A Deep Reinforcement Learning based Offloading Scheme in Ad-hoc Mobile Clouds

Duc Van Le* and Chen-Khong Tham†
*Dept. of Computer Science, National University of Singapore
†Dept. of Electrical and Computer Engineering, National University of Singapore
E-mails: anhduc.mta@gmail.com, eletck@nus.edu.sg

***Abstract*—In this paper, we consider the problem of making an optimal offloading decision for a mobile user in an ad-hoc mobile cloud in which the mobile user can offload his computation tasks to nearby mobile cloudlets via a device-to-device (D2D) communication-enabled cellular network. We propose a deep reinforcement learning (DRL)-based offloading scheme which enables the user to make near-optimal offloading decisions by taking into account uncertainties of user's and cloudlets' movements and the cloudlets' resource availabilities. We first propose a Markov decision process (MDP)-based offloading problem formulation which considers the composite states of the user's and cloudlets' queue states and the distance states between the user and cloudlets as the system state space. The objective of the formulated MDP-based problem is to determine the optimal actions on how many tasks the user should process locally and how many tasks to offload to each cloudlet at each observed system state such that the user's utility obtained by task execution is maximized while minimizing the energy consumption, task processing delay, task loss probability and required payment. Then, we use a deep reinforcement learning scheme, called deep Q-network (DQN) to learn an efficient solution for the proposed MDP-based offloading problem. Extensive simulations were performed to evaluate the performance of the proposed offloading scheme. The simulation results validate the effectiveness of the offloading policies obtained by the proposed scheme.**

***Index Terms*—Ad-hoc Mobile Cloud; Computation Offloading; Deep Reinforcement Learning; Deep Q-Network**

I. Introduction

An ad-hoc mobile cloud [1] has been recently introduced as a promising cloud architecture, in which a mobile device user (e.g., smart phones and tablets) can offload its tasks to other nearby mobile devices, called *mobile cloudlets*, in an ad-hoc fashion. By offloading, the mobile user can save its energy and speed up processing-intensive mobile applications. In addition, since the user can utilize direct communication technologies such as Bluetooth and Wi-Fi Direct to communicate with the cloudlet, the offloading latency is lower compared to the case of accessing the traditional centralized resource clouds (e.g., Amazon EC2 and Microsoft Azure) through the backbone network via 3G/4G cellular networks or Wi-Fi access points.

However, making an optimal offloading decision in the ad-hoc mobile cloud has to face challenges due to uncertainties of the user and cloudlet movements and the cloudlets' resource availabilities. For example, offloading may result in higher energy consumption compared to local processing since a large amount of energy may be consumed as a result of retransmissions when the task data is transmitted over a bad connection arising from movements of the user and cloudlets. In addition, offloading may also result in a higher task processing delay arising from the communication delay. The user also has to make a certain payment for the use of the cloudlet's resource and communication bandwidth. Furthermore, the challenge includes not only how to make an optimal "offload or not" decision but also how to distribute the proper number of tasks to cloudlets with different movement and resource availability conditions.

In this paper, we focus on designing an offloading algorithm which can enable the user to make optimal offloading decisions, subject to the dynamics of the system in terms of user and cloudlet behaviors. More specifically, we formulate the offloading problem as a Markov decision process (MDP) which considers the composite states of the user's and cloudlets' queue states and the distance states between the user and cloudlets as a state space. The main objective is to find the optimal actions on how many tasks the user should process locally and how many tasks to offload to each cloudlet such that the user's utility obtained by task execution is maximized while minimizing the energy consumption, task processing delay, required payment and task loss probability. A DRL algorithm, called a deep Q-network (DQN) [2] is used to learn the optimal offloading decision for the MDP-based offloading problem.

In the proposed DQN-based offloading algorithm, the user continuously learns online and improves its offloading decisions through a trial-and-error learning scheme. The state observation information and learning experience are combined in a nonlinear function approximator in the form of a neural network, which is iteratively trained and gives the user the best decision to take at each system state. The DQN algorithm is adopted in our study since it has achieved excellent performance in the high-dimensional decision-making problems which are similar to our formulated offloading decision problem. Furthermore, we evaluate the proposed offloading scheme in various simulations in which an ad-hoc mobile cloud environment is simulated. The simulation results show that the proposed scheme performs well in terms of the effectiveness of the offloading decisions.

