A Dueling Deep Q-Learning Approach to Blockchain-Enabled Industrial Internet of Things

Chao Qiu, Haipeng Yao, Fangmin Xu, and Chenglin Zhao

***Abstract*—With the development of communication technologies and smart manufacturing, industrial Internet of Things (IIoT) has emerged. Software defined networking (SDN), a promising paradigm shift, has provided a viable way to manage IIoT dynamically, called software defined industrial Internet of things (SDIIoT). In SDIIoT, huge amounts of data and flows are generated by industrial devices, where a physically distributed and logically centralized control plane is necessary. However, one of the most intractable problems is how to reach consensus among multiple controllers under complex industrial environments. In this paper, we propose a blockchain-based consensus protocol in SDIIoT, along with detailed consensus steps and theoretical analysis. In this blockchain-based SDIIoT, we jointly consider the trust features of consensus nodes and controllers, as well as the computational capability of blockchain system. Accordingly, we formulate view changes, access selection, and computational resources allocation as a joint optimization problem. We describe this problem as a Markov decision process by defining state space, action space, and reward function. Then we use a novel dueling deep Q-learning approach to solve this problem. Simulation results are presented to show the effectiveness of our proposed scheme.**

***Index Terms*—Industrial Internet of things (IIoT), Software Defined Networking (SDN), Blockchain, Dueling deep Q-learning.**

I. Introduction

Recently, there are a growing number of applications that use Internet of Things (IoT) technologies in several industries, such as industrial monitoring, industrial management, smart manufacturing, smart factory, etc. Industrial Internet of Things (IIoT) has emerged and attracted lots of attentions from industry and academia [1]. Due to millions of devices and a large amount of data in IIoT, it is barely possible to meet the demands of high bandwidth, ubiquitous accessibility, and dynamic management by traditional static network architecture.

Software defined networking (SDN) [2] has been seen as a promising paradigm shift to solve these issues. It is an approach to provide the abstraction of underlying infrastructures, separate data plane and control plane, and introduce the ability of network reconfigurability. Such aspects of SDN are believed to simplify and improve the management of IIoT with the potential of tailoring network behaviors according to different demands. Some works have advocated to employ SDN in IIoT, called SDIIoT [3]. Software defined routing management, edge computing, flow scheduling have been researched in excellent literature [4]–[6].

Despite all these excellent strengths, there are some concerns and arguments about the widespread adoption of SDIIoT. For example, some approaches in SDN are only based on a single controller. With the increasing number of industrial devices, switches, data, and flows, it can be anticipated that one controller in SDIIoT will fail to deal with all requests, which poses serious limitations to the scalability and reliability of SDIIoT. Therefore, the physically distributed and logically centralized control plane is necessary in SDIIoT [7]. The distributed SDIIoT enables to ease the issues of a single controller, such as performance bottlenecks, single point of failure (SPOF), etc. Some excellent distributed SDN architectures have been designed in the recent literature, such as Onix [8], Hyperflow [9], Kandoo [10], ONOS [11], and Espresso [12]. Whether these traditional distributed architectures are suitable for IIoT needs to be further researched.

However, how to reach consensus in the distributed SDIIoT is challenging. For instance, some industrial devices are controlled by one controller. The events and the OpenFlow commands in a subset of industrial devices need to be kept consensus among all controllers, so as to mitigate the issue brought by the misunderstanding of global network views. Some researches have deployed different approaches to achieve consensus. To be more specific, Onix replicates and distributes network information base (NIB), and HyperFlow uses a publish/subscribe messaging paradigm.

Although these methods enable to reach consensus among multiple controller instances, there are numerous significant challenges remaining to be addressed before their widespread deployments in SDIIoT, including 1) the extra overheads exchanged among multiple controller instances, which may significantly weaken the performance of controllers' inherent functions (e.g., routing decisions, network management, etc.), especially under the limited capacities of industrial devices, 2) the poor safety and

C. Qiu, H. Yao, F. Xu, and C. Zhao are with Beijing Key Laboratory of Space-ground Interconnection and Convergence, Beijing University of Posts and Telecommunications, Beijing, 100876, China.

liveness properties. For example, each controller in NIB holds a set of key-value pair and is identified by a flat, 128-bit, global identifier [8], which is easy to be broken by some adversaries, let alone the publish/subscribe mode in HyperFlow, 3) the limited scope of available network size, which means that they are only intended for small to medium-sized SDN networks, not large-scale industrial scenarios. These challenges need to be broadly tackled through comprehensive research efforts.

Recently, *Blockchain* (BC) [13] has been emerged as a novel technique, which can be used to address the above challenges. BC is a distributed ledger to record transactions, and provides trustworthy services to a group of nodes without central authority. For the distributed SDIIoT, BC can act as a trusted third and 'out-of-band' party to collect and synchronize network-wide views from industrial devices (e.g., network events, network topology, and OpenFlow commands, etc.) safely, dependably, and traceability.

In this paper, we propose a BC-based consensus protocol in distributed SDIIoT, along with detailed consensus steps and theoretical analysis. Specifically, it is a permissioned BC. Permissioned BC addresses several problems that have been researched in the field of distributed computing over decades, such as *Byzantine Fault Tolerance (BFT)*. Based on these researches, we jointly consider the trust features of nodes and controllers, as well as the computational capability of BC systems. Accordingly, we formulate view changes, access selection, and computational resources allocation as a joint optimization problem. We describe this problem as a Markov decision process by defining state space, action space, and reward function. Then we use a novel dueling deep Q-learning approach to solve this problem. The distinct contributions of this paper are as follows.

- We propose a BC-based consensus protocol to simplify and secure the collection and synchronization of network states in distributed SDIIoT. And we give consensus steps in detail, along with theoretical analysis.
- Taking a joint consideration of trust features and computational capability in BC system, we formulate view changes, access selection, and computational resources allocation as a joint optimization problem. We describe it as a Markov decision process by defining state space, action space, and reward function.
- In order to address this issue, we propose a novel dueling deep Q-learning approach to learn the optimal strategy.
- Simulation results with different system parameters are presented to show the effectiveness of our proposed scheme.

The rest of this paper is organized as follows. In Section II, we present some related works about SDIIoT, distributed SDN, BC, and BC consensus protocols. Section III describes the BC-based consensus protocol, followed by the system model in Section IV. In Section V, we describe view changes, access selection, and computational resources allocation as a Markov decision process. And we use a novel dueling deep Q-learning approach to solve this issue in Section VI. Simulation results are presented and discussed in Section VII. Finally, conclusions and future works are given in Section VIII.

II. Related Work

In this section, we briefly present the recent advances in SDIIoT and distributed SDN. Some challenges are discussed as well. Then the overviews of BC, especially the consensus protocols in BC, are presented.

A. Software Defined Industrial Internet of Things

There are a number of advances in IIoT, such as industrial wireless sensor networks, industrial big data, cloud computing, and fog computing. These emerging techniques bring the explosion of intelligent devices, industrial robots that are supported by wired or wireless networks. Due to millions of devices and data in IIoT, it is barely possible to meet the demands of high bandwidth, ubiquitous accessibility, and dynamic management by traditional TCP/IP networks.

Due to its reconfigurability, real-time, reliability, and flexibility, SDN is a promising architecture to solve the above problems. Therefore, lots of works have employed SDN in IIoT. Then we introduce some parts of them, according to the advantages of SDN.

1) Reconfigurability of SDN: SDN benefits the management of IIoT by supporting networks reconfigurability. The authors in [4] considered the dynamics of multiple controllers and multiple sinks in a smart factory. Reconfigurability of SDN was used to ensure that each sensor was covered anytime and anywhere. In this way, the total deployment cost was significantly decreased. In addition, due to reconfigurability, manufacturing resources enable to be allocated dynamically. Thus, Wan *et al.* in [14] proposed a software defined industrial network to manage manufacturing resources dynamically.

