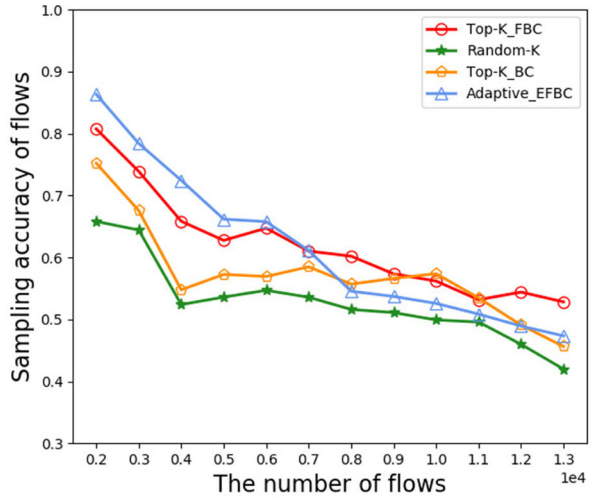
Then, the SDN controller will send the flow numbers of selected nodes to the deep neural network. For evaluating the sampling accuracy through a complete experiments, the four node selection algorithms should be combined with equal sampling duration or dynamic learning-based sampling duration respectively. The flow-level sampling accuracy is expressed as the rate of the number of captured flows to that of the total flows in the network throughout the experiment.

Figure 4 shows the performance comparison after applying multiple approaches with different flow numbers at the same packet transmission time when $K = 6$ and the collector's capacity C is 2000 packets/s. Obviously, as shown in Fig. 4a, the adaptive extend FBC method has the highest accuracy when the flow number is no more than 6000 compared with other three traditional algorithms. However, the accuracy of Top-k based on FBC or BC algorithm becomes higher with the increase of flow number. For example, when flow number rises to 8000, the accuracy of adaptive method decreases significantly. It's probable because many low-impact nodes are born in the network, even through iterative computation. In this way, the equal sampling duration used here wastes the finite sampling resources, which leads to a low accuracy.

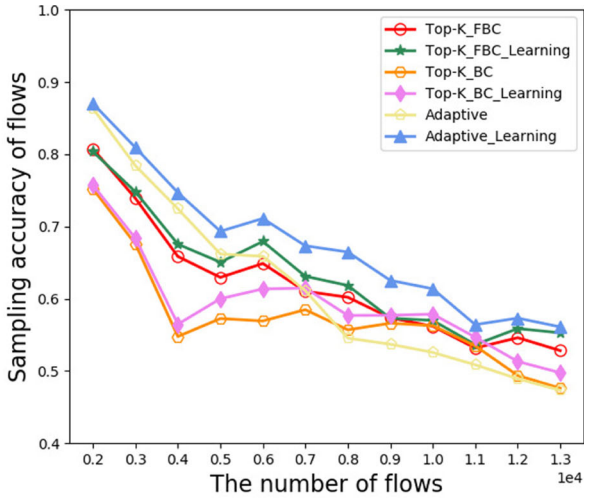
Hence, to address the issue, a Double DQN-based resource allocation algorithm is introduced, with an appropriate training episodes after multiple offline testing. We choose the FBC, BC and adaptive EFBC algorithms in Fig. 4a to combine with the learning-based duration allocation. In the latter figures, the result of the two learning-based method is under the optimal training episodes marked in Fig. 7.

Figure 4b shows that the learning-based duration allocation is effective for adaptive method EFBC whose accuracy is improved by 11.4%, compared with the basic adaptive method when the flow number is 8000. In

