IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 10, 2021, accepted March 18, 2021, date of publication March 24, 2021, date of current version April 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3068459

Deep Reinforcement Learning-Based Traffic Sampling for Multiple Traffic Analyzers on Software-Defined Networks

SUNGHWAN KIM[1], (Student Member, IEEE), SEUNGHYUN YOON[1], (Graduate Student Member, IEEE), AND HYUK LIM[2], (Member, IEEE)

[1]School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology (GIST), Gwangju 61005, Republic of Korea
[2]AI Graduate School, Gwangju Institute of Science and Technology (GIST), Gwangju 61005, Republic of Korea

Corresponding author: Hyuk Lim (hlim@gist.ac.kr)

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) funded by the Korean Government (MSIT) under Grant 2017-0-00421, and in part by the IITP funded by the MSIT through the Artificial Intelligence Graduate School Program (GIST) under Grant 2019-0-01842.

ABSTRACT Intrusion detection system (IDS) and deep packet inspection (DPI) are widely used to detect network attacks and anomalies, thereby enhancing cyber-security. Conventional traffic analyzers such as IDS have fixed locations and a limited capacity to perform DPI on large volumes of network traffic. Nowadays, software-defined networking (SDN) technology, which provides flexibility, elasticity, and programmability by decoupling the network control and data planes, makes it possible to capture entire or a certain portion of data traffic flows on SDN-capable switches and steer the captured network traffic to one of the traffic analyzers on the network. Therefore, how to sample network traffic and where to steer the sampled traffic among multiple traffic analyzers are critical problems facing cyber-security. Since there is a possibility that potentially useful information will be lost in not-captured traffic, deciding the sampling points and sampling rates of network traffic remains important. Additionally, after determining the sampling points and rates, sampled traffic must be sent to one of the multiple traffic analyzers for traffic inspection, which may incur additional network delivery overheads. We propose a less-intrusive traffic sampling mechanism for multiple traffic analyzers on an SDN-capable network using a deep deterministic policy gradient (DDPG), which is a representative deep reinforcement learning (DRL) algorithm for continuous action control. The proposed system learns sampling resource allocation policy under the uncertainty of flow distribution according to sampled traffic inspection results obtained from multiple traffic analyzers. Through extensive simulations and the SDN-based testbed experiments, we demonstrate that the proposed approach has a high probability of capturing malicious flows while maintaining a balanced load of multiple traffic analyzers and reducing flow monitoring overheads.

INDEX TERMS Cyber-security, network traffic monitoring, intrusion detection system, software-defined networking, deep reinforcement learning.

I. INTRODUCTION

The Internet is an integral part of the daily lives of many people and this has caused a tremendous increase in network traffic. Due to increasing numbers of Internet users and network traffic, the importance of and demand for cyber-security has steadily increased because it can provide a defense against network attacks and anomalies, such as denial of service (DoS) attacks, man-in-the-middle attacks, and malware. Advanced persistent threats (APT) are a particularly serious threat to the Internet. APT uses a variety of attack techniques and operates silently such that an attacker can maintain control over a target system for an extended period of time without being detected [1]. In other words, APT attacks are difficult to detect because they steal valuable information via stealthy and persistent attack processes from specific target establishments, such as government agencies, corporations, or the military. The purpose of an APT attack is

The associate editor coordinating the review of this manuscript and approving it for publication was Thanh Ngoc Dinh.

not to paralyze the target network, but to steal and seize data in order to make a monetary demand for the release of the seized data. The attacker invades the target network, plants APT malware on the victim's machine, and transforms the machine into a zombie. Malware is used by the attacker to remotely control the compromised machine and steal sensitive data, such as confidential and personal information, over a long period. Some well-known malware, such as GhostNet, Taidoor, IXESHE, and Trojan, establishes communication channels with their command and control (C&C) servers in order to transfer data between the attacker and zombie by exploiting domain name servers. Even Facebook, Google, and Twitter have been exploited as C&C servers for malware. Such malware can avoid being caught via a vaccination program using a polymorphic engine,i.e., a computer program that can be used to mutate a program into a subsequent version of itself based on different code yet operating with the same functionality. Furthermore, malware can use vulnerabilities in authenticated IP and open ports to bypass firewalls, encrypt packets, and use cloud services to make network based anomaly detection difficult. For example, secure sockets layer encryption can evade uniform resource locator patterns and hypertext transfer protocol (HTTP) header-based detection [2].

Network attacks and anomalies including APT malware propagation can be inhibited by capturing suspicious packets in the network and analyzing them using deep packet inspection (DPI), which can inspect even the payload of a single packet. DPI retains numerous packets in the memory of the inspection device until it has acquired sufficient information to match the types of packet already identified. Once a new packet has been matched to the packet list already identified by the device, it determines the application generating and sending the packet. If the application cannot be identified even after DPI inspects the packet header and payload, it examines the pattern of how packets are exchanged between hosts. This heuristic approach can effectively avoid the problem of packet inspection due to data encryption.

An intrusion detection system (IDS) is a widely used representative network security system for detecting network anomalies or attacks via the inspection of traffic on the network or system. An IDS is classified as host- or network based according to its location and as signature- or statistical anomaly-based according to the method of packet inspection. A host-based IDS detects attacks within specific systems, whereas a network-based IDS, typically located on the backbone, monitors and inspects network traffic in an effort to detect suspicious or malicious traffic. A signature-based IDS performs detection using a signature, which is a database of already known attack patterns, whereas a statistical anomaly-based IDS detects network anomalies based on defined normal behavior pattern or machine learning (ML) model, rather than signatures. Several types of IDS, such as host-, network-, signature-, and anomaly-based IDS, can be used hierarchically as a means to obtain multidimensional information for the DPI.

In order to detect network attacks or anomalies, data collection is required when analyzing network traffic using various ML methods equipped with IDS and DPI. This is because traffic analyzers such as IDS have fixed locations and a limited capacity to perform DPI for vast amounts of network traffic. Consequently, it is necessary to use a sampling and steering method to selectively capture network traffic and forward it to a destination for inspection. Sampling and steering of network traffic can be easily implemented using software-defined networking (SDN) technology. SDN provides flexibility, elasticity, and programmability by decoupling both the network control and data planes and including a centralized SDN controller, which remotely controls SDN-capable switches or routers using an OpenFlow protocol through the control plane.

In legacy networks, due to their inflexibility, it is difficult to steer network traffic to an analyzer. Using SDN, it is possible to steer network traffic to any destination via packet duplication and dynamic re-routing. Furthermore, it is possible to sample traffic from the entirety of the network traffic simply using a queue rate limit policy in the OpenFlow protocol while maintaining the total aggregate volume of the network traffic below the total traffic processing capacity of the IDS [3]. Consequently, it is not necessary to consider where the monitoring devices are located. By updating the forwarding table, we can flexibly and dynamically sample traffic and steer this sampled traffic to any destination. However, determining the sampling point and rate of network traffic monitoring remains crucial because there exists a certain probability that the potentially useful information may be lost if some traffic is discarded without being captured. Additionally, the steering of sampled traffic can cause additional traffic overheads on the networks; thus, we need to consider where to send the sampled traffic for less expensive traffic analysis.

In recent years, deep reinforcement learning (DRL) combining reinforcement learning (RL) with deep neural networks, has been adopted for intelligent network operations such as routing optimization and adaptive resource allocation. DRL has great potential for solving the automatic resource allocation problem under time-varying environments in which future information is uncertain [4], [5]. In this paper, we present an intelligent traffic sampling system for multiple traffic analyzers on an SDN-based network that decides sampling points, rates, and the traffic analyzer used for each sampling point using a deep deterministic policy gradient (DDPG) algorithm. The main contribution of this paper is summarized as follows.

- We formulate the problem of sampling points and rate decisions for multiple traffic analyzers on an SDN-capable network as a Markov decision process (MDP) considering network flow uncertainty by maximizing the long-term objective function for enhanced traffic monitoring performance.
- We adopt the DDPG algorithm to automatically determine sampling rates and traffic analyzer of each

sampling point in order to achieve less intrusive traffic monitoring while increasing the monitoring performance of the network flow.
- This approach gradually learns a better sampling policy under future network flow uncertainties to sample more malicious flows, reduce sampled traffic steering overheads, and conduct load-balancing between multiple traffic analyzers.

