Distributed V2V Computation Offloading Based on Dynamic Pricing Using Deep Reinforcement Learning

Jinming Shi, Jun Du, Jian Wang, Jian Yuan
Department of Electronic Engineering, Tsinghua University, Beijing, 100084, China
E-mail: shijinmings@163.com, blgdujun@gmail.com, {jian-wang, jyuan}@tsinghua.edu.cn

***Abstract*—Vehicular computation offloading is a promising paradigm that improves the computing capability of vehicles to support autonomous driving and various on-board infotainment services. Comparing with accessing the remote cloud, distributed vehicle-to-vehicle (V2V) computation offloading is more efficient and suitable for delay-sensitive tasks by taking advantage of vehicular idle computing resources. Due to the high dynamic vehicular environment and the variation of available vehicular computing resources, it is a great challenge to design an effective task offloading mechanism to efficiently utilize vehicular computing resources. In this paper, we investigate the computation task allocation among vehicles, and propose a distributed V2V computation offloading framework, in which wireless channel states and variation of idle computing resources are both considered. Specially, we formulate the task allocation problem as a sequential decision making problem, which can be solved by using deep reinforcement learning. Considering that vehicles with idle computing resources may not share their computing resources voluntarily, we thus propose a dynamic pricing scheme that motivates vehicles to contribute their computing resources according to the price they receive. The performance of designed task allocation mechanism is validated by simulation results which reveal the effectiveness of our mechanism compared to the other algorithms.**

I. Introduction

With the emergency of varieties of vehicular applications, vehicles on the roads have growing demands in wireless communication and computations. In some cases, the computing resources on-board are very limited for a vehicle to process a number of computation-intensive or delay-sensitive tasks, the vehicle has to offload part of its computation tasks to other devices which have more computing resources. An option is to access the remote cloud, but it suffers from the long latency [1], which is not suitable for delay-sensitive tasks. Mobile edge computing (MEC) is a promising paradigm that can provide low-latency task offloading service. However, due to the high dynamics of vehicular environments, the traditional MEC network architecture cannot directly be applied to the Internet of Vehicles. With vehicles equipped with increasing amount of computing resources, it is considered that vehicles on a road can form a distributed system, which can manage computation tasks more efficiently and cost-effectively [2]. Therefore, it is necessary to design an effective scheme that allocates idle vehicular computing resources to the vehicles that demand more computation capability. In this work, we will focus on the distributed vehicle-to-vehicle (V2V) computation offloading mechanism to optimize the utilization of computing resources and the efficiency of task offloading.

There are some existing works investigating the computation offloading in vehicular networks. Sun *et al.* [3] proposed a distributed task offloading framework based on multi-armed bandit theory, where vehicles learn the task offloading performance of its surrounding vehicles to minimize the average offloading delay. In [4], an Ant Colony Optimization (ACO) based algorithm was proposed to schedule tasks efficiently without infrastructures or centralized control. Both of the works above assumed that vehicles share idle computing resources voluntarily and did not consider the incentive mechanism of such resources sharing. In order to motivate vehicles to share their idle computing resources, Su *et al.* [5] proposed a distributed computation task allocation framework, where vehicles charge the service requester dynamically according to the amount of idle resources, the vehicle density, and the velocity difference. In [6], a reverse auction based V2V computation offloading scheme was proposed, where idle computing resources are sold among vehicles. However, all of the works mentioned above did not consider the cost of executing a task and the variation of the remaining available computing resources in the service provider, in fact, both of which are very important in evaluating the willingness of service provider in real world.

To solve the problems mentioned above, this work proposes a distributed task offloading framework that considers four aspects: the V2V link state, the motivation of service vehicles, the cost of both task vehicle and service vehicles, and the variation of available computing resources in service vehicles. We present a scenario where a task vehicle with limited computation capability offloads part of its computation tasks to neighbouring service vehicles with idle computing resources, and the resource allocation is managed by a base station. In addition, a dynamic pricing scheme is proposed to motivate vehicles to contribute their computing resources according to the price they receive. The problem of resource allocation is formulated as a sequential decision making problem which aims to maximize the cumulative reward of the task vehicle for offloading tasks. By employing deep reinforcement learning, the algorithm efficiently allocates idle computing resources to all of the tasks under high dynamic vehicular environment.

