モバイルセンサーネットワークにおける学習能力を持つ攻撃端末の発見

Boqi GAO†, Takuya MAEKAWA†, Daichi AMAGATA†, and Takahiro HARA†

† Osaka University, Graduate School of Information Science and Technology
1–5, Yamadaoka, Suita, Osaka 565–0871 Japan
E-mail: †{gao.boqi,maekawa,amagata.daichi,hara}@ist.osaka-u.ac.jp

Abstract Mobile wireless sensor networks (WSNs) are facing threats from malicious nodes which disturb packet transmissions, leading to the poor performance of mobile WSNs. Existing studies provided some methods, such as decision tree based classification, to detect these malicious nodes. However, these malicious nodes are assumed to follow only pre-defined attack models and do not have any learning ability. This limits the malicious nodes in regard of development of machine learning technologies. For this reason, in this study, we design a reinforcement learning based malicious nodes, and define a novel observation space and a sparse reward function for them. We also propose an adaptive learning method to detect these malicious nodes. Extensive experiments show that compared with existing attack models, our malicious nodes can make networks perform worse while not being detected. We also investigate the performance of our detecting method and confirm that our method significantly outperforms the state-of-the-art methods in terms of detection accuracy and false detection rate.

Key words Mobile WSN, reinforcement learning, security

1. Introduction

Driven by extensive growth of Internet of Things (IoT) and artificial intelligence (AI) technologies, as an important application of them, a mobile wireless sensor network (WSN) has demonstrated its feasibility, which requires long range communications and capability of monitoring. However, due to the self-organized characteristic, malicious nodes can easily join a mobile WSN. That is, mobile WSNs are inherently vulnerable to malicious nodes, and the malicious nodes disrupt communications in mobile WSNs, causing packet transmission failures; all of which presents security challenges to mobile WSN environments.

Existing studies have proposed a variety of routing attack models [6]. These attack models mainly aim at disturbing packet transmissions and causing redundant traffic. One of the most aggressive attacks, grey hole attack [22], has been proved to have the ability to destroy the routing procedure and data transmissions of a sensor network. Rule-based countermeasures have been proposed for avoiding attacks and/or detecting grey hole attacks in mobile WSNs [25], [26]. However, the existing grey hole attack follows only pre-defined routines, which means that they cannot modify their attack patterns. This leads to the result that the malicious nodes are easy to be detected by corresponding countermeasures [1], [24]. Moreover, the assumption that malicious nodes have no learning ability is extremely unrealistic in a world with rapid development of machine learning. Thus, this study assumes that malicious nodes have learning ability, and they can learn from the countermeasures to avoid being detected. Similarly, the above attack countermeasures cannot deal well with an attack model which is capable of learning, because preparing a static rule for an attack model which can change its behavior is impractical. Therefore, the attack countermeasures should also be have learning ability.

Contribution. To address the problem of designing an grey hole attack model with learning ability, we utilize reinforcement learning to let the malicious nodes learn by themselves in a mobile WSN. In our method, malicious nodes obtain a positive reward when they deteriorate the performance of the mobile WSN. However, if these nodes behave too maliciously, they may be easily detected by a countermeasure of the mobile WSN. We assume that they receive a large negative reward if they are detected within a time threshold. In a large number of episodes, the malicious nodes will finally learn how to perform maliciously as much as possible while not being detected. In addition, we design a method that robustly detects the above attack model. Over a certain period, the detection method is updated adaptively to find the rule of attack detection. The principal contributions of our work are:

- We design a novel reinforcement learning based grey hole attack model in mobile WSNs. To our knowledge, this is the first work that applies reinforcement learning algorithms to build a smart grey hole attack model in mobile WSNs.
- We propose a method for discrete-time adaptively up-

dating the countermeasure. This method can update countermeasures to detect our novel attack model.

- We conduct extensive experiments to investigate the performance of our attack model and our corresponding countermeasure. From the results of these experiments, we confirm that our attack model is hardly detected by existing countermeasures. Moreover, our countermeasure outperforms the state-of-the-art.

Organization. We review related works in Section 2. Section 3 introduces the assumption in this paper. Our proposed method is described in Section 4. The experimental results are illustrated in Section 5. Section 6 summarizes our conclusion.

