Intelligent Automation & Soft Computing
DOI:10.32604/iasc.2022.018881
Article

Tech Science Press

An Energy Aware Algorithm for Edge Task Offloading

Ao Xiong[1], Meng Chen[1,*], Shaoyong Guo[1], Yongjie Li[2], Yujing Zhao[2], Qinghai Ou[3], Chuan Liu[4], Siwen Xu[5] and Xiangang Liu[6]

[1]Beijing University of Posts and Telecommunications, Beijing, 100876, China
[2]Communication Operation Center State Grid Henan Electric Power Company, Information & Telecommunication Company, Zheng Zhou, 450052, China
[3]Fibrlink Communications Co., Ltd, Beijing, 102200, China
[4]Global Energy Interconnection Research Institute Co., Ltd, Beijing, 102200, China
[5]Université Paul Sabatier-Toulouse 3[5], Toulouse, 31000, France
[6]China Electronics Standardization Institute, Beijing, 100007, China
*Corresponding Author: Meng Chen. Email: 1342029007@qq.com
Received: 24 March 2021; Accepted: 29 April 2021

Abstract: To solve the problem of energy consumption optimization of edge servers in the process of edge task unloading, we propose a task unloading algorithm based on reinforcement learning in this paper. The algorithm observes and analyzes the current environment state, selects the deployment location of edge tasks according to current states, and realizes the edge task unloading oriented to energy consumption optimization. To achieve the above goals, we first construct a network energy consumption model including servers' energy consumption and link transmission energy consumption, which improves the accuracy of network energy consumption evaluation. Because of the complexity and variability of the edge environment, this paper designs a task unloading algorithm based on Proximal Policy Optimization (PPO), besides we use Dijkstra to determine the connection path between edge servers where adjacent tasks are deployed. Finally, lots of simulation experiments verify the effectiveness of the proposed method in the process of task unloading. Compared with contrast algorithms, the average energy saving of the proposed algorithm can reach 22.69%.

Keywords: Edge computing; energy consumption; task offloading; reinforcement learning

1 Introduction

In recent years, with the rapid development of Internet of things (IoT) technology, tens of billions of terminal devices (TD) in IoT network have realized economic and efficient interconnection. According to the prediction of Cisco visual network index, by 2023, IoT devices will account for 50% of all networked devices, and the number of connections between devices will reach 14.7 billion [1]. Massive data in IoT network promotes the generation of new services, such as Internet of vehicles, face recognition and so on. These services are delay sensitive tasks and require a lot of computing and storage resources.

However, limited by the physical size of the IoT device, the computing and storage resources of terminal devices are limited, and its battery life is short. If tasks are run on the terminal device, it is difficult to meet the user's requirements, which will lead to bad user experience.

Cloud computing offloads the task to the remote cloud servers, which improves the service response speed by using the computing resources in the cloud [2]. However, due to the high delay between the terminal and the cloud, it is difficult to meet the requirements of delay sensitive tasks. Edge computing is a potential solution to the problem. By deploying edge servers at the network edge near the mobile terminal, the computing resources provided by the remote cloud are extended to the location closer to the terminal. Based on edge computing, tasks can be executed directly on edge servers, which saving the communication time of data transmission to the cloud. The main task unloading methods in edge computing environment are binary offloading and partial offloading [3]. The former is aimed at some tasks can't be partitioned or highly integrated. In this way, each task in the queue to be unloaded is executed on the local device, or completely unloaded to edge servers. The latter allows tasks to be partitioned on local devices and edge servers. In this paper, we consider using the binary offloading to unload tasks that cannot be partitioned in IoT environment. For an edge network which performs the unloading task, the total energy consumption consists of two parts: servers energy consumption and links energy consumption. With the explosive growth of IoT devices, the world's data centers are expected to account for 20 percent of the world's electricity consumption by 2025 [4]. Therefore, it is necessary to design an energy aware task offload algorithm.

The problem of energy consumption optimization in the process of task unloading has attracted extensive attention. However, most of existing works focus on the energy consumption management and resource allocation of terminal equipment, lacks energy management method of edge servers, Fig. 1 describes the task unloading process in detail. In Fig. 1 each terminal device has a task queue to be unloaded, each task in the queue is represented by $T_{i,j}$, i is the number of terminal equipment, and j is the number of the task to be unloaded in the task queue. And $T_{i,j+1}$ depends on the output of $T_{i,j}$. We assume that the terminal only completes the data collection and does not participate in the execution of the task.

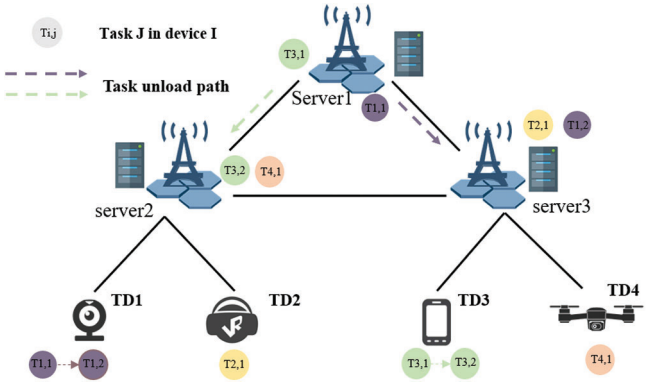

Figure 1: Task offloading in edge computing environment

We focus on the energy consumption optimization of edge servers during task unloading.

