Reinforcement Learning Based MEC Architecturewith Energy-Efficient Optimization for ARANs

Qiang He
Northeastern University, Shenyang, China

Yingjie Lv
Northeastern University, Shenyang, China

Li Zhen
Xi'an University of Posts and Telecommunications, Xi'an, China

Keping Yu
Waseda University, Tokyo, Japan

ICC 2022 - IEEE International Conference on Communications | 978-1-5386-8347-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICC45855.2022.9839063

***Abstract*—Aerial Radio Access Networks (ARANs) are used to connect aerial nodes (such as satellites, aircraft, floating balloons) and ground infrastructures, which enables a global network coverage and provides a wide range of high-quality network services. At present, extensive researches are to integrate it with Mobile Edge Computing (MEC), to achieve more efficient data computing, data storage, and cache. In this paper, we primarily focus on exploring the edge computing architecture integrated with ARANs. Since the existing MEC architecture is not deeply integrated with ARANs, we propose the scenario of a complete four-tier MEC architecture that allows MECs and ARANs to collaborate effectively. Besides, for environmental protection and cost reduction, we propose a Q-learning algorithm based on the improved $\epsilon - greedy$ model to complete the MEC server selection and resource allocation. Finally, the simulation results are compared with other benchmark methods, and the effectiveness of the proposed method is proved. The energy consumption of the proposed method is significantly reduced.**

***Index Terms*—ARANs, MEC, Computation Offloading, Energy Consumption, Q-learning**

I. Introduction

With the development of software technology, computing-intensive applications have been developed to a large extent. Meanwhile, many mobile devices consume a large of computing energy. For example, the total number of Steam games in the U.S. has reached 50,046 and is still growing [1]. The existing mobile device computing resources are extremely limited, and hardly meet the user's calculation requirements. Recently, remote computing offloading has been widely studied to achieve the efficient utilization of computing resources. While remote computing offloading is possible over the existing Radio Access Network (RAN), this is subject to long propagation delays, and is difficult to provide the same quality of service to areas with inadequate coverage. The existing conditions are difficult to meet the requirements for these computationally dense applications [2].

Aerial Radio Access Networks (ARANs) in 6G networks allow for worldwide coverage, which can solve the problem of inadequate coverage of existing wireless networks. At the same time, MEC has emerged as an open, easily scalable, and collaborative ecosystem. It is deployed closer to the User Equipment (UE), with lower propagation latency and greater flexibility [3]. Users can choose to migrate tasks to nearby MEC servers for computation to reduce computation latency. At present, one of the research hot spots is to deeply integrate ARANs with MEC so that the UEs can choose to offload tasks to nearby MEC servers or ARANs further away, depending on latency needs. Thereby, different computing requirements of users can be satisfied by MEC.

Until now, some MEC architectures have been proposed by some scholars. For example, Li et al. [4] proposed a general edge computing architecture composed of terminal layer, MEC layer and cloud layer. Zhou et al. [5] proposed a decentralized distributed MEC architecture, based on multilevel a distributed data center to provide a business application system with unified could-based operation bearing environment, support service capabilities. Chen et al. [6] described a Multi-Access MEC architecture that uses the MEC as an open wireless access platform, allowing authorized third parties to access and use storage, network, computing and other resources. But these architectures lack integration with ARANs. In this paper, we explore a migration computing architecture of deep convergence of MEC and ARANs, dividing the functions of MEC servers and ARANs networks to enable greater collaboration between them. We expect this architecture to better serve computationally intensive applications.

In addition, how to reduce the energy consumption during the calculation offloading is also a hot issue for many experts and scholars. According to [7], global emissions are still up to 400 Mt CO2 in 2021. At present, the corresponding energy consumption model has been proposed, which divides the total

Qiang He, Yingjie Lv are with College of Medicine and Biological Information Engineering, Northeastern University, China (Email: heqiang@bmie.neu.edu.cn, csharper@yeah.net). Li Zhen is with Shaanxi Key Laboratory of Information Communication Network and Security (Email: lzhen@xupt.edu.cn). Keping Yu is with Global Information and Telecommunication Institute, Waseda University, Tokyo, Japan, and also with Shaanxi Key Laboratory of Information Communication Network and Security, Xi'an University of Posts and Telecommunications, Xi'an, China (Email: keping.yu@aoni.waseda.jp).

