TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 03/14 pp1009–1022
DOI: 10.26599/TST.2022.9010045
Volume 28, Number 6, December 2023

Optimized Consensus for Blockchain in Internet of Things Networks via Reinforcement Learning

Yifei Zou, Zongjing Jin, Yanwei Zheng*, Dongxiao Yu, and Tian Lan

Abstract: Most blockchain systems currently adopt resource-consuming protocols to achieve consensus between miners; for example, the Proof-of-Work (PoW) and Practical Byzantine Fault Tolerant (PBFT) schemes, which have a high consumption of computing/communication resources and usually require reliable communications with bounded delay. However, these protocols may be unsuitable for Internet of Things (IoT) networks because the IoT devices are usually lightweight, battery-operated, and deployed in an unreliable wireless environment. Therefore, this paper studies an efficient consensus protocol for blockchain in IoT networks via reinforcement learning. Specifically, the consensus protocol in this work is designed on the basis of the Proof-of-Communication (PoC) scheme directly in a single-hop wireless network with unreliable communications. A distributed MultiAgent Reinforcement Learning (MARL) algorithm is proposed to improve the efficiency and fairness of consensus for miners in the blockchain system. In this algorithm, each agent uses a matrix to depict the efficiency and fairness of the recent consensus and tunes its actions and rewards carefully in an actor-critic framework to seek effective performance. Empirical results from the simulation show that the fairness of consensus in the proposed algorithm is guaranteed, and the efficiency nearly reaches a centralized optimal solution.

Key words: consensus in blockchain; Proof-of-Communication (PoC); MultiAgent Reinforcement Learning (MARL); Internet of Things (IoT) networks

1 Introduction

The past decades have witnessed rapid development and wide deployment of Internet of Things (IoT) networks in various scenarios, such as automated industrial systems[1], smart home applications[2], and autonomous driving services[3]. By contrast, the massive and heteroid devices in IoT networks take additional challenges for these devices to reach valid agreements efficiently. To solve this problem, the blockchain has been considered an important technique to maintain trust and secure the environment in a decentralized IoT network[4–7]. Specifically, the devices in blockchain reach valid agreements periodically via the consensus protocol. The agreements achieved by the devices are then stored in the distributed ledgers on the chain structure. Specifically, the response time and the security of high-level applications provided by the blockchain system are highly determined by the efficiency and reliability of the consensus protocol, and the distributed ledgers on the chain prevent tampering with the agreements. Thus, an efficient consensus protocol for blockchain in IoT networks is always preferred.

Various protocols have been proposed due to the importance of a fast and reliable consensus protocol in the blockchain. These protocols can

• Yifei Zou, Zongjing Jin, Yanwei Zheng, and Dongxiao Yu are with the Institute of Intelligent Computing, School of Computer Science and Technology, Shandong University, Qingdao 266237, China. E-mail: yfzou@sdu.edu.cn; 202015073@mail.sdu.edu.cn; zhengyw@sdu.edu.cn; dxyu@sdu.edu.cn;
• Tian Lan is with the Department of Electrical and Computer Engineering, George Washington University, Washington, DC 20052, USA. E-mail: tlan@gwu.edu.
∗ To whom correspondence should be addressed.

Manuscript received: 2022-07-22; revised: 2022-08-29; accepted: 2022-10-04

be mainly divided into two groups according to their communication models. Assuming that reliable communications exist between each pair of devices, a series of protocols are proposed to support the complex applications in reality, such as Proof-of-Work (PoW)[8, 9], Proof-of-Activity (PoA)[10], Practical Byzantine Fault Tolerance (PBFT)[11], and HotStuff[12]. These famous protocols effectively support their high-level applications. However, their high resource consumption and requirement for reliable communications have become the largest obstacles to implementing these protocols in IoT networks because most of the devices in IoT networks are lightweight, battery-operated and communicate via an unreliable wireless channel environment. Another group of works is specifically designed for lightweight devices in IoT with an unreliable communication model. For example, in Refs. [13–16], miners achieve the consensus with a Proof-of-Communication (PoC) scheme based on the physical interference communication model. Specifically, after considering the collision, interference, and failure in the wireless channel, the devices in Refs. [13, 14] adapt their transmission probability by statistically randomized algorithms and the tree structures are built in Refs. [15, 16], both providing reliable communications between devices. A PoC scheme is then adopted by devices to achieve the consensus. However, the statistically randomized algorithms in Refs. [13, 14] only approach the optimal solution with a constant factor, and the tree constructions in Refs. [15, 16] increase the time complexity for a consensus. An efficient consensus protocol is designed in this paper for blockchain in an unreliable communication model.

Notably, optimizing the efficiency of consensus protocol is difficult, especially in a distributed wireless network, because the consensus is a global state in which all devices agree on the same decision/opinion. By contrast, only local knowledge can be used for each device to reach the global agreement, and the communications between devices are unreliable due to the open access wireless channel. The previously distributed algorithms clarify the direct realization of the consensus in an unreliable wireless network. However, their performance still approximates the optimal solution with some important factors. For example, a two-round leader election phase is proposed in Ref. [13] to elect a leader for making the agreement. As proven in Ref. [13], the two-round leader election is only successful with a probability of $1/16$. Thus, the algorithm in Ref. [13] takes 32 rounds in expectation to achieve a consensus, and this result is far from the optimal solution in which the consensus can be achieved in each round of communications.

Different from the previous works, which use statistical and randomized algorithms to achieve consensus, a MultiAgent Reinforcement Learning (MARL) scheme is employed in this paper to optimize the usage of the wireless channel for leader election, and the elected leader is allowed to make the consensus. A well-trained reinforcement learning model can have a better performance in choosing the approximate actions than the randomized algorithm, but the proposed algorithm has higher efficiency in achieving the consensus than that in Ref. [13]. Moreover, the proposed MARL algorithm is a distributed one deployed in each of the devices. An agent is used to indicate the MARL algorithm in each of the devices. Different from the previous MARL works, in most of which multiple agents either compete or cooperate with each other for the final reward, the multiple agents in the current work have the competition and cooperation relationship simultaneously for a successful leader election. To handle this complex case, each agent in our algorithm uses a matrix to depict the efficiency and fairness of the recent leader election and tunes its reward carefully according to its matrix and action in the last time step. The actor-critic framework from Refs. [17–19] is also adopted in the MARL algorithm, which enhances the efficiency and accuracy of the algorithm. The contribution of this paper is summarized in the following.

A lightweight consensus protocol is proposed in this paper for blockchain in IoT networks via reinforcement learning, demonstrating an almost optimal efficiency. Different from the famous PoW and Proof of Stake (PoS) schemes based on reliable communications, the introduced protocol has minimal requirements for computing and storage resources to achieve consensus. Additionally, an MARL scheme under the actor-critic framework is proposed to help devices choose the most approximate actions when achieving consensus. Numerical results from the simulation show that the efficiency of the consensus protocol nearly reaches the optimal solution.

The authors of this study believe that the closest effort to their work in recent years is Refs. [13–16], which are all committed to providing a lightweight consensus algorithm suitable for decentralized IoT application scenarios. Compared with previous work, the current study has a considerable advantage in channel utilization efficiency when combined with reinforcement learning. Therefore, the authors believe that this work provides a novel perspective and option for the selection of consensus algorithms for the large-scale application of blockchain technology in IoT scenarios in the future.

Roadmap. The rest of this paper is organized as follows. Section 2 presents the related work. Section 3 introduces the specific definition of the network model and problem definition. Section 4 provides the algorithm description. Section 5 presents an extensive simulation to evaluate the performance of the proposed algorithm. Section 6 concludes this paper.