In the remainder of this paper, we first review related works in Section II. Section III describes the system model and problem statement. In Section IV, we describe the proposed DRL-based offloading algorithm. Simulation results and analysis

978-1-5386-5979-3/18/$31.00 ©2018 IEEE

are presented in Section V, followed by the conclusion in Section VI.

II. Related Work

A. Computation Offloading in Ad-hoc Mobile Clouds

The problem of computation offloading in a general mobile cloud computing system has been extensively studied in the literature [1], [3]. From the perspective of the user, computation offloading can help to save energy and speed up the computation process. In general, an offloading algorithm can enable the user to make three types of offloading decisions which are local execution, full offloading and partial offloading [3].

Many studies [4], [5], [6] have proposed various offloading algorithms to enable the mobile user to make an offloading decision in the ad-hoc mobile cloud. The general objective is to make optimal offloading decisions such that a maximum number of tasks can be executed while minimizing the energy consumption, delay and other costs, such as communication cost caused by offloading.

For example, Zhang *et al.* [4] proposed an MDP-based offloading algorithm that helps the user decide whether to offload the computation task of the same application phase in a multi-phase mobile application to all nearby cloudlets. Upon making an offloading decision, the user gets the result from the earliest completed mobile cloudlet. Moreover, the work in [5] considered a similar problem to ours, in which the user has a set of tasks which can be executed in parallel on the user and cloudlets. The user targets to distribute an optimal number of tasks to each cloudlet such that the task execution reward is maximized while the processing cost is minimized.

However, these studies differ from ours in that they assumed specific models for the user's task execution demand, cloudlet's resources and availability of cloudlets, which may not always reflect a realistic ad-hoc mobile cloud accurately. In this study, we propose a model-free deep reinforcement learning (DRL)-based offloading algorithm which does not require any assumption on the system model.

B. Reinforcement Learning Background

Reinforcement learning (RL) [7] is a mathematical framework for experience-driven autonomous learning through interaction. In the standard RL model, an autonomous learning agent interacts with an environment through a sequence of observations, actions and rewards. At each time step t, the agent first observes a state s_t from its environment, then takes an action a_t and receives a scalar reward r_t as a feedback. After doing the action a_t, the environment transits to a new state s_{t+1}. The process continues and the goal of the agent is to learn a policy (i.e., an action selection strategy) π that maximizes the expected reward over the long run. Formally, the RL can be described as a Markov decision process (MDP) in which the environment's response about the next state s_{t+1} depends only on the state s_t and taken action a_t. Moreover, the main focus of the RL is on learning without the knowledge of the underlying environment model. The most popular model-free RL algorithm is Q-learning proposed by Watkins *et al.* [8]

Deep RL has emerged as a class of RL schemes which uses the deep learning [9] to enable reinforcement learning to scale to decision-making problems with high-dimensional state and action spaces [10]. In general, the deep RL is based on training neural networks to learn the value function $Q(s, a)$ according to which the optimal policy is obtained.

The first well-known success story of deep RL is the deep Q-network (DQN) algorithm which was developed by Mnih *et al.* [2] to play the Atari 2600 video games at superhuman level, directly from game screen images. Furthermore, Hasselt *et al.* [11] proposed a Double DQN algorithm which integrates the Double Q-learning to enhance the performance of the DQN algorithm.

In this paper, we apply the DQN algorithm for a mobile user to learn a policy which gives efficient offloading decisions that the user should make at each system states.

III. System Model

A. System Assumption

In this paper, we consider an ad-hoc mobile cloud in which a mobile user (thereafter referred to simply as "user") can offload its computation tasks to N mobile cloudlets with available resources in terms of storage and computing capabilities. When the user has a number of computation tasks to be executed, it first seeks for the cloudlets which are currently located in its D2D communication range, denoted by R_c. Then, the user may offload a part of its tasks to the cloudlets. Upon receiving the task from the user, the cloudlets execute the tasks and send the results back to the user.

