Reinforcement Learning Based Mobile Offloading for Cloud-based Malware Detection

Xiaoyue Wan*, Geyi Sheng*, Yanda Li*, Liang Xiao*, Xiaojiang Du†

*Dept. of Communication Engineering, Xiamen University, Xiamen, China. Email: lxiao@xmu.edu.cn

†Dept. of Computer and Information Science, Temple University, Philadelphia, USA. Email: dxj@ieee.org

***Abstract*—Cloud-based malware detection improves the detection performance for mobile devices that offload their malware detection tasks to security servers with much larger malware database and powerful computational resources. In this paper, we investigate the competition of the radio transmission bandwidths and the data sharing of the security server in the dynamic malware detection game, in which each mobile device chooses its offloading rate of the application traces to the security server. As the Q-learning technique has a slow learning rate in the game with high dimension, we have designed a mobile malware detection based on hotbooting-Q techniques, which initiates the quality values based on the malware detection experience. We propose an offloading strategy based on deep Q-network technique with a deep convolutional neural network to further improve the detection speed, the detection accuracy, and the utility. Preliminary simulation results verify the detection gain of the scheme compared with the Q-learning based strategy.**

***Index Terms*—Cloud-based malware detection, reinforcement learning, deep-Q network, mobile offloading**

I. Introduction

Mobile devices, such as smartphones, can apply machine learning techniques such as Bayes network and random forest classifiers in [1] to evaluate the runtime behaviors of the applications (Apps) to detect malwares. The traces or log data generated by the Apps must be processed in real time to avoid zero-day attacks [2]–[5]. However, with limited battery life, computation resources, and radio bandwidth, mobile devices cannot always update the local malware database and process all of the traces of the Apps in time, and thus are vulnerable to zero-day attacks [6]–[9]. Therefore, mobile devices can resort to the security servers at the cloud to detect malwares. As shown in [10], by offloading some detection tasks to security servers with larger malware database, faster processing speed, and more powerful security services, mobile devices can increase the battery life, improve the detection speed and accuracy, and even address zero-day attacks.

The performance of the cloud-based malware detection depends on the offloading. For instance, if too many App traces are offloaded to the cloud, a mobile device can suffer a longer detection delay because of the limited computation resources of the server and radio network congestion. In the cloud-based malware detection game as formulated in [11], each mobile device uploads a portion of the App traces to the cloud via the serving access points (APs) or base stations (BSs) and utilizes the powerful computation resources and the large database with real-time updated malware signatures of the security server. However, the game in [11] has been simplified, especially the channel model and the malware detection accuracy model. Therefore, in this paper, we further improve the game model for the mobile offloading in the cloud-based malware detection.

As a mobile device usually has difficulty obtaining the radio transmission and trace generation models of other mobile devices, the mobile offloading of a mobile device in a dynamic game can be formulated as a Markov decision process (MDP). A reinforcement learning (RL) technique can be applied for a mobile device to detect malware in the dynamic game. A Q-learning based mobile offloading strategy has been proposed in [11] to derive the optimal offloading rate of the App traces without being aware of the channel models in dynamic radio environments. However, the mobile offloading in [11] suffers from a slow learning rate in a large-scale network with a large number of feasible offloading rates, and thus suffers serious performance degradation in the malware detection. Therefore, in this paper, we propose a hotbooting Q-learning based offloading strategy that exploits the malware detection experiences to initialize the expected long-term utility values for each state-action pair and accelerates the learning rate of the standard Q-learning algorithm that initiates all the Q value with zeros.

The deep Q-network (DQN) proposed in [12] exploits a deep convolutional neural network (CNN) to accelerate the learning speed, especially for the high-dimension cases. Therefore, we propose a DQN-based mobile offloading strategy to further enhance the malware detection performance. Simulation results show that the algorithm can increase the malware detection accuracy and reduce the detection delay as compared to a Q-learning based malware detection scheme.

The main contributions of our work can be summarized as follows:

1) We improve the game model in [11] by introducing the malware detection accuracy function in the utility of the mobile device.

This work was supported in part by National Natural Science Foundation of China under Grant 61671396 and CCF-Venustech Hongyan Research Initiative (2016-010).

2) A hotbooting-Q and DQN based malware detection scheme is developed to improve detection accuracy and speed compared to the scheme previously proposed in [11].

The rest of this paper is organized as follows. We review the related work in Section II and formulate the cloud-based malware detection game in Section III. We propose the RL-based mobile offloading algorithms for malware detections in Section IV. We provide simulation results in Section V and conclude the paper in Section VI.

