Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 8017334, 12 pages
https://doi.org/10.1155/2021/8017334

Research Article

Deep Reinforcement Learning for Scheduling in an Edge Computing-Based Industrial Internet of Things

Jingjing Wu,[1] **Guoliang Zhang**,[1] **Jiaqi Nie**,[1] **Yuhuai Peng**,[1] **and Yunhou Zhang**[2]

[1]*School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China*
[2]*Northeast Branch of State Grid Corporation of China, Shenyang 110180, China*

Correspondence should be addressed to Yunhou Zhang; zhangyunhou@ne.sgcc.com.cn

Received 4 June 2021; Accepted 22 July 2021; Published 10 August 2021

Academic Editor: Xiaojie Wang

The demand for improving productivity in manufacturing systems makes the industrial Internet of things (IIoT) an important research area spawned by the Internet of things (IoT). In IIoT systems, there is an increasing demand for different types of industrial equipment to exchange stream data with different delays. Communications between massive heterogeneous industrial devices and clouds will cause high latency and require high network bandwidth. The introduction of edge computing in the IIoT can address unacceptable processing latency and reduce the heavy link burden. However, the limited resources in edge computing servers are one of the difficulties in formulating communication scheduling and resource allocation strategies. In this article, we use deep reinforcement learning (DRL) to solve the scheduling problem in edge computing to improve the quality of services provided to users in IIoT applications. First, we propose a hierarchical scheduling model considering the central-edge computing heterogeneous architecture. Then, according to the model characteristics, a deep intelligent scheduling algorithm (DISA) based on a double deep Q network (DDQN) framework is proposed to make scheduling decisions for communication. We compare DISA with other baseline solutions using various performance metrics. Simulation results show that the proposed algorithm is more effective than other baseline algorithms.

1. Introduction

With the rapid development of the Internet of things (IoT), an increasing number of daily services can easily obtain seamless network connectivity everywhere. Among the IoT extensions, the industrial Internet of things (IIoT) is considered a promising technology and has attracted much attention [1–3]. With the popularization of modern industry, IIoT devices, such as wireless sensors, programmable logic controllers (PLCs), remote terminal units (RTUs), and smart switches, are used to improve manufacturing efficiency and realize traditional industry intellectualization. In recent years, the IIoT has been utilized in many fields, including mining, healthcare monitoring, energy generation, and smart factories. Nevertheless, with the explosive growth in IIoT applications, the network environment becomes increasingly complex, which leads to unprecedented challenges, e.g., intermittent wireless connections, scarce spectrum resources, and high propagation delay. Further research needs to be performed to address the aforementioned problems.

Interconnected sensors or smart devices can collect data and interact through modern industrial network infrastructure via the Internet. In this case, these sensors and devices generate large amounts of data that need further processing, which provides intelligence to both continuous environmental monitoring and data analysis [4]. In traditional cloud-based network architecture, all data must be uploaded to a centralized server, and the processed solutions need to be sent back from the cloud to terminal sensors and devices. This process creates high communication latency between end users and the cloud located far away from most end users. Introducing the edge computing [5, 6] paradigm into the IIoT is widely accepted as a promising technology to address the aforementioned problems. According to this paradigm, heavy computational tasks or multiple functions can be delivered at the edge of the network. In edge computing,

the massive on-site data generated by different types of end users can be analyzed at the network edge, rather than transmitting data to distant centers to address delay and bandwidth concerns. Compared with cloud computing, the introduction of edge computing can reduce network congestion and promote resource optimization. Edge computing is more suitable for integrating with the IIoT with a large number of end users, and the architecture can be considered for future IIoT infrastructures.

With the rapid growth in edge computing-based IIoT, to be more competitive, a manufacturing system should have good flexibility, fast response capabilities, and sophisticated heterogeneous structures. Therefore, scheduling [7, 8] plays an important role in ensuring the reliability and responsiveness of a manufacturing system. Communication devices are dedicated to transmitting data, while control devices perform real-time scheduling for dynamic decision-making information. In an IIoT system, most of the industrial devices generate regular packets, such as the packets obtained by detecting or collecting on-site environment. However, when emergency events occur, some urgent packets are generated, which need to be delivered to the destination node within a specified deadline [9]. Edge computing servers must provide different levels of services to each request, which means that some types of requests may have higher priorities than others. Accordingly, scheduling algorithms for edge computing must satisfy expectations for each type of IIoT request in heterogeneous applications without wasting resources.

Additionally, the existing network protocols face tremendous pressure in industrial applications with massive data, limited transmission bandwidth, high data rates, and low-latency requirements. When the actual states of the network change and the transmission strategies need to be adjusted, the existing network protocols lack intelligence. Therefore, with the growth in network scale and the explosion in data, artificial intelligence (AI) [10] has been drastically promoted in recent years. The AI technique has already made breakthroughs in a variety of areas, such as robot control, automatic drive, and speech recognition. Compared with conventional methods, deep reinforcement learning (DRL) [11] that emerged from AI has shown great advantages in large-scale networks. For example, trained architectures can be utilized to monitor data processing, classification and decision-making with high accuracy. Once abnormal traffic occurs, the DRL architectures can quickly make judgments before deterioration spreads in the networks. The reason for the abovementioned performance improvement is that DRL can efficiently extract the features from the sample data and learn the relationships among multiple metrics by training a large quantity of data. By accumulating action experiences from interactions with the target by reinforcing actions leading to higher rewards, DRL can learn successful policies progressively. However, modeling and prediction of communication networks are very difficult because they have become more sophisticated and dynamic. Hence, deploying a more intelligent scheduling algorithm in networks is a necessary condition for rational allocation network resources.

In this paper, we propose a deep intelligent scheduling algorithm (DISA), an intelligence-driven experiential network architecture that exploits edge computing and DRL for scheduling. Our principal contributions are summarized as follows.

(1) We use the idea of DRL to describe the scheduling problem in edge computing-based IIoT and define the corresponding state space, action space, reward function, and value function

(2) We propose DISA, an edge computing-based network architecture for communication scheduling. DISA adapts the DDQN scheme, which guarantees the stability of dynamically generated policies

(3) Effective simulation experiments demonstrate that the proposed DISA can effectively create scheduling strategies for traffic flows. Numerical results verify the superiority of DISA over the typical baseline algorithms

The remaining sections of this paper are organized as follows. Section 2 reviews the related work done in other studies for the proposed problem. Then, problem descriptions and scheduling models are discussed in Section 3. Section 4 introduces the implementation details of our DISA. Thereafter, simulation results are analyzed in Section 5. Finally, we conclude the paper in Section 6.

2. Related Work

This section discusses the major works that have been cited in response to the problem raised in the preceding section.