To represent the user's computation task, we use a simplified model which defines a computation task with three parameters of J, B, O which denote the number of CPU cycles required to execute one task and the numbers of data transmission bits required to offload the task and receive the result to/from the cloudlet, respectively. It is assumed that the user has a queue with a limited size to store its task which may arrive depending on its application execution demand. The tasks are stored and processed at the user's queue according to a FCFS (First-Come-First-Served) manner. Furthermore, let f^u denote the CPU speed of the user.

When the user offloads the task to the cloudlet, it has to make a certain payment for the cloudlet's resource usage. If we denote η_i as the price for the cloudlet i to execute a task, it is calculated as $\eta_i = \varsigma f_i^c$ where $\varsigma > 0$ is an execution payment constant and f_i^c $(i = 1, ..., N)$ denotes the CPU speed of the cloudlet i. Each cloudlet stores the tasks from the user in a dedicated queue and executes them sequentially.

For communication between the user and cloudlets, we consider a Device-to-Device (D2D) communication-enabled cellular network [12], which includes cellular and D2D communication modes. More specifically, for offloading the tasks, the user first starts the D2D mode using a direct link (e.g., Wi-Fi Direct or Bluetooth links) to transmit the task-related data to the cloudlet. During the data transmission time, if the D2D link between the user and the cloudlet is broken due to the user and cloudlet mobility, the communication mode

is automatically switched to the cellular mode (e.g., 3G/4G links) to transmit the rest of the data to the cloudlet.

Furthermore, to estimate the communication energy consumption in both D2D and cellular modes, we adopt the linear model which was empirically derived and validated in [13]. If we denote r_m as the achievable data rate, the transmission power, denoted by P_m is calculated as follows [13]

$$P_m = \alpha_m r_m + \beta_m \tag{1}$$

where α_m and β_m are parameters whose values depend on the communication mode. Note that the subscript $m \in \{d2d, cel\}$, where $d2d$ and cel represent D2D and cellular modes, respectively. In this study, we use values of $\alpha_{\text{d2d}} = 283.17$ mW/Mbps and $\beta_{\text{d2d}} = 132.86$ mW for the D2D Wi-Fi network as derived in [13]. For the 3G cellular network, the value of β_{cel} is 817.88 mW while the value of α_{cel} is 868.98 mW/Mbps and 122.12 mW/Mbps for the up-link and down-link, respectively. Also, it is assumed that the D2D mode uses the out-of-band D2D communication with Wi-Fi Direct [12]. Let χ_{cel} denote the price of using the cellular bandwidth to transmit one data bit while the D2D link is regarded as free due to the usage of the unlicensed spectrum.

Time is logically divided into intervals of T, which are defined as decision periods. At the beginning of a decision period, the user first discovers the cloudlets in its D2D communication range and then makes an offloading decision how many tasks the user should process locally and how many tasks to offload to each cloudlet based on the user's observation on the number of tasks currently remaining in the user's queue, the distance between the user and each cloudlet, and the queue status of the cloudlets.

IV. Obtaining Offloading Policy through Deep Reinforcement Learning

A. MDP-based Offloading Problem Formulation

The problem of making an offloading decision for the user is formulated as a finite MDP which is defined as a tuple $M = (S, A, P, R)$, where S and A denote state and action spaces, $P(s'|s, a)$ indicates the transition probability from state $s \in S$ to $s' \in S$ after tasking an action $a \in A$ and $R(s, a)$ denotes the immediate reward obtained by doing action a at state s. A policy, denoted by π is defined as a mapping from a state s to an action a, i.e., $\pi(s) = a$. The main goal of the user to find the optimal policy π^* which maximizes the total amount of reward it can obtain over the long run.

1) State and Action Spaces: The state space S consisting of information about the user's and cloudlets' queue and the distance between the user and cloudlets is defined as follows:

$$S = \{s = (Q^u, Q^c, D)\} \tag{2}$$

where Q^u, Q^c and D denote the user's queue state, cloudlet's queue state and distance state, respectively.

