IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON INNOVATION AND APPLICATION OF INTELLIGENT PROCESSING OF DATA, INFORMATION AND KNOWLEDGE AS RESOURCES IN EDGE COMPUTING

Received February 28, 2020, accepted March 29, 2020, date of publication April 21, 2020, date of current version May 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2989353

A Method for Deploying Distributed Denial of Service Attack Defense Strategies on Edge Servers Using Reinforcement Learning

HAODI ZHANG[1], JIANYE HAO[2], AND XIAOHONG LI[1], (Member, IEEE)
[1]Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University, Tianjin 300350, China
[2]College of Intelligence and Computing, Tianjin University, Tianjin 300350, China

Corresponding author: Xiaohong Li (xiaohongli@tju.edu.cn)

This work was supported by the National Science Foundation of China under Grant 61872262 and Grant 61572349.

ABSTRACT Cloud-based filtering, as the most commonly used distributed denial of service attack mitigation method in the industry, has flaws that can cause privacy leaks and delays like other cloud applications. A new DDoS mitigation method which moving cloud filtering services to edge servers is proposed in this paper. In this method, the edge servers are deployed at various router locations and run classifiers to filter the traffic passing through. For cutting attack traffic, reserving user traffic and reducing inspection delays, a novel deep reinforcement learning framework is developed to balance the deployment of computing resource and tasks allocation, in which graph neural network used to coding the network structure information transformation as vector, and the traffic information to input into Q-Network to obtain the best allocation results. The simulation experiments show that our method has advantages in optimizing effects and computing time compared with other deployment methods.

INDEX TERMS Edge computing, distributed denial of service attack, reinforcement learning, graph convolutional network.

I. INTRODUCTION

With the advent of the Internet of Everything, more and more devices are connected to the Internet, and they produce more data. These explosive growth data have brought huge convenience to people's lives, brought huge challenges to the existing network systems as well, especially network security. One of the most serious danger in the current network is caused by distributed denial of service(DDoS) attacks(e.g. [1], [2]), which target the availability of a system or service. A DDoS attack is a traffic attack which exhausts resources(bandwidth or service) of the target. An attacker controls a large number of hosts (called bots) that send attack traffic when they receive instructions. The victim of the attack can't provide services to its legitimate users or customers, so it will suffer financial loss and doubt caused by their user's dissatisfaction and loss of trust.

DDoS attacks are difficult to defend. This is because DDoS attack traffic is not only attack traffic caused by network system vulnerabilities (protocols, physics), but also a large amount of ''legal'' traffic that meets system regulations. Although these ''legal'' traffic still has characteristics that prove it to be attack traffic, the system cannot intercept it using existing methods. Cloud-beased filtering is the most used DDoS attack mitigation method today. It redirects the traffic that reaches the protection target to the cloud server, intercepts the attack traffic by the cloud traffic filtering program, and returns the remaining traffic back. This solution has some drawbacks such as privacy violation and latency. At the same time, such services will also be the target of DDoS attacks. Edge computing is a new type of computing model [3] using arbitrary computing and network resources from the data source to the computing center to operate on uplink and downlink data. Compared with cloud computing, it greatly reduces system latency [4] and protects the privacy of service objects [5].

There are two most important DDoS response methods, throttling(e.g. [6]–[8]) and filtering(e.g. [2], [9]). According to the characteristics of attack packets, filtering method separates them from normal network traffic and deletes them.

The associate editor coordinating the review of this manuscript and approving it for publication was Honghao Gao.

Throttling method is to restrict the abnormal traffic in the network, or to restrict the access of the network traffic with the characteristic under the condition of determining the attack characteristics. The filtering method, which is based classier, introduces inspection delay. When the traffic is too large, the delay can be terrible because it is related to the quadratic of the traffic size. Router Throttling is more focused on securing servers and services, and the way it handles traffic does not cause a lot of latency. However, this method will cause a large amount of attack traffic to flow into the server and legitimate traffic will be discarded incorrectly, as it intercepts traffic by probability rather than the traffic characteristics. In order to take advantage of the two methods, we should use a classifier to intercept traffic and avoid deploying it on routers where traffic is concentrated.

The location of the DDoS defense strategy is also important. The network can be thought of as a tree with the target server as the root. The traffic flows from leaf nodes and is collected at the root node. Usually, we will deploy the classifier on the server side, or redirect the aggregated traffic to the cloud server for classification. Because of the computational overhead associated with deploying a defense strategy, this is a vulnerability for DDoS attacks that generate a large amount of traffic [10]. It can lead to degradation of protected services. Therefore, a way to classify traffic at distributed points is a good way to mitigate DDoS attacks. It can classify traffic before it is aggregated by deploying a classifier upstream, or reduce the amount of downstream traffic.

What computing resources in the network should we use to classify traffic? Obviously, it is related to network structure and real-time traffic. Computing resource allocation is an important issue for edge computing ([11], [12]). In order to reduce duplicate traffic inspections and reduce latency, we need a good defense resource allocation system that can dynamically select important locations to deploy classifiers based on changes in traffic.

In this paper, we model a distributed deployment classifier as a graph-based combinatorial optimization problem. Calculating the optimal solution using the exhaustive method is computationally inefficient. The greedy algorithm can get an acceptable approximate solution, but for real-time systems, its response time is difficult to meet the real situation. We propose a classifier deployment method based on reinforcement learning, which uses GCN to learn network structure information, and selects the optimal modification scheme in the current situation at each step until all nodes are selected. To demonstrate the performance of our method, we conduct extensive simulation experiments. First, by comparing the greedy method, we prove that we can get a good approximate solution and significantly reduce the calculation time. Secondly, by comparing distributed throttling methods, it is proved that deploying a classifier can obtain better benefits. Finally, we compare with several random distributed deployment methods to prove our advantage in deployment strategy.

II. RELATED WORK

A. EDGE COMPUTING AND THROTTLING

There have recently been studies on DDoS mitigation using edge computing. Reference [13] modified the basic framework of edge computing to mitigate DDoS attacks. Reference [14] provides effective DDoS prevention solutions by multi-access edge computing (MAEC). The focus of these efforts is to improve existing DDoS identification and response methods and improve their performance. A strategy to deploy these response methods in place is still missing.