2.1. Edge Computing-Based IIoT. There have been many studies on edge computing in recent years, and edge computing provides computational and processing facilities at the edge of the network. The authors in [12] presented an edge-cloud interplay based on software-defined network (SDN) to handle flow scheduling among edge and cloud devices. In this respect, a multiobjective evolutionary algorithm based on TChebycheff decomposition was designed for scheduling in an IIoT environment. The authors in [13] expounded the development and integration process of IIoT and edge computing through an extensive review of the research achievements. They proposed a reference architecture of IIoT edge computing and carried out a comprehensive explanation from a number of performance indicators. The work in [14] drew on the idea of blockchain and solved the problem of traffic classification in the Internet of things. A voting-based consensus algorithm was designed to synchronize and update the binary coding tree set and hash table required for the classification of extended hash streams based on edge nodes. In a survey paper [15], the authors investigated the motivation of the edge cloud environment, the latest research results, key enabling technologies and possible future applications. The purpose was to fully understand the edge computing issue through this comprehensive discussion. The authors in [16] roughly summarized the main

structure of edge computing technology as fog computing, mobile edge computing and cloudlet. Under each model structure, they gave detailed tutorials on principles, system architecture, standards, and applications.

2.2. Scheduling Technologies in IIoT. Currently, an increasing number of researchers and practitioners are attempting to solve dynamic scheduling problems in the field of wireless networks. Literature [17] proposed an efficient packet emergency sensing scheduling algorithm for smart cities. The algorithm divided data packets into three priority levels. High-priority data packets were sent to the destination node first, while low-priority data packets only need to be delivered before the deadline. The sensor scheduling problem in power constrained wireless networks was studied in Reference [18]. The communication channels in the wireless network were modeled as ergodic Markov chains, and different transmission power levels were used in different channels to ensure the success of service transmission. The authors in [19] proposed an offline scheduling algorithm based on imitation learning. Specifically, they described the scheduling problem as an optimization problem, established the system model, designed the imitation learning scheduling algorithm, and obtained the optimal scheduling results. In [20], a real-time scheduling scheme based on bargaining game was presented to achieve real-time scheduling in the manufacturing workshop. This paper proposed a flexible workshop architecture, which provided a new paradigm for manufacturing enterprises to improve real-time scheduling efficiency to eliminate the impact of abnormal events. The authors in [21] adopted Lyapunov optimization technology to solve the asymptotically optimal solution to the mobile edge computing offloading scheduling problem under the condition of partial network knowledge. The Lyapunov optimization problem is decomposed into a knapsack problem for solving asymptotically optimal scheduling.

2.3. Intelligence-Driven Architecture in IIoT. Because traditional schemes rely heavily on manual processes when configuring data transmission strategies, it is a great challenge to design dynamic near-optimal control decisions in large networks. Currently, a lot of research work has shifted its focus to the direction of how to make the industrial IoT more intelligent in data transmission and network management. An online task scheduling algorithm based on imitation learning was proposed in [22] to minimize system energy consumption while meeting task delay requirements. This article was an early endeavor to use intelligent learning for online task scheduling in the vehicular edge computing network, which allowed the learning agent to consistently follow the expert's strategy and had a tolerable theoretical performance gap. The authors in [23] introduced deep learning of the IoT to edge computing to make the network performance optimized and user privacy security when uploading packets. The edge computing technology reduced the network data volume from IoT terminals to cloud servers, because the edge nodes uploaded intermediate packets instead of input packets. The work in [24] proposed a priority-aware reinforcement learning-based integrated design network subsystem. This method automatically assigned sampling rates and backoff delays to the control and network subsystems in the industrial Internet of things system. In order to improve the system performance of highly coupled industrial IoT, according to the characteristics of industrial systems, Reference [25] leveraged reinforcement learning technology to automatically configure control and network systems in dynamic industrial environments. Three new strategies are designed to accelerate the convergence of reinforcement learning. The authors in [26] proposed a service quality-aware secure routing protocol (DQSP) based on deep reinforcement learning. While ensuring QoS, this method extracted knowledge from historical traffic demands by interacting with the underlying network environment and dynamically optimized routing strategies.

However, further studies are still necessary on dynamic scheduling considering several performance metrics in IIoT applications. Moreover, few studies have investigated scheduling algorithms supported by intelligence-driven architectures.

3. Problem Definition and Models

3.1. Network Framework. In this section, we adopt a hierarchical structure to generalize all the contents in the IIoT network, as shown in Figure 1. There are three layers in this structure: the device layer, the edge intelligence layer, and the centralized intelligence layer. The device layer is composed of all objectives, workmen, users, and smart terminals that can collect industrial data from live environments. The edge intelligence layer provides distributed, low-latency, and limited computing resources between the device layer and the higher layer. Lightweight DRL-based data distributed computing and edge processing features are implemented in this layer. We introduce the DRL agent into the edge intelligence layer to maintain equivalent performance and offload computational tasks from the cloud. The centralized intelligence layer consists of cloud data centers that aggregate data from lower layers. DRL-based data validation and central processing features are implemented in this layer. We introduce the DRL agent into this layer to optimize network performance and take global control.

The basic method by which smart devices, edge computing servers, and cloud data centers operate and interact is as follows. In the network, different applications generate different traffic types, which occupy different network resources. Here, the collected industrial traffic flows are divided into two categories: computing-intensive traffic flows and time-sensitive traffic flows. Computing-intensive traffic flows require more bandwidth, and the quality of transmission is more crucial. In contrast, time-sensitive traffic flows are sensitive to delay, and latency is more crucial. Thus, all the forwarding decisions are determined by the gateway node set in the corresponding layer. The device layer gateway can process a traffic flow locally, transmit it to an edge computing server, or transmit it to a cloud data center. The control flow can be adopted by the gateway based on the traffic flow classification. The edge intelligence layer is an intermediate layer and is closer to users than the cloud. Edge computing servers can process traffic flow scheduling and routing or forward the

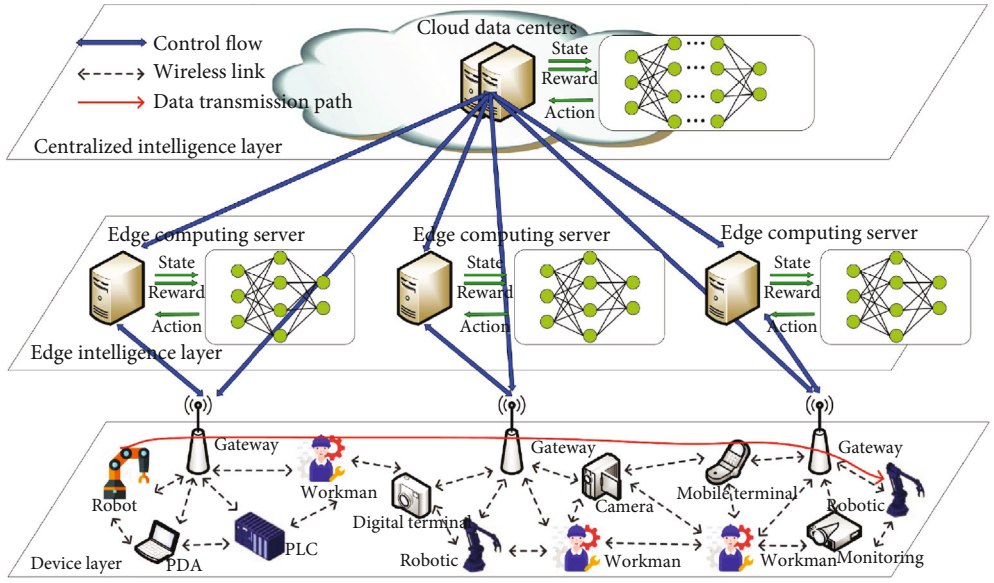

Figure 1: Network frame.

control flows to the higher layer. Time-sensitive traffic flows can be handled in this layer and reduce the overall service delay. The centralized intelligence layer mainly processes computing-intensive traffic flows or the flows forwarded by the lower layer and sends the response back to the lower users. The DRL module is installed in the data center to provide higher service and more efficient resource utilization.