The user's queue state $Q^u \in \mathbb{Q}^u$ is defined as the number of tasks currently in the user's queue, where the state space $\mathbb{Q}^u = \{0, 1, \ldots, |Q^u|\}$. Note that $|Q^u|$ denotes the size of the user's queue. The cloudlet's queue state Q^c is a composite state of N cloudlet queue states, i.e., $Q^c = \{(Q_1^c, Q_2^c, \ldots, Q_N^c)|Q_i^c \in \{0, 1, \ldots, |Q_i^c|\}\}$, where $|Q_i^c|$ denotes the queue size of the cloudlet i and Q_i^c is the number of tasks currently stored in the queue of cloudlet i.

The distance state D is a composite state of N cloudlet distance states i.e., $D = \{(D_1, \ldots, D_N)|D_i \in \{1, 2, \ldots, H\}\}$ where D_i is the distance state of the cloudlet i and H is the number of possible distance states. More specifically, we adopt a Markov chain model (MCM) [14] to describe the distance between the user and a cloudlet. A cloudlet i having distance d_i to the user is said to have a distance state $h (h = 1, \ldots, H)$ if $(h-1)W \leq d_i < hW$ where $W = R_c/(H-1)$. The state H is used to represent the distance state of the cloudlet which is not in the user's D2D range (i.e., $d_i > R_c$). Note that in this study, we include the distance information in the system state since the distance state affects the residual link duration between the user and cloudlet, as derived in [14].

The action space A is defined as

$$A = \{a = (a_0, \ldots, a_i, \ldots, a_N)\big|a_i \in \{0, 1, \ldots, a_{\max}\}\} \tag{3}$$

where $a_{\max}$ is the maximum number of tasks to be locally processed or offloaded to a cloudlet in each decision period. An action $a \in A$ can be considered as the task distribution decision in which a_0 is the number of tasks for local processing and a_i $(i = 1, \ldots, N)$ is the number of tasks to be offloaded to the cloudlet i. Let $A(s) \subseteq A$ be the set of actions that can be taken at the state s. The set $A(s)$ is determined such that the user can only offload the tasks to cloudlets with an available queue space in its D2D communication range. In addition, the total number of tasks per action a must be equal to or less than the number of tasks currently remained in the user's queue.

2) Immediate Reward: The main goal of the user is to make an optimal offloading action at each system state with the objective of maximizing the utility while minimizing the energy consumption, processing delay, required payment and and task's loss probability. Therefore, we define the immediate reward function $R(s, a)$ given an action a at state s as follows

$$R(s, a) = U(s, a) - C(s, a) \tag{4}$$

where $U(s, a)$ and $C(s, a)$ are immediate utility and cost functions. For the utility, we adopt the logarithmic utility function as

$$U(s, a) = \rho \sum_{i=0}^{N} \log(1 + a_i) \tag{5}$$

where ρ is a utility constant. The immediate cost function $C(s, a)$ is defined as:

$$C(s, a) = \omega_1 I(s, a) + \omega_2 E(s, a) + \omega_3 D(s, a) + \omega_4 \Gamma(s, a) \tag{6}$$

where $I(s, a)$, $E(a, s)$, $D(s, a)$ and $\Gamma(s, a)$ are immediate required payment, energy consumption, delay and task loss probability, respectively.

The immediate payment $I(s, a)$ is defined as the total payment that the user has to make when taking an action a at

state s. If we denote τ^i_{d2d} as the D2D link duration between the user and cloudlet i, the maximum amount of transmission data via the D2D link is $\Phi_i = \tau^i_{\text{d2d}} r_{\text{d2d}}$. The value of $I(s,a)$ is calculated as

$$I(s,a) = \sum_{i=1}^{i=N} a_i \Big(\eta_i + \chi_{\text{cel}} \max(0, a_i B - \Phi_i) \Big) \quad (7)$$

where $\max(0, a_i B - \Phi_i)$ function gives an amount of data transmitted via the cellular link after the D2D link is broken. Recall that η_i and χ_{cel} are prices for the use of cloudlet's computation resources and cellular bandwidth, respectively.