II. Related Work

The theoretic game study on mobile offloading has helped build efficient offloading strategies. The non-cooperative offloading game formulated in [13] investigates the three-tier cloud offloading architecture that adjusts the computation offloading to the resource-limited cloudlet in order to reduce the task execution time. A cooperative offloading strategy proposed in [14] carries out the energy-traffic tradeoff problem to save energy and reduce the Internet data traffic of wireless local area networks (WLAN). The decentralized computation offloading game formulated in [15] helps develop an offloading strategy that analyzes the offloading decision of mobile devices with a reduced cloud computation cost. The offloading game formulated in [16] analyzes the computation offloading problem in cloudlet-based mobile cloud computing via multiple wireless APs to save transmission energy and reduce delays.

A two-dimensional anti-jamming communication scheme based on a deep Q-network algorithm as developed in [17] applies a deep convolution neural network to accelerate the learning speed with a large number of frequency channels. Reinforcement learning (RL) technique has been applied to mobile devices to detect malwares via mobile offloading. The mobile offloading strategy proposed in [18] applies Q-learning technique to help mobile devices derive the optimal offloading rates against smart attackers. An offloading algorithm based on Q-learning is proposed in [11] for the mobile device to update its offloading rate based on the observed state, which consists of the previous actions of the opponents and the channel gain in dynamic environments. In [19], our previous work in [11] is extended by deriving the Nash Equilibrium (NE) of the malware detection game and improving the performance with Dyna architecture. Compared to our previous work in [11], [19], we improve the malware detection performance with the hotbooting techniques and DQN technique.

III. Cloud-based Malware Detection Game

We consider the cloud-based malware detection of M mobile devices as shown in Fig. 1. Without loss of generality, mobile device i must process $C_i^{(k)}$ traces generated by the Apps at time k and chooses the proportion of the traces to offload to the security server at the cloud via the serving BS/AP, denoted by $x_i^{(k)}$, with $0 \leq x_i^{(k)} \leq 1$ and $1 \leq i \leq M$. More specifically, mobile device i processes all of the traces locally to detect malware if $x_i^{(k)} = 0$; it offloads all the traces

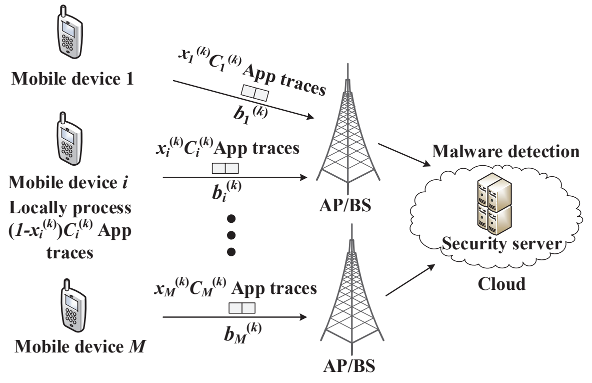

Mobile device 1
$x_1^{(k)}C_1^{(k)}$ App traces
$b_1^{(k)}$
Mobile device i
Locally process $(1-x_i^{(k)})C_i^{(k)}$ App traces
$x_i^{(k)}C_i^{(k)}$App traces
$b_i^{(k)}$
AP/BS
Malware detection
Security server
Cloud
Mobile device M
$x_M^{(k)}C_M^{(k)}$ App traces
$b_M^{(k)}$
AP/BS

Fig. 1. Cloud-based malware detection game with M mobile devices, in which mobile device i offloads $x_i^{(k)}C_i^{(k)}$ of the App traces to a security server via the serving AP/BS and locally processes the $\left(1 - x_i^{(k)}\right) C_i^{(k)}$ App traces to detect malware at time k.

to the security server if $x_i^{(k)} = 1$. $x_i^{(k)}C_i^{(k)}$ traces are uploaded to the security server and the rest $\left(1 - x_i^{(k)}\right) C_i^{(k)}$ traces are processed locally if $0 < x_i^{(k)} < 1$. For simplicity, the offloading rate $x_i^{(k)}$ is quantized into $N_x + 1$ equally spaced levels, i.e., $x_i^{(k)} \in \{\frac{l}{N_x}\}_{0 \leq l \leq N_x}$. Let $\mathbf{x}_{-i}^{(k)} = \left[x_j^{(k)}\right]_{1 \leq j \neq i \leq M}$ be the offloading strategies of other devices except i.

With the normalized computation resources compared to that of the mobile devices denoted by R, the security server applies a classification algorithm, such as the k-means algorithm to scan the $\sum_{m=1}^{M} x_m^{(k)} C_m^{(k)}$ App traces sent by the M mobile devices. Mobile device i receives $x_i^{(k)}C_i^{(k)} / \sum_{m=1}^{M} x_m^{(k)} C_m^{(k)}$ of the computation resources to scan the app traces at the server. Let $f(x)$ denote the malware detection accuracy function of the classification algorithm applied by the server to process x traces, and γ be the detection accuracy gain to the mobile device. Thus $f\left(\sum_{m=1}^{M} x_m^{(k)} C_m^{(k)}\right)$ represents the malware detection accuracy of the security server, which increases with the total size of the malware samples. Due to a limited computation capacity of the security server, we assume $f(x) = \log(1+x)$, whose increase rate decreases gradually.