The rest of this paper is organized as follows. Section II

978-1-7281-3106-1/20/$31.00 ©2020 IEEE

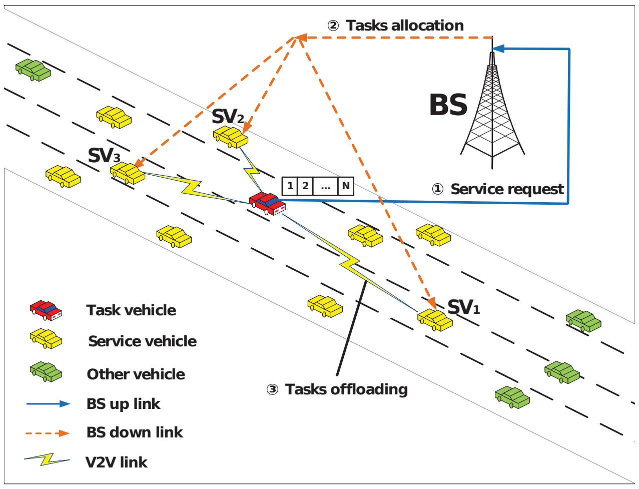

Fig. 1. System architecture.

presents the system model of V2V task offloading. In Section III, the task allocation problem is formulated. The detailed learning-based task allocation algorithm is demonstrated in Section IV. Simulation results are provided in section V, and conclusions are drawn in Section VI.

II. System Model

A. System Overview

As illustrated in Fig. 1, there is one vehicle which does not have enough computing resources to accomplish all of its tasks during a period of time. This vehicle has to offload part of its tasks to neighbouring vehicles within its communication range. We denote the vehicle as the task vehicle (TV). The vehicles surrounding the TV which are available to provide task offloading are called the service vehicles (SVs). We assume that during the period of task offloading, there are K SVs, denoted as $\mathcal{SV} = \{SV_1, SV_2, ..., SV_K\}$, around the TV, and all the vehicles on the road are able to access the base station (BS) all the time. When the TV has tasks to be offloaded, it first sends the task offloading request to the BS, then the BS decides which SV to process the task and sets the price that the TV should pay for the task to the selected SV. Similar to [7], we divide the system time into several periods, and in a given period of time, the TV has N tasks to be offloaded, we denote task n as $\phi_n = \{D_n, C_n, \tau_n\}, n \in \{1, 2, ..., N\}$, where D_n is the input data size of the task, C_n is the CPU cycles required to accomplish the computation, and τ_n is the maximum tolerable delay of task.

We assume that each vehicle drives in a constant velocity during the given period, and the velocity of different vehicles follows a uniform distribution. The distance between the TV and SV is utilized to evaluate the channel state while the relative position and relative speed of TV and SV are exploited to evaluate the link duration.

B. Task Delay

For each vehicle on the road, if the task of a vehicle is processed locally, the total delay can be given as

$$t_n^{loc} = \frac{C_n}{f_n}, \tag{1}$$

while if a task is offloaded to other vehicle, the total delay is

$$\begin{aligned} t_n^{off} &= t_n^{up} + t_n^{comp} + t_n^{down} \\ &= \frac{D_n}{r_{t,k}} + \frac{C_n}{f_k} + \frac{\delta D_n}{r_{k,t}}, \end{aligned} \tag{2}$$

where $r_{t,k}$ is the available transmission rate between the TV and SV_k, f_k is the allocated computing resources of SV_k for the offloading task, δ is the ratio between the output data size and the input data size, and in general $\delta \ll 1$ which can be ignored in the computation of task delay. We denote $r_{t,k}$ as

$$r_{t,k} = B_{t,k} \log_2\left(1 + \frac{P_t d_{t,k}^{-\beta} h_{t,k}^2}{N_0 + I_{t,k}}\right), \tag{3}$$

where $B_{t,k}$ is the allocated channel bandwidth, P_t is the power of transmitter, $d_{t,k}$ is the distance between the TV and SV_k, β is the path loss exponent, $h_{t,k}$ is the channel gain, N_0 is the power of Gaussian White Noise, and $I_{t,k}$ is the interference introduced by other V2V transmissions.

