Blockchain-Based Edge Computing Resource Allocation in IoT: A Deep Reinforcement Learning Approach

Ying He, Yuhang Wang, Chao Qiu, Qiuzhen Lin, *Member, IEEE*, Jianqiang Li, *Member, IEEE*, and Zhong Ming, *Member, IEEE*

***Abstract*—With the exponential growth in the number of Internet-of-Things (IoT) devices, the cloud-centric computing paradigm can hardly meet the increasingly high requirements for low latency, high bandwidth, ease of availability, and more intelligent services. Therefore, a distributed and decentralized computing architecture is imperative, where edge-centric computing, such as fog computing and mist computing, has been recently proposed. Edge-centric computing resources can be managed locally and personally rather than being administered by a remote centralized third party. However, security and privacy issues are the main challenges due to the absence of trust between the IoT devices and edge computing nodes (ECNs). A blockchain, as a decentralized, trustless, and immutable public ledger, can well solve the trust-absence issue. In this article, we first elaborate on the security and privacy issues of edge-computing-enabled IoT, and then present the key characteristics of blockchains, which make blockchains well suited for the edge-centric IoT scenarios. Furthermore, we propose a general framework for blockchain-based edge-computing-enabled IoT scenarios that specifies the step-by-step procedure of a single transaction between an IoT end and an ECN. In addition, we design a smart contract within a private blockchain network that exploits the state-of-the-art machine learning algorithm, asynchronous advantage actor–critic (A3C), to allocate the edge computing resources, which exemplifies how artificial intelligence (AI) can be combined with blockchains. We further discuss the benefits of the convergence of AI and blockchains. Finally, simulation results are presented.**

***Index Terms*—Asynchronous advantage actor–critic (A3C) algorithm, blockchain, edge-centric computing, Internet of Things (IoT).**

I. Introduction

WITH the rapid development of Internet of Things (IoT), a growing number of physical devices are being connected at an unprecedented rate [1]. The connectivity of these IoT devices will definitely generate a large amount of data into networks, which requires massive computing resources, storage space, and communication bandwidth for data analytics. Big data and data analytics play a significant role in the advancement of IoT [2], which is usually implemented by the support of cloud computing. Unfortunately, despite the "unlimited" computational capacities, the cloud-centric computing paradigm exposes some issues, like high transmission cost, transmission congestions, and long service latency, which makes cloud computing infeasible in many IoT application scenarios that require real-time interactions or mobility [3], [4].

On the other hand, cloud-centric computing, as a type of centralized computing architecture, also exhibits some downsides [5]–[7]. The first primary issue is the loss of privacy, for instance, some sensitive IoT data, such as surveillance videos, locations, healthcare data, meter readings, etc., are released directly to cloud-based centralized services, such as storage services, location services, healthcare websites, etc. The second primary problem is the complete delegation of the applications from the clients to the clouds, which is only based on an unilateral trust. Third, centralization results in the missed opportunities of exploiting the enormous amount of computational resources, storage, and communication capability of the advanced personal devices. Finally, cloud-centric computing is vulnerable to single point of failure, and it is not scalable to deal with the hundreds of billions of data service requests generated by the exponentially increasing IoT devices [8].

Therefore, the trend has been recently geared toward a decentralized computing paradigm, where edge-centric computing, such as fog computing and mist computing, has been proposed to be utilized in IoT scenarios [9]–[11]. In edge-centric computing architecture, the computing resources, various applications and services are transferred from the centralized nodes to the periphery of the networks that is much closer to the end IoT devices, henceforth, low-latency, fast-response, and location-awareness services can be provided [12]–[14]. Furthermore, in this novel decentralized paradigm, the edge computing devices can be administered locally rather than being controlled by a third-party centralized cloud node [8], henceforth, various innovative, customized or availability-oriented applications and services can be more conveniently realized. Since IoT applications might require fast-response, low-latency and privacy-preserving services, thus edge-centric

Manuscript received April 30, 2020; revised July 22, 2020 and October 9, 2020; accepted October 16, 2020. Date of publication November 3, 2020; date of current version February 4, 2021. This work was supported in part by the Natural Science Foundation of China under Grant 62002238, Grant 61836005, and Grant 61976142; and in part by Foundation for Distinguished Young Talents in Higher Education of Guangdong, China, under Grant 2019KQNCX125. *(Corresponding author: Jianqiang Li.)*

Ying He, Yuhang Wang, Qiuzhen Lin, Jianqiang Li, and Zhong Ming are with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: lijq@szu.edu.cn; mingz@szu.edu.cn).

Chao Qiu is with the College of Intelligence and Computing, School of Computer Science and Technology, Tianjin University, Tianjin 300350, China.

Digital Object Identifier 10.1109/JIOT.2020.3035437

computing is preferably well suited for handling with IoT data. It might be noted that personal advanced IoT devices can also play the roles of edge computing nodes (ECNs), and share their resources as a service with other IoT devices.

However, security and privacy are two primary challenges for IoT nodes when making transactions with ECNs due to the lack of trust relationships. The most recent advanced technology, blockchains, as a trustless and immutable public ledger can be an appropriate solution to tackle this trust-absence problem. The distributed consensus mechanism and the key management system in blockchains help facilitate the sharing of resources and services, and make the transactions being processed in a cryptographically verifiable manner. Essentially, the transactions are secured because they are trackable and irreversible.

Blockchain technology has been developing rapidly in the recent years, especially in IoT scenarios [15]–[17]. One well-known blockchain app platform, Ethereum, implements a nearly Turing-complete language on its blockchain, which means any program codes designed by clients are allowed to be executed on blockchains in an automatic and autonomous manner. The autonomy is achieved by smart contract enabled by blockchains. Contract clauses embedded in smart contracts will be executed automatically when a certain condition is satisfied. Started originally as an Ethereum fork, Monax Industry provides a customizable "Blockchain as a service," allowing clients to define arbitrary rules for their own protocol by writing smart contracts that will be executed on a private blockchain network [18].

In this article, the main contributions include as follows.

1) We first elaborate on the security and privacy issues of edge computing-enabled IoT, and then present the key characteristics of blockchains that allow blockchains well suited for the edge-centric IoT scenarios.
2) A general framework for blockchain-based edge computing-enabled IoT scenarios is proposed, where the procedure of a single transaction is specified step by step from the beginning of issuing a data service request by an IoT node to how finally the service is completed and recorded on the blockchains.
3) We design a smart contract within a private blockchain network, which exploits the state-of-the-art machine learning algorithm, asynchronous advantage actor–critic (A3C), to solve the edge computing resources allocation problem. Particularly, different from the existing relevant schemes, this proposed scheme can distinguish different Quality of Service (QoS) requirements from multiple service subscribers, which leads to higher efficiency. This exemplifies how artificial intelligence (AI) can be combined with blockchains, and later we further discuss the benefits of the convergence of AI and blockchains.
4) Simulation results are finally presented to show the effectiveness of utilizing the A3C algorithm to solve the edge computing resource allocation problem.

The remainder of this article is organized as follows. In Section II, the background is presented where security and privacy issues of edge computing-enabled IoT are mainly discussed. Section III illustrate the proposed framework and the system model. In Section IV, how the smart contract is designed to allocate the edge computing resources is formulated as a deep reinforcement learning problem. Section V presents the solution with the A3C algorithm. Simulation results are discussed in Section VI. Finally, conclusions and future works are presented in Section VII.

II. Proposed Framework and System Model

In this section, the system architecture is first described. A framework of blockchain-based edge computing in IoT environment is presented in the second part. Finally, the system model is discussed.

A. System Architecture

In this article, the system architecture is presented in Fig. 1, which consists of four layers: 1) the IoT device layer; 2) the subscriber layer; 3) the edge computing layer; and 4) the cloud layer. In the bottom layer, the IoT devices, such as various smartphones, smart cars, sensors, laptops, smart homes, etc., monitor and gather raw data from the surrounding environments. In the subscriber layer, each data service subscriber (DSS) possesses and controls several IoT devices, and sends the filtered raw data to the edge computing layer for computing so as to do data analysis or preprocessing in a timely manner. If required, the output data after being processed by the edge computing can be sent back to DSSs or delivered to the cloud for further data analysis or long-term storage. Here, we consider that the DSSs apply real-time interactive applications, i.e., the data analysis results should be returned to the DSSs timely. For the edge computing layer, there exists a set of ECNs, each of which is formed by multiple low-power computing resources and can provide data computing services. Specifically, the ECNs can be enacted by any IoT devices with surplus computing resources. For any distributed computing system, scheduling plays an important role due to the fact that the performance of the applications depends primarily on its efficiency [19]. Therefore, when a DSS submits a data service request, a most appropriate ECN should be assigned to offer the computing service for the DSS based on the current system state (e.g., each ECN's running workload) and the DSS's QoS requirement (e.g., the shortest service delay). Unexpectedly, if the ECNs have insufficient computing resources to process the input data, they can offload the incoming data to the cloud at the cost of the increased latency in the response time and extra consumption in the network resources.