Router Throttling using reinforcement learning is a novel approach. It deploys Throttlings on routers on various paths and uses reinforcement learning to learn the optimal throttling rate to avoid server overload. But it is difficult to use for high-quality service because it randomly discards parckets based on probability. It provides us with a useful idea that we can clean the traffic in a distributed way, so we can deal with the triffic before it is aggregated.

B. STRATEGY ASSIGNMENT ON GRAPH

The network interdiction problem is well studied [15]. To interdict an illegal network flow, Guo *et al.* introduce the Network Flow Interdiction Game(NFIG) [16] model, a stackelberg security game, to allocate a fixed number of security resources on the network. Then, they study repeated network interdiction games with no prior knowledge of the adversary and the environment, which can model many real world network security domains [17]. Their methods can be used to intercept smuggling. Unlike guns and drugs, illegal traffic in virtual networks is harder to detect and there are more ways to check for interceptions. Therefore, the choice we need to make is not whether to deploy defense strategy, but what kind of defense strategy to deploy. And the smuggling problem is not very sensitive to the defensive strategy calculation time.

Most related is the recent work by Vaněk *et al.* [18] since they formulate the problem, how to allocate resource for malicious packet detection in computer networks, as a graph-based security game with multiple resources of heterogeneous capabilities and propose a mathematical program for finding optimal solutions. They focus on detecting malicious activity by checking packets and intercepting them. Unlike their work, Our goal is to minimize the amount of attack traffic that reaches a protected target, not to prevent some packets from escaping. Then they treat network latency as a constraint rather than an optimization goal, but in DDoS attack, it is also the target of the attacker to degrade the service quality. As with most DDoS defenses, we focus on the security of a critical target (server or host) rather than multiple targets of varying importance. Thus, the methodology proposed in existing works cannot be directly applied to our setting.

C. COMBINATORIAL OPTIMIZATION ALGORITHMS OVER GRAPHS

There has been some seminal work on using deep learning to learn heuristics for combinatorial problems, including

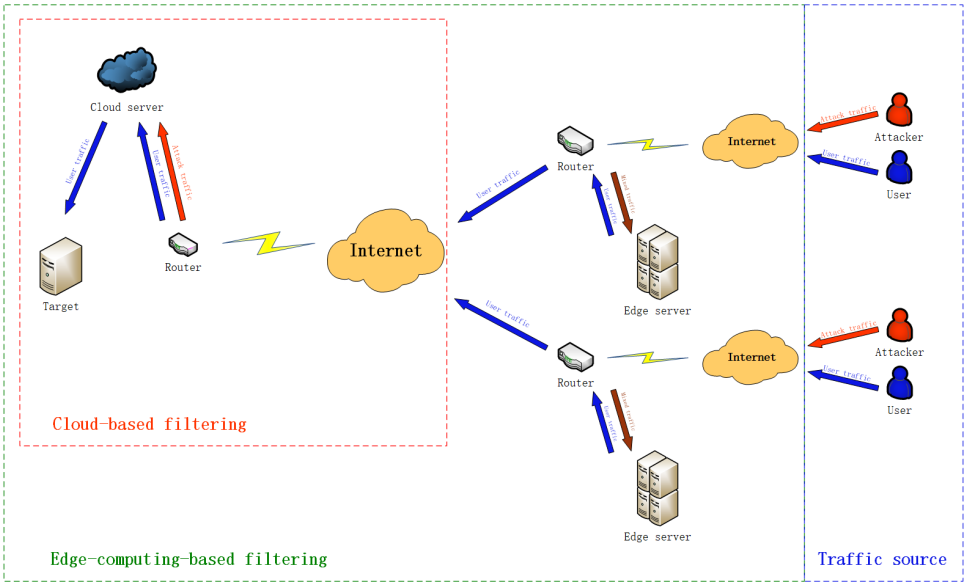

FIGURE 1. Schematic diagram of two defense methods. The cloud-based filtering method only redirects traffic at the user interface, resulting in a large amount of traffic accumulation.

the Traveling Salesman Problem(e.g. [19]–[21]). In [22], Dai *et al.* propose a unique combination of reinforcement learning and graph embedding to address NP-hard combinatorial optimization problems.

The learned greedy policy behaves like a meta-algorithm that incrementally constructs a solution, and the action is determined by the output of a graph embedding network capturing the current state of the solution. Though our problems are not the same as TSP (Traveling Salesman Problem), we can still use their ideas to make our models more robust and faster to compute.

III. PRELIMINARIES

As shown in Fig.1, cloud-based filtering redirects traffic from the network to the cloud server, which filters attack traffic with a preset classification method, and then sends the remaining user traffic to the protection target. It is currently the most commonly used DDoS defense service, which has flaws that can cause privacy leaks and delays like other cloud applications. To overcome the above problems, we put classifier in the edge server, routers in the network directly use the edge server deployed at this point to classify and send the remaining traffic to the next node. Those routers can be controlled by the defender or a third-party ISP that provides services to the defender. Obviously, using cloud services can be seen as deploying filters at the root node. Users and attackers are the source of the traffic, and like other works, it is represented by the average rate of traffic. Traffic follows the path from the user (attacker) to the target server, and if they encounter a classifier, they will be inspected. Those that are determined to be legitimate will pass through the edge server and send to the next node. They also include traffic from attackers that are misjudged, and the traffic of legitimate users may also be intercepted incorrectly, and the inspection of traffic will cause a certain delay. These will reduce the quality of the target server service.