2) Real-time of SDN: Real-time communications are necessary for time-sensitive applications in IIoT. Traditional transmission schedules of time-sensitive traffic are synthesized offline and fixed in a period of time. Nayak *et al.* in [6] presented a software defined architecture to add and remove network applications. Global network views in the control plane were used to schedule and route time-sensitive flows dynamically in IIoT. Aiming at real-time reservations in industry 4.0, Silva *et al.* in [15] expanded OpenFlow protocol in industrial scenarios. They presented the implementation of OpenFlow in IIoT, and showed the improved time efficiency.

3) Reliability of SDN: The connectivity of enormous industrial devices requires reliable management and control systems. Some works have utilized the reliability of SDN to solve the problem. For example, Moness *et al.* in [16] employed a hybrid software defined approach in IIoT. By this software defined IIoT, the reliability of event-driven applications and the capacities of handling real-time packets were significantly improved. In order to enhance the reliability of SDIIoT, Baddeley *et al.* in [7] considered a two-layer slicing mechanism to relieve SDN control overheads. Here, they implemented more than one controller, but did not mention how to reach consensus among them. Using the reliability of SDN, Yan *et al.* in [17] devised a DDoS mitigation system, integrating edge computing, fog computing, cloud computing, and SDN. A large number of industrial devices were managed to mitigate DDoS attacks under this multi-level system. Energy consumption, hardware malfunctions and wireless channel congestion in wireless sensor networks lead to the problem of reliability in IIoT, Duan *et al.* in [18] studied a software defined wireless sensor network in IIoT. This network addressed the problem of node failure and coverage to improve the reliability of IIoT.

4) Flexibility of SDN: Industrial networks have transformed from single-function networks to large-scale and heterogeneous networks. Therefore, flexible management of SDN is necessary to handle the differences of devices and networks. Duan *et al.* in [19] described a software defined and virtual multiple networks control framework. By this framework, heterogeneous and extensible networks were implemented. The authors in [20] considered SDN as an important part to enable flexible interfacing in IIoT based smart grid. Therefore, they devised a novel SDIIoT framework to provide the flexible recovery of failure nodes in smart grid. Multi-functionality control and real-time monitoring were achieved by this framework. Similarly, Wan *et al.* in [3] designed a software defined IIoT framework to achieve flexible networks. They considered different IIoT layers, and used this software defined IIoT framework to manage industrial devices and communication interfaces.

As we can see that there is an emerging trend to employ SDN in IoT. More than one controller has been considered in some works, called distributed SDIIoT. Whether traditional distributed SDN architectures are suitable in IIoT, and how to reach consensus among controllers need to be further researched. Therefore, we will present some traditional distributed SDN architectures, along with the corresponding consensus methods.

B. Distributed SDN Architectures

The physically distributed and logically centralized control plane architecture is more and more popular to ease the performance bottlenecks and SPOF in SDIIoT. Traditionally, there are lots of works to design distributed SDN architectures [21]. Based on the configuration methods between controllers and switches, we classify them as statically configured control architecture and dynamically configured control architecture.

1) Statically Configured Control Architecture: The authors in [9] designed the first distributed SDN control plane for OpenFlow, called HyperFlow. Specially, it was an application in NOX controller [22], and synchronized controllers' network-wide views (i.e., local controller instance's events) by publish/subscribe messaging paradigm. The work of [23] deployed a network virtualization layer between control plane and data plane, and virtualized five primary slicing dimensions, including bandwidth, topology, traffic, device CPU, and forwarding tables. Here, slicing and virtualizing made the possibility of distributed SDN control plane. Koponen *et al.* in [8] considered a Onix architecture. Each controller instance was in charge of a subset of NIB to aggregate and share network-wide views. The authors in [10] presented a two-layer architecture, called Kandoo. Here, top-layer controller was used to maintain and synchronize network-wide states.

2) Dynamically Configured Control Architecture: Statically configured control architectures lead to the uneven load distribution among controllers. Therefore, Dixit *et al.* in [24] proposed an elastic distributed control plane architecture, called ElastiCon. They used controllers pool to build the distributed control plane. Similarly, Berde *et al.* in [11] designed an experimental distributed SDN control plane, called ONOS. Each switch belonged to one master controller and several backup controllers to guard against the failures of controllers. Load balancing mechanisms were considered in ONOS as well. The work in [25] presented a hierarchical SDN control architecture with the joint consideration of load balancing and energy consumption of multiple controllers.

As we can see from the above researches, one of the most important issues in distributed SDN architectures is how to reach consensus among multiple controller instances. No matter which consensus protocols (e.g., publish/subscribe, slicing, NIB, and controllers pool, etc.), lots of challenges are remained to be solved in SDIIoT as follows.

- The traditional consensus protocols are implemented 'in-band', which leads to the extra overheads among controllers and in each controller to weaken the inherent functions of controllers (e.g. routing decisions, and network management, etc.). Especially under the limited capacities of industrial devices, this problem is more severe. Therefore, a third party and 'out-of-band' consensus protocol is necessary in SDIIoT.
- Safety and liveness properties are ignored in traditional works. As the brain of SDN, the failures of control

plane have incalculable consequences. In open industrial networks, a number of adversaries and attacks are easier to affect safe communications. Therefore, trustworthy and dependable consensus protocols are stringently required SDIIoT.
- With the widespread deployments of industrial networks, the limited scopes of these consensus protocols are challenging. A large-scale consensus protocol should be employed in future SDIIoT.

These challenges have fueled the needs to explore new consensus protocols in SDIIoT. Inspire by the successful integration BC with secure key synchronization [26], and data sharing [27] in intelligent transportation system, we consider BC is a potential solution for these problems.

C. Overview of Blockchain

With the emergence of Bitcoin in late 2008 [13], lots of attentions have been attracted by BC. BC is a distributed system to provide dependable services to a group of nodes that don't fully trust each other. All nodes identify themselves by public keys, and communicate with others by transactions. They collect transactions to form a block periodically. In addition, this block is encrypted by public key, message authentication code (MAC), signature, and collision-resistant hashing to provide high-level security protection. All other nodes need to validate these cryptographic methods. Finally, an authenticated block is appended into the chain. Here, consensus protocol is necessary to ensure that all nodes agree on a unique order where this block is appended [28].

Therefore, consensus protocols are important in BC. Consensus algorithms that are implemented in BC vary, which can be classified into working-based algorithms and replica-based algorithms.

- Working-based algorithms. In these algorithms, lots of participants (miners) contribute their CPU power to work on an extra hard computation task. The winner of them enables to propose a block and synchronize it with the rest of participants. The working-based algorithms always appear coupled to cryptocurrency, e.g., Bitcoin [13] and Ether in Ethereum [29]. Some typical algorithms are Proof of Work (PoW), Proof of Stake (PoS), Proof of Elapsed Time (PoET), etc. They enable to work with a large number of nodes, but impose too much costs in terms of resources and time. Therefore, they are widely used in permissionless blockchain, such as Bitcoin and Ethereum.
- Replica-based algorithms. This type of algorithms tolerates *Byzantine* nodes that may be subverted by some adversaries and against the common goal of reaching consensus maliciously [30]. And they use state machine replication mechanism to deal with these *Byzantine* nodes. In this mechanism, they transfer the replica of a transaction or a block to others. When a certain number of nodes validate it, consensus exists and finality occurs. The most prominent replica-based algorithms are *Practical Byzantine Fault-Tolerance (PBFT)* [31], *Paxos* [32], etc. Their advantages are low costs and low latency. But since they rely on a *primary*, which can be smartly malicious, to order a transaction or a block, they are not robust. Therefore, they always operate in partially trusted environments, i.e., permissioned blockchain, such as Hyperledger Fabric [33].

For a comprehensive perspective, we offer an at-a-glance view of the main considerations, pros and cons of these two types of consensus algorithms in Table I.

As we can see that BC, as a third party system, enables to achieve an agreement among nodes without a central trust broker, with the features of dependability, and in largely available system scale. Therefore, BC could be a potential approach to address the aforementioned challenges in distributed SDIIoT. In addition, the consensus of distributed SDIIoT operates in partially trusted environment. Therefore, we utilize permissioned blockchain to reach consensus in distributed SDIIoT.