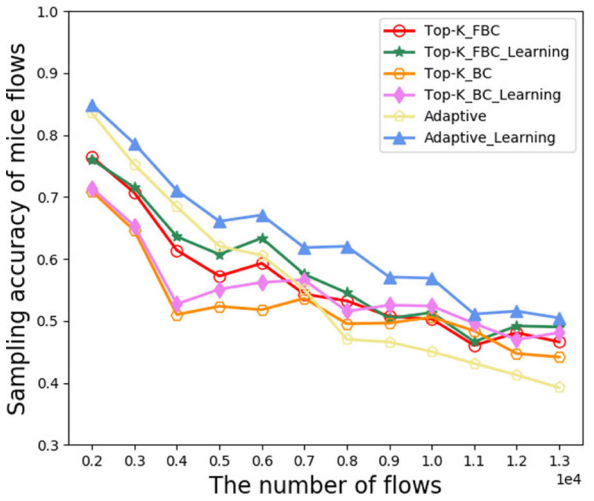
[2] http://mawi.wide.ad.jp/mawi/



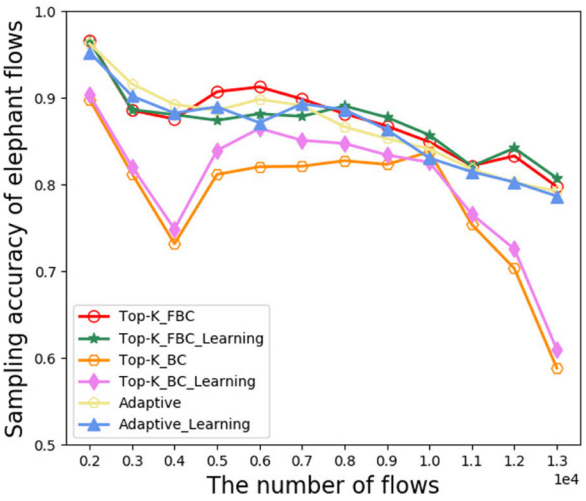
(a) Accuracy of total flows in a equal sampling duration



(b) Accuracy of total flows in a learning duration allocation



(c) Accuracy of mice flows



(d) Accuracy of elephant flows

Fig. 4 Comparison with respect to different algorithms

addition, the learning-based adaptive method is 5.9%, 10.4% higher than basic Top-k FBC and BC algorithm. It is quite possible that the adaptive learning method precisely identifies the nodes that actually cover more flows without the interference of redundant flows, and allocates more sampling resources to these high-impact nodes, from which it is more likely to capture flows. Meanwhile, the learning-based scheme contributes to allocating a small and appropriate amount of sampling duration to low-impact nodes for capturing the extra flows covered by them. On the other hand, the adaptive learning method can improve the accuracy of mice flows by 8.1% while maintaining the accuracy of elephant flows, which can be observed from the Fig. 4c and d. In fact, the promotions of total sampling accuracy is mainly due to that of mice flows. Furthermore, at the fixed collector capacity, the learning-based Top-k has only a few improvements in accuracy, especially when the flow number is too small or large. From the point of the downward trend in Fig. 4, an appropriate collector capacity may be also a key factor to improve sampling accuracy.

Therefore, in Fig. 5, different methods are compared with respect to collector capacity. We only select the Top-k FBC method in this experiment because of its better performance than Top-k BC as shown in Fig. 4. Clearly, the sampling accuracy increases with the collector capacity growing from 1000 to 8000 packets/s. For Top-k and learning-based Top-k, whatever value K is, the sampling accuracy keeps relatively stable and approaches each other gradually with higher collector capacity. Meanwhile, the adaptive learning-based method continues to be optimal even the growth of its accuracy slows down, especially when the capacity is large. The reason lies in that the enough resources help to achieve

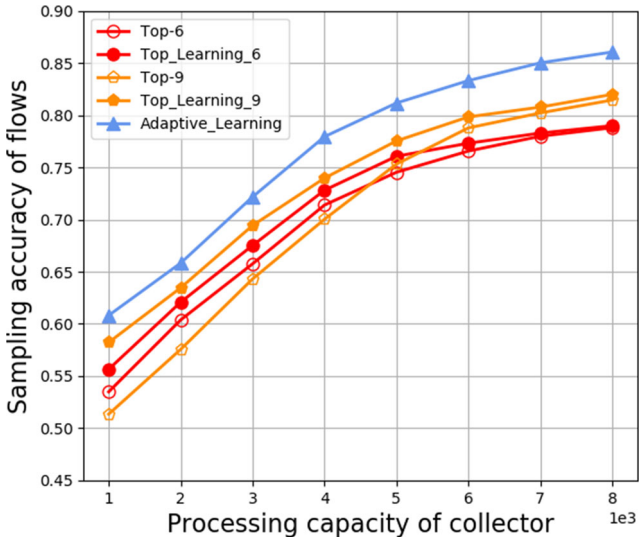
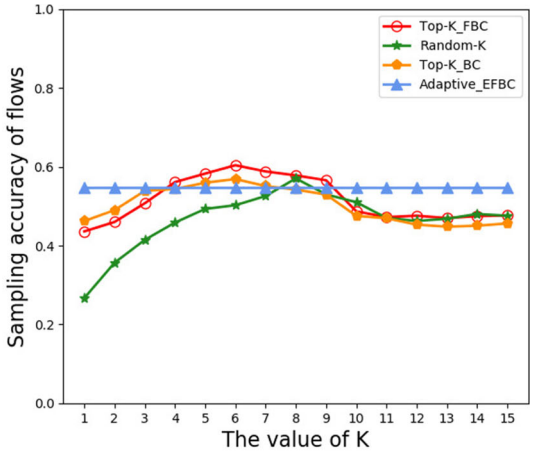