We consider the network as a directed tree-like graph $G = (V, E)$, with each vertex $v \in V$ representing a router where defense strategies deploy and each edge $e \in E$ is the path in the routing table. r is the root of the graph G, which is the protection target. We can deploy the classifier on any node in graph G. As shown in Fig.1, the problem we are focusing on is single-target DDoS mitigation. Although the network is very complex, we can still simplify it into a tree. This is because the routing table is relatively stable, and traffic to the destination server can flow in along a fixed path. Since the focus of our work is on computational resource allocation, we do not consider classifier performance and only simulate its effects. We define a classifier as a 3-tuple $c = (a, l, d)$, which denotes the performance of a defense strategy. The parameter $a(l) \in [0, 1]$ represents the proportion of adversary(legal) triffic that classifier can interdicte. We define $C =< c_i >$ as the set of all optional classifiers. $O_v =< o_i >$ is the set of options on node v, where $o_i \in \{0, 1\}$ and $o_i = 1$ indicates that the i-th classifier is selected. For node v, the percentage of adversary(legal) triffic intercepted is $a_v = \prod_{i=1}^{|O_v|}(a_i)^{o_i}(l_v = \prod_{i=1}^{|O_v|}(l_i)^{o_i})$. The parameter $d \in [0, 1]$ denotes the factor of penalty caused by delay for detecting, and $d_v = \prod_{i=1}^{|O_v|}(d_i)^{o_i}$ is the factor on node v. A defender strategy $S =< O_v >$ is an allocation of all options in the nodes set V. Obviously, the number of deployment options available to the defender is $|O_v|^{|V|}$, which grows exponentially as the number of nodes increases.

All leaf nodes are traffic soures. They are not terminal equipment, but the boundaries of the systems we defend. Traffic is the sum of all traffic connected to this leaf node. Denote $L = \{v|v \in V, D_v = \phi\}$ as the set of all leaf nodes

and D_v as the set of child nodes of the node v. A variable $t\_a_v^{in}$ is the attack triffic that flows into the node v, i.e.,

$$t\_a_v^{in} = \begin{cases} t, & v \in L \\ \sum_{i \in D_v} t\_a_i^{out}, & v \notin L \end{cases} \tag{1}$$

where t is the triffic coming from ouside the system, and $t\_a_i^{out}$ represents the triffic which flows out from the node i, i.e. $t\_a_v^{out} = a_v t\_a_v^{in}$. $t\_l_v^{in}$ has a similar definition, i.e.,

$$t\_l_v^{in} = \begin{cases} t, & v \in L \\ \sum_{i \in D_v} t\_l_i^{out}, & v \notin L \end{cases} \tag{2}$$

The change in traffic reflects the effect of the deployment strategy. Obviously, the triffic that flows into the protected targets are $t\_a_r^{out}$ and $t\_l_r^{out}$.

There are three objectives: cutting attack triffic, reserving legal triffic as much as possible and reducing inspection delays. When the triffic that flows into the network is determined, our target is a function related to the defense strategy. Let $R_1(S)$ denotes whole flows that reach the protection target, i.e. $R_1(S) = t\_a_r^{out}$. Denote $R_2(S)$ as triffic that are mistakenly interdicted, i.e. $R_2(S) = \sum_{v \in L} t\_l_v^{in} - t\_l_r^{out}$. For the node v, its penalty caused by delay for detecting is $d_v^i(t\_a_v^{in} + t\_l_v^{in})$, and the whole penalty $R_3(S) = \sum_{v \in V} d_v(t\_a_v^{in} + t\_l_v^{in})$. The global goal R is defined as:

$$R(S) = \alpha R_1(S) + \beta R_2(S) + \gamma R_3(S) \tag{3}$$

where α, β, and γ are the weights of three goals. Obviously, the defender want to find a best strategy O to minimize the global goal R.

IV. SOLUTION APPROACH

In order to get the optimal solution of (3), we need to search the entire combined space. Because this problem has exponentially large numbers of combinations, approximation algorithms are often used to find approximate solutions. The greedy algorithm is a common method for solving such problems. We can use it to get an approximate solution and reduce the calculation time. Starting from an empty set $S = \phi$ (None of the nodes deployed classifiers), we iteratively assign the best classifer i^* on the best location v^*, which brings the minimal reward $R(S \cup (v^*, i^*)) - R(S)$, until all nodes are assigned or all rewards are positive. Although this method can obtain approximate solutions, it still cannot meet the requirements of real-time systems. When the number of nodes is very large, it can be computationally intensive.

In order to solve these problems, we have designed a method based on GCN and reinforcement learning, which can learn the structural information of the graph and solve it in a similar way to the greedy algorithm.

Our reinforcement learning framework is illustrated in Fig. 2. The network information is divided into two categories: network structure and network traffic. The graph structure information is extracted through the method of GCN. Then combine them with the traffic information to get the node vector. The input of the Q-network is the node vector and the output is Q-value. We select the 2-tuple (v^*, i^*) with the smallest q-value for deployment. This method can greatly reduce the computation time.

A. GRAPH EMBEDDING

Since we are optimizing over a tree-like graph G, we expect that the Q-function should take into account the graph structure and the current partial solution S. So, $\sum_{o_i \in O_v} o_i = 1$ for all nodes $v \in S$, and the nodes are connected according to the graph structure. In this way, when calculating the optimal classifier in the node v, not only the traffic of the node, but also the information of other nodes should be considered. Due to both state and node information are complex, hard to describe in closed form. We will use a Graph Convolutional Network(GCN) method to embed the graph G.

An intorduction of the method that is used to embed the tree-like graph is provided here. This graph embedding network will compute a p-dimensional feature embedding μ_v for each node $v \in V$. More specifically, we use a recursive method to calculate the node representation μ_v. In each iteration of calculation, the node v receives information from its children nodes and calculates the latest μ_v as:

$$\mu_v^{(t+1)} \leftarrow F(O_v, \{\mu_u^{(t)}\}_{u \in D_v}; \Theta), \tag{4}$$

where D_v is the set of children of node v in graph G, and F is a nonlinear function like a neural network. The initial embedding $\mu_v^{(0)}$ at each node is 0.

Based on the (4), we know that the node embedding update process is based on the graph topology and the embedding from the previous round. A node only gets the information from its children nodes. Obviously, only if there are enough iterations, the node can get all the structural information of a subtree which rooted at it.