III. Blockchain-Based Consensus Protocol

We have presented that the existing consensus protocols are challenging in SDIIoT, and BC could be a potential approach to address these issues in the previous section. In this section, we propose a novel BC-based consensus protocol in distributed SDIIoT. We begin with network model. Then we give an overview of BC-based consensus protocol. Finally, we present the detailed steps of the consensus protocol, along with theoretical analysis.

A. Network Model

There are C controllers in distributed SDIIoT, which are represented by $\mathcal{C} = \{1, \ldots, C\}$. Each of them can communicate with the third-party BC consensus system. This BC system consists of N nodes, i.e., physical machines, denoted by $\mathcal{N} = \{1, \ldots, N\}$. Like other robust BFT protocols, these N nodes are under *Byzantine* failure model to make consensus, where at most $f = \frac{N-1}{3}$ nodes are faulty [31]. And any finite number of controllers can behave arbitrarily to issue correct or incorrect transactions to the BC system. Some strong adversaries can collude with each other to compromise the replicated service. However, they can't break cryptographic technologies, i.e., signatures, MACs, and collision-resistant hashing. We denote the messages with cryptographic technologies as follows [31].

- $\langle m \rangle_{\sigma_i}$ means that message m is signed with a pubilc key from node i.
- $\langle m \rangle_{\sigma_{i,j}}$ means that message m is authenticated by node i with a MAC for node j.

TABLE I: Comparison of working-based consensus algorithms and replica-based consensus algorithms.

Algorithm \ items	Speed	Scalability	Finality	Typical approach	Application
Working-based consensus algorithms	Poor	Good	Poor	PoW, PoS, PoET, etc.	Permissionless blockchain (Bitcoin, Ethereum, etc.)
Replica-based consensus algorithms	Good	Moderate	Good	PBFT, Paxos, etc.	Permissioned blockchain (Hyperledger Fabric, etc.)

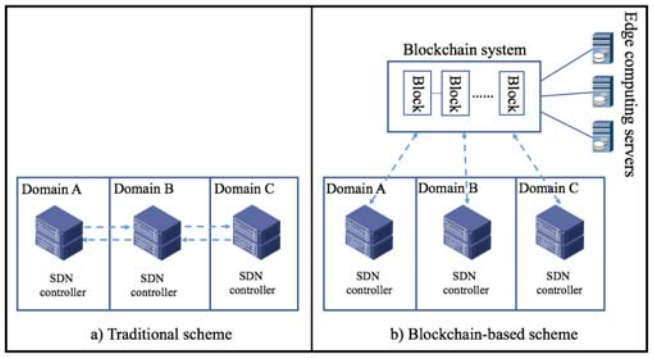

Fig. 1: The different network structures of traditional scheme and BC-based scheme.

- $\langle m \rangle_{\sigma_{\vec{i}}}$ means that message m is authenticated by an array of MACs with node i for every replicas.

Fig. 1 shows the different network structures of traditional scheme and BC-based scheme. It is worth mentioning that we use edge computing servers to do some computations related to the above cryptography so as to improve the performance. There are E edge computing servers, and the set of computing servers is represented by $\mathcal{E} = \{1, 2, ..., E\}$.

B. Overview of BC-Based Consensus Protocol

Each controller collects its local events and OpenFlow commands as Transaction #1, Transaction #2, ..., Transaction #n, which is called the collection period. The format of a transaction is shown in Table II. The number of this transaction denotes the position of this transaction. The signature and MAC make sure the integrity and authentication of this transaction. The payloads include local events and OpenFlow commands that need to be synchronized.

After the collection period, all controllers issue consensus requests to the third-party BC system. According to a policy, the BC system only enables one controller to access, and reply it by an admission message. Then, this controller sends an un-validated block with block header and transactions, whose format is presented in Table III. After reaching consensus, the BC system sends the corresponding validated block to the entire controllers. Finally, all controllers learn the payloads in each transaction to know the events and OpenFlow commands from other controller. These steps are in the consensus period. By this way, network-wide views can be synchronized in distributed SDIIoT.

TABLE II: The format of a transaction.

The number of this transaction in the block.
The Signature of this transaction.
MAC of this transaction.
Payloads, including local events and OpenFlow commands.

TABLE III: The format of a block.

Field	Description
Version	Block version number.
Timestamp	Creation time of this block.
Controller ID	The identifier of this controller.
Block ID	The identifier of this block.
Block payload	Transactions in this block (Transaction #1, ..., Transaction #n).

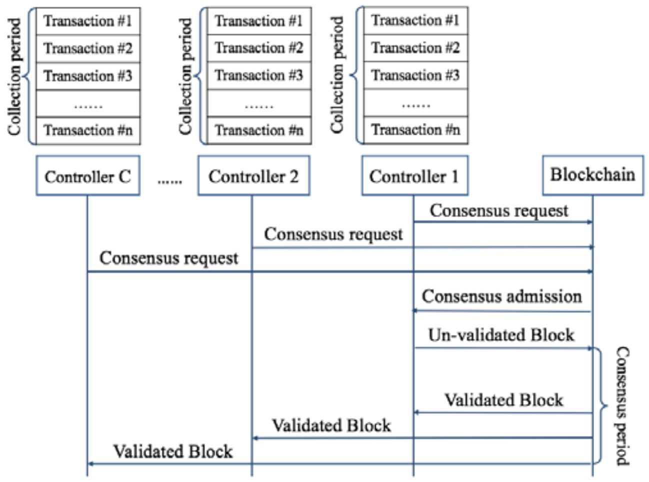

Fig. 2: The overview of consensus procedures in blockchain structure.

For a comprehensive perspective, Fig. 2 offers the consensus procedures in blockchain structure. Here, controller 1 is the selected controller to access to the BC system.

C. Detailed Steps and Theoretical Analysis

After giving the overview of BC-based consensus protocol, we will introduce the detailed steps in the consensus period, along with the theoretical analysis of each step.

Based on PBFT [31], the BC-based consensus protocol is depicted in Fig. 3. The numbering of each step in this

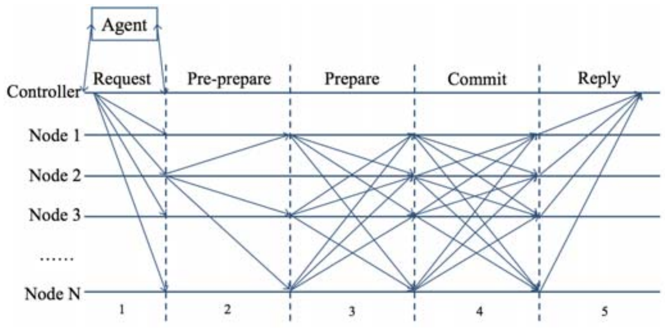

Fig. 3: The detailed procedures in the BC-based consensus protocol.

figure is the same as the one used in the remainder of this subsection.

1. The controller sends the un-validated block to all nodes. According to a certain policy, which will be introduced in the following section, an agent chooses one controller to access to the BC system, and this node is the primary node p. Making decisions about which node is primary node is known as view change protocol. The view change protocol used in our proposed scheme will be introduced in the following section. This selected controller sends a block message $\langle\langle block\rangle_{\sigma_c}, c\rangle_{\sigma_{\vec{c}}}$ to all nodes, where c denotes the controller ID. It is encrypted with the private signature of controller c, and authenticated with MACs for all nodes. When receiving this message, only primary node p verifies the MAC. If valid, the signature will be verified then. If still valid, it verifies the signature and MAC of each transaction in this block, then moves to the following steps. The success rate of all verification will be recorded and reported to the agent. If this block has already been executed, the primary will resend the validated block to this controller.

Theoretical analysis. We consider an uncivil execution during which a fraction g of transactions sent by the controller are correct [34]. The more trusted controller has the bigger g. We assume verifying one signature, generating one MAC, and verifying one MAC require θ, α, and α cycles, respectively, and the controller issues batches of transactions of size b. As the work in [31], we ignore the cost of sending or receiving transactions. Therefore, the cost at the primary is

$$(1+\frac{b}{g})(\theta+\alpha), \tag{1}$$

and there is no cost at non-primary nodes.