2. Related work

Reinforcement learning. Reinforcement learning (RL) has become an increasing popular research area due to its effectiveness in various tasks. As a model-free reinforcement learning technique, the Q-learning algorithm can derive the optimal strategy with probability one if all the feasible actions are repeatedly sampled over all the states in a Markovian decision process [20]. Mnih et al. [17] firstly successfully approximated Q function with a deep convolutional neural network, and enabled their agent to beat a human expert in several Atari games. Later on, RL algorithms have been applied to many WSN-related applications such as scheduling of energy harvesting node [3] and detecting the rough edge of a VANET [15]. In the domain of WSN security, Li et al. [14] used game theory to protect packet transmissions against smart attacks. Xiao et al. [27] utilized RL against smart jamming. These works assume that malicious nodes follow existing patterns, and the proposed methods are only effective for pre-defined attack models. Therefore, for a malicious node that can learn, simply extending these existing methods is not trivial.

Grey hole attack model and countermeasures in WSNs. Security of mobile WSNs is a challenging issue, and many existing studies concerned about it. Existing studies have proposed plenty attack models [13]. Grey hole attack is a special one among them. In the grey hole attack model, malicious nodes can selectively (randomly) forward packets or reply requests to drag routes to themselves. The ordinary grey hole attack is a smart version of the famous black hole attack [23], and it is hard to be detected by ordinary reputation based detection methods [22]. Some works have developed techniques for avoiding the attack with multipath approaches [26], analyzing the impact of the attack [25], and preventing the attack [21]. However, the existing grey hole attack model and corresponding countermeasures do not assume that the malicious nodes can equip with learning ability.

3. Assumption

3.1 Network model

Our network environment is assumed to be a mobile WSN consisting of n wireless nodes with unique identifiers (i.e., node ID). These nodes can move without restriction, while directly communicating with other nodes if they are within the communication range. We assume that all nodes have the same communication range, and if a given node is within the communication range of other nodes, it is a neighboring node of them.

As a routing protocol, AODV [19], which is a standard routing protocol in mobile WSNs, is employed. In AODV, when a source node `s` wants to send a data packet to a destination node `d`, if there is no valid route existing in routing table of `s`, `s` broadcasts a route request (RReq) to create a packet transmission route, and this message is transmitted by some intermediated nodes. When node `d` receives this RReq, it sends a route reply (RRep) toward `s` to create a valid route, then data packet will be transmitted through this route between `s` and `d`. To maintain the route, route error (Rerr) and hello messages are also utilized. When a node detects a link failure in an active route, it sends an Rerr to notify other nodes of the link failure. Each node checks whether it has sent a message or not within an interval called hello interval. If not, it broadcasts a hello message to notify other nodes of its existence (see [19] for other details).

3.2 Mobile nodes

In this paper, we consider a mobile WSN contains two categories of nodes: malicious nodes, which do grey hole attack, and normal nodes. We assume that each category of nodes hold their own servers (respectively normal server and malicious server) with machine learning functions. In addition, we assume that malicious nodes conspire together to destroy the regular routing procedure and disturb the proper data transmissions in a mobile WSN. Furthermore, equipped with computation power, caching resources, and the malicious server, these malicious nodes can use reinforcement learning to learn not to be detected while executing malicious behaviors. On the other hand, the normal nodes attempt to update their detecting method to follow the latest behavioral pattern of the malicious nodes. The normal nodes work together to detect malicious nodes with a specific data collecting method (e.g., flooding through the network).

3.3 Deep Q-learning for malicious server

We assume that malicious nodes employ deep Q-learning [17] as the reinforcement learning algorithm. Below we present a brief summary of deep Q-learning.

Reinforcement learning deals with learning an optimal pol-

icy π^* for an agent interacting in an unknown environment. At each time step t, an agent observes the current state s_t of the environment, decides on an action a_t according to a policy π, and observes a reward signal r_t. The goal of the agent is to find a policy that maximizes the expected sum of discounted rewards R_t

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}, \tag{1}$$

where T is the time at which the episode terminates, and $\gamma \in [0,1]$ is a discount factor that determines the importance of future rewards. The Q-function of a given policy π is defined as the expected return from executing an action a in a state s:

$$Q^{\pi}(s,a) = \mathbb{E}[R_t | s_t = s, a_t = a] \tag{2}$$

Finally, the updating rule for the Q-values for an action selection policy π is as follows:

$$\begin{aligned} &Q^{\pi}(s_t,a_t) = (1-\alpha)Q^{\pi}(s_t,a_t) + \\ &\alpha[R_t(s_t,a_t) + \gamma \max_a Q^{\pi}(s_{t+1},a)] \end{aligned} \tag{3}$$

where

$$0<\alpha<1 \tag{4}$$

is the learning rate. It has been proved that through sufficiently large number of learning iterations, the Q-learning algorithm converges and returns the optimal policy π^* [11].

