Improved Flow Awareness by Intelligent Collaborative Sampling in Software Defined Networks

Jun Deng, He Cai, and Xiaofei Wang(✉)

Tianjin Key Laboratory of Advanced Networking,
College of Intelligence and Computing, Tianjin University, Tianjin, China
xiaofeiwang@tju.edu.cn

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 load in 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 cooperative caching, content sharing and intensive-computation offloading. Therefore, global accurate flow awareness in Mobile Edge Computing (MEC) system is an urgent need for fine-grained quality of service guarantee, security analysis, content awareness, internal network management and so on. These applications in MEC business model requires more flows, with their deep information (e.g., security, application type, flow behavior) which is brought by the payload of sampled packets.

V. C. M. Leung et al. (Eds.): 5GWN 2019, LNICST 278, pp. 182–194, 2019.
https://doi.org/10.1007/978-3-030-17513-9_13

Especially for security of network edge devices, DOS attacks could be packet-based with only a small amount packets and hide in short-life **mice flows** [1,2], which makes themselves hard to capture.

It is challenging to deeply obtain flow information while sampling network-wise flows accurately under DPI technology [3] or statistics-based reports created by other measurements based on telemetry or sketch [4–6]. DPI is used to inspect packet payload at the gateways or access points, which is, however, hard to be deployed on all nodes [7]. This 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 [8,9]. However, the sampled packets of the same flow is non-consecutive, which is useless for recognizing the flow-level deep information. Fortunately, systematic packet sampling (SPS) which samples within a certain duration can capture consecutive packets [10,11]. The SDN with SPS is conducive to the global and cyclical sampling, which can achieve high-accuracy flow-level sampling. Besides, the MEC concept along with SDN can enable more applications requiring computing and storage resources [12].

However, due to restrictions on the resources (e,g., the processing capacity of collector) of MEC system, the selected sampling nodes should not be too many and the sampling duration cannot be assigned unlimitedly. Existing studies [10,11,13] have proposed different approaches by SPS to select the sampling nodes or allocate the resource in a real network, but they are limited to a certain extent. For example, study [10] 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 are not guaranteed to cover all the flows in the network, and the decision of sampling rate cannot change dynamically according to the active flows covered by the nodes. In addition, study [13] selects sampling nodes based on a greedy strategy which can cover all the active flows, and it proposes a adaptive scheme to adjust the sampling rate considering the bandwidth resource. But its algorithm results in a reduced accuracy. Fortunately, a new technology, namely deep reinforcement learning (DRL), has an advantage to obtain the resource allocation policy for solving the complexity problem in real wireless environment [14,15].

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 by 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 selected 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 with 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 the accuracy of mice flows by 8.1%.

The remainder of this paper is organized as follows: The system model and problem formulation is discussed in Sect. 2. Then the learning-based sampling strategy and the resource allocation algorithm will be discussed in Sect. 3 in detail along with experiments in Sect. 4. Finally, conclusions are given in Sect. 5.

2 System Model and Problem Formulation

2.1 Flow-Awareness Architecture Based on SDN in MEC

The flow-awareness architecture based on SDN in MEC is illustrated in Fig. 1. 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 an edge server. In our flow-awareness architecture, any base station and edge server can be selected as a sampling node. Besides, the SDN controller gathers and sends the flow information generated by these nodes to the learning-based neural network, after which the controller can receive a feedback about sampling duration allocation. Another component, traffic collector is responsible for collecting the sampled packets and informing the controller of the number of flows.

2.2 Sampling Modeling and Objective

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

Quantification of the Influence and Cost. The quantization of influence is based on the Flow Betweenness Centrality (FBC) [10,16], and we extended it. We define the set R including all nodes, so the number of nodes is $n = |R|$. Let F_i^c be the set including all active flows currently passing through R_i, and F^c is the set including all current active flows in the network, and $F^c = \bigcup_{i=1}^{n} F_i^c$. For each $f_k \in F_i^c$, suppose f_k transmits packets as a Poisson process with a rate λ_k in a unit time t [17], and λ_k is independent of the other flows. Thus, the probability that f_k is captured by R_i within t is

$$P\{N_k(t) > 0\} = 1 - P\{N_k(t) = 0\} = 1 - \frac{1}{e^{\lambda_k \cdot t}}. \tag{1}$$