energy consumption calculated by offloading into the energy consumption in the process of resource transmission and the period of resource computing. The energy consumption can be reduced by optimizing the number of MEC servers and the selection of transmission channels [8]. Considering the need for lower response time for some of the user's tasks, latency constraints have to be taken into account. Haawadalla et al. [9] uses PSO (Particle Swarm Algorithm) to solve energy consumption optimization problems for real-time systems. However, the research does not involve the time delay constraint, and PSO can not be proved to converge, and it is more likely to fall into local optimality. Bai et al. [10] introduced DNN (deep neural network) to solve the energy consumption optimization problem of MEC. In fact, DNN training requires sufficient datasets and can only be used for classification and regression problems. For decision problems where the MEC server is used as a computational offload target, the results must be further analyzed and processed to complete. Therefore we have to establish a method that addresses the decisional problem of computational offloading and that can avoid falling into local optima to some extent.

To address the above problems, we study the MEC architecture for ARANs and propose a Q-learning algorithm to achieve MEC server selection and resource allocation. The main contributions of this paper are summarized as follows:

1) We propose a four-tier MEC architecture of ARANs and add the scheduling module of the MEC server, to reduce the queuing delay of computing offloading.

2) We use the Q-learning algorithm based on the improved $\epsilon - greedy$ model to perform computing offloading and complete the MEC server selection and resource allocation.

3) Through simulation experiments, we show the defectives of our proposed algorithm on energy consumption.

The remainder of this paper is organized as follows. In Section II, we describe the MEC architecture for ARANs. In Section III, we develop a minimized energy consumption model with time delay constraints. In Section IV, we propose a Q-learning algorithm based on the improved $\epsilon - greedy$ model for offloading and resource allocation. In Section V, we carry out simulation experiments and analyze the experimental results. Finally, Section V concludes this paper.

II. Architecture

As shown in Fig. 1, a MEC architecture for ARANs is devised, which is composed of four parts: data upload layer, data processing layer, data storage layer, and data application layer.

The data upload layer mainly consists of a communication base station and a series of various terminal devices, such as sensors, vehicles in motion, wireless users carrying kinds of wearable devices, smartphones, computer devices, and gaming devices. These terminal devices are used to collect information, such as data that users need to request for processing, real-time monitoring services, etc. If these data or services cannot be processed locally, they will be offloaded via the base station to the MEC server for computing.

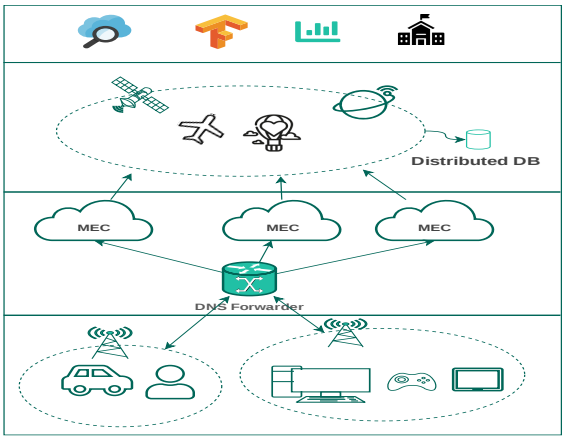

Fig. 1. MEC Architecture for ARANs

The data processing layer consists of multiple MEC servers and a DNS forwarder for MEC servers. The purpose of placing the MEC server is to compute and process the data from user types of equipment UEs, but such computing tasks will generate a lot of power consumption. We reduce the power consumption by optimizing the computational offloading to the MEC server. For this, we introduce a DNS forwarder. The DNS forwarder, which is reflected in the migration tasks needs to conduct DNS redirection and select the MEC server with the lowest energy consumption.

The data storage layer is designed to be the core layer of ARANs, and it is also composed of aircraft, satellites, and many other non-ground pieces of devices. Based on the load status of the MEC server, such devices can handle some difficult tasks for the MEC server, and schedule them to achieve a good load balancing.

The data usage layer is designed to observe and apply the data. On the one hand, users can find their related data for planning and decision-making. On the other hand, after the removal of sensitive private information in the data, the data source for research institutions can be provided, such as establishing data sets, training on a deep neural network with TensorFlow, carrying on the effective public big data statistics, and forecasting user demand.

