Intelligent Mining Pool Selection in the Case of Unobservable Block Withholding Attack

Kentaro Fujita, Yuanyu Zhang, Masahiro Sasabe and Shoji Kasahara
Graduate School of Science and Technology, Nara Institute of Science and Technology,
8916-5 Takayama-cho, Ikoma, Nara 630-0192, Japan.
fujita.kentaro.fk0@is.naist.jp, {yy90zhang, m-sasabe, kasahara}@ieee.org

***Abstract*—In Proof-of-Work (PoW) blockchain systems, miners can select the pool to join for maximizing their revenues, leading to the mining pool selection problem. A pool can infiltrate its mining power into other pools for only obtaining additional revenues without substantially contributing to their mining work. This is called Block WithHolding (BWH) attack and significantly affects the pool selection of miners. We therefore investigate the mining pool selection issue under the BWH attack in this paper. Previous studies rely on an arguable and impractical assumption that miners can observe the attack to calculate their payoffs. This paper however focuses on *unobservable* BWH attack and applies reinforcement learning (RL) techniques to analyze the intelligent pool selection of miners. We adopt three typical RL models, i.e., Q-Learning (QL), Deep Q Network (DQN) and Advanced Actor-Critic (A2C) to dynamically learn the optimal pool selection policies of an intelligent miner, and use a discrete-event simulator to measure the reward of the miner. Simulation results are also provided to demonstrate the learning performances of the three models.**

***Index Terms*—Blockchain, Mining Pool, Block Withholding Attack, Reinforcement Learning**

I. Introduction

Proof-of-Work (PoW) blockchains rely on mining to ensure the tamper-proof feature [1]. Mining is a competition, where only the first participant finding the latest *valid* block (called full PoW) is the winner and receives rewards. Valid blocks have hash values satisfying a *system-wide* difficulty requirement, e.g., the first leading n bits must be zeros. Finding valid blocks is computationally expensive, because the only way is to relentlessly try hash calculations with different values called nonce. Thus, solo miners can hardly win the competition and usually form mining pools to compete with other pools and miners.

Each pool has a *pool-wide* difficulty, which is easier to meet than the system-wide difficulty. Miners submit blocks satisfying the pool-wide difficulty (called partial PoW) to the pool manager in exchange for rewards. The pool manager distributes rewards to the miners based on the number of submitted partial PoWs (PPoWs). Miners can select the pool to join in order to maximize their revenues. This is called mining pool selection and has been studied in [2]–[4]. For example, the authors in [2] applied the evolutionary game theory to model the pool selection process of miners and obtained stable pool population states. In [3], [4] the coalition formation game theory was adopted to investigate the formation of mining pools and the properties of the pools.

Block WithHolding (BWH) attack is another critical issue due to the existence of competing pools. In BWH attack, a pool dispatches some of its miners as spies into another pool, who work as normal miners in the attacked pool in most cases, submitting PPoWs and obtaining rewards. However, when they find a full PoW (FPoW), they just discard it. In this way, the spy miners only obtain rewards without substantially contributing to the mining of the attacked pool. The obtained revenues will be re-distributed among the miners of the *attacking* pool (including the spy miners). Thus, pools can obtain more revenues by launching BWH attack to other pools [5]. Since BWH attack has significant impacts on the pool selection, this paper therefore investigates the mining pool selection problem in the presence of BWH attack.

Some recent work has been done to examine the mining pool selection under BWH attack [6], [7]. In [6], the authors applied the evolutionary game theory to model the mining pool selection process of miners. They investigated how the mining strategies can be changed to drive the populations of pools to stable states. However, the authors ignored the rewards from attacked pools to simplify the analysis, leaving the incentive for launching the BWH attack arguable. Taking the revenues from attacked pools into consideration, the authors in [7] also investigated the mining pool selection problem from the perspective of evolutionary game, providing more practical and meaningful insights into the problem.

Although the above studies represent significant research progress in this area, they rely on an arguable and impractical assumption: Miners can observe the BWH attack to calculate their payoffs. In practice, the BWH attack is *unobservable*, because no attackers are willing to expose them. Due to the lack of attack information, the formulation of miner payoffs is not available and thus the evolutionary game theory is no longer applicable. Therefore, this paper solves the problem from a new perspective, i.e., the perspective of Reinforcement Learning (RL). To be specific, we consider a PoW blockchain system with one RL miner and a set of non-RL miners. We first consider a simple case where only the RL miner switches mining pools and adopt the Q-Learning (QL) to dynamically learn the optimal pool selection polices. We then focus on a general case where all miners can switch their pools. In this case, Deep Q Network (DQN) and Advanced Actor-Critic (A2C) are applied to address the issue of huge state space. To estimate the reward of the RL miner, we use a discrete-

event simulator called PoolSim [8] to model the mining, pool selection and BWH attack inside the system. Simulation results and comparisons with random and deterministic pool selection methods are also provided to show the performances of the RL-based pool selection methods.