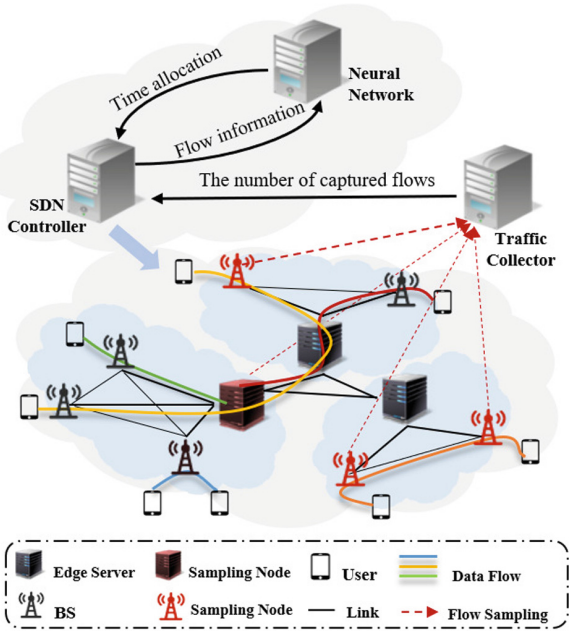


Fig. 1. Illustration of flow-awareness architecture in SDN.

and the expected number of active flows captured by R_i is $\sum_{f_k \in F_i^c} P\{N_k(t) > 0\}$. The influence of R_i within t can be quantified as (2), which describes the rate of the number of flows that a node can capture within t to the number of network-wise current active flows.

$$I_i(t) = \sum_{f_k \in F_i^c} P\left\{N_k\left(t\right) > 0\right\} / |F^c|. \tag{2}$$

Cost w_i is the total number of packets captured from R_i during t, which can be expressed as $w_i = \frac{v_i}{T/t}$, where v_i is the current packet speed (packets/T) during the sampling period T.

Objective. We formulate the influence-based sampling model in (3), with the constraints (4) and (5), where $\widehat{t_i}$ is the sampling duration allocated to R_i, and C is the capacity of collector (packets/T).

$$\max \sum_{i=1}^{n} I_i(\widehat{t_i}). \tag{3}$$

$$\text{s.t.} \sum_{i}^{n} w_i \cdot (\widehat{t_i}/t) \leq C. \tag{4}$$

$$0 \leq \widehat{t_i} \leq T. \tag{5}$$

The goal is to work out corresponding $\widehat{t_i}$ for each R_i to maximize the total influences of the system during T under the constraints of C. Maximizing the value of (3) means capturing the most flows, which is equivalent to optimizing sampling accuracy.

The complexity for solving the problem is $O(2^n)$ considering whether a node is assigned sampling duration or not. Nonetheless, the lacking of the exact packet arrival strength λ_k make the solution is not completely practical. In this paper, we find a reinforcement learning method to replace the traditional solution.

2.3 Specifics of MDP Model

In this section, we describe a brief review of reinforcement learning and Q-learning, and then the specifics of Markov Decision Process is presented.

Q-Learning. Q-Learning is one of widely-used model free method in Reinforcement learning, where an agent can gain reward and automatically learn to behave by interacting with the external environment and constantly trying to act. The Q-Learning algorithm which is used for bandwidth resource allocation here can be modeled as a MDP with followed components: state space, action space and reward function.

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 is the number of the flows 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 of each node $\widehat{t_i^j}$ 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

$$\widehat{t^j} = \frac{\mathrm{C} \cdot T}{\sum_i^n \mathrm{r}_i^j \cdot \mathrm{v}_i^j}, \quad i, j \in \mathbb{Z}, \tag{6}$$

then,

$$\widehat{t_i^j} = r_i^j \cdot \widehat{t^j}. \tag{7}$$

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

$$\mathrm{s}_j = [q^j, \quad r^j] \in S, \tag{8}$$

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

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

$$A = [a_j^1,\ a_j^{-1},\ a_j^0], \tag{9}$$

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.