B. Framework of Blockchain-Based Edge Computing in IoT Environments

In this part, we present a framework that specifies how the transactions are handled from the IoT devices to the ECNs via blockchain.

We assume a specific kind of crypto token, just like Ether in the Ethereum, circulating under this framework. The DSSs who create computing requests pay the crypto tokens, and on the other hand, ECNs that contribute the computing resources and bandwidth can earn the tokens in the form of fees from

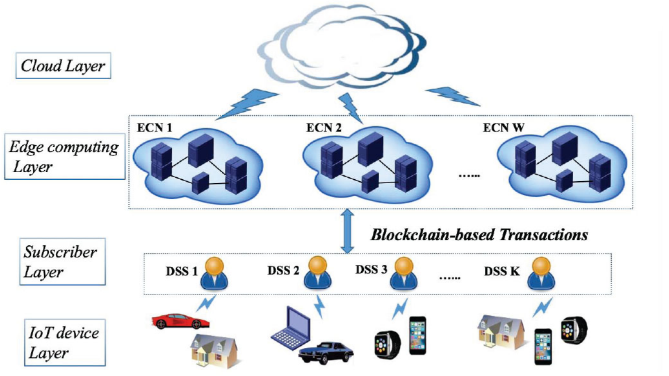


Fig. 1. Overview of the system architecture.

the DSSs. Any transfer of the crypto token in the system can be considered to have been confirmed with the same security as the underlying proof of work or proof of stake blockchain. Additionally, consensus in this framework is also provided by the underlying blockchain platform, for example Ethereum. Distributing newly minted tokens is an extra manner to encourage participation in the computing jobs and verification jobs. Since we focus on the computing part of this framework in this article, the initial allocation of the crypto tokens and additional tokens issued according to a certain inflation rate will not be discussed in detail.

The core unit of the IoT streaming data is termed as a segment, that is to say, the streaming data collected from the IoT devices is divided into segments. A sequential number is created for each segment that identifies their proper ordering [20]. Specifically, hash of the data payload of each segment should also be created, which can be used to attest the data integrity. For this framework, the blockchain uses the segments as the unit of work for computing and payment.

The entire procedure of a specific transaction is depicted in Fig. 2, where we consider the scenario that DSS K submits a computing request onto the blockchain. The procedure is discussed as follows.

1) DSS K creates a computing request onto the blockchain and deposits some tokens in escrow as well.
2) Upon receiving the request, the smart contract assigns a specific ECN w to serve for DSS k. The principle of selecting a node for computing services varies from each scenario. For example, in [21] the nodes for performing the transcoding jobs are selected based on delegated proof of stake, i.e., the nodes with the most cumulative stakes, including both their own and the stakes delegated from other nodes can finally play the computing role. In this article, we use the deep reinforcement learning algorithm to do with the ECN selection.
3) Assume that ECN w is finally assigned. After that ECN w sends a notice of reception of the computing job, and then the IoT devices of DSS k can deliver the segments to ECN w for computing, which contains the signature verifying the input data from DSS k.
4) Once the computation task is fulfilled, the ECN sends a claim of completing the computing job to the blockchain, and meanwhile appends the computing receipt data as a transaction to the Merkle tree in one block. The computing receipt would include the hash of the input segment data, the sequential number of this segment data, the hash of the output data, the signature of the input data from DSS k, and the signature of output data from ECN w as well. Afterwards, DSS k invokes the verification process of its computing work.
5) At the same time, ECN w is required to deliver the input segment data payload to the storage platform for the subsequential verification, and makes sure that the data is stored before the start of the verification process, and it is maintained long enough for the whole process.
6) The verifier that works based on certain protocol will extract the data from the storage platform to verify the correctness of the computing job done by ECN w. Usually, to alleviate the burden of verification computing, only a small percentage of the segments are chosed to be challenged. If the work has been approved to be correct, the verifier will send a notice of the verification to the smart contract. Accordingly, ECN w will earn some crypto tokens from DSS k's account for completing the computing job. However, if the result is incorrect, ECN w would get certain punishement and no earnings, the process repeats once more, and the computing job will be done by some other ECN.
7) Ultimately, this verifiable transaction will be recorded on the blockchain.

In the following, we will amplify and specify the second step in the above procedure, that is to say, how to assign the ECNs to the requesting users.

C. System Model

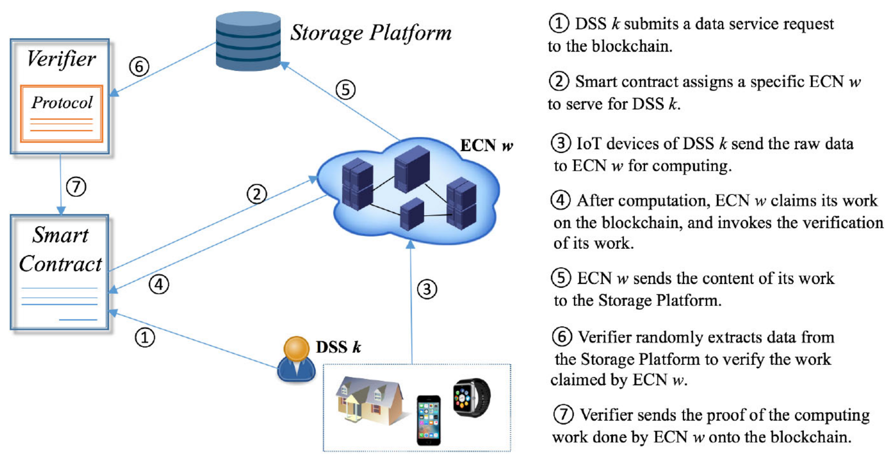
Assume that there are in total K DSSs labeled as d_k where $k \in \{1, 2, \dots, K\}$ and W ECNs given by f_w where $w \in \{1, 2, \dots, W\}$. Different DSSs have different criteria for computing, which can be measured by the service delay time. For example, some DSSs prefer the minimal service delay time even if they cost more, whereas other DSSs may require to minimize the cost even if the computation takes longer. To satisfy the requirments of the service quality of DSS d_k, a threshold for the service delay time is required and denoted as τ_k^{th}. That is to say, the total delay time for serving one segment coming from DSS d_k, denoted as t_k, should meet the requirement $t_k \leq \tau_k^{\text{th}}$.

In this article, we consider that the total delay time for serving one segment from DSS d_k is composed of two elements, i.e.,

$$t_k = t_k^{\text{net}} + t_k^{\text{comp}} \tag{1}$$

where the first element t_k^{net} represents the network delay caused by the wireless transmission from the IoT devices to the designated ECN and from the ECN back to the DSS d_k. With regard to the second element t_k^{comp}, which indicates the sojourn time of one segment spending at the ECN system, i.e., the total delay of both the waiting time and the service time.

We will discuss the above two components of the service delay time in the following, respectively.



① DSS *k* submits a data service request to the blockchain.

② Smart contract assigns a specific ECN *w* to serve for DSS *k*.

③ IoT devices of DSS *k* send the raw data to ECN *w* for computing.

④ After computation, ECN *w* claims its work on the blockchain, and invokes the verification of its work.

⑤ ECN *w* sends the content of its work to the Storage Platform.

⑥ Verifier randomly extracts data from the Storage Platform to verify the work claimed by ECN *w*.

⑦ Verifier sends the proof of the computing work done by ECN *w* onto the blockchain.

Fig. 2. Procedure of one specific transaction.

1) Network Delay Model: As a usual, DSSs are closely located with their IoT devices, therefore, without loss of generality, we assume that IoT devices of each DSS are located at the same position with the DSS. Mostly the size of the data needed to be transmitted from ECNs back to DSSs is small, thus this part of the network delay is omitted, and we only focus on the network delay caused by delivering the raw segment from IoT devices to ECNs. In fact, the network delay is affected by many factors, such as the transmission distance, the wireless channel conditions, and some other unpredicted causes. Practically, the network delay can be evaluated from the training data periodically sent from the IoT devices to the ECNs [12]. Here, as our previous work [22], we consider the wireless channels between the ECNs and the DSSs are realistic time-varying channels, and they are modeled as finite-state Markov channels (FSMCs). The received SNR is modeled as a random variable γ_k^w, and the transition follows a Markov process. For details, please refer to Section III-B in [22]. Therefore, the network delay t_k^{net} can be expressed as $o_k/b_k \log(1+\gamma_k^w)$ following Shannon equation, where o_k represents data size for one segment, and b_k is the bandwidth.