2 Related Work

As mentioned above, the development of consensus protocols in blockchain in recent years can be divided into two groups. The first one considers the satisfaction of the complex demands of reality based on reliable device-to-device communication assumptions. The aforementioned protocols usually have a high consumption of computing or storage resources and numerous communications to ensure the security of consensus and priority of devices. For example, the PoW in Refs. [8, 9], the Proof of Space in Ref. [20], the PoA in Ref. [10], and the Proof of Reputation in Ref. [21] are all based on reliable communication assumptions and have high requirements for computing/storage/bandwidth resources. This group of consensus protocols is unsuitable for IoT networks because the IoT devices are usually lightweight, battery-operated, and deployed in an unreliable wireless environment. The second group of consensus protocols, including Refs. [13–16], are specifically designed for consensus under an unreliable communication environment. Specifically, the distributed protocols are proposed in Refs. [13, 14] for devices to achieve the consensus in single-hop wireless networks with $O(\log n)$ time complexity. The core idea of the above works is a randomized leader election algorithm, which ensures fairness by allowing nodes to compete for the leader with the same transmission probability. However, the randomization in the leader election algorithm also reduces the efficiency of the consensus. Other approaches to consensus for blockchain in multihop wireless networks are given in Refs. [15, 16]. The algorithms in Refs. [15, 16] first build some special tree structures based on the physical interference model. The consensus can then be made within $O(\log n)$ and $O(n \log^* n)$ time steps. However, the time used for the tree construction is non-negligible. Similar to the settings of Refs. [13, 14], the current work considers the consensus for blockchain in a single-hop wireless network with unreliable communications. The largest difference is that the MARL method is used in the proposed algorithm to help nodes choose the most appropriate rather than the randomized leader algorithms in Refs. [13, 14]. Thus, the performance of the algorithm can be optimized.

The related literature regarding the reinforcement learning techniques used in this study is provided in the following. The well-known early works regarding MARL include independent Q-learning in Ref. [22], value decomposition networks in Ref. [23], and QMIX in Ref. [24]. The studies on the MARL later extended into several directions according to their different training models and relationship settings. A comprehensive survey for MARL in recent years can be found in Ref. [25].

According to their training models, the studies on MARL can be divided into two categories: the centralized one, which relies on a centralized server to merge the learning results (e.g., the COunterfactual Multi-Agent (COMA) policy gradients in Ref. [26] and Multi-Agent Deep Deterministic Policy Gradient (MADDPG) in Ref. [27]); and the distributed one, in which the single-agent reinforcement learning scheme is deployed on each device[28, 29]. According to the relationship between multiple agents in reinforcement learning, the works of MARL can also be divided into three groups: the fully cooperative MARL[30, 31], the fully competitive MARL[32, 33], and the mixed cooperative and competitive MARL[34–36]. Specifically, in the fully cooperative learning mode, the interests, rewards, and goals of agents are the same, which is friendly for designing efficient MARL algorithms. The cooperative MARL is also termed team Markov games in Refs. [37–39]. For the smart agents in the fully competitive learning mode, the gain of one party is the loss of another party. Thus, the key issue lies in reaching the balance between agents and maximizing the final global reward. Some typical works include Ref. [40]. In the mixed setting of cooperation and competition[41–43], the agent is divided into multiple groups. The agents in the same group have the same reward and will cooperate. A competitive relationship exists between agents in different groups because the rewards of two different groups are negatively correlated.

The centralized training model is generally more accurate than the previous one due to the availability of global information. Meanwhile, the distributed training model is suitable for IoT networks. Compared with the fully cooperative and fully competitive MARL, the mixed setting of cooperation and competition is complex but has a high potential to solve some difficult problems. The proposed MARL in this paper is a distributed one with a mixed setting to optimize the consensus for blockchain in IoT networks. Additionally, the mixture of competition and cooperation among agents in this work is more complex than the previous ones because each pair of nodes has a competition and cooperation relationship with each other, which also complicates the design of the proposed MARL algorithm.

3 Model and Problem Definition

3.1 Network and communication model

A single-hop network model, in which n nodes are arbitrarily deployed in a two-dimensional Euclidean space, is considered. All nodes initially wake up with a synchronized clock, and their communications are the roundly based half-duplex mode. Specifically, the time in the network is split into synchronization rounds, and each split time is sufficient for nodes to transmit or receive a signal. At the start of every round, every node in the network will opt to send messages or listen to the channel to receive a signal. However, these nodes cannot perform both because their transceivers are half-duplex. For a wireless signal from its transmitter u to its receiver v[†], the Signal to Interference plus Noise Ratio (SINR) model in this work is a typical physical interference model used to describe simultaneous message reception, interference, and competition among nodes. Graph models to describe network communication may ignore some physical characteristics in the process of information dissemination. Using a graphical model can simplify the communication model and facilitate the discussion and analysis of the problem. However, an information transfer model that is close to the real world will increase the value of the current work for practical applications. Compared with the graph-based models, the SINR model in this paper is realistic in describing global interference and signal reception in the physical world, which has been extensively considered in Refs. [13–16]. The SINR model is comprehensively described in the following:

$$\text{Signal}(v) = \sum_{i \in I} P_i \cdot d(i, v)^{-\eta} + N,$$

$$\text{SINR}(u, v) = \frac{P_u \cdot d(u, v)^{-\eta}}{\sum_{x \in I \backslash \{u\}} P_x \cdot d(x, v)^{-\eta} + N}.$$

In the above SINR equations, the strength of the signal received by v is marked as Signal(v). SINR(u, v) is the SINR of the signal to receiver v from transmitter u. P_x is the transmission power of node x. $d(x, v)$ is the Euclidean distance between nodes x and v. i is an agent and I is defined as the set of transmitters in the current round. $\eta \in (2, 6]$ is the parameter for path loss. N is the ambient noise. η and N are determined by the environment.

When SINR$(u, v) \geqslant \epsilon$, the receiver v can decode the signal from u. ϵ is a threshold determined by hardware and is typically larger than 1. Compared with the graph-based interference model, the SINR model is considerably realistic and accurate in describing the global accumulated interference and the reception of signals.

Every node is assumed to have a uniform transmission power P and transmission range R to formulate the single-hop wireless environment. All nodes are within the transmission range of each other and contain their transmission power $P \geqslant N\epsilon R^{\eta}$.

3.2 Problem definition

The transactions between nodes in a blockchain system occur due to some real-life activities. The nodes must periodically achieve a consensus to ensure the security and privacy of the transactions in a distributed framework. Specifically, each period will have a block containing all the transactions that occurred. All nodes should then reach a consensus on this block; that is, add this block on the blockchain if all the transactions in this block are valid or discard this block otherwise. Thus, the characteristics of the consensus in a blockchain directly determine the efficiency and security of the services based on the blockchain system. Similar to the characteristics of the classical consensus problem defined in Ref. [44], the consensus in blockchain is defined with the following characteristics to ensure that all nodes can quickly reach an agreement for the transactions that occur in the blockchain system:

(1) Termination. Each node makes the decision in finite time steps.

(2) Validity. A block will be added to the blockchain by all nodes if and only if all its transactions are valid.

[†] Who transmits/receives a signal is termed as the transmitter/receiver for short.

(3) Agreement. All nodes will have the same decision on whether to accept or abandon the new block.

Knowledge and capability of the nodes: Different from the previous works, the consensus process in blockchain is optimized in this study with an MARL algorithm. Thus, each node is assumed to have a unique ID, a lightweight computing and storage unit, and a half-duplex transceiver for transmitting and receiving messages. Additionally, the physical carrier sensing function is required in the transceiver, which helps the node determine the usage of the channel.

4 Algorithm Description

4.1 Framework of the consensus protocol

The four-stage consensus framework has been widely utilized in previous work to reach a consensus within the blockchain system. The current study also considers the problem under such a framework. The following is an elaborate description of the four-stage consensus framework.

(1) Leader Election (LE) stage. In this stage, a leader is elected from all the nodes within the blockchain system.

(2) Block Proposal (BP) stage. The elected leader proposes a block containing the transaction records and relevant information regarding the previous block within the network. The newly generated block is then propagated to all the other nodes within the blockchain system.

(3) Block Validation (BV) stage. Once the block is received from the leader, other nodes will validate the content of the block and return their confirmation (i.e., agree or disagree with the proposed block) to the leader.

(4) Chain Updating (CU) stage. All nodes, including the leader, reach the same agreement on whether the proposed block should be added to the main chain.