Based on the malware detection accuracy, response speed, and the transmission cost, the utility of mobile device i at time k denoted by $u_i^{(k)}$ according to [11] is given by:

$$u_i^{(k)}\left(x_i^{(k)}, \mathbf{x}_{-i}^{(k)}\right) = \left(\frac{R^{(k)}}{\sum_{m=1}^{M} x_m^{(k)} C_m^{(k)}} - \frac{\omega}{b_i^{(k)}}\right) x_i^{(k)} C_i^{(k)} + \gamma \log\left(1 + \sum_{m=1}^{M} x_m^{(k)} C_m^{(k)}\right). \tag{1}$$

Important symbols are summarized in Table I.

TABLE I
SUMMARY OF SYMBOLS AND NOTATIONS

M	Number of mobile devices
${C_i}^{(k)}$	Amount of App traces for mobile device i at time k
${b_i}^{(k)}$	Transmission bandwidth of device i at time k
${x_i}^{(k)}$	Offloading rate of device i at time k
N_x	Quantized levels of offloading rates
ω	Coefficient of transmission cost
γ	Detection accuracy gain
R	Computation resources of server
p_i	Computation capacity of device i
$\mathbf{s}_i^{(k)}$	States of mobile device i at time k
$\mathbf{S}$	State space
u_i	Utility of mobile device i
x_i^*	Optimal offloading rate of device i
$\mathbf{x}_{-i}^*$	Optimal strategy set of other devices except i
Q_i	Q function of device i
$\boldsymbol{\varphi}$	State sequence
$\mathcal{D}$	Replay memory
θ	CNN filter weights
W	Size of the CNN inputs

IV. Malware Detection Based on Reinforcement learning

The repeated interactions among the M mobile devices in their offloading for the cloud-based malware detection can be formulated as a dynamic malware detection game. The optimal offloading strategy of a mobile device depends on the radio channel state and the size of generated App traces of the mobile devices, which are challenging for a mobile device to estimate. We present two reinforcement learning based malware detection schemes to improve the malware detection performance of the Q-learning based mobile offloading as developed in [11] and accelerate the learning speed of Q-learning in time-variant wireless networks as in the following.

A. Hotbooting Q-learning based Malware Detection

Similar to the Q-learning based malware detection scheme in [11], the mobile device maintains a Q-function denoted by $Q_i(\mathbf{s}, x)$, which is the expected long-term utility of the mobile device. The all-zero Q-value initialization in [11] is universally valid but not efficient. Therefore, we use a hotbooting technique that initializes the Q-value based on the training data obtained in advance in similar scenarios to decrease the random explorations at the beginning and thus accelerate the learning speed in the dynamic game.

More specifically, in the dynamic game, mobile device i receives its App traces $C_i^{(k)}$ at time k and quantizes them into N_x levels. The system state denoted by $\mathbf{s}_i^{(k)}$ consists of the current radio bandwidth, and the previous offloading rates of other devices, i.e., $\mathbf{s}_i^{(k)} = \left[\mathbf{x}_{-i}^{(k-1)}, b_i^{(k)}\right] \in \mathbf{S}$, where $\mathbf{S}$ is the vector space of all the possible states.

As shown in Algorithm 1, E cloud-based malware detection experiments are performed before the beginning of the game, in which mobile device i chooses its offloading rate $x_i^{(k)} \in \mathbf{X}$ with ε-greed policy, where $\mathbf{X}$ is the action set consisting of the feasible offloading rates in similar scenarios. The device observes the next state $\mathbf{s}_i^{(k+1)}$ and the current utility $u_i^{(k)}$. The mobile device i updates the Q-function at time k, according to the iterative Bellman equation as:

$$Q_i\left(\mathbf{s}_i^{(k)}, x_i^{(k)}\right) \leftarrow (1-\alpha) Q_i\left(\mathbf{s}_i^{(k)}, x_i^{(k)}\right) + \alpha\left(u_i\left(\mathbf{s}_i^{(k)}, x_i^{(k)}\right) + \delta V_i\left(\mathbf{s}_i^{(k+1)}\right)\right) \quad (2)$$

$$V_i\left(\mathbf{s}_i^{(k)}\right) = \max_{x_i^{(k)} \in \mathbf{X}} Q_i\left(\mathbf{s}_i^{(k)}, x_i^{(k)}\right), \quad (3)$$

where $\alpha \in (0, 1]$ is the learning rate and $\delta \in (0, 1]$ is the discount factor indicating the greedy behavior of the mobile device. The resulting Q-function denoted by Q^* is then used to initialize Q-value in the hotbooting-Q based algorithm.