3.2. Scheduling Model. We describe the edge computing-based IIoT scheduling problem in this section. The network topology is modeled as an undirected graph $G(V, E)$. Here, V is the set of IIoT devices, and $E = \{(i, j) \mid i, j \in V\}$ is a set of wireless links. A traffic flow is denoted as a tuple $F = (s, d, b, t_D)$, where $s \in V$ is the source node, $d \in V$ is the destination node, b is the size of traffic flow generated by the device, and t_D is the deadline before which the flow must be transmitted. We denote the set of traffic flows $F = \{1, 2, \cdots, f\}$ and use f to refer to the f^{th} traffic flow. The system operates in a frame-based time-division-multiplexing manner, and the set of time slots for scheduling is denoted as $T = \{0, 1, \cdots, t_{\max}\}$ with frame length $|t_{\max}|$, as shown in Figure 2. Assume there are K channels between two wireless nodes, and the frame lengths on each channel are the same. Additional notations used in this paper are summarized in Table 1.

In our scheduling model, we consider an incremental traffic model. In this model, each traffic flow generated by the end device is individual, and once resources are allocated to it, the traffic flow in the network cannot be redistributed. When network resources cannot successfully provide services for a certain traffic flow, the flow is rejected immediately without suspension. At every scheduling interval, the gateway in the device layer classifies the traffic flows. This step determines at which higher layer the traffic flow can be handled. If the traffic flow is assigned to the edge layer, then the edge layer gateway determines whether the traffic flow can be processed locally, submitted to the centralized layer or

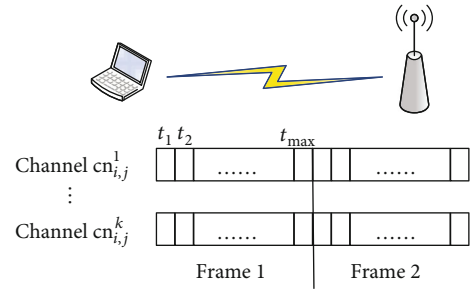

Figure 2: Scheduling model.

rejected based on its deadline. We assume that the post back of the analysis is small, so the feedback transmission delay is identical. Hence, the analysis latency in edge computing server l can be represented by

$$T_f^{\text{ANA}} = T_{t,f}^l + T_{c,f}^l + T_f^{\text{FB}}. \tag{1}$$

If the traffic flow is submitted to the centralized layer because of the lack of computing resources in the edge layer, the analysis latency of flow f runs on the cloud can be calculated as

$$T_f^{\text{ANA}} = T_{t,f}^l + T_{c,f}^{l'} + T_{t,f}^{l-\text{cloud}} + T_{c,f}^{\text{cloud}} + T_f^{\text{FB}}. \tag{2}$$

Here, $T_{c,f}^{l'}$ represents the analysis latency of flow f when the edge layer computing resources are insufficient, and $T_{c,f}^{l'} < T_{c,f}^l$. Thus, the analysis redundancy on the edge layer can be reduced as much as possible.

Table 1: Definitions of notations.

Notation	Definition
t_n^f	The transmission time slots of flow f
$cn_{i,j}^k$	The k^{th} channel connecting i and j
Δt	The time interval of time slot t, $t \geq 0$
$c_{i,j}$	The data rate a time slot can support for link (i, j)
T_f^{ANA}	The analysis latency of flow f
$T_{t,f}^l$	The transmission time between the device and the edge
$T_{t,f}^{l-\text{cloud}}$	The transmission time between the edge and the cloud
$T_{t,f}^{\text{cloud}}$	The transmission time between the device and the cloud
$T_{c,f}^l$	The analysis time on edge computing server l
$T_{c,f}^{l'}$	The preanalysis time on edge computing server l
$T_{c,f}^{\text{cloud}}$	The analysis time on the cloud data center
T_f^{FB}	The analysis feedback of flow f

In the cloud analysis model, the analysis latency of flow f can be represented by

$$T_f^{\text{ANA}} = T_{t,f}^{\text{cloud}} + T_{c,f}^{\text{cloud}} + T_f^{\text{FB}}. \quad (3)$$

We assume that $T_{t,f}^l + T_{t,f}^{l-\text{cloud}} = T_{t,f}^{\text{cloud}}$, and we can infer that the largest analysis latency comes from equation (2). As long as $T_f^{\text{ANA}} \leq T_D$, the traffic flow can be served.

In the scheduling process, the network calculates the channel and the number of time slots for the traffic flow. For instance, traffic flow f is considered, and the size of the traffic flow determines the transmission time slots for accommodating the flow.

$$t_n^f = \frac{b}{c_{i,j}}. \quad (4)$$

We define the execution interval period for a traffic flow denoted by $\text{EIP}_{i,j}^f$, and

$$\text{EIP}_{i,j}^f = \frac{N_F \bullet t_{\max}}{t_n^f}. \quad (5)$$

If a traffic flow has a large quantity of data, it has an intensive execution frequency within the scheduling process; here, N_F represents the number of frames during scheduling. Since a large data size results in a dense execution interval period, computing-intensive traffic flows prefer to choose the channel with more bandwidth, while time-sensitive traffic flows prefer to choose the channel with minimum delay. Once an appropriate channel is allocated, several time slots are allocated to each of the flows corresponding to their individual size. To avoid the overallocation of network resources, flows with the same channel can be distinguished by time slots, as shown in Figure 2.

4. Proposed DISA Mechanism

In this section, we formulate the scheduling problem as a DRL process, including the state space $\mathcal{S}$, action space $\mathcal{A}$, and rewards $\mathcal{R}$. Then, we consider the unique characteristics of dynamic time slot provisioning and enable the DRL agent to optimize the problem.

