Deep Reinforcement Learning-based Offloading for Latency Minimization in 3-tier V2X Networks

Hieu Dinh*, Nang Hung Nguyen*, Trung Thanh Nguyen*,
Thanh Hung Nguyen*, Truong Thao Nguyen†, Phi Le Nguyen*‡
*School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Vietnam
{hieu.dv170072@sis, hung.nn184118@sis, thanh.nt176874@sis, hungnt@soict, lenp@soict}.hust.edu.vn
†The National Institute of Advanced Industrial Science and Technology (AIST), Japan; nguyen.truong@aist.go.jp
‡Corresponding author: lenp@soict.hust.edu.vn

2022 IEEE Wireless Communications and Networking Conference (WCNC) | 978-1-6654-4266-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/WCNC51071.2022.9771583

***Abstract*—Multi-access edge computing (MEC) is seen as an effective technique for decreasing service latency in a V2X network by offloading computational activities. With MEC, a three-tier offloading architecture can be developed, where a vehicle can offload computational tasks to a cloud by communicating with a base station (gNB) and Road Side Units (RSUs). In this paper, we focus on three-tier V2X networks which rely on three offloading paths: Vehicle-to-Infrastructure (i.e., vehicle to RSU), Vehicle-to-Cloud (i.e., vehicle to gNB), and Infrastructure-to-Cloud (i.e., RSU to gNB). We propose an offloading strategy based on deep reinforcement learning with the goal of reducing the average latency of tasks. To be more specific, we leverage the Deep Q Network to estimate the goodness of action-state value to determine the offloading decision. We also propose a novel exploration scheme and a new model training strategy. The experimental findings indicate that our proposed offloading method outperforms the state-of-the-art, particularly in critical circumstances characterized by a high rate of vehicle arrival or packet generation.**

***Index Terms*—V2X, Offloading, RSU, 3-tier, MEC, Deep reinforcement learning.**

I. Introduction

In recent years, the advancement of information technology and telecommunication infrastructure has enabled more and more high-performance computing applications, such as self-driving cars, traffic bright, to be integrated into vehicles. Such applications require considerable computational resources. Moreover, many of these applications are time-sensitive that usually require to be processed in real-time. Therefore, handling tasks directly on vehicles becomes a challenge due to the limited computing power of vehicles. To this end, cloud computing has emerged as a potential solution. The tasks from the vehicles can now be offloaded and processed at powerful cloud servers to utilize their enormous computing resources. However, this approach suffers from some inherent drawbacks. The first one is the bottleneck at the cloud servers. As all the tasks are offloaded to the cloud servers, they may be stuck here and causes a significant delay. The second obstacle resides in the long distance between the cloud servers and the vehicles that causes a significant delay in transmitting data.

To this end, a distributed approach based on Mobile Edge Computing (MEC) has been utilized. Rather than having vehicle tasks offloaded and processed at cloud servers, they will be moved to computing units along the side of the road (namely Road Side Units, RSU) instead. On the one hand, RSUs are placed close to vehicles, allowing vehicles to quickly offload tasks to the RSU, thereby reducing transmission latency. On the other hand, as the tasks from vehicles have been distributed to all RSUs, the bottleneck phenomenon caused by the cloud computing approach can be resolved.

Despite advantages, the MEC-based offloading approach still copes with some limitations. The main drawback of this approach is that the computing resources of RSUs are limited and much smaller than that of cloud servers. Besides, RSUs also suffer from small transmission range. Therefore, when the number of tasks is large, using edge computing alone can not satisfy the demand. Consequently, combining both the MEC and cloud computing to exploit their complementary advantages is a potential solution for offloading in V2X.

In the literature, many efforts have been devoted to investigate the offloading problem in V2X networks. Early works focused on two-tier model, which uses either cloud computing or MEC in offloading. The authors in [1]–[3] considered the MEC-enabled networks. They proposed offloading algorithms to maximize the long-term utility maximization [1], minimize the computation and communication overhead [2], and energy consumption reduction [3]. In [4]–[6], the authors investigated the task execution collaboration between platoons and a MEC server, i.e., that could send tasks to other platoon members, the MEC server, or process locally.