The remainder of this paper is organized as follows. We present the RL-based mining pool selection methods in Section II. In Section III, we present the evaluation scenario and numerical results. Finally, we conclude this paper in Section IV.

II. Pool Selection based on Reinforcement Learning

This section presents our RL-based mining pool selection methods under unobservable BWH attack. We first give a brief overview of QL, DQN and A2C, and then introduce the learning flow of our RL-based solutions.

A. Reinforcement Learning (RL)

RL is a machine learning framework for an agent to learn the optimal policies in response to an environment. Let $\mathcal{S}$ and $\mathcal{A}$ denote the environment state space and action set, respectively. At time t, the agent observes the environment state $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$ according to a policy $\pi(s_t, a_t)$, which specifies the probability of taking each action a_t. As a result, the agent receives reward $r_t \in \mathcal{R}$ and the environment moves to the next state s_{t+1}. The value of taking action a_t at state s_t is measured by the action-value function $Q(s_t, a_t)$. The value of being at state s_t when following the policy π is given by the state-value function $V^\pi(s_t) = \sum_{a_t} \pi(s_t, a_t) Q(s_t, a_t)$. Value functions represent the future expected discounted cumulative reward. The goal of the agent is to find the optimal policies π^* that maximize the value functions. RL models can be value-based, policy-based or a mixture. Value-based models try to find the value functions and then choose the policy with the largest value. Policy-based methods try to optimize the policy directly without using the value functions as a middleman. For a detailed introduction of RL, please refer to [9], [10].

1) QL: QL is value-based and iteratively updates the action values $Q(s_t, a_t)$ to the optimal as follows:

$$\underbrace{Q(s_t, a_t)}_{new} \leftarrow \underbrace{Q(s_t, a_t)}_{old} + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - \underbrace{Q(s_t, a_t)}_{old}),$$

where $r_t + \gamma \max_a Q(s_{t+1}, a)$ is the future discounted reward, α is the learning rate and γ is the discount rate. In general, QL agents apply the ϵ-greedy method to select an action at each time, where the action with the largest Q value is selected with probability $1 - \epsilon$ and other actions with probability ϵ. ϵ decreases as the learning goes on, meaning that the action with the largest Q value is more likely to be selected.

2) DQN: QL agents need to store the action values for all states, requiring a huge memory space. DQN was invented to reduce the memory usage by estimating the action values using neural networks. DQN uses two neural networks (i.e., Q-network with parameter θ and target network with parameter θ^-) to ensure stable learning. The parameters are updated in a way that minimizes the following loss function:

$$L = \left(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta^-) - Q(s_t, a_t; \theta) \right)^2.$$

Note that θ^- is a copy of θ from a few steps ago and only θ is updated during learning.

3) A2C: A2C is a type of Actor-Critic model, which is a mixture of value-based and policy-based models. The actor applies the policy-based method to find the current optimal policy. The critic adopts the value-based method to evaluate the value of the actor's policy and notifies the actor the direction of updating the policy. In A2C, the critic uses an Advantage function as the value function to measure how good action a_t is *compared with other actions* at state s_t. Formally, the Advantage function is given by

$$\begin{aligned} A^\pi(s_t, a_t) &= Q^\pi(s_t, a_t) - V^\pi(s_t, a_t) \\ &= \left\{ \sum_{i=0}^{k-1} \gamma^i r_{i+1} + \gamma^k V^\pi(s_{t+k}) \right\} - V^\pi(s, a), \end{aligned}$$

where k is the advantage step. The actor and critic can be independent or integrated. In this paper, we use the integrated type, where the actor and critic work as a whole by sharing one neural network with parameter θ. The loss functions to update θ is as follows:

$$\begin{aligned} L =& \{Q^\pi(s, a; \theta_c) - V^\pi(s; \theta_c)\}^2 - \mathbb{E}[\log \pi_{\theta_a}(s, a) A^\pi(s)] \\ & - \sum_{a \in \mathcal{A}} \pi_{\theta_a}(s, a) \log \pi_\theta(s, a). \end{aligned}$$

For fast and synchronous learning, A2C uses multiple processors to interact with the environment in parallel.