4.1. DRL Formulation

4.1.1. State. Let s_t denote the network state at time t ($s_t \in \mathcal{S}$), which is composed of the source node, destination node, number of transmission time slots, and time slot occupancy. Therefore, we define the array as

$$s_t = \left\{ i, j, t_n^f, \left\{ t_1^{m,k}, t_2^{m,k}, t_3^k, t_4^k \right\} \mid m \in [1, M], k \in [1, K] \right\}, \quad (6)$$

which summarizes the network information at time interval Δt. The features are explained as follows.

$t_1^{m,k}$ is the available time slot of each execution interval period of the total $|M|$ execution periods according to the size of the traffic flow in the k^{th} channel of the total $|K|$ channels for link (i, j).

$t_2^{m,k}$ is the initial allocation index number of the available time slots in every execution interval period. For $\exists t_1^{m,k} = 0$, $m \in [1, 2, \cdots, M]$, $k \in [1, 2, \cdots, K]$, then $t_2^{m,k}$ is invalid.

t_3^k is the total number of available time slots, which reflects the degree of occupancy situation along the link.

t_4^k is the total number of continuously available time slot blocks, which reflects the degree of fragmentation of resources along the link. This feature helps the agent identify those links that potentially divide the time slots into fragments. Too much fragmentation in time slot may affect access to subsequent traffic flows. Similar to the traditional scheduling problems, these features enable the agent to perceive the capacity, status, traffic load, and security of each wireless link.

Figure 3 shows an example of constructing s_t in DISA. For the sake of simplicity, we assume that there is only one frame. We assume that all the nodes in the network are equal and that there are three traffic flows that arrive in order. There are 2 channels ($K = 2$) in the network, and the slot bitmask on each wireless link corresponds to its time slot utilization. Based on the previous definition, we assume that the three traffic flows require 1, 4, and 2 time slots. In this example, one frame can be divided into 4 execution interval periods at most ($M = 4$).

The first flow needs to select 1 time slot in one frame time of the 2 channels. For instance, the available time slots in channel 1 ($m = 1$, $k = 1$) have a value of $t_1^{1,1} = 2$, and the initial allocation index number is $t_2^{1,1} = 1$. The remaining array elements for channel 1 are $t_3^1 = 6$ and $t_4^1 = 3$. Since the traffic flow needs to be executed once within a frame, time slot 1 in channel 1 is allocated based on the principle of early processing. For the second traffic flow, we determine that not every $t_2^{m,k}$ of 4 execution interval periods in channel 1 is valid, so we can only find idle time slots on channel 2. The four execution

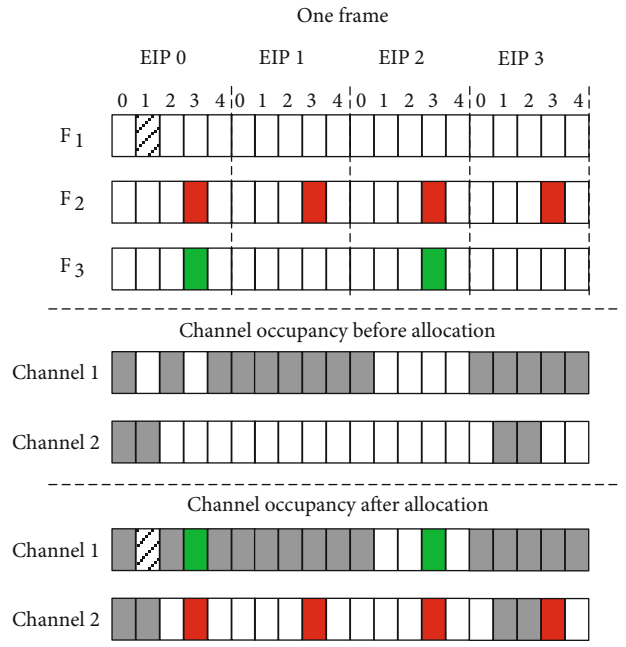

Figure 3: An illustration of state status in DISA.

interval periods have 3, 5, 5, and 3 available time slots ($m = 1, 2, 3, 4, k = 2$). Their initial allocation index numbers are 2, 0, 0, and 0. To maintain equal execution intervals, we choose the time slots with index number 3 in each execution interval period. For the third traffic flow, which requires 2 time slots during scheduling, we need to allocate two execution interval periods in a frame. Other values of state s_t can be seen in Table 2. For each s_t, we assume that both M and K are constants. When the number of possible candidate execution frequencies or channels is less than the array dimension, we assign a constant array to ensure a unified format of s_t.

4.1.2. Action. In the approach, the agent determines which channel and time slot combination is available to assign to the current network state, and the action space is denoted as

$$\mathscr{A} = \{a_1, a_2, \cdots, a_{K \bullet M}\}. \tag{7}$$

An action refers to a channel from the K^{th} candidates and one of the M time slots on the selected channel obtained by the gateway. Therefore, the action space includes $K \bullet M$ actions.

4.1.3. Reward. The reward is the objective of the algorithm. The agent relies on rewards to evaluate the effectiveness of the action and further improve the policies. For any state $s_t \in \mathscr{S}$, $r_t \in \mathscr{R}$ is the immediate reward that numerically characterizes the performance of an action a_t from the discrete set.

The network receives a reward $r_t = 0$ if traffic flow F is successfully received. Otherwise, $r_t = -1$. As a result, to avoid congestion action for computing-intensive traffic flows and reduce delay for time-sensitive traffic flows, the objective of the algorithm should be expressed as finding the optimal policy. The details are described in the next section.

Table 2: State values.

Channel 1 states before allocation					
$t_1^{1,1} = 2$	$t_1^{2,1} = 0$	$t_1^{3,1} = 4$	$t_1^{4,1} = 0$	$t_3^1 = 6$	$t_4^1 = 3$
$t_2^{1,1} = 1$	$t_2^{2,1} = \varnothing$	$t_2^{3,1} = 1$	$t_2^{4,1} = \varnothing$		
Channel 2 states before allocation					
$t_1^{1,2} = 3$	$t_1^{2,2} = 5$	$t_1^{3,2} = 5$	$t_1^{4,2} = 3$	$t_3^2 = 16$	$t_4^2 = 2$
$t_2^{1,2} = 2$	$t_2^{2,2} = 0$	$t_2^{3,2} = 0$	$t_2^{4,2} = 0$		

4.2. Process of DISA. To allocate channels and time slots efficiently, we use the double deep Q network (DDQN) architecture [24] with experience replay and a greedy policy to solve the reinforcement learning problem. This architecture not only yields more accurate value estimations but also leads to much higher learning stability.