Reward Function. The reward function needs to take into account the objectives of flow-awareness framework. As the (3) 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, during the j-th period, we define F_j^s as the non redundant flows sampled by all the node in the network, and F_j^t is all flows covered by all the nodes. So the sampling accuracy in the j-th period is denoted as

$$\rho = \frac{F_j^s}{F_j^t}. \tag{10}$$

Then, in our flow-awareness architecture, reward function $R(s_j, \phi(s_j))$ is defined as

$$R(s_j, \phi(s_j)) = e^{\rho}, \tag{11}$$

when the agent perform the action $\phi(s_j)$ upon the state s_j.

2.4 Problem Formulation

Q-Learning solution is proposed here. Based on (11), taking into account the lone-term discount rewards, the expectation of maximizing objective upon the initial state $s_j = s$ is defined by a state value function, called Q-function:

$$Q(s, \varphi) = E\left[\sum_{j=1}^{J\to\infty} (\gamma)^{j-1} \cdot R(s, \varphi) | s_1 = s\right] \tag{12}$$

where $\gamma \in [0, 1)$ is the discount factor, and $\varphi = \phi(s_j)$.

The Q-function obeys a Bellman criterion. Let the $s^{'} = s_{j+1}$ is the next state after taking the action φ, so the (12) can be optimized as:

$$Q(s, \varphi) = R(s, \varphi) + \gamma \max_{\varphi^{'}} Q(s^{'}, \varphi^{'}). \tag{13}$$

After taking the optimal action, the iterative formula of Q-function can be obtained as:

$$Q^{j+1}(s,\varphi) = Q^{j}(s,\varphi) + \alpha \cdot \left(R(s,\varphi) + \gamma \max_{\varphi'} Q^{j}(s^{'},\varphi^{'}) - Q^{j}(s,\varphi) \right), \quad (14)$$

where $\alpha \in [0,1]$ is the learning rate. The (14) expresses that after the agent performs the current optimal action $\phi(s_j)$, the state s_j turn to the state s_{j+1} and the corresponding reward is fed back.

Q-Learning use 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.

3 Sampling Strategy Based on DRL

In this section, we formulate the dynamic sampling duration allocation problem as a deep reinforcement learning process.

3.1 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 have 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

$$Q(s,\varphi) = Q((s,\varphi);\theta_j). \quad (15)$$

In addition, the corresponding state transition $\widehat{T_j} = (\mathrm{s}_j, \phi(\mathrm{s}_j), \mathrm{R}(\mathrm{s}_j, \phi(\mathrm{s}_j)), \mathrm{s}_{j+1})$ will be stored in experience pool with a pre-defined capacity M at each training step. After the samples in the pool have accumulated to a certain number, the agent will sample the M_j batches (past experience) randomly for the reflective learning, which is what we called experience replay.

In our model, a more reliable algorithm Double Deep Q-Network (Double DQN) [18] is used for training to achieve appropriate resource allocation, which can address the issue of over-estimation Q-value in DQN, and then accelerate convergence velocity of iteration. It is because Double DQN use different Q-network, known as main network $Q(s,\varphi;\theta_j)$ and target network $\widehat{Q}(s,\varphi;\widehat{\theta}_j)$ respectively, to select action and evaluate action.

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

$$L(\theta_j) = E_{(\mathrm{s},\varphi,\mathrm{R}(\mathrm{s},\varphi),\mathrm{s}')\in \mathrm{M}_j} \left[\left(\widehat{y_j} - Q(s,\varphi;\theta_j) \right)^2 \right], \quad (16)$$

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. Algorithm 1 shows the details of training process based on Double DQN.

Algorithm 1 Double DQN-based Sampling Resource Allocation Algorithm

Initialization:

Initialize action space A.
Initialize experience replay memory M.
Initialize main network Q with random weights θ.
Initialize target network $\widehat{Q}$ with $\widehat{\theta} = \theta$.