C. Utility Model

In order to quantify the efficiency of performing a task in a vehicle, we define the following utility function similar to [8], which represents the satisfaction of task execution related to the task completion time,

$$U_n = \log(1 + (\tau_n - t_n)). \tag{4}$$

In general, when an SV performs a computation task, it also carries out its local tasks at the same time. The total utility of a vehicle performing its local tasks in the given period is

$$U_k = \sum_{l=1}^{L} \log\left(1 + \left(\tau_l - \frac{C_l}{f_l}\right)\right). \tag{5}$$

We define the total available computation capability of SV_k as F_k, then we have

$$\sum_{l=1}^{L} f_l \le F_k, \tag{6}$$

and the minimum frequency allocated to a local task is

$$f_l^{min} = \frac{C_l}{\tau_l}. \tag{7}$$

Since we are mainly concerned with the utility of offloading tasks and for simplicity, we assume that the completion time of a local task in an SV is denoted as $(1 - \alpha_k)\tau_l$, where $\alpha_k \in [0, 1)$, and the amount of computing resources allocated to the local task is

$$f_l = \frac{C_l}{(1 - \alpha_k)\tau_l}, \tag{8}$$

and then we have

$$\sum_{l=1}^{L} \frac{C_l}{(1-\alpha_k)\tau_l} \leq F_k. \tag{9}$$

We define F_k^{min} as the minimum computing resources that local tasks demand in the given period, and we have

$$F_k^{min} = \sum_{l=1}^{L} \frac{C_l}{\tau_l} \leq (1-\alpha_k)F_k, \tag{10}$$

$$\alpha_k \leq 1 - \frac{F_k^{min}}{F_k} = \alpha_k^{max}. \tag{11}$$

By combining (5) and (8), we have the total utility of accomplishing all of local tasks in SV_k in the given period

$$U_k(\alpha_k) = \sum_{l=1}^{L} \log(1+\alpha_k\tau_l). \tag{12}$$

One can notice that $U_k(\alpha_k)$ is a monotonic increasing function and from (5), $\alpha_k = 0$ means that SV_k allocates the minimum computing resources to each of its local tasks, while $\alpha_k = \alpha_k^{max}$ means that SV_k allocates all of its computing resources to its local tasks. If the TV pays SV_k with price $\gamma_n C_n$ for offloading task n, where γ_n is the unit price per cycle, and the amount of allocated computing resources for task n is

$$f_n = F_k - \frac{F_k^{min}}{1-\alpha_k}, \tag{13}$$

then the final utility of SV_k for executing task n is

$$U_k^n = \gamma_k C_n + U_k(\alpha_k) - U_k(\alpha_k^{max}), \tag{14}$$

and the utility of TV for offloading task n is

$$U_t^n = \begin{cases} \log(1+(\tau_n - t_n^{off})) - \gamma_n C_n, & t_n \leq \tau_n, \\ C', & t_n > \tau_n, \end{cases} \tag{15}$$

where $C' < 0$ is a constant which represents the penalty that a task cannot be executed before the maximum tolerable delay.