The proposed algorithm is also based on the framework of the four-stage consensus. Specifically, the designed reinforcement learning algorithm ensures that in the LE stage, only one node is definitely elected as the leader and others know the elected result. A detailed description is provided in the next subsection. In the BP stage, the leader proposes a block, and all other nodes listen to receive the block. Only one node broadcast is available. Thus, the SINR equation indicates that the block from the leader can be received by all listening nodes. In the BV stage, the leader listens, and the other nodes will transmit with power P if it does not agree on the proposed block. The leader can sense a busy channel[‡] with its physical carrier sensing function when some nodes disagree on the proposed block due to the accumulative feature of wireless signals. Finally, the leader broadcasts the state of the sensed channel in the BV stage, and all nodes make the final decision according to the state of the sensed channel in the BV stage. Thus, if the sensed channel is busy in the BV stage, then all nodes in the CU stage will discard this block. Otherwise, this block will be added to the main chain.

Notably, the famous PoW in Ref. [8], namely the wChain in Ref. [14] and the BLOWN in Ref. [16], are all based on such a scheme. The difference among them is as follows: in PoW, the leader is elected by solving a puzzle; in wChain, the leader is the root node of its tree structure; in BLOWN, the leader is elected with a statistically randomized algorithm. The difference between the proposed algorithm and the aforementioned is that the leader in the introduced algorithm is elected by a well-designed reinforcement learning scheme, which has nearly optimal efficiency. The following subsection provides a detailed description of the MARL LE algorithm.

4.2 Multiagent reinforcement learning leader election algorithm

The proposed algorithm is introduced in the following three parts. First, the definitions and descriptions of the necessary elements required by the algorithm are introduced. Second, the above definitions and descriptions are used to introduce the detailed derivation process of the proposed algorithm. Last, the overall description of the algorithm and the overall structure of the neural network are provided.

4.2.1 Definitions and descriptions of necessary elements of the algorithm

When designing a reinforcement learning scheme, the following four parts should generally be carefully considered: the state space $\mathcal{S}$, the action space $\mathcal{A}$, the reward $\mathcal{R}$, and the transition probability $\mathcal{P}_{ss'}$ between the current state and the next state. In the MARL algorithm, n agents are deployed on each of the nodes. In each LE stage t, an agent i takes an action a_t^i in the current state s_t^i, immediately receives the reward r^i and

[‡] A channel is busy when some nodes are transmitting; otherwise, we say the channel is idle.

comes to the new state s^i_{t+1}, and iteratively updates its transition probability $\mathcal{P}_{ss'}$ to expect a high reward return U^i_t (i.e., the cumulative reward). Specifically, in the problem, each agent i takes the channel utilization index as its state s^i_t. The channel utilization index defined below can fully and effectively measure the fairness of channel utilization of each agent. Thus, taking the channel utilization index as the state of an agent can enable the agent to obtain additional completed channel information and choose additional appropriate actions. Each agent takes an action a^i_t according to the state of this round; that is, an agent chooses to transmit messages or listen to the channel in the current round t. Thus, the new state of the channel will be determined when all agents complete their actions synchronously. During this time, all agents have knowledge of the following: whether the LE of this round is successful; if successful, which agent is elected as the leader. After knowing this information, each agent can update its new state s^i_{t+1} locally and obtain the corresponding reward and punishment r^i. Each agent can then adjust its policy according to the reward and punishment obtained. Thus, designing appropriate reward functions can help each agent gradually avoid all the wrong decisions and learn good decisions after iterative training steps to determine the optimal policy.

Terms other than state transition probability mentioned above will be defined step-by-step for the problem in the following sections. Most reinforcement learning scenarios preset the state transition to be random. However, in the current problem, the determination of the new state is jointly determined by all agents, which is no longer random (as mentioned above, the determination of the new state of the channel is jointly determined by all agents after the synchronized actions), thereby simplifying the problem.

Leader election: Each LE requires the main and auxiliary rounds. In the main round, each agent chooses to send a message according to the probability learned by itself (this type of agent is named B1) or listen to the channel (this type of agent is named B2). If each agent B2 has successfully decoded information in the first round, then this agent will remain silent in the auxiliary round to inform the success of agent B1 that chose to send a message to the LE in the main round. Each agent B1 listens in the auxiliary round; if the sensed channel is idle, then the agent has been successfully elected as the leader. In this case, the main round is defined as a successful round t_s. Otherwise, in the auxiliary round, if agent B2 cannot successfully decode the message in the main time slot due to the transmission of multiple nodes or the silence of all nodes, then the agent will choose to send a message in the auxiliary round. In this case, agent B1 senses a busy channel, and the main round is defined as a failure for the LE. The auxiliary round is used for nodes to verify the usage of the channel in the main round. The following mainly discusses the situation in the main round.

States: In each round t, the channel utilization index $s^i(t_s)$ of each agent i is defined as the state $s^i(t)$ of each agent, that is, $s^i(t) = s^i(t_s)$. The channel utilization index $s^i(t_s)$ describes how many successful rounds agent i has experienced since its successful election as the leader last time, specified in the following:

$$s^i(t_s{+}1)=\begin{cases} s^i(t_s)+1, & \text{if } i \text{ is not elected as the leader;}\\ 0, & \text{if } i \text{ is elected as the leader}\end{cases} \tag{1}$$

Notably, this index is updated only when an agent is elected as the leader in this round (successful round), where i represents the ID of the node and t_s represents the successful rounds of electing leaders during the training process. The index update is shown in the above formula. For example, if agent i is elected as the leader in this successful round, then its index will be reset to zero while that of other agents is increased by one. This formula indicates that the state transition is deterministic, which simplifies the complexity of the learning environment and facilitates the easy training of the neural network.

The average index is derived from above: $\bar{s}(t_s) = \sum_{i=1}^{n} s^i(t_s)/n$. For each agent i, a global index table is maintained locally to calculate $\bar{s}(t_s)$. For example, Fig. 1 shows the situation where only node 3 is elected as the leader.

Actions: In each round t, each agent i has an independent probability of p_i to transmit a message or an independent probability of $1-p_i$ to listen. The action of an agent i in round t is denoted as $A^i(t)$,

ID	**1**	**2**	**3**	**4**	**…**	***n***
Index	6	4	8	3	…	3

Index table in round t

Node 3 is elected as the leader

ID	**1**	**2**	**3**	**4**	**…**	***n***
Index	7	5	0	4	…	4

Index table in round t+1

Fig. 1 Example of how the index changes.

$$A^i(t)=\begin{cases}0, & \text{if the node does not send a message;}\\ 1, & \text{if the node sends a message.}\end{cases}$$

Reward: Immediate reward is the value that the environment feeds back to the agent. The reward of agent i and the observed value of the reward are recorded as variables R^i and r^i, respectively. The discount return of agent i is the weighted sum of its reward,

$$U_t^i = R_t^i + \gamma \cdot R_{t+1}^i + \gamma^2 \cdot R_{t+2}^i + \gamma^3 \cdot R_{t+3}^i + \cdots$$

where $\gamma \in [0,1]$ is the discount factor. From a mathematical viewpoint, an infinite sum of rewards may not converge to a finite value and is difficult to address in equations. However, the infinite sum will converge with a discount factor and under reasonable conditions.

Reward shaping: In the setting of multiagent cooperation, the reward design is generally $R^1 = R^2 = \cdots = R^n$. In the case of multiagent competition, the reward design is $R^1 \propto -R^2$. However, the problem lies in a mixed task with cooperation and competition: all agents must not only cooperate in electing a leader but must also compete for the leadership of each round. Specifically, after all agents complete their action in each round, only two cases remain in the channel, which is denoted as $C(t)$,

$$C(t)=\begin{cases}1, & \text{if only one node transmits in main round;}\\ 0, & \text{otherwise}\end{cases} \tag{2}$$

If an agent's $A(t) = 1$ at time t, its index $s^i(t_s)$ is abbreviated as s_a, and each agent can learn s_a from its updated index table. Thus, the reward is designed as follows:

$$r^i(t)=\begin{cases}1, \ C(t)=1 \text{ and } s_a \geqslant \bar{s}(t_s);\\ -1, \ C(t)=1, \ s^i(t_s)<\bar{s}(t_s), \text{ and } A^i(t)=1;\\ -1, \ C(t)=1, \ s^i(t_s)\geqslant\bar{s}(t_s), \text{ and } A^i(t)=0;\\ 0, \text{ otherwise}\end{cases} \tag{3}$$

As shown in Eq. (3), for each agent i in the main round t with $C(t) = 1$, which indicates that t is a successful round for LE. At this time, if the index of the elected leader is no smaller than the average index, that is, $s_a \geqslant \bar{s}(t_s)$, then this index is a good result because all agents cooperate to elect a leader and consider the fairness of the index. Therefore, all agents will be rewarded with 1. However, if the index of the elected leader is less than the average index, that is, $s_a < \bar{s}(t_s)$, then the leader in the current round is believed to be elected too frequently (i.e., ($s^i(t_s) < \bar{s}(t_s)$ and $A^i(t) = 1$)). Therefore, this leader is given a punishment by setting the reward as -1. Additionally, the agents who should have competed for the leader but did not, that is, ($s^i(t_s) \geqslant \bar{s}(t_s)$ and $A^i(t) = 0$), are punished. The two aforementioned situations are considered the embodiment of the essence of the problem as a mixed task type of competition and cooperation. In other general situations, the agents are neither good nor bad; thus, they are neither rewarded nor punished.