Instead of performing the Q-learning updates in an on-line learning fashion, it is popular to use experience replay to break the correlation between successive samples [2]. At each time step, an agent experience (s_t, a_t, r_t, s_{t+1}) is stored in a replay memory, and the Q-learning updates are done on batches of experiences randomly sampled from the memory. In the training of the deep Q-learning agent, we assume that malicious nodes share the same malicious server, so note that the s_t and s_{t+1} in one experience tuple (s_t, a_t, r_t, s_{t+1}) are the *state* and *next state* of a particular malicious node.

At every training step, the next action is generated by using an ϵ-greedy strategy: with a probability ϵ, the next action is selected randomly, and with probability $1-\epsilon$, the next action is the best one obtained by the deep Q-learning algorithm. In our method, we start with $\epsilon = 1$ and progressively decay ϵ.

4. Proposed method

4.1 Overview

Here we describe the overview of our method. The method, which is based on machine learning techniques, consists of a training phase and a test phase. Figure 1 presents an overview of the training phase of the proposed method. In the training phase, we simulate various environments with different environmental parameters, e.g., initial node positions, and train the malicious server with reinforcement learning algorithms in an on-line manner, using data obtained from the malicious nodes in the environments. Simultaneously, the normal server is also trained by the data collected from the normal nodes. It is important to recall that in the training phase, all the malicious and normal nodes respectively share their own server. In other words, both the servers can monitor the whole mobile WSN, and can utilize all the data from their nodes because the training phase is based on simulations.

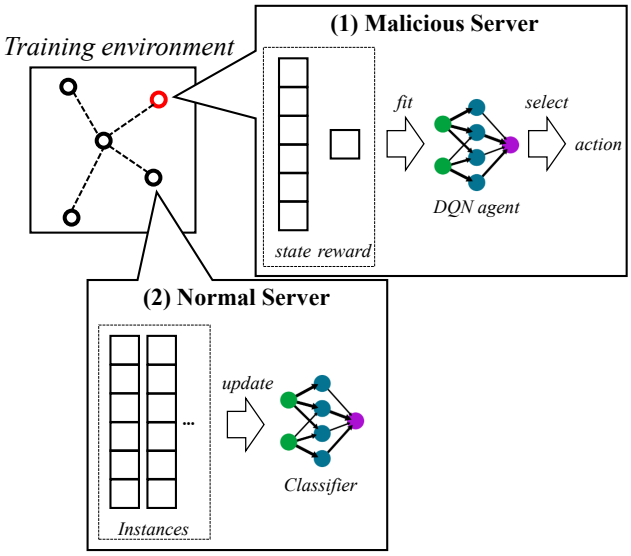

Figure 1 Overview of the training phase of the proposed method. (1) Malicious server monitors the mobile WSN, and utilizes our method to obtain *state* and *reward*. Then the malicious server updates the deep Q-learning agent and selects an *action*. (2) Normal server collects instances from the normal nodes then updates the classifier.

The reason why we train servers on different network environments is derived from the fact that only training on a single network environment is not general [10], i.e., the normal server may overfit the behavioral pattern of the malicious nodes in a single network environment. For example, assume that we have network environments A and B, and we trained the normal server in network environment A. This would easily detect malicious nodes in network environment A because the normal server fits the malicious data perfectly. However, it may not work well in network environment B, as the normal server has no training data of network environment B.

In the test phase, each normal/malicious node is equipped with a fixed normal/malicious server obtained in training phase. Normal nodes use the trained normal server to detect

Table 1 Information observed by each node

Information	Definition
NReqRec	# RReq overheard from one neighbor
NRepRec	# RRep overheard from one neighbor
NRepSen	# RRep sent to one neighbor
NRerRec	# Rerr overheard from one neighbor
NRerSen	# Rerr sent to one neighbor
NHelRec	# hello messages overheard from one neighbor
NDatRec	# data packets overheard from one neighbor
NDatSen	# data packets sent to one neighbor
TReqRec	Total # RReq overheard by observing node
TReqSen	Total # RReq sent by observing node
TRepRec	Total # RRep overheard by observing node
TRepSen	Total # RRep sent by observing node
TRerRec	Total # Rerr overheard by observing node
TRerSen	Total # Rerr sent by observing node
THelRec	Total # hello messages overheard by observing node
THelSen	Total # hello messages sent by observing node
TDatRec	Total # data packets overheard by observing node
TDatSen	Total # data packets sent by observing node
ActNei	Active # neighbors met from start
TPckSen	Total # packets sent
TPckRec	Total # packets received

malicious nodes, whose action is decided by their malicious servers. Note that each server is no more updated in the test phase because, (i) for the normal server, the category of a neighboring node is unknown, and (ii) for the malicious server, the reward is unknown. Therefore, in the test phase, both malicious and normal nodes do not need to collect data from other nodes.