Algorithm 1 Hotbooting preparation
```
 1: Initialize $\alpha, \delta, \beta, \mathbf{Q}_i = \mathbf{0}, \mathbf{V}_i = \mathbf{0}, \mathbf{s}_i^1 = \left[\mathbf{0}, b_i^1\right]$
 2: for $e = 1, 2, 3, ..., E$ do
 3:     Emulate a similar environment
 4:     Receive the traces with size $C_i$ and quantize them into $N_x + 1$ levels
 5:     for $k = 1, 2, 3, ...$ do
 6:         Choose $x_i^{(k)} \in \{\frac{l}{N_x}\}_{0 \le l \le N_x}$ with $\varepsilon$-greed policy
 7:         Obtain $u_i^{(k)}$ via (1)
 8:         Update $Q_i^*\left(\mathbf{s}_i^{(k)}, x_i^{(k)}\right)$ via (2)
 9:         Update $V_i^*\left(\mathbf{s}_i^{(k)}\right)$ via (3)
10:     end for
11: end for
```

As shown in Algorithm 2, the ε-greed policy is applied to choose the "optimal" offloading rate that maximizes the Q-function with a high probability $1-\varepsilon$, and other rates with a small probability, i.e., the offloading rate of mobile device i is chosen according to the following:

$$\Pr\left(x_i^{(k)} = \tilde{x}\right) = \begin{cases} 1-\varepsilon, & \text{if } \tilde{x} = \arg\max\limits_{x_i^{(k)} \in \mathbf{X}} Q_i\left(\mathbf{s}_i^{(k+1)}, x_i^{(k)}\right) \\ \frac{\varepsilon}{N_x}, & \text{otherwise.} \end{cases} \quad (4)$$

The mobile device offloads the $x_i^{(k)} C_i^{(k)}$ App traces to the security server, which uses the classification algorithm such as [3] to detect malware and sends the detection report back. Then mobile device i observes $\mathbf{x}_{-i}$ and the utility. The Q function is updated via (2) and (3).

B. DQN-based Malware Detection

However, the convergence time required by the Q-learning based offloading scheme increases with the number of the action-state space, which in turn increases with the number of mobile devices and the quantized levels of offloading rates. Therefore, we combine the Q-learning based scheme and deep learning to increase the detection accuracy and reduce the detection delay in the game with "high dimension". More specifically, a convolutional deep neural network is used to accelerate the convergence rate of Q-learning algorithm. The DQN-based malware detection updates the Q-function of mobile device i for each action-state pair given by:

$$Q_i(\mathbf{s}_i^{(k)}, x_i^{(k)}) = \mathbb{E}_{\mathbf{s}_i^{(k+1)}} \Big[u_i^{(k)} + \gamma \max_{x_i^{(k+1)} \in \mathbf{X}} Q_i \left(\mathbf{s}_i^{(k+1)}, x_i^{(k+1)} | \mathbf{s}_i^{(k)}, x_i^{(k)} \right) \Big]. \quad (5)$$

As shown in Fig. 2, the Q-value in (5) for each offloading rate x is estimated using the convolutional deep neural network, which consists of two convolutional (Conv) layers and two fully connected (FC) layers. The first Conv layer includes 20 filters each with size 3×3 and stride 1, and the second Conv layer has 40 filters each with size 2×2 and stride 1. Both convolutional layers use the rectified linear unit (ReLU) as the activation function. The first FC layer involves 180 rectified linear units, and the second FC layer has $N_x + 1$ units for the action set. According to [12], the filter weights of the four layers in the CNN at time k are denoted by $\theta^{(k)}$. The CNN parameters are summarized in Table II.

TABLE II
CNN PARAMETERS IN THE MOBILE OFFLOADING

Layer	Conv1	Conv2	FC1	FC1
Input	6×6	$4 \times 4 \times 20$	360	180
Filter size	2×2	2×2	/	/
Stride	1	1	/	/
Filter number	20	40	180	N_x
Activation	ReLU	ReLU	ReLU	ReLU
Output	$4 \times 4 \times 20$	$3 \times 3 \times 40$	180	$N_x + 1$

The state sequence of mobile device i is denoted by $\boldsymbol{\varphi}_i^{(k)}$ and consists of the current $W = 11$ system states, i.e., $\boldsymbol{\varphi}_i^{(k)} = \left(\mathbf{s}_i^{(k-W)}, \mathbf{s}_i^{(k-W-1)}, ..., \mathbf{s}_i^{(k)} \right)$, which is then reshaped into a 6×6 matrix as the input to the CNN. The