Now, we discuss the parameterization of (4). We design F to update a p-dimensional embedding μ_v as:

$$\mu_v^{(t+1)} \leftarrow relu(\theta_1 O_v + \theta_2 \sum_{u \in D_v} \mu_u^{(t)}), \tag{5}$$

where $\theta_1 \in \mathbb{R}^{p \times |O_v|}$ and $\theta_2 \in \mathbb{R}^{p \times p}$ are the model parameters, and $relu$ is the rectified linear unit ($relu(x) = max(0, x)$) applied elementwise to its input. The 0-1 vector O_v represents the option of the node v.

B. PARAMETERIZING Q-FUNCTION

When the embedding is finished, we will use them to calculate the Q-function. More specifically, we use μ_v for node v and the sum of all nodes embedding $\sum_{v \in V} \mu_v$ represents the partial solution $p(S)$. In order to calculate the Q-value of each classifier of each node, we also need the trafiic information. So, we define the Q-function $Q(p(S), v; \Theta)$ as follows:

$$Q(p(S), v; \Theta) = \theta_3[relu([\theta_4 \sum_{u \in V} \mu_u, \theta_5 \mu_v],), t\_a_v^{out}, t\_l_v^{out}], \tag{6}$$

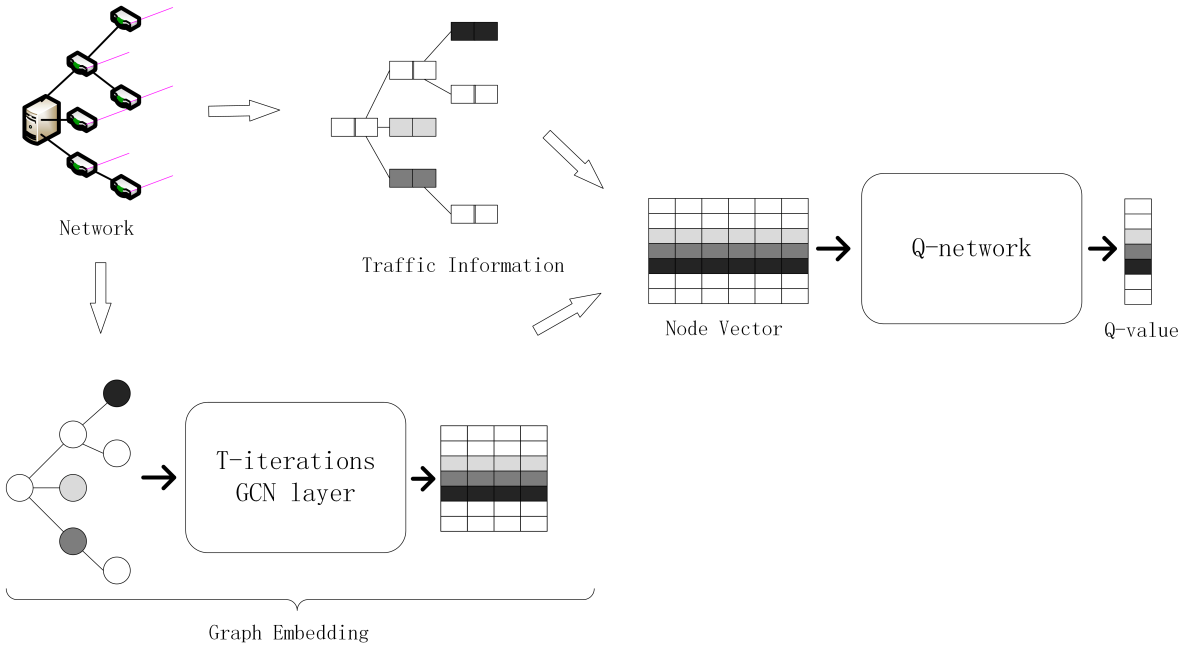

FIGURE 2. Reinforcement learning framework. Separately extract network structure information and flow information, and use them as input to the Q-network.

where $\theta_3 \in \mathbb{R}^{|O_v|\times(2p+2)}$ and $\theta_4, \theta_5 \in \mathbb{R}^{p\times p}$ are the model parameters. $[\cdot\,,\cdot\,]$ is the concatenation operator. The result of (6) is the $|O_v|$-dimensional vector, which is the Q-value of all the classifiers on node v. It is difficult to train parameters $\Theta = \{\theta_i\}_{i=1}^{5}$ due to the lack of training labels. So, we train these parameters using reinforcement learning.

C. Q-LEARNING

We show an approach that uses reinforcement learning to learn the best combination of a network structure with varying traffic by learning the greedy algorithms. The Q-function defined in the previous section is naturally applicable to the reinforcement learning (RL) formulation [23].

We define the states, actions and rewards in the reinforcement learning framework as follows:

- *State*: a state S is a set of current options of all nodes(including not selected). Since our embedding μ_v has already contained the option information, we can denote the state as a p-dimensional space, $\sum_{v\in V}\mu_v$. We can see that such representations can be used in different tree structures;
- *Actions*: the action (v, i) is a 2-tuple, where v is a node of G without a classifier deployed and i is a classifier. We represent actions as their corresponding p-dimensional node embedding μ_v. Due to the number of classifiers is determined, the result of Q-function is the set of Q-value of each classifier;
- *Transition*: transition is deterministic here. For a gaven action (v, i), we set the node v to use the i-th classifier;
- *Rewards*: our algorithm learns the greedy algorithm, so the reward function $r(S, v, i)$, that the action (v, i) is taken at state S, is defined the same way as it, i.e.

$$r(S, v, i) = R(S \cup (v, i)) - R(S), \qquad (7)$$

  and $R(\phi) = \alpha R_1(S)$;
- *Policy*: based on Q-function, a deterministic greedy policy $\pi((v, i)|S) := \arg\min_{v\notin S,i} Q(S, v, i)$ will be used. Choosing action (v, i) corresponds to adding a node of G to the current partial solution, and getting a reward $r(S, v, i)$.

We will use an n-step Q-learning method to learn the approximate optimal solution under different triffic.