Moreover, we assume that both malicious and normal nodes observe the behavior of its neighboring node. Before we describe the data collected for the normal and malicious servers, we present the information related to the neighboring node observed by each node (Table 1). Note that, in AODV, messages are transmitted by broadcasting, and thus observing nodes can overhear the messages.

4.2 Smart grey hole attack model

Here we describe our smart grey hole attack. This novel attack model is based on reinforcement learning. Malicious nodes equipped with this attack model aim at disturbing the routing and data transmission of a mobile WSN while not being detected.

4.2.1 Ordinary grey hole attack model

Before we illustrate our smart grey hole attack model, we first describe the details of ordinary grey hole attack model.

The ordinary grey hole attack model contains mainly two parts. i) When a grey hole malicious node receives an RReq, it may perform normally (follow the routing protocol of this mobile WSN), or reply an RRep with one hop count, which means that the malicious node states that it is near the destination. ii) When the malicious node receives a data packet, it may also perform normally, or just drop it. The percentage that the node performs normally or not is decided by a malicious ratio (e.g., 50%).

However, this static pre-defined malicious ratio makes the grey hole malicious nodes easily detected by some machine learning based detection methods. Therefore, in this study, we employ reinforcement learning to enable the malicious node to learn the most proper behavior. The actions of our smart grey hole attack model are the same as ordinary grey hole attack model.

4.2.2 Overview of smart grey hole attack model

We describe the overview of smart grey hole attack model. In the training phase, we run sufficient episodes of simulations, and in each episode, we employ different network parameters. During each training episode, the malicious server monitors the mobile WSN and employs our method to obtain a reward and a state after a malicious node receives any packet. The malicious server then updates itself, and selects an action for this malicious node. When all malicious nodes in the mobile WSN are detected or the simulation time goes over the threshold, an episode ends.

In the test phase, each malicious node is installed with a trained malicious server obtained in the training phase. The malicious server of each node is no more updated. However, it still utilizes our method to obtain state and selects action for its malicious node.

4.2.3 State and reward engineering

As introduced in Section 3.3, reinforcement learning utilizes a series of tuples (s_t, a_t, r_t, s_{t+1}) for updating the policy of an agent. However, different from existing simple game-like learning tasks (e.g., Cart-pole game from OpenAI Gym (注1)), which have obvious state and reward declaration, and Atari game missions, which can utilize convolutional neural networks to conveniently extract features from game picture frames, the malicious server does not have an existing method to obtain state and reward from monitoring data. Thus, we design a method that obtains the state and reward.

State engineering. A malicious node observes a *state*, then selects an *action* according to the decision of the malicious server. To enable the malicious nodes to understand its situation, we design the state (as a vector to input into neural networks) for the malicious nodes by extracting information from the most recent received packets. In particular, we extract packet category ID and neighboring node ID. Packet category ID shows the category of this received packet (e.g., RReq is 1 and RRep is 2), and node ID represents that ID of the node who sends this packet. Note that the packet

(注1)：https://gym.openai.com/

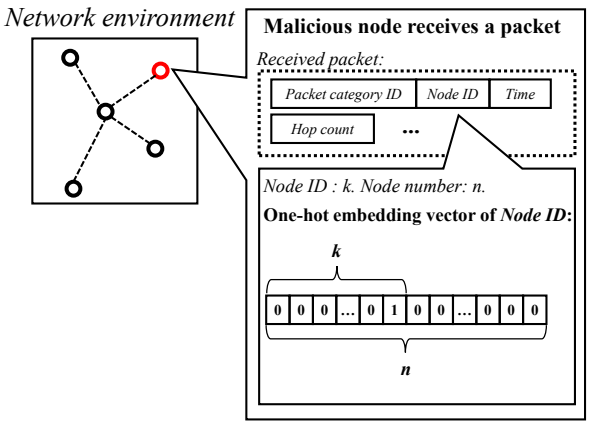

Figure 2 One hot encoding of node ID

category ID and the node ID are not numeric data, but categorical data. For instance, compared with node ID 3, node ID 2 and node ID 8 have just the same differences. However, if we just utilize these node IDs (2, 3, 8) as input of the neural networks, the neural networks of malicious server treat 2 more similar to 3 than 8 [7]. As a result, we employ one-hot encoding [8] to encode the node ID and packet category ID, for the purpose of appropriately processing categorical data.