III. Formulation of Task Allocation Problem

We assume that the utility of an SV after accomplishing an offloading task is proportional to the CPU cycles required of the task, i.e.

$$U_k^n = \gamma_k C_n + U(\alpha_k) - U(\alpha_k^{max}) = \Gamma C_n, \tag{16}$$

where Γ is a constant, which is called the basic unit price. According to (12), (13) and (16), we can notice that with the payment of TV increasing, α_k will be lower which means that the offloading task will get more computing resources. Since $U_k(\alpha_k)$ is monotonic increasing and $\alpha_k \in [0, \alpha_k^{max}]$, the domain of γ_n is $(\Gamma, \Gamma + U_k(\alpha_k^{max})/C_n]$. Our goal is to maximize the mean utility of TV for offloading all tasks during the given period. The problem of optimizing the mean utility of TV is formulated as follows:

$$\begin{aligned} \max \quad & \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} x_k^n(\log(1+(\tau_n - t_n^{off})) - \gamma_n C_n), \\ \text{s.t.} \quad & C1: \gamma_n - \Gamma > 0, \quad \forall n \in N, \\ & C2: \tau_n - t_n^{off} \geq 0, \quad \forall n \in N, \\ & C3: x_k^n \in \{0,1\}, \quad \forall n \in N, k \in K, \\ & C4: \sum_{k=1}^{K} x_k^n = 1, \quad \forall n \in N. \end{aligned} \tag{17}$$

In (17), the constraint $C1$ indicates that the price the TV paid to SVs must be higher than the basic unit price, and $C2$ ensures that the total task delay is not beyond the deadline of task. In $C3$, $x_k^n = 1$ means that task n is offloaded to SV_k, while $x_k^n = 0$ means not. $C4$ represents the task should only choose one SV to offload. The problem formulated in (17) is a mixed integer nonlinear programming (MINLP) problem, which is difficult to solve. To deal with this problem, we reformulate the problem as a Markov decision process (MDP) and solve it using deep reinforcement learning in the next section.

IV. Double Deep Q-Network Based Task Offloading

A. Environment Model

In the proposed task offloading framework, the server of BS acts as the agent that manages task allocation between the TV and SVs. Vehicles in the coverage of the BS are assumed to be connected with the BS all the time, and the BS is able to learn the states of the vehicles in real time. The state of an SV contains the position, velocity, computation capability, and profiles of local tasks to be executed. After receiving the states of SVs, the agent is able to estimate the state of wireless channels and the links duration between the TV and SVs. In addition, the agent can determine the interval of the service price and remaining available computing resources of SVs.

When a TV has a number of tasks which cannot be processed because of its limited computing resources, it will send a request of tasks offloading to the BS. This request message contains the input data size D_n, the computation size C_n, and the maximum tolerable delay τ_n of tasks. Then the BS allocates the tasks to SVs successively. After an SV receives an allocation message from the BS, this SV will reserve part of its computing resources for the task according to the service price. As long as the TV pays the service price and uploads the data of task to this SV, this SV will start to perform the task and then send the results back to the TV. The basic model of the environment is presented as follows.

*1) **System State:*** The state space is represented as

$$\begin{aligned} s(t) = \{ & r_1(t), r_2(t), ..., r_K(t), w_1(t), w_2(t), ..., w_K(t), \\ & v_1(t), v_2(t), ..., v_K(t), u_1(t), u_2(t), ..., u_K(t), \\ & D(t), C(t), \tau(t)\}, \end{aligned} \tag{18}$$

where $r_k(t)$ is the signal noise ratio (SNR) of channel which is measured according to the distance between the TV and SV_k,

$w_k(t)$ is the remaining available time of the link between the TV and SV_k, $v_k(t)$ is the remaining computing resources of SV_k, and $u_k(t)$ is the maximum total utility of local tasks of SV_k, which equals $U(\alpha_k^{max})$. $D(t), C(t), \tau(t)$ are the data size, the computation size, and the maximum tolerable delay of offloading task in moment t respectively.

We assume that the BS allocates offloading tasks successively. In addition, during the period of task allocation, the demand of local computing resources in each SV is supposed to be unchanged, i.e., the remaining computing resources and the maximum total utility of local tasks of an SV only depend on the amount of accepted offloading tasks. By taking an action that contains choosing an SV and determining the price paid to the SV, the SV reserves the computing resources for the offloading task, the remaining computing resources and the maximum total utility of local tasks of the SV are changed. The SNR and link duration also change because of the motion of vehicles, and the current state transitions to the next state.