II. RELATED WORKS

A. TRAFFIC SAMPLING

Cisco′s NetFlow, a representative traffic sampling technique for traditional networks, installs probe modules at switches in order to collect flow-based traffic statistics, after which traffic statistics collected from each module are sent to a central collector. NetFlow collects traffic statistics with 7-tuples-based flow information, which denotes the interface, protocol, source IP, source port, destination IP, destination port, and IP type of service [6]. The sampling tool JFlow is similar to NetFlow for Juniper's network device [7]. sFlow, an industry standard for exporting packet at Layer 2 of the open systems interconnection reference model, samples data in the sFlow datagram format, and sends it to the sFlow collector. The sFlow datagram consists of two sampled components; a flow sample able to randomly sample $1/n$ packets (i.e., 1 out of each 1000 packets) and an interface counter sample that acquires interface information according to the polling interval [8]. Traffic sampling methods in traditional networks require that the sampling agent is installed at the sampling point according to the network device vendor. Generally, only limited flow information can be collected.

By virtue of its flexibility and programmability, SDN technology has become popular in the development of more efficient network monitoring schemes with higher accuracy and lower overheads. In [9], Tootoonchian *et al.* proposed a query-based monitoring system called OpenTM able to measures a traffic matrix representing the amount of data traffic between each source switch and destination switch in an SDN-capable network. In [10], Yu *et al.* proposed a link utilization monitoring tool called FlowSense, which utilizes the (OFPT_PACKET_IN) and (OFPT_FLOW_REMOVED) messages in OpenFlow protocol in order to estimate the link utilization of each flow. In [11], Chowdhury *et al.* presented a network traffic monitoring framework for the SDN controller PayLess. This led to a reduction in the network overheads caused by the conventional periodic polling method of collecting flow statistics and performs real-time information collection based on a representational state transfer (REST) interface. In [12], Queiroz *et al.* proposed a granular traffic monitoring method that collects and generates traffic statistics in real time using counter values defined by OpenFlow protocol. Yang and Yeung designed a periodic flow statistics collection scheme in SDN-capable switches to reduce the bandwidth cost of the SDN control plane and the number of flow rules on the switch for collecting flow statistics [13]. Yahyaoui and Zhani represented a flow monitoring method to minimize monitoring messages, bandwidth consumption, and respect reporting delay at the control plane. They formulated the problem of which switch would report statistics on which flow and provided a heuristic solution [14].

To improve the ability of IDS to detect malicious traffic, in [15], Ha *et al.* proposed an algorithm to determine the sampling rate per switch that minimizes the capture-failure rate of malicious flows, indicating the probability that an IDS will not recognize malicious activity in the networks. Their approach updates the sampling rates of SDN-capable switches by exploiting inspection results under a single limited IDS processing capacity. In [16], Ha *et al.* presented a suspicious flow forwarding method via load balancing based on a principal component analysis; this method is applicable for multiple IDSs, grouping flows by examining the relationship of switches in their path and forwarding all flows in the same group to the same IDS. In [17], Yoon *et al.* proposed a sampling point and rate determination method using a betweenness centrality measure and a traffic matrix for scalable network traffic sampling in SDN-capable networks. Those authors also introduced various sampling rate decision methods for selected sampling points. In [18], Wang *et al.* presented a time-based flow-aware sampling architecture for SDN-capable networks. They analyzed the spatial-temporal factors of SDN-capable switches in networks as a means for deciding sampling points and durations.

Liu *et al.* developed OpenMeasure, which determines the most informative flows for sampling using an online learning algorithm based on the multi-armed bandit. This approach selects which switch to place sampling flow rules in consideration of the limited resources of the switch for flow rules [19]. Deng *et al.* proposed a deep Q-network (DQN)-based traffic sampling framework as a means to sample more short-life time mice flows that have fewer than a certain number of packets without redundancy in mobile edge computing with SDN environment [20]. Castillo *et al.* proposed IPro, a traffic monitoring architecture using RL, which focuses on the problem of control plane overheads and extra additional CPU usage of the SDN controller [21]. Phan *et al.* proposed DeepGuard, a fine-grained flow monitoring for anomaly detection in SDN. DeepGuard adopts a double DQN algorithm to learn a flow matching strategy to maximize flow granularity on SDN-capable switches that plays an important role in anomaly detection by providing more detailed information about flows. To avoid data plane performance degradation, DeepGuard keeps the number of flow rules, which increases proportionally to flow granularity, below the capacity of SDN-capable switches [22].

B. DEEP REINFORCEMENT LEARNING

A typical ML problem that RL deals with can be described as follows. An agent searching for an environment recognizes the current state and takes an action. The agent obtains a reward, which can be either positive or negative, from the environment. The sequential behavioral decision problem can then be expressed in an MDP consisting of

state space, action space, transition probability, and reward function. The RL algorithm is used by the agent to determine a policy defined by a set of behaviors maximizing the rewards that will be accumulated in the future. An optimal value function and optimal policy of the MDP can be determined using Bellman's expectation equation and Bellman's optimality equation. The Bellman equations can be solved using dynamic programming, which has evolved into SARSA and Q-learning. Q-learning, a representative RL algorithm, is widely used because of its model-free nature and the fact that it can operate without prior knowledge of future rewards or transition probabilities of the system; it also incorporates off-policy methods that learn about optimal policies while following behavioral policies. Therefore, Q-learning is adapted for operating a system in real-time given uncertainties in future information [23].

For real world applications such as network environment management and operation, RL algorithms such as Q-learning encounter complexity problems due to the large space of state and action in MDP. To solve the curse of dimension problems caused by table value update-based Q-learning, RL has been combined with deep learning in a DRL. The objective of the DRL is to learn an optimal policy by utilizing a multi-layer perceptron (MLP)-based non-linear function approximator, which represents the probability distribution of a DRL agent's action strategies in order to maximize the expected long-term reward. DQN, a typical DRL algorithm, is widely used to solve the complex state space problem in Q-learning [24]; however, since DQN has a discrete action space, it is difficult to apply in environments for which the number of actions increases or continuous action control is required.

To solve this problem, a DDPG using an actor-critic method based on a deterministic policy gradient (DPG) algorithm was proposed. The actor-critic method comprises a critic model that updates an action-value function to maximize long-term rewards and an actor model that determines policies according to the critic model. The DPG finds the optimal deterministic policy using a policy gradient method that optimizes the parameterized policy for non-linear function approximators such as MLP via the gradient ascent algorithm. The DDPG is a model-free, off-policy, and actor-critic algorithm to solve MDP that has tremendous state and action spaces [25], [26].

III. TRAFFIC SAMPLING SYSTEM MODEL USING MDP

A. OVERALL SYSTEM ARCHITECTURE

We consider an SDN-based traffic sampling system for multiple traffic analyzers, as shown in Figure 1, that monitors network traffic using a traffic sampling scheme under the condition of unknown future information. We use a per-switch sampling approach, which configures the sampling rate, not for each flow but for each switch [17]. The reason for choosing a per-switch sampling scheme is that it is more scalable. With increasing network scale, the speed of the scale of the number of flows generally increases more rapidly than that of

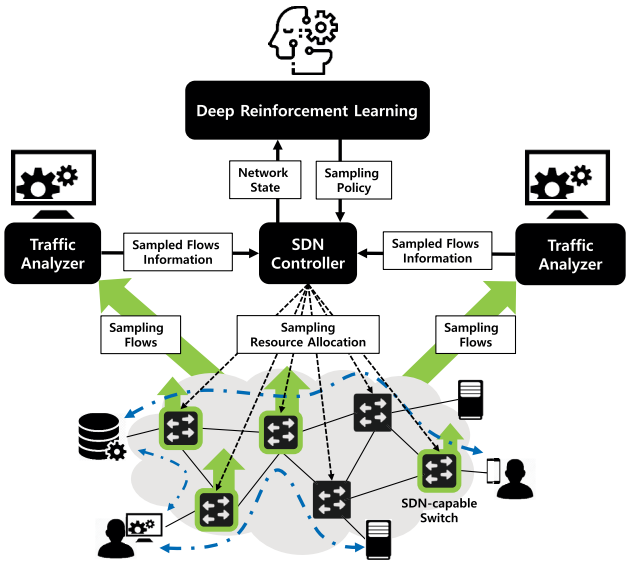


FIGURE 1. Traffic sampling system architecture with multiple traffic analyzers on an SDN-capable network.

the number of switches. A per-flow sampling approach thus requires considerably more computational time to determine the sampling rate. Additionally, it is possible to perform sampling on all switches; however, this is not appropriate for scalable sampling. Sampling by selecting specific switches instead of sampling from all switches reduces the processing overhead of the SDN controller for updating the flow table of all the switches. It is important to determine where to sample the maximum possible number of flows because the nature of malicious flows in the network environment is not known. The traffic sampling system is associated with the SDN controller, SDN-capable switches, and multiple traffic analyzers.