Figure 2 illustrates our one-hot encoding for node ID. In a mobile WSN of n nodes, the node ID k of a received packet is transformed into a vector with length n, with the $k-th$ *dimension* of the vector is 1, and the rest is 0. The one-hot encoding of packet category ID is the same as that of the node ID. A malicious node can learn which neighboring node sends this packet, and learns the behaviors of its neighboring nodes from these two settings.

However, the malicious nodes cannot recognize their situations clearly based only on these two states, because the received packet category ID and neighboring node ID may always be similar during the training simulation. Therefore, we also extract ratios that can well describe the malicious nodes' situations. Table 2 presents a summary of states. In this table, the first and second lines are the one-hot vectors of packet category ID and neighboring node ID. Then we calculate the connection between sent and received numbers of each category of packet and total sent and received numbers. These ratios can not only represent the node density (ActNeiRatio) near a malicious node, but also can illustrate the packets sent and received frequency of a particular category of packet. For example, if a malicious node sends too many RReps, its RepTotSenRatio is extremely high. Under such a situation, it will be soon detected by some machine learning based detecting method. Consequently, this malicious node will try to send fewer times of RReps not to be detected. Moreover, another advantage of the state is that

Table 2 States of malicious nodes

States	Definition
Packet category	One-hot vector of most recently received packet category ID
Neighbor node	One-hot vector of most recently received neighboring node ID
ActNeiRatio	1/ActNei
ReqTotSenRatio	TReqSen / TPckSen
RepTotSenRatio	TRepSen / TPckSen
RerTotSenRatio	TRerSen / TPckSen
HelTotSenRatio	THelSen / TPckSen
DatTotSenRatio	TDatSen / TPckSen
ReqTotRecRatio	TReqRec / TPckRec
RepTotRecRatio	TRepRec / TPckRec
RerTotRecRatio	TRerRec / TPckRec
HelTotRecRatio	THelRec / TPckRec
DatTotRecRatio	TDatRec / TPckRec
ReqRecTotSenRatio	TReqRec / TPckSen
RepRecTotSenRatio	TRepRec / TPckSen
RerRecTotSenRatio	TRerRec / TPckSen
HelRecTotSenRatio	THelRec / TPckSen
DatRecTotSenRatio	TDatRec / TPckSen
ReqSenTotRecRatio	TReqSen / TPckRec
RepSenTotRecRatio	TRepSen / TPckRec
RerSenTotRecRatio	TRerSen / TPckRec
HelSenTotRecRatio	THelSen / TPckRec
DatSenTotRecRatio	TDatSen / TPckRec

the values of these processed ratios are all between [0, 1], which means our state design does not need normalization, and can converge rapidly.

We finally concatenate one-hot vector of packet category ID, one-hot vector of neighboring node ID, and extracted ratios together as our state vector.

Reward shaping. The design of reward in reinforcement learning is important because it describes how the agent ought to behave. Therefore, we explicitly design the reward for the malicious server.

Our smart grey hole attack model aims at i) dropping data packets while ii) trying not to be detected. The result of dropping data packets can be represented by how much the transmission rate (total # of data packets received by destination node/total # of data packets sent by source node) is lowered, and the result of trying not to be detected can be represented as how long the malicious nodes keep not being detected. As a result, the following rewards for shaping the reward function of the malicious server are considered.

- Positive reward for lowering the transmission rate.
- Positive reward for not being detected before a predefined time threshold.
- Negative reward for being detected.

Although we form the principle of our reward function, set-

ting magnitudes and frequencies of rewards is difficult, and they often depend on applications. For instance, in Atari Pong game, the rewards are bounded by -1 and $+1$, while in Atari Mr. Pac-Man eating a single ghost can yield a reward of up to $+1600$. To overcome this, we employ clipped reward [18], where if the malicious nodes are all detected before the pre-defined threshold, the reward is -1. Otherwise, the reward is $+1$. For lowering the transmission rate, clipping all the rewards to $+1$ cannot show the decrease of transmission rate, so we bound the reward, as $1 - transmisson\ rate$, to reduce imbalance. We then add the positive and negative reward as a final reward.