*2) **Actions**:* The action for offloading a task is given as

$$a_n(t) = \{(x_1^n(t), \gamma_1^n(t)), (x_2^n(t), \gamma_2^n(t)), ..., (x_K^n(t), \gamma_K^n(t))\}, \tag{19}$$

where $x_k^n(t) \in \{0, 1\}$, and $x_k^n(t) = 1$ indicates that SV_k is chosen to perform task n while $x_k^n(t) = 0$ means that SV_k is not chosen for task n. $\gamma_k^n(t)$ is the price paid to SV_k for task n, and the domain of $(\gamma_k^n(t) - \Gamma)C_n$ is $[0, U_k(\alpha_k^{max})]$. In this work, we discretize the value of $(\gamma_k^n(t) - \Gamma)C_n$ into P levels, which is denoted as $\{0, (1/P)U_k(\alpha_k^{max}), (2/P)U_k(\alpha_k^{max}), ..., U_k(\alpha_k^{max})\}$.

*3) **Rewards**:* Since we aim to maximize the mean utility of offloading tasks in the TV, the offloading reward of TV under current state $s(t)$ and conducted action $a_n(t)$ is defined as

$$R_n = \begin{cases} \sum_{k=1}^{K} x_k^n(t)(log(1 + (\tau_n - t_n)) - \gamma_k^n(t)C_n), & t_n \le \tau_n, \\ C', & t_n > \tau_n. \end{cases} \tag{20}$$

After all tasks are offloaded, the mean utility of all offloading tasks can be obtained as

$$R = \frac{1}{N}\sum_{n=1}^{N} R_n. \tag{21}$$

Due to the dynamic vehicular environment, changing state of V2V communication links, the remaining available computing resources, and the price interval of SVs are all difficult to predict, and it is hard to give a precise state transition probability as well.

B. Double DQN based Task Offloading Algorithm

To solve the problem, we employ the model-free based double DQN [9] method. Since our goal is to maximize the mean utility of offloading tasks in the TV during a period, according to (21), we need to obtain the optimal policy that maximize the expectation of the cumulative rewards of the TV. We first evaluate the value of each action taken in every state. In a certain state s_t, the value of action a_t is denoted as $Q^\pi(s_t, a_t)$, which means the expected reward after conducting action a_t in state s_t according to a certain policy π, and we have

$$\begin{aligned} Q^\pi(s_t, a_t) &= \mathbb{E}[R_t + \lambda R_{t+1} + \lambda R_{t+2} + \cdots | s_t, a_t] \\ &= \mathbb{E}_{s_{t+1}}[R_t + \lambda Q^\pi(s_{t+1}, a_{t+1}) | s_t, a_t], \end{aligned} \tag{22}$$

where R_t is the reward under state s_t and conducted action a_t, and $\lambda \in [0, 1)$ is the discount factor. Then we define the value function of optimal policy as

$$Q^*(s_t, a_t) = \max_\pi Q^\pi(s_t, a_t). \tag{23}$$

According to (22), we obtain the optimal value function

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}}[R_t + \lambda \max_a Q^*(s_{t+1}, a) | s_t, a_t], \tag{24}$$

and the optimal policy is evaluated by

$$\pi(s_{t+1}) = arg \max_a Q(s_{t+1}, a). \tag{25}$$

In order to obtain the optimal policy, a value iteration based on available experiences (s_i, a_i, R_i, s_{i+1}) is utilized, and the update equation of the value function is

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)). \tag{26}$$

Considering that the state space and action space could be extremely large, it is difficult to find the optimal policy by looking up the table of Q-values. A deep neural network (DNN) called Q-network is employed to approximate the Q-value function [10], which is denoted as $Q(s, a; \theta)$, where θ represents the weights of DNN. There are two Q-networks in double DQN, main Q-network and target Q-network. The main Q-network is used to determine the greedy policy while target Q-network is used to evaluate the value of policy.