Iteration:

1: 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.
2: **for** episode $j = 1$ **to** J **do**
3: Generate a random number g
4: if $g \leq \varepsilon$
5: randomly choose an action $\phi(s_j)$
6: else
7: choose action $\arg\max_{\phi(s_j)} Q(s_j, \phi(s_j); \theta_j)$
8: Execute action $\phi(s_j)$, gain the reward $R(s_j, \phi(s_j))$, and the new observation s_{j+1}.
9: Construct the transition $\widehat{T_j} = (s_j, \phi(s_j), R(s_j, \phi(s_j)), s_{j+1})$, and store $\widehat{T_j}$ in M.
10: Sample M_j mini-batch of transitions from M randomly.
11: Update θ_j by minimizing loss with partial derivative $\frac{\partial L(\theta_j)}{\partial \theta_j}$.
12: Every κ update $\widehat{\theta}_j = \theta_j$.

4 Trace-Driven Evaluation and Results

4.1 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 6 Dell servers, and each server has a 20-core CPU and runs Ubuntu 16.06.2 LTS. One of the servers runs the Floodlight which is mainly responsible for the statistics of flow information of nodes. 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 neural network, the two fully-connected hidden layers with the first-layer 128 neurons and the second-layer 64 neurons is used to serve as the eval and target Q network. Table 1 gives the remaining parameter setting.

The network topology we deploy on the remaining 3 servers comes from the open dataset provided by Shanghai Telecom[1] [19]. In order to simulate data

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

Table 1. Parameter setting

Symbol	Range	Description
M	2000	Experience pool capacity
M_j	128	Minibatch size
γ	0.9	Discount factor
ϵ	0.1	Final exploration probability
α	0.05	Learning rate
κ	250	The period of replacing target Q network

packet transmission, virtual switches are used to replace base stations and edge servers. This is equivalent to deploying a switch upon each base station or edge server. As shown in Fig. 2, 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 $T = 1$ s. 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 sets of experiments. The radio of elephant flows to mices flow obeys the 2–8 principle [1].

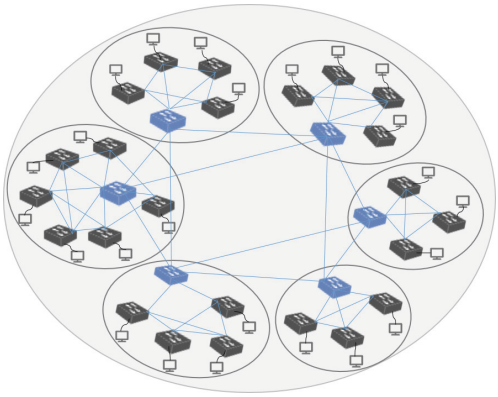


Fig. 2. Experimental topology.

4.2 Results and Analysis

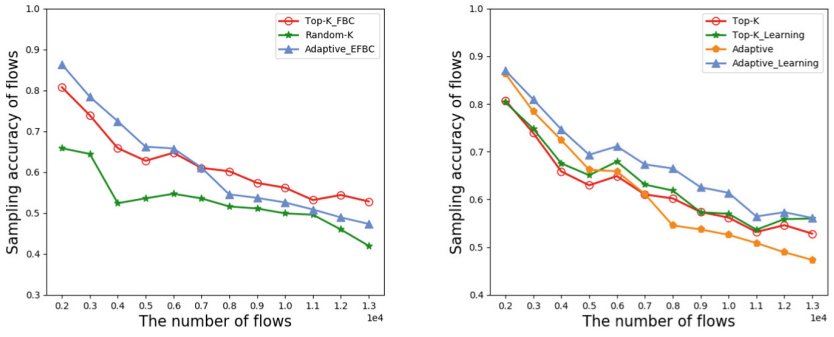
We implement three algorithms for selecting nodes.

(1) Top-k based on Flow Betweenness Centrality (FBC) [10]: Select the fixed K sampling nodes covering the most current active flows.

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

(2) Random-k: Select K nodes randomly.
(3) Adaptive selection based on the extend FBC: Select the sampling nodes by an iterative computing method. In the first iteration, the node with the most current active flow is selected, which is the same as Top-k based on FBC. But at the subsequent iteration, the flows covered by the selected nodes is no longer considered when the remaining nodes calculate the number of flows covered by their own. The iterative selection rounds will not stop until the number of flows at a node is calculated to zero. This original method can ensure coverage of all the flows in the network while reducing sampling nodes.