Note that in our setting, we utilize the sparse reward function. That is, the malicious server does not receive any non-zero reward until an episode is done. Note that an episode is done when i) all the malicious nodes are detected or ii) the simulation time goes over the threshold. We finally summarize our reward:

$$reward = \begin{cases} 0 & (not\ end) \\ 1 + 1 - transmissionrate & (end\ \&\ not\ detected) \\ -1 + 1 - transmissionrate & (end\ \&\ detected) \end{cases}$$

4.3 Countermeasure

This section describes our countermeasure for smart grey hole attack model.

4.3.1 Overview of countermeasure

In our countermeasure, we aim at building a robust classifier to detect smart grey hole malicious nodes. This neural network classifier in the normal server is built to classify the neighboring nodes of each normal nodes. To obtain this classifier, in the training phase, the classifier is trained simultaneously when the malicious server is updated. That is, in the large number of episodes in the training phase, the classifier is also trained in each episode. Finally, a trained classifier is obtained after the training phase.

In the test phase, we install this trained normal server on each normal node, and normal nodes utilize majority votes to decide the category of a neighboring node. For instance, if one node `a` has 3 neighboring normal nodes, and two of the normal nodes classify this node as malicious node, then this node `a` will be decided as malicious node.

4.3.2 Frequent adaption in the training phase

In the training phase, because the malicious nodes employ reinforcement learning to adjust their behavior pattern, the normal nodes, simultaneously, need to update their server frequently. We thus make the normal nodes connect to the normal server to update the classifier and detect malicious nodes frequently. Before each adaption, the normal server collects data (features and categories of neighboring nodes) from all the normal nodes, then the normal server uses this data to update itself.

As the features of the normal server, we extract inherent features by the method in [10]. We also add another feature, the ActNeiRatio, which is calculated by 1/ActNei, to enable the normal nodes to figure out the number of surrounding active neighboring nodes. Note that we utilize SMOTE [5] to over-sample the minority class so that the proportion of training instances from each class is equal.

4.3.3 Pre-training of normal classifier

In our method, we train a neural network classifier and utilize it to detect malicious nodes. However, if we randomly initialize the neural network classifier, it will have low accuracy and may converge slowly. Therefore, before the start of training phase, for the purpose of fast adaption and accelerating training speed, we pre-train the classifier with data obtained in different network environments. In particular, we run simulations with normal nodes and malicious nodes with ordinary grey hole attack models to obtain data. Then we utilize the data to construct a pre-trained classifier. Finally, we run the training phase on this pre-trained classifier instead of a randomly initialized classifier.

5. Experiments

5.1 Setting

We used Qualnet 7.4 network simulator(注2). Each node transmitted messages and data packets, whose payload sizes were 256 bytes, using an IEEE 802.11b device. The communication range of each node was adjusted to roughly 100 meters, and the network bandwidth was 11Mbps. As with existing works [9], [12], we used the random way point model [4], with a maximum movement speed of v_{max} and pause time of 0. (The velocity of each node was randomly chosen from $(0, v_{max}]$.) When there were n nodes in a network, there were $n \cdot m$ ($m \in [0.1, 0.4]$) malicious nodes in the network. The normal nodes contact with the normal server to detect neighboring malicious nodes and update the classifier in every 0.4 seconds. The threshold time for giving positive was set as 10 seconds. We randomly chose a pair of source node and destination node every f seconds. If the source node has an active route to the destination node, the source node sends a data packet to the destination node directly. Otherwise the source node broadcasts an RReq to find a route to the destination node. The network parameters are described in Table 3.

Evaluation method. We run simulation of 1500 episodes in the training phase. We trained both malicious and nor-

(注2)：`http://web.scalable-networks.com/qualnet-network-simulator-software`

Table 3 Parameter configuration

Parameter	Values
n	20
$m\ of\ training$	0.1, 0.2, 0.4
$m\ of\ testing$	0.3
v_{max} [m/sec]	4.0
Network size [m^2]	300 × 300
f [sec]	0.2

mal servers simultaneously with random initial node positions and random node IDs. We also selected m from 0.1, 0.2, and 0.4 for each episode in the training phase, while in test phase, m was 0.3.