2. The primary node multicasts PRE-PREPARE message to other replica nodes. Finishing the verification, primary p sends a PRE-PREPARE message to all other replica nodes, as $\langle PRE-PREPARE, p, c, H(m)\rangle_{\sigma_{\vec{p}}}$, which is authenticated with MACs for each replica node. Here, p and $H(m)$ mean the primary node ID and the hashed result of the issued block, respectively. The replica node verifies the MAC of primary p, as well as the signature and MAC of each transaction. Then it enters the following steps.

Theoretical analysis. In this phase, primary p generates $(N-1)$ MACs for all replicas. Each replica verifies one MAC from primary p, as well as $\frac{b}{g}$ signatures and MACs from transactions. The cost at the primary is

$$(N-1)\alpha, \tag{2}$$

and the cost at each replica node is

$$\alpha+\frac{b}{g}(\theta+\alpha). \tag{3}$$

3. The replicas send PREPARE message to others. After verifying the validity of MACs and signatures, each replica replies the PRE-PREPARE message with sending a PREPARE message to all nodes, as $\langle PREPARE, p, c, H(m), n\rangle_{\sigma_n}$, where n denotes the replica node ID. When each replica node collects $2f$ matching PREPARE messages with its local PRE-PREPARE message, it will enter the following steps.

Theoretical analysis. In this phase, primary p needs to verify $2f$ MACs. Each replica node generates $(N-1)$ MACs and verifies $2f$ MACs. The cost at the primary is

$$2f\alpha, \tag{4}$$

and the cost at each replica is

$$(N-1+2f)\alpha. \tag{5}$$

4. All nodes send COMMIT message to others. Following the reception of $2f$ matching PREPARE messages, node n sends a COMMIT message to all others, as $\langle COMMIT, p, c, H(m), n\rangle_{\sigma_{\vec{n}}}$. After receiving $2f$ matching COMMIT messages, it will enter the following steps.

Theoretical analysis. In this phase, primary p needs to generate $(N-1)$ MACs, and verify $2f$ MACs. Each replica generates $(N-1)$ MACs, and verifies $2f$ MAC. Therefore, the costs at the primary and the replica are both

$$(N-1+2f)\alpha. \tag{6}$$

5. The nodes send the validated block to all controllers. Node n sends a REPLY message $\langle REPLY, block, n\rangle_{\sigma_{n,c}}$ to all controllers, where $block$ is the validated block. When each controller receives $2f$ valid and matching REPLY messages, it accepts this validated block and updates the corresponding network views.

Theoretical analysis. In this phase, the primary and the replicas need to generate $\frac{b}{g}$ MACs for one controllers. Therefore, the total costs at the primary and the replica are both

$$\frac{b}{g}C\alpha. \tag{7}$$

For one transaction, the cost at the primary is

$$(\frac{1}{g}+\frac{1}{b})\theta+(\frac{1}{b}+\frac{C+1}{g})\alpha+\frac{2N+4f-2}{b}\alpha \tag{8}$$

For one transaction, the cost at the replica is

$$\frac{1}{g}\theta+\frac{2+C}{g}\alpha+\frac{2N+4f-2}{b}\alpha \tag{9}$$

We assume multi-core computation modules run in parallel on distinct cores. Each core has the computation speed of φ Hz. In addition, we consider an uncivil execution where the primary is not fully trusted. The primary has k trust to affect the system performance, and $k \in [0,1]$. The more trusted primary node has the bigger k. Therefore, the throughput of the consensus protocol is at most

$$\begin{aligned}\min[&\frac{k\varphi}{(\frac{1}{g}+\frac{1}{b})\theta+(\frac{1}{b}+\frac{C+1}{g})\alpha+\frac{2N+4f-2}{b}\alpha},\\ &\frac{k\varphi}{\frac{1}{g}\theta+\frac{2+C}{g}\alpha+\frac{2N+4f-2}{b}\alpha}]trx/s\end{aligned} \tag{10}$$

IV. System Model

As shown in (10), there are two factors to affect the throughput of the consensus protocol, including trust features of nodes and controllers, and the computational capability of the BC system. Thus, in this section, we present trust feature model and computation model.

A. Trust Feature Model

We consider trust features of nodes and controllers in the system. Due to the lack of centralized security services and prior security association, all nodes and controllers have diverse trust features, such as safe or compromised. It is barely possible to exactly know what the trust feature is for one node or one controller at the next time instant. Thus, the trust features of an arbitrary node $n \in \{1,2,\ldots,N\}$ and an arbitrary controller $c \in \{1,2,\ldots,C\}$ can be modelled as random variables δ^n and η^c. δ^n and η^c can be divided into discrete levels, denoted by $\xi = \{\xi_0,\xi_1,\ldots,\xi_{L-1}\}$, and $\mathcal{D} = \{\mathcal{D}_0,\mathcal{D}_1,\ldots,\mathcal{D}_{H-1}\}$, respectively, where L and H are the number of available trust features for nodes and controllers. We assume trust features realization of δ^n and η^c to be $\delta^n(t)$ and $\eta^c(t)$ at time slot t, respectively. There are T time slots during the period of time, which starts from when the controller issues the un-validated block, and terminates when the controller is replied with the validated block. Let $t \in \{0,1,2,\ldots,T-1\}$ denote the time instant.

We model the transition of trust features in nodes and controllers as a Markov chain as follows:

- For node n, let the transition probability of $\delta^n(t)$ from one state $\mathcal{X}_s$ to another state $\mathcal{Y}_s$ be $\kappa_{\mathcal{X}_s\mathcal{Y}_s}(t)$. The $L\times L$ transition probability matrix $\mathcal{K}^n(t)$ of the trust feature in node n is denoted as

$$\mathcal{K}^n(t) = [\kappa_{\mathcal{X}_s\mathcal{Y}_s}(t)]_{L\times L}, \tag{11}$$

  where $\kappa_{\mathcal{X}_s\mathcal{Y}_s}(t) = Pr(\delta^n(t+1) = \mathcal{Y}_s|\delta^n(t) = \mathcal{X}_s)$, and $\mathcal{X}_s,\mathcal{Y}_s \in \xi$.
- For controller c, the transition probability of $\eta^c(t)$ is $\gamma_{\theta_s\phi_s}(t)$. The $H\times H$ transition probability matrix $\Upsilon^c(t)$ of the trust feature in controller c is denoted as

$$\Upsilon^c(t) = [\gamma_{\theta_s\phi_s}(t)]_{H\times H}, \tag{12}$$

  where $\gamma_{\theta_s\phi_s}(t) = Pr(\eta^c(t+1) = \phi_s|\eta^c(t) = \theta_s)$, and $\theta_s,\phi_s \in \mathcal{D}$.

B. Computation Model

There are a number of computation tasks in the primary and the replicas, such as verifying signatures, generating MACs, and verifying MACs. Let $T_m = \{s_m,q_m\}$ denote a computation task related to message m, where s_m means the size of message m, and q_m is the required number of CPU cycles to complete this task.

In order to improve the system throughput, in this paper, we use virtual computing resources from edge computing servers to do computation tasks in the BC system. There are lots of edge computing servers and some other computation tasks that also use the computation resources in edge computing servers, thus we don't exactly know the computational resources for the BC system at the next time instant. Therefore, we model the computation resources of edge computing server e for the BC system as a random variable ζ^e. To discretize the values of computation capabilities, ζ^e can be partitioned into Y discrete intervals as $\mathcal{Y} = \{\mathcal{Y}_0,\mathcal{Y}_1,\ldots,\mathcal{Y}_{Y-1}\}$. The computation resources from edge computing server e at time slot t can be denoted as $\zeta^e(t)$, $t \in \{0,1,2,\ldots,T-1\}$. Based on a certain transition probability, $\zeta^e(t)$ changes from one state to another. Let $\vartheta_{a_sb_s}(t)$ denote the transition probability. The $Y\times Y$ computation state transition probability matrix is represented as:

$$\Pi^n(t) = [\vartheta_{a_sb_s}(t)]_{Y\times Y}, \tag{13}$$

where $\vartheta_{a_sb_s}(t) = Pr(\zeta^e(t+1) = b_s|\zeta^e(t) = a_s)$, and $a_s,b_s \in \mathcal{Y}$.