Recently, some works try to leverage MEC and cloud computing's complementary advantages for offloading by simultaneously using two paths: Vehicle-to-RSU and RSU-to-gNB [7], [8]. The authors in [7] aimed to maximize the system utility by optimizing both the offloading strategy and resource allocation. In [8], the authors proposed an optimization algorithm for the traffic and capacity allocation problem in a three-tier V2X. The authors in [9] considered vertical and horizontal offloading and formulated the targeted problem under mixed-integer nonlinear programming. They then use the branch-and-bound technique to find the optimal solution. The authors in [10] proposed and mathematically formulate the 3-tier V2X network that considers three offloading paths (vehicle-to-gNB, vehicle-to-RSU) RSU-to-gNB). To minimize the average task latency, they tried to optimize the offloading probability with a constraint of Poison distributions and a predetermined parameter. The proposed algorithm thus can not handle the cases when the vehicle and packet have a different distribution or dynamic network.

In this paper, we focus on offloading scheme in the three-tier V2X as in [10] while handling the network's dynamic and uncertainty. Our objective is to minimize the average task latency (i.e., including both the transmission and processing latency). Specifically, we investigate a V2X model, where the vehicle arrival, task generation processes, and packet size do not follow any specific pattern. Moreover, all network parameters are not given in advance. We then use deep reinforcement learning to help the vehicles and RSUs learn from their experience and make the right offloading decisions. Specifically, we exploit the Deep Q Network (DQN) to estimate the goodness of action-state value. Based on that, every agent will make the appropriate offloading decision. We propose a novel action selection scheme that uses the Sigmoid function. We also present a new training strategy for DQN to alleviate the unreliability of the DQN model at the very first steps. We conduct extensive experiments to evaluate and compare the performance of the proposed offloading scheme with exiting approaches. Experimental results show that our proposed offloading algorithm outperforms the state-of-the-art, especially when the network is under critical situations with a high vehicle arrival or packet generation rate.

II. Preliminaries

A. Q-learning

Q-learning is one of the most famous off-policy tabular methods used in Reinforcement Learning [11]. It rates the rationality of a given state-action pair by a state-action value and updates the state-action values of all the possible state-action pairs by the time. Specifically, it uses a table (called the Q-table) to map the state-action space $\mathcal{S}$ and $\mathcal{A}$ to the numeric space, i.e., $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. After the state-action values converge, all the actions of the agent are guided by this value-based policy. The simplest version, i.e., one-step Q-learning, adheres to the following update rule:

$$\begin{aligned} Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha[R_{t+1} \\ +\gamma \max_{A_{t+1}} Q(S_{t+1}, A_{t+1})], \end{aligned}$$

where $Q(s, a)$ denotes the state-action value of a specific action a in action space $\mathcal{A}(s)$, s is the state in which a happens, and R denotes the reward signal received from the environment. The subscript t tells the time-step in which the value is observable. The learning rate α and discount factor γ determine the algorithm's learning speed. These two parameters are scalar values in the range $(0, 1)$. According to this update rule, the agent first performs an action $A_t \in \mathcal{A}(S_t)$ then observes the reward signal as a scalar at the subsequent time step R_{t+1} and finds itself in a new state S_{t+1}. Afterwards, the agent updates its Q-table. The update process is delayed by one time-step, hence, one-step Q-learning.

However, tabular methods are limited within their world-view representation: explicit pairs of states and actions as a table. As the state space grows large, it is unfeasible for Q-learning-based methods to learn a decent policy within the reasonable time. Deep Q-Network (DQN), thus, came up as an inevitable solution.

B. Deep Q Network

Deep Q-Network (DQN) is the abstract version of the vanilla Q-learning which can address the problem of continuous spaces and extremely large sets of state-action pairings, [12]. In DQN, the mapping from the state space to the action-value space is presented as a nonlinear neural network which enables the agent to learn highly complicated mapping functions. DQN combines off-policy learning with bootstrapping and function approximation, which inevitably causes the deadly-triad problem introduced in [13]. To address this problem, the DQN architecture consists of two structure-identical function-separated networks. One is used to make decisions and gets updated more frequently, and is called the main network. The other is a periodically synchronized mirror of the main network, which is more stable and is called the target network. DQN also retains a vast buffer of prior experience and samples data from it (experience replay). As DQN is an off-policy method, it is valid to utilize this technique to reduce the bias of the samples. With an adequate experience buffer, DQN can avoid bias effectively. In details, the buffer size M records quartets of $(S_t, A_t, R_{t+1}, S_{t+1})$ at every time-step. The records then randomly taken out in batches of size N to train the main network as minimizing the following loss function $\mathcal{L}$:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left[R_t + \gamma \max_{A_{t+1}} Q'(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]^2, \quad (1)$$

where $Q'(\cdot)$ and $Q(\cdot)$ denote the outcome from the target network and the main network, respectively. After some m times update, the target network copies the weights of the main network, which makes it the m-update delayed mirror of the main network.