Algorithm 2 Hotbooting Q-learning based malware detection

```
1:  Initialize $\alpha, \delta, \mathbf{s}_i^1 = [\mathbf{0}, b_i^1]$, $\mathbf{V}_i = \mathbf{0}$, and $\mathbf{Q}_i = \mathbf{Q}_i^*$
2:  for $k = 1, 2, 3, ...$ do
3:      Receive the traces with size $C_i^{(k)}$ and quantize them into $N_x + 1$ levels
4:      Choose $x_i^{(k)}$ via (4)
5:      Send $x_i^{(k)} C_i^{(k)}$ App traces to the security server
6:      Detect malwares for the $\left(1 - x_i^{(k)}\right) C_i^{(k)}$ App traces
7:      Obtain $u_i^{(k)}$ via (1)
8:      Observe $\mathbf{s}_i^{(k+1)} = \left[\mathbf{x}_{-i}^{(k)}, b_i^{(k+1)}\right]$
9:      Update $Q_i\left(\mathbf{s}_i^{(k)}, x_i^{(k)}\right)$ via (2)
10:     Update $V_i\left(\mathbf{s}_i^{(k)}\right)$ via (3)
11: end for
```

outputs of the CNN are the estimated Q-value of each action, $Q\left(\boldsymbol{\varphi}_i^{(k)}, \mathbf{x}_i^{(k)} | \theta^{(k)}\right)$ for a given system state sequence, $\boldsymbol{\varphi}_i^{(k)}$. The CNN filter weights $\theta^{(k)}$ are updated at time k according to the experience replay.

The malware detection experience of mobile device i is denoted by $\mathbf{d}_i^{(k)} = \left(\boldsymbol{\varphi}_i^{(k)}, \mathbf{s}_i^{(k)}, u_i^{(k)}, \boldsymbol{\varphi}_i^{(k+1)} \right)$, and the replay memory at time k is given by $\mathcal{D} = \{\mathbf{d}_i^{(1)}, ..., \mathbf{d}_i^{(k)}\}$. The experience replay randomly chooses an experience $\mathbf{d}_i^{(j)} \in \mathcal{D}$, with $1 \le j \le k$. The CNN weights $\theta^{(k)}$ are updated according to the stochastic gradient algorithm.

The mean-squared error of the target values is minimized over the minibatch updates, and the loss function denoted by L is chosen as:

$$L\left(\theta^{(k)}\right) = \mathbb{E}_{\boldsymbol{\varphi}_i^{(k)}, x_i^{(k)}, u_i^{(k)}, \boldsymbol{\varphi}_i^{(k+1)}} \Bigg(u_i^{(k)} + \gamma \max_{x_i^{(k+1)} \in \mathbf{X}} Q\left(\mathbf{s}_i^{(k+1)}, x_i^{(k+1)}; \theta^{(k-1)}\right) - Q\left(\mathbf{s}_i^{(k)}, x_i^{(k)}; \theta^{(k)}\right) \Bigg)^2. \quad (6)$$

The stochastic gradient of the loss function is given by:

$$\nabla L\left(\theta^{(k)}\right) = -\mathbb{E}_{\boldsymbol{\varphi}_i^{(k)}, x_i^{(k)}, u_i^{(k)}, \boldsymbol{\varphi}_i^{(k+1)}} \Bigg[\Bigg(u_i^{(k)} + \gamma \max_{x_i^{(k+1)} \in \mathbf{X}} Q\left(\mathbf{s}_i^{(k+1)}, x_i^{(k+1)}; \theta^{(k-1)}\right) - Q\left(\mathbf{s}_i^{(k)}, x_i^{(k)}; \theta^{(k)}\right) \Bigg) \nabla Q\left(\mathbf{s}_i^{(k)}, x_i^{(k)}; \theta^{(k)}\right) \Bigg]. \quad (7)$$

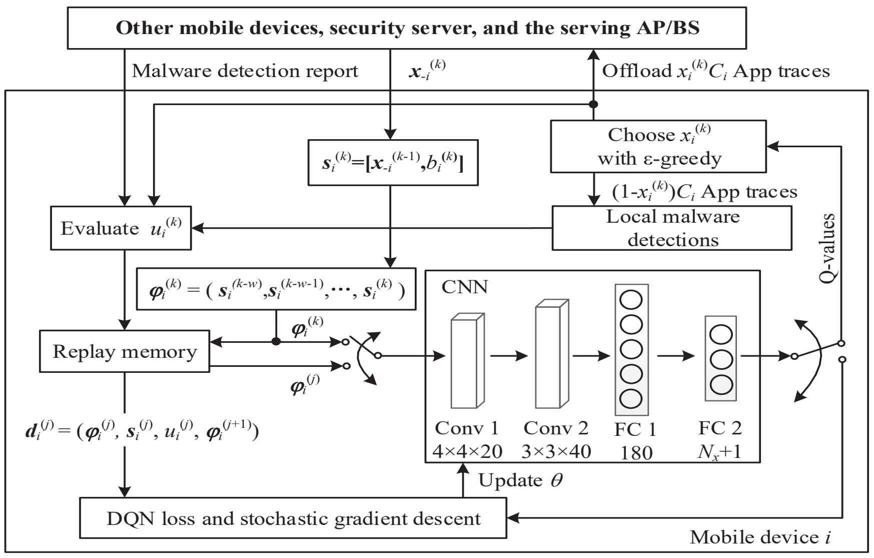
This process repeats N times at each time and $\theta^{(k)}$ are chosen according to the randomly selected experiences $\mathbf{d}_i^{(j)} \in \mathcal{D}$, as shown in Algorithm 3.