III. System Model

This is a multi-access MEC system with multiple UEs. To improve the transmission rate of UEs to MEC servers, we consider an OFDMA network, in which the wireless channel is divided into several subcarriers. We define it as $n \in N = \{1, 2...N\}$. In addition, we define the UE set as $i \in U = \{1, 2...U\}$, and assume that each UE at least has one service to process. We define all services as $s \in S = \{1, 2...S\}$. Besides, we consider that there deploys $m \in M = \{1, 2...M\}$ MECs. We use a quaternion $T(D_s, W_s, T_s^{max}, R_s)$ to represent the task where D_s is the data size of the task, W_s is the count of CPU cycles per bit, T_s^{max} is the maximum user-requested time delay, and R_s is the priority of the task, which should

Algorithm 1: Computational migration policy

Input: $T(D_s, W_s, T_s, R_s)$
Output: Computing strategy s

Initialize s;
if $R_s = 1 \, and \, \frac{D_s * C_s}{K_s} < T_s$ **then**
  s=Local computing;
else if $R_s = -1$ **then**
  s=Calculation on the ARANS network;
else if then
  s=Calculation on the MEC servers according to priority;
end
return s

be an integer with a larger value indicating a higher priority. Special, we use $R_s = -1$ to show that there is no priority for the current task.

A. Computing strategy

To balance the computational priority and computational latency, as well as to fully utilize the resources of the MEC servers and ARANs network, we propose the computational migration policy algorithm, which is presented in Algorithm 1. Local computation is selected when the computation task satisfies $R_s = 1$, i.e., the task has general priority, and the local device can meet the computation latency requirements. In addition, When tasks have no priority requirements, we choose cloud computing to save valuable MEC server resources. Finally, tasks that have priority are selected to be computed on the MEC server according to the priority level to obtain lower computational latency.

B. Computing model

We use K_s^{local} to represent CPU frequency of local computing, and use C_s to represent CPU clock cycles per bit of data spent locally. We could get the local computation time for service s from [11]:

$$T_{s,m}^{local} = \frac{D_s C_s}{K_{s,m}^{local}} \tag{1}$$

According to [12], the energy comsumption of service s is written as:

$$E_s^{local} = k_0 (K_s^{local})^2 D_s C_s \tag{2}$$

where $k_0 = 1.0 * 10^{-28}$ is a energy factor of different CPUs.

In the same way, we could define the time when MEC m deals with service s like this:

$$T_s^{MEC} = \frac{D_s C_s}{K_s^{MEC}} \tag{3}$$

where $K_{s,m}^{MEC}$ is the CPU frequency of MEC m. Next, we define the energy consumption of service s like this:

$$E_s^{MEC} = k_0 (K_s^{MEC})^2 D_s C_s \tag{4}$$

Since we are using an OFDMA network, the radio channel will be divided into several subcarriers [13], so the data rate of the service from the UEs to the MEC servers can be written as:

$$R_{s,m} = B \sum_{i=1}^{N} w_{s,m,n} log_2^{(1+\frac{G_{s,m,n} P_{s,m,n}}{\sigma^2})} \tag{5}$$

where B is the channel bandwidth, and $w_{s,m,n} = \{0, 1\}$ indicates whether to choose the subcarriers n. When $w_{s,m,n} = 0$, the subcarrier is selected. Otherwise, the subcarrier is not selected. σ^2 is noise, $P_{s,m,n}$ is the transmitted power from service s to the MEC m, and $G_{s,m,n}$ is the channel gain of the wireless transmission channel, defined as follows:

$$G_{m,s,n} = \frac{\alpha_n}{(H_{s,m} + R_{s,m})^2} \tag{6}$$

where $R_{s,m}$ is the horizontal distance from service s to MEC m, and $H_{s,m}$ is the vertical distance. $\frac{1}{(H_{s,m}+R_{s,m})^2}$ and α_n are large-scale path loss component and small-scale fading component [14], where α_n is defined as follows:

$$\alpha_n = g_0 G_0 \tag{7}$$

So we can get the delay from service s to MEC server m:

$$T_{s,m} = \frac{D_s}{R_{s,m}} \tag{8}$$

Now we can get the energy consumption needed to transmit task s:

$$E_{s,m}^{trans} = P_{s,m} T_{s,m} \tag{9}$$

The total energy consumption is the sum of the transmission energy consumption and the calculated energy consumption of the MEC server:

$$\begin{aligned}
&min(u, w, P, K) \\
&= \sum_{s=1}^{S} \sum_{m=1}^{M} u_{s,m} E_{s,m}^{r} \\
&= \sum_{s=1}^{S} \sum_{m=1}^{M} u_{s,m} (k_0 k_{m,s}^2 D_s C_s + P_{s,m} T_{s,m}) \\
&st. \\
&C1 : u_{s,m} = \{0, 1\} \\
&C2 : \sum_{m=1}^{M} u_{s,m} = 1 \\
&C3 : w_{s,m,n} = \{0, 1\} \\
&C4 : \sum_{s=1}^{S} \sum_{m=1}^{M} w_{s,m,n} = 1 \\
&C5 : T_{s,m}^{MEC} + T_{s,m}^{trans} < T_s^{max}
\end{aligned} \tag{10}$$

where constraint $C1$ is an on/off switch, which represents whether to offload service s to MEC server m. When $u_{s,m}$=1, service m is offloaded to MEC m, otherwise the service will not be offloaded on MEC m. Constraint $C2$ ensures that every service can be only offloaded on one MEC. Constraint $C3$ represents whether service s is to select the current infinite transmission subcarrier. Constraint $C4$ guarantees that every subcarrier can assign at most one service. Constraint $C5$

guarantees that the delay of every task is lower than the user needs.

We note that this is an MINLP problem, which is NP-hard in general. Although there are traditional heuristic algorithms such as PSO that obtain possible optimal solutions. However, such schemes are more prone to blind search leading to falling into local optimum. Therefore, in this paper, we try to introduce reinforcement learning to solve this problem, using the Q-learning algorithm based on an improved $\epsilon - greedy$ model to avoid falling into local optimality.

IV. Proposed Algorithm

We can use a quaternion $E =< S, A, P, R >$ to represent the decision process of computing offload. E is the environment in which the agent learns, A is the action made by the agent, P is the probability of the agent transferring from one state to another, and R is the reward the agent returned from E [15].

Actions: The service needs to select target server m for computing offloads and channel n to transport. So we should set A_t like this:

$$A(t) = \{x_{00}, x_{01}...x_{M1}...x_{MN}\} \tag{11}$$

States: We define the states using the offloaded MEC number at time t as follows:

$$S(t) = \{w_1, w_2...w_M\} \tag{12}$$

where w_m represents the offloading decision of the service.

Reward: The reward at time t is defined as follows:

$$R_(t) = \frac{1}{E(A(t), S(t))} \tag{13}$$

There are two solutions for the agent to select actions: exploration and exploitation [16], where exploration implies taking a random action and exploitation implies using an already existing action. To balance the count of explorations and exploits, ϵ is usually used to denote the probability of exploration. To search faster, ϵ should be set large enough at the early stage of training, and ϵ should be gradually reduced to avoid falling into local optimality, thus we can obtain the dynamic ϵ like this:

$$\varepsilon = 1 - \frac{1}{1 + e^{-\gamma(i - T\ )}} \tag{14}$$

where T is the total cycles of training, i is used to represent the current number of cycles. The possibility of exploration will gradually decrease with the increase of training times. γ is used to represent the probability of exploration. Next, we use $M(s_t, a_t)$ to represent average rewards:

$$M(s_t, a_t) = \frac{R(s_t, a_t) - M(s_t, a_t)}{K(s_t, a_t)} + M(s_t, a_t) \tag{15}$$

We use K to save the selected counts of the action a_t. The improved $\epsilon - greedy$ model is showed in Algorithm 2.

Then we should update the reward in Q-table according to the action we have selected. It writes like this [17]:

$$Q(S_t, A_t) += \alpha(R_t + \gamma maxQ(S_t\prime, A_t) - Q(S_t, A_t)) \tag{16}$$

Algorithm 2: Improved $\epsilon - greedy$ model

Input: Current state $s_t \in S$, action set A, reward function R

Output: Selection strategy A'

Initialize the ordered set A';
for *i=1:T* **do**
  $K(s_t, a_t) = \{0\}$;
  $\epsilon = 1 - \frac{1}{1+e^{-\gamma(i-T)}}$;
  if $rand() < \epsilon$ **then**
    a_t=random choice from A;
  else
    Perturb indexes with the same maximum value;
    Select action a_t with the highest reward in state s_t;
  end
  Add a_t to A';
  $M(s_t, a_t) = M(s_t, a_t) + \frac{(R(s_t, a_t) - M(s_t, a_t))}{K(s_t, a_t)}$;
  $K(s_t, a_t) = K(s_t, a_t) + 1$;
end

Algorithm 3: Q-learning based $\epsilon - greedy$ model

Input: State space S, action Space A, learning rate α, discount factor γ.