4.2.2 Derivation process of the algorithm

The **policy function** must also be defined to construct the mapping of ($\pi : \mathcal{S} \to \mathcal{A}$) to achieve the maximum return,

$$\pi(a \mid s) \triangleq M(A = a \mid S = s).$$

The input of the policy function π is stated as s, and the output is a probability value of action A between 0 and 1. Whenever a state s is observed, the policy function is used to calculate the probability of each action. Specially, $M(A = a \mid S = s)$ is the probability that the agent takes an action a when it is in the state s. Random sampling is then given to obtain action A, and the agent will execute action A.

The most effective method to obtain such a policy function is to use neural network $\pi(a \mid s; \boldsymbol{\theta})$ to approximate the policy function $\pi(a \mid s)$. The neural network $\pi(a \mid s; \boldsymbol{\theta})$ is called a **policy network**. $\boldsymbol{\theta}$ represents the parameters of the neural network and is randomly initialized at the beginning. The collected states, actions, and rewards are then used to update $\boldsymbol{\theta}$. A policy network means approximating policy function with the neural network. Each agent can have its policy network. The current problem is discrete control, and the policy network of each agent i is denoted as

$$\hat{\boldsymbol{f}} = \pi(\cdot \mid s; \boldsymbol{\theta}_i).$$

The input of the policy network is state s, and the output is a vector $\hat{\boldsymbol{f}}$. The dimension of vector $\hat{\boldsymbol{f}}$ is the size of the action space $|\mathcal{A}^i|$. Each element of $\hat{\boldsymbol{f}}$ represents the probability of action. The elements of $\hat{\boldsymbol{f}}$ are all positive real numbers and add up to 1. Random sampling according to $\hat{\boldsymbol{f}}$ is performed during decision making to obtain action a^i, and agent i executes this action. For a typical Reinforcement Learning (RL) problem, the following two indicators are also necessary to solve the RL problem. The first indicator is the **action value function (Q-value function)**, which represents the cumulative expected reward R_t after taking action a_t from state s_t and following the policy π. The action value function of the agent is recorded as

$$Q_\pi\left(s_t, a_t\right)=E\left[R_t \mid s_t, a_t\right]=E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid s_t, a_t\right].$$

The second indicator is the **state value function**, denoted as $V_\pi(S)$, which is used repeatedly in policy learning methods. This indicator is the expectation of the action value function $Q_\pi(S, A)$ regarding the current policy π. The state value function of the agent is as follows:

$$V_\pi\left(s_t\right)=E_{A_t \sim \pi(\cdot \mid s_t)}\left[Q_\pi\left(s_t, A_t\right)\right]= \sum_{a \in \mathcal{A}} \pi\left(a \mid s_t\right) \cdot Q_\pi\left(s_t, a\right) \tag{4}$$

If a policy is good, then the mean value of $V_\pi(S)$ should be large for all states S. Therefore, the objective function is defined as

$$J(\boldsymbol{\theta})=E_S\left[V_\pi(S)\right].$$

This objective function excludes the factor of state S and only relies on the parameter $\boldsymbol{\theta}$ of the policy network π. A superior policy indicates a large $J(\boldsymbol{\theta})$. Updating the parameter $\boldsymbol{\theta}$ of the policy network aims to gradually increase the objective function $J(\boldsymbol{\theta})$, thus strengthening the policy network. This gradient of the objective function is called the policy gradient, which can be written in the form of the following theorem. The subsequent derivation of the algorithm will be based on this theorem. The policy gradient theorem was independently proposed by Marbach and Tsitsiklis in 1999[45] and Sutton et al. in 2000[18],

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}= E_S\left[E_{A \sim \pi(\cdot \mid S ; \boldsymbol{\theta})}\left[\frac{\partial \ln \pi(A \mid S ; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(S, A)\right]\right] \tag{5}$$

Policy learning can then be expressed as an optimization problem: $\max_{\boldsymbol{\theta}}\left\{J(\boldsymbol{\theta}) \triangleq E_S\left[V_\pi(S)\right]\right\}$. The simplest algorithm to solve this maximization problem is gradient ascent: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}+\beta \cdot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, where β is the learning rate, which must be manually adjusted, and $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is the policy gradient. According to the policy gradient in Eq. ((5)),

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})= E_S\left[E_{A \sim \pi(| S ; \boldsymbol{\theta})}\left[Q_\pi(S, A) \cdot \nabla_{\boldsymbol{\theta}} \ln \pi(A \mid S ; \boldsymbol{\theta})\right]\right] \tag{6}$$

Analytically determining this expectation is impossible because the probability density function of state S is unknown. However, the expectation in the policy gradient can be approximated through the Monte Carlo approximation of expectation[46]. Monte Carlo is an umbrella term for a broad class of randomized algorithms that use random samples to estimate true values. One state s is observed from the environment at a time corresponding to the observed value of the random variable S. According to the current policy network (policy network parameters must be the latest), an action is then randomly sampled: $a \sim \pi(\cdot \mid s ; \boldsymbol{\theta})$, and the stochastic gradient is calculated as follows:

$$\boldsymbol{g}(s, a ; \boldsymbol{\theta}) \triangleq Q_\pi(s, a) \cdot \nabla_{\boldsymbol{\theta}} \ln \pi(a \mid s ; \boldsymbol{\theta}) \tag{7}$$

$\boldsymbol{g}(s, a ; \boldsymbol{\theta})$ is the unbiased estimate of policy gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})=E_S\left[E_{A \sim \pi(\cdot \mid S ; \boldsymbol{\theta})}[\boldsymbol{g}(S, A ; \boldsymbol{\theta})]\right].$$

Applying the above conclusions, stochastic gradient ascent can be performed to update $\boldsymbol{\theta}$. Therefore, the objective function $J(\boldsymbol{\theta})$ grows step by step: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}+\beta \cdot \boldsymbol{g}(s, a ; \boldsymbol{\theta})$. However, this method still fails, and $g(s, a ; \boldsymbol{\theta})$ cannot be calculated because the action value function $Q_\pi(s, a)$ is unknown. In RL literature, $Q_\pi(s, a)$ is approximated by two methods: one approach is REINFORCE proposed by Williams in 1992[47], which approximates $Q_\pi(s, a)$ with the return u of the actual observation; the other approach is actor-critic, which uses a neural network $q(s, a ; \boldsymbol{\omega})$ to approximate $Q_\pi(s, a)$, where $\boldsymbol{\omega}$ represents the parameters of the neural network and is randomly initialized at the beginning. The collected states, actions, and rewards are then used to update $\boldsymbol{\omega}$. The second solution is adopted in the current study. Next, the actor-critic method is comprehensively introduced. The policy network $\pi(a \mid s ; \boldsymbol{\theta})$ is equivalent to an actor, which makes action a based on state s. Value network $q(s, a ; \boldsymbol{\omega})$ is equivalent to the critic, who scores the performance of the actor and quantifies how good or bad action a is in state s. The relationship between the policy network (actor) and the value network (critic) is shown below.