The execution time of computation task T_m can be denoted as

$$t_m = \frac{q_m}{\zeta^e(t)}. \tag{14}$$

Thus, the computation rate is

$$CompR^e(t) = a^e(t)\frac{s_m}{t_m} = a^e(t)\frac{\zeta^e(t)s_m}{q_m}, \tag{15}$$

where $a^e(t)$ means whether or not edge computing server e is allocated to the BC system at time slot t. $a^e(t) = 1$

denotes edge computing server e is allocated to the BC system; otherwise $a^e(t) = 0$. At one time slot, there is only one edge computing server allocated to the BC system, thus $\sum_{e=1}^{E} a^e(t) = 1$.

V. Problem Formulation

We have presented the throughput of the BC system, and the models of two factors related to the throughput. In order to improve the performance, we need to make the joint decisions about view changes, access selection, and computational resources allocation. In this section, we formulate this issue as a Markov decision process by defining state space, action space, and reward function.

A. State Space

The state space is the trust features of all nodes and controllers, and the computation capabilities of all edge computing servers. Therefore, the state space can be represented as follows.

$$S(t) = \begin{bmatrix} \delta^1(t) & \delta^2(t) & \dots & \delta^N(t) \\ \eta^1(t) & \eta^2(t) & \dots & \eta^C(t) \\ \zeta^1(t) & \zeta^2(t) & \dots & \zeta^E(t) \end{bmatrix}. \tag{16}$$

B. Action Space

The agent mainly needs to decide view changes (i.e., which node is the primary node), access selection (i.e., which controller can access to the BC system), and computational resources allocation (i.e., which edge computing server should be allocated to the BC system). Thus, the action space is denoted by

$$A(t) = \{A^N(t), A^C(t), A^E(t)\}, \tag{17}$$

where $A^N(t), A^C(t)$, and $A^E(t)$ represent:

- $A^N(t) = [a^1(t), a^2(t), ..., a^n(t), ..., a^N(t)]$, which means whether or not node n is the primary. And $a^n(t) \in \{0, 1\}$, where $a^n(t) = 1$ means node n is the primary node, otherwise it is the replica node. Note that the BC system only has one primary, thus $\sum_{n=1}^{N} a^n(t) = 1$.
- $A^C(t) = [a^1(t), a^2(t), ..., a^c(t), ..., a^C(t)]$ decides which controller can access to the BC system. And $a^c(t) \in \{0, 1\}$, where $a^c(t) = 1$ represents controller c can access, otherwise $a^c(t) = 0$. Note that at one time slot, only one controller enables to access to the BC system, thus $\sum_{c=1}^{C} a^c(t) = 1$.
- $A^E(t) = [a^1(t), a^2(t), ..., a^e(t), ..., a^E(t)]$ determines which edge computing server is allocated to the BC system. And $a^c(t) \in \{0, 1\}$, where $a^e(t) = 1$ denotes edge computing server e is allocated, otherwise $a^e(t) = 0$. Similarly, $\sum_{e=1}^{E} a^e(t) = 1$.

C. Reward Function

We model the throughput of the BC system as the reward function. From (10) and the above description, we define the reward function as:

$$\begin{aligned} &\min[\frac{k'\varphi'}{(\frac{1}{g'} + \frac{1}{b})\theta + (\frac{1}{b} + \frac{C+1}{g'})\alpha + \frac{2N+4f-2}{b}\alpha}, \\ &\frac{k\varphi}{\frac{1}{g'}\theta + \frac{2+C}{g'}\alpha + \frac{2N+4f-2}{b}\alpha}]trx/s \end{aligned} \tag{18}$$

where $k' = \sum_{n=1}^{N} a^n(t)\delta^n(t)$, $\varphi' = \sum_{e=1}^{E} a^e(t)CompR^e(t)$, and $g' = \sum_{c=1}^{C} a^c(t)\eta^c(t)$.

VI. Dueling Deep Q-Learning

After the problem formulation, we consider a dueling deep Q-learning approach to address this problem. In this section, we begin with the introduction of Q-learning and deep Q-learning. Then we present dueling deep Q-learning that is used in this paper.

A. Q-Learning

In the problem formulation, the agent needs to decide which node is the primary, which controller enables to access to the BC system, and which edge computing server could be offloaded to execute computation tasks. Since the system is high-dynamical and high-dimensional, it is hard to make the joint and optimal decisions by traditional methods. Therefore, we use a Q-learning approach to solve this issue.

In the Q-learning model, the agent interacts with the environment by perceptions and actions. In one interaction step, the agent receives current state s from the environment, then selects an action a as the output. This action generates next state s', and the value of this state transition is measured by a scalar reward r. The agent selects actions to obtain the maximum long-term rewards. It learns to do this over several interaction steps by systematic trials and errors, guided by Q-learning [35]. Q-learning is a model-free algorithm using delay rewards. It aims to find a policy π, mapping states and actions, to maximize the long-term rewards.

There are two popular approaches to denote the feedback from each step in terms of long-term rewards, namely state-value function $V^\pi(s)$ and action-state value function $Q^\pi(s, a)$. $V^\pi(s)$ means the expected total reward ins state s:

$$V^\pi(s) = E^\pi[\sum_{k=1}^{\infty} \gamma^k r_{t+k+1} | s_t = s], \tag{19}$$

where $E^\pi[*]$ means mathematical expectation, r_{t+k+1} means the immediate reward at time slot $t + k + 1$, and $\gamma \in (0, 1]$ is discount factor to balance immediate reward and future reward. $Q^{\pi}(s,a)$ denotes the expected total rewards in state s and action a:

$$Q^{\pi}(s,a) = E^{\pi}[\sum_{k=1}^{\infty}\gamma^{k} r_{t+k+1}|s_t = s, a_t = a]. \tag{20}$$

Q-learning evaluates $Q(s,a)$ by a temporal difference method as follows.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)), \tag{21}$$

where α is learning rate in Q-learning, and $\alpha \in (0,1]$. The action with maximum $Q(s,a)$ can be chosen by the agent at each step. In the traditional Q-learning, each $Q(s,a)$ is put into Q-table. However, with the rapid increase of data dimension, it is challenging to put all $Q(s,a)$ into Q-table.

B. Deep Q-Learning

The rise of deep learning has provided a new tool to overcome the challenges. The most important property of deep learning is that deep networks enable to find the low-dimensional features of high-dimensional data by crafting weights and biases in deep networks. Therefore, many researches have advocated to use deep networks to approximate $Q(s,a)$ instead of Q-table, i.e., $Q(s,a,\omega) \approx Q(s,a)$, where ω is the set of weights and biases in deep networks [36]. This is the core idea of deep Q-learning (DQN).

In order to address the fundamental instability problem of approximating $Q(s,a)$, there are two improvements in DQN, including experience replay and fixed target networks: 1) Experience replay stores the transitions as a set of $\{state, action, reward, state_{next}\}$ in a finite-sized cyclic buffer, and the agent randomly samples batches of them to train deep networks, instead of only the current ones. By this way, the temporal correlations that can adversely affect DQN are broken. 2) Fixed target networks have the same architecture as the evaluated ones, but are kept frozen for a period of time. The evaluated networks are trained in each step to minimize loss function $L(\omega)$ to evaluate real $Q(s,a)$, and $L(\omega)$ is represented as

$$L(\omega) = E[(r + \gamma max_{a'} Q(s',a',\omega^{-}) - Q(s,a,\omega))^2], \tag{22}$$

where ω^{-} is the weights and biases set in target networks, and ω is the weights and biases set in evaluated networks. During training, the weights and biases in target networks are updated with evaluated networks periodically.