Output: Q-values for every state-action pair.

Initialize $Q(s_t, a_t) = \{0\}$;
foreach *each episode* **do**
  Initialize s_t;
  Choose an action a_t from A use algoritm2.
  Allocate computation resource $K_{s,m}^{MEC}$ and transmission power $P_{s,m,n}$;
  Run action a_t, and get r_t according to (21), (22) and get next s_{t+1};
  Update $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma maxQ(s_t\prime, a_t) - Q(s_t, a_t))$;
  $s_t = s_{t+1}$;
end

where $\gamma \in (0, 1)$ is the reward decay rate, $\alpha \in (0, 1)$ is the learning rate, $S_t\prime$ is the next state of the environment return to agent. Algorithm 3 shows this algorithm in detail.

V. Simulation Results

In this section, we simulate the proposed algorithm, where the parameters are given in Table 1. Here, we set noise σ^2 to $1.0{*}10^{-13}$ (w), set subcarrier bandwidth B to $1.2{*}10^4$ (w), set the channel power gain at the reference distance 1 m g_0 to the interval of $[1.4 * 10^{-4}, 1.4 * 10^{-3}]$, set the data size of each UE to the interval of [1000,2000] (bits), set maximum task deadline T_{max} to 5.5 (ms), all UEs are randomly initialized in the rectangular region of $[0, 420] * [0, 420]$ (m), cpi of MEC servers is set to the interval of $[1.0{*}10^3, 3.0{*}10^3]$ (cycles/bit).

To test the performance of our proposed algorithm, the following algorithms are used as comparison algorithms.

TABLE I
SIMLUATION PARAMETERS.

Parameters	Description	Values
P	Location of UEs	[0,420] × [0,420] m
D_s	Data Size	[1000,2000] bit
go	Channel power gain	$[1.4 * 10^{-4}, 1.4 * 10^{-3}]$
Go	Channel gain coefficient k	2.285
σ^2	noise	$1.0 * 10^{-13}$ w
B	Subcarrier bandwidth	$1.2 * 10^4$ w
C_s	CPI of data	$[1.0 * 10^3, 3.0 * 10^3]$ cycles/bit
T_{max}	Maximum task deadline	5.5 ms

- Random offloading: Each service is randomly assigned MEC servers and transmission subcarriers.
- Round Robin: Each service select MEC servers and subcarriers in turn to perform computation offloading.
- Greedy offloading: Each service offloading to the nearest MEC server.

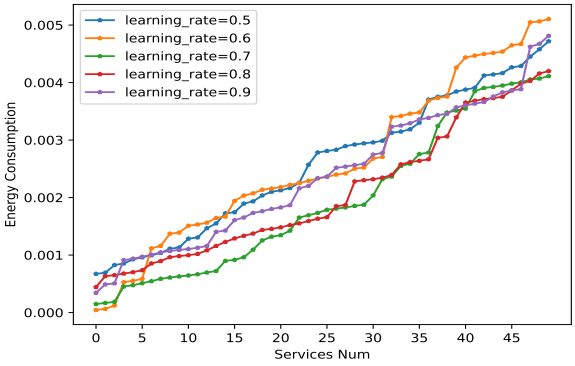

Fig. 2. Energy consumption at different learning rates.

Fig. 2 gives the status of the energy consumption of computational offloading with the increase of the number of services at different learning rates, and we can see that the energy consumption decreases gradually with the increase of the learning rate. When the learning rate increases to 0.7, the energy consumption increases instead. The learning rate usually indicates how well the agent absorbs the content acquired by exploration. If the learning rate is too high, the agent will get more explored content, and it is more difficult to get convergence at this time. If the learning rate is too low, the agent gets very little content of exploration and will prefer to select the content already learned, thus falling into local optimum.