B. RL Model for Pool Selection

We consider a PoW blockchain system consisting of multiple pools, each of which can launch the BWH attack to other pools. The system parameters are as follows:

- $\mathcal{N}$: the set of miners with $|\mathcal{N}| = N$ (one RL miner and $N - 1$ non-RL miners).
- $\mathcal{M}$: the set of mining pools with $|\mathcal{M}| = M$.
- $\boldsymbol{h} = [h_1, h_2, \ldots, h_N]$: the miner hash rate profile with h_i being the hash rate of miner $i \in \mathcal{N}$ and $\sum_i^N h_i = 1$.
- $\mathcal{H} = \{\boldsymbol{h} : \sum_i^N h_i = 1\}$: the set of hash rate profiles.
- $\boldsymbol{d} = [d_1, d_2, \ldots, d_M]$: the PPoW difficulty profile with d_j corresponding to pool $j \in \mathcal{M}$.
- D: the FPoW difficulty.

We first focus on the simple case, where *only* the RL miner can switch the pools that he/she stays in. We then focus on the general case, where all the miners can switch pools.

1) Simple Case: In this case, the RL miner adopts the QL method in Section II-A1 to determine the optimal pool selection policy. The action space is $\mathcal{A} = \mathcal{M}$, where each element $a \in \mathcal{A}$ represents the pool that the RL miner selects. Since only the RL miner can switch pools, we can define

the state as the pool in which the RL miner currently stay. Thus, the state space is $\mathcal{S}_{simple} = \mathcal{M}$. Since the BWH attack is unobservable, miners can not calculate their immediate reward r_t after selecting action a_t at time s_t. Thus, we use the simulator PoolSim in [8] (see Section II-C) to obtain the reward r_t of the RL miner.

2) General Case: In this case, the RL miner switches pools based on RL methods, while the other miners independently and randomly switch their pools with equal probability p. If we follow the definition of the state space in the simple case, the state space would be $\mathcal{S} = \mathcal{M}^N$ of size M^N. When N and M are large, the state space becomes extremely huge. Thus, in this case, we adopt the DQN and A2C (the integrated type) methods for the RL miner to obtain the optimal pool selection policy. In addition, we introduce a new definition of the state space, which combines the current pool index of the RL miner and the hash rate profile. Formally, we define the state space by $\mathcal{S}_{general} = \mathcal{M} \times \mathcal{H}$, where $\times$ represents the Cartesian product. The action space is also $\mathcal{A}$ in this case and PoolSim is used as well to determine the reward r_t.

C. PoolSim

PoolSim is a discrete-event simulator to simulate the mining process in a PoW blockchain system [8]. The basic event in PoolSim is finding a PPoW and the time interval between two successive PPoW events of a miner follows an exponential distribution with parameter proportional to his/her hash rate. PoolSim uses a global event queue to store the PPoW events of all miners and processes the events chronologically. When processing a PPoW, PoolSim checks whether the PPoW is an FPoW. If yes, some reward distribution function will be called to distribute the reward among the miners in the pool that submitting the FPoW. We adopt the proportional reward distribution function in this paper, where the reward of a miner is proportional to the ratio between the number of PPoWs he/she submits to the total number of PPoWs received by the pool manager. BWH attack can also be implemented in PoolSim by simply ignoring the FPoWs of the spy miners.

D. Flow of Simulation and Learning

The entire flow of simulation and learning is shown in Algorithm 1. In the initialization stage, we determine the system parameters $\{\mathcal{N}, \mathcal{M}, \boldsymbol{h}, \boldsymbol{d}, D\}$ and action space $\mathcal{A}$. In particular, we randomly generate the hash rate of each miner. For RL parameters, we initialize the Q value $Q(s, a)$ of each state-action pair (s, a) to a random value in $[0, 1]$ (simple case), and initialize the neural network parameter θ used in DQN and A2C (general case). In addition, we randomly assign each miner to one of the pools to generate the initial state s. After the initialization, the flow is divided into $T_{episode}$ *episodes*, each consisting of T_{round} *rounds*. Here, a round corresponds to the discovery of one FPoW. In each episode, we record the cumulative reward R earned by the RL miner to evaluate his/her learning performance. Also, the parameter ϵ in the ϵ-greedy action selection is initialized for the QL and DQN methods. In each round, the RL miner selects an action a based on the RL model (e.g., ϵ-greedy), executes the simulator PoolSim, measures his/her reward r and updates the RL parameters (i.e., Q value $Q(s, a)$ for the simple case and θ for the general case).