C. Dueling Deep Q-Learning Approach

However, for the majority of states in our system, the choice of actions in the agent has no repercussion with what happens, i.e., actions have no relationship with states. According to the work in [37], dueling DQN is more efficient than natural DQN.

In the dueling DQN, there is another value function, $A(s,a)$, which represents the relative advantage of a action, called state-action value function. Learning $A(s,a)$ is easier to know which action has better consequences. Instead of one single stream following the output layer of deep networks, there are two separate streams in dueling DQN, where one computes state-value function $V(s)$, and another computes state-action value function $A(s,a)$, called dueling architecture. Finally, these two streams are aggregated as a output $Q(s,a)$. This combination module is denoted as follows:

$$Q(s,a;\omega,\varrho,\zeta) = V(s;\omega,\varrho) + A(s,a;\omega,\zeta), \tag{23}$$

where $V(s;\omega,\varrho)$ is a scalar, and $A(s,a;\omega,\zeta)$ is an $|\mathcal{A}|-$dimensional vector. ϱ and ζ are the parameters of two separate streams.

According to the work in [37], considering the unidentifiability of (23), we replace (23) as:

$$Q(s,a;\omega,\varrho,\zeta) = V(s;\omega,\varrho) + (A(s,a;\omega,\zeta) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\omega,\zeta)). \tag{24}$$

As the output of dueling architecture is $Q(s,a)$, it can be trained by many existing algorithms. In this paper, we use dueling architecture in natural DQN. Therefore, the pseudocode of dueling DQN is presented in Algorithm 1, where $\epsilon-$greedy policy is used to balance the exploitation and the exploration.

VII. Simulation Results and Discussions

In this section, we use computer simulation to estimate the performance of our proposed scheme. First, we present simulation settings, followed by simulation results and the corresponding discussions.

some discussions about the simulation results.

A. Simulation Settings

In this simulation, hardware environment is a GPU-based server, and this server has 8GB 1867MHz LPDDR3, 2GHz Intel Core i5, and 256G memory. Software environment is Python 2.7.10 with TensorFlow 1.4.0 [38].

We assume that there are four consensus nodes, two controllers, and two edge computing servers. The trust feature of each node is very safe, safe, medium, compromised, and very compromised, whose transition probability matrix is

$$\mathcal{K} = \begin{bmatrix} 0.5 & 0.15 & 0.125 & 0.12 & 0.105 \\ 0.15 & 0.5 & 0.125 & 0.12 & 0.105 \\ 0.105 & 0.15 & 0.5 & 0.125 & 0.12 \\ 0.105 & 0.12 & 0.125 & 0.5 & 0.15 \\ 0.105 & 0.12 & 0.125 & 0.15 & 0.05 \end{bmatrix}. \tag{26}$$

Algorithm 1 Dueling DQN

```
1:  Initialization:
    Initialize evaluated deep networks with weights and biases set $w$.
    Initialize target deep networks with weights and biases set $w'$.
2:  for $k = 1 : K$ do
3:      Reset the environment with a randomly initial observation $s_{ini}$, and $s_t = s_{ini}$.
4:      while $s_t != s_{terminal}$ do
5:          Select action $a_t$ based on $\epsilon$−greedy policy.
6:          Obtain immediate reward $r_t$ and next observation $s_{t+1}$.
7:          Store experience $(s_t, a_t, r_t, s_{t+1})$ into experience replay memory.
8:          Randomly sample some batches of $(s_i, a_i, r_i, s_{i+1})$ from experience replay memory.
9:          Calculate two streams of evaluated deep networks, including $V(s;\omega,\varrho)$ and $A(s,a;\omega,\zeta)$, and combine them as $Q(s,a;\omega,\varrho,\zeta)$ using (24).
10:         Calculate target Q-value $Q_{target}(s)$ in target deep networks:
            if $s'$ is $s_{terminal}$
                $Q_{target}(s) = r_s$,
            else
                $Q_{target}(s) = r_s + \gamma max_{a'} Q(s', a'; \omega', \varrho', \zeta')$.
11:         Train evaluated deep networks to minimize loss function $L(w)$

            $L(\omega,\varrho,\zeta) = E[(Q_{target}(s) - Q(s,a;\omega,\varrho,\zeta))^2]$.   (25)
12:         Every some steps, update target deep networks.
13:         $s_t \leftarrow s_{t+1}$
14:     end while
15: end for
```

Similarly, the trust feature of each controller can be very safe, safe, medium, compromised, and very compromised. We set the transition probability matrix as

$$\Upsilon = \begin{bmatrix} 0.45 & 0.16 & 0.14 & 0.13 & 0.12 \\ 0.16 & 0.45 & 0.14 & 0.13 & 0.12 \\ 0.12 & 0.16 & 0.45 & 0.14 & 0.13 \\ 0.12 & 0.13 & 0.16 & 0.45 & 0.14 \\ 0.12 & 0.13 & 0.14 & 0.16 & 0.45 \end{bmatrix}. \tag{27}$$

We assume the computational capability of each edge computing server is high, medium, low, and very low, and set the transition probability as

$$\Pi = \begin{bmatrix} 0.5 & 0.3 & 0.15 & 0.05 \\ 0.3 & 0.5 & 0.15 & 0.05 \\ 0.15 & 0.3 & 0.5 & 0.05 \\ 0.15 & 0.3 & 0.5 & 0.05 \end{bmatrix}. \tag{28}$$

The values of the rest of parameters are summarized in Table IV.

For the performance comparison, there are four schemes simulated:

- Proposed dueling DQN-based scheme with view changes, access selection, and edge computing servers. We call it duelingDQN-based scheme in the remainder of this section.
- Proposed dueling DQN-based scheme with view changes, edge computing servers, but without access selection. We call it dueling DQN-based scheme without controller choice in the remainder of this section.
- Proposed dueling DQN-based scheme with access selection, edge computing servers, but with the traditional view changes protocol in [31]. We call it duelingDQN-based scheme without node choice in the remainder of this section.
- Proposed dueling DQN-based scheme with view changes, access selection, but without edge computing servers, which only uses the local computing capabilities in the BC system. We call it duelingDQN-based scheme without computation offloading in the remainder of this section.
- Existing scheme with traditional view changes, without access selection, and with local computational capabilities. We call it existing scheme in the remainder of this section.

B. Simulation Results

Fig. 4 shows the relationship between training episodes and the throughput of the BC system under different schemes. Each point is the average throughput per episode. The agent runs in AdamOptimizer [39] with the learning rate of $1e^{-5}$. As we can see from this figure, with the joint consideration of node's trust feature, controller's trust feature, and offloading the computation task to edge computing servers, the BC system has the better performance. The reason is that the more trusted node is less possible to slow the system performance down, the more trusted controller issues the higher fraction of correct transactions, and with the help of edge computing servers, computation tasks can be executed more quickly. This figure shows the convergence performance of dueling DQN as well. At the beginning of learning and training, dueling DQN takes some trials and errors. With the increase of episodes, the throughput turns to be stable, which means the agent has learned the optimal policies to maximize the long-term rewards.

In addition, Fig. 5 shows the relationship between the learning loss in (22) and the training steps of DuelingDQN-based scheme when the agent runs in the same parameters as above. At the beginning of learning, deep networks have no knowledge of the uncertain environment, and

TABLE IV: Parameters setting in the simulation.

Parameter	Value	Description
θ	8 Mcycles	The required number of CPU cycles to verify one signature.
α	0.05 Mcycles	The required number of CPU cycles to verify and generate one MAC.
b	1Mb	The batch size of a block.
γ	0.9	The discount factor.