2) Edge Computing Delay Model: Suppose that the data segments from the IoT devices of DSS d_k arrive according to a Poisson distribution with the rate of λ_k, $k = 1, 2, \ldots, K$. For the edge computing layer, it is assumed that the computing capability of each ECN is different, and we suppose that ECN f_w can provide computing service at an average service rate of μ_w.

Here, we assume that one data segment from DSS d_k is assigned to be served by ECN f_w based on the smart contract. The edge computing delay for serving the segment can be divided into two parts: 1) the waiting time in the queue of ECN f_w and 2) the computing time in the server, denoted as D_q^w and D_t^w, respectively. Therefore, the total edge computing delay time t_k^{comp} can be expressed as

$$t_k^{\text{comp}} = D_q^w + D_t^w. \tag{2}$$

The average computing time of one data segment in the server of ECN f_w with one CPU can be calculated by one over the average service rate, namely, $1/\mu_w$. Then, expression (2) can be updated as

$$t_k^{\text{comp}} = D_q^w + \frac{1}{\mu_w}. \tag{3}$$

The queueing delay also involves two parts: 1) the remaining processing time of the current computing task in the server and 2) the sum of the computing time of all the segments in the queue. Then, (3) can be written as

$$t_k^{\text{comp}} = D_r^w + \frac{n_w}{\mu_w} \tag{4}$$

where D_r^w represents the remaining processing time of the segment being in the server and n_w stands for the number of segments in the queue before the newly arrived segement from DSS d_k, which is actually dynamic over time. Since we consider the first-come, first-served discipline that means each ECN computes one segment at a time from the front of the queue. After the computing service is completed, the result data will be delivered back to the DSSs or the cloud, and the number of the segments in this ECN queue reduces by one.

For simplicity, we use the average remaining processing time to approximate D_r^w, which can be expressed as [23]

$$D_r^w = \frac{1}{2}\overline{\lambda}\frac{1}{{\mu_w}^2} \tag{5}$$

where $\overline{\lambda}$ stands for the average arrival rate of ECN f_w. Therefore, the total delay time of edge computing can be finalized as

$$t_k^{\text{comp}} = \frac{1}{2}\overline{\lambda}\frac{1}{{\mu_w}^2} + \frac{n_w}{\mu_w}. \tag{6}$$

Until now, the expression of the total delay of serving one segment from DSS d_k by ECN f_w can be shown as follows:

$$t_k = \frac{o_k}{b_k \log\left(1+\gamma_k^w\right)} + \frac{1}{2}\overline{\lambda}\frac{1}{{\mu_w}^2} + \frac{n_w}{\mu_w} \leq \tau_k^{\text{th}}. \tag{7}$$

Note that the time spent on the blockchain as well as in the verification process is not included in the total delay time. Since we consider the real-time interactive edge computing applications, the computing result should be sent immediately back to the ECN f_w or the cloud for storage once the computation is completed. In the meantime, the computing result is delivered to the storage platform waiting for verification. Later on, if the result can successfully pass through the verification process, ECN f_w can earn the tokens previously escrowed on the blockchain by the DSS d_k, and the transacton is recorded onto the blockchain. In case the computing result fails the verfication, the ECN f_w cannot gain any tokens, instead, it gets slashed a certain percentage of its guarantee deposit, and the DSS d_k is refunded. However, the probability of verification failure is relatively low, this is because for ECNs malicious behavior will not bring any benefits rather than slashed deposit and damaged reputation.

III. Problem Formulation

In this section, we focus on the problem of how the smart contract would allocate the computing resources to the segments, that is to say, when a segment from a DSS newly arrives, which ECN should the segment join so as to satisfy the DSS's requirement and maximize the ECNs' revenue? Or should it be blocked or serviced by the remote cloud? We aim at serving as more segments as possible, and maxmizing the total revenue of ECNs.

We formulate this problem as a continuous-time Markov decision process. The existing works on continuous-time Markov decision processes can be categorized into two types: the first type deals with continuous-time jump Markov decision processes wherein the decisions are only performed at certain time instants, and this can be ascribed into semi-Markov or even discrete-time Markov decision processes; in the second type studies, the decision maker can choose actions continuously in time [24]. In this article, we focus on the semi-Markov decision process (SMDP), under which the decisions are uniformly referred as actions, and the time instants that an action is determined are called decision epochs.

A. System State

For the network of W ECNs, the state of the system can be represented by a W-dimensional vector

$$\mathbf{x}(t) = [n_1(t), n_2(t), \ldots, n_w(t), \ldots, n_W(t)] \in \mathbf{S} \tag{8}$$

where $n_w(t)$ denotes the number of segments presently being in the queue of the wth ECN at the decision epoch t. $\mathbf{S}$ is the state space. We assume that the buffer size of each ECN is infinite, i.e., there is no limit on the number of segments in the queue of each ECN, however, since the computing for the segments has a delay requirement from the DSSs, thus actually each queue has an upperbound of the number of segments.

In our scenario, the system state evolves as a birth/death process that there are two independent exponential clocks running: 1) for the current state $\mathbf{x}$ at decision epoch t, generally one that will take the system into another state $\mathbf{x}+\mathbf{e}_w$ if a segment from one DSS newly arrives and visits the wth ECN for computing and 2) the other which will evolve the system into state $\mathbf{x}-\mathbf{e}_w$ if the wth ECN releases the computing resource. Here, $\mathbf{e}_w$ is unit W-dimensional vector, which represents an increase or decrease of the number of segments by 1 in the corresponding wth ECN, respectively.

B. System Action

When a new segment arrives, the smart contract makes a scheduling decision. For the SMDP, the natural decision epochs should be the segments' arrival points. However, each time when an ECN completes the computing job, the state of the system also changes. Therefore, similar to [25], we make the decision epochs to be the set of all segment arrival and computing resource release time instants. Let us denote that $t_0 = 0$, and the decision epochs are taken to be the instances $t_m, m = 0, 1, 2, \ldots$ At each decision epoch, the smart contract makes an action for each DSS's segment that which ECN should this request visit. Supposing that at this decision epoch, only computing resource release happens without any new arrivals, actually no actions should be performed. We can define the action $\mathbf{a}(t_m)$ at decision epoch t_m as

$$\mathbf{a}(t_m) = [\mathbf{a}_1(t_m), \mathbf{a}_2(t_m), \ldots, \mathbf{a}_W(t_m)] \tag{9}$$

where the wth element can be extended as

$$\begin{aligned}\mathbf{a}_w(t_m) = \big[a_{w,1}(t_m), a_{w,2}(t_m), \ldots, a_{w,K}(t_m)\big]\\ \forall w \in \{1, 2, \ldots, W\}\end{aligned} \tag{10}$$

where $a_{w,j}(t_m) \in \{0, 1\}\ \forall j = 1, 2, \ldots, K$, denotes the action of the wth ECN for a new segment from the jth DSS at decision epoch t_m. If $a_{w,j}(t_m) = 1$, the new segment that arrives at decision epoch t_m is admitted to be serviced at the wth ECN. Conversely, if $a_{w,j}(t_m) = 0$, it will not be served by the wth ECN.

Since DSS d_j has a total delay requirement τ_j^{th}, the sum time of the network delay and the computing service delay at the assigned ECN cannot surpass this threshold as we discussed in Section III-C. For example, for the wth ECN, the following constraint should be satisfied when a segment from DSS d_j newly arrives:

$$a_{w,j}(t_m) \neq 1,\ \text{if}\ \frac{o_k}{b_k \log\big(1+\gamma_k^w\big)} + \frac{1}{2}\bar{\lambda}\frac{1}{{\mu_w}^2} + \frac{n_w}{\mu_w} > \tau_j^{\text{th}}. \tag{11}$$

Here, for the system action we restrict that $\sum_{w=1}^{W} a_{w,j}(t_m) \leq 1$, that is to say, at one decision epoch, at most one ECN is allowed to provide computing service to a newly arrived segment. Provided that all the ECNs cannot satisfy the jth DSS's requirement, the actions can only be $a_{w,j}(t_m) = 0\ \forall w = 1, 2, \ldots, W$. However, when the system state is $\mathbf{x} = [0, 0, \ldots, 0]$, the specific action $\mathbf{a}(t_m) = [\mathbf{0}, \mathbf{0}, \ldots, \mathbf{0}]$ should not be the only possible action, otherwise, new segments are never admitted to visit the ECNs and the system cannot evolve.