The proposed system includes three processes: sampling, inspection, and reconfiguration. The sampling process samples network traffic from each sampling point and steers the sampled traffic to multiple traffic analyzers. For the sampling process, one may collect every packet from the entire network. However, this approach would deteriorate the network service performance of normal data traffic. Instead, we sample a certain portion of traffic packets from a subset of switches, which will be determined by a DRL algorithm. This packet sampling is implemented by using a queue rate limit policy in OpenFlow protocol to keep maintaining the aggregate amount of the sampled traffic lower than the traffic processing capacity of traffic analyzers. Using the OpenFlow protocol, we can set the max-rate parameter of the queue and the per-switch sampling resource can be adjusted by the max-rate parameter. With port mirroring and ENQUEUE action, traffic at the SDN-capable switch can be mirrored to the queue attached to the port with a specific max-rate parameter for per-switch traffic sampling resource allocation [3]. This rate-regulated sampling process does not cause a bottleneck as done in distributed DoS (DDoS) attacks because the number of packets sampled from each switch is

appropriately decided according to the total processing capacity of traffic analyzers. Subsequently, in the inspection process, the traffic analyzer inspects the forwarded sampled traffic and reports the inspection result, containing information regarding the number of non-redundant sampled flows, to the SDN controller. Finally, during the reconfiguration process, the SDN controller adjusts the sampling rate of each switch according to the policy determined by the DRL module on the SDN controller. The entire process is conducted periodically over time.

B. SYSTEM MODEL

We assume that there are n SDN-capable switches and l traffic analyzers in the network. The SDN-capable switches and traffic analyzers are denoted by $\boldsymbol{O} = [\, o_1\ o_2\ \ldots o_n\,]^T$ and $\boldsymbol{D} = [\, d_1\ d_2\ \ldots d_l\,]^T$, respectively. The SDN controller periodically selects m sampling points among n SDN-capable switches and allocates their sampling rates. Let T denote the sampling resource allocation period. The processing capacity of traffic analyzer d_k is denoted by c_k (packets/T) and the total capacity of l traffic analyzers can then be calculated as follows:

$$\sum_{k=1}^{l} c_k = C. \tag{1}$$

Let $\boldsymbol{F^{tot}}$ be the set of total flows in the network during T and $\boldsymbol{F^{samp}}$ be the set of total non-redundant sampled flows in the network during T. Note that $\boldsymbol{F^{tot}}$ can be changed dynamically because of the uncertainty characteristics of the network flows. $\boldsymbol{F^{samp}}$ is determined periodically according to the sampling resource allocation policy. For sampling points and rate decisions, we define the ordered m-tuple of sampling points during T as $\boldsymbol{P} = (p_1,\ p_2,\ \ldots,\ p_m)$ and $p_j \in \boldsymbol{O}$. Once the sampling point is determined, the volume of the sampled traffic at each sampling point is set according to the order of the m-tuple and the sampling rate reduction ratio $r \in [0, 1)$. The sampling rate difference at each sampling point decreases by r. The number of sampling points m and the sampling rate reduction ratio r may vary depending on the network status. This enables us to define the sampling rate vector as $\boldsymbol{x} = [\, x_1\ x_2\ \ldots x_m\,]^T$, where x_j indicates the sampling rate at a sampling point p_j and the unit of x_j is the number of packets per T. Because the overall amount of sampled traffic should not exceed the total processing capacity of the corresponding traffic analyzers, the relationship between C and x is given by

$$\sum_{j=1}^{m} x_j = \frac{x_1 \cdot (1 - r^m)}{1 - r} = C, \tag{2}$$

and the amount of each sampling rate x_j is

$$x_j = \frac{1 - r}{1 - r^m} \times r^{j-1} \times C. \tag{3}$$

$\boldsymbol{G} = [\, g_1\ g_2\ \ldots\ g_l\,]^T$ is the set of groups used in traffic analyzer selection for which m sampling points are clustered into l groups. According to the traffic analyzer selection $\boldsymbol{G}$, we can obtain the resource utilization for traffic analyzer c_k during T as follows:

$$u_k = \sum_{j \in g_k} \frac{w_j}{c_k}, \tag{4}$$

where w_j is the sampling cost of p_j during T, and w_j is equal to x_j if x_j is equal to or less than the data rate of p_j during T, otherwise w_j is equal to the data rate of p_j during T. When the sampling points and traffic analyzer selection have been defined, the traffic sampling system steers the sampled traffic to the selected traffic analyzer, giving the overhead of the traffic sampling system. $\boldsymbol{v} = [\, v_1\ v_2\ \ldots\ v_l\,]^T$ is the set of steering overheads for the traffic analyzer in which the element v_k is given by

$$v_k = \sum_{j \in g_k} w_j \cdot \mathrm{hop}(p_j \rightarrow d_k), \tag{5}$$

where $\mathrm{hop}(p_j \rightarrow d_k)$ is the number of hops from the jth sampling point to the destination traffic analyzer d_k.

We assume that the traffic sampling system operates on a discrete-time basis based on a time step and gradually learns better sampling policies using reinforcement learning via continuous time steps under the condition of uncertainty in future information. The length of a time step is one time unit, i.e., the time it takes for a single state to change. In this paper, we set the time step as the sampling period T and consider two uncertainties in future information: *the occurrence of network flows* and *the data rate of these flows*. Information is available for both of these uncertainties from the initial state to the current state (t). The information for the time step $t + 1$ is not given for most real networks. Therefore, using an MDP, we need to model the system to find the optimal policy; accordingly, we used a DDPG technique as the optimization method. In the following subsections, the DDPG technique is applied to the traffic sampling with multiple traffic analyzers on the SDN-capable network, formulating the state and action space, transition probabilities, and reward function of the system as an MDP with the assumption of uncertainty.

C. STATE AND ACTION SPACE

The state space of the traffic sampling system consists of the allocated sampling resource state and the selected traffic analyzer state. Let s_t denote the state at time step t, which can be represented by the set of ordered pairs for m sampling points and the selected traffic analyzers for p_j as follows:

$$s_t = [(p_j,\ d_k)\,|\,p_j \in \boldsymbol{O}\ \textit{and}\ d_k \in \boldsymbol{D}], \tag{6}$$

where state s_t indicates that m-ordered selected sampling points with traffic analyzers are present at each sampling point. Let $\boldsymbol{S}$ denote the state space. The number of tuples in a state $s_t \in \boldsymbol{S}$ is fixed to the number of sampling points m. For example, the first tuple in s_t corresponds to the first sampling point and its traffic analyzer. These tuples are selected at each time step t from the state space, and the total size of $\boldsymbol{S}$ can be

calculated as

$$|\boldsymbol{S}| = \frac{n!}{(n-m)!} \times l^m. \tag{7}$$

Note that there are n candidate switches and m sampling points are selected. Then, the sampled traffic will be forwarded to one of l traffic analyzers.

At each time step t, tuples in state s_t can be selected by action a_t, which comprises three micro-actions. The first determines the sampling points $p_j \in \boldsymbol{O}$, the second determines which traffic analyzer to assign at each sampling point, and the third determines the reduction of the sampling rate. The set of actions at time step t is represented by

$$a_t = (p_j,\ d_k,\ r), \tag{8}$$

where action a_t indicates one of n switches with one traffic analyzer for the selected switch and a sampling rate reduction ratio r at time step t. If the agent performs action a_t in state s_t, the tuple of the sampling point and the corresponding traffic analyzer, i.e., (p_j, d_k), determined by the action a_t is selected as the first element of the next state s_{t+1}. Then, the last element of the state s_t is eliminated in the next state s_{t+1} so the number of tuples in state s_{t+1} is fixed to the number of sampling points m. The switch selected in a_t has the largest sampling rate because its sampling rate vector x is determined by the order of the sampling points. The sampling rate of the sampling points changes according to the selected sampling rate reduction ratio r.