To investigate the effectiveness of the proposed countermeasure (P-detection for short), we used MLP [16] as a competitor. This is a state-of-the-art technique utilizing MultiLayer Perceptron for classification. The method employs the following features: TReqRec, TReqSen, TRepRec, TRepSen, TRerRec, TRerSen, TDatRec, TDatSen(注3), number of neighboring nodes (NeiNum), ratio of routing table update with respect to entries (PCR), and ratio of routing table update with respect to hop counts (PCH). Note that we also update MLP in the same way as the proposed countermeasure.

Furthermore, to investigate the effectiveness of our proposed grey hole attack model (P-attack for short), we compared P-attack with ordinary grey hole attack model (O-attack for short) with a malicious percentage of 50%.

Criteria. We focus on the following criteria to measure the performance of MLP and P-detection.

- Accuracy: This is represented by $\frac{|T_{nor\to nor,mal\to mal}|}{|T|}$, where $T_{nor\to nor,mal\to mal}$ and T are respectively the set of correctly classified **instances** and the set of all **instances**.
- Detection rate: This is represented by $\frac{|T_{mal\to mal}|}{|T_{mal}|}$, where $T_{mal\to mal}$ and T_{mal} are respectively the set of correctly classified **instances** describing malicious nodes and the set of all **instances** describing malicious nodes.
- Mis-Detection rate: This is represented by $\frac{|T_{nor\to mal}|}{|T_{nor}|}$, where $T_{nor\to mal}$ and T_{nor} are respectively the set of wrongly classified **instances** describing normal nodes and the set of all **instances** describing normal nodes.
- Detection time: This shows how long time normal nodes spend to detect all the malicious nodes in a mobile WSN.

5.2 Result

Detection time. Table 4 shows the detection time of P-detection and MLP for P-attack and O-attack. For both the attack models, P-detection detects all the malicious nodes much faster than MLP. This is because MLP employs different features from the P-detection. In particular, MLP only utilizes numbers of packets, which is hard for the neural networks to process when the numbers grow larger. However, our countermeasure utilizes the ratios, and the neural networks can process them more smoothly with gradient descent. As a result, it is obvious that our features can describe the malicious behaviors better. Moreover, compared with O-attack, normal nodes spend much more time to detect P-attack. This is because malicious nodes of P-attack can learn from the detecting method and change their behaviors, and they are hardly detected. We can see from this result that P-attack can exist longer than an O-attack with a detecting method employed by the mobile WSN.

Table 4 Detection time

	P-detection	MLP
P-attack [sec]	10.4	28.8
O-attack [sec]	2.4	9.6

Table 5 Results on O-attack

	P-detection	MLP
Accuracy	0.95	0.85
Detection rate	0.94	0.86
Mis-detection rate	0.02	0.14

Table 6 Results on P-attack

	P-detection	MLP
Accuracy	0.76	0.58
Detection rate	0.88	0.60
Mis-detection rate	0.23	0.41

Table 7 Decrease of transmission rate

	Avg first 50 episodes	Avg last 50 episodes
Transmission rate	71.7	64.7

Accuracy, detection rate, and mis-detection rate. Tables 5 and 6 show the experimental results of accuracy, detection rate, and mis-detection rate of P-detection and MLP on both P-attack and O-attack. In Table 5, we can see that our P-detection has good performance for detecting O-attack, and outperforms MLP. This is because O-attack cannot adaptively change its performance ratio, while P-detection learns its behavioral pattern rapidly. MLP also has good result because although MLP utilizes packet numbers, it can still learn from the training data when the numbers are small. In Table 6, P-detection outperforms MLP on all the three criteria with regard to detecting P-attack. However, compared with results in Table 5, performance of P-detection and MLP decrease. This is because P-attack tries to update itself by changing behaviors. These results show that P-attack can actually learn to avoid being detected, and our proposed detection method is effective and can be em-

(注3) : MLP does not convert these listed features to ratios, which differ from our method.

ployed for detecting this form of attack in mobile WSN.

Decrease of detection rate. To show the performance of P-attack, we present the decrease of transmission rate of mobile WSN in the training phase (Table 7). This table shows the result of the average transmission rate of the first and the last 50 episodes. From this result, we can see that the transmission rate drops after the training phase. This result shows that P-attack can learn to disturb the transmission of data packet in mobile WSNs.

6. Conclusion

This paper presented a smart grey hole attack model and its countermeasure in mobile WSN. We constructed a reinforcement learning based attack model and detect it by our adaptive server. The experiment results revealed that the attack can learn from the state-of-art countermeasures and extend the lifetime of malicious nodes. Our countermeasure, as well, outperformed the state-of-art and detected the malicious nodes rapidly. As a part of future work, we plan to investigate on super all-round attack with not only the grey hole attack function, but can harm a whole mobile WSN in different ways, and we likewise plan to design a countermeasure of it.