Based on the above analysis, the action space $\mathbf{A}_\mathbf{x}$ of a given state $\mathbf{x}$ is defined as

$$\begin{aligned}\mathbf{A}_\mathbf{x} = \big\{\mathbf{a} \in \mathbf{A} : a_{w,j}(t_m) \neq 1 \\ \text{if}\ \frac{o_k}{b_k \log\big(1+\gamma_k^w\big)} + \frac{1}{2}\bar{\lambda}\frac{1}{{\mu_w}^2} + \frac{n_w}{\mu_w} > \tau_j^{\text{th}} \\ \sum_{w=1}^{W} a_{w,j}(t_m) \leq 1 \\ \text{and}\quad \mathbf{a}(t_m) \neq [\mathbf{0}, \mathbf{0}, \ldots, \mathbf{0}]\ \text{if}\ \mathbf{x} = [0, 0, \ldots, 0]\end{aligned}$$

$$\forall w \in 1, 2, \ldots, W$$
$$\forall j \in 1, 2, \ldots, K\big\}. \tag{12}$$

C. State Dynamics

The state dynamics of the system can be characterized by the state transition probabilities of the embedded Markov chain. Every continuous-time Markov chain has an associated embedded Markov chain (also known as a jump process), which can help find the transition probability. The transition probability can be written as

$$p_{\mathbf{xy}}(\mathbf{a}) \triangleq \mathbf{P}(\mathbf{x}(t_{k+1}) = \mathbf{y}|\mathbf{x}(t_k) = \mathbf{x}, \mathbf{a}(t_k) = \mathbf{a}) \tag{13}$$

which stands for the one-step probability of going from the current state $\mathbf{x}$ at decision epoch t_k to the next state $\mathbf{y}$ at decision epoch t_{k+1} when action $\mathbf{a}$ is taken. A transition matrix, formed by all possible $p_{\mathbf{xy}}$, can describe the probabilities of particular transitions and an initial state across the state space. However, although the transition matrix determines the probabilistic behavior of the embedded Markov chain, it cannot fully capture the behavior of the continuous-time Markov chain because it lacks the specification of the rates at which the transitions occur.

A transition rate matrix, whose entries $q_{\mathbf{xy}}$ $\forall \mathbf{x}, \mathbf{y} \in \mathbf{S}$ are nonnegative, describes the rates of the transition process from one state to another jump state. The entries that represent the process standing still without any jumps (i.e., elements $q_{\mathbf{xx}}$ $\forall \mathbf{x} \in \mathbf{S}$) are filled to make each row of the transition rate matrix sum up to zero. Note that $q_{\mathbf{xy}}$ $\forall \mathbf{x}, \mathbf{y} \in \mathbf{S}$ are rates, not probabilities, while they must be nonnegative, they are not bounded by 1. Based on the transition rate matrix, the transition probability can be obtained. When the system enters into a state $\mathbf{x}$, the length of time it spends at state $\mathbf{x}$ is exponentially distributed with rate $\sum_{\mathbf{x}\neq\mathbf{y}} q_{\mathbf{xy}}$ and when it leaves state $\mathbf{x}$ it will go to state $\mathbf{y}$ with probability

$$p_{\mathbf{xy}} = \frac{q_{\mathbf{xy}}}{\sum_{\mathbf{x}\neq\mathbf{y}} q_{\mathbf{xy}}}, \quad \mathbf{x} \neq \mathbf{y}. \tag{14}$$

The above equation indicates that an event of certain transition occurs with a probability equal to the ratio between the rate of that particular transition and the total cumulative transition rate. Therefore, for our formulated problem, the transition probabilities of the embedded Markov chain are

$$p_{\mathbf{xy}}(\mathbf{a}(t_m)) = \begin{cases} \dfrac{\sum_{j=1}^{K} a_{w,j}(t_m)\lambda_j}{\sum_{w=1}^{W}\left[\sum_{j=1}^{K} a_{w,j}(t_m)\lambda_j + \mu_w\right]} \\ \text{if } \mathbf{y} = \mathbf{x} + \mathbf{e}_w \\ \dfrac{\mu_w}{\sum_{w=1}^{W}\left[\sum_{j=1}^{K} a_{w,j}(t_m)\lambda_j + \mu_w\right]} \\ \text{if } \mathbf{y} = \mathbf{x} - \mathbf{e}_w \end{cases} \tag{15}$$

where $\forall w \in 1, 2, \ldots, W$.

D. Policy and Reward Function

For each given state $\mathbf{x} \in \mathbf{S}$, an action $\mathbf{a} \in \mathbf{A_x}$ is chosen according to a policy $\pi_{\mathbf{x}} \in \Pi$, where Π is a set of admissible policies defined as

$$\Pi = \{\pi : \mathbf{x} \to \mathbf{A}|\pi_{\mathbf{x}} \in \mathbf{A_x} \quad \forall \mathbf{x} \in \mathbf{S}\}. \tag{16}$$

In this article, we consider the average comprehensive revenue (in the form of tokens) of all the ECNs as our system's reward. Assume that a segment from DSS d_k is assigned to be served by ECN f_w, of which the obtained average income R_{f_w} will be

$$R_{f_w} = \left(1 - p_b^w\right)I_k - p_b^w \varepsilon_s F_o \tag{17}$$

where p_b^w represents the probability that ECN f_w fails in accomplishing the computation task and cannot pass through the verification, and it will get punished by slashing a certain percentage ε_s of its guarantee deposit F_o. I_k is the amount of tokens ECN f_w can earn in the form of the fee from DSS d_k with a probability of $(1 - p_b^w)$.

On the other hand, to undertake the computation task, ECN f_w has to consume energy, and pays the protocol fees to the blockchain. Therefore, the cost of ECN f_w can be expressed as

$$C_{f_w} = e_c^w \frac{1}{\mu_w} P_c^w + F_p \tag{18}$$

where the unit price for the energy consumption at ECN f_w is defined as e_c^w per Joule, the processing time for computing one segment can be seen as $1/\mu_w$ (second/segment), and P_c^w represents the consumed energy in unit time (Joule/second). F_p is the protocol fee paid to the blockchain, and actually this amount of tokens is transferred to a virtual machine node that works as a substitute of the blockchain to execute the complex smart contract.

Based on the above description, the system reward function can be shown as

$$\begin{aligned} r(t_m; \mathbf{x}, \mathbf{a}) &= \sum_{j=1}^{K}\sum_{w=1}^{W} a_{w,j}(t_m)\left[R_{f_w}(t_m) - C_{f_w}(t_m)\right] \\ &= \sum_{j=1}^{K}\sum_{w=1}^{W} a_{w,j}(t_m)\Big[\left(1 - p_b^w\right)I_k - p_b^w \varepsilon_s F_o \\ &\qquad - e_c^w \frac{1}{\mu_w} P_c^w - F_p\Big] \end{aligned} \tag{19}$$

which is an immediate reward at the decision epoch t_m, namely, ECNs would get $R(\mathbf{x}, \mathbf{a}, t_m)$ tokens in state $\mathbf{x}$ when action $\mathbf{a}$ is performed. Since we consider the long-run comprehensive revenue of ECNs, we aim at finding an optimal policy π to maximize the cumulative reward as

$$R_\pi = \sum_{m=0}^{T} \gamma^m r(t_m) \tag{20}$$

where γ is the discount factor, and T-step future is considered.

IV. Solution With A3C Algorithm

In this article, we consider a reinforcement learning-based approach to derive the optimal policy for the above formulated SMDP problem. Specifically, we used the A3C algorithm [26], which is a state-of-the-art actor–critic RL algorithm, to train the learning agents so as to obtain the optimal policy. A3C algorithm was recently released by GoogleMind and it has successfully been utilized to train the neural networks in many applications [27], [28]. A3C and the underlying framework of this algorithm is based on the traditional actor–critic algorithm. Unlike the well-known deep Q-learning algorithm [29] that is a value-based RL, A3C algorithm is essentially a policy gradient method. A3C directly outputs the explicit policy rather than giving out the indirect value function as a suggestion, which has proven to be faster, simpler, more robust and to perform better if tuned well. On top of all that, it can work in scenarios with both continuous and discrete action spaces. Therefore, it is very promising in handling with very complex problems.

A. Policy Gradient Training

The actor–critic algorithm involves two neural networks: 1) the actor network with parameter θ_π and 2) the critic network with parameter θ_v. The learning agent interacts with the environment and uses the observations to train these two neural networks so as to obtain the optimal policy that can maximize its reward. The role of the *actor* network is to define a parameterized policy, which takes the system state as an input, and generates the probabilities for each possible action through the neural network with parameter θ_π. Essentially, the policy, denoted as $\pi(\mathbf{a}|\mathbf{x};\theta_\pi)$, is the conditional probabilities of performing a certain action at a given state with a specific neural network parameter. The training is dedicated to iteratively update the parameter θ_π to make the output policy optimal for the expected rewards.