V. SIMULATION RESULTS

Simulations were performed to evaluate the performance of the mobile offloading schemes in the cloud-based malware detection based on the Markov chain-based channel model, with $M = 2$ mobile devices, C_1 and C_2 randomly chosen from $\{1.55, 1.50, 1.45, 1.40\}$, and $p_1 = p_2 = 0.08$, $\omega = 0.1$, $\gamma = 1.5$, and $R = 1$, if not specified otherwise. The radio bandwidths of the mobile devices were randomly chosen at each from $b_1 \in \{\frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}\}$ and $b_2 \in \{\frac{1}{10}, \frac{1}{9}, \frac{1}{8}, \frac{1}{7}, \frac{1}{6}\}$ in the simulations. Let $p_i^(k)$ denote the computation capacity of mobile device i at time k. Based on the transmission delay and the processing delay of the security server, the malware detection delay of mobile device i at time slot k denoted by $T_i^{(k)}$, is given by:

$$T_i^{(k)} = \max\left(\frac{x_i^{(k)} C_i^{(k)}}{b_i^{(k)}} + \frac{\sum_{m=1}^{M} x_m^{(k)} C_m^{(k)}}{R^{(k)}}, \ \frac{\left(1 - x_i^{(k)}\right) C_i^{(k)}}{p_i^{(k)}} \right). \quad (8)$$

As shown in Fig. 3(a), the hotbooting Q-based malware detection increases the detection accuracy and reduces the



Other mobile devices, security server, and the serving AP/BS
Malware detection report
$x_{-i}^{(k)}$
Offload $x_i^{(k)}C_i$ App traces
Choose $x_i^{(k)}$ with ε-greedy
$s_i^{(k)}=[x_{-i}^{(k-1)},b_i^{(k)}]$
$(1-x_i^{(k)})C_i$ App traces
Evaluate $u_i^{(k)}$
Local malware detections
Q-values
$\varphi_i^{(k)}=(s_i^{(k-w)},s_i^{(k-w-1)},\cdots,s_i^{(k)})$
CNN
$\varphi_i^{(k)}$
Replay memory
$\varphi_i^{(j)}$
$d_i^{(j)}=(\varphi_i^{(j)},s_i^{(j)},u_i^{(j)},\varphi_i^{(j+1)})$
Conv 1 4×4×20
Conv 2 3×3×40
FC 1 180
FC 2 N_x+1
Update θ
DQN loss and stochastic gradient descent
Mobile device i

Fig. 2. Illustration of the DQN-based malware detection.

Algorithm 3 Cloud-based malware detection for a mobile device with DQN

```
1:  **Initialize** $\theta, \alpha, \delta, \mathbf{x}_{-i}^0 = \mathbf{0}$
2:  **for** $k = 1, 2, 3, ...$ **do**
3:      Receive the traces with size $C_i^{(k)}$ and quantize them into $N_x + 1$ levels
4:      **if** $k \le M$ **then**
5:          Choose $x_i^{(k)} \in \{\frac{l}{N_x}\}_{0 \le l \le N_x}$ at random
6:      **else**
7:          $\boldsymbol{\varphi}_i^{(k)} = \left(\mathbf{s}_i^{(k-W)}, \mathbf{s}_i^{(k-W-1)}, \cdots, \mathbf{s}_i^{(k)}\right)$
8:          Obtain $Q\left(\boldsymbol{\varphi}_i^{(k)}, x_i^{(k)}\right)$ as the CNN outputs with $\boldsymbol{\varphi}^{(k)}$ and $\theta^{(k)}$.
9:          Choose $x_i^{(k)}$ via (4)
10:     **end if**
11:     Send $x_i^{(k)} C_i^{(k)}$ App traces to the security server
12:     Detect malware for the $\left(1 - x_i^{(k)}\right) C_i^{(k)}$ App traces
13:     Obtain $u_i^{(k)}$ via (1)
14:     $\boldsymbol{s}_i^{(k+1)} = \left[\mathbf{x}_{-i}^{(k)}, b_i^{(k+1)}\right]$
15:     $\boldsymbol{\varphi}_i^{(k+1)} = \left(\mathbf{s}_i^{(k-W+1)}, \mathbf{s}_i^{(k-W)}, ..., \mathbf{s}_i^{(k+1)}\right)$
16:     $\mathcal{D} \longleftarrow \mathcal{D} \cup \{\boldsymbol{\varphi}_i^{(k)}, \mathbf{s}_i^{(k)}, u_i^{(k)}, \boldsymbol{\varphi}_i^{(k+1)}\}$
17:     **for** $j = 1, 2, ..., N$ **do**
18:         Select $\left(\boldsymbol{\varphi}_i^{(j)}, \mathbf{s}_i^{(j)}, u_i^{(j)}, \boldsymbol{\varphi}_i^{(j+1)}\right) \in \mathcal{D}$ randomly
19:         Calculate $L\left(\theta^{(k)}\right)$ via (6)
20:     **end for**
21:     Update $\theta^{(k)}$ via (7)
22: **end for**
```