D. TRANSITION PROBABILITY AND REWARD FUNCTION

The probability that the traffic sampling system undergoes a transition from state s_t to state s_{t+1} when action a_t is taken can be represented as $\mathbb{P}_{sa} : (s_t, a_t) \rightarrow s_{a_t}$. Since the sampling rate of each switch can be changed dynamically using an SDN controller, transition probability $\mathbb{P}_{sa}$ can be determined using the probability of selecting action a_t at state s_t defined by policy π. The comprehensive reward of the proposed traffic sampling system should be designed to achieve three main goals: (1) fair-share flow monitoring, (2) load balancing for multiple traffic analyzers, and (3) reducing sampled traffic steering overheads for multiple traffic analyzers. To achieve fair-share flow monitoring, we define the flow fair-share reward r^f as follows:

$$r^f = \frac{|F^{samp}|}{|F^{tot}|}. \tag{9}$$

For balanced utilization of traffic analyzers, we define traffic analyzer load-balancing reward r^u and adopt Jain's fairness index [27] as follows:

$$r^u = \frac{\left(\sum_{k=1}^{l} u_k\right)^2}{l \cdot \sum_{k=1}^{l} {u_k}^2}. \tag{10}$$

To consider sampled traffic steering overheads, we define normalized total sampled traffic steering overhead penalty r^v as follows:

$$r^v = \frac{|\sum_{k=1}^{l} v_k - C|}{\{max(\text{hop}(p_j \rightarrow d_k)) - 1\} \cdot C} \tag{11}$$

Let R denote the reward function that returns a cost value indicating whether the proposed system utilizes multiple traffic analyzers equally to sample network flows fairly taking into account sampled traffic steering overheads. When action a_t is taken in state s_t, the reward function is defined as follows:

$$R(s_t, a_t) = (r^f \cdot r^u) - r^v, \tag{12}$$

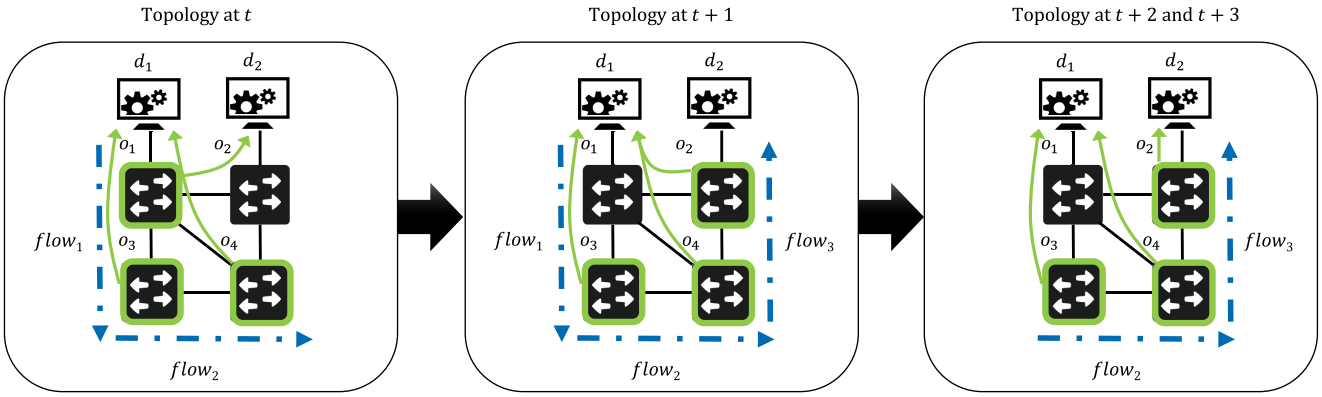
where the range of $R(s_t, a_t)$ is $[-1, 1]$, representing the performance of the proposed traffic sampling system.

To consider the impact of a current action on future rewards, we define the total expected discounted reward under policy π as follows:

$$R_t^\pi = R(s_t, a_t) + \sum_{i=1}^{\infty} \gamma^i \cdot R(s_{t+i}, a_{t+i}), \tag{13}$$

where $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards from time step $t+1$ until the infinite time step. In other words, R_t^π is defined as the sum of the current reward at time step t and the discounted reward from time step $t+1$ to the infinite time step. For example, $\gamma = 0$ denotes that the traffic sampling system considers only the current reward, whereas $\gamma = 1$ means that the system considers the long-term reward at an infinite time step with the same weight as the current reward. The objective of the traffic sampling system is to identify an action-selection policy that increases the total expected discounted reward R_t^π to increase the sampling performance for multiple traffic analyzers on the SDN-capable network.

Figure 2 and Table 1 show a simple network topology and its corresponding state, action, and reward table. The network has four switches and two traffic analyzers. The capacity of traffic analyzers c_1 and c_2 is 50 packets per second (pps), the number of sampling points m is 3, and the sampling rate reduction ratio of sampling point r during sampling period t is 0.9. At t, sampling points, traffic analyzers and the sampling reduction ratio are selected such that their state is $\{(o_3, d_1), (o_4, d_1), (o_1, d_2)\}$ and action $(o_2, d_1, 0.8)$ is taken to change the state to $\{(o_2, d_1), (o_3, d_1), (o_4, d_1)\}$. The corresponding sampling rate vector $\boldsymbol{x} = [41\ 33\ 26]^T$ can be obtained by (3) at $t+1$. At time $t+1$, a new $flow_3$ begins, enabling us to obtain the per-flow sampled data rate as follows. For o_2, the sampled data rate of $flow_3$ is 41 pps. For o_3, the sampled data rate of $flow_1$ is 11 pps and the sampled data rate of $flow_2$ is 22 pps. We assume that if more than one flow goes through the same switch simultaneously each flow is sampled proportionally according to its data rate. For o_4, the sampled data rate $flow_2$ is 10.4 pps and the sampled data rate $flow_3$ is 15.6 pps. Additionally, we can obtain the resource utilization of traffic analyzer vector u and sampled traffic steering overhead vector v using (4) and (5). Consequently, the fair-share reward r^f, load-balancing reward r^u, and steering overhead penalty r^v are 1.0, 0.5, and 1 according to (9), (10), and (11), respectively. Finally, we obtain the reward when action a_t is taken in state s_t -0.5 using (12). Similarly, at $t+1$, when taking action $(o_2, d_2, 0.7)$, the state is changed to $\{(o_2, d_2), (o_3, d_1), (o_4, d_1)\}$ and the reward is increasing to 0.45 according to (12).



- Sampling period: $T = 1\ sec$
- Capacity of traffic analyzers: $c_1 = 50$ (packets/sec, pps) and $c_2 = 50\ pps$
- Data rate of flows: $flow_1 = 100\ pps$, $flow_2$=200 pps , and $flow_3$=300 pps

FIGURE 2. Simple network topology with respect to time steps.

TABLE 1. An example of state, action, and reward table.

Time	State	Action	$\boldsymbol{x}$	$\boldsymbol{F^{samp}}$	$\boldsymbol{u}$	$\boldsymbol{v}$	$R(s_t, a_t)$
t	$[(o_3, d_1), (o_4, d_1), (o_1, d_2)]$	$(o_2, d_1, 0.8)$	$[37\ 33\ 30]^T$	$\{flow_1, flow_2\}$	$[1.4\ 0.6]^T$	$[140\ 60]^T$	-0.14
$t+1$	$[(o_2, d_1), (o_3, d_1), (o_4, d_1)]$	$(o_2, d_2, 0.7)$	$[41\ 33\ 26]^T$	$\{flow_1, flow_2, flow_3\}$	$[2.0\ 0.0]^T$	$[200\ 0]^T$	-0.5
$t+2$	$[(o_2, d_2), (o_3, d_1), (o_4, d_1)]$	$(o_2, d_2, 0.62)$	$[46\ 32\ 22]^T$	$\{flow_2, flow_3\}$	$[1.08\ 0.92]^T$	$[108\ 46]^T$	0.45
$t+3$	$[(o_2, d_2), (o_3, d_1), (o_4, d_1)]$	$(o_2, d_2, 0.62)$	$[108\ 46\ 19]^T$	$\{flow_2, flow_3\}$	$[1.0\ 1.0]^T$	$[100\ 50]^T$	0.5

IV. SAMPLING POLICY UPDATE ALGORITHM USING DEEP DETERMINISTIC POLICY GRADIENT

A. PROBLEM FORMULATION

Our goal is to identify the optimal action-selection policy leading to an increase in sampling accuracy with balanced resource utilization of multiple traffic analyzers and reducing sampled traffic steering overheads. To achieve this objective, we consider the action-value function that returns the total expected discounted reward when action a_t is taken in state s_t following policy π as:

$$Q(s_t, a_t) = \mathbb{E}\left\{R_t^{\pi}\right\} = \mathbb{E}\left\{R(s_t, a_t) + \sum_i \gamma^i \cdot R(s_{t+i}, a_{t+i})\right\}. \quad (14)$$

To approximate the optimal action-value function, Q^* is defined as

$$Q^*(s, a) = \mathbb{E}\left\{R(s_t, a_t) + \sum_i \gamma^i \max_{a_{t+i}} R(s_{t+i}, a_{t+i})|_{s=s_t, a=a_t}\right\}. \quad (15)$$

Q-learning, a representative off-policy algorithm of RL, aims to approximate this value using both behavior and target policies. It adopts the epsilon greedy method for behavior policy, i.e., $\mu(s_t) = argmax_{a_t} Q(s_t, a_t)$; however, in continuous action space, it is difficult to discriminate the optimal action from the action-value function because the agent is required to explore every Q-value, and since complexity grows exponentially according to the size of the state and action spaces.