Algorithm 1 Simulation and Learning Flow

Initialization: $\{\mathcal{N}, \mathcal{M}, \boldsymbol{h}, \boldsymbol{d}, D, \mathcal{A}\}$, $\{Q(s, a), \forall a, s\}$ (QL), θ (DQN and A2C), state s, $round \leftarrow 1$, $episode \leftarrow 1$;
while $episode \leq T_{episode}$ **do**
  $R \leftarrow 0$, $\epsilon \leftarrow \frac{0.5}{episode}$ (ϵ-greedy for QL and DQN);
  while $round \leq T_{round}$ **do**
    RL miner selects an action a at current state s;
    Environment moves to the next state s';
    RL miner executes PoolSim;
    RL miner obtains r and updates $R \leftarrow R + r$;
    RL miner updates Q value $Q(s, a)$ (simple case);
    RL miner updates θ (general case);
    $round \leftarrow round + 1$, $s \leftarrow s'$;
  end
  $episode \leftarrow episode + 1$;
end

TABLE I: Parameter Setting

(a) Parameter setting.

Parameter	Value
Number of miners N	100
Number of pools M	2
FPoW difficulty D	10000
Pool 1 difficulty d_1	10
Pool 2 difficulty d_2	10
Initial ϵ (QL and DQN)	0.5
$T_{episode}$	500
T_{round}	1000

(b) Miner hash rate distribution.

Parameter	Value
Mean	20
Standard deviation	5
Min	0.001
Max	100

III. Numerical Results

In this section, we investigate the performances of our RL-based pool selection methods.

A. Evaluation Scenario and Parameter Settings

We aim to explore how the Cumulative Reward Per Episode (CRPE) of the RL miner varies as the learning goes on (i.e., the episode increases). Without loss of generality, we assume the reward for an FPoW is 1. Like [2], [5], [6], we consider the typical case with two mining pools. In particular, we assume one pool (say Pool 1) attacks the other (say Pool 2) using 20% of its miner population (e.g., 2 miners out of 10). For comparison, we consider other two pool selection methods, i.e., random selection, where the RL miner randomly selects one of the two pools with equal probability 0.5, and deterministic selection, where the RL miner stays in one of the two pools all the time. We use the parameters in Table Ia and generate the hash rate of each non-RL miner based on the distribution in Table Ib. The hash rate of the RL miner is fixed as the average rate of general miners (i.e., 20).

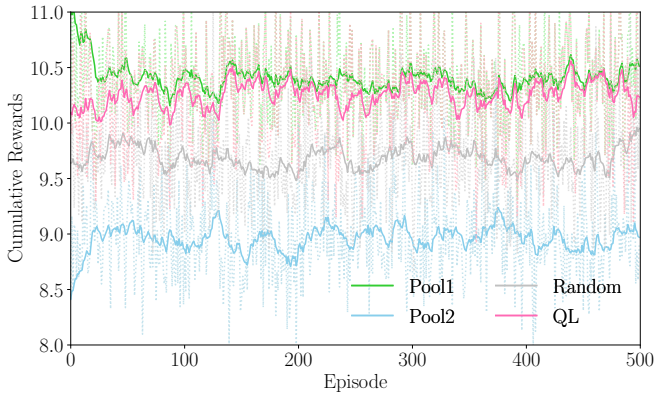

Fig. 1: CRPE of simple case.

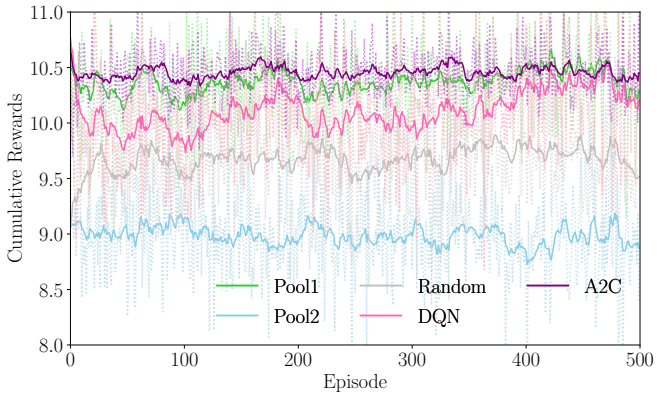

Fig. 2: CRPE of general case.