III. Proposal

A. Network model

In this paper, we utilize the network model that was initially proposed and mathematically formulated in [14]. There are three main components in the model: vehicles (**V**), infrastructures (**I**) and the network or cloud (**N**). This structure enables three bidirectional communication links: V2I, V2N, and I2N, which stand for vehicle-to-infrastructure, vehicle-to-network, and infrastructure-to-network, respectively.

In detail, we consider an urban area where N Road-Side Units (RSUs or infrastructures) are located along the roadsides and the vehicles are running on the road. Moreover, we also set up one cloud server (hereafter we name it gNB) at the center of the area (as in Figure 1). Each vehicle generates computational tasks that follow a random distribution. Due to the vehicles' limited computational capability, the tasks must be indirectly processed either at an RSU or the gNB. Therefore, RSUs and gNB play roles of processing devices whereas vehicles are terminal ends. After being processed at processing devices, the tasks are directly sent back to the vehicle based on three types of possible transmission routes. ① Tasks that are transmitted directly to the gNB, processed then sent back to the vehicle. ② Tasks that are transmitted directly to an RSU, processed at

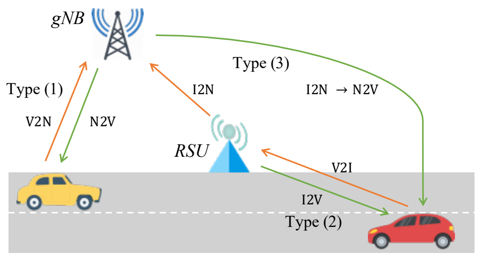

Fig. 1: V2X Network Model

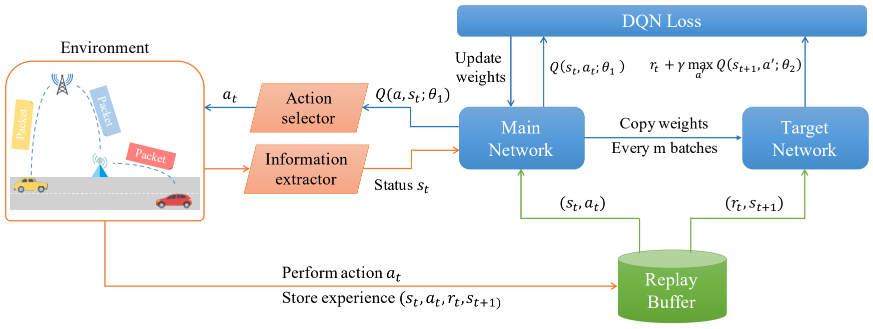

Fig. 2: DQN-based offloading mechanism

that RSU then sent back to the vehicle; and (3) tasks that are indirectly transmitted to the gNB via the I2N link, processed at the gNB then sent back to the vehicle. It is notable that while (1) and (2) involve one communication link (either V2I or V2N), (3) requires two communication links (V2I and I2N) where the RSUs are bottlenecks. In this work, our goal is to learn a routing policy that minimizes the total latency of all tasks transmitted throughout the networks and their drop ratio.

B. Q-learning based modelling

This section introduces the definition of the terms in the Q-learning framework concerning our targeted problem.

1) Environment and agents: The network serves as the environment in our model, with each vehicle and RSU acting as an agent. We take a model-free approach, which eliminates the requirement for agents to acquire prior knowledge about their environment. Agents interact with their environment by offloading computational tasks and receiving processed results. Packet-related metrics such as delay and packet drop ratio provide observation of the environment.

2) Agents: There are two types of action spaces, one for vehicles and the other for RSUs. When a vehicle needs to process a task, it may offload it to the RSU (referred to *RSU_offloading*) or gNB (*gNB_offloading*). Meanwhile, when an RSU receives a task offloaded from a vehicle, it has the option of processing the task locally (*local_process* or sending it to the gNB (*gNB_offloading*).

3) States: A state is represented by a vector that carries information about the status of vehicles, remote sensing units, and tasks. These information can be grouped into: computational resource-related, communication-related, and processing-related. The first group includes details on the computer resources needed to process the packet, such as the packet size and the number of CPU cycles required. The second group comprises estimates for the amount of time to offload jobs and receive the results. The last one offers information about the time it will take to complete the task.