The trajectory τ of a SMDP can be denoted by a sequence $\mathbf{x}_0, \mathbf{a}_0, r_1, \mathbf{x}_1, \mathbf{a}_1, r_2, \ldots, \mathbf{x}_{T-1}, \mathbf{a}_{T-1}, r_T$, where r_{t+1} represents the obtained reward when the system state transfers from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$ by taking action $\mathbf{a}_t$. T indicates the terminal step or time which is prescribed in advance. We consider the discounted cumulative reward, denoted by $R_\tau = r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^{T-1} r_T$, where γ is the discount factor and $0 \leq \gamma \leq 1$. We aim at maximizing the expected discounted cumulative reward $\mathbb{E}[R_\tau|\pi;\theta_\pi]$, and thus finding how to shift the probability distribution $\pi(\mathbf{a}|\mathbf{x};\theta_\pi)$ through its parameter θ_π becomes a key point to increase the reward.

The idea of the policy gradient method is to estimate the gradient of the expected discounted cumulative reward $\mathbb{E}[R_\tau|\pi;\theta_\pi]$ with respect to parameter θ_π by observing the trajectories following that policy π. The gradient can be derived as

$$\begin{aligned}\nabla_{\theta_\pi}\mathbb{E}[R_\tau|\pi;\theta_\pi] &= \nabla_{\theta_\pi}\sum \pi(\mathbf{a}|\mathbf{x};\theta_\pi)R_\tau(\mathbf{x},\mathbf{a})\\ &= \sum \nabla_{\theta_\pi}\pi(\mathbf{a}|\mathbf{x};\theta_\pi)R_\tau(\mathbf{x},\mathbf{a})\\ &= \sum \pi(\mathbf{a}|\mathbf{x};\theta_\pi)\frac{\nabla_{\theta_\pi}\pi(\mathbf{a}|\mathbf{x};\theta_\pi)}{\pi(\mathbf{a}|\mathbf{x};\theta_\pi)}R_\tau(\mathbf{x},\mathbf{a})\\ &= \sum \pi(\mathbf{a}|\mathbf{x};\theta_\pi)\nabla_{\theta_\pi}\log\pi(\mathbf{a}|\mathbf{x};\theta_\pi)R_\tau(\mathbf{x},\mathbf{a})\\ &= \mathbb{E}\big[R_\tau(\mathbf{x},\mathbf{a})\nabla_{\theta_\pi}\log\pi(\mathbf{a}|\mathbf{x};\theta_\pi)\big]\\ &= \sum_{t=0}^{T-1} R_t(\mathbf{x}_t,\mathbf{a}_t)\nabla_{\theta_\pi}\log\pi(\mathbf{a}_t|\mathbf{x}_t;\theta_\pi) \qquad (21)\end{aligned}$$

where the first step is the definition of expectation; the second step swaps the sum and the gradient; in the third step, simultaneously multiply and divide by $\pi(\mathbf{a}|\mathbf{x};\theta_\pi)$; the fourth step uses the proposition $\nabla_\theta \log z = (1/z)\nabla_\theta(z)$; for the last two steps, the derivation is also based on the definition of expectation. The update of the parameter θ_π for the actor neural network can be expressed as

$$\theta_\pi \leftarrow \theta_\pi + \alpha_\pi \sum_{t=0}^{T-1} R_t(\mathbf{x}_t,\mathbf{a}_t)\nabla_{\theta_\pi}\log\pi(\mathbf{a}_t|\mathbf{x}_t;\theta_\pi) \qquad (22)$$

where α_π is the learning rate for training the actor neural network.

The role of the *critic* network is to estimate the state-value function, denoted by $V_\pi(\mathbf{x};\theta_v)$, which is the expected sum of rewards when starting at the state x following the policy π, and it can be expressed as

$$V_\pi(\mathbf{x};\theta) = \mathbb{E}[R_\tau|\mathbf{x}_0 = \mathbf{x};\pi]. \qquad (23)$$

The state-action value function, denoted by $Q_\pi(\mathbf{x},\mathbf{a};\theta_v)$, represents the expected sum of rewards when starting in state $\mathbf{x}$, taking action $\mathbf{a}$ and from then on following the policy π. Thus

$$Q_\pi(\mathbf{x},\mathbf{a};\theta_v) = \mathbb{E}[R_\tau|\mathbf{x}_0 = \mathbf{x};\mathbf{a}_0 = \mathbf{a};\pi]. \qquad (24)$$

The advantage function, denoted by $A_\pi(\mathbf{x},\mathbf{a};\theta)$, is defined as

$$A_\pi(\mathbf{x},\mathbf{a};\theta_v) = Q_\pi(\mathbf{x},\mathbf{a};\theta_v) - V_\pi(\mathbf{x};\theta_v) \qquad (25)$$

which represents the difference of the expected sum of rewards when we specifically choose the action $\mathbf{a}$ in the state $\mathbf{x}$, compared with the expected sum of rewards when we draw the action from the policy π. The advantage funtion is used instead of the expected discounted rewards $\mathbb{E}[R_\tau|\pi;\theta_\pi]$ in the computation of the gradient, and the update of parameter θ_π should be revised as

$$\theta_\pi \leftarrow \theta_\pi + \alpha \sum_{t=0}^{T-1} A_\pi(\mathbf{x}_t,\mathbf{a}_t;\theta_v)\nabla_{\theta_\pi}\log\pi(\mathbf{a}_t|\mathbf{x}_t;\theta_\pi). \qquad (26)$$

The insight of this upgrade is to allow the learning agent to decide not only how good its actions were but also how much better the actions turn out to be than expected. In this article, we consider the more advanced estimate of the advantage function as

$$A_\pi(\mathbf{x}_t,\mathbf{a}_t;\theta_v) = r_t + \gamma V_\pi(\mathbf{x}_{t+1};\theta_v) - V_\pi,(\mathbf{x}_t;\theta_v) \qquad (27)$$

which can substantially reduce the variance of the policy gradients [30], [31]. Please refer to [30] for more details. Therefore, the critic network parameter θ_v can be trained as

$$\theta_v \leftarrow \theta_v - \alpha_v \sum_{t=0}^{T-1}(r_t + \gamma V_\pi(\mathbf{x}_{t+1};\theta_v) - V_\pi(\mathbf{x}_t;\theta_v))^2 \qquad (28)$$

where α_v is the learning rate for training the critic neural network.

In order to encourage the learning agent to do more exploration, the entropy of the policy $H(\pi(\mathbf{a}_t|\mathbf{x}_t;\theta_\pi))$ is introduced. $H(\pi(\mathbf{a}_t|\mathbf{x}_t;\theta_\pi))$ works as the underlying meanings of entropy:

following the policy π, if the output actions are with relatively similar probabilities, the entropy will be high; oppositely, if a single action is preferred with a large probability, the entropy will be low. The entropy term is added into the policy loss function and the update of the parameter θ_π for the actor network is upgraded as

$$\theta_\pi \leftarrow \theta_\pi + \alpha_\pi \sum_{t=0}^{T-1} A_\pi(\mathbf{x}_t, \mathbf{a}_t; \theta_v) \nabla_{\theta_\pi} \log \pi(\mathbf{a}_t|\mathbf{x}_t; \theta_\pi) + \beta \nabla_{\theta_\pi} H(\pi(\mathbf{a}_t|\mathbf{x}_t; \theta_\pi)) \tag{29}$$

where the parameter β should be set to a large value at the beginning of the training so as to encourage the exploration, whereas it should decrease over time to focus on the exploiting for the purpose of improving the rewards.

B. Asynchronous Training

Reinforcement learning is known to be unstable or even to diverge when a nonlinear approximator is used to represent the value function [29], where one cause of this instability is the correlation of the consecutive state updates. To get around this issue, deep Q-learning used a biologically mechanism, termed as experience replay. The learning agent's large amount of history experiences are stored into a replay memory, and then a batch of the experiences are randomly sampled from the buffer to train the deep neural network. By randomizing over the experiences, the correlations in the observation states can be removed.

For the A3C algorithm, experience replay is no longer used, but instead it gets many learning agents working simultaneously. Each learning agent interacts with its own incarnation of the environment and collects experience. Since the experiences of each learning agent are independent, the overall experiences available for training the neural networks are becoming more diverse and uncorrelated.