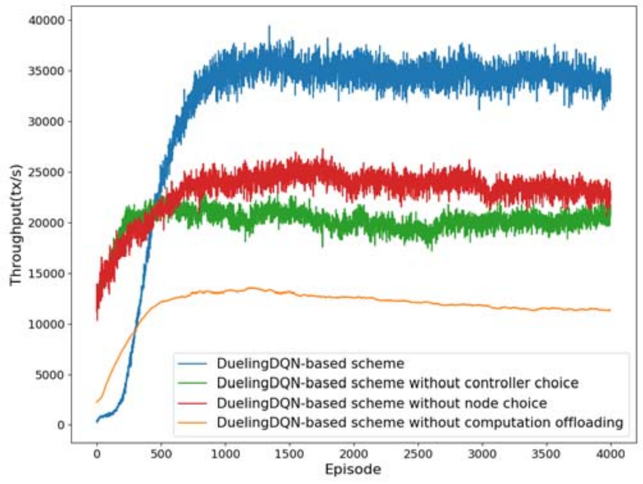

Fig. 4: Training curves tracking the throughput of the BC system under different schemes.

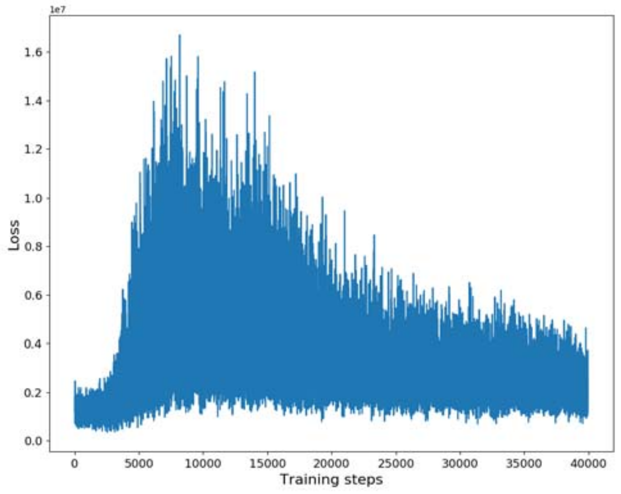

Fig. 5: Training curves tracking the learning loss under DuelingDQN-based scheme.

with the increase of new experiences, the learning loss is higher and higher. When the cyclic buffer of experiences in dueling DQN is full, the agent has some knowledge of the environment, which leads to the decrease of the learning loss. Such increasing and decreasing of the learning loss indicate the effectiveness of deep networks.

Fig. 6 shows the relationship between training episodes and the throughput of different learning rates under Dueling DQN-based scheme. As we can see from this figure, the learning rate has effects on the convergence performance. The learning rate means the length of learning step to minimize the loss function. The bigger learning rate denotes the longer learning step. The longer learning steps are likely to miss the global optimum, which leads to the highly scaled curves when learning rates are 0.01 and 0.001. The shorter learning steps may lead to the slower convergence speed, because more steps are necessary to achieve the global optimum. Compared with two curves of blue and orange, although the orange one has the faster convergence speed, its curve is unstable after the convergence. Therefore, we choose the learning rate as $1e^{-5}$ in the simulation. Because its convergence speed is acceptable and it has better learning stability.

Fig. 7 shows the learning loss of natural DQN and

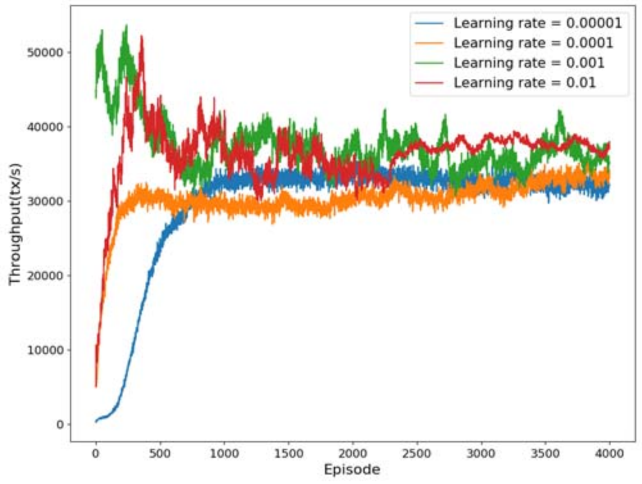

Fig. 6: Training curves tracking the throughput of the BC system under different learning rates.

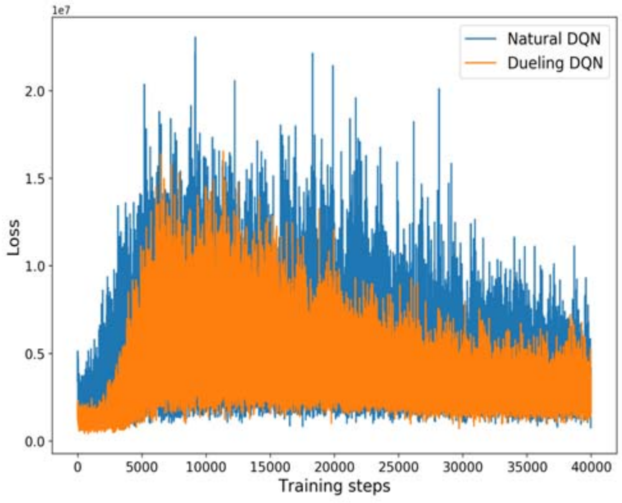

Fig. 7: Training curves tracking the learning loss of natural DQN and dueling DQN.

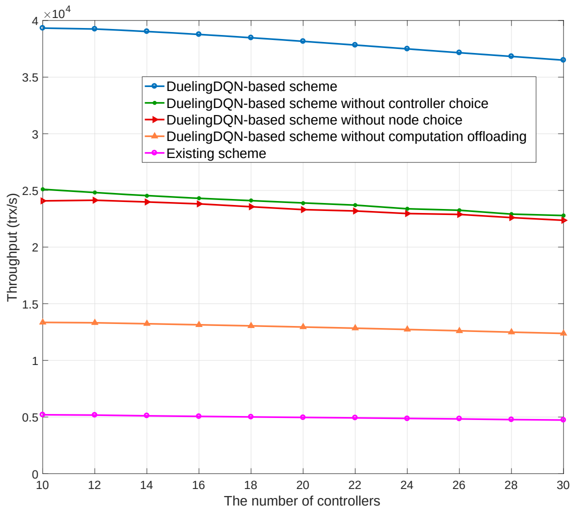

Fig. 8: The throughput versus the number of controllers under different schemes.

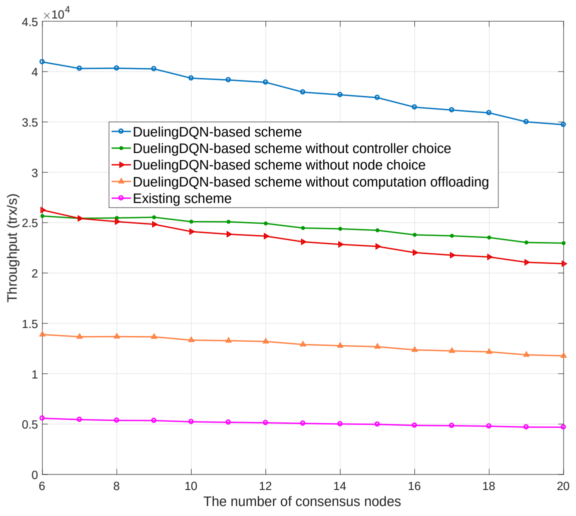

Fig. 9: The throughput versus the number of consensus nodes under different schemes.

dueling DQN. As we can see, the learning loss in dueling DQN decreases more quickly than natural DQN, which indicates dueling DQN has better learning effectiveness. The reason is that in our BC system, the choices of which node is the primary, which controller can access to the BC system, and which edge computing server should execute the computation tasks have no relationship with states. Learning which action has better consequences is more efficient than learning which state is better. In dueling DQN, one stream learns state-action value function $A(s,a)$, which is more useful to help the agent make the good choices. Therefore, the learning loss in dueling DQN decreases more fast than natural DQN.

After the effective training of deep networks, we use them in the following simulations. Fig. 8 shows the relationship between the number of controllers and the system throughput under different schemes. As the increase of the number of controllers, the system throughput decrease. The reason is that more controllers need more computational operations about verifying signatures, MACs. But with the joint consideration of trust features of controllers and nodes, as well as using edge computing servers, our proposed scheme, as shown in the blue curve, has better performance.