In this paper, we first construct a mathematical model including server energy consumption and link energy consumption. Because of the complexity of the edge computing environment, it is necessary to use a reliable and scalable learning algorithm. Based on the above analysis, we design an edge task unloading algorithm based on Proximate Policy Optimization (PPO). The technical contributions of this paper are summarized as follows:

- To accurately describe the energy consumption of edge network during task unloading, we construct a mathematical model to describe the energy consumption, which includes server energy consumption and link energy consumption. The processing energy consumption of the server is directly proportional to the CPU utilization, and the transmission energy consumption of the link is proportional to the bandwidth utilization.
- In this paper, the reward function of reinforcement learning algorithm is designed according to the network energy consumption model, the greater the energy consumption, the smaller reward value. Task offload strategy designed in this paper is responsible for selecting the deployable edge server, and the path between adjacent servers is determined by Dijkstra algorithm. Once a task is unloaded, the available resources (environment states) in the edge computing environment are updated.

The rest of this paper is organized as follows: In the second part, this paper briefly reviews related works. In the third part, we construct a mathematical model to describe the energy consumption of edge network, then constructs the problem as ILP model after considering various constraints. In the fifth part, the task unloading algorithm is simulated and the simulation results are analyzed. The last part summarizes the work of this paper and supplements the parts to be improved.

2 Related Work

In edge computing environment, terminal devices can choose to upload some tasks to the edge server. Through the above operations, not only can the energy consumption of the device be reduced, but also the risk of privacy leakage in traditional cloud computing can be decreased, and the real-time performance of task processing can be improved. However, the existing literature focuses on reducing terminal energy consumption and improving the response speed of unloading tasks. Reference [5] considered how to allocate channel resources, reference [6] focus on how to allocate computing resources, [7–9] comprehensive consideration of multiple factors to reduce terminal energy consumption. By offloading tasks to edge servers, terminal energy consumption can be reduced. However, current researches mainly focus on the allocation of channel resources and computing resources in edge networks, only a few researches involve the energy consumption of edge servers. Reference [10] studied how to ensure the longest running time of the whole system through mutual task unloading among servers in the scenario of limited energy of each edge service. However, this method can only prolong the survival time of the whole edge server node. Considering the energy consumption of data transmission between edge servers, the above research does not reduce the energy consumption of edge server.

Dynamic voltage Scaling (DVS) can dynamically adjust voltage frequency to reduce energy consumption and ensure the quality of service of real-time tasks [11,12] studies how to use DVS to unload tasks from terminal devices to edge servers. The experimental results show that DVS can not only complete the task processing within the specified time, but also reduce the terminal energy consumption. [13,14] further discussed the joint optimization of TD energy consumption and task processing time in Mobile Edge Computing (MEC) system using DVS technology, but none of the above studies involved the energy consumption of edge server.

3 Problem Formulation

In this section, we first construct a task unloading model with energy consumption as the optimization goal in IoT. After considering the constraints of bandwidth, computing resources and traffic conservation, we get an optimization problem model.

3.1 Physical Network Model

In this paper, undirected graph $G(V,L)$ is used to represent the edge network, where V is the set of physical nodes in the network, L is the set of physical links in the edge network, $l_{vu} \in L$. Computing power of each physical node is represented by C_v, the load capacity of each physical link is represented by C_l, and physical link l_{vu} connects node v and node u.

3.2 Task Queue Model

In this section, task queue on each terminal is modeled as a directed graph in this paper, and the tasks to be unloaded in the queue are not repeated. Each task queue can be described by a four tuple $ts = \{(v_s, v_d), R_{ts}, b_{ts}\}$, where v_s and v_d represent the target node and start node of the task queue respectively, and R_{ts} represents the detailed information of the task queue to be unloaded, including the task type and the data dependency between tasks. In this paper, we assume that the transmission bandwidth between tasks in the same queue is same, and the value is b_{ts}. cr_t represents the number of CPUs required to complete task t, and ct_t represents the throughput of task t.

3.3 Energy Consumption Model

The energy consumption model of edge network constructed in this paper includes the energy consumption of edge server and the energy consumption of physical link transmission traffic.

In addition to the energy consumption of computing tasks, the energy consumption of storage devices and communication devices on the edge server is also considerable. Therefore, the energy consumption of the server is modeled as the power consumption of the server starting up and processing the unloading task. The former is the energy required by the edge server to maintain its normal operation, which depends on whether there are tasks deployed on the edge server, regardless of the number of deployment tasks. The latter is positively correlated with CPU utilization. We use N_t^v to indicate the number of tasks t deployed on edge server v.

$$N_t^v = \left\lceil \frac{\sum_{t \in T} z_{ts,t}^v \cdot b_{ts}}{ct_t} \right\rceil \quad \forall v \in V, \forall t \in T \tag{1}$$