For the implementation of the asynchronous training, each of the learning agents should run on a separate processor thread, so that the training works in parallel. There is a global network that holds the shared parameters (θ_π, θ_v), and it is constantly and asynchronously being updated by each of the learning agents. The asynchronous update means that each learning agent updates the shared parameters once it terminates one training episode. Then the global network holds the newly updated parameters, and the next learning agents will use the new parameters to start its own training. Since all the learning agents work in parallel, a linear speedup is expected to be proportional with the number of agents.

V. Simulation Results and Discussion

In the simulation, we consider there are three DSSs requesting data processing services, and in the edge computing layers, there are three ECNs providing the computing services. The computing segments generated from each DSSs follow a Poisson arrival process with different average arrival rate λ_i. The ECNs have different computing rate, noted as μ_l. We assume the IoT devices are randomly located within a range around the ECNs, and the wireless channel states between the IoT devices and the ECNs follow a Markov process as [22]. The detailed parameters are listed in Table I.

TABLE I
Parameter Values in Simulations

Parameter	Value
Mini-batch sizes	50, 50
Experience replay buffer sizes	5000, 5000
Discount factor	0.9
Exploration probability	0.01
Total training steps	100,000
Processing rate	[1, 7.5, 6.5]
Segment arrival rate	[0.5, 1.5, 3.5]
Delay constraints	[0.1, 7.9, 10]
Failure probabilities p_b^w	[0.31, 0.33, 0.37]
Unit price for energy consumption e_c^w	[0.2, 0.22, 0.2]
Consumed energy in unit time p_c^w	[0.17, 0.2, 0.12]

In addition,we used a GPU-based server,which has 4 Nvidia GPUs with version GTX TITAN. The CPU is Intel Xeno E5-2683 v3 with 128-GB memory. Software environment we utilized is Pytorch 1.3.1 with Python 3.7 on Windows 10.

In the proposed scheme, we consider the various delay constraints from different DSSs, the time-varying channel states, and the time-varying computing states of different ECNs. For comparison, four baseline schemes are considered in the simulation part.

1) *Proposed Scheme With Strict Delay Constraints:* The delay constraints from all the DSSs are extremely strict.
2) *Proposed Scheme Without Delay Constraints:* All the DSSs have no delay constraints.
3) *Proposed Scheme With Static Channels:* The wireless channel states are assumed to be static, which does not change dynamically.
4) *Existing Scheme With Identical Settings:* For the experiments under various task arrival rates, all the DSSs have the same delay requirements; For the experiments under various edge computing processing rates, all the ECNs have the same processing rates.

A. Convergence Performance

Fig. 3 shows the convergence performance of the proposed scheme. From Fig. 3, we can observe that the average reward of the proposed scheme is very low at the beginning of the learning process. With the increase of the number of the episode, the average reward increases until it reaches a relatively stable value, which is around 150, and in the meantime, the loss function decreases nearly to zero. This shows the convergence performance of the proposed scheme, which converges at a reasonable speed around 1.5104 decision episodes.

B. Performance Under Various Segment Arrival Rates

In this experiment, we attempt to demonstrate the average performance of edge computing resource allocation per

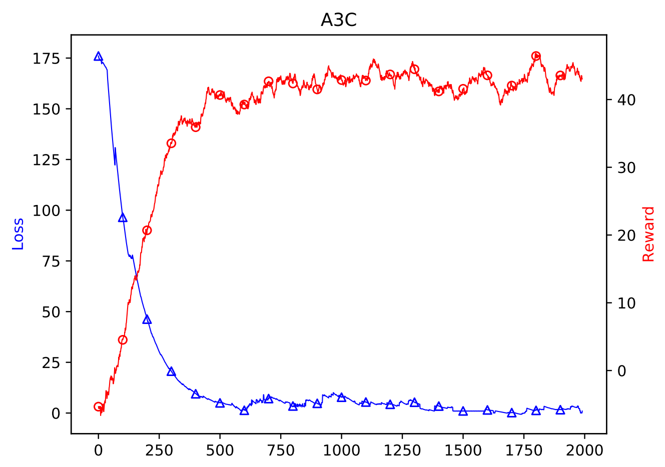
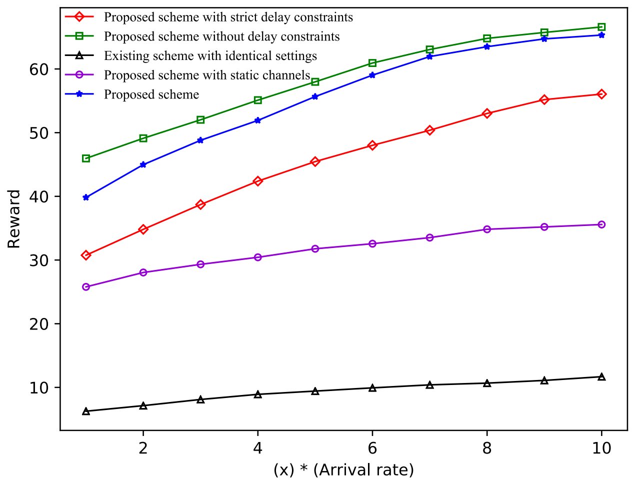


Fig. 3. Convergence performance with the proposed scheme.

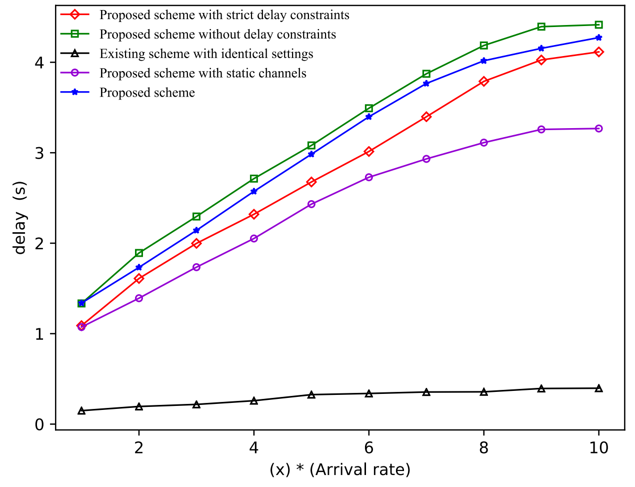
episode in terms of the average reward, the average total delay, and the average task drop rate under different times of the originally set arrival rates λ_i. The simulation results are illustrated in Fig. 4.

Fig. 4(a) exhibits the average reward performance of the proposed scheme and the compared baseline schemes. From the results, we can see that with the increase of segment arrival rates, the rewards of all the schemes increase, but gradually tend to be steady, which is due to the limited edge computing processing rates. For the higer arrival rates, the edge nodes cannot process all the computing requests, thus just maintain a relatively stable value. In contrast, for the proposed scheme with strict delay constraints, lots of segments may be dropped off due to the delay limitations, which leads to the lower reward. Particularly, we can also observe that the gap between the reward of the proposed scheme and the proposed scheme without delay constraints narrows down with the increase of arrival rates. This is because for higher arrival rates, both the schemes can process nearly identical numbers of segments during a specific period of time (i.e., an episode). Furthermore, compared with the existing scheme with identical settings, where the delay constraints are the same from all the DSSs, our proposed scheme performs much better. Due to the limited computing resources, not every requirement can be satisfied. Based on the segment arrival rates, the wireless channel qualities between DSSs and edge nodes, and DSSs' delay constraints to selectively allocate the optimal edge nodes to serve users, which can optimze the total reward.

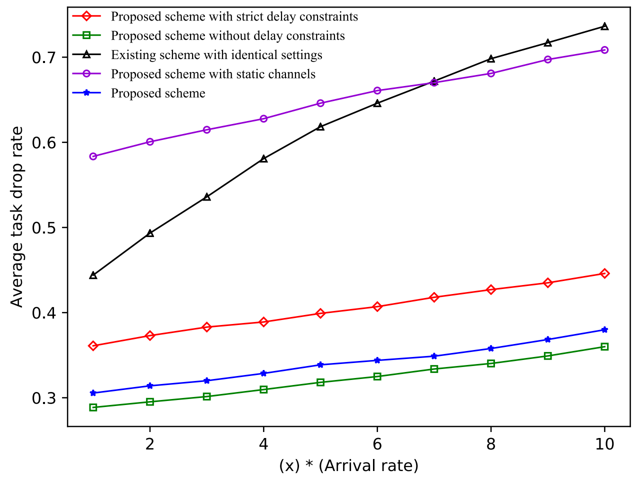
Fig. 4(b) shows the average total delay per episode, which includes the transmission delay, the queuing dealy and the computing delay. For the proposed scheme without delay constraints, the delay is the largest. This is because it is inclined to prioritize the numbers of connections between DSSs and edge nodes without the consideration of delay constraints, which produces larger reward, but higher delay. Contrarily, for the proposed scheme with strict delay constraints, a large amount of requests will be dropped off with the increase of arrival rates, which will lead to smaller delay, but lower reward. For the proposed scheme with static channels, it ignores part of the transmission delay, thus the total delay is relatively lower.