Then, the SDN controller will send the flow number of selected nodes to the neural network. For evaluating the sampling accuracy through a complete experiments, the three 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.



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

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

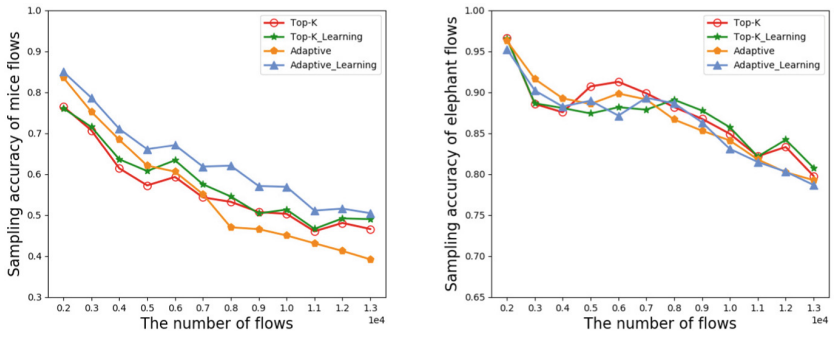
Fig. 3. Comparison with respect to different algorithms.

Figures 3 and 4 show the performance comparison after applying multiple approaches with different flow number at the same time of packet transmission when $K = 6$ and the collector's capacity C is 2000 packets/s. Obviously, as shown in Fig. 3(a), the sampling accuracy is in a downward trend. Besides, the adaptive method has the highest accuracy when the flow number is no more than 6000 compared with the other three traditional algorithms. However, the accuracy of Top-k based on FBC 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 computing. In this way, the method of equal

sampling duration we use here wastes the finite sampling resource, which leads to a low accuracy.

Hence, for addressing the issue, a Double DQN-based resource allocation algorithm is introduced, with an appropriate training episodes after multiple offline testing. We choose the two best algorithms in Fig. 3(a) to combine with the learning-based method.

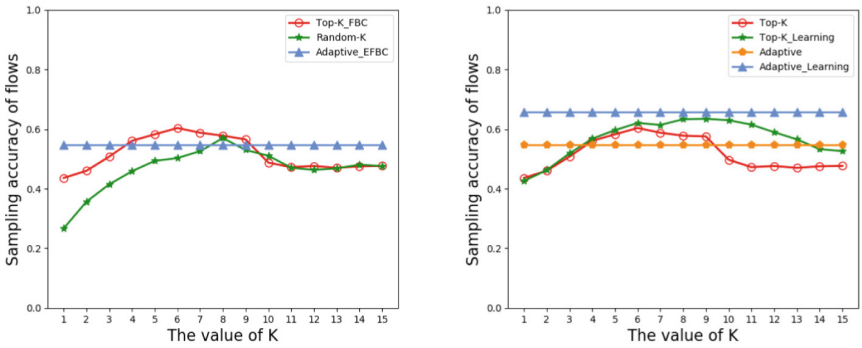
Figure 3(b) shows that the adaptive learning-based method increases the accuracy by around 5.9%, 4.2% and 11.4%, compared with the basic Top-K, learning-based Top-k and basic adaptive method when the flow number is 8000. It is quiet possible that the adaptive learning-based method precisely identifies the nodes that actually covers more flows without the interference of redundant flows, and allocates more sampling resource to these nodes. However, the learning-based Top-k has only a few improvements in accuracy, which indicates the learning-based method does not work when the K is too small, e.g. K = 6. Meanwhile, the learning-based adaptive 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. 4(a) and (b). In fact, the raises of total sampling accuracy mainly due to that of mice flows.



(a) Accuracy of mice flows (b) Accuracy of elephant flows

Fig. 4. Comparison with respect to different algorithms.

Also, we compare the sampling accuracy at different K for each algorithm when flow number is 8000. As Fig. 5(a) shows, for the traditional algorithm, 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. 5(b). Moreover, when $K = 9$, the accuracy of learning-based Top-k reaches a maximum but is still less than the learning-based adaptive method, from which we can infer that the iterative selection can indeed distribute resources more centrally to high-impact sampling nodes.