In order to perform end-to-end learning of the parameters in $Q(S, v, i; \Theta)$, we use a combination of DQN [24], as illustrated in Algorithm 1. To choose a node without a classifier to perform classifier selection under the current combination and get a better reward we turn the combinatiorial problem into an n-step sequence dicision problem. To solve sequential decision problems we can learn estimates for the optimal value of each action, defined as the expected sum of future rewards when taking that action and following the optimal policy thereafter.

For a given network structure, we use L different inputs traffic for training. We use *episode* to represent a complete sequence of classifier selections from an empty state S to the terminal state S'. And a *step* within an episode is an action selection process. Each step, DQN updates the function approximator's parameters by performing a gradient step to minimize the squared loss:

$$(y - Q(S \cup (v^*, i^*), v^*, i^*; \Theta_1))^2, \qquad (8)$$

Algorithm 1 Q-Learning for the Greedy Algorithm

```
Initialize experience replay memory M = φ;
Initialize random parameters Θ1 and Θ2 = Θ1;
Input the network structure G;
time = 0;
for time < K do
    Set the triffic in network G;
    episode = 0;
    for episode < L do
        Initialize the state S = φ;
        step = 0;
        Θ2 = Θ1;
        for step < T do
            if random(0, 1) < ε then
                (v, i) = arg min_{v∉S} Q(S, v, i; Θ1);
            else
                random node v ∉ S;
                random classifier i;
            end
            r = R(S ∪ (v, i)) − R(S));
            S = S ∪ (v, i);
            Add tuple (S, (v, i), r, S ∪ (v, i)) to M;
            if t ≥ N then
                Sample random batch
                (S', (v', i'), r', S' ∪ (v', i')) from M;
                if step == T − 1 then
                    y = r'
                else
                    y = r' + γ min_{(v*,i*)} Q(S' ∪
                    (v', i'), v*, i*; Θ2);
                end
                Update Θ1 by SGD over (8);
            end
        end
    end
end
return Θ1;
```

where Θ_1 is the parameter of the eval network, and $y = r(S, v, i) + \gamma \min_{(v,i)} Q(S \cup (v, i), v, i; \Theta_2)$ for a state S except the terminal S'. Θ_2 is the parameter of the target network, it is updated at the beginning of every episode by the eval network, i.e. $\Theta_2 = \Theta_1$. In order to break the correlation between learning data, we use *experiencereplay* to update the function approximator with a batch of samples from a detaset M. The Memory M is populated during previous step. Instead of performing a gradient step in loss of the current result, stochastic gradient descent updates are performed on a random sample of tuples drawn from M.

V. EVALUATION

The purpose of the experiment is to demonstrate that our method can achieve losses similar to the greedy algorithm and effectively reduce computation time. We also compared the throttling method to prove that our method is based on the throttling method considering multiple losses. When evaluating the solution quality on an instance of a different number of nodes, we use the approximation ratio of each method relative to the benchmark solution. The approximation ratio of a solution S to a problem instance G is defined as The definition of an approximation ratio of a problem instance G in the triffic state T is $\mathcal{A}(G, T) = \frac{\mathcal{O}(G,T)}{\mathcal{B}(G,T)}$, where $\mathcal{O}(G, T)$ is the objective value and the $\mathcal{B}(G, T)$ is the best solution to the benchmark method.

A. EXPERIMENT SETUP

To simulate a DDoS attack on a single target, we designed a simulation program to meet the experimental requirements. It regards the internal processing of the router as a black box, only paying attention to the traffic size, omitting the specific details that do not impair the experimental evaluation. Specifically, we see the network as a tree-like graph with each router as a node. For each node, we only care about the size of the incoming and outgoing traffic and the properties of the classifier. This is enough to prove the function of our method. At the same time, this analog method is scalable, for example, we can increase the bandwidth of the link to simulate the link blocking situation.

Our experimental setup is simliar to the one by (Yau). The bandwidth and traffic rates are measured in Mbit/s. In our simulation program, leaf nodes generate traffic from external systems, and the triffic of other nodes flows from their children. The triffic which flows from the node is ralated to incoming traffic and the properties of the classifier. Since we are concerned with the overall effect, we use average traffic instead of instantaneous rate. Each leaf node receives traffic at a constant rate, and the legal and attack traffic rates are evenly seledced in [0, 30] and [30, 300]. In each episode, the received traffic is different.

The optional defense strategy for each node is no defense policy and two different classifiers. We use the three values of interception rate, false positive rate and delay penalty to represent the attributes of the classifier. The probability of two classifiers intercepting attack traffic is 0.95 and 0.9, and the probability of error interception is 0.12 and 0.1. At the same time, the penalty factors for delay are 0.001 and 0.0005. We assign appropriate values for the three parameters α, β, γ. In our experiments we set $\alpha = 0.6$, $\beta = 0.3$ and $\gamma = 0.1$. This is because we assume that attack traffic reaches the server as the biggest threat, and secondly we need to reduce the loss and delay of legitimate user traffic.

B. THROTTLING METHOD

We compare our approach to the throttling method of (Kleanthis Malialis). We use neural networks instead of the tile coding methods they use. We use neural networks instead of the tile coding methods, for the representation of the continuous state space. The input to the neural network is the flow information 2-tuple (a_i, l_i), where $a_i(l_i)$ is the attack(legal) traffic which flows into the node i. There are