Fig. 5 Accuracy of total flows at different collector capacity

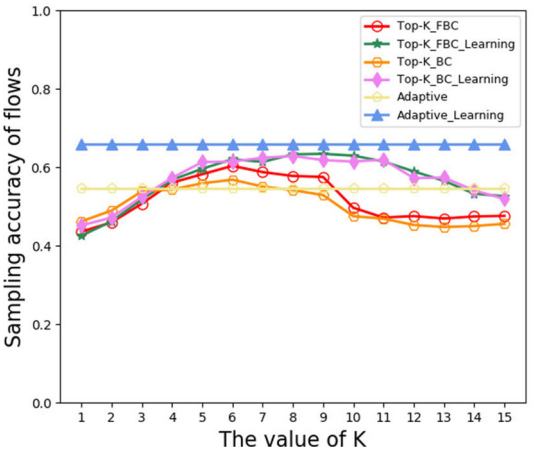
a more suitable allocation, in which the method can capture more flows with the full coverage of flows. Note that the given value of capacity is only to prove the validity of our algorithm.

Also, we compare the sampling accuracy at different K for each algorithm when flow number is 8000. As Fig. 6a shows, for the two traditional algorithms, many flows cannot be sampled when K is too small. On the contrary, the sampling resources allocated to each node will be relatively limited, which also results in a lower accuracy. The basic adaptive method has no relation to K, and its performance is inferior to the others sometimes. However, through a reformed learning way, the accuracy can be improved obviously and keep a stable value, as shown in Fig. 6b. Moreover, neither of the two Top-k methods can surpass the adaptive learning method, even if they get the maximum. For example, when K=9, the accuracy of Top-k FBC method gets the maximum but is still less than the learning-based adaptive method, from which we infer that our algorithm can indeed distribute resources more centrally to high-impact sampling nodes after iterative selection. On the other hand, the few improvements of accuracy in learning-based Top-k indicates the learning method does not work when K is too small or large.

Figure 7 shows the comparison of sampling accuracy at the training episodes varied from 1000 to 16000. As a whole, for the adaptive learning-based method, the accuracy increases with the increase of training steps. However, the learning-based Top-k has an obvious fluctuation. It may be that the flows of some nodes in the network change abruptly, which leads to a huge difference of state, and then disturbs the proportion of duration allocation, finally reduces the sampling accuracy. Importantly, the adaptive mode with a coverage of all the flows can alleviate the fluctuation. In