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

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

Fig. 5. Effect of K.

5 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 two traditional algorithms, our proposed approach can achieve excellent performance in terms of the flow-level sampling accuracy, especially for mice flows.

References

1. Benson, T., Akella, A., Maltz, D.A.: Network traffic characteristics of data centers in the wild. In: ACM SIGCOMM Conference on Internet Measurement, pp. 267–280, November 2010
2. Patel, M., et al.: Mobile-edge computing–introductory technical white paper. White Paper, Mobile-Edge Computing (MEC) Industry Initiative (2014)
3. Bremler-Barr, A., Harchol, Y., Hay, D., Koral, Y.: Deep packet inspection as a service. In: ACM CoNEXT, pp. 271–282, December 2014
4. Zhu, Y.B., Kang, N.X., Cao, J.X., Greenberg, A., Lu, G.H.: Packet-level telemetry in large datacenter networks. ACM SIGCOMM **45**(4), 479–491 (2015)
5. Su, Z.Y., Wang, T., Hamdi, M.: COSTA: cross-layer optimization for sketch-based software defined measurement task assignment. In: IEEE IWQoS, pp. 183–188, June 2015
6. Yu, M., Jose, L., Miao, R.: Software defined traffic measurement with OpenSketch. In: USENIX NSDI, pp. 29–42, April 2013
7. Bouet, M., Leguay, J., Conan, V.: Cost-based placement of virtualized deep packet inspection functions in SDN. In: IEEE MILCOM, pp. 992–997, February 2014
8. Suh, J., Kwon, T.T., Dixon, C., Felter, W., Carter, J.: OpenSample: a low-latency, sampling-based measurement platform for commodity SDN. In: IEEE ICDCS, pp. 228–237, July 2014

9. Xing, C.Y., Ding, K., Hu, C., Chen, M.: Sample and fetch-based large flow detection mechanism in software defined networks. IEEE Commun. Lett. **20**(9), 1764–1767 (2016)
10. Yoon, S., Ha, T., Kim, S., Lim, H.: Scalable traffic sampling using centrality measure on software-defined networks. IEEE Commun. Mag. **55**, 43–49 (2017)
11. Ha, T., Kim, S., An, N., Narantuya, J., Jeong, C., Kim, J.W.: Suspicious traffic sampling for intrusion detection in software-defined networks. Comput. Netw. **109**(2), 172–182 (2016)
12. Abbas, N., Zhang, Y., Taherkordi, A., Skeie, T.: Mobile edge computing: a survey. IEEE Internet Things J. **5**(1), 450–465 (2018)
13. Su, Z., Wang, T., Xia, Y., Hamdi, M.: CeMon: a cost-effective flow monitoring system in software defined networks. Comput. Netw. **92**(1), 101–115 (2015)
14. He, Y., Zhao, N., Yin, H.X.: Integrated networking, caching, and computing for connected vehicles: a deep reinforcement learning approach. IEEE Trans. Veh. Technol. **67**(1), 44–55 (2018)
15. Li, H., Gao, H., Lv, T.J., Lu, Y.: Deep Q-learning based dynamic resource allocation for self-powered ultra-dense networks. In: IEEE ICC, pp. 1–6, July 2018
16. Lu, L., Chen, D., Ren, X.L., Zhang, Q.M., Zhang, Y.C.: Vital nodes identification in complex networks. Phys. Rep. **65**, 1–63 (2016)
17. Vanini, E., Pan, R., Alizadeh, M., Taheri, P., Edsall, T.: Let it flow resilient asymmetric load balancing with flowlet switching. In: USENIX NSDI, pp. 407–420, May 2017
18. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. In: Proceedings 30th AAAI Conference Artificial Intelligence, pp. 2094–2100 (2016)
19. Wang, S., Zhao, Y., Xu, J., Yuan, J., Hsu, C.H.: Edge server placement in mobile edge computing, June 2018. https://doi.org/10.1016/j.jpdc.2018.06.008