B. Evaluation Results

We first investigate the simple case with QL-based pool selection. In this case, the learning rate is set as $\alpha = 0.2$ and the discount rate is $\gamma = 0.99$. Fig. 1 shows the CRPE of the RL miner versus the index of episode, when he/she applies the QL-based selection (QL), random selection (Random) and deterministic selection (Pool1 and Pool2). We smoothen the raw data (dotted lines) using an exponential moving average with $weight = 0.9$ (solid lines). We can see from Fig. 1 that staying in Pool 1 is the optimal strategy to maximize the CRPE performance, while staying in Pool 2 is the worst. This is because launching the BWH attack yields an advantage for Pool 1 in the mining competition [7]. Intuitively, random selection results in moderate CRPE performance between the optimal and worst ones. The most important finding in Fig. 1 is that the CRPE performance of the QL-based selection will finally converges to the optimal one. This suggests that the QL method is effective for miners to gradually learn the optimal pool selection policy in the case of unobservable BWH.

Next, we focus on the general case with DQN and A2C respectively. In this case, non-RL miners switch pools independently and randomly with equal probability 0.2. We set the discount rate for both DQN and A2C as $\gamma = 0.99$, and the advantage step k and number of processors used by A2C as 2 and 4, respectively. Fig. 2 summarizes the CRPE results of the DQN-based and A2C-based as well as the other three selection methods. We can see from Fig. 2 that both the CRPE values of the DQN-based and A2C-based methods converge to that of staying in Pool 1, implying that both methods can be applied in the general case to dynamically learn the optimal pool selection policy in the case of unobservable BWH. More importantly, we can see that A2C-based selection outperforms the DQN-based selection in terms of both the CRPE performance and convergence rate.

IV. Conclusions

This paper has investigated the mining pool selection problem under unobservable Block WithHolding (BWH) attack from the novel Reinforcement Learning (RL) perspective. Three typical RL models, i.e., Q-Learning (QL), Deep Q Network (DQN) and Advantage Actor-Critic (A2C) have been applied to find the optimal pool selection policies of miners. The results in this paper have shown that all the three models are effective to learn the optimal pool selection policies and A2C outperforms DQN in terms of both the reward and convergence performances. For more valuable findings, we plan to apply our RL-based pool selection models in real-world blockchain systems (e.g., Bitcoin) to investigate the learning performances.

Acknowledgments

This work was supported in part by KAKENHI (A) under Grant 19H01103, SCAT Research Grant and The Telecommunications Advancement Foundation.

References

[1] J. Garay, A. Kiayias, and N. Leonardos, "The Bitcoin Backbone Protocol: Analysis and Applications," in *Proc. of Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2015, pp. 281–310.

[2] X. Liu, W. Wang, D. Niyato, N. Zhao, and P. Wang, "Evolutionary Game for Mining Pool Selection in Blockchain Networks," *IEEE Wireless Communications Letters*, vol. 7, no. 5, pp. 760–763, 2018.

[3] Y. Lewenberg, Y. Bachrach, Y. Sompolinsky, A. Zohar, and J. S. Rosenschein, "Bitcoin Mining Pools: A Cooperative Game Theoretic Analysis," in *Proc. of International Conference on Autonomous Agents and Multiagent Systems*. Citeseer, 2015, pp. 919–927.

[4] L. Brünjes, A. Kiayias, E. Koutsoupias, and A.-P. Stouka, "Reward Sharing Schemes for Stake Pools," in *Proc. of IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2020, pp. 256–275.

[5] I. Eyal, "The Miner's Dilemma," in *Proc. of IEEE Symposium on Security and Privacy*, 2015, pp. 89–103.

[6] S. Kim and S.-G. Hahn, "Mining Pool Manipulation in Blockchain Network over Evolutionary Block Withholding Attack," *IEEE Access*, vol. 7, pp. 144 230–144 244, 2019.

[7] K. Fujita, Y. Zhang, M. Sasabe, and S. Kasahara, "Mining Pool Selection Problem in the Presence of Block Withholding Attack," in *Proc. of IEEE International Conference on Blockchain (Blockchain)*, November 2020, pp. 321–326.

[8] S. M. Werner and D. Perez, "PoolSim: A Discrete-Event Mining Pool Simulation Framework," in *Mathematical Research for Blockchain Economy*. Springer, 2020, pp. 167–182.

[9] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[10] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A Survey of Optimization Methods from a Machine Learning Perspective," *IEEE Transactions on Cybernetics*, 2019.