Fig. 3 shows the energy consumption of computational

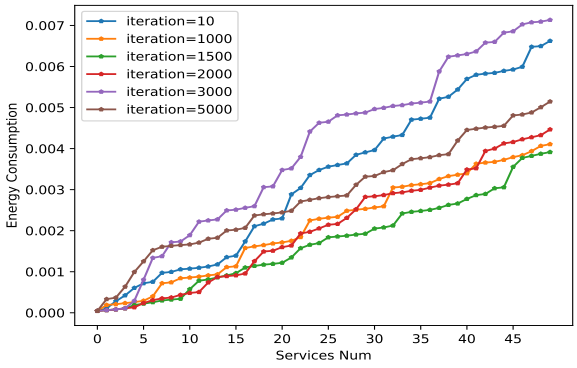

Fig. 3. Energy consumption under different training times.

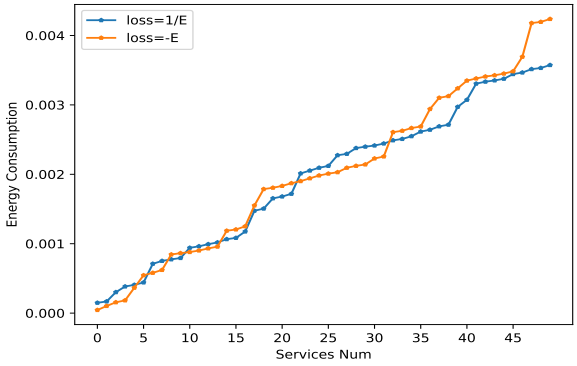

Fig. 4. Energy consumption under different loss functions.

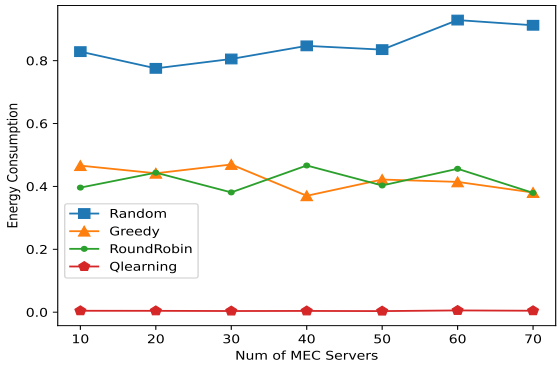

Fig. 5. Energy consumption with the increase of MEC servers.

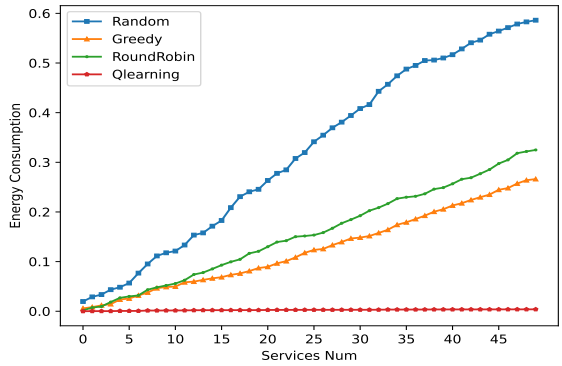

Fig. 6. Energy consumption increases with MEC.

offloading with different training times. We can see that the energy consumption gradually decreases as the training times increase. The reward obtained by the agent gradually increases, and the optimization ability for the target problem is improved. When the training times reach 1500, the energy consumption starts to drop, at which point the optimal solution is missed due to overlearning so the energy consumption shows an increasing trend.

In Fig. 4, we compare the performance of the Q-learning algorithm under different reward functions. We can see that regardless of total energy consumption or average energy consumption, $reward = -E$ is lower than $reward = \frac{1}{E}$.

Fig. 5 compares the energy consumption of different algorithms as the number of MECs increases, and we can see that Q-learning is more stable than other algorithms in terms of energy consumption, and it does not fall into a selection dilemma due to the increase in the number of MEC servers.

Fig. 6 compares the Q-learning algorithm based on improved $\epsilon - greedy$ with Random, Greedy, and RoundRobin algorithms, and we can see that the energy consumption of our algorithm is lower than the other algorithms as the number of services increases, so our algorithm achieves better results. Q-learning can respond to the feedback from the environment and update the selection policy in time to achieve dynamic and timely select the MEC server and transmission channels, so it can achieve better results compared to traditional algorithms.