Instead of reward R, the objective function of policy learning $J(\boldsymbol{\theta})$ lies in the expectation of return U. The current reward R is still meaningless to the policy network. Training the policy network (actor) requires the return U, which is the weighted sum of all the rewards in the future. The value network (critic) can estimate the expectation of return U and thus help train the policy network (actor).

4.2.3 Specific description of the algorithm and the overall network structure

Training of the policy network (actor): The policy network (actor) wants to improve his acting, but the actor does not know what kind of performance is effective;

thus, he needs the help of the value network (critic). After the actor makes an action a, the critic will give a score $\hat{q} \triangleq q(s, a; \boldsymbol{\omega})$ and provide the score as feedback to the actor to help the actor make improvements. Using the current state s and the critic's score $\hat{q}$, the actor calculates the approximate policy gradient and then updates his parameter $\boldsymbol{\theta}$. Thus, the performance of the actor is praised by the critic and the score $\hat{q}$ increases. The basic idea of training the policy network is to update the parameter $\boldsymbol{\theta}$ with the approximation of policy gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. The unbiased estimator of the policy gradient was previously derived in Eq. ((7)). Value network $q(s, a; \boldsymbol{\omega})$ is the approximation of the action value function $Q_\pi(s, a)$. Therefore, Q_π in Eq. ((7)) above is replaced by the value network to obtain the approximate policy gradient,

$$\hat{\boldsymbol{g}}(s, a; \boldsymbol{\theta}) \triangleq \underbrace{q(s, a; \boldsymbol{\omega})}_{\text{critic's rating}} \cdot \nabla_{\boldsymbol{\theta}} \ln \pi(a \mid s; \boldsymbol{\theta}) \qquad (8)$$

Finally, the parameters of the policy network are updated by gradient ascent: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \cdot \hat{\boldsymbol{g}}(s, a; \boldsymbol{\theta})$. Training of the value network (critic): The above analysis reveals that the above method of training policy network does not really contribute to the improvement of actors but only increases their suitability for the critic. Therefore, the level of the critic is also crucial, and the level of the actors can be truly improved only when the critic's score $\hat{q}$ truly reflects the value of the action Q_π. Initially, the value network parameter $\boldsymbol{\omega}$ is random. The State-Action-Reward-State'-Action' (SARSA) algorithm originated in Ref. [48] can be used to update $\boldsymbol{\omega}$ and improve the level of the critic. Each time a reward r is observed in the environment, r is regarded as truth and used to calibrate the ratings of the critic.

In the SARSA algorithm, the value network at time t outputs:

$$\hat{q}_t = q\left(s_t, a_t; \boldsymbol{\omega}\right),$$

which is an estimate of the action value function. At time are actually observed; therefore, the Temporal-Difference (TD) target can be calculated as follows:

$$\hat{y}_t \triangleq r_t + \gamma \cdot q\left(s_{t+1}, a_{t+1}; \boldsymbol{\omega}\right),$$

which is also an estimate of the action value function $Q_\pi\left(s_t, a_t\right)$. $\hat{y}_t$ is partly based on the actually observed reward r_t; thus, $\hat{y}_t$ is believed to be closer to the truth than $q\left(s_t, a_t; \boldsymbol{\omega}\right)$. Therefore, $\hat{y}_t$ is fixed, and $q\left(s_t, a_t; \boldsymbol{\omega}\right)$ is encouraged to approach $\hat{y}_t$. The SARSA algorithm specifically updates the value network parameter $\boldsymbol{\omega}$ in this way. The loss function is defined as follows:

$$L(\boldsymbol{\omega}) \triangleq \frac{1}{2}\left[q\left(s_t, a_t; \boldsymbol{\omega}\right) - \hat{y}_t\right]^2.$$

Let $\hat{q}_t \triangleq q\left(s_t, a_t; \boldsymbol{\omega}\right)$, the gradient of the loss function is

$$\nabla_{\boldsymbol{\omega}} L(\boldsymbol{\omega}) = \underbrace{\left(\hat{q}_t - \hat{y}_t\right)}_{\text{TD error } \delta_t} \cdot \nabla_{\boldsymbol{\omega}} q\left(s_t, a_t; \boldsymbol{\omega}\right).$$

$\boldsymbol{\omega}$ is updated with gradient descent,

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \alpha \cdot \nabla_{\boldsymbol{\omega}} L(\boldsymbol{\omega}).$$

Updating $\boldsymbol{\omega}$ in this way makes $q\left(s_t, a_t; \boldsymbol{\omega}\right)$ close to $\hat{y}_t$. SARSA can be understood this way: the observed reward r_t is used to calibrate the critic's score $q\left(s_t, a_t; \boldsymbol{\omega}\right)$. Finally, the flow chart and the pseudocode of the algorithm are given in Fig. 2 and Algorithm 1, respectively.

5 Simulation Result

The experimental results from the simulation are presented in this section to evaluate the efficiency of the proposed algorithm in achieving the consensus. As mentioned in the framework of the consensus protocol, the LE process directly impacts the efficiency of consensus. Once a leader is elected, completing the BP, BC, and CU stages takes three additional rounds. Thus, in the following, the efficiency of the LE algorithm is observed in each episode, which contains 1000 main and auxiliary rounds.

Parameter setting. This simulation has n agents randomly and uniformly distributed in a circular area with a radius of $R = 100$ m, and the minimum distance between agents is 1 m. Ambient noise N is normalized to 1 dB, the transmission range of the equipment is set to 200 m, and the maximum transmission power $P = 2R^\eta N\epsilon$. If unspecified, then all agent parameters are set as follows: policy network learning rate is set to 0.0003, value network learning rate to 0.0005, $\gamma = 0.9$, and $T_{\max} = 1000$. All results are provided on TensorFlow 1.8 and Python 3.6 platforms. All simulation results are generated on a computer configured with Intel Core i7-8565U@1.80 GHz.

Simulated results. First, as shown in Fig. 3, the convergence of the Deep Reinforcement Learning (DRL) algorithm with different numbers of agents is evaluated using the variation of the total reward obtained from a single agent in each episode during the training process. The results reveal that the agent can always gradually obtain the maximum total reward and converge after sufficient episodes. With the increase in n, raising

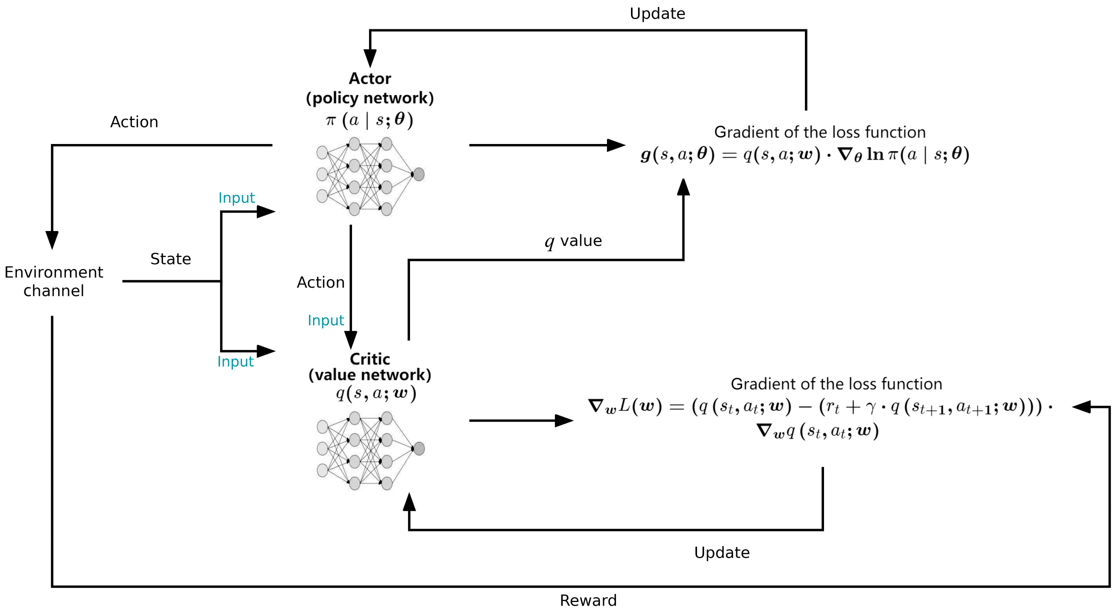

Update
Actor
(policy network)
$\pi\,(a \mid s; \boldsymbol{\theta})$
Action
Gradient of the loss function
$\boldsymbol{g}(s, a; \boldsymbol{\theta}) = q(s, a; \boldsymbol{w}) \cdot \nabla_{\boldsymbol{\theta}} \ln \pi(a \mid s; \boldsymbol{\theta})$
Input
q value
State
Environment channel
Action
Input
Input
Critic
(value network)
$q(s, a; \boldsymbol{w})$
Gradient of the loss function
$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}) = (q\,(s_t, a_t; \boldsymbol{w}) - (r_t + \gamma \cdot q\,(s_{t+1}, a_{t+1}; \boldsymbol{w}))) \cdot \nabla_{\boldsymbol{w}} q\,(s_t, a_t; \boldsymbol{w})$
Update
Reward