4) Reward: The reward signal quantifies an action's goodness and is calculated using the reward function. An agent will learn the best policy by optimizing the cumulative reward, i. e., promoting desirable actions while penalizing bad ones. Because our goal is to minimize task latency, we design the reward function based on the total time required to transmit and process the task. Let us denote by A_t the action corresponding to the task J_t, and $T(J_t), P(J_t)$ the time for transmitting and processing the task, respectively. The reward function is mathematically defined as follows.

$$\mathcal{R}(A_t) = \begin{cases} -(T(J_t) + P(J_t)), & \text{if } J_t \text{ is not dropped} \\ -C, & \text{otherwise,} \end{cases} \tag{2}$$

where C is the maximum delay of all tasks processed so far. The rationale behind the design of the reward function is that actions that cause task cancellation are considered undesirable thus it is punished by a large negative reward. On the other hand, other actions are rewarded by the negative of the task's latency. Specifically, the action resulting in a more negligible latency obtains a higher reward. Here we use the negative rewarding scheme, which helps to stimulate the exploration process in our reinforcement learning scheme.

C. DQN-based offloading

1) DQN architecture: We use the standard DQN architecture with a distributed approach. Each vehicle and RSU has its own DQN network, which is in charge of determining the offloading action. Because a vehicle moves at a relatively high speed, we find that maintaining a small network in each vehicle is more computationally efficient. Therefore, the vehicle DQN model has only one modest hidden dense layer. An RSU, on the other hand, operates ceaselessly during the simulation period, thus its network should be large enough to learn such a massive amount of data. If the network is overly vast, however, learning would take an RSU an inordinate amount of time. For these reasons, we create an RSU's primary network, which consists of two large hidden dense layers.

Fig.2. shows the detail of the proposed offloading mechanism. Besides the basic modules of DQN, such as the Q-networks, target network, and the replay buffer, we propose two additional modules, namely information extractor and action selector. The information extractor is responsible for extracting the state representation information as presented in Section III-B3. The state information is then passed to the Q-network to derive the Q-values of the actions. The calculated Q-values are fed into the action selector, which selects the most appropriate action based on the algorithm proposed in Section III-C2. One of the most challenging issues in using DQN is handling the very first iterations. As the Q network is usually initialized with random weights, Q-values calculated in the first iterations are unreliable. To this end, we propose a novel DQN training strategy in Section III-C3.

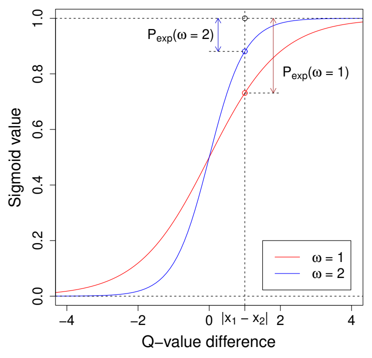

Fig. 3: SEM mechanism

Algorithm 1 SEM-based selection

```
Input: State-action values $x_1$, $x_2$
Output: Action to perform
 1: $a \leftarrow$ None
 2: $\Delta_x \leftarrow |x_1 - x_2|$
 3: $P_{exp} \leftarrow 1 - \frac{1}{1+e^{-\Delta_x}}$
 4: $\varepsilon \sim (0, 1)$
 5: if $\varepsilon \leq P_{exp}$ then
 6:     $a \leftarrow \text{argmin}\{x_1, x_2\}$
 7: else
 8:     $a \leftarrow \text{argmax}\{x_1, x_2\}$
 9: end if
10: return $a$
```

TABLE I: Simulation parameters

Factor	*Value*
The gNB's CPU	256 GHz
RSU's CPU	64 GHz
CPU cycles for a task	0.2 GHz
Mean packet size	500 kb
RSU-gNB's link bandwidth	10 Gbps
Vehicle-RSU's link bandwidth	1 Gbps
Vehicle-gNB's link bandwidth	500 Mbps
Vehicle arrival interval ($\frac{1}{\lambda_v}$)	$5 \sim 9$
Task emergence rate (λ_d)	$70 \sim 100$
Road length	1500 m
Vehicle's speed	12 m/s

2) Action selection strategy: The action selector uses the Q-values estimated by the main network to choose the most appropriate action. A trivial way is to select the action whose expected return is the highest. However, this greedy strategy may result in a non-exploration dilemma where the agent relies only on experience instead of exploring new knowledge. To this end, we propose a novel exploration approach named *Sigmoid-based Exploration Mechanism (SEM)*. In SEM, the exploration probability is dynamically adjusted based on experience. Specifically, we decrease the exploration probability if the gap between actions' rewards is significantly large and increase otherwise. When the variance between the actions' rewards is large, it means that some actions are significantly better than others. Therefore, decreasing the exploration (hence increasing the exploitation) will increase the chance to select the best action. In contrast, when the gap between the actions' rewards is small, there is no significant difference between actions' quality. Therefore, by increasing the exploration probability, the agent can learn about many actions instead of only focusing on the best ones.