Figure 4 illustrates the DISA architecture, in which a DRL trains and optimizes the actions to address channel selection and transmission scheduling. DISA takes advantage of the edge computing networking paradigm for centralized and automated control of the IIoT device layer management. Specifically, a corresponding gateway interacts with the current DRL agent to collect network states and traffic flow requests and develop scheduling strategies. Upon receiving a traffic flow F (step 1) generated by the end device, the layer gateway fetches the current network state, including the in-service wireless channels, time slot resources, and topology abstraction, and then generates tailored state data s_t for DISA (step 2). The neural network input is a given state s_t, while the output is the value of each function. The action values can be represented by $Q(s_t, a_t; \theta)$, where θ denotes the parameters of the neural network. For each action $a_t \in \mathscr{A}$ in that given state (step 3), which corresponds to a particular channel and time slot combination (step 4), the layer gateway attempts to set up the corresponding wireless connection (step 5). The network receives the scheduling strategies related to the previous operations as feedback and produces an immediate reward r_t for the agent; then, the network moves to the next state s_{t+1}. Then, r_t, s_t, a_t, and s_{t+1} are stored in a replay memory denoted by $\mathscr{D}$ (step 6), from which DISA derives training signals for updating the DRL agent (step 7).

The important ingredient for training the traditional DQN is that it maintains two independent and identical neural networks, a target DQN ($Q(s_t, a_t; \theta')$) and an evaluate DQN ($Q(s_t, a_t; \theta)$). The evaluate DQN is utilized to compute the Q value for each action, while the target DQN produces the Q values to train the parameters of the evaluate DQN. Afterward, the action with the maximum Q value is chosen to set the transmission for F. Both the evaluate DQN and the target DQN employ the same neural network structure as the basic module, which uses a simple fully connected neural network, including one hidden layer. The neural network starts in state s_t and follows the value of each action. It attempts to minimize the loss function defined as

$$L^{\text{DQN}}(\theta) = E\left[\left(Y_t^Q - Q(s_t, a_t; \theta)\right)^2\right]. \tag{8}$$

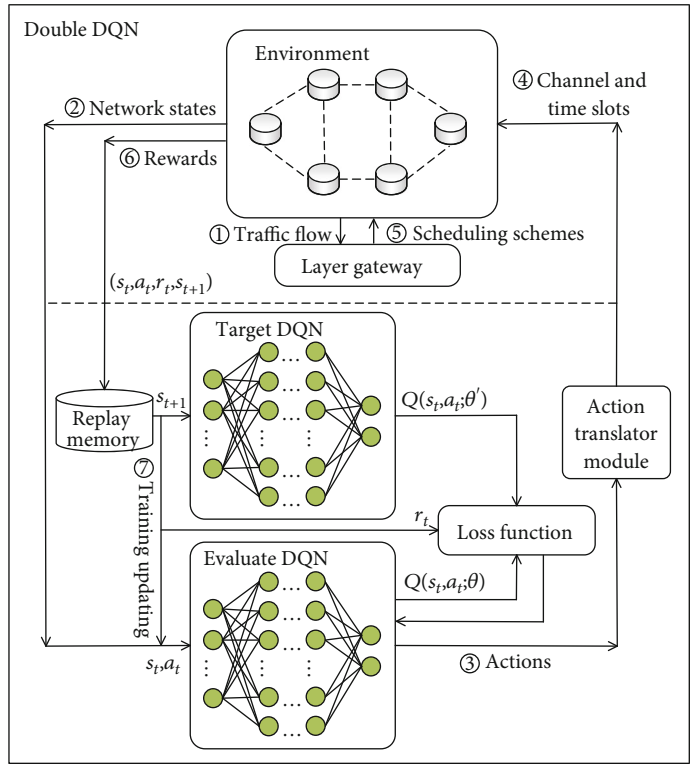

Figure 4: An illustration of the DDQN architecture.

Here, Y_t^Q is the target Q value represented as

$$Y_t^Q = r_{t+1} + \gamma \max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}; \theta'\right). \quad (9)$$

In equation (9), state s_{t+1} is the next state after performing action a_t in state s_t, and action a_{t+1} is an optional action in state s_{t+1}. $\gamma \in [0, 1]$ is a discount factor that trades off the importance of immediate and later rewards. As mentioned above, we can use the experience tuples (s_t, a_t, r_t, s_{t+1}) stored in the replay memory to train the neural network. The target Q value Y_t^Q is determined according to the immediate reward r_{t+1} and the maximum value of $Q(s_{t+1}, a_{t+1})$ obtained by inputting s_{t+1} into the target DQN. Therefore, Y_t^Q can be further expanded as

$$Y_t^Q = r_{t+1} + \gamma Q\left(s_{t+1}, \operatorname*{argmax}_{a_t} Q\left(s_{t+1}, a_t; \theta'\right); \theta'\right). \quad (10)$$

DQN uses the same network parameters for the selection and evaluation of an action, which leads to overoptimistic action values. To avoid this situation, DQN can decouple selection and evaluation so that the double DQN method is proposed. In DDQN, the target value of (10) can be written as

$$Y_t^{\text{DoubleQ}} = r_{t+1} + \gamma Q\left(s_{t+1}, \operatorname*{argmax}_{a_t} Q(s_{t+1}, a_t; \theta); \theta'\right). \quad (11)$$

We choose actions according to the parameters from the evaluate DQN and use the target DQN parameters to measure the value of $Q(s_{t+1}, a_{t+1})$. Then, the loss function is defined as

$$L^{\text{DDQN}}(\theta) = E\left[\left(Y_t^{\text{DoubleQ}} - Q(s_t, a_t; \theta)\right)^2\right]. \quad (12)$$

The overall algorithm is summarized in Algorithm 1.

5. Simulation and Analysis

In this section, we first present the experimental setup and then demonstrate the performance of the proposed DISA compared with several baseline schemes.

5.1. Simulation Setup. All simulation experiments are implemented in a Python environment with TensorFlow. We use a computer with a 5.0 GHz Intel i7 CPU and 16 GB of ARM. We generate topologies of three sizes, namely, small, medium, and large. Each network comprises 15, 22, and 30 gateways, and every gateway is attached to 3 to 5 end devices. Figure 5 shows the experimental environment for this layered structure. It is also assumed that all wireless links have equal bandwidth and that the available channels on each wireless link are set to 8. The number of time slots on each channel is the least common multiple of the number of time slots required by the traffic flows, and the length of each time slot is 0.5 ms. We generate two kinds of traffic flows between nodes. Each traffic flow contains 100 data packets. The average data size of computing-intensive traffic flows is set to 200

1. Initialize the evaluate network with random weights and biases as θ;
2. Initialize the target network as a copy of the evaluate network weights and biases as θ';
3. Initialize replay memory $\mathscr{D}$;
4. for i=1 to *MaxEpisode***do**
5. Initialize state s_t in equation (6);
6. Input the system state s_t into the evaluate DQN;
7. Compute the Q value $Q(s_t, a_t; \theta)$;
8. With probability ε, choose an action a_t;
9. Execute action a_t, receive a reward r_t and observe the next state s_{t+1};
10. Store interaction tuple (s_t, a_t, r_t, s_{t+1}) in $\mathscr{D}$;
11. **for** j =1 to *MaxStep***do**
12. Sample a random transition (s_j, a_j, r_j, s_{j+1}) from $\mathscr{D}$;
13. Compute the target Q value
$y_j = r_{j+1} + \gamma Q(s_{j+1}, \underset{a_j}{\operatorname{argmax}} Q(s_{j+1}, a_j; \theta); \theta')$;
14. Train the network to minimize the loss function
$L(\theta) = E[(y_j - Q(s_j, a_j; \theta))^2]$;
15. Perform gradient descent with respect to θ;
16. Update target networks every N_{DDQN} steps
$\theta' \longleftarrow \theta$;
17. **end for**
18. end for

Algorithm 1: Procedures of DDQN-based DISA.