(a) Accuracy of total flows in a equal sampling duration



(b) Accuracy of total flows in a learning-based duration allocation

Fig. 6 Effect of K

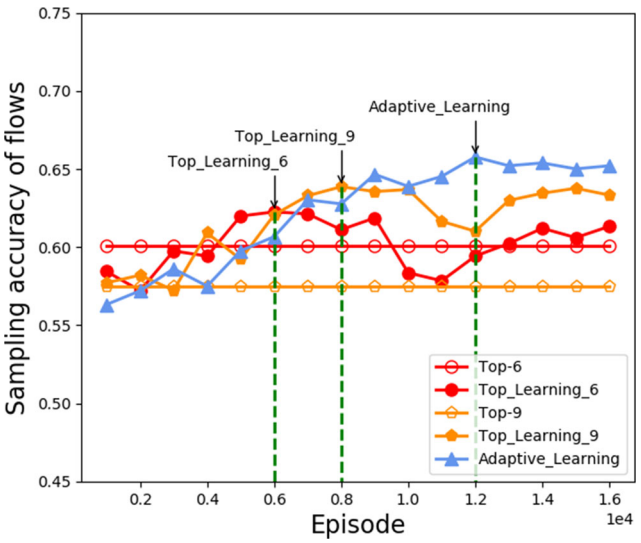


Fig. 7 Accuracy of total flows under different training episode

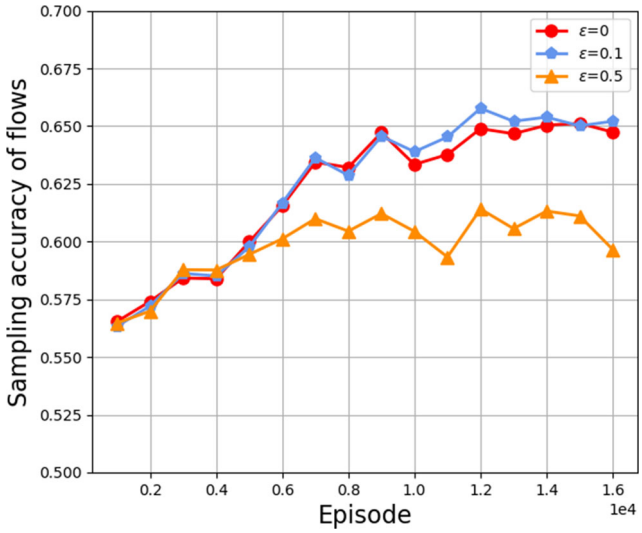


Fig. 8 Accuracy of total flows under different final exploration probability

addition, we can see that the best result of each algorithm is marked by an arrow along with a specific episode, at which the method mentioned above stops learning.

The effect of final exploration probability ϵ on the sampling accuracy is given in the Fig. 8. The comparison occurs among the optimal adaptive learning-based method with $\epsilon = 0, \epsilon = 0.1$ and $\epsilon = 0.5$. It can be seen that the result of $\epsilon = 0$ and $\epsilon = 0.1$ are similar, and both of them have a good performance improvement, compared with $\epsilon = 0.5$. As we know, the small ϵ may make the convergence performance of the DRL algorithm far from the optimal value. In our experiment, in order to maintain a certain exploration probability to cope with the changes of real-time network environment, we let $\epsilon = 0.1$.

6 Conclusion

In this article, we proposed a flow-awareness framework based Double DQN and SDN techniques among MEC nodes, with a learning-based sampling strategy, for adaptively selecting sampling nodes and dynamically allocating sampling resource. We carried out the detailed experiments based on realistic traffic traces. Finally, compared with three traditional algorithms, our proposed approach can achieve excellent performance in terms of the flow-level sampling accuracy, especially for mice flows.

Acknowledgments This work is partially supported by the National Key R&D Program of China (2018YFC0809803), China NSFC (Youth) through grant 61702364, China NSFC GD Joint fund U1701263.

References

1. Li XH, Wang XF, Li KQ, Han Z, Leung VCM (2017) Collaborative Multi-Tier caching in heterogeneous networks: modeling, analysis, and design. IEEE Trans 16(10):6926–6939
2. Wang XF, Zhang Y, Leung VCM, Guizani N, Jiang TP (2018) D2D big data: content deliveries over wireless device-to-device sharing in realistic large scale mobile networks. IEEE Wirel Commun 25(1):32–38
3. Wang XF, Chen M, Leung VCM, Hwang Z, Hwang K (2018) Integrating social networks with mobile device-to-device services, IEEE Trans, https://doi.org/10.1109/TSC.2018.2867437
4. Wang XF, Hang YW, Wang CY et al In-edge AI: intelligentizing mobile edge computing, caching and communication by federated learning, IEEE Network Magazine, arXiv:1809.07857
5. Li XH, Wang XF, Wang PJ, Han Z (2018) Hierarchical edge caching in device-to-device aided mobile networks: modeling, optimization, and design. IEEE JSAC 36(8):1768–1785
6. Nunna S, Kousaridas A, Ibrahim M et al (2015) Enabling real-time context-aware collaboration through 5G and mobile edge computing, in ITNG, pp 601–605
7. Benson T, Akella A, Maltz DA (2010) Network traffic characteristics of data centers in the wild, in ACM SIGCOMM, pp 267–280
8. Patel M et al (2014) Mobile-edge computing introductory technical white paper. White Paper, Mobile-Edge Computing (MEC) Industry Initiative
9. Barr AB, Harchol Y, Hay D, Koral Y (2014) Deep packet inspection as a service, in ACM coNEXT, pp 271–282
10. Zhu YB, Kang NX, Cao JX, Greenberg A, Lu GH (2015) Packet-level telemetry in large datacenter networks. ACM SIGCOMM 45(4):479–491
11. Paolucci F, Sgambelluri A, Cugini F (2018) Network telemetry streaming services in SDN-based disaggregated optical networks. J Lightwave Technol 32(15):3142–3149
12. Su Z, Wang T, Hamdi M (2015) COSTA: cross-layer Optimization for sketch-based software defined measurement task assignment, in IEEE IWQos, pp 183–188
13. Yu M, Jose L, Miao R (2013) Software defined traffic measurement with OpenSketch, in USENIX NSDI, pp 29–42
14. Bouet M, Leguay J, Conan V (2014) Cost-based placement of virtualized deep packet inspection functions in SDN, in IEEE MILCOM, pp 992–997
15. Suh J, Kwon TT, Dixon C, Felter W, Carter J (2014) Opensample: a low-latency, sampling-based measurement platform for commodity SDN, in IEEE ICDCS, pp 228–237
16. Xing CY, Ding K, Hu C, Chen M (2016) Sample and fetch-based large flow detection mechanism in software defined networks. IEEE Communication Letters 20(9):1764–1767
17. Liu Q, Li P, Zhao W, Cai W, Yu S, Leung VCM (2018) A survey on security threats and defensive techniques of machine learning: a data driven view. IEEE Access 6:12103–12117
18. Yoon S, Ha T, Kim S, Lim H (2017) Scalable traffic sampling using centrality measure on software-defined networks. IEEE Communications Magazine 55(7):43–49
19. Ha T, Kim S, An N, Narantuya J, Jeong C, Kim JW (2016) Suspicious traffic sampling for intrusion detection in software-defined networks. Comput Netw 109(2):172–182
20. Abbas N, Zhang Y, Taherkordi A, Skeie T (2018) Mobile edge computing: a survey. IEEE Internet Things J 5(1):450–465
21. Su Z, Wang T, Xia Y, Hamdi M (2015) CeMon: a cost-effective flow monitoring system in software defined networks. Comput Netw 92(1):101–115