There are two agent types (vehicles and RSUs) in our investigated model. Each agent type possesses two actions. *RSU_offloading* and *gNB_offloading* are for vehicles, while *local_processing* and *gNB_offloading* are for RSUs. Let a_1 and a_2 denote the two actions, respectively. Moreover, suppose that x_1 and x_2 are the action values of a_1 and a_2 at the current state. Then, the probability for performing exploration is mathematically defined as shown in line 3 of Algorithm 1. Where ω is a tunable parameter. The *sigmoid* function is selected because its value ranges from 0 to 1. Moreover, the greater the value of the variable $|x_1 - x_2|$, the smaller the value of P_{exp}. Besides, the gradient of the function can be adjusted by tunning the value of ω. Figure 3 illustrates the shape of SEM exploration function when varying ω. We can see that with the same value of $x_1 - x_2$, increasing of ω results in a smaller exploration probability. Moreover, the slope of the σ function also increases in that case. Decreasing the ω value results in the smoother variation of the sigmoid value. When $\omega = 0$, the exploration probability becomes a constant which does not depend on the value of $|x_1 - x_2|$. In contrast, when ω is sufficiently large, P_{exp} asymptotically approaches 0, and the agent does not perform exploration.

3) Training process: When training a DQN network, the target network is used as the reference point. The network parameters are then updated using the Gradient Descent method, which minimizes the difference between the Q value predicted by the main network and the targeted network's calculated Q value. However, because the target network is initialized randomly during the initial iterations, its Q values do not accurately reflect the goodness of actions. As a result, using DQN's default training strategy results in an unstable network that is difficult to converge to the optimal parameter set. To overcome this shortcoming, instead of using the targeted network as a reference point in the first N iterations, we'll use a Poisson model [14] to mathematically model the task latency and use the theoretically estimated latency as a reference. Furthermore, after the first N iterations, we will utilize the standard method of updating the weights, which minimizes the loss between the prediction network and the target network.

IV. Performance evaluation

A. Experiment methodology

We evaluate the performance of the proposed method and compare it with two approaches. In the first approach, we used the fixed offloading probabilities (named as Constant Offloading Setting or COS). Specifically, we vary the probabilities for offloading to RSU (denoted as p_R) and gNB (indicated as p_L), i.e., vehicles offload p_L% of the tasks to the gNB, $(1 - p_L)$% of the tasks to RSU while RSUs offload p_R% of the tasks to the gNB. We also use the Multi Agents Bandit (MAB) algorithm as the second competitor.

The target of this work is to reduce the amount of time to complete a task. Thus, we use the average latency and packet loss ratio as the two metrics in this evaluation. We simulate the network using an in-house simulator written in Java. In which, vehicle arriving and packet generation follows the Poisson process. We also conduct various experiments to investigate the impacts of the packet generation rates, the number of the RSUs, and the vehicle arrival rates to the average latency. The packet generation rate is varied from $\frac{1}{100}$ to $\frac{1}{70}$. The number of the RSUs changes from 5 to 9 while the vehicle arrival interval is set from 5 to 9. The average packet size ranges from 0.6 to 0.9. Other system parameters are given in Table I, in which the configurations of the gNB, RSU, and vehicles