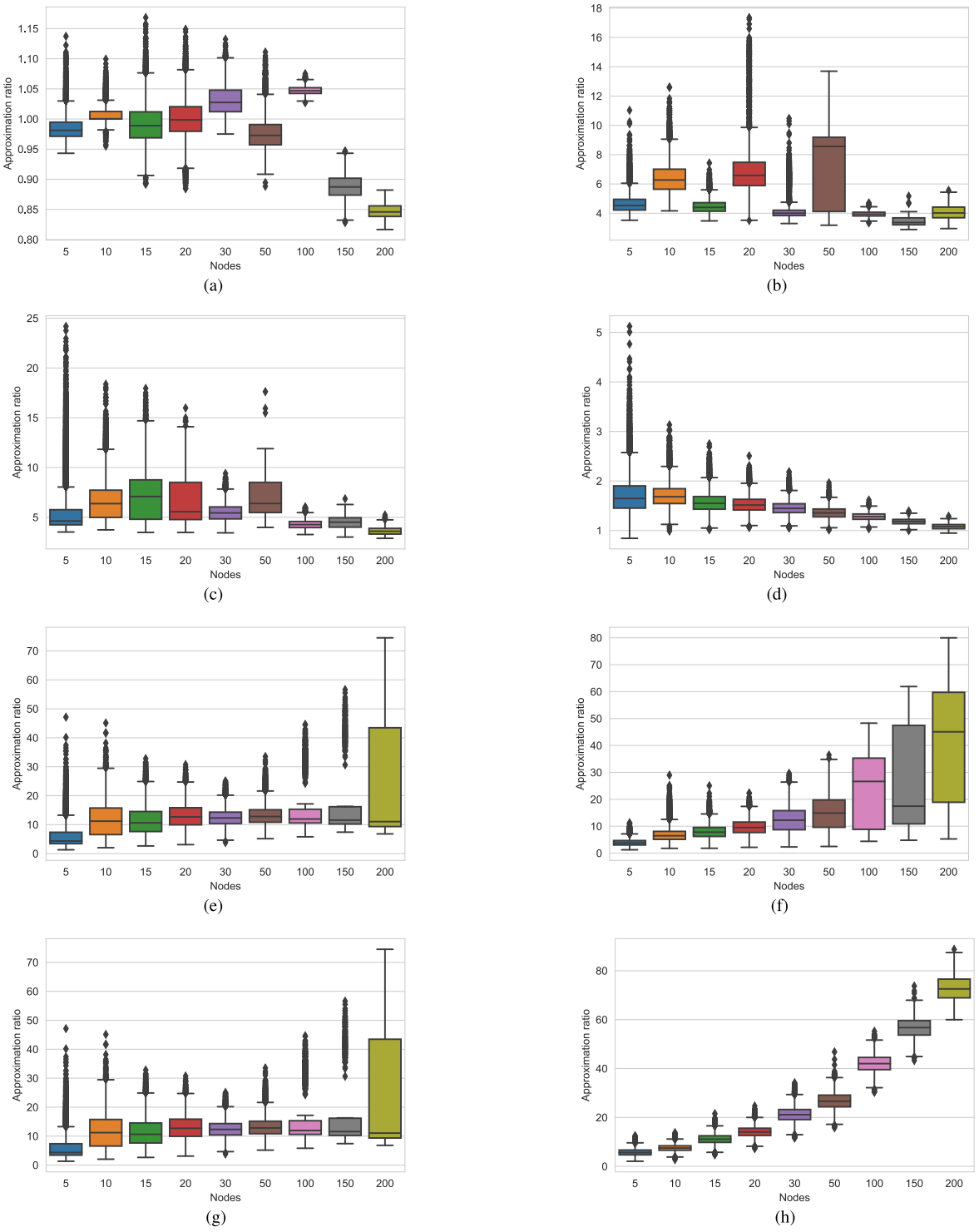

FIGURE 3. Taking the greedy algorithm as a benchmark, the approximate ratio of eight methods: (a) ***OurMethod***. (b) ***Throttling(H)***. (c) ***Throttling(H/2)***. (d) ***Root***. (e) ***1/4 Nodes***. (f) ***Half***. (g) ***3/4 Nodes***. (h) ***All***.

ten actions in the action space: $0.0, 0.1, \ldots, 0.9$ which correspond to $0\%, 10\%, \ldots, 90\%$ traffic drop probabilites. Due to the nature of the attack, the state of network has not surely been affected by the actions taken by the agents at the previous time step. This is because the attack traffic is controlled by the attacker in DDoS and the legal traffic depends on the user's request or use. So we only consider instant rewards, which means that Agents only focus on the best response to the current state, i.e. set the discount factor to 0. In our setup, the acceptable upper triffic boundary for the protection target is not considered. We set the loss for each throttling i: $\alpha a_i + \beta l_i$.

TABLE 1. Taking the optimal solution as a benchmark, the approximate ratio of the greedy algorithm and our algorithm.

Nodes	Greedy	Our mathod
5	1.0027	1.0002
6	1.0027	1.0006
7	1.0031	1.0005
8	1.0027	1.0078
9	1.0023	1.0011
10	1.0025	1.0029
11	1.0023	1.0106
12	1.0027	1.0086
13	1.0026	1.0173
14	1.0023	1.0012
15	1.004	1.0018

We consider two deployment methods. We set the height of the tree to H. The first method *Throttling*(H) is to deploy the throttle on all leaf nodes. The second method *Throttling*($H/2$) is to deploy the throttle at nodes whose height is $\lfloor \frac{H}{2} \rfloor$ and leaf nodes whose height is smaller than $\lfloor \frac{H}{2} \rfloor$.

C. RANDOM DEPLOYMENT

In order to prove that our method can get a good deployment scheme, we compare our method with several random deployment schemes. *Root*, randomly deploy a classifier at the root node. *ALL*, randomly select a classifier for deployment at all nodes. *Half*, randomly select half of the nodes to deploy the classifier. $1/4$*Nodes* and $3/4$*Nodes*, randomly select a quarter node and three quarter nodes to deploy a classifier.

D. PERFORMANCE COMPARISION

First, we need to compare the optimization results. All computations are performed on a 64-bit PC with 8.0 GB RAM and a 3.20 GHz CPU. Table 1 shows the approximate ratio of the two methods using the optimal solution obtained by the exhaustive method as the benchmark when 5-15 nodes are used. Both methods can get an acceptable approximate solution, and our method is very close to, or even better than, the greedy method. Due to the computational complexity of the exhaustive method, in the following experiments, we use the solution of the greedy method as a benchmark to compare the results of various methods.