B. DEEP DETERMINISTIC POLICY GRADIENT FOR SAMPLING RESOURCE ALLOCATION

To determine the action-selection policy that increases the total expected discounted reward defined by (13), we consider a DDPG algorithm, which has model-free, off-policy, and actor-critic characteristics. This algorithm can be used to approximate the optimal action-value function in a continuous action space using two neural networks as shown in Figure 3. The first neural network is used for approximating the action-value function, termed a critic-network. Its inputs are action and observation, and the output is the value of the action-value function, i.e., $Q(s_t, a_t)$. The second neural network is an actor-network that approximates behavior policy. Its input is observation and its output is action-value, i.e., $\mu(s_t)$. Let $Q(s, a|\theta^Q)$ and $\mu(s|\theta^\mu)$ denote the critic-network and actor-network, respectively. The network functions for the critic and actor target network are thus represented as Q' and μ', respectively.

The DDPG uses θ^Q for the critic-network and θ^μ for the actor-network to parameterize non-linear function approximators. θ^Q and θ^μ are updated according to loss function and policy gradient, respectively. The weight of critic-network θ^Q is optimized by minimizing loss as follows:

$$L(\theta^Q) = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2, \quad (16)$$

where $y_i = R(s_i, a_i) + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$. The DDPG optimizes θ^μ by updating $J(\theta^\mu)$, which is the objective function of policy gradient method in the direction

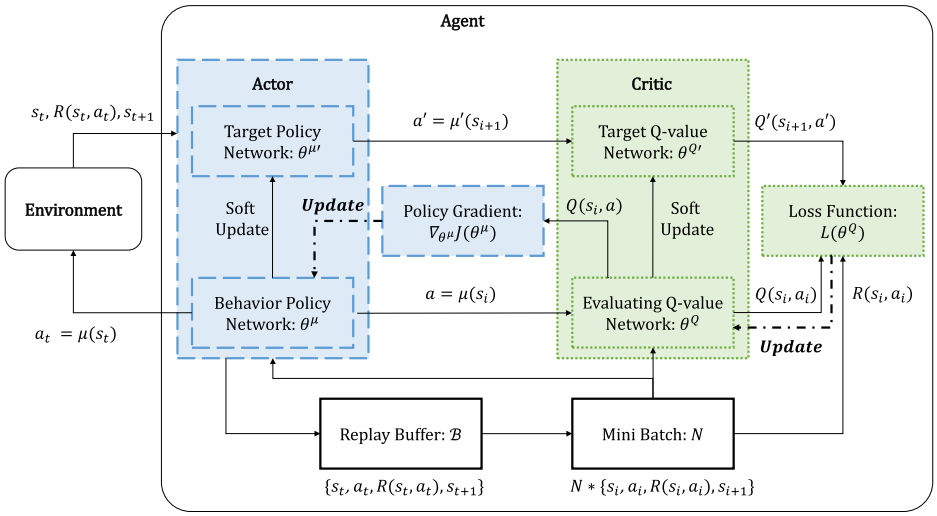


FIGURE 3. DDPG training process diagram.

of $\nabla_{\theta^\mu} J(\theta^\mu)$ defined as follows:

$$\nabla_{\theta^\mu} J(\theta^\mu) \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_i}. \quad (17)$$

To update the target networks gradually, the agent updates $\theta^{Q'}$ and $\theta^{\mu'}$ according to

$$\theta^{Q'} = \tau_c \theta^Q + (1 - \tau_c)\theta^{Q'}, \quad (18)$$

and

$$\theta^{\mu'} = \tau_a \theta^\mu + (1 - \tau_a)\theta^{\mu'}, \quad (19)$$

where τ_c and τ_a are positive small numbers between 0 and 1 used to learn rates corresponding to the critic-network and actor- network, respectively.

The proposed DDPG-based sampling policy update solution is described in Algorithm 1. The critic-network, actor-network, and target networks are initialized with experience replay buffer and initial observation in lines 1 – 5. Overall, the algorithm consists of a loop for time step t. Lines 6 – 15 show the loop for each time step t. This loop is repeated infinitely. One action a_t is selected by following the actor-network policy with adaptive noise process $\mathcal{N}$ in line 8. To assure exploration efficiency, the algorithm adds adaptive noise to the parameters of the neural network policy, not the action space [28]. In lines 9 – 10, the reward $R(s_t, a_t)$ and the next state s_{t+1} are observed by taking the selected action a_t, and the observed transition is stored as replay buffer $\mathcal{B}$. Based on randomly sampled mini batch from $\mathcal{B}$, θ^Q and θ^μ are updated according to update rules (18) and (19) in lines 11 – 14.

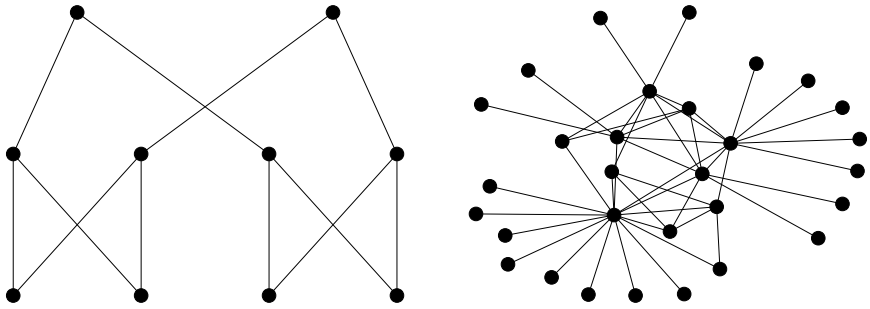
Algorithm 1 The DDPG-Based Sampling Algorithm

1: // Initialization
2: Set the critic-network $Q(s, a|\theta^Q)$ and actor-network $\mu(s|\theta^\mu)$ with randomly generated weight θ^Q and θ^μ
3: Set target networks Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
4: Set experience replay buffer $\mathcal{B}$
5: Set initial state s_0 according to the initial sampling policy
6: // Parameter updating
7: **for** each time step **do**
8: Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}$ following the parameter noise for exploration
9: Take action a_t and observe $R(s_t, a_t)$, s_{t+1}
10: Store transition $\{s_t, a_t, R(s_t, a_t), s_{t+1}\}$ in $\mathcal{B}$
11: Randomly sample a batch of N transitions $\{s_i, a_i, R(s_i, a_i), s_{i+1}\}$ from $\mathcal{B}$
12: Set $y_i = R(s_i, a_i) + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q})$
13: Update critic θ^Q and actor θ^μ in (16) and (17)
14: Update the targets softly in (18) and (19)
15: **end for**

V. PERFORMANCE EVALUATION

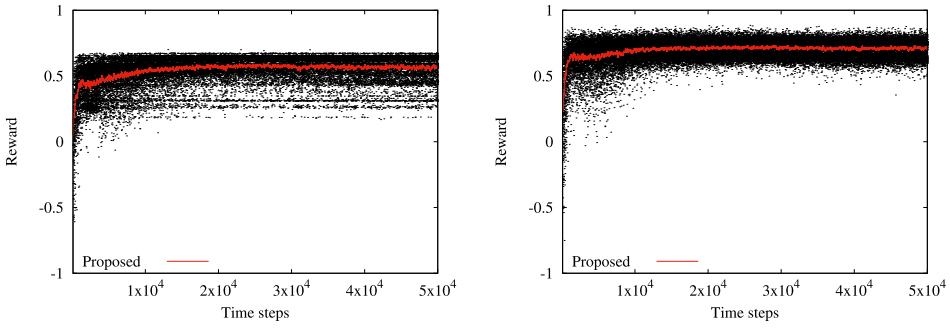
A. SIMULATION RESULTS

We have evaluated the performance of the proposed DDPG sampling algorithm by conducting simulations using a Python networkX library for network topology generation and a stable-baseline framework for DDPG algorithm [29], [30]. We considered two kinds of network topologies: a fat-tree topology for data center networks and a random Internet autonomous system (AS) topology as shown in Figure 4(a) and Figure 4(b), respectively. The fat-tree topology presents an interconnected structure between switches that do not require high-speed links or specialized equipment even when going up the tree by making all switch elements the same [31]. The random Internet AS topology resembles Internet AS network with three tiers for the AS node i.e., tier-1, a mid-level, customer or content-provider, and two types for a bidirectional link terms transit and



(a) Fat-tree data center topology with 10 switches. (b) Internet AS topology with 30 switches

FIGURE 4. Network topologies for performance evaluation.



(a) (b)

FIGURE 5. Rewards of the agent with respect to the number of time steps: a) fat-tree topology; b) Internet AS topology.

peering. The transit link provides all route information on the Internet by an AS in one direction, whereas the peering link only exchanges route information between ASs [32].