(a)



(b)



(c)

Fig. 4. Average edge computing resource allocation performance versus segment arrival rates. (a) Average reward per episode. (b) Average delay per episode. (c) Average task drop rate per episode.

Fig. 4(c) demonstrates the average task drop rate per episode. From this figure, we can see that with the increase of segement arrival rate, the drop rate will dramatically increase for the existing scheme with identical settings. This is because when the segment arrival rate is small, random allocation is executed, however, when a large amount of segments arrive, random allocation cannot satisfy the DSS's requirements, which leads to higher task drop rate. Comparitively, our proposed scheme can flexibly and effectively allocate the ECNs. With the increase of segment arrival rate, the drop rate will also increase, but not dramatically, which proves the stability of our proposed scheme. Moreover, for the proposed scheme without delay constraints the average drop rate is the lowest.

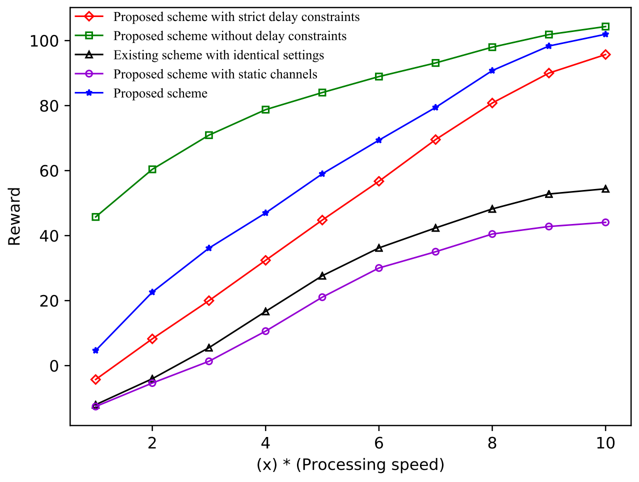
C. Performance Under Various Edge Processing Rates

We do this experiment to exhibit the average performance of edge computing resource allocation per episode in terms of the average reward, the average total delay, and the average task drop rate under different times of the originally set edge computing processing rates μ_i. The simulation results are shown in Fig. 5.

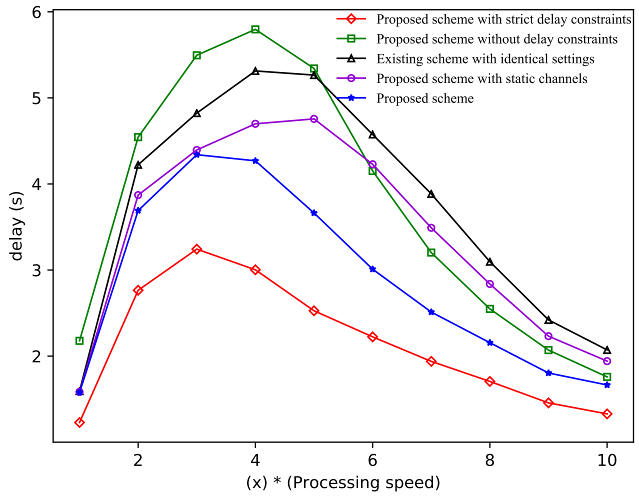
Fig. 5(a) shows the average reward performance of the proposed scheme and other baseline schemes with different processing speed/rates. From this figure, we can observe that with the increase of edge processing speed, the rewards of all the schemes increase, but finally tend to be steady due to the fixed segment arrival rates. Particularly, at the lower processing speed, the gap between the proposed scheme and the proposed scheme without delay constraints is relatively larger. As the processing speed grows, the gap becomes smaller, and evetually tends to converge. This is because, with higher processing speed, more computing requests with various delay constraints of the proposed scheme can be accepted and satisfied, and more reward is obtained. If the processing speed is large enough, the proposed scheme can meet up all the requirements, thus it performs as well as the proposed scheme without delay constraints.

Fig. 5(b) shows the average total delay per episode versus the processing speed. From the figure, we can see that with the increase of processing speed, the delay of all the schemes increase dramatically until reaches to a peak, and then it declines gradually. At the beginning, as the processing speed increases, the edge nodes can process more segments, therefore the delay grows significantly. When the edge processing speed equals to the segment arrival rate, the climax arises, after which the processing speed surpasses the segment arrival rate, so the delay declines. As for the proposed scheme without delay constraints, the delay declines rapidly after passing the climax, and it performs better than both the existing scheme. This indicates that our proposed scheme can allocate the most appropriate edge nodes to the upcoming segments for computing, and proves the effectiveness of the proposed scheme.

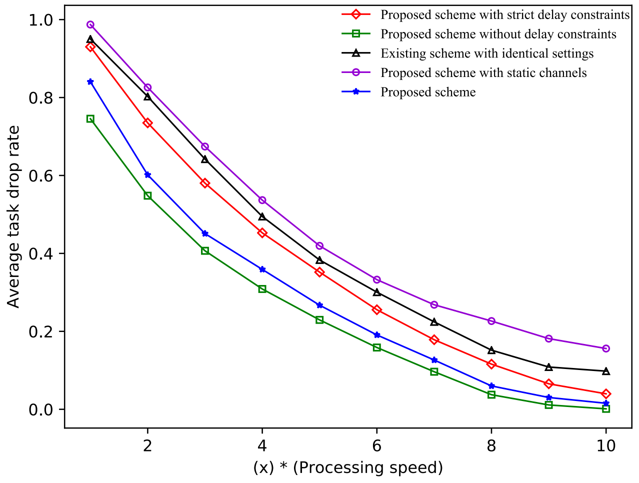
Fig. 5(c) demonstrates the average task drop rate per episode with different processing speed. It is shown that the average drop rates of all the schemes decline with the increase of processing speed because more and more segments can be accepted and processed. For the proposed scheme without delay constraints, the drop rate is the lowest.



(a)



(b)



(c)

Fig. 5. Average edge computing resource allocation performance versus edge processing rates. (a) Average reward per episode. (b) Average delay per episode. (c) Average task drop rate per episode.

D. Effect of Verification Failure Probability

Fig. 6 illustrates the average reward per episode with different failure probabilities of the first ECN. Actually, originally in the experiment, we set $p_b^w = [0.1, 0.2, 0.3]$, $w = 1, 2, 3$ where

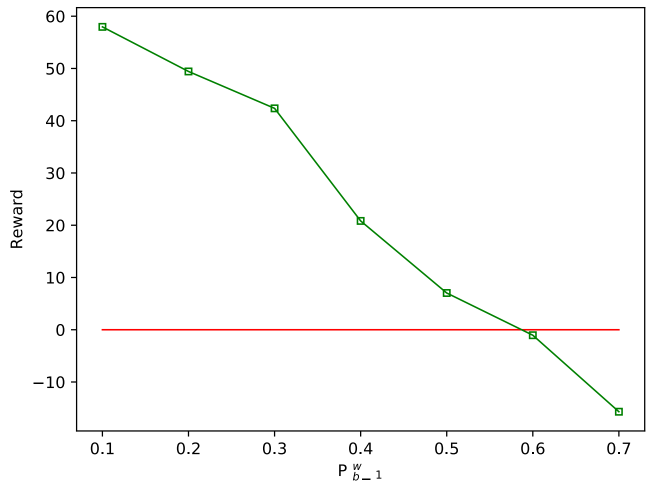


Fig. 6. Average reward with different verification failure probabilities.

each item denotes the failure probability of ECN f_w fails in accomplishing the computation task and cannot pass through the verification in the blockchain system, and it will get punished by slashing a certain percentage of its guarantee deposit. Later on, in order to explore the effect of verification failure probability on the performance, in each experiment, we add 0.1 to each item of p_b^w. The results are shown in Fig. 6. From this figure, we can see that as the failure probabilites increase, the reward will decline significantly. When the failure probability is over a certain value, the reward becomes negative. This also indicates the effectiveness of the blockchain system on the proposed resource allocation scheme.