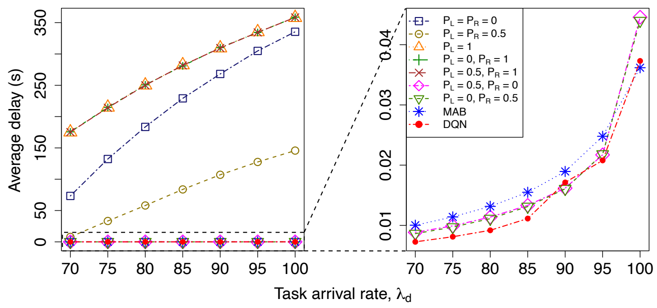

Fig. 4: Impacts of task arrival rate on average latency

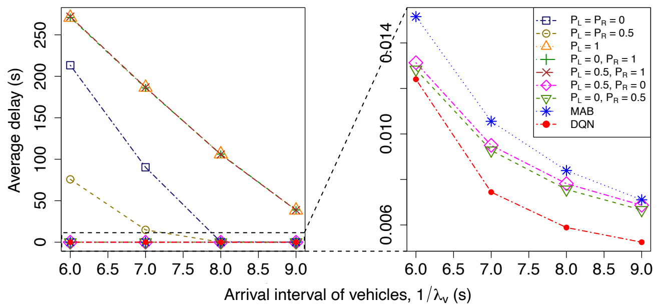

Fig. 5: Impacts of vehicle arrival rate

TABLE II: Packet drop ratio when varying the task arrival rate

Task arrival rate (%)	DQN	MAB	COS						
			(0.0, 0.0)	(0.0, 0.5)	(0.0, 1.0)	(0.5, 0.0)	(0.5, 0.5)	(0.5, 1.0)	(1.0, 0.0)
70	**0.00007**	0.00018	0.82748	0.00025	0.69154	0.00024	0.05966	0.69071	0.69016
75	**0.00007**	0.00021	0.89533	0.00028	0.72820	0.00026	0.22015	0.72710	0.72760
80	**0.00007**	0.00025	0.92065	0.00031	0.75176	0.00031	0.34510	0.75263	0.75249
85	**0.00010**	0.00031	0.93341	0.00035	0.76937	0.00035	0.42376	0.77017	0.77036
90	**0.00017**	0.00041	0.94107	0.00042	0.78238	0.00042	0.46217	0.78243	0.78157
95	**0.00018**	0.00053	0.94674	0.00051	0.79103	0.00052	0.48164	0.79199	0.79196
100	**0.00039**	0.00080	0.94990	0.00076	0.79947	0.00077	0.49310	0.79938	0.79866

TABLE III: The packet drop ratio when varying the vehicle arrival rate

Vehicle arrival rate	DQN	MAB	COS						
			(0.0, 0.0)	(0.0, 0.5)	(0.0, 1.0)	(0.5, 0.0)	(0.5, 0.5)	(0.5, 1.0)	(1.0, 0.0)
6	**0.00012**	0.00039	0.92979	0.00054	0.76419	0.00055	0.40599	0.76462	0.76438
7	**0.00006**	0.00019	0.85990	0.00030	0.70178	0.00030	0.10800	0.70225	0.70203
8	**0.00005**	0.00015	0.01451	0.00028	0.57041	0.00030	0.00020	0.56798	0.57067
9	**0.00005**	0.00011	0.00346	0.00020	0.26210	0.00019	0.00010	0.26186	0.26015

are proposed by [15]. Moreover, the RSUs were placed evenly in a straight road.

B. Numerical results

1) Impacts of task arrival rate: In this experiment, we investigate the impacts of task arrival rate on the performance of the proposed method. We fixed the number of RSUs to 5. The arrival interval of vehicles followed the Poisson process with an average value of 5 seconds. The average packet size was set to 550 kb. Tasks were generated according to the Poison process with an average number of tasks from 70 to 100 per second. The experimental results are shown in Fig.4 and Table II. It can be seen that both the latency and the packet drop ratio increase as the task generation frequency increases. This phenomenon is expectable because when the task generation frequency increases, the traffic imposed on the communication links and the workload burden on the RSU and gNB also increases. DQN achieves the best performance regarding both the latency and packet drop ratio. Concerning the task latency, DQN outperforms the others when the task arrival rate is smaller than 90. Beyond that, the latency caused by DQN is similar to that of COS with (P_R, P_L) set to $(0.5, 0)$ or $(0, 0.5)$. It is worth noting that, although COS with (P_R, P_L) of $(0.5, 0)$ or $(0, 0.5)$ attain a similar latency to DQN, their packet drop ratios are much higher than that of DQN. Specifically, COS$(0.5, 0)$ and COS$(0, 0.5)$ increase the drop ratio by $3.2\times$ on average and by $4.2\times$ in the worst case compared to DQN. In comparison with MAB, DQN reduces the average latency by at least 31% and reduces the packet drop ratio by more than 6%.