VI. CONCLUSION

In this paper, we study the MEC optimization problem by considering the energy consumption in ARANs. We establish an MEC architecture for ARANs. Besides, the Q-learning algorithm based on the improved $\epsilon - greedy$ model is used for online learning to optimize the energy consumption of computing offloading. Finally, we verify the effectiveness of the algorithm compared with the baseline algorithms. Our method obtains lower energy consumption than the three algorithms (Random, Round, and Greedy).

ACKNOWLEDGMENTS

This work was supported in part by the Japan Society for the Promotion of Science (JSPS) Grants-in-Aid for Scientific Research (KAKENHI) under Grant JP18K18044 and JP21K17736, in part by the Open Fund of the Shaanxi Key Laboratory of Informaiton Communication Network and Security under Grants ICNS202001, in part by the Natural Science Foundation of China (NSFC) under Grant 61901370, in part by the Key Research and Development Program of Shaanxi under Grant 2021GY-043, and in part by the National Natural Science Foundation of China under Grant 71601104.

REFERENCES

[1] PCGAME, "Steam just reached 50,000 total games listed," 2021. https://www.pcgamesn.com/steam/total-games, Last accessed on 2021-2-12.
[2] F. Zhou, N. Wang, G. Luo, L. Fan, and W. Chen, "Edge caching in multi-uav-enabled radio access networks: 3d modeling and spectral efficiency optimization," *IEEE Transactions on Signal and Information Processing over Networks*, 2020.
[3] N.-N. Dao, Q.-V. Pham, D.-T. Do, and S. Dustdar, "The sky is the edge—toward mobile coverage from the sky," *IEEE Internet Computing*, vol. 25, no. 2, pp. 101–108, 2021.
[4] L. Linzhe, Z. Peilei, and Z. S. Peng CHENG, "Architecture, challenges and applications of edge computing," *Big Data Research*, vol. 5, no. 2, p. 0, 2019.
[5] C. Zhong and X. Yuan, "Distributed cloud service engine based on edge computing," in *Proceedings of the 2018 International Conference on Transportation & Logistics, Information & Communication, Smart City (TLICSC 2018)*, pp. 442–450, Atlantis Press, 2018/12.
[6] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.
[7] International Energy Agency, "Global energy review 2021," 2021. https://www.iea.org/reports/global-energy-review-2021.
[8] M. Wang, S. Shi, S. Gu, N. Zhang, and X. Gu, "Intelligent resource allocation in uav-enabled mobile edge computing networks," in *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, pp. 1–5, 2020.
[9] M. Haawadalla, "Power efficient scheduling scheme based on pso for real time systems," *International Journal of Computer Applications*, vol. 111, no. 4, pp. 24–30, 2015.
[10] K. Bai, S. Liu, and Y. Yi, "High speed and energy efficient deep neural network for edge computing," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, SEC '19, (New York, NY, USA), p. 347–349, Association for Computing Machinery, 2019.
[11] 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.
[12] Y. Ye, X. Bu, Y. Kai, and H. Zhu, "Energy efficient mobile edge computing using joint benders decomposition and distributed dinkelbach algorithm," in *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2018.
[13] X. Yang, X. Yu, and A. Rao, "Efficient energy joint computation offloading and resource optimization in multi-access mec systems," in *2019 IEEE 2nd International Conference on Electronic Information and Communication Technology (ICEICT)*, pp. 151–155, 2019.
[14] L. Wang, P. Huang, K. Wang, G. Zhang, and K. Yang, "Rl-based user association and resource allocation for multi-uav enabled mec," in *2019 15th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2019.
[15] Q. He, X. Wang, Y. Zhao, B. Yi, X. Lu, M. Yang, and M. Huang, "Reinforcement-learning-based competitive opinion maximization approach in signed social networks," *IEEE Transactions on Computational Social Systems*, pp. 1–10, 2021. doi: 10.1109/TCSS.2021.3120421.
[16] R. Sutton and A. Barto, *Reinforcement Learning:An Introduction*. Reinforcement Learning:An Introduction, 1998.
[17] Q. He, H. Fang, J. Zhang, and X. Wang, "Dynamic opinion maximization in social networks," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021. doi: 10.1109/TKDE.2021.3077491.