The immediate energy consumption $E(s,a)$ is defined as the total energy that user has to consume given action a at state s. The value of $E(s,a)$ includes total energy consumed by local execution and communication. That is

$$E(s,a) = a_0 J \xi^u + E_{\text{com}}(s,a) \quad (8)$$

where ξ^u is the user's local energy consumption per CPU cycle and $E_{\text{com}}(s,a)$ denotes the communication energy consumption for offloading the tasks and receiving the the results. The value of $E_{\text{com}}(s,a)$ is calculated as

$$E_{\text{com}}(s,a) = \sum_{i=1}^{N} \Big(\frac{\min(\Phi_i, a_i B) P_{\text{d2d}}}{r_{\text{d2d}}} + \frac{\max(0, a_i B - \Phi_i) P^u_{\text{cel}}}{r_{\text{cel}}} + \frac{a_i O P^d_{\text{cel}}}{r_{\text{cel}}} \Big) \quad (9)$$

where $\min(\Phi_i, a_i B)$ is an amount of data transmitted via the D2D link, and P^u_{cel} and P^d_{cel} denote transmission powers on cellular up-link and down-link, respectively. In (9), the first and second terms are the energy consumed for transmitting the data via the D2D link and cellular up link, respectively, while the final term is the energy consumed on receiving the results via the cellular down-link.

The immediate delay $D(s,a)$ is defined as a sum of average values of the waiting time at the user's and cloudlet's queues, the communication time of task offloading and the processing time by the user or cloudlets. That is

$$D(s,a) = \frac{(t^w + t^e + t^{\text{com}})}{\sum_{i=0}^{N} a_i} \quad (10)$$

where t^w, t^e and t^{com} denote the total waiting time, total execution time and total communication time, respectively. They are calculated as

$$\begin{cases} t^w = t^w_u + \sum_{i=1}^{N} (t^w_u + \frac{JQ^i_c}{f^c_i}) \\ t^e = \frac{Ja_0}{f^u} + \sum_{i=1}^{N} \frac{Ja_i}{f^c_i} \\ t^{\text{com}} = \sum_{i=1}^{N} (\frac{\min(\Phi_i, a_i B)}{r_{\text{d2d}}} + \frac{\max(0, a_i B - \Phi_i) + O a_i}{r_{\text{cel}}}) \end{cases} \quad (11)$$

where t^w_u is the average waiting time of the task at the user's queue. Moreover, the communication delay includes the data transmission time for offloading the task and receiving the result via a the D2D and cellular links.

We define $\Gamma(s,a)$ as the probability that a task can be lost since it arrives when the user's queue is already full. If we denote μ_{task} as the average number of task arrivals during a decision period of T, the value of $\Upsilon(s,a)$ is calculated as

$$\Gamma(s,a) = \frac{\max\Big(0, \mu_{\text{task}} - \Upsilon_{\text{avai}}\Big)}{\mu_{\text{task}}} \quad (12)$$

where $\Upsilon_{\text{avai}} = |Q^u| - Q^u + \sum_{i=0}^{N} a_i$ denotes the available space of the user's queue after taking an action a. Recall that Q^u is the number of remaining tasks at state s.

B. DQN-based Offloading Decision Algorithm

To find the solution to the above formulated MDP-based offloading problem, we propose an online learning scheme based on the model-free deep RL algorithm, called deep Q-network (DQN) [2]. In the DQN-based learning scheme, the user acts as an agent which interacts with the ad-hoc mobile cloud environment to select an offloading action a_t for state s_t at time-step t in a fashion that maximizes the future discounted reward over a long run. More specifically, a deep neural network, called a deep Q-network, is used to approximate the optimal action-value function [2]

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \sum_{k=1}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi] \quad (13)$$

which is the maximum sum of reward r_t discounted by γ after taking the action a at the state s at the time-step t in the offloading policy π. Note that $\mathbb{E}[.]$ denote the expectation function.

The Q-network can be considered as a neural network approximator with an approximate action-value function $Q(s,a;\theta)$ with weights θ. At each decision period, the user first takes the state vector $s = (Q^u, Q^c, D)$ as the input of the Q-network and obtains the Q-values $Q(s,.)$ for all possible action a as outputs. Then, the user selects the action according to the ε-greedy method.