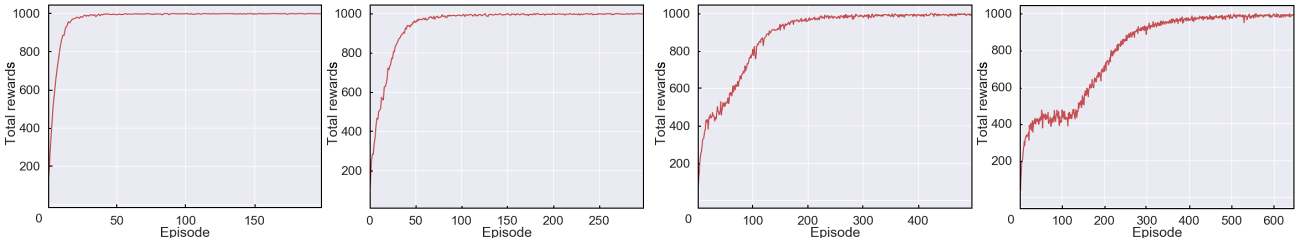
Fig. 2 Actor-critic framework of our MAAC algorithm.

Algorithm 1 MAAC algorithm

```
for $i = 1, 2, \ldots, n$ do
    Initialize parameters $\boldsymbol{\theta}^i$ in the policy network $\pi^i\left(\cdot \mid s^i; \boldsymbol{\theta}^i\right)$ and parameters $\boldsymbol{w}^i$ in the value network $\hat{q}^i\left(s^i, \tilde{a}^i; \boldsymbol{w}^i\right)$.
    Initialize state (index) $s^i(t) = s^i(t_s) = 0$ and the local index table.
end for
for episode $= 1, 2, \ldots, E_{\max}$ do:
    for $t = 1, 2, \ldots, T_{\max}$ do:
        for each agent $i = 1, 2, \ldots, n$ do:
            The policy network makes a decision based on the channel state $s^i_t$ at the current time $t$: $a^i_t \sim \pi^i(\cdot \mid s^i_t; \boldsymbol{\theta}^i)$. Specifically, let the agent sample the action $a^i_t$ based on the probability distribution of the current policy network output and execute this action.
            Obtain a new state $s^i_{t+1}$ (1) and a new reward $r^i_t$ Eq. (3) from the current channel state. Make decisions based on the policy network: $\hat{a}^i_{t+1} \sim \pi^i(\cdot \mid s^i_{t+1}; \boldsymbol{\theta}^i)$, but do not let the agent perform the action $\tilde{a^i}_{t+1}$.
            Let the value network score the policy network: $\hat{q}^i_t = q^i(s^i_t, a^i_t; \boldsymbol{\omega}^i)$ and $\hat{q}^i_{t+1} = q^i(s^i_{t+1}, \tilde{a}^i_{t+1}; \boldsymbol{\omega}^i)$.
            Calculate TD target and TD error: $\hat{y}^i_t = r^i_t + \gamma \cdot \hat{q}^i_{t+1}$ and $\delta^i_t = \hat{q}^i_t - \hat{y}^i_t$.
            Update value network: $\boldsymbol{\omega}^i \leftarrow \boldsymbol{\omega}^i - \alpha \cdot \delta^i_t \cdot \nabla_{\boldsymbol{\omega}^i} q^i(s^i_t, a^i_t; \boldsymbol{\omega}^i)$.
            Update policy network: $\boldsymbol{\theta}^i \leftarrow \boldsymbol{\theta}^i + \beta \cdot \hat{q}^i_t \cdot \nabla_{\boldsymbol{\theta}^i} \ln \pi^i(a^i_t \mid s^i_t; \boldsymbol{\theta}^i)$.
        end for
    end for
end for
```

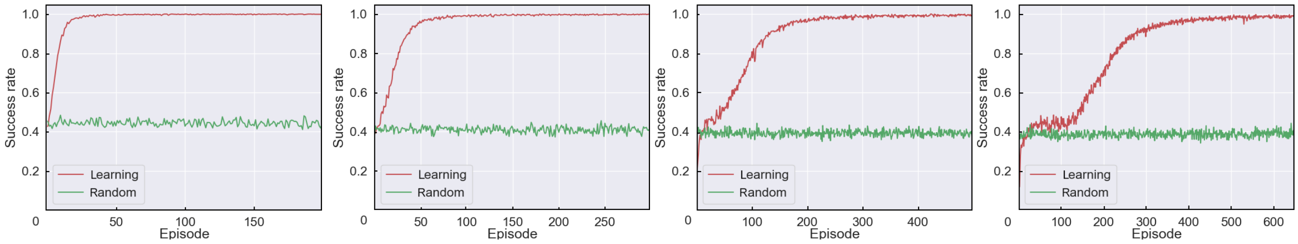
episodes (i.e., training time) facilitates the convergence of total reward to the maximum value. This advantage facilitates the easy deployment of the proposed algorithm in large-scale distributed network scenarios without degrading performance. Figure 3 demonstrates that the proposed algorithm converges well, which directly proves the "termination" property. Combined with the four aforementioned stages of consensus that guarantee "validity" and "agreement", the proposed algorithm fits well with the three properties of consensus algorithms.

Second, as shown in Fig. 4, the success rate (the ratio of successful rounds to the total rounds) of the introduced learning algorithm in each episode and that of the previous randomized algorithm are compared[13]. The results show that despite the randomized algorithm with a stable success rate, the proposed learning algorithm can outperform the stochastic algorithm after sufficient training. Notably, the success rate after the algorithm has sufficiently converged can reach almost 100%, which is approximately 2.5 times larger than that of the randomized algorithm. Moreover, the success rate of the randomized algorithm decreases when n increases. The success rate has been proven to decrease to the lower bound of 1/e in Ref. [13] when n is infinite. By contrast, the performance of the proposed algorithm is insensitive to the increase in n. When n increases, only the corresponding episode must be increased to reach a 100% success rate. Therefore, the efficiency of the



(a) $n = 3$ (b) $n = 5$ (c) $n = 8$ (d) $n = 10$

Fig. 3 Total reward obtained from a single agent in each episode.