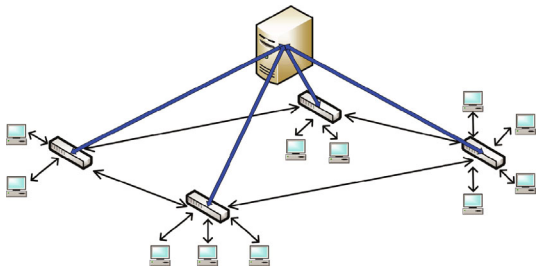

Figure 5: Topology considered in our experiment. Every gateway is attached to 3 to 5 end devices.

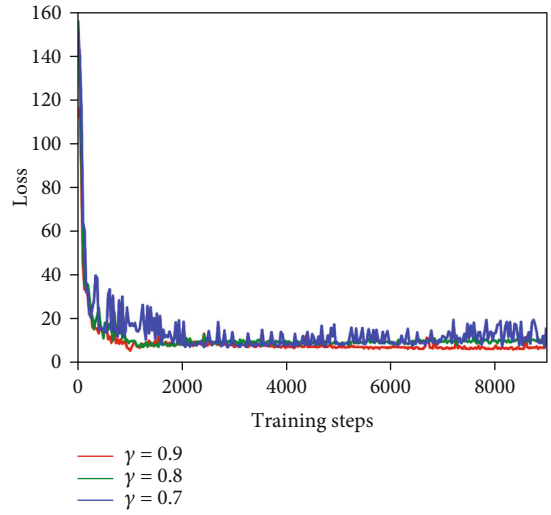

Figure 6: DISA loss function for different discount factors.

bytes, and the average data size of time-sensitive traffic flow is set to 50 bytes. The period of each flow is randomly picked within the range of $2^{7\sim10}$ ms. The relative deadline of each flow is equal to its period. Each simulation experiment without training process is repeated 1000 times, and the average value is obtained as the result of the experiment.

5.2. Result Analysis. We first study the training phase performance of the proposed DISA. The loss function evaluation and the analysis latency (T_f^{ANA}) are two methods to determine how well the model is trained. Figure 6 shows the loss function against the iteration steps of the proposed DISA algorithm at different discount factors γ. The loss value in Figure 6 is obtained during the training process according to (12). It can be seen that all loss values decrease and converge with the increase in the number of iteration steps. After approximately 2,000 iteration steps, the loss value is stable at a low level, which shows the convergence of the DISA algorithm and the effectiveness of the training method. The discount factors γ are set as 0.9, 0.8, and 0.7. We can see that the DISA has the fastest convergence when $\gamma = 0.9$.

Figure 7 depicts the impact of three scale network topologies (small, middle, and large) with 200 data flows on the convergence performance of the analysis latency. In Figure 7, we can observe that the analysis latency is very high at the beginning of the training process. However, as the number of episodes increases, the curves descend and then fluctuate slightly. This is because the Q value estimation needs to gradually improve and the accumulative performance reaches a stable state approximately 60 episodes after the model has been fully trained. In addition, our scheme is not sensitive to the network topology setting because the

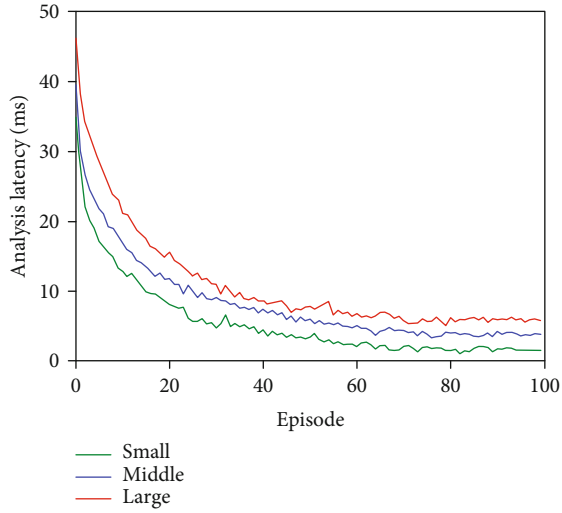

Figure 7: Analysis latency of DISA in different network topologies.

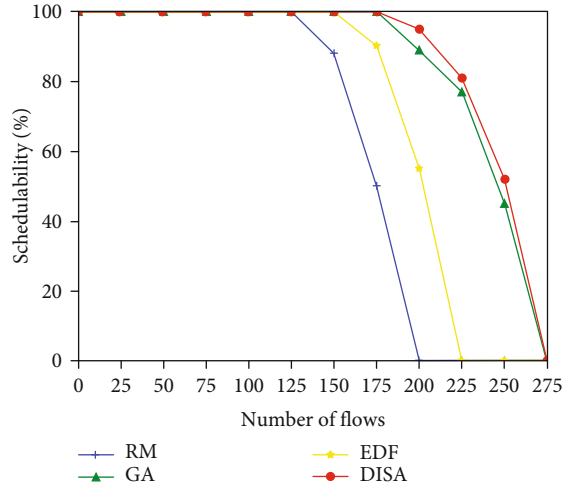

Figure 8: Schedulability of different scheduling algorithms.

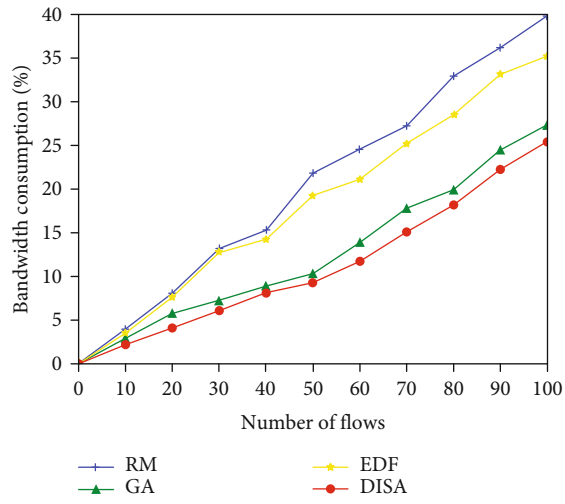

Figure 9: Bandwidth utilization in different scheduling algorithms.

trends of the three curves are similar regardless of the specific values. When the network topology is large, the analysis latency after convergence is the highest, and the value is higher than 6 ms. When the network topology is small, the analysis latency after convergence is the lowest, and the difference between the two topologies is nearly 4 ms. We can conclude that the larger the network topology is, the higher the network complexity and the longer the analysis latency.