Since the energy consumption of the edge server is positively related to the CPU utilization, the processing energy consumption of edge server is calculated as follows:

$$\begin{aligned} p_t^v &= \frac{p_h^s - p_b^s}{C_v} \cdot cr_t \cdot N_t^v \cdot \frac{\sum_{t \in T} z_{ts,t}^v \cdot b_{ts}}{ct_t \cdot N_t^v} \\ &= \frac{p_h^s - p_b^s}{C_v} \cdot cr_t \cdot \frac{\sum_{t \in T} z_{ts,t}^v \cdot b_{ts}}{ct_t} \end{aligned} \tag{2}$$

p_b^s is start up energy consumption, and p_h^s is the energy consumption when the edge server is running at full load. Therefore, the energy consumption p_v of the edge server can be expressed as:

$$p_v = p_b^s \cdot \min\left\{1, \sum_{t \in T} \sum_{ts \in TS} z_{ts,t}^v\right\} + \sum_{t \in T} p_t^v \tag{3}$$

In the above formula, $\min\left\{1, \sum_{t \in T} \sum_{ts \in TS} z_{ts,t}^v\right\} \in \{0,1\}$ indicates that the power consumption can only be calculated once. $\sum_{t \in T} p_t^v$ represents the energy consumption generated by all tasks deployed on edge server v.

Similarly, the physical energy consumption in edge network also includes the power consumption of switches on links and the transmission energy consumption when the link transmits the traffic between servers. The former depends on the power on state of the switch on the link, and latter depends on the bandwidth utilization of the physical link. In this section, r_l is used to represent the bandwidth utilization of link l. the calculation results are as follows:

$$r_l = \frac{\sum_{ts \in TS} w_{ts}^{vfug} \cdot y_{ts}^{uvl} \cdot b_{ts}}{C_l} \forall l \in L, \quad \forall t \in T \tag{4}$$

w_{ts}^{vfug} indicates that task f on task queue ts is deployed on node v and task g is deployed on node u. y_{ts}^{uvl} indicates whether the task ts deployed on link l passes through node v and node u in turn.

Based on the above analysis, the total energy consumption of l can be calculated as:

$$p_l = p_b^l \cdot \min\left\{1, \sum_{t \in T} \sum_{ts \in TS} w_{ts}^{vfug} \cdot y_{ts}^{uvl} \cdot b_{ts}\right\} + \left(p_h^l - p_b^l\right) \cdot r_l \tag{5}$$

p_b^l represents the power consumption of the switch on the link, and $\min\left\{1, \sum_{t \in T} \sum_{ts \in TS} w_{ts}^{vfug} \cdot y_{ts}^{uvl} \cdot b_{ts}\right\} \in \{0, 1\}$ indicates that the power consumption cannot be calculated repeatedly. Once a task is unloaded on the server, the switch must be kept on. p_h^l represents the energy consumption of the link at full load. Hence, the total energy consumption of the edge network can be expressed as:

$$p_{total} = \sum_{v \in V} p_v + \sum_{l \in L} p_l \tag{6}$$

3.4 ILP Model

In this section, after considering the constraints of processing sequence of tasks in the queue, computing capacity constraints, and network bandwidth constraints, the unloading problem of terminal tasks is established as an ILP model with energy consumption as the optimization objective.

Firstly, to meet traffic constraints in the process of task unloading, this section assumes that a task sequence to be unloaded is ts, t_i and t_j are two tasks to be unloaded in ts, and task t_j must be executed later. The above constraints are expressed as follows:

$$C1: \sum_{u \in V, g \in ts} w_{ts}^{vfug} - \sum_{u \in V, g \in ts} w_{ts}^{ufvg} = \begin{cases} 1 & if \quad f = t_i \\ -1 & if \quad f = t_j \\ 0 & otherwise \end{cases} \tag{7}$$

We assume that the tasks in the task sequence can only be unloaded on one edge server:

$$C2: \sum_{v \in V} z_{ts,t}^{v} = 1 \tag{8}$$

The tasks in sequences to be unloaded must be unloaded according to the dependency relationship between tasks. The mathematical formula is used to describe the following formula:

$$C3: \sum_{u\in V, v\in V} w_{ts}^{vfug} = x_{ts}^{fg} \tag{9}$$

In addition, this section also considers the computing capacity constraints of edge servers and the bandwidth constraints of physical links.

$$\begin{aligned} &C4: \sum N_t^v \cdot cr_t \le C_v \\ &C5: \sum w_{ts}^{vfug} \cdot ct_t \cdot b_{ts} \le C_l \end{aligned} \tag{10}$$

In conclusion, the ILP model for Energy Efficient Task Offload (EETO) problem can be expressed as follows:

$$\begin{aligned} &\min \quad p_{total} = \sum_{v\in V} p_v + \sum_{l\in L} p_l \\ &subject \quad to: \quad C1, C2, C3, C4, C5 \end{aligned} \tag{11}$$

4 Proposed Algorithm

4.1 Markov Decision Process for Task Offloading

Markov chain is a probability model, the future state is only related to current states. Markov decision process (MDP) is a decision process based on Markov chain. MDP can be represented by a five tuple (S, A, P, R, γ). S is the state space observed by agent; A represents the action space; P is the set of transition probabilities, the finite set of probabilities that an agent enters a specific state after executing action $a_i \in A$ in a certain state. R represents a set of immediate rewards after performing the action; γ is the discount coefficient.