2) Impacts of vehicle arrival rate: In this experiment, we study the impacts of the vehicle arrival rate. We fixed the number of RSUs to 5, the task arrival rate to 100, and the packet size to 550. We changed the average arrival interval of vehicles from 6 to 9 seconds and see the variation of the task latency and packet drop ratio. The results are illustrated in Fig.5 and Table III. We can see that the task latency and packet drop ratio decrease as the arrival rate of vehicles increases. The reason is that the more the vehicles, the more the tasks generated; thus, the more the load imposed on both the communication channels and RSUs and gNB. It can be seen that DQN always achieves the smallest latency and packet drop ratio. Similar to the experiment shown in the previous section, among the remaining algorithms, MAB and COS$(0.5, 0)$ and COS$(0, 0.5)$ attain the best performance. When the vehicles arrive with a very high frequency, the latency of MAB and COS$(0.5, 0)$ and COS$(0, 0.5)$ is close to DQN. However, as the vehicle arrival rate decreases, DQN improves latency much better. Therefore, the gap between DQN and the others increases quickly as the frequency of vehicle appearance decreases. In particular, when the vehicle arrival interval is 9 s, DQN reduces latency 26.6% compared to MAB and 23.6% compared to COS.

Concerning the packet drop ratio, DQN has superior performance compared to other algorithms. In all cases, the packet ratio of DQN does not exceed 0.3%. Moreover, the packet drop ratio caused by DQN significantly decreases when decreasing the vehicle arrival rate. When the vehicle arrival interval is greater than or equals 8 seconds, DQN guarantees 99.97% packets successfully processed. The packet drop ratio of DQN is smaller than $0.43\times$ that of MAB and $0.23\times$ that of the best setting of COS (i.e., COS$(0.5, 0)$ and COS$(0, 0.5)$).

3) Impacts of the number of RSUs: The vehicle followed the Poisson process with an average arrival interval of 5 seconds. The task generation rate was set to 90, and the average packet size was set to 550 kb. We changed the number of RSUs from 6 to 9. The experimental results are depicted in Fig.6 and Table IV. With $P_R = 1$ or $P_L = 1$, the performance of COS remains constant regardless of the number of RSUs. This is because, in these configurations, all tasks are transferred to gNB. The other methods achieve a smaller latency and packet drop ratio when increasing the number of RSUs. DQN outperforms the others concerning both latency and packet drop ratio. DQN reduces the latency by 9.5% on average and

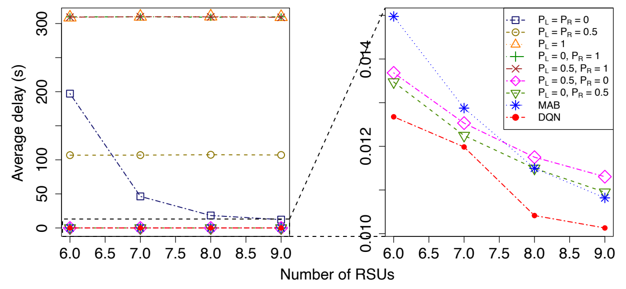

Fig. 6: Impacts of the number of RSUs

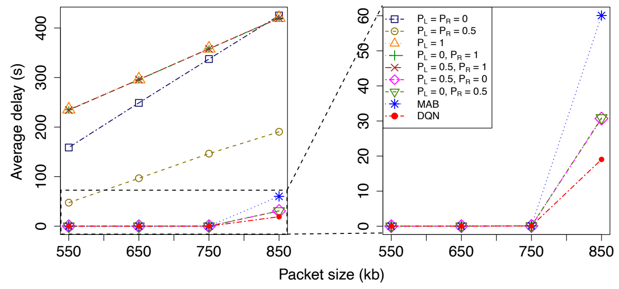

Fig. 7: Impacts of packet size on the task latency

TABLE IV: Packet drop ratio when varying the number of RSUs

The number of RSUs	DQN	MAB	COS						
			(0.0, 0.0)	(0.0, 0.5)	(0.0, 1.0)	(0.5, 0.0)	(0.5, 0.5)	(0.5, 1.0)	(1.0, 0.0)
6	**0.00019**	0.00043	0.89377	0.00045	0.78274	0.00045	0.46100	0.78270	0.78186
7	**0.00018**	0.00035	0.26653	0.00038	0.78201	0.00039	0.46188	0.78248	0.78175
8	**0.00016**	0.00042	0.014565	0.00043	0.78208	0.00043	0.46347	0.78148	0.78267
9	**0.00016**	0.00034	0.13623	0.00036	0.78235	0.00036	0.46233	0.78135	0.78178