The process of double DQN contains a number of episodes. In each episode, the agent collects initial state s_1 from the environment, and then in each step, to explore the optimal policy, it chooses an action randomly with probability ϵ and with probability $1 - \epsilon$, chooses the action corresponding to the maximum Q-value. According to the chosen action, the agent determines the target SV and the service price that the TV should pay to the SV. Then the SV allocates the computing resources according to the service price, and the agent computes reward R_t with the service price and the allocated computing resources. The state transmits from current state into the next state, the agent stores experience (s_t, a_t, R_t, s_{t+1}) into the replay buffer. Finally, the agent samples a batch of experiences from the replay buffer and update the weights of the main Q-network and target Q-network. The detailed task offloading algorithm is presented in Algorithm 1.

V. Simulation Results

In this section, we present the simulation results to estimate the performance of our proposed scheme. The detailed simulation parameters are listed in Table I. For comparison, we also provide another two task allocation algorithms, which are defined as follows:

Algorithm 1 Computation Task Offloading Algorithm based on Double DQN

```
Initialize: Initialize replay memory $\mathcal{M}$.
Initialize target network updating frequency $N^-$.
Initialize main Q-network $Q(s,a;\theta)$ with weights $\theta$.
Initialize target Q-network $Q^-(s,a;\theta^-)$ with weights $\theta^- = \theta$.
for each episode do
  Collect initial observation state $s_1$.
  for step n=1,2,...,N do
    Choose a random probability $p$.
    if $p < \epsilon$ then
      Select action $a_n$ randomly in the action set.
    else
      Choose action
      $$a_n = arg\max_a Q(s_n, a, \theta).$$
    end if
    According to $a_n$, determine the SV and service price $\gamma_n C_n$ of task $n$.
    The SV determines resources $f_n$ allocated to task $n$, and the BS computes reward $R_n$ with $f_n$ and $\gamma_n C_n$.
    The state transits from $s_n$ to $s_{n+1}$, and store $(s_n, a_n, R_n, s_{n+1})$ into $\mathcal{M}$.
    Sample a batch of $\mathcal{B}$ experiences $(s_i, a_i, R_i, s_{i+1})$ randomly from $\mathcal{M}$.
    Calculate the target Q-value with
    $$y_i^- = R_i + \lambda Q^-(s_{i+1}, arg\max_a Q(s_{i+1}, a, \theta); \theta^-).$$
    Update the main Q-network by minimize loss function
    $$L(\theta) = \frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} (y_i^- - Q(s_i, a_i; \theta))^2,$$
    In every $N^-$ steps, Update the parameters of the target Q-network by
    $$\theta^- = \eta\theta + (1-\eta)\theta^-.$$
  end for
end for
```

- **Greedy-based algorithm (GBA):** For each offloading task, we always choose the SV which has the most remaining computing resources to offload, and the TV tries all the available prices and selects the price corresponding to the maximum utility.
- **Random-based algorithm (RBA):** For each offloading task of the TV, we randomly choose an SV to offload, and the pricing strategy is the same as GBA.

In our simulation, there are 30 offloading tasks in total in a period and the number of SVs around the TV varies from 5 to 50. The range of relative speed between TV and SVs is $[-30, 30]$ km/h. During the process of task offloading, each of the SVs also executes $1 \sim 10$ local tasks.

TABLE I
SIMULATION PARAMETERS

Parameter	Value
Number of SVs	$5 \sim 50$
Number of SV local tasks	$1 \sim 10$
Number of offloading tasks	30
V2V bandwidth (MHz)	6
Maximum transmission range (m)	500
Relative velocity (km/h)	$[-30, 30]$
Computing capability of SV (GHz)	$[5, 10]$
Data size of task (MB)	$[0.05, 0.2]$
Computation size of task (10^9 cycles)	$[0.2, 1]$
Maximum delay of task (s)	$[0.5, 2]$
Basic unit price	0.1