An edge server in the edge network can be used as an agent to obtain the available resources and task unloading information of the edge network topology through the perceptron installed in physical network. In a certain state, the environment state after agent performs the action is only related to current state, independent of the historical state, and has no aftereffect. Therefore, the edge task unloading problem can be expressed as an MDP model. The problem of edge task offloading based on MDP is presented as follows:

State space S: for $s_l \in S$, and $s_l = \langle U_{cpu}(l), U_{bw}(l) \rangle$, which indicates the bandwidth resource utilization of CPU and physical link of each edge server in the edge network when l edge task has been unloaded.

Action space A: $a_l \in A$ means that the agent selects an edge server in the edge network according to the specific strategy and the current state s_l, then deploys the $l+1$ task in the task queue.

Action execution function: $step(s_l, a) = \langle r_l, s_{l'}, l' \rangle$, This function represents the immediate reward r_l, subsequent state $s_{l'}$ and the number of edge tasks that have been successfully unloaded. If edge task satisfies the constraints (7)–(10) in the process of unloading, it means that the task can be successfully unloaded to the edge network. Reward function $Reward(s_l, a_l)$ represents the immediate reward obtained by the unloading action a_l in the state s_l. The goal of this paper is to reduce the energy consumption in the edge network, therefore, reward function in this paper can be expressed as follows:

$$Reward(s_l, a_l) = N - p_{total} \tag{12}$$

In above formula, the source of action a_l is from policy π. π is a mapping from state space s_l to action space a_l:

$$a_l = \pi(s_l) \tag{13}$$

The optimization goal of the MDP model established in this paper is to get an optimization strategy, it maximizes the goal of reinforcement learning-the expectation of cumulative return value:

$$\pi^* = \arg\max_{\pi} E\left[\sum_{l=0}^{\infty} \gamma^t \cdot \mathrm{Re}ward(s_l, a_l)\right] \tag{14}$$

γ^t is discount factor, and its value decreases with time.

4.2 PPO Algorithm Framework

Because the environment of edge computing network is complex and changeable, to learn in this challenging environment, it is necessary to use a reliable and scalable intelligent algorithm [15]. Because PPO algorithm guarantees stability by binding the range of parameter update to the trust area, this paper considers using this algorithm to complete the unloading of edge tasks [16].

PPO algorithm is a deep reinforcement learning algorithm based on actor-critic framework. Its architecture contains two actor networks, Actor 1 and Actor 2. Actor 1 represents the latest policy π, which guides the agent to interact with the environment. Critic evaluates the current strategy according to the reward, then updates the parameters in the critic network through back propagation of the loss function. Actor 2 stands for the old strategy π_{old}, after the agent trains a certain number of steps, it uses the parameters in Actor1 to update Actor2. Repeating above process until PPO algorithm converges, we get a trained edge task unloading model based on Actor-Critic framework.

$$\theta_{new} = \theta_{old} + \alpha \nabla_{\theta} J \tag{15}$$

θ_{old} and θ_{new} respectively represent the strategy parameters before and after the update, α represents the update step size, and $\nabla_{\theta} J$ is the objective function gradient. The key of the policy gradient algorithm is the update step size. If the update step size is not selected properly, the algorithm may collapse. PPO decomposes the return function into the return function corresponding to the old strategy plus other items. Once the other items in the new strategy are greater than or 0, the return function can be guaranteed to be monotonous.

$$\eta(\tilde{\pi}) = \eta(\pi) + Es_0, \quad a_0 \ldots \tilde{\pi}\left[\sum_{t=0}^{\infty} \gamma^t A_{\pi}(s_t, a_t)\right] \tag{16}$$

$A_{\pi}(s_t, a_t)$ is the dominance function. The calculation of the dominance function is shown in the following formula.

$$A_{\pi}(s_t, a_t) = Q_{\pi}(s, a) - V_{\pi}(s) \tag{17}$$

PPO algorithm optimizes the parameter θ to satisfy the following equation.

$$\max_{\theta} E\left[\frac{\pi_{\theta(a|s)}}{\pi_{\theta_{old}(a|s)}} A_{\theta_{old}}(s, a)\right] \tag{18}$$

where $\pi_{\theta(a|s)}$ is the probability of using strategy π to take action a in state s, and the above formula should satisfy $D_{KL}^{\max}(\theta_{old}, \theta) \leq \eta$, $D_{KL}^{\max}(\theta_{old}, \theta) \leq \eta$ represents the maximum divergence between the old policy parameter and the new policy parameter.

$$L^{KLPEN}(\theta) = E_t\left[\frac{\pi_{\theta(a|s)}}{\pi_{\theta_{old}(a|s)}} A_{\theta_{old}}(s, a) - \beta KL[\pi_{\theta_{old}}, \tilde{\pi}_{\theta}]\right] \tag{19}$$

The policy update formula of PPO is shown in the above formula, but there is a problem that the super parameter β is difficult to determine. PPO considers another method to limit the update step size of the policy.

$$r_t(\theta) = \frac{\pi_{\theta(a_t|s_t)}}{\pi_{\theta_{old}(a_t|s_t)}} \tag{20}$$

When policy does not change, $r_t(\theta) = 1$. and PPO algorithms consider using $clip()$ to limit the similarity between the old and new policies. The improved policy update method is shown in the following formula.