Acknowledgment

This research is partially supported by JSPS Grant-in-Aid for Scientific Research (A) Grant Number JP26240013.

References

[1] S. Abbas, M. Merabti, D. Llewellyn-Jones, and K. Kifayat. Lightweight sybil attack detection in manets. *IEEE Systems Journal*, 7(2):236–248, 2013.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba. Hindsight experience replay. In *NIPS*, pages 5048–5058, 2017.

[3] H. Ayatollahi, C. Tapparello, and W. Heinzelman. Reinforcement learning in mimo wireless networks with energy harvesting. In *ICC*, pages 1–6, 2017.

[4] C. Bettstetter, H. Hartenstein, and X. Pérez-Costa. Stochastic properties of the random waypoint mobility model. *Wireless Networks*, 10(5):555–567, 2004.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[6] X. Chen, K. Makki, K. Yen, and N. Pissinou. Sensor network security: A survey. *IEEE Communications Surveys and Tutorials*, 11(2):52–73, 2009.

[7] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.

[8] A. Coates and A. Y. Ng. The importance of encoding versus training with sparse coding and vector quantization. In *ICML*, pages 921–928, 2011.

[9] W. Galuba, P. Papadimitratos, M. Poturalski, K. Aberer, Z. Despotovic, and W. Kellerer. Castor: Scalable secure routing for ad hoc networks. In *INFOCOM*, pages 1–9, 2010.

[10] B. Gao, T. Maekawa, D. Amagata, and T. Hara. Environment-adaptive malicious node detection in manets with ensemble learning. In *ICDCS*, pages 556–566, 2018.

[11] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *ICML*, pages 2829–2838, 2016.

[12] B. Karaoglu and W. Heinzelman. Cooperative load balancing and dynamic channel allocation for cluster-based mobile ad hoc networks. *TMC*, 14(5):951–963, 2015.

[13] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. In *SNPA*, pages 113–127, 2003.

[14] Y. Li, L. Xiao, H. Dai, and H. V. Poor. Game theoretic study of protecting mimo transmissions against smart attacks. In *ICC*, pages 1–6, 2017.

[15] X. Lu, X. Wan, L. Xiao, Y. Tang, and W. Zhuang. Learning-based rogue edge detection in vanets with ambient radio signals. In *ICC*, pages 1–6, 2018.

[16] A. Mitrokotsa and C. Dimitrakakis. Intrusion detection in manet using classification algorithms: The effects of cost and model selection. *Ad Hoc Networks*, 11(1):226–237, 2013.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[19] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *WMCSA*, pages 90–100, 1999.

[20] E. Rodrigues Gomes and R. Kowalczyk. Dynamic analysis of multiagent q-learning with ε-greedy exploration. In *ICML*, pages 369–376, 2009.

[21] N. Schweitzer, A. Stulman, R. D. Margalit, and A. Shabtai. Contradiction based gray-hole attack minimization for ad-hoc networks. *TMC*, 16(8):2174–2183, 2017.

[22] J. Sen, M. G. Chandra, S. Harihara, H. Reddy, and P. Balamuralidhar. A mechanism for detection of gray hole attack in mobile ad hoc networks. In *ICICS*, pages 1–5, 2007.

[23] S. Shahabi, M. Ghazvini, and M. Bakhtiarian. A modified algorithm to improve security and performance of aodv protocol against black hole attack. *Wireless Networks*, 22(5):1505–1511, 2016.

[24] S. K. Shandilya and S. Sahu. A trust based security scheme for rreq flooding attack in manet. *International Journal of Computer Applications*, 5(12):4–8, 2010.

[25] T. Shu and M. Krunz. Privacy-preserving and truthful detection of packet dropping attacks in wireless ad hoc networks. *TMC*, 14(4):813–828, 2015.

[26] F.-H. Tseng, L.-D. Chou, and H.-C. Chao. A survey of black hole attacks in wireless mobile ad hoc networks. *Human-centric Computing and Information Sciences*, 1(1):4, 2011.

[27] L. Xiao, Y. Li, C. Dai, H. Dai, and H. V. Poor. Reinforcement learning-based noma power allocation in the presence of smart jamming. *TVT*, 67(4):3377–3389, 2018.