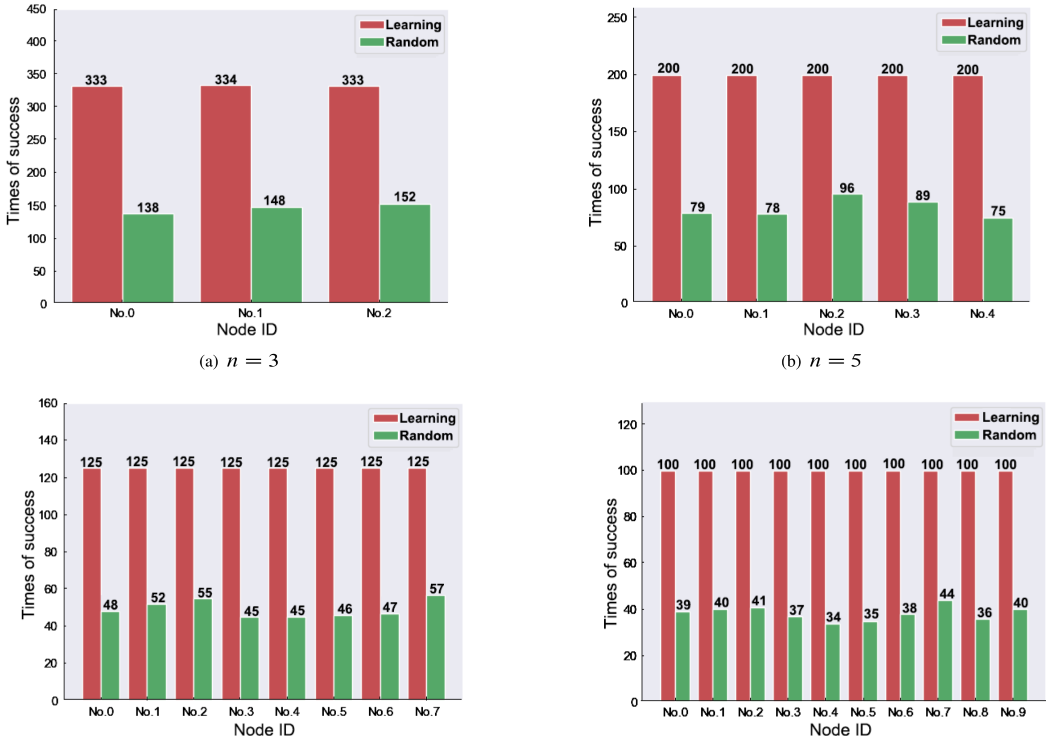


(a) $n = 3$ (b) $n = 5$ (c) $n = 8$ (d) $n = 10$

Fig. 4 Comparison of the success rate in each episode between the learning and randomized algorithms.

proposed algorithm on consensus is optimized.

Finally, as shown in Fig. 5, the number of times that the agents are elected as the leader when the proposed learning algorithm reaches the maximum reward is compared with the counts of the randomized algorithm in Ref. [13]. The results show that the randomized algorithm only guarantees the fairness of the probability. However, a considerable amount of uncertainty will always remain in the actual situation due to the randomization, which is illustrated by the slightly equal blue bars in the figure. In the case of convergence, the proposed algorithm can finally ensure that the number of times each agent is elected as the leader is exactly the same, as shown by the absolute



(a) $n = 3$ (b) $n = 5$

(c) $n = 8$ (d) $n = 10$

Fig. 5 Comparison of the number of times that each agent is elected between the learning algorithm and the randomized algorithm.

equality of the red bars in the figure, thus ensuring actual fairness.

The experimental results generally show that when our algorithm is well trained to the final convergence, each agent will be elected as the leader in turn, i.e., each agent is determinately re-elected as a leader if it has elapsed an interval of $n-1$ successful rounds since it was last elected as leader. Thus, the efficiency of the proposed algorithm on consensus is optimal.

6 Conclusion

Different from the previous resource-consuming protocols for consensus in blockchain, a lightweight consensus protocol is designed in this paper for the blockchain system in a single-hop wireless network. A distributed MARL algorithm is designed to optimize the efficiency of the protocol in achieving the consensus. Numerical results from the simulation show that the proposed protocol can improve the efficiency of the consensus in blockchain by 250% compared with previous work, demonstrating optimal efficiency. Future work will extend to the multihop and multiple-channel scenarios.

Acknowledgment

This work was partially supported by the National Key Research and Development Program of China (No. 2020YFB1005900), the National Natural Science Foundation of China (Nos. 62102232, 62122042, and 61971269), and the Natural Science Foundation of Shandong Province (No. ZR2021QF064).

References

[1] W. P. Wang, Z. R. Wang, Z. F. Zhou, H. X. Deng, W. L. Zhao, C. Y. Wang, and Y. Z. Guo, Anomaly detection of industrial control systems based on transfer learning, *Tsinghua Science and Technology*, vol. 26, no. 6, pp. 821–832, 2021.

[2] Z. N. Mohammad, F. Farha, A. O. M. Abuassba, S. K. Yang, and F. Zhou, Access control and authorization in smart homes: A survey, *Tsinghua Science and Technology*, vol. 26, no. 6, pp. 906–917, 2021.

[3] X. L. Xu, H. Y. Li, W. J. Xu, Z. J. Liu, L. Yao, and F. Dai, Artificial intelligence for edge service optimization in Internet of Vehicles: A survey, *Tsinghua Science and Technology*, vol. 27, no. 2, pp. 270–287, 2022.

[4] M. S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli, and M. H. Rehmani, Applications of blockchains in the internet of things: A comprehensive survey, *IEEE Commun. Surv. Tutorials*, vol. 21, no. 2, pp. 1676–1717, 2019.

[5] K. Biswas and V. Muthukkumarasamy, Securing smart cities using blockchain technology, in *Proc. 18th Int. Conf. on High Performance Computing and Communications*; *IEEE 14th Int. Conf. on Smart City*; *IEEE 2nd Int. Conf. on Data Science and Systems*, Sydney, Australia, 2016, pp. 1392–1393.

[6] P. T. S. Liu, Medical record system using blockchain, big data and tokenization, in *Proc. 18th Int. Conf. on Information and Communications Security*, Singapore, 2016, pp. 254–261.

[7] X. Yue, H. J. Wang, D. W. Jin, M. Q. Li, and W. Jiang, Healthcare data gateways: Found healthcare intelligence on blockchain with novel privacy risk control, *J. Med. Syst.*, vol. 40, no. 10, p. 218, 2016.

[8] N. Satoshi, Bitcoin: A peer-to-peer electronic cash system, https://nakamotoinstitute.org/bitcoin/, 2008.

[9] B. Fisch, J. Bonnerau, N. Greco, and J. Benet, Scaling proof-of-replication for filecoin mining, Technical report, Stanford University, Palo Alto, CA, USA, https://research.protocol.ai/publications/scaling-proof-of-replication-for-filecoin-mining/, 2018.

[10] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, Proof of activity: Extending bitcoin's proof of work via proof of stake, *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 3, pp. 34–37, 2014.

[11] M. Castro and B. Liskov, Practical Byzantine fault tolerance, in *Proc. 3rd Symp. on Operating Systems Design and Implementation*, New Orleans, LA, USA, 1999, pp. 173–186.

[12] M. F. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, HotStuff: BFT consensus with linearity and responsiveness, in *Proc. 2019 ACM Symp. on Principles of Distributed Computing*, Toronto, Canada, 2019, pp. 347–356.

[13] Y. F. Zou, M. H. Xu, J. G. Yu, F. Zhao, and X. Z. Cheng, A fast consensus for permissioned wireless blockchains, *IEEE Internet Things J.*, doi: 10.1109/JIOT.2021.3124022.

[14] M. H. Xu, C. C. Liu, Y. F. Zou, F. Zhao, J. G. Yu, and X. Z. Cheng, wChain: A fast fault-tolerant blockchain protocol for multihop wireless networks, *IEEE Trans. Wirel. Commun.*, vol. 20, no. 10, pp. 6915–6926, 2021.

[15] L. Yang, Y. F. Zou, M. H. Xu, Y. C. Xu, D. X. Yu, and X. Z. Cheng, Distributed consensus for blockchains in internet-of-things networks, *Tsinghua Science and Technology*, vol. 27, no. 5, pp. 817–831, 2022.

[16] M. H. Xu, F. Zhao, Y. F. Zou, C. C. Liu, X. Z. Cheng, and F. Dressler, BLOWN: A blockchain protocol for single-hop wireless networks under adversarial SINR, *IEEE Trans. Mob. Comput.*, doi: 10.1109/TMC.2022.3162117.

[17] R. S. Sutton, Temporal credit assignment in reinforcement learning, PhD dissertation, Univ. Mass. Amherst, Amherst, MA, USA, 1984.

[18] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in *Proc. 12th Int. Conf. on Neural Information Processing Systems*, Denver, CO, USA, 2000,

pp. 1057–1063.

[19] A. G. Barto, R. S. Sutton, and C. W. Anderson, Neuronlike adaptive elements that can solve difficult learning control problems, *IEEE Trans. Syst. Man Cybern.*, vol. SMC-13, no. 5, pp. 834–846, 1983.