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta)A_t, clip(r_t(\theta)), 1-\epsilon, 1+\epsilon)A_t] \tag{21}$$

4.3 PPO Algorithm Implement

To optimize the energy consumption in the process of edge task unloading, algorithm designed in this paper mainly includes the following three modules: 1) construction of edge network environment and parameter setting; 2) edge task unloading model training; 3) output of energy consumption aware unloading scheme.

As mentioned in 4.2, the actor network of PPO algorithm designed in this paper is composed of two neural networks, Actor 1 and Actor 2. Actor 1 guides the agent to interact with the environment, obtains transfer samples and caches them. The policy parameters in Actor 2 represents the old policies. After a period of iteration, the parameters in Actor 1 will be used to update the parameters in Actor 2. The critic network consists of a neural network. Training steps of unloading model are as follows:

Step 1: Input current state into Actor 1, and the agent selects an action based on $a_l = \pi(s_l)$. After repeating the above process, the agent continuously interacts with the edge network for T time steps, collects the historical interaction information and caches it.

Step 2: Use (17) to calculate the advantage function of each time step.

Step 3: Use the following formula to calculate the loss function of the critic network, update the critic network parameters ϕ according to the function back propagation.

$$L(\phi) = -\sum_{t=1}^{T}\left(\sum\nolimits_{i>t}\gamma^{i-t}r_i - V_\phi(s_t)\right)^2 \tag{22}$$

Step4: Update the parameters of Actor1 by (17) and (21).

Step5: Repeat step 4. After a certain step, use the network parameters in Actor1 to update the parameters of Actor 2.

Step 6: Repeat steps 1–5.

Based on above analysis, the edge task unloading algorithm based on PPO is described in the following table.

5 Performance Evaluation

5.1 Simulation Setting

In this section, two kinds of network topology are used to verify the algorithm proposed in the previous chapter. First network topology is composed of five edge servers and eight physical links. Second physical topology consists of 8 edge servers and 12 physical links. To verify the energy optimization performance of the task offload algorithm proposed in this paper, the energy consumption of 10, 15, 20 … 60 task queues on the terminal device is simulated and measured. We assume that three types of terminal tasks need to be unloaded at the edge network, and detailed parameters settings of each task are shown in the Tab. 1. We also assume that the bandwidth of each physical link is 1000 Mbps, and the available computing

resources of each edge server are 9 or 10 CPUs. Task types in each task queue are randomly selected from the above three types of tasks and consist of at most three tasks. The number of tasks in each task queue is evenly distributed between 2 and 3. In addition, this paper also assumes that the required bandwidth of all task queues is evenly distributed in the range of [40, 50 Mbps]. The startup energy consumption and full load operation energy consumption of the server and physical link are set to 170 and 800 W, 100 and 600 W respectively. The task parameters used for simulation are shown in Tab. 2.

Table 1: Edge task offloading algorithm

Algorithm: Edge task unloading algorithm based on PPO
Input: Initial state of the edge network and tasks queue to be unloaded
Output: Edge task offloading scheme
1. // 1) Build edge network environment and set parameters
2. Initialize the edge network environment and queue to be unloaded, and set the super parameters in PPO algorithm
3. // 2) PPO algorithm based on AC framework is trained
4. Initializes the interactive cache, which is used for historical information collection
5. for $episode \in \{1, \ldots M\}$
6. for $t \in \{1, \ldots T\}$
7. According to the execution action $a_l = \pi(s_l)$, reward r_l and new status s_{l+1} are obtained, and the historical $\{s_l, a_l, r_l\}$ interaction information is collected
8. end for
9. for $t \in \{1, \ldots T\}$
10. Update Actor1 network parameters according to (17) and (21)
11. Update Critic network parameters according to (22) back propagation
12. if $t\%circle == 0$
13. Update Actor 2 with the parameters in Actor 1
14. end if
15. end for
16. end for
17. // 3) Output the unloading scheme of task queues to be unloaded
18. while $q\_task.length! = 0$
19. Output the unloading position (Server ID) of edge task
20. Actor1 chooses actions according to the current policy and moves to the next state
21. $s \leftarrow s'$
22. end while

This paper uses the following four indicators to evaluate the proposed algorithm: 1. **Total energy consumption of the network**: including energy consumption when processing terminal tasks and energy consumption in the communication process. 2. **Number of CPUs**: the total number of CPUs consumed when the offload task is executed at the edge network. 3. **Physical network bandwidth**: the total

bandwidth consumed by the physical network when the edge network transmits inter task traffic. 4. **Offloading success rate**: the percentage of terminal tasks successfully unloaded to the edge server.

Table 2: Relation between custom throughput of task and CPU

Task type	Throughput (Mbps)	CPU demand
1	100	1
2	80	1
3	50	4

To verify the effectiveness of the proposed algorithm. In this section, the algorithm (PPO_EM) proposed in the previous section is compared with random algorithm and task unloading algorithm based on PPO but without considering start up energy consumption (PPO_NEM).

5.2 Simulation Results

PPO_EM is implemented on a computer equipped with inter (R) core i5-9300 h and 16 g memory. The program running environment is Python 3.7.4, Tensorflow 1.15.0; Fig. 2 shows the convergence of PPO_EM in the training process. During training process, the number of task queues to be deployed is set to 80, learning rate of actor network and critical network are set to 0.0001, and reward discount coefficient is set to 0.9. The parameters of Actor 2 are updated with the network parameters in Actor 1 every 15 steps. As can be seen from Fig. 2 at the beginning of training, the results of PPO_EM algorithm fluctuate due to the randomly selected task deployment scheme, but with the increase of training times, reward function gradually converges to the optimal value at about 150 steps.