In the simulation environment, the parameter values for both topologies are summarized in Table 3. The flow data rate was set from 1 to 20 Mbps, and it was assumed that 2% of the total flows were malicious. To consider network flow uncertainty, we randomly selected source and destination switches for the shortest path routing of flows and changed the data rate of these flows at every time step with a probability of 10%. For the malicious activity, we randomly selected 2% of total flows as malicious flows for arbitrarily time steps. The processing capacity of each traffic analyzer was set to 1 Gbps. We assumed that if a packet of a malicious flow was captured by a traffic analyzer, one alert was generated for that flow.

Figure 5(a) and Figure 5(b) show the rewards of the agent with respect to the number of time steps. The reported line in both figures represents a moving average of 100 time steps. It is seen that the rewards keep increasing over the time steps in both topologies and it indicates the algorithm can choose better actions as it experiences iteration. We set the hyper-parameter values of the DDPG algorithm as shown in Table 2.

We compared the performance of the proposed sampling method against that of random sampling and flow betweenness centrality (FBC) sampling methods.

TABLE 2. Hyper-parameter values used for the DDPG algorithm.

Hyper-parameter	Value
Discount factor γ	0.99
Replay buffer $\mathcal{B}$	50,000
Mini batch N	128
Critic learning rate τ_c	0.001
Actor learning rate τ_a	0.0001

The random sampling selects sampling points, the rates of sampling at each point, and the traffic analyzer randomly at each time step. FBC selects sampling points among the switches based on betweenness centrality in graph theory and determined the sampling rate according to rate-proportional sampling, which samples traffic proportional to the data rate of each flow [17]. For traffic analyzer selection, we set the FBC to conduct greedy selection according to the sampling rate of the selected sampling points. In other words, the sampling switch with the highest sampling rate selects the traffic analyzer with the most available resources.

We evaluated three metrics, i.e., are capture-failure rate, sampled traffic steering overhead, and load-balancing for traffic analyzers as a means to compare the performance of the three sampling methods.

- Capture-failure rate indicates the probability that a packet of malicious flow cannot be captured at each switch in its path [15].

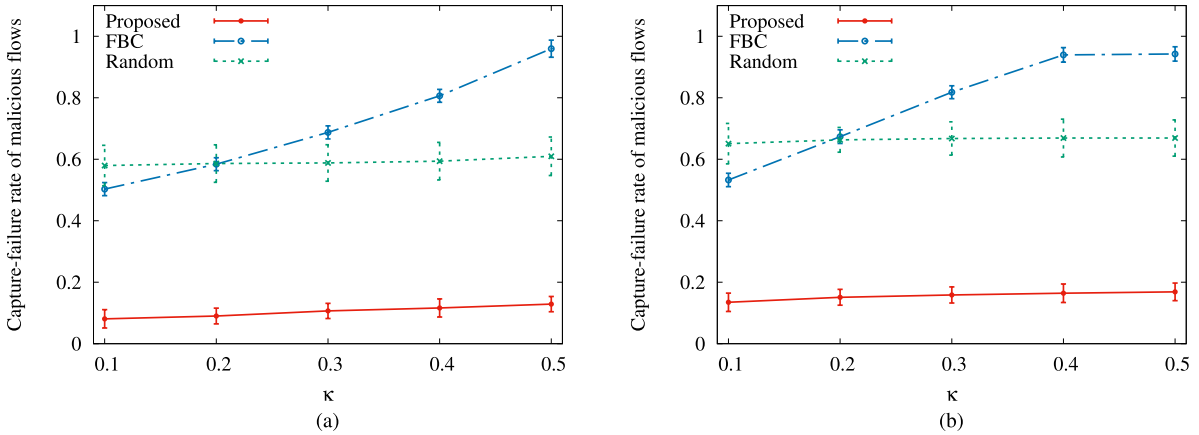


FIGURE 6. Capture-failure rate of malicious flows with respect to κ: a) fat-tree topology; b) Internet AS topology.

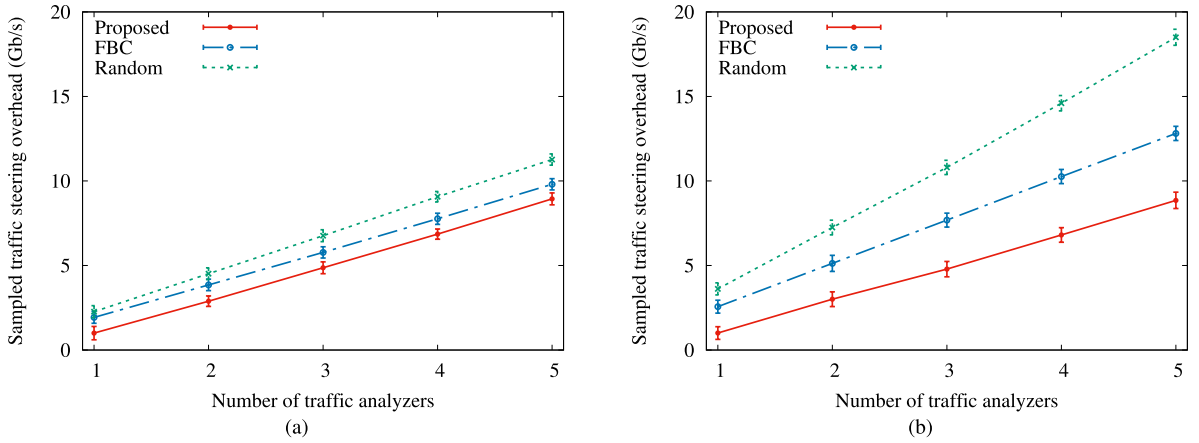


FIGURE 7. Traffic steering overhead with respect to the number of traffic analyzers: a) fat-tree topology; b) Internet AS topology.

- Traffic steering overhead represents the total data traffic volume for the sampled traffic steering from the sampling point to the traffic analyzer.
- Load-balancing is Jain's fairness index for resource utilization of multiple traffic analyzers as represented in (10).

The reported values for the performance comparison were an average of 500 iterations with a 95% confidence interval.

Figure 6(a) and Figure 6(b) show that the proposed method maintains a low capture-failure rate for malicious flows relative to other methods, even with increasing $\kappa \in [0, 1]$; this represents a ratio of switches excluded from malicious flow routing. For example, if κ is set at 0.1, any switches with a top 10% betweenness centrality value are excluded from the path when routing malicious flows. As the ratio of switches excluded from routing malicious flows increases, more malicious flows do not pass through switches with high betweenness centrality values. Consequently, the proposed method can capture malicious flows that do not pass through a switch with a high betweenness centrality value.

Figure 7(a), Figure 7(b), Figure 8(a), and Figure 8(b) show that, even with an increase in the number of traffic analyzers, the proposed method achieves lower sampled traffic steering overheads than other methods while maintaining a load-balancing of traffic analyzers over 88%. The load balancing of traffic analyzers represents Jain's fairness index of resource utilization for traffic analyzers. An index closer to 1 implies that the load is well balanced. FBCs using the greedy traffic analyzer selection method show more balanced loads for multiple traffic analyzers than the proposed method, however they show more traffic steering overheads because the FBC method allocates the load of the traffic analyzers considering only the available resources of the traffic analyzers. This demonstrates the presence of a trade-off between reducing sampled traffic steering overheads and load balancing across multiple analyzers.

TABLE 3. Parameter values used for network topologies.

Parameter	Fat-tree	Internet AS	Testbed
Number of switches n	10	30	10
Number of sampling points m	5	10	4
Number of traffic analyzers l	3	5	2
Number of flows $\lvert \boldsymbol{F}^{tot} \rvert$	3,000	50,000	10,500
Total capacity of traffic analyzers C	3 Gbps	5 Gbps	2 Gpbs
Sampling period T (= time step t)	1 sec	1 sec	5 sec

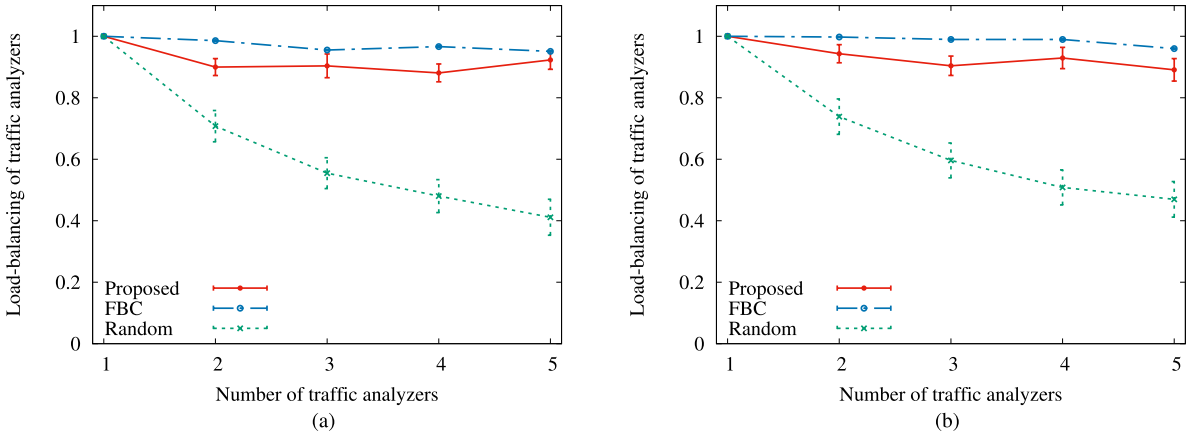


FIGURE 8. Load-balancing of traffic analyzers with respect to the number of traffic analyzers: a) fat-tree topology; b) Internet AS topology.