Furthermore, the Q-network is trained by iteratively adjusting the weights θ to minimize a sequence of the loss functions, where the loss function at time-step t is defined as

$$L_t(\theta_t) = \mathbb{E}[\big(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_{t-1}) - Q(s_t, a_t; \theta_t)\big)^2] \quad (14)$$

In other words, given a transition $< s_t, a_t, r_t, s_{t+1} >$, the weights θ of the Q-network are updated in a way that minimizes the squared error loss between the current predicted Q-value of $Q(s_t, a_t)$ and the target Q-value of $\big(r_t + \gamma \max_{a'} Q(s_{t+1}, a')\big)$.

Moreover, in the DQN algorithm, the experience replay technique is adopted as the training method to address the instability of the Q-network due to use of non-linear approximation function. More specifically, the user's experiences, $e_t =< s_t, a_t, r_t, s_{t+1} >$ are stored into a replay memory $\Omega = \{e_{t-\Psi}, ..., e_t\}$, where Ψ is the replay memory capacity. At each time step, a random mini-batch of transitions from the replay memory is chosen to train the Q-network, instead of the most recent transition e_t.

The detailed DQN-based offloading decision algorithm is presented in Algorithm 1. The recursions in lines (2)-(4)

Algorithm 1 DQN-based Offloading Decision Algorithm

1: **for** t = 1, 2, ... **do**
2: Observe current state s_t
3: Select action a_t according to ε-greedy policy
4: Offload the tasks according to action a_t and observe reward $r_t = R(s_t, a_t)$ and next state s_{t+1}
5: Store experience $< s_t, a_t, r_t, s_{t+1} >$ into replay memory Ω
6: Randomly select a set of transitions $< s, a, r, s' >$ from replay memory Ω
7: Train the Q-network based on selected transitions using $\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)^2$ as loss function
8: **end for**

present the user's action on making a offloading decision at the beginning of each decision period according to the Q-values which are estimated using the Q-network. In lines (5)-(7), the Q-network is trained using the experience replay method.

V. Performance Evaluation

A. Simulation Setup

We evaluate the performance of the proposed offloading algorithm in simulations which consider an ad-hoc mobile cloud consisting of $N = 4$ mobile cloudlets. Major simulation parameters are summarized in Table I.

To simulate the arrival of the user's task, we use a Poisson process with a parameter λ, which is considered as the estimated value of the average number of arrived tasks μ_{task} during a period T. Furthermore, we adopt the semi-Markov smooth mobility (SMS) model [15] to simulate the mobility of the user and cloudlets, in which the average movement speed of the user and cloudlets is set to 4 m/s. Moreover, we consider a network area of 500×500 m.

In the DQN-based learning algorithm, the discount factor is $\gamma = 0.9$ while the replay memory capacity Ψ is set to 10^4. The DQN learning algorithm is implemented using the Tensorflow APIs, in which a gradient descent optimization algorithm, called RMSProp is used as the optimization method for training the Q-network. Lastly, the behaviour action selection during the training is ε-greedy with ε which is annealed linearly from 1 to 0.1.

Five performance metrics are used, which are total utility, total energy consumption, total payment, average delay and task loss ratio. Note that the task loss ratio is calculated as a ratio of the number of lost tasks due to the user's queue overflow to the total number of tasks generated during the simulation time.

B. Performance Analysis

We analyze the performance of the proposed algorithm in various simulation scenarios where the user's task arrival rate λ varies from 1 to 5.

Fig. 1 shows the learning curves which present total reward obtained by the offloading policy learned in each training episode over variation of the task arrival rate. Note that an episode includes 5000 iterations (i.e., the number of decision periods), in each of which the user selects an offloading action for each state according to the current policy learned by the proposed algorithm.