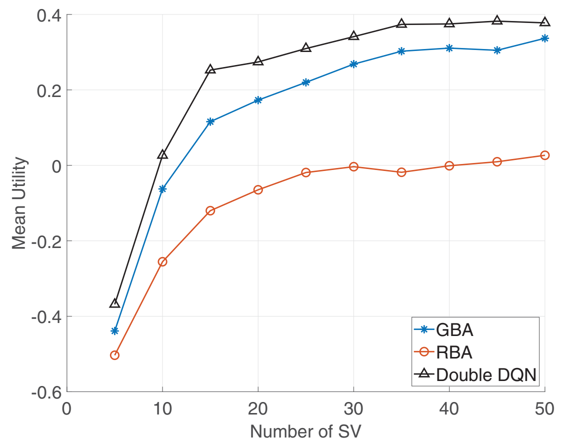

Fig. 2. The mean utility of TV for offloading a task versus different number of SVs.

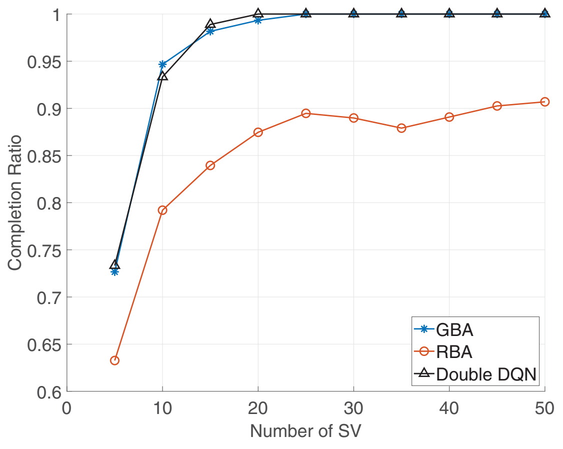

Fig. 3. The completion ratio of offloading tasks versus different number of SVs.

We simulate the variation of the mean utility of TV under different numbers of SVs by applying the proposed double DQN based algotithm, GBA and RBA, and the results are

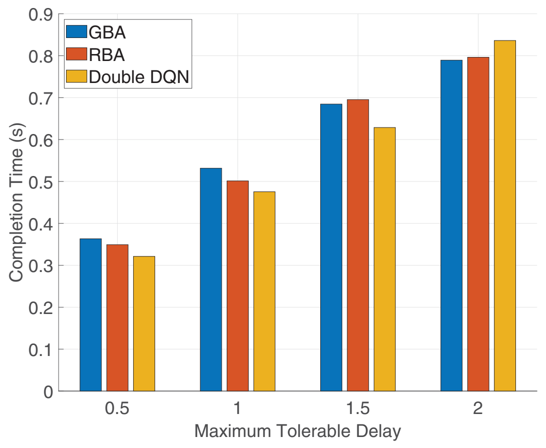

Fig. 4. The average completion time of tasks with different maximum tolerable delays when the number of SVs is 10.

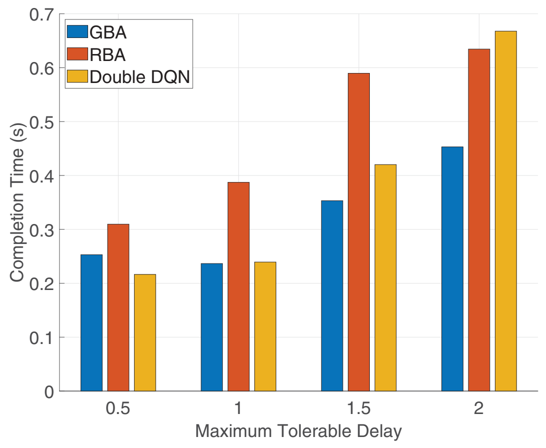

Fig. 5. The average completion time of tasks with different maximum tolerable delays when the number of SVs is 30.

shown in Fig. 2. We notice that the mean utility of TV for offloading tasks under the proposed algorithm is always higher than GBA and RBA. This is because our proposed algorithm aims to maximize the cumulative utility during the task allocation, while GBA aims to maximize the utility of each step and does not consider the after-effects of the action in each step, which may not obtain the maximum total utility.