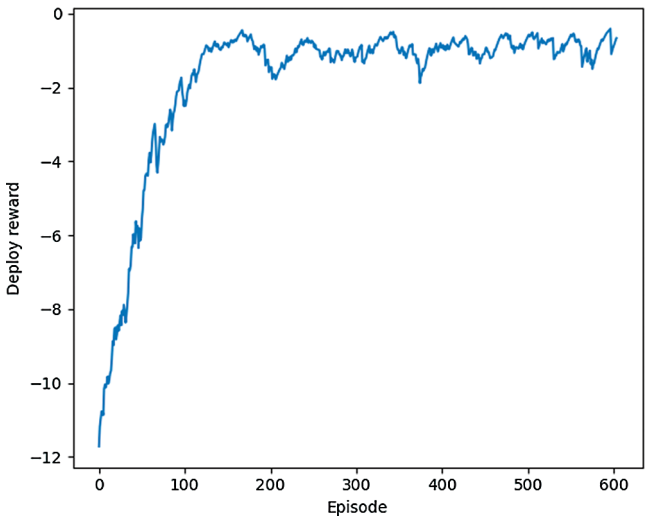

Figure 2: Reward value with the number of iterations

Energy Consumption: Figs. 3a and 3b show the network energy consumption when the task unloading strategy is executed in topology 1 with 5 nodes, 8 physical links, and topology 2 with 8 nodes and 12 physical links, respectively. Compared with random algorithm, algorithm designed in this paper can save 22.69% energy on average when executing task unloading. The reasons are as follows: Random algorithm only considers the network topology, does not consider the factor of network energy consumption. When

building the MDP based optimization model, the algorithm designed in this paper designs the reward function of each step from the perspective of energy consumption, realizes the joint optimization of server energy consumption and physical link, and minimizes the total energy consumption in the process of task unloading. Compared with the PPO_NEM which does not consider the power consumption of power on, the algorithm designed in this paper only starts when the task is deployed on the corresponding edge server, so it is better than the two comparative algorithms in energy consumption.

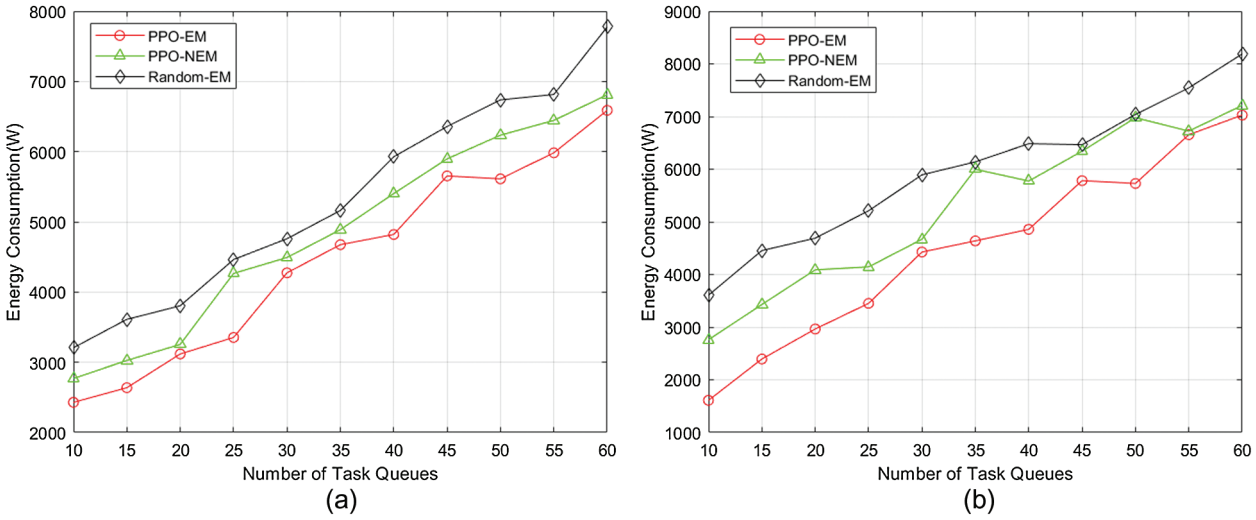

Figure 3: a: Total amount of energy consumption with 5 nodes, 8 physical links b: Total amount of energy consumption with 8 nodes and 12 physical links

Offloading Success Rate: Figs. 4a and 4b show the success rate of task unloading in topology 1 with 5 nodes, 8 physical links, and topology 2 with 8 nodes and 12 physical links. It can be seen from figures that with the increase of the number of task queues to be deployed, the success rate of unloading tasks based on the three algorithms decreases. Because the number of tasks in the queue is randomly distributed between 2 and 3, it is possible that the deployment task queue increases, but the overall required resources decrease. This can explain the phenomenon that the unloading success rate increases slightly with the increase of task queue in the simulation diagram. It can be seen from the figure that the task unloading algorithm based on PPO designed in this paper has a better unloading success rate than the random algorithm. The reason may be that the reuse rate of the same type of tasks in the random algorithm is low, the edge network resources are limited, and the repeated deployment of tasks consumes more computing and bandwidth resources, thus affecting the task unloading success rate.