VI. Conclusion

In this article, we first studied the security and privacy issues of edge computing-enabled IoT, and then the characteristics of blockchains that allow blockchains well suited for the IoT scenarios were presented. A general framework for blockchain-based edge computing-enabled IoT scenarios was proposed. Within the proposed framework, the complete procedure of a transaction was specified step by step. Furthermore, a smart contract was designed within a private blockchain network, which exploited the state-of-the-art reinforcement learning, A3C algorithm, to solve the edge computing resources allocation problem. Specifically, different from the existing edge computing resource allocation schemes, this proposed scheme focused on multiple service subscribers, and distinguished different QoS requirements from subscribers, which leads to higher efficiency. This exemplifies how AI can be combined with blockchains. Finally, simulation results were presented to show the effectiveness of the proposed edge computing resource allocation scheme. In future work, we will further consider joint optimization of the blockchain parameters and the edge computing resource alloction.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 4th Quart., 2015.

[2] M. M. Rathore, A. Ahmad, A. Paul, and S. Rho, "Urban planning and building smart cities based on the Internet of Things using big data analytics," *Comput. Netw.*, vol. 101, pp. 63–80, Jun. 2016.

[3] C. Qiu, X. Wang, H. Yao, J. Du, F. R. Yu, and S. Guo, "Networking integrated cloud-edge-end in IoT: A blockchain-assisted collective q-learning approach," *IEEE Internet Things J.*, early access, Jul. 7, 2020, doi: 10.1109/JIOT.2020.3007650.

[4] C. Qiu, F. R. Yu, H. Yao, C. Jiang, F. Xu, and C. Zhao, "Blockchain-based software-defined industrial Internet of Things: A dueling deep *Q*-learning approach," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4627–4639, Jun. 2019.

[5] P. G. Lopez *et al.*, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, 2015.

[6] J. Du, L. Zhao, J. Feng, and X. Chu, "Computation offloading and resource allocation in mixed fog/cloud computing systems with min–max fairness guarantee," *IEEE Trans. Commun.*, vol. 66, no. 4, pp. 1594–1608, Apr. 2018.

[7] J. Du, L. Zhao, X. Chu, F. R. Yu, J. Feng, and I. Chih-Lin, "Enabling low-latency applications in LTE-A based mixed fog/cloud computing systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1757–1771, Feb. 2019.

[8] K. Yeow, A. Gani, R. W. Ahmad, J. J. Rodrigues, and K. Ko, "Decentralized consensus for edge-centric Internet of Things: A review, taxonomy, and research issues," *IEEE Access*, vol. 6, pp. 1513–1524, 2018.

[9] S. Shen, Y. Han, X. Wang, and Y. Wang, "Computation offloading with multiple agents in edge-computing–supported IoT," *ACM Trans. Sens. Netw.*, vol. 16, no. 1, pp. 1–27, 2019.

[10] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.

[11] X. Wang, C. Wang, X. Li, V. C. Leung, and T. Taleb, "Federated deep reinforcement learning for Internet of Things with decentralized cooperative edge caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, Oct. 2020.

[12] H. Zhang, Y. Xiao, S. Bu, D. Niyato, F. R. Yu, and Z. Han, "Computing resource allocation in three-tier IoT fog networks: A joint optimization approach combining Stackelberg game and matching," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1204–1215, Oct. 2017.

[13] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Netw.*, vol. 33, no. 5, pp. 156–165, Sep./Oct. 2019.

[14] X. Li, X. Wang, P.-J. Wan, Z. Han, and V. C. Leung, "Hierarchical edge caching in device-to-device aided mobile networks: Modeling, optimization, and design," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 8, pp. 1768–1785, Aug. 2018.

[15] R. Yang, F. R. Yu, P. Si, Z. Yang, and Y. Zhang, "Integrated blockchain and edge computing systems: A survey, some research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1508–1532, 2nd Quart., 2019.

[16] M. Liu, F. R. Yu, Y. Teng, V. C. Leung, and M. Song, "Computation offloading and content caching in wireless blockchain networks with mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11008–11021, Nov. 2018.

[17] J. Xie *et al.*, "A survey of blockchain technology applied to smart cities: Research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2794–2830, 3rd Quart., 2019.

[18] *Dual Integrations*, Monax Ind., New York, NY, USA, 2016. [Online]. Available: https://monax.io/explainers/dual_integration/

[19] P. K. Sharma, M.-Y. Chen, and J. H. Park, "A software defined fog node based distributed blockchain cloud architecture for IoT," *IEEE Access*, vol. 6, pp. 115–124, 2017.

[20] S. Kolozali, M. Bermudez-Edo, D. Puschmann, F. Ganz, and P. Barnaghi, "A knowledge-based approach for real-time IoT data stream annotation and processing," in *Proc. IEEE Int. Conf. Internet Things (iThings) Green Comput. Commun. (GreenCom) Cyber Phys. Soc. Comput. (CPSCom)*, Taipei, Taiwan, 2014, pp. 215–222.

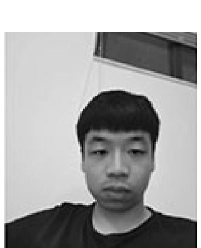
[21] P. Doug and T. Eric. (2017). *Livepeer Whitepaper: Protocol and Economic Incentives for a Decentralized Live Video Streaming Network*. [Online]. Available: https://github.com/livepeer/wiki/blob/master/WHITEPAPER.md#livepeer-protocol

[22] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.

[23] L. Wei, J. Cai, C. H. Foh, and B. He, "QoS-aware resource allocation for video transcoding in clouds," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 1, pp. 49–61, Jan. 2017.
[24] X. Guo and O. Hernández-Lerma, "Continuous-time controlled Markov chains with discounted rewards," *Acta Applicandae Mathematica*, vol. 79, no. 3, pp. 195–216, 2003.
[25] F. Yu and V. Krishnamurthy, "Optimal joint session admission control in integrated WLAN and CDMA cellular networks with vertical handoff," *IEEE Trans. Mobile Comput.*, vol. 6, no. 1, pp. 126–139, Jan. 2007.
[26] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[27] M. Jaderberg *et al.*, "Reinforcement learning with unsupervised auxiliary tasks," 2016. [Online]. Available: arXiv:1611.05397.
[28] Y. Wu and Y. Tian, "Training agent for first-person shooter game with actor-critic curriculum learning," in *Proc. Int. Conf. Learn. Represent.*, 2016.
[29] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[30] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015. [Online]. Available: arXiv:1506.02438.
[31] H. Mao, "Neural Adaptive video streaming with pensieve," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sic., Massachusetts Inst. Technol., Cambridge, MA, USA, 2017.

Ying He received the Ph.D. degree from Dalian University of Technology, Dalian, China, and Carleton University, Ottawa, ON, Canada, in 2019.

She is currently an Assistant Professor with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. She has published over 20 research papers since 2015. Her current research interests include reinforcement learning, intelligent transportation systems, wireless networks, and blockchain.



Yuhang Wang received the B.S. degree in computer science and technology from Hebei Agricultural University, Baoding, China, in 2019. He is currently pursuing the M.S. degree with Shenzhen University, Shenzhen, China.

His current research interests include machine learning, wireless networks, and mobile-edge computing and caching.

Chao Qiu received the B.S. degree in communication engineering from China Agricultural University, Beijing, China, in 2013, and the Ph.D. degree in information and communication engineering from Beijing University of Posts and Telecommunications, Beijing, in 2019.

She is currently a Lecturer with the School of Computer Science and Technology, College of Intelligence and Computing, Tianjin University, Tianjin, China. From September 2017 to September 2018, she visited Carleton University, Ottawa, ON, Canada, as a visiting scholar. Her current research interests include machine learning, software-defined networking, and blockchain.

Qiuzhen Lin (Member, IEEE) received the B.S. degree from Zhaoqing University, Zhaoqing, China, in 2007, the M.S. degree from Shenzhen University, Shenzhen, China, in 2010, and the Ph.D. degree from the Department of Electronic Engineering, City University of Hong Kong, Hong Kong, in 2014.

He is currently an Associate Professor with the College of Computer Science and Software Engineering, Shenzhen University. He has published over 20 research papers since 2008. His current research interests include artificial immune system, multiobjective optimization, and dynamic system.

Jianqiang Li (Member, IEEE) received the B.S. and Ph.D. degrees from the South China University of Technology, Guangzhou, China, in 2003 and 2008, respectively.

He is a Professor with the College of Computer and Software Engineering, Shenzhen University, Shenzhen, China. He led three projects of the National Natural Science Foundation, and three projects of the Natural Science Foundation of Guangdong province, China. His major research interests include robotic, hybrid systems, Internet of Thing, and embedded systems.

Zhong Ming (Member, IEEE) received the Ph.D. degree in computer science and technology from Sun Yat-sen University, Guangzhou, China, in 2003.

He is currently a Professor with the National Engineering Laboratory for Big Data System Computing Technology and the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include software engineering and Web intelligence.