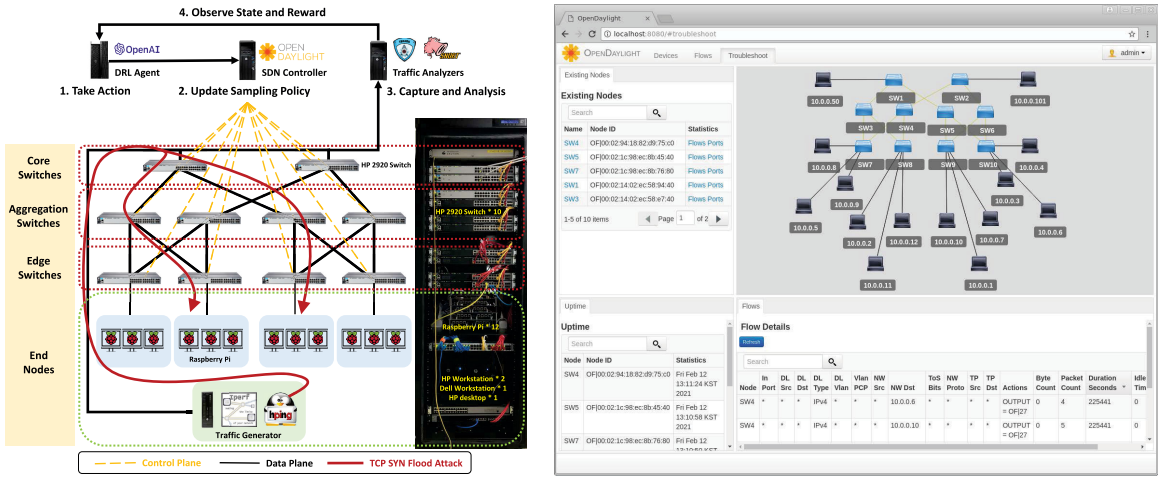


FIGURE 9. SDN testbed network topology: a) architecture; b) GUI on OpenDaylight.

B. EXPERIMENT RESULTS

To verify and empirically evaluate the proposed algorithm, we constructed an SDN testbed and implemented the proposed DRL-based traffic sampling system on a real testbed. Figure 9 depicts the architecture and graphical user interface (GUI) of the SDN testbed network topology. The testbed consists of a DRL agent, an SDN controller, two traffic analyzers, 10 HP 2920 SDN-capable switches, 12 Raspberry Pi end nodes, and a traffic generator. The component specifications of the testbed are summarized in Table 4. We implemented the DRL agent on a Dell workstation with NVIDIA TITAN Volta graphics processing unit (GPU) using OpenAI framework and the SDN controller on an HP workstation with OpenDaylight. For multiple traffic analyzers, we utilized TShark for flow statistics analysis and Snort for malicious flow detection on the other HP workstation. TShark is a command line interface-based network traffic analyzer, and Snort is a lightweight IDS for UNIX and Windows. Especially in Snort, dangerous payloads or suspicious anomalies can be detected by monitoring network traffic in real-time with DPI. We constructed a fat-tree network topology with 10 HP 2920 SDN-capable switches and 12 Raspberry Pies. For traffic generation, we used Iperf for normal transmission control protocol (TCP) flows and Hping for TCP synchronize sequence numbers (SYN) packet flooding DDoS attack on an HP desktop.

The SDN testbed operates as follows. Firstly, according to the sampling resource allocation policy calculated at the DRL agent, the OpenDaylight SDN controller updates the traffic sampling policy and samples packets from each sampling points to the traffic analyzer server. Then, the traffic analyzer server analyzes captured packets from sampling points. There are two traffic analyzers in the traffic analyzer server. We use TShark to analyze how many flows are captured and utilize Snort to detect malicious flows from captured packets. Finally, the analysis results are reported to the DRL agent for rewards.

In the testbed configuration, the parameter values for the testbed topology are summarized in Table 3. The traffic sampling resource allocation policy was updated every 5 seconds ($T = 5$ sec), and the processing capacity of each traffic analyzer was set to 1 Gbps. The maximum number of flows

TABLE 4. Components of the SDN testbed network topology.

Component	Model	Hardware	Software
DRL agent	Dell T7920	CPU: Intel Xeon Gold 6136 / RAM: 256GB / Disk: 4TB / GPU: NVIDIA TITAN Volta	OpenAI
SDN controller	HP Z420	CPU: Intel Xeon E5-1620 / RAM: 64GB / Disk: 3TB / GPU: NVIDIA Quadro K600	OpenDaylight
Traffic analyzer	HP Z420	CPU: Intel Xeon E5-1620 / RAM: 64GB / Disk: 3TB / GPU: NVIDIA Quadro K600	TShark and Snort
SDN switch	HP 2920	CPU: ARM Cortex-A9 / RAM: 1GB / NIC: 1Gbps 24ports and 10Gpbs 2ports	-
End node	Raspberry Pi 3	CPU: ARM Cortex-A7 / RAM: 1GB / NIC: 100Mbps	Iperf
Traffic generator	HP ProDesk 600	CPU: Intel Core i7-6700 / RAM: 8GB /Disk: 500GB	Iperf and Hping

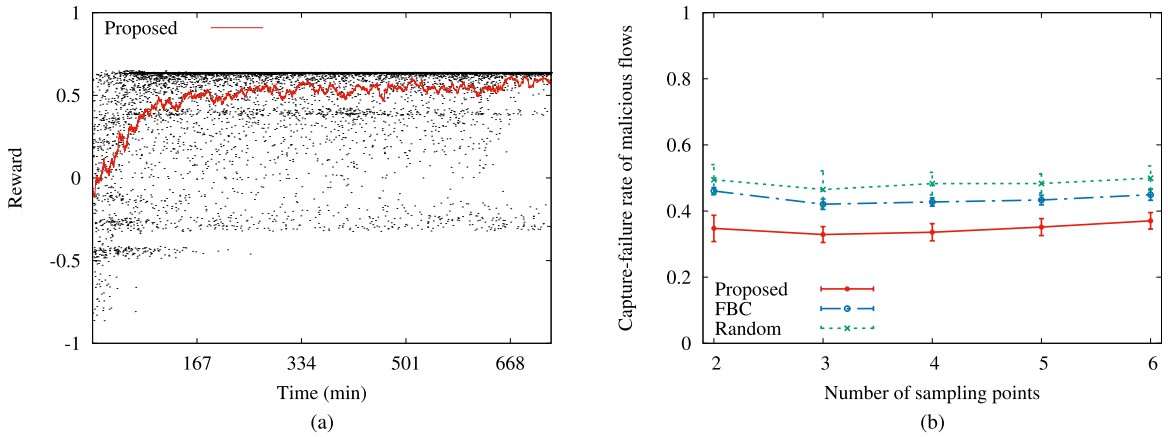


FIGURE 10. SDN testbed experiment results: a) training result; b) TCP SYN flooding attack detection result.

F^{tot} generated by the traffic generator was set to 10,500 (500 normal flows and 10,000 malicious flows). The average data rate of flows was set to 10 Mbps for normal flows and 9.6 Mbps for malicious flows, respectively. To account for the uncertainty in the malicious activity, Hping on the traffic generator created malicious flows with only a 5% probability. Total 10,000 malicious flows were generated with randomly generated source IP address arbitrarily for the victim who randomly selected one of Raspberry Pi end nodes. In the testbed configuration, if a packet of a malicious flow was captured by Snort, one IDS alert was generated for that flow.

Figure 10(a) shows the rewards of the agent with respect to the time. Each point is a measured reward at a sampling resource allocation period ($T = 5$ sec), and the line represents a moving average of 100 points. We set the hyper-parameter values of the DDPG algorithm as shown in Table 2. As shown in the figure, the DRL agent gets negative rewards at the beginning of training because it is more likely to behave randomly, but it gets more positive rewards as training time increases.