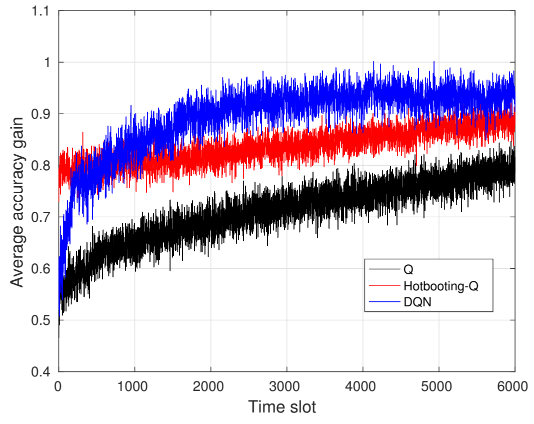
detection delay compared with the Q-learning based scheme. For instance, the detection accuracy gain of the hotbooting-Q algorithm is 15.1% higher than Q-learning at the 3000-th time slot. The DQN-based malware detection further improves the detection accuracy, which is 24.5% higher than Q-learning.

As shown in Fig. 3(b), the DQN-based malware detection has the fastest learning rate, the highest malware detection accuracy, the shortest detection delay, and the highest utility of the mobile device. For example, the detection delay of DQN-based scheme reduces by 24.6% and 35.3% at the 2000-th time slot, respectively, compared with hotbooting-Q and Q-learning based malware detection schemes.

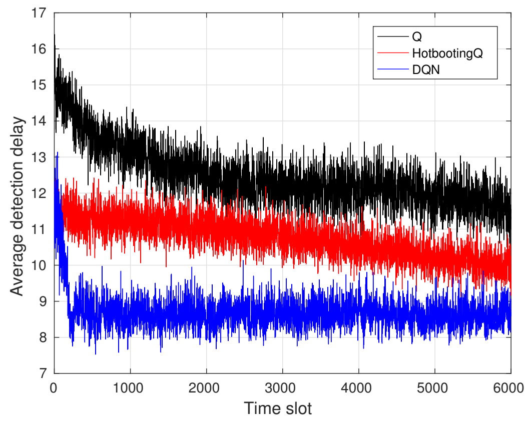
In Fig. 3(c), the DQN based malware detection scheme significantly outperforms the Q-learning based scheme with higher utility. For instance, the DQN based strategy increases the utility of the mobile device by 16.3%, compared with the Q-learning scheme. Furthermore, the hotbooting Q-learning strategy has a higher utility than the other strategies at the beginning of the game, which is 31.0% higher than the Q-learning based strategy.

VI. Conclusion

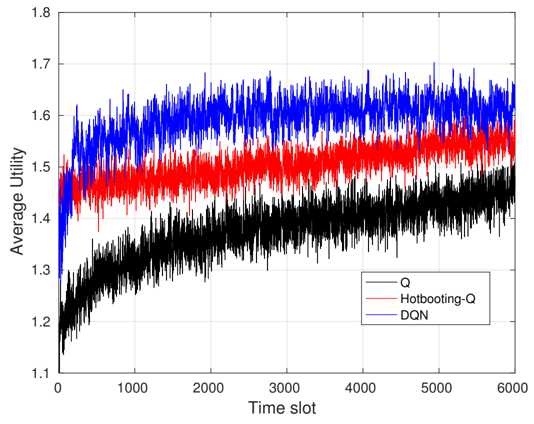
In this paper, we have formulated a cloud-based malware detection game, in which the mobile devices compete for the limited radio transmission resource and cooperate to improve the malware detection accuracy of the security server. A hotbooting-Q based mobile offloading strategy has been proposed to improve the malware detection performance compared to the Q-learning based scheme, and the performance is further improved by the DQN-based malware detection. As shown in the simulation results, the proposed offloading schemes accelerate the learning speed, increase the malware detection accuracy, reduce the detection delay, and thus improve the utility. For instance, the DQN-based malware