We use box plots to represent the approximation of the three methods to greedy algorithms. Fig. 3(a) shows that our method approximate ratio can be close to 1, even better than our benchmark. This proves that the structural information has an impact on the way the classifier is deployed. At the same time, we see that our results are relatively stable and less prone to large losses. As the number of nodes increases and the structure information becomes more complex, our method can still maintain a good approximate ratio. This shows its advantages. Fig. 3 (b) and (c) are approximate ratios of the throttling method and the greedy method as benchmarks. Although they can effectively reduce the load on the server and cause less latency, it may discard large amounts of legitimate traffic. At the same time, the effect of

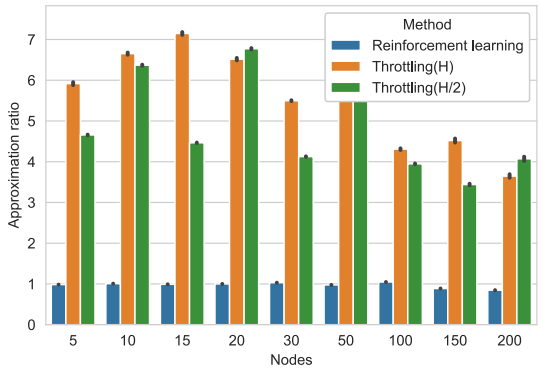

FIGURE 4. Average approximation ratio of the three methods.

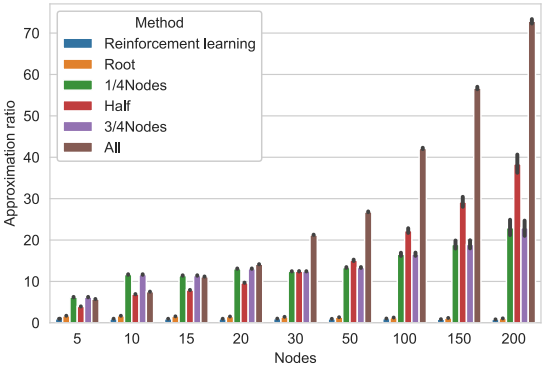

FIGURE 5. Average approximation ratio of the six methods.

the throttling method is not stable enough, and bad solutions will cause huge losses. Fig. 4 shows that our method has significant advantages. The average approximation ratio of our method is much lower than the two throttling methods. Although it is important to ensure the normal operation of the server, most services today need to guarantee the quality of service. We need to better distinguish attack traffic from legitimate traffic for interception.

To prove the superiority of our deployment method, we compared several random deployment methods. Fig. 3 (d)-(h) are boxplots with the greedy solution as a benchmark. Compared to these deployment methods, our method has significant advantages. This is because it can find the most suitable nodes based on traffic changes. As shown in Fig. 5, the number of nodes increases, the traffic in the system increases, and the effect of several random deployment methods other than *Root* deteriorates significantly. This indicates that the loss is related to real-time traffic. Even though the *Root* method still maintains excellent results, our method is still better than it.

Fig. 6 shows that as the number of nodes increases, the calculation time of the exhaustive method increases significantly. Although it can obtain the optimal solution, it obviously cannot meet the real-time performance of the system. The calculation time of our method is extremely competitive. This is because the number of times that R is calculated by the greedy algorithm is related to the square of the

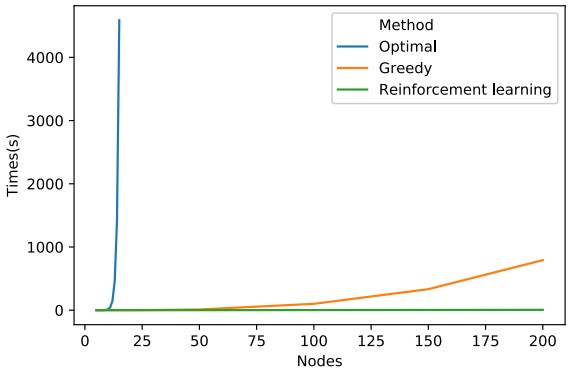

FIGURE 6. Comparison of three methods in running time.

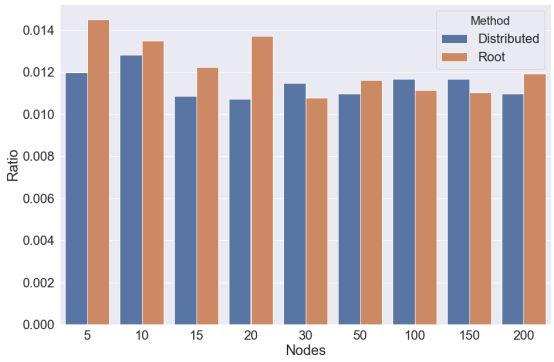

FIGURE 7. The ratio of attack traffic to the server by the two methods.

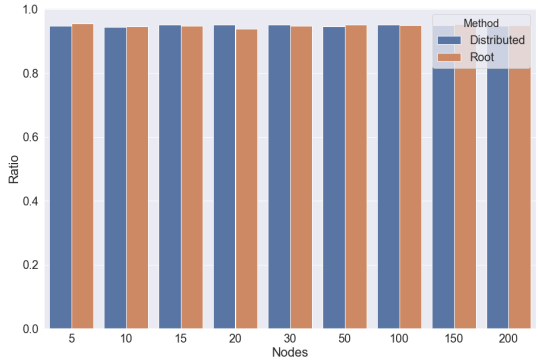

FIGURE 8. The ratio of user traffic to the server by two methods.

number of nodes $|V|$. Our method can still make decisions in seconds when the node is increased to 200. This facilitates the real-time response to the attacker's dynamic traffic adjustment.

E. SIMULATION

We used a simulation experiment on the network simulator OMNeT ++ to compare the distributed method with the root node filtering method. Our traffic data is KDD19, and the classifier is a neural network. The number of nodes in the network is 5, 10, 15, 20, 30, 50, 100, 150 and 200, respectively. Since the verification time is greater than the

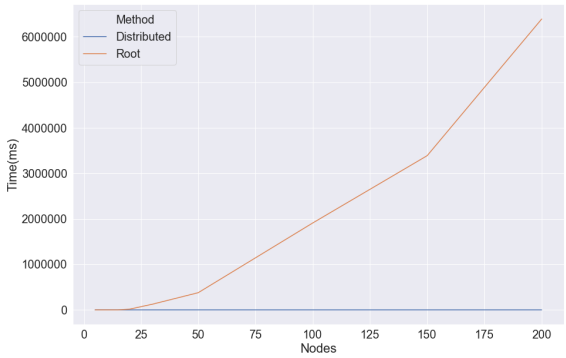

FIGURE 9. The cumulative delay time of the two methods.

delay time in the simulation, a verification delay will occur when the flow is too large. Fig. 7 and Fig. 8 show that the two methods have little difference in attack traffic interception and protection of user traffic. Fig. 9 The biggest advantage of our method is that it greatly reduces the inspection delay. Fig. 9 reflects its significant advantage.