At a real industrial site, the schedule scheme is required to generate an available schedule in seconds when a traffic flow occurs. We compare the network performance of our proposed DISA against the following three schemes.

(1) Rate monotonic (RM) scheduling [27]: in this method, traffic priorities are assigned statically and inversely proportional to the traffic periods

(2) Earliest deadline first (EDF) scheduling [28]: EDF assigns priorities based on absolute deadlines, and the traffic flow with the earliest deadline has the highest probability

(3) Genetic algorithm (GA) [29]: the GA converts two separate sets of routing and scheduling constraints into one set of constraints and uses a single step to solve the scheduling problem

What the experiment considers to be schedulable is whether the scheduler returns a feasible solution within the time limit. Figure 8 illustrates the schedulability of the RM, GA, EDF, and DISA algorithms under different network traffic loads. As shown in Figure 8, all the algorithms are schedulable when the number of traffic flows is below 125, and the schedulability drops dramatically after the network load increases. As the network load increases, additional constraints make it harder for the scheduler to find a feasible solution. The schedulability of the four algorithms is always 100% with a network load less than 100. In addition, the value obtained by the DISA is higher than that of RM, GA, and EDF. We can see that, compared to the RM algorithm, the DISA can provide scheduling for more than 50 traffic flows because DISA can usually select more suitable time slots and make more intelligent decisions than traditional algorithms due to the DRL process.

Next, we present the bandwidth consumption of the RM, GA, EDF, and DISA algorithms under different network traffic loads. Here, bandwidth consumption is defined as the ratio of the bandwidth occupied by the traffic flows to the total bandwidth of the occupied wireless links. This ratio reflects the frequency of use of the wireless link. The larger the value is, the more consistently the resources are allocated. In Figure 9, we test the bandwidth consumption performance for the four algorithms. We assume that the network load is below 150 because all traffic flows are scheduled. As shown in Figure 9, the bandwidth consumption of the four algorithms increases as the number of traffic loads increases because the new incoming traffic flows need more bandwidth resources. We can see that the RM algorithm and the EDF

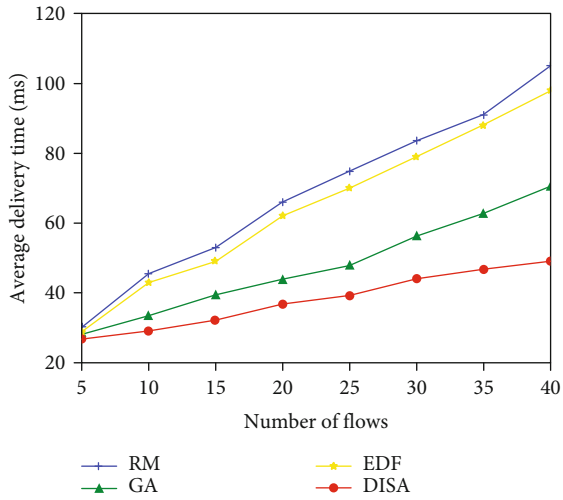

Figure 10: Average delivery time in different scheduling algorithms.

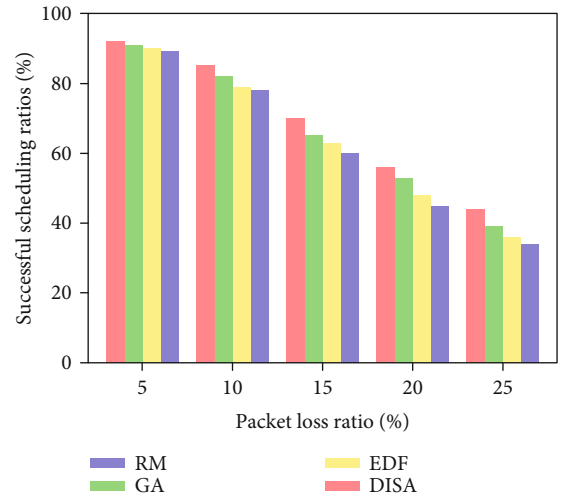

Figure 11: Successful scheduling ratio in different scheduling algorithms.

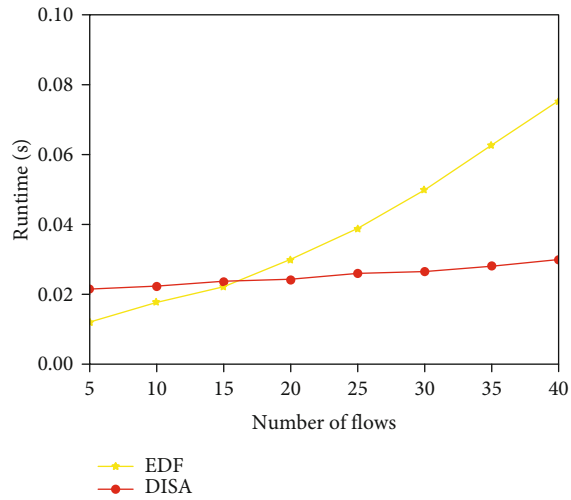

Figure 12: Runtime in different scheduling algorithms.

algorithm have the highest bandwidth consumption because they use a fixed scheduling strategy every time. Constraints in the GA algorithm include load balancing, while the DISA can intelligently arrange the network status, so their bandwidth consumption is relatively low.

We also evaluate the average delivery time of the RM, GA, EDF, and DISA algorithms under different network traffic loads. Here, average delivery time includes the end-to-end transmission time of the traffic flows and the analysis latency of the traffic flows. As shown in Figure 10, the average delivery time of the four algorithms increases as the number of traffic loads increases because the transmission delay is amortized on each traffic flow. It can be observed that the average delivery time is closely related to the bandwidth consumption of the link in the network. High bandwidth consumption means that these links have less bandwidth resources. Traffic flow through these links will increase the transmission delay. High schedulability of the DISA algorithm means shorter end-to-end delay and better utilization of link bandwidth. DISA can achieve load balancing on different links and has achieved the optimal average delivery time. We can see that the RM algorithm and the EDF algorithm have the longest average delivery time, which is nearly 60 ms higher than the shortest DISA algorithm.

We analyze the probability of successful scheduling when the network load is heavy and packet loss occurs. The comparison result for the four algorithms under different packet loss ratios is shown in Figure 11. Here, we assume that the experiment is running on a large topology and devices with 175 traffic flows. Due to insufficient network resources, packet loss occurs randomly. The larger the packet loss ratio is, the fewer idle time slots in the network for scheduling. In Figure 11, we can see that when packet loss occurs in the network, all scheduling algorithms have a failure ratio. The situation becomes increasingly worse. For instance, when the packet loss ratio changes from 5% to 25%, a successful scheduling ratio for DISA can obtain up to a nearly 50% reduction. The successful scheduling ratio for DISA is significantly higher than that of the other three algorithms. DISA ensures network resource allocation by sensing the network status and thus achieves successful scheduling with a higher probability.