[20] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, Proofs of space, in *Proc. 35th Annu. Cryptology Conf.*, Santa Barbara, CA, USA, 2015, pp. 585–605.

[21] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, Permacoin: Repurposing bitcoin work for data preservation, in *Proc. 2014 IEEE Symp. on Security and Privacy*, Berkeley, CA, USA, 2014, pp. 475–490.

[22] M. Tan, Multi-agent reinforcement learning: Independent vs. cooperative agents, in *Machine Learning Proceedings 1993*. Amsterdam, the Netherlands: Elsevier, 1993, pp. 330–337.

[23] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, et al., Value-decomposition networks for cooperative multi-agent learning based on team reward, in *Proc. 17th Int. Conf. on Autonomous Agents and MultiAgent Systems*, Stockholm, Sweden, 2017, pp. 2085–2087.

[24] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. N. Foerster, and S. Whiteson, QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning, in *Proc. 35th Int. Conf. on Machine Learning*, Stockholmsmässan, Stockholm, Sweden, 2018, pp. 4292–4301.

[25] K. Q. Zhang, Z. R. Yang, and T. Başar, Multi-agent reinforcement learning: A selective overview of theories and algorithms, in *Handbook of Reinforcement Learning and Control*, K. G. Vamvoudakis, Y. Wan, F. L. Lewis, and D. Cansever, eds. Cham, Germany: Springer, 2021, pp. 321–384.

[26] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, Counterfactual multi-agent policy gradients, in *Proc. 32nd AAAI Conf. on Artificial Intelligence*, Palo Alto, CA, USA, 2018, pp. 2974–2982.

[27] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, Multi-agent actor-critic for mixed cooperative-competitive environments, in *Proc. 31st Int. Conf. on Neural Information Processing Systems*, Long Beach, CA, USA, 2017, pp. 6382–6393.

[28] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. Whiteson, Stabilising experience replay for deep multi-agent reinforcement learning, in *Proc. 34th Int. Conf. on Machine Learning*, Sydney, Australia, 2017, pp. 1146–1155.

[29] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, Multiagent cooperation and competition with deep reinforcement learning, *PLoS One*, vol. 12, no. 4, p. e0172395, 2017.

[30] A. Lazaridou, A. Peysakhovich, and M. Baroni, Multi-agent cooperation and the emergence of (natural) language, arXiv preprint arXiv: 1612.07182, 2017.

[31] I. Mordatch and P. Abbeel, Emergence of grounded compositional language in multi-agent populations, in *Proc. 32nd AAAI Conf. on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conf. and 8th AAAI Symp. on Educational Advances in Artificial Intelligence*, New Orleans, LA, USA, 2018, p. 183.

[32] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, Emergent complexity via multi-agent competition, present at Proc. 6th Int. Conf. on Learning Representations, Vancouver, Canada, 2018.

[33] M. Raghu, A. Irpan, J. Andreas, R. Kleinberg, Q. V. Le, and J. M. Kleinberg, Can deep reinforcement learning solve Erdos-Selfridge-spencer games? in *Proc. 35th Int. Conf. on Machine Learning*, Stockholmsmässan, Sweden, 2018, pp. 4235–4243.

[34] J. Z. Leibo, V. Zambaldi, M. Lanctot, and J. Marecki, Multi-agent reinforcement learning in sequential social dilemmas, in *Proc. 16th Conf. on Autonomous Agents and MultiAgent Systems*, São Paulo, Brazil, 2017, pp. 464–473.

[35] A. Lerer and A. Peysakhovich, Maintaining cooperation in complex social dilemmas using deep reinforcement learning, arXiv preprint arXiv: 1707.01068, 2018.

[36] J. Z. Leibo, J. Perolat, E. Hughes, S. Wheelwright, A. H. Marblestone, E. Duéñez-Guzmán, P. Sunehag, I. Dunning, and T. Graepel, Malthusian reinforcement learning, in *Proc. 18th Int. Conf. on Autonomous Agents and MultiAgent Systems*, Montreal, Canada, 2019, pp. 1099–1107.

[37] Y. C. Ho, Team decision theory and information structures, *Proc. IEEE*, vol. 68, no. 6, pp. 644–654, 1980.

[38] X. F. Wang and T. Sandholm, Reinforcement learning to play an optimal Nash equilibrium in team Markov games, in *Proc. 15th Int. Conf. on Neural Information Processing Systems*, Cambridge, MA, USA, 2002, pp. 1603–1610.

[39] T. Yoshikawa, Decomposition of dynamic team decision problems, *IEEE Trans. Autom. Control*, vol. 23, no. 4, pp. 627–632, 1978.

[40] M. L. Littman, Markov games as a framework for multi-agent reinforcement learning, in *Machine Learning Proceedings 1994*, W. W. Cohen and H. Hirshpp, eds. Amsterdam, the Netherlands: Elsevier, 1994, pp. 157–163.

[41] J. L. Hu and M. P. Wellman, Nash q-learning for general-sum stochastic games, *J. Mach. Learn. Res.*, vol. 4, pp. 1039–1069, 2003.

[42] M. G. Lagoudakis and R. Parr, Learning in zero-sum team Markov games using factored value functions, in *Proc. 15th Int. Conf. on Neural Information Processing Systems*, Cambridge, MA, USA, 2002, pp. 1659–1666.

[43] M. L. Littman, Friend-or-foe Q-learning in general-sum games, in *Proc. 18th Int. Conf. on Machine Learning*, San Francisco, CA, USA, 2001, pp. 322–328.

[44] C. Dwork, N. Lynch, and L. Stockmeyer, Consensus in

the presence of partial synchrony (Preliminary version), in *Proc. 3rd Annu. ACM Symp. on Principles of Distributed Computing*, Vancouver British Columbia, Canada, 1984, pp. 103–118.

[45] P. Marbach and J. N. Tsitsiklis, Simulation-based optimization of Markov reward processes: Implementation issues, in *Proc. 38th IEEE Conf. on Decision and Control*, Phoenix, AZ, USA, 1999, pp. 1769–1774.

[46] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[47] R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, *Mach. Learn.*, vol. 8, nos. 3&4, pp. 229–256, 1992.

[48] R. S. Sutton, Generalization in reinforcement learning: Successful examples using sparse coarse coding, in *Proc. 8th Int. Conf. on Neural Information Processing Systems*, Denver, CO, USA, 1995, pp. 1038–1044.

Yifei Zou received the BEng degree from Wuhan University, China in 2016, and the PhD degree from the University of Hong Kong, China in 2020. He is currently an assistant professor at the School of Computer Science and Technology, Shandong University, Qingdao, China. His research interests include wireless networks, ad hoc networks, and distributed computing.

Zongjing Jin received the BEng degree from Shandong University, Qingdao, China in 2020. He is currently a master student at the School of Computer Science and Technology, Shandong University, Qingdao, China. His research interests include wireless networks, distributed computing, and deep reinforcement learning.

Yanwei Zheng received the PhD degree in 2019 from Beihang University, Beijing, China in 2019. He is currently an assistant researcher at the School of Computer Science and Technology, Shandong University, Qingdao, China. His research interests include object navigation and computer vision.

Dongxiao Yu received the BS degree from Shandong University, China in 2006, and the PhD degree from the University of Hong Kong, China in 2014. He became an associate professor at the School of Computer Science and Technology, Huazhong University of Science and Technology in 2016. He is currently a professor at the School of Computer Science and Technology, Shandong University. His research interests include wireless networks, distributed computing, and graph algorithms.

Tian Lan received the BEng degree from Tsinghua University, China in 2003, the MEng degree from the University of Toronto, Canada in 2005, and the PhD degree from Princeton University, USA in 2010. He is currently a full professor of electrical and computer engineering at George Washington University, USA. His research interests include network optimization, algorithms, and machine learning. He received the Meta Research Award in 2021, SecureComm Best Paper Award in 2019, SEAS Faculty Recognition Award in 2018, Hegarty Faculty Innovation Award in 2017, AT&T VURI Award in 2015, IEEE INFOCOM Best Paper Award in 2012, Wu Prizes for Excellence at Princeton University in 2010, IEEE GLOBECOM Best Paper Award in 2009, and IEEE Signal Processing Society Best Paper Award in 2008.