(a) Detection accuracy gain



(b) Detection delay



(c) Utility of the mobile device

Fig. 3. Performance of the reinforcement learning based malware detection in the dynamic game with the radio bandwidths of the two mobile devices randomly chosen from $\{\frac{1}{7},\frac{1}{6},\frac{1}{5},\frac{1}{4},\frac{1}{3}\}$ and $\{\frac{1}{10},\frac{1}{9},\frac{1}{8},\frac{1}{7},\frac{1}{6}\}$, respectively.

detection increases the detection accuracy gain and the utility of the mobile device by 24.8% and 17.5%, respectively, after 3500 time slots compared with the Q-learning based strategy.

REFERENCES

[1] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, vol. 20, no. 1, pp. 343–357, Jan. 2016.

[2] A. S. Shamili, C. Bauckhage, and T. Alpcan, "Malware detection on mobile devices using distributed machine learning," in *Proc. Int'l Conf. Pattern Recognition*, pp. 4348–4351, Istanbul, Turkey, Aug. 2010.

[3] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for android," in *Proc. ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, pp. 15–26, Chicago, Oct. 2011.

[4] S. Yerima, S. Sezer, G. McWilliams, and I. Muttik, "A new android malware detection approach using Bayesian classification," in *Proc. IEEE Int'l Conf. Advanced Information Networking and Applications (AINA)*, pp. 121–128, Barcelona, Spain, Mar. 2013.

[5] S. Liang and X. Du, "Permission-combination-based scheme for android mobile malware detection," in *IEEE Int'l Conf. Commun. (ICC)*, pp. 2301–2306, Sydney, NSW, Australia, Jun. 2014.

[6] X. Du, Y. Xiao, M. Guizani, and H. H. Chen, "An effective key management scheme for heterogeneous sensor networks," *Ad Hoc Networks*, vol. 5, no. 1, pp. 24–34, Jan. 2007.

[7] X. Du and H. H. Chen, "Security in wireless sensor networks," *IEEE Wireless Commun.*, vol. 15, no. 4, pp. 60–66, Aug. 2008.

[8] Y. Xiao, H. H. Chen, X. Du, and M. Guizani, "Stream-based cipher feedback mode in wireless error channel," *IEEE Tran. Wireless Commun.*, vol. 8, no. 2, pp. 622–626, Feb. 2009.

[9] X. Du, M. Guizani, Y. Xiao, and H. H. Chen, "A routing-driven elliptic curve cryptography based key management scheme for heterogeneous sensor networks," *IEEE Trans. Wireless Commun.*, vol. 8, no. 3, pp. 1223–1229, Mar. 2009.

[10] A. Houmansadr, S. A. Zonouz, and R. Berthier, "A cloud-based intrusion detection and response system for mobile phones," in *Proc. IEEE Int'l Conf. Dependable Systems and Networks Workshop*, pp. 31 – 32, Hong Kong, China, Jun. 2011.

[11] Y. Li, J. Liu, Q. Li, and L. Xiao, "Mobile cloud offloading for malware detections with learning," in *Proc. IEEE Int'l Conf. Computer Commun. (INFOCOM) BigSecurity Wksp.*, pp. 226–230, Hongkong, China, Apr. 2015.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Jan. 2015.

[13] V. Cardellini, V. D. N. Personé, V. D. Valerio, et al., "A game-theoretic approach to computation offloading in mobile cloud computing," *Springer Mathematical Programming*, vol. 157, no. 2, pp. 421–449, Jun. 2016.

[14] J. Song, Y. Cui, M. Li, and J. Qiu, "Energy-traffic tradeoff cooperative offloading for mobile cloud computing," in *Quality of Service*, pp. 284–289, Hong Kong, China, May 2014.

[15] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, Apr. 2015.

[16] X. Ma, C. Lin, X. Xiang, and C. Chen, "Game-theoretic analysis of computation offloading for cloudlet-based mobile cloud computing," in *Proc. ACM Int'l Conf. Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 271–278, Cancun, Mexico, Nov. 2015.

[17] G. Han, L. Xiao, and H. V. Poor, "Two-dimensional anti-jamming communication based on deep reinforcement learning," in *Proc. IEEE Int'l Conf. Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, Mar. 2017.

[18] L. Xiao, C. Xie, T. Chen, and H. Dai, "A mobile offloading game against smart attacks," *IEEE Access*, vol. 4, pp. 2281–2291, May 2016.

[19] L. Xiao, Y. Li, X. Huang, and X. Du, "Cloud-based malware detetction game for mobile device with offloading," *IEEE Tran. Mobile Computing, in press.*