Finally, in order to verify the efficiency of the DISA algorithm, we analyze the runtime of the algorithm under different network traffic loads. Here, we only compare the DISA algorithm and the EDF algorithm because they are similar in time scale. As shown in Figure 12, the running time of both algorithms increases as the traffic load increases. Among them, the running time of DISA algorithm has a gentle upward trend, while the change trend of EDF algorithm is more obvious. This is because the DISA algorithm has a training process, and the requirement for computing resources in the network has not changed much. On the other hand, with the increase in traffic loads, the computational complexity of EDF algorithm increases, and its need for CPU computing resources increases.

6. Conclusions

Millions of IIoT end devices generate a billion bytes of data at the edge of the network. Driven by this trend, the combination of edge computing and deep reinforcement learning has received a tremendous amount of attention. In this paper, we proposed a deep intelligent scheduling algorithm (DISA) for the cloud-edge environment. By employing the classic double deep Q network (DDQN) architecture in intelligent scheduling, our DISA can identify reasonable channel and time slot combinations with competitive performance. Following the DDQN framework, the agent separates selected action from the target Q network, leading to more effective parameter training. Extensive simulation experiments were implemented, demonstrating that our DISA can obtain better network performance than the traditional scheduling schemes.

Data Availability

The authors confirm that the data supporting the findings of this study are available within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1702000 and in part by the Fundamental Research Funds for the Central Universities under Grant N2116012, Grant N2116013, and Grant N180708009.

References

[1] K. K. Raymond, G. Stefanos, and J. H. Park, "Cryptographic solutions for industrial Internet-of-things: research challenges and opportunities," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3567–3569, 2018.

[2] M. Z. Hasan and H. Al-Rizzo, "Optimization of sensor deployment for industrial Internet of things using a multiswarm algorithm," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10344–10362, 2019.

[3] K. Tange, M. De Donno, X. Fafoutis, and N. Dragoni, "A systematic survey of industrial Internet of things security: requirements and fog computing opportunities," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2489–2520, 2020.

[4] X. Li, D. Li, J. Wan, C. Liu, and M. Imran, "Adaptive transmission optimization in SDN-based industrial Internet of things with edge computing," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1351–1360, 2018.

[5] Z. Ning, P. Dong, X. Wang et al., "Mobile edge computing enabled 5G health monitoring for Internet of medical things: a decentralized game theoretic approach," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 2, pp. 463–478, 2021.

[6] M. Aazam, K. A. Harras, and S. Zeadally, "Fog computing for 5G tactile industrial Internet of things: QoE-aware resource allocation model," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 5, pp. 3085–3092, 2019.

[7] V. Kotsiou, G. Z. Papadopoulos, P. Chatzimisios, and F. Theoleyre, "LDSF: low-latency distributed scheduling function for industrial Internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8688–8699, 2020.

[8] Z. Ning, P. Dong, X. Wang et al., "Partial computation offloading and adaptive task scheduling for 5G-enabled vehicular networks," *IEEE Transactions on Mobile Computing*, p. 1, 2020.

[9] M. O. Ojo, S. Giordano, D. Adami, and M. Pagano, "Throughput maximizing and fair scheduling algorithms in industrial Internet of things networks," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3400–3410, 2019.

[10] Z. Ning, S. Sun, X. Wang et al., "Blockchain-enabled intelligent transportation systems: a distributed crowdsensing framework," *IEEE Transactions on Mobile Computing*, p. 1, 2021.

[11] Z. Lv, Y. Han, A. K. Singh, G. Manogaran, and H. Lv, "Trustworthiness in industrial IoT systems based on artificial intelligence," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 2, pp. 1496–1504, 2021.

[12] 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 Communications Magazine*, vol. 56, no. 2, pp. 44–51, 2018.

[13] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge computing in industrial Internet of things: architecture, advances and challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020.

[14] H. Qi, J. Wang, W. Li, Y. Wang, and T. Qiu, "A blockchain-driven IIoT traffic classification service for edge computing," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2124–2134, 2021.

[15] J. Pan and J. McElhannon, "Future edge cloud and edge computing for Internet of things applications," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, 2018.

[16] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for Internet of things: a primer," *Journal of Digital Communications and Networks*, vol. 4, pp. 77–86, 2018.

[17] T. Qiu, K. Zheng, M. Han, C. P. Chen, and M. Xu, "A data-emergency-aware scheduling scheme for Internet of things in smart cities," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2042–2051, 2018.

[18] H. Tang, J. Wang, L. Song, and J. Song, "Minimizing age of information with power constraints: multi-user opportunistic scheduling in multi-state time-varying channels," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 5, pp. 854–868, 2020.

[19] X. Wang, Z. Ning, S. Guo, M. Wen, and V. Poor, "Minimizing the age-of-critical-information: an imitation learning-based scheduling approach under partial observations," *IEEE Transactions on Mobile Computing*, 2021.

[20] J. Wang, Y. Zhang, Y. Liu, and N. Wu, "Multiagent and bargaining-game-based real-time scheduling for Internet of things-enabled flexible job shop," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2518–2531, 2019.

[21] X. Lyu, W. Ni, H. Tian et al., "Optimal schedule of mobile edge computing for Internet of things using partial information," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2606–2615, 2017.

[22] X. Wang, Z. Ning, S. Guo, and L. Wang, "Imitation learning enabled task scheduling for online vehicular edge computing," *IEEE Transactions on Mobile Computing*, 2020.

[23] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: deep learning for the Internet of things with edge computing," *IEEE Network*, vol. 64, no. 6, pp. 96–101, 2018.

[24] H. Xu, X. Liu, W. G. Hatcher, G. Xu, W. Liao, and W. Yu, "Priority-aware reinforcement learning-based integrated design of networking and control for industrial Internet of things," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4668–4680, 2021.

[25] H. Xu, X. Liu, W. Yu, D. Griffith, and N. Golmie, "Reinforcement learning-based control and networking co-design for industrial Internet of things," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 5, pp. 885–898, 2020.

[26] X. Guo, H. Lin, Z. Li, and M. Peng, "Deep-reinforcement-learning-based QoS-aware secure routing for SDN-IoT," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6242–6251, 2020.

[27] H. Wang, L. Shu, W. Yin, Y. Xiao, and J. Cao, "Hyperbolic utilization bounds for rate monotonic scheduling on homogeneous multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1510–1521, 2014.

[28] A. A. Ayele, V. S. Rao, K. G. Dileep, and R. K. Bokka, "Combining EDF and LST to enhance the performance of real-time task scheduling," in *International Conference on ICT in Business Industry & Government (ICTBIG)*, Indore, India, 2016.

[29] M. Pahlevan and R. Obermaisser, "Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks," in *IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 337–344, Turin, Italy, 2018.