VI. CONCLUSION

We are the first to propose a defense strategy of dynamic deployment of classifiers in network topology to minimize the impact of DDoS attacks on protected targets. It uses graph embedding and reinforcement learning to solve combinatorial problems. The simulation results show that our method can not only ensure that the results are close to the approximate solution, but also greatly reduce the computation time. Compared with throttling method, deploying classifier also has great advantages. It can also solve other flow problems based on diagrams (trees). As future work, more traffic-related work can be done in a similar way.

REFERENCES

[1] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 39–53, Apr. 2004.

[2] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred, "Statistical approaches to DDoS attack detection and response," in *Proc. DARPA Inf. Survivability Conf. Expo.*, Apr. 2003, pp. 303–314.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[4] J. Ravi, W. Shi, and C.-Z. Xu, "Personalized email management at network edges," *IEEE Internet Comput.*, vol. 9, no. 2, pp. 54–60, Mar. 2005.

[5] S. Yi, Z. Qin, and Q. Li, "Security and privacy issues of fog computing: A survey," in *Proc. Int. Conf. Wireless Algorithms, Syst., Appl.* Springer, 2015, pp. 685–695.

[6] D. K. Y. Yau, J. C. S. Lui, F. Liang, and Y. Yam, "Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles," *IEEE/ACM Trans. Netw.*, vol. 13, no. 1, pp. 29–42, Feb. 2005.

[7] K. Malialis and D. Kudenko, "Distributed response to network intrusions using multiagent reinforcement learning," *Eng. Appl. Artif. Intell.*, vol. 41, pp. 270–284, May 2015.

[8] K. Malialis, S. Devlin, and D. Kudenko, "Distributed reinforcement learning for adaptive and robust network intrusion response," *Connection Sci.*, vol. 27, no. 3, pp. 234–252, Jul. 2015.

[9] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang, "SAVE: Source address validity enforcement protocol," in *Proc. 21st Annu. Joint Conf. IEEE Comput. Commun. Societies*, Jun. 2002, pp. 1557–1566.

[10] K. Kumar, R. C. Joshi, and K. Singh, "A distributed approach using entropy to detect DDoS attacks in ISP domain," in *Proc. Int. Conf. Signal Process., Commun. Netw.*, Feb. 2007, pp. 331–337.
[11] K. Toczé and S. Nadjm-Tehrani, "A taxonomy for management and optimization of multiple resources in edge computing," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–23, Jun. 2018.
[12] R. Beraldi, A. Mtibaa, and H. Alnuweiri, "Cooperative load balancing scheme for edge computing resources," in *Proc. 2nd Int. Conf. Fog Mobile Edge Comput. (FMEC)*, May 2017, pp. 94–100.
[13] K. Bhardwaj, J. C. Miranda, and A. Gavrilovska, "Towards iot-ddos prevention using edge computing," in *Proc. USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, 2018, pp. 1–7.
[14] N.-N. Dao, D.-N. Vu, Y. Lee, M. Park, and S. Cho, "MAEC-X: DDoS prevention leveraging multi-access edge computing," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2018, pp. 245–248.
[15] R. L. Church, M. P. Scaparra, and R. S. Middleton, "Identifying critical infrastructure: The median and covering facility interdiction problems," *Ann. Assoc. Amer. Geographers*, vol. 94, no. 3, pp. 491–502, Sep. 2004.
[16] Q. Guo, A. Bo, Y. Zick, and C. Miao, "Optimal interdiction of illegal network flow," in *Proc. Int. Joint Conf. Artif. Intell.*, 2016, pp. 1–7.
[17] Q. Guo, B. An, and L. Tran-Thanh, "Playing repeated network interdiction games with semi-bandit feedback," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, Aug. 2017, pp. 1–9.
[18] O. Vaněk, Z. Yin, M. Jain, B. Bošanskỳ, M. Tambe, and M. Pěchouček, "Game-theoretic resource allocation for malicious packet detection in computer networks," in *Proc. 11th Int. Conf. Auto. Agents Multiagent Syst.* vol. 2, 2012, pp. 905–912.
[19] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2692–2700.
[20] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016, *arXiv:1611.09940*. [Online]. Available: http://arxiv.org/abs/1611.09940
[21] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, Oct. 2016.
[22] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6348–6358.
[23] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 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

HAODI ZHANG was born in Hohhot, China, in 1994. He received the bachelor's degree from Jilin University, in 2017. He is currently pursuing the master's degree with the College of Intelligence and Computing, Tianjin University, China. His current research interest includes cybersecurity.

JIANYE HAO received the B.E. degree from the School of Computer Science and Technology, Harbin Institute of Technology, in 2008, and the Ph.D. degree in computer science and engineering from The Chinese University of Hong Kong, in 2013. He was a Postdoctoral Research Fellow of the Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology (MIT), and the Pillar of Information System Technology and Design (ISTD), Singapore University of Technology and Design (SUTD), from 2013 to 2015. He was also a Lecturer at ISTD, SUTD, from November 2014 to March 2015. He was a Visiting Research Fellow of the National University of Singapore and the University of Wollongong, in 2011 and 2014, respectively. He is currently an Associate Professor with the School of Software, Tianjin University, leading the lab of deep reinforcement learning.

XIAOHONG LI (Member, IEEE) received the Ph.D. degree from Tianjin University, Tianjin, China, where she is currently a full Tenured Professor with the School of Computer Science and Technology. Her current research interests include knowledge engineering, trusted computing, and security software engineering.