TABLE I
Simulation settings

Parameters	Values
Number of mobile cloudlets N	4
User's queue size $\lvert Q^u \rvert$	10 tasks
Cloudlets' queue size $\lvert Q_i^c \rvert$	4 tasks
Number of distance states H	4
# of CPU cycles per tasks J	[50, 100] Gcycles
Task's data sizes B, O	[0.5, 0.7], [0.05, 0.1] Gb
CPU speeds f_u, f_i^c	[0.5, 2] Gcycles
Constants $(\varsigma, \chi_{\text{cel}}, \rho, \xi^u)$	$(10^{-8}, 10^{-6}, 10^4, 5\text{mJ})$
Weights $(\omega_1, \omega_2, \omega_3, \omega_4)$	$(10^3, 10^{-5}, 10^{-5}, 10^3)$
Data rates r_{d2d}, r_{cel}	$(10, 5)$ Mbps
D2D communication range R_c	200 m
Decision period T	60 s

As shown in Fig. 1, the total reward obtained in each episode increases steadily when the learning time i.e., the number of episodes, increases from 1 to 200. Then, all learning curves become stable and no longer increase when the episode number is higher than 200. This result indicates that the proposed DQN-based learning algorithm converges after 200 learning episodes.

Figs. 2-6 show the quality of the learned offloading policy. More specifically, Fig. 2 exhibits the total utility that the user obtain during 300^{th} episode over variation of the task arrival rate λ. The total utility greatly increases as the value of λ is increased from 1 to 3 since a higher number of tasks are executed by the user and cloudlets. Moreover, when the value of λ is higher than 3, the utility becomes slightly greater since the user and cloudlet nearly reach their maximum computation capabilities. They cannot execute more tasks although the number of arrived tasks is higher.

Furthermore, over the increase of the task arrival rate, the user consumes a higher amount of energy for local execution and communication and makes a higher payment, as shown in Figs. 3 and 4 since a higher number of tasks is processed.

Fig. 5 shows the average delay over the variation of the task arrival rate. When the task arrival rate increases, the delay also increases. The main reason is that the tasks' waiting time at the user's and cloudlets' queues is longer when more tasks arrive. Given the system capability related to the user's and cloudlets' storage and computation resources, if more tasks are generated, a newly arrived task will have to wait at the queue for a longer time for all the previously stored tasks to be processed.

In addition, when the rate of task arrival increases, at a time instance, the probability that the queue is already full increases. Therefore, the number of lost tasks due to the lack of user's queue space will be higher as shown in Fig. 6.

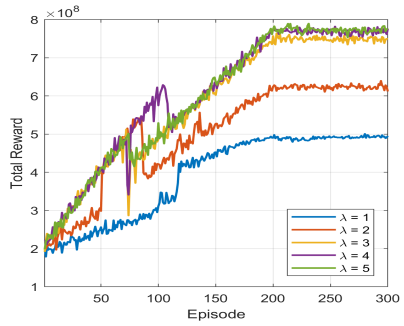

Fig. 1. Learning curves.

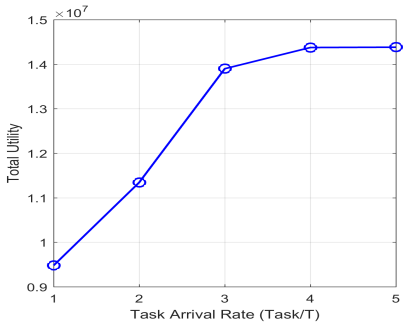

Fig. 2. Total utility.

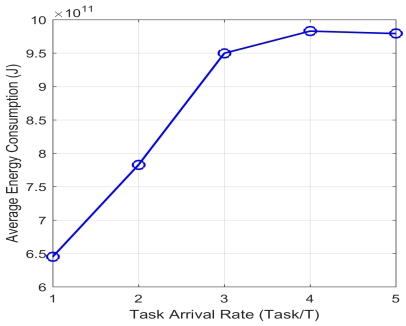

Fig. 3. Total energy consumption.

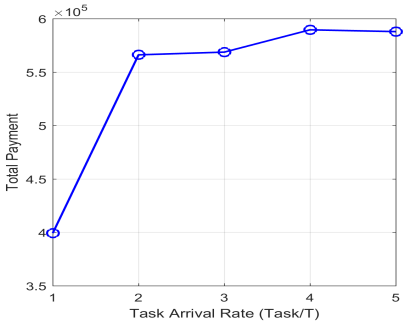

Fig. 4. Total required payment.