Fig. 9 shows the relationship between the number of nodes and the system throughput under different schemes. As we can see, more nodes lead to the less system throughput. The reason is that with the increase of the number of nodes, more signatures and MACs need to be verified and generated, which need more CPU cycles so as to decrease the system throughput. But, the performance of our proposed scheme is still the best.

Fig. 10 shows the relationship between the batch size of a block and the system throughput. The bigger block enables to contain more transactions so as to synchronize more local network events among controllers, which increases the system throughput. As we can see, our scheme also has the better performance.

VIII. Conclusions and Future Work

In this paper, we proposed a blockchain-based consensus protocol in distributed SDIIoT, along with detailed consen-

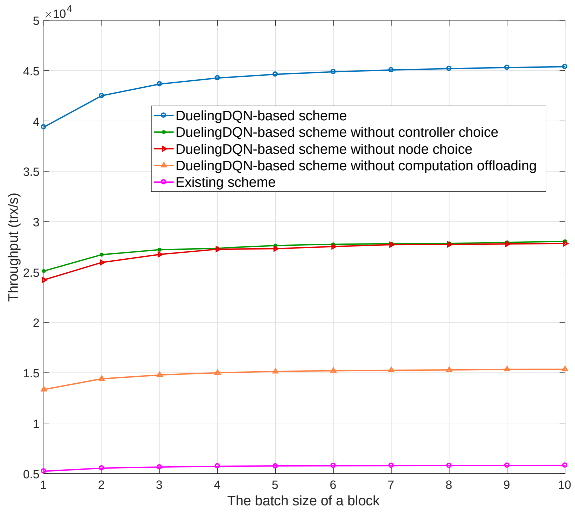

Fig. 10: The throughput versus the batch size of the blocks under different schemes.

sus steps and theoretical analysis. We jointly considered the trust features of nodes and controllers, as well as the computational capability of the system. We formulated view changes, access selection, and computational resources allocation as a joint optimization problem. Then we described it as a Markov decision process by defining state space, action space, and reward function. In addition, we proposed a novel dueling deep Q-learning approach to solve this problem. Simulation results showed the effectiveness and the convergence performance of our proposed scheme with different scenarios. How to measure the trust features of nodes and controllers in SDIIoT is very important. Some future works are in progress to solve this problem.

References

[1] J.-Q. Li, F. R. Yu, G. Deng, C. Luo, Z. Ming, and Q. Yan, "Industrial internet: A survey on the enabling technologies, applications, and challenges," *IEEE Comm. Surveys & Tutorials*, vol. 19, no. 3, pp. 1504–1526, 2017.

[2] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Comm. Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2015.

[3] J. Wan, S. Tang, Z. Shu, D. Li, S. Wang, M. Imran, and A. V. Vasilakos, "Software-defined industrial internet of things in the context of industry 4.0," *IEEE Sensors Journal*, vol. 16, no. 20, pp. 7373–7380, 2016.

[4] H. R. Faragardi, H. Fotohi, T. Nolte, and R. Rahmani, "A cost efficient design of a multi-sink multi-controller WSN in a smart factory," in *Proc. Conf. High Performance Comp. and Comm.*, 2017.

[5] K. Kaur, S. Garg, G. S. Aujla, N. Kumar, J. J. Rodrigues, and M. Guizani, "Edge computing in the industrial internet of things environment: Software-defined-networks-based edge-cloud interplay," *IEEE Comm. Mag.*, vol. 56, no. 2, pp. 44–51, 2018.

[6] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental flow scheduling and routing in time-sensitive software-defined networks," *IEEE Trans. on Industrial Informatics*, vol. 14, no. 5, pp. 2066–2075, 2018.

[7] M. Baddeley, R. Nejabati, G. Oikonomou, S. Gormus, M. Sooriyabandara, and D. Simeonidou, "Isolating SDN control traffic with layer-2 slicing in 6TiSCH industrial IoT networks," in *Proc. Conf. Net. Function Virtualization and Software Defined Net.' 17*, 2017, pp. 247–251.

[8] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[9] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Pro. Conf. Internet Netw. Manag.*, 2010, pp. 3–3.

[10] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proc. Conf. Hot Topics in Software Defined Net.*, 2012, pp. 19–24.

[11] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proc. Conf. Hot Topics in Software Defined Net.*, 2014, pp. 1–6.

[12] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain *et al.*, "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering," in *Proc. Conf. ACM SIGCOMM*, 2017, pp. 432–445.

[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[14] J. Wan, B. Chen, M. Imran, F. Tao, D. Li, C. Liu, and S. Ahmad, "Toward dynamic resources management for iot-based manufacturing," *IEEE Comm. Mag.*, vol. 56, no. 2, pp. 52–59, 2018.

[15] L. Silva, P. Gonçalves, R. Marau, P. Pedreiras, and L. Almeida, "Extending openflow with flexible time-triggered real-time communication services," in *Proc. Conf. Emerging Tech. and Factory Automation' 17*, 2017, pp. 1–8.

[16] M. Moness, A. M. Moustafa, A.-R. H. Muhammad, and A.-S. A. Younis, "Hybrid controller for a software-defined architecture of industrial internet lab-scale process," in *Proc. Conf. Comp. Engineering and Sys.' 17*, 2017, pp. 266–271.

[17] Q. Yan, W. Huang, X. Luo, Q. Gong, and F. R. Yu, "A multi-level DDoS mitigation framework for the industrial internet of things," *IEEE Comm. Mag.*, vol. 56, no. 2, pp. 30–36, 2018.

[18] Y. Duan, W. Li, X. Fu, Y. Luo, and L. Yang, "A methodology for reliability of WSN based on software defined network in adaptive industrial environment," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 1, pp. 74–82, 2018.

[19] Y. Duan, W. Li, Y. Zhong, and X. Fu, "A multi-network control framework based on industrial internet of things," in *Proc. Conf. Net., Sensing, and Control' 16*, 2016, pp. 1–5.

[20] S. Al-Rubaye, E. Kadhum, Q. Ni, and A. Anpalagan, "Industrial internet of things driven by SDN platform for smart grid resiliency," *IEEE Internet of Things Journal*, 2017.

[21] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy and challenges," *IEEE Commu. Surveys & Tutorials*, 2017.

[22] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *ACM SIGCOMM Comp. Commu. Review*, vol. 38, no. 3, pp. 105–110, 2008.

[23] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, vol. 1, p. 132, 2009.

[24] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "Elasticon; an elastic distributed sdn controller," in *Pro. Conf. Architectures for Net. and Commu. Sys.*, 2014, pp. 17–27.

[25] C. Qiu, C. Zhao, F. Xu, and T. Yang, "Sleeping mode of multi-controller in green software-defined networking," *EURASIP Journal on Wireless Commu. and Net.*, vol. 2016, no. 1, p. 282, 2016.

[26] A. Lei, H. Cruickshank, Y. Cao, P. Asuquo, C. P. A. Ogah, and Z. Sun, "Blockchain-based dynamic key management for heterogeneous intelligent transportation systems," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1832–1843, 2017.

[27] M. Singh and S. Kim, "Blockchain based intelligent vehicle data sharing framework," *arXiv preprint arXiv:1708.09721*, 2017.

[28] A. Dorri, M. Steger, S. S. Kanhere, and R. Jurdak, "Blockchain: A distributed solution to automotive security and privacy," *IEEE Comm. Maga.*, vol. 55, no. 12, pp. 119–125, 2017.

[29] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.

[30] C. Cachin and M. Vukolić, "Blockchains consensus protocols in the wild," *arXiv preprint arXiv:1707.01873*, 2017.

[31] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. on Comp. Sys.*, vol. 20, no. 4, pp. 398–461, 2002.

[32] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[33] C. Cachin, "Architecture of the Hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[34] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making byzantine fault tolerant systems tolerate byzantine faults." in *NSDI*, vol. 9, 2009, pp. 153–168.

[35] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[37] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[38] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[39] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project Adam: Building an efficient and scalable deep learning training system." in *OSDI*, vol. 14, 2014, pp. 571–582.