Consumed Bandwidth: Fig. 5 shows total link bandwidth cost when unloading tasks in the network topology with the number of physical nodes being 5 and the number of physical links being 8. It can be seen from the above figure that with the increase of the number of task queues to be unloaded, the total amount of bandwidth consumed in the network is also increasing. What's more, it can be seen from the figure that the algorithm designed in this paper consumes the least network bandwidth Computer algorithm consumes the most network bandwidth on the edge network. The reasons are as follows: Although the random algorithm uses the shortest path connection when connecting the edge servers deployed with adjacent tasks, the edge servers are randomly selected during the unloading process. Therefore, compared with the PPO_EM algorithm which considers the link energy consumption, the algorithm designed in this paper has less hops and less network bandwidth consumption when implementing the routing between adjacent tasks.

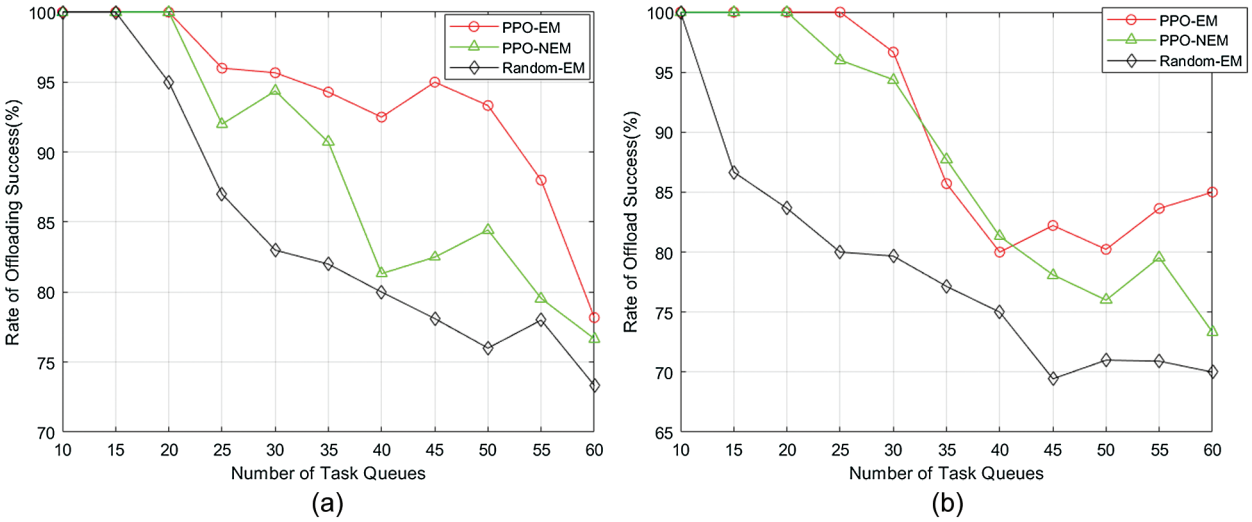

Figure 4: a: Offloading Sucess rate with 5 nodes, 8 physical links b: Offloading Sucess rate with 8 nodes and 12 physical links

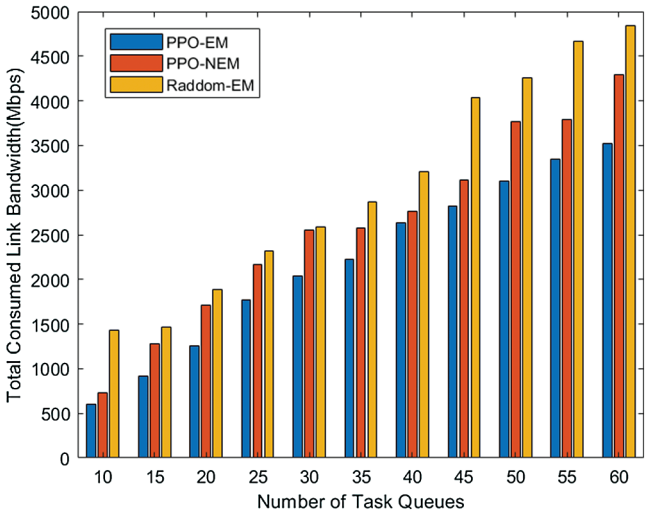

Figure 5: Total amount of consumed bandwidth

Consumed CPUs: Fig. 6 shows the total number of CPU consumed by the edge network when the task is unloaded in the physical topology with 5 physical nodes and 8 physical links. It can be seen from Fig. 6 that when the three offload strategies are implemented in the edge network, the number of CPU consumed in the network increases with the number of task queues to be deployed. From the overall trend of CPU consumption, the CPU consumption of the task offload strategy designed in this paper is slightly better than that of the algorithm without considering power consumption, and it is far better than the algorithm based on random policy for task unloading. The reasons are as follows: because the random algorithm randomly selects the edge servers to be deployed, the same type of tasks in different queues need to be deployed repeatedly. In contrast, PPO_EM improves the utilization of the same task type by aggregating task queue requests. Therefore, the random offload strategy consumes more CPU.