TABLE V: Packet drop ratio when varying the packet size

Packet size (kb)	DQN	MAB	COS						
			(0.0, 0.0)	(0.0, 0.5)	(0.0, 1.0)	(0.5, 0.0)	(0.5, 0.5)	(0.5, 1.0)	(1.0, 0.0)
850	**0.11326**	0.25889	0.95786	0.19117	0.81496	0.19082	0.49943	0.81455	0.81438
750	**0.00050**	0.00230	0.91992	0.00110	0.79877	0.00109	0.49436	0.79955	0.79987
650	**0.00014**	0.00042	0.93851	0.00047	0.77619	0.00050	0.44817	0.77581	0.77644
550	**0.00009**	0.00025	0.91363	0.00032	0.74214	0.00031	0.29447	0.74228	0.74321

by 15.3% in the best case, compared to MAB. Among all settings of COS, COS$(0.5, 0)$ and COS$(0, 0.5)$ attain the best performance. Even that COS$(0.5, 0)$ and COS$(0, 0.5)$ cause latency $1.1\times$ more than DQN.

Regarding the packet drop ratio, DQN guarantees more than 99.98% packets successfully processed. Meanwhile, the packet drop ratio of MAB is 0.038% on average and 0.042% in the worst case. The packet drop ratio of the best COS setting ranges from 0.035% to 0.046%.

4) Impacts of packet size: Finally, we analysis the impacts of packet size on the performance of the proposed method. In this experiment, we fixed the number of RSUs to 5. The average vehicle arrival interval was set to 5 s, and the task generation rate was set to 70. The average packet size was changed from 550 to 850 kb. Trivially, all the methods degrade when the packet size increases. However, the degradation speed of DQN is the slowest. DQN achieves a much better performance against the others, especially when the packet size is significantly large. For example, when the packet size is 850, DQN reduces the latency more than 68.3% compared to MAB and more than 37.9% compared to the best setting of COS. The reason is that in DQN, the decision to offload is based on several environment state factors, including the packet size. MAB and COS, on the other hand, don't take packet size into account.

V. Conclusion

We developed a DQN-based offloading mechanism to minimize the average latency of tasks. Our experiments demonstrated that the proposed approach can substantially decrease latency and packet loss ratio compared to existing methods. Expressly, in all scenarios, we can guarantee more than 99.9% of packets successfully delivered. Depending on the traffic load condition and Road Side Units, our approach reduced the task latency from 9.5% to 68.3% on average compared to the others.

Acknowledgment

This work was supported by JST, ACT-X Grant Number JPMJAX190C, Japan. This research is funded by the Hanoi University of Science and Technology (HUST) under project number T2021-PC-019.

References

[1] X. Chen et al. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet Things J.*, 6(3):4005–4018, 2019.

[2] J. Wang, D. Feng, S. Zhang, J. Tang, and T. Q. S. Quek. Computation offloading for mobile edge computing enabled vehicular networks. *IEEE Access*, 7:62624–62632, 2019.

[3] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE Access*, 4:5896–5907, 2016.

[4] X. Fan, T. Cui, C. Cao, Q. Chen, and K.S. Kwak. Minimum-cost offloading for collaborative task execution of mec-assisted platooning. *Sensors*, 19(847), 2019.

[5] T. Cui, Y. Hu, B. Shen, and Q. Chen. Task offloading based on lyapunov optimization for mec-assisted vehicular platooning networks. *Sensors*, 19(4974), 2019.

[6] H. Wang, X. Li, H. Ji, and H. Zhang. Federated offloading scheme to minimize latency in mec-enabled vehicular networks. In *Proc. IEEE GLOBECOM Workshops*, pages 1–6, 2018.

[7] 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.*, 68(8):7944–7956, 2019.

[8] Y. Lin, Y. Lai, J. Huang, and H. Chien. Three-tier capacity and traffic allocation for core, edges, and devices for mobile edge computing. *IEEE Trans. Netw. Service Manag.*, 15(3):923–933, 2018.

[9] M. Thai, Y. Lin, Y. Lai, and H. Chien. Workload and capacity optimization for cloud-edge computing systems with vertical and horizontal offloading. *IEEE Trans. Netw. Service Manag.*, 17(1):227–238, 2020.

[10] P. L. Nguyen et al. Modeling and minimizing latency in three-tier v2x networks. In *Proc. GLOBECOM 2020*, pages 1–6, 2020.

[11] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[13] Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad, 2018.

[14] Phi Le Nguyen, Ren-Hung Hwang, Pham Minh Khiem, Kien Nguyen, and Ying-Dar Lin. Modeling and minimizing latency in three-tier v2x networks. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE, 2020.

[15] Z. Ning, P. Dong, X. Kong, and F. Xia. A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things. *IEEE Internet of Things J.*, 6(3):4804–4814, 2019.