In Fig. 3, the tasks completion ratio before maximum tolerable delay under the proposed algorithm is close to GBA and higher than RBA. Considering that GBA always chooses the vehicle with maximum remaining computing resources, which makes tasks accomplished before the deadline with greater probability. In the proposed algorithm, the utility will be negative if a task cannot be accomplished before deadline, and the optimal policy ensures more tasks finished on time.

Fig. 4 and Fig 5 illustrate the average completion time of tasks with different maximum tolerable delays under different traffic densities. When there are fewer SVs available compared to the number of tasks, the completion time of each type of tasks under the three algorithms seems to be close. In the proposed algorithm, the completion time of tasks with $\tau = \{0.5, 1, 1.5\}$ is slightly less than the other algorithms. When there are enough SVs available, the completion time of tasks with smaller τ is also less than the other two algorithms. In addition, comparing to GBA and RBA, the difference among the completion time of different types of tasks under our proposed algorithm is larger, which ensures the total utility of offloading tasks much closer to the maximum.

VI. Conclusion

In this paper, we have investigated the problem of distributed V2V computation offloading. The problem of task offloading is formulated as a sequential decision making problem. Based on reinforcement learning, we have proposed a task offloading scheme that both considers the incentive of sharing idle computing resources and the cost of offloading tasks. Simulation results have demonstrated that our proposed scheme has better performance under different vehicle densities compared to the other regular algorithms.

Acknowledgment

This work was supported by the National Key Research and Development Program of China under Grant 2016YFB0100902, the National Natural Science Foundation China under project 61971257, China Postdoctoral Science Foundation under grant 2018M640130 and special grant 2019T120091.

References

[1] J. Du, C. Jiang, A. Benslimane, S. Guo, and Y. Ren, “Stackelberg differential game based resource sharing in hierarchical fog-cloud computing,” in *IEEE Global Commun. Conf. (GLOBECOM’19)*. Waikoloa, Hawaii, USA, 9-13 Dec. 2019.

[2] S. Abdelhamid, H. S. Hassanein, and G. Takahara, “Vehicle as a resource (vaar),” *IEEE Network*, vol. 29, no. 1, pp. 12–17, Jan. 2015.

[3] Y. Sun, X. Guo, J. Song, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, “Adaptive learning-based task offloading for vehicular edge computing systems,” *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3061–3074, Apr. 2019.

[4] J. Feng, Z. Liu, C. Wu, and Y. Ji, “Ave: Autonomous vehicular edge computing framework with aco-based scheduling,” *IEEE Trans. Veh. Technol.*, vol. 66, no. 12, pp. 10 660–10 675, Dec. 2017.

[5] Z. Su, Y. Hui, and T. H. Luan, “Distributed task allocation to enable collaborative autonomous driving with network softwarization,” *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2175–2189, Oct. 2018.

[6] M. Liwang, S. Dai, Z. Gao, Y. Tang, and H. Dai, “A truthful reverse-auction mechanism for computation offloading in cloud-enabled vehicular network,” *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4214–4227, Jun. 2019.

[7] J. Du, C. Jiang, H. Zhang, Y. Ren, and M. Guizani, “Auction design and analysis for SDN-based traffic offloading in hybrid satellite-terrestrial networks,” *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2202–2217, Oct. 2018.

[8] J. Zhao, Q. Li, Y. Gong, and K. Zhang, “Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks,” *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944–7956, Aug. 2019.

[9] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Thirtieth AAAI Conf. on Artificial Intell.* Phoenix, Arizona, USA, 12-17 Feb. 2016.

[10] J. Wang, C. Jiang, H. Zhang, Y. Ren, K. Chen, and L. Hanzo, “Thirty years of machine learning: The road to pareto-optimal wireless networks,” *IEEE Commun. Surveys & Tutorials (DOI:10.1109/COMST.2020.2965856)*, 2020.