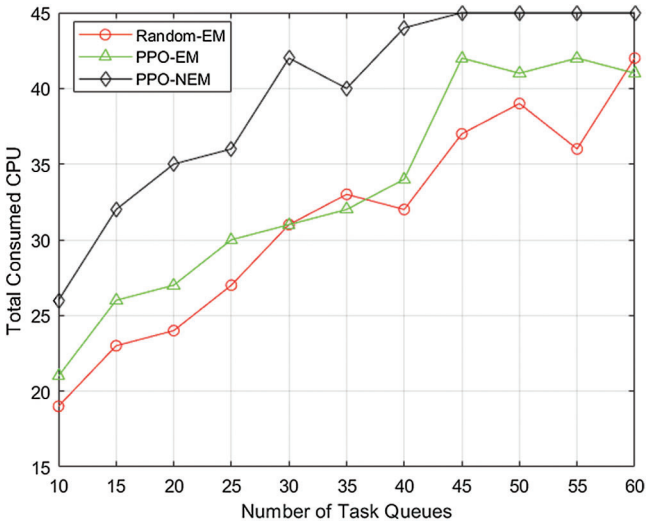

Figure 6: Number of consumed CPUs

6 Conclusion

In this paper, we focus on the optimization of energy consumption of edge server in the process of task unloading. To improve the accuracy of server energy consumption evaluation, we first construct a server energy consumption model including both startup energy consumption and processing energy consumption, then we describe the model as an optimization problem model for energy consumption optimization. Then, a task unloading strategy based on PPO is proposed to solve the approximate optimal task unloading scheme. Simulation results shows that compared with the random algorithm, the proposed algorithm can save 22.69% energy on average.

Funding Statement: This work was supported by State Grid Corporation of China science and technology project "Key technology and application of new multi-mode intelligent network for State Grid" (5700-202024176A-0-0-00).

Conflicts of Interest: We declare that we have no conflicts of interest to report regarding the present study.

References

[1] H. Wang, J. Yong, Q. Liu and A. Yang, "A novel GLS consensus algorithm for alliance chain in edge computing environment," *Computers, Materials & Continua*, vol. 65, no. 1, pp. 963–976, 2020.

[2] H. Zhang, G. Chen and X. Li, "Resource management in cloud computing with optimal pricing policies," *Computer Systems Science and Engineering*, vol. 34, no. 4, pp. 249–254, 2019.

[3] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[4] N. R. Sivakumar, "Stabilizing energy consumption in unequal clusters of wireless sensor networks," *Computers, Materials & Continua*, vol. 64, no. 1, pp. 81–96, 2020.

[5] A. Abdelnasser, E. Hossain and D. I. Kim, "Clustering and resource allocation for dense femtocells in a two-tier cellular OFDMA network," *IEEE Transactions on Wireless Communications*, vol. 13, no. 3, pp. 1628–1641, 2014.

[6] J. Oueis, E. C. Strinati and S. Barbarossa, "The fog balancing: Load distribution for small cell cloud computing," in *2015 IEEE 81st Vehicular Technology Conf.*, Glasgow, UK, pp. 1–6, 2015.

[7] J. Rubio, A. Pascual-Iserte, J. del Olmo and J. Vidal, "User association for load balancing in heterogeneous networks powered with energy harvesting sources," in *2014 IEEE Globecom Workshops*, Austin, TX, USA, pp. 1248–1253, 2014.

[8] H. M. Wu, Q. S. Wang and K. Wolter, "Tradeoff between performance improvement and energy saving in mobile cloud offloading systems," in *2013 IEEE Int. Conf. on Communications Workshops*, Budapest, Hungary, pp. 728–732, 2013.

[9] S. Sardellitti, G. Scutari and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[10] L. Chen, S. Zhou and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[11] J. J. Han and Q. H. Li, "Dynamic power-aware scheduling algorithms for real-time task sets with fault-tolerance in parallel and distributed computing environment," in *19th IEEE Int. Parallel and Distributed Processing Symp.*, Denver, CO, USA, pp. 10–14, 2005.

[12] Y. Wang, M. Sheng, L. Wang, Y. Zhang, Y. Shi *et al.,* "Energy-optimal partial computation offloading using dynamic voltage scaling," in *2015 IEEE Int. Conf. on Communication Workshop*, London, UK, pp. 2695–2700, 2015.

[13] Y. Wang, M. Sheng, X. Wang, L. Wang and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[14] Y. Wang, M. Sheng, X. Wang and J. Li, "Cooperative dynamic voltage scaling and radio resource allocation for energy-efficient multiuser mobile edge computing," in *2018 IEEE Int. Conf. on Communications*, Kansas City, MO, USA, pp. 1–6, 2018.

[15] N. Yuan, C. Jia, J. Lu, S. Guo, W. Li *et al.,* "A DRL-based container placement scheme with auxiliary tasks," *Computers, Materials & Continua*, vol. 64, no. 3, pp. 1657–1671, 2020.

[16] L. Li, Y. Wei, L. Zhang and X. Wang, "Efficient virtual resource allocation in mobile edge networks based on machine learning," *Journal of Cyber Security*, vol. 2, no. 3, pp. 141–150, 2020.