Figure 10(b) shows the average capture-failure rate of TCP SYN flooding malicious flows with respect to the number of sampling points depending on the sampling resource allocation methods. The reported values are an average of 500 iterations with 95% confidence intervals. For the proposed DDPG-based sampling resource allocation method, we utilized the trained model through 10,000 time steps ($\approx$ 833 min) for each point. As shown in the figure, even if the number of sampling points increases, the proposed method can achieve better attack detection performance compared to the other methods. In other words, the proposed method maintains a low capture-failure rate for malicious flows relative to the other methods because it can capture more malicious packets.

VI. CONCLUSION

Network traffic monitoring plays a significant role in cybersecurity, particularly in the detection of network attacks and anomalies. Using SDN functionalities, network traffic sampling is possible for multiple traffic analyzers, such as IDS using DPI. In this paper, we focused on the sampling point and rate decision problem for multiple traffic analyzers under the condition of network flow uncertainty. This problem was formulated as discrete-time MDP, where the uncertainty is the dynamic characteristics of network flows, such as data rate and flow path. We proposed a DDPG-based approach that could handle MDPs with continuous action spaces in order to adaptively solve such problems using available real-time traffic analyzers monitoring results. These simulation results demonstrate that the proposed method achieves less intrusive monitoring than either the random or FBC sampling methods, and can more accurately choose actions for capturing malicious flows following further iterations. Finally, we implemented the proposed DRL-based traffic sampling system in an empirical SDN testbed environment. We have shown that the proposed system can capture more malicious flows by learning a better sampling resource allocation policy compared to other non-learning-based sampling resource allocation methods in the SDN testbed network topology.

REFERENCES

[1] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Netw. Secur.*, vol. 2011, no. 8, pp. 16–19, Aug. 2011.

[2] G. Zhao, K. Xu, L. Xu, and B. Wu, "Detecting APT malware infections based on malicious DNS and traffic analysis," *IEEE Access*, vol. 3, pp. 1132–1142, 2015.
[3] T. Ha, S. Yoon, S. Kim, and H. Lim, "RTSS: Random traffic sampling and steering for OpenFlow-enabled networks," *ACM Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*. Irvine, CA, USA: Student Workshop, Dec. 2016.
[4] T. A. Q. Pham, Y. Hadjadj-Aoul, and A. Outtagarts, "Deep reinforcement learning based QoS-aware routing in knowledge-defined networking," in *Proc. Int. Conf. Heterogeneous Netw. Qual., Rel., Secur. Robustness*, 2018, pp. 14–26.
[5] S. Yoon, J.-H. Cho, D. S. Kim, T. J. Moore, F. F. Nelson, H. Lim, N. Leslie, and C. Kamhoua, "Moving target defense for in-vehicle software-defined networking: Ip shuffling in network slicing with multiagent deep reinforcement learning," *Proc. SPIE*, vol. 11413, Apr. 2020, Art. no. 114131U1.
[6] R. Sommer and A. Feldmann, "NetFlow: Information loss or win?" in *Proc. 2nd ACM SIGCOMM Workshop Internet Measurment (IMW)*, 2002, pp. 173–174.
[7] A. C. Myers, "JFlow: Practical mostly-static information flow control," in *Proc. 26th ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang. (POPL)*, 1999, pp. 228–241.
[8] M. Wang, B. Li, and Z. Li, "SFlow: Towards resource-efficient and agile service federation in service overlay networks," in *Proc. 24th Int. Conf. Distrib. Comput. Syst.*, 2004, pp. 628–635.
[9] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for OpenFlow networks," in *Proc. Int. Conf. Passive Act. Netw. Meas. (PAM)*, 2010, pp. 201–210.
[10] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring network utilization with zero measurement cost," in *Proc. Int. Conf. Passive Act. Netw. Meas. (PAM)*, 2013, pp. 31–41.
[11] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for software defined networks," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–9.
[12] W. Queiroz, M. A. M. Capretz, and M. Dantas, "An approach for SDN traffic monitoring based on big data techniques," *J. Netw. Comput. Appl.*, vol. 131, pp. 28–39, Apr. 2019.
[13] Z. Yang and K. L. Yeung, "Flow monitoring scheme design in SDN," *Comput. Netw.*, vol. 167, Feb. 2020, Art. no. 107007.
[14] H. Yahyaoui and M. F. Zhani, "On providing low-cost flow monitoring for SDN networks," in *Proc. IEEE 9th Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2020, pp. 1–6.
[15] T. Ha, S. Kim, N. An, J. Narantuya, C. Jeong, J. Kim, and H. Lim, "Suspicious traffic sampling for intrusion detection in software-defined networks," *Comput. Netw.*, vol. 109, pp. 172–182, Nov. 2016.
[16] T. Ha, S. Yoon, A. C. Risdianto, J. Kim, and H. Lim, "Suspicious flow forwarding for multiple intrusion detection systems on software-defined networks," *IEEE Netw.*, vol. 30, no. 6, pp. 22–27, Nov. 2016.
[17] S. Yoon, T. Ha, S. Kim, and H. Lim, "Scalable traffic sampling using centrality measure on software-defined networks," *IEEE Commun. Mag.*, vol. 55, no. 7, pp. 43–49, Jul. 2017.
[18] X. Wang, X. Li, S. Pack, Z. Han, and V. C. M. Leung, "STCS: Spatial-temporal collaborative sampling in flow-aware software defined networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 999–1013, Jun. 2020.
[19] C. Liu, A. Malboubi, and C.-N. Chuah, "OpenMeasure: Adaptive flow measurement & inference with online learning in SDN," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2016, pp. 47–52.
[20] H. Cai, J. Deng, S. Chen, X. Wang, S. Pack, and Z. Han, "Improved flow awareness by spatio-temporal collaborative sampling in software defined networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 182–194.
[21] E. F. Castillo, O. M. C. Rendon, A. Ordonez, and L. Z. Granville, "IPro: An approach for intelligent SDN monitoring," *Comput. Netw.*, vol. 170, Apr. 2020, Art. no. 107108.
[22] 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 Trans. Netw. Service Manage.*, vol. 17, no. 3, pp. 1349–1362, Sep. 2020.
[23] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, vol. 135. Cambridge, MA, USA: MIT Press, 1998.
[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602
[25] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 387–395.
[26] 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," 2015, *arXiv:1509.02971*. [Online]. Available: http://arxiv.org/abs/1509.02971
[27] R. K. Jain, D. M. W. Chiu, and W. R. Hawe, *A Quantitative Measure of Fairness and Discrimination*. Hudson, MA, USA: Eastern Research Laboratory, Digital Equipment Corporation, 1984.
[28] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," 2017, *arXiv:1706.01905*. [Online]. Available: http://arxiv.org/abs/1706.01905
[29] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proc. Python Sci. Conf. (SciPy)*, 2008, pp. 11–15.
[30] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," 2018. [Online]. Available: https://github.com/hill-a/stable-baselines
[31] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM Conf. Data Commun. SIGCOMM*, 2008, pp. 63–74.
[32] A. Elmokashfi, A. Kvalbein, and C. Dovrolis, "On the scalability of BGP: The role of topology growth," *IEEE J. Sel. Areas Commun.*, vol. 28, no. 8, pp. 1250–1261, Oct. 2010.

SUNGHWAN KIM (Student Member, IEEE) received the B.S. degree from the School of Computer Science and Engineering, Dongguk University, Seoul, South Korea, in 2015, and the M.S. degree from the School of Electrical Engineering and Computer Science (EECS), Gwangju Institute of Science and Technology (GIST), Gwangju, South Korea, in 2017, where he is currently pursuing the Ph.D. degree. His research interests include software-defined networking (SDN), deep reinforcement learning (DRL), cyber-security for various network domains, and next-generation firewall.

SEUNGHYUN YOON (Graduate Student Member, IEEE) received the B.S. degree from the School of Computer Science and Electrical Engineering, Handong Global University, Pohang, South Korea, in 2016. He is currently pursuing the Ph.D. degree with the School of Electrical Engineering and Computer Science (EECS), Gwangju Institute of Science and Technology (GIST). He is currently a Visiting Scholar with the Department of Computer Science, Virginia Tech. His research interests include software-defined networking (SDN), deep reinforcement learning (DRL), cyber-security for various network domains, and moving target defense (MTD).

HYUK LIM (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the School of Electrical Engineering and Computer Science, Seoul National University, Seoul, Republic of Korea, in 1996, 1998, and 2003, respectively. From 2003 to 2006, he was a Postdoctoral Research Associate with the Department of Computer Science, University of Illinois at Urbana–Champaign, Champaign, IL, USA. He is currently a Full Professor with the AI Graduate School and jointly with the School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology (GIST), Gwangju, Republic of Korea. His research interests include wired and wireless networks, cyber-security, and artificial intelligence.