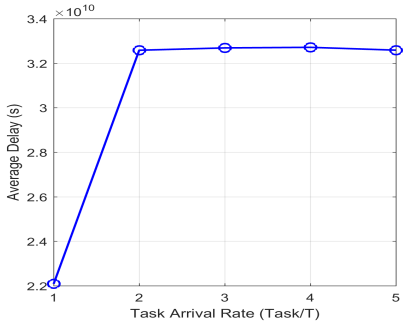

Fig. 5. Average delay.

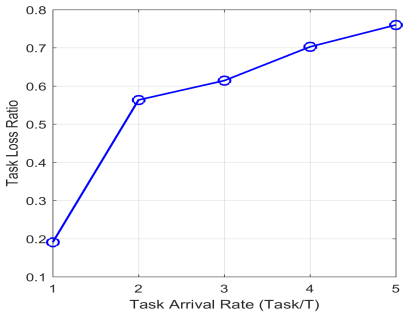

Fig. 6. Task loss ratio.

VI. Conclusion

In this paper, we have proposed a deep reinforcement learning (DRL)-based offloading algorithm for the user to obtain the optimal offloading policy in an ad-hoc mobile cloud. The offloading problem is formulated as a Markov decision process (MDP) with the main objective of making an optimal offloading action decision at each system state such that the utility obtained by the task execution is maximized while minimizing the energy consumption, processing delay, required payment and task loss probability. An DRL scheme, namely deep Q-network (DQN) is adopted to learn a near-optimal offloading policy for the proposed MDP-based offloading problem. We evaluated the performance of the proposed offloading scheme in various simulations in which the user uses the proposed learning algorithm to make offloading decisions in realistic scenarios. The simulation results confirm that the proposed schemes perform well in terms of various performance metrics.

Acknowledgment

This research/project is supported by the National Research Foundation, Prime Minister's Office, Singapore, under its Campus for Research Excellence and Technological Enterprise (CREATE) programme, and an MOE AcRF Tier 1 FRC Research Grant.

References

[1] M. Chen, Y. Hao, Y. Li, C. F. Lai, and D. Wu, "On the computation offloading at ad hoc cloudlet: architecture and service modes," *IEEE Commun. Mag.*, vol. 53, no. 6, pp. 18–24, June 2015.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 2017.

[4] Y. Zhang, D. Niyato, and P. Wang, "Offloading in mobile cloudlet systems with intermittent connectivity," *IEEE Trans. Mobile Comput.*, vol. 14, no. 12, pp. 2516–2529, Dec 2015.

[5] T. Truong-Huu, C. K. Tham, and D. Niyato, "To offload or to wait: An opportunistic offloading algorithm for parallel tasks in a mobile cloud," in *Proc. IEEE CloudCom*, Dec 2014, pp. 182–189.

[6] D. V. Le and C.-K. Tham, "An optimization-based approach to offloading in ad-hoc mobile clouds," in *Proc. 2017 IEEE GLOBECOM*, Dec. 2017.

[7] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

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

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 512, pp. 436–444, 2015.

[10] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," Aug. 2017, arXiv:1708.05866.

[11] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with Double Q-learning," in *Proc. AAAI*, Feb. 2016, pp. 2094–2100.

[12] A. Asadi, Q. Wang, and V. Mancuso, "A survey on device-to-device communication in cellular networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 1801–1819, Fourth quarter 2014.

[13] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4G LTE networks," in *Proc. the 10th MobiSys*, 2012, pp. 225–238.

[14] S. Xu, K. L. Blackmore, and H. M. Jones, "An analysis framework for mobility metrics in mobile ad hoc networks," *EURASIP J. Wirel. Commun. Netw.*, vol. 2007, pp. 1–16, 2007.

[15] M. Zhao and W. Wang, "A unified mobility model for analysis and simulation of mobile wireless networks," *Wirel. Netw.*, vol. 15, no. 3, pp. 365–389, 2009.