22. Tahaei H, Salleh RB, Razak MF, Ko K, Anuar NB (2017) An efficient sampling and classification approach for flow detection in SDN-based big data centers, in IEEE AINA, pp 1106–1115
23. He Y, Zhao N, Yin HX (2018) Integrated networking, caching, and computing for connected vehicles: a deep reinforcement learning approach. IEEE Trans 67(1):44–55
24. Qiu C, Yao HP, Yu R, Xu FM, Zhao CL (2019) Deep Q-learning aided networking, caching, and computing resources allocation in software-defined satellite-terrestrial networks, in IEEE Transactions
25. Li H, Gao H, Lv TJ, Lu Y (2018) Deep Q-learning based dynamic resource allocation for self-powered ultra-dense networks, in IEEE ICC, pp 1–6
26. He Y, Zhang Z, Yu FR, Zhao N (2017) Deep-reinforcement-learning-based optimization for cache-enabled opportunistic interference alignment wireless networks. IEEE Trans 66(11):10433–10445
27. Taleb T, Samdanis K, Mada B, Flinck H, Dutta S (2017) On multi-access edge computing: a survey of the emerging 5g network edge cloud architecture and orchestration. IEEE Communications Surveys & Tutorials 19(3):1657–1687
28. Tong L, Gao W (2016) Application-aware traffic scheduling for workload offloading in mobile clouds, in IEEE INFOCOM, pp 1–9
29. Lu L, Chen D, Ren XL, Zhang QM, Zhang YC, Zhou T (2016) Vital nodes identification in complex networks. Phys Rep 65:1–63
30. Bai W, Chen L, Chen K, Han DS, Tian C, Wang H (2017) PIAS: practical information-agnostic flow scheduling for commodity data centers. IEEE TON 25(4):1954–1967
31. Vanini E, Pan R, Alizadeh M, Taheri P (2017) Let it flow resilient asymmetric load balancing with flowlet switching, in USENIX NSDI, pp 407–420
32. Alizadeh M, Yang S, Sharif M, Taheri P, Katti S (2013) pFabric: minimal near-optimal datacenter transport. ACM SIGCOMM 43(4):435–446
33. Mnih V, Kavukcuoglu K, Silver D, Rusu AA (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533
34. Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning, in AAAI, pp 2094–2100
35. Wilson AC, Roelofs R, Stern M, Srebro N, Recht B (2017) The marginal value of adaptive gradient methods in machine learning, in NIPS, vol 30
36. Wang S, Zhao Y, Xu J, Yuan J, Hsu CH (2018) Edge server placement in mobile edge computing, in https://doi.org/10.1016/j.jpdc.2018.06.008

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.