On the Possibility of Bringing the Unmanned Aerial Vehicles to the Network Edge

Mohammad Firouzi Eyal De Lara
Department of Computer Science, University of Toronto
{firouzi, delara}@cs.toronto.edu

Abstract— **The emerging area of edge computing is a motivation for deploying various applications on the edge of the network. In this work, we will assess the possibility of bringing drone delivery system to the network edge with focusing on a specific task that a drone must perform in delivering a package. We do not evaluate the problem in a traditional manner where the drone's decision system is hard-coded. Instead, we assume its decision system as a neural network which gives the drone the ability of learning tasks. However, our focus in this paper is not to train the drone to learn different tasks. We evaluate the possibility of putting the drone's decision system (i.e. the neural network) on the edge, rather only putting it only in the drone's body itself. This approach has benefits such as reducing the drone's battery consumption, and the possibility of leveraging concurrent decision making using mini-batches of data in multi-agent settings. With an assumption that drones are a company product and therefore have a same decision making system, we evaluate the possibility of deployment of this system in a simulated two-tier network using a trained drone.**

I. INTRODUCTION

Cloud computing is a technology to provide everything to clients as services through internet connection [1]. This technology provides required resources for client that can be used for various tasks such as running applications, or storing data. Nowadays businesses and clients use these resources to deploy various kinds of applications that require near-real-time computations. For example, in [2], authors proposed an architecture to deploy augmented reality (AR) algorithms over the local network. In [3], authors proposed a game service platform that can handle a multi-user real-time game screen and sound on the cloud and provides them for user through real-time streaming. Although cloud platforms provides various services, they are currently restricted in bandwidth limit and latency. These restrictions limit the applicability of using cloud services for sensitive real-time applications.

A proposed solution to this limited bandwidth and latency is to use edge computing, a variant of cloud computing where the resources are closer to the clients (on the network edge). This approach can effectively reduce client bandwidth consumption, and the latency of accessing their requirements. However, it is still not sufficient for many real-time applications. Several works have been focusing on reducing the existing gap through developing platforms for specific applications [4], [5]. There are also works that tackle this problem through developing optimization approaches for edge applications [6]. In this work, we focus on addressing the existing problems on deploying a package delivery system on the network edge, or on the cloud. We also state the advantages and disadvantages of this deployment on the network edge.

In a reinforcement learning problem, an agent interacts with an environment through its actions and its purpose is to learn how to behave in a way that its cumulative reward is maximized. Recent breakthroughs in reinforcement learning is a motivation to implement these agent decision making systems on the network edge for real-world applications. For example, one application which is also the focus of our work is a package delivery system that its agents are drones. Currently, there are companies such as Amazon, that are developing an infrastructure for this system. There are also research publications that propose reinforcement learning algorithms for drone controlling systems [7], [8]. In this work, the drone's landing task is particularly discussed. A similar approach can be done for other parts of this delivery system. Placing merely a neural network on the edge has been previously studied for other applications such as face recognition [10]. To the best of our knowledge, however, this problem has not been studied yet in a reinforcement learning setting.

Deploying the controlling system of these drones on the edge has several benefits. First, the drone battery consumption can effectively be decreased through using edge resources. At each time step, in a reinforcement learning setup, given the current environment state, the drone must take an action. This decision making consumes electrical power and deployment of a shared decision making system for several drones in a local place can reduce the energy consumption effectively. The ideal case is that a drone does not have any decision making system in its physical body, which we call a brainless drone. Second, the actions can be chosen concurrently on the edge using mini-batches of drone states. For development purposes, this in particularly useful in case of using neural networks as agent's decision making system since there are existing frameworks that can handle a group of samples concurrently. Third, this approach can be leveraged to handle multi-agent environments by implementing agents' collaborative decision making system on the network edge.

There are also several challenges with regard to this approach. For instance, the mobility patterns of the drone is particularly important and dismissing this challenge may result crash or fall down during the time that the drone is disconnected from the edge, or a cloud server. There are approaches such as service migration, or pre-migration [9] to address this issue. Also, during the disconnect times the drone must have a policy over its actions. One approach is that the agent keeps its position in air, or follows a hard-coded policy. This problem prevents the ideal case of having a completely brainless drone.

Rest of the paper is organized as follows: In the next section we describe the background on reinforcement learning that is used to train the simulated agent. In the section 3, we describe simulated environments for both the drone and the network. In section 4, we report our experimental results. In the last section, we provide the conclusion and future works.

II. Reinforcement Learning

A. *Problem Definition*

In this work, a reinforcement learning problem is formalized as a discrete-time partially observed Markov decision process (POMDP) defined with the tuple (S, A, T, R, O, γ). At each time step, the environment is in a state $s \in S$. The agent takes an action $a \in A$, gets a reward $r = R(s, a)$, and moves to a another state s' where s' is sampled from the transition probability $T(s'|s, a)$. In the state s', the agent sees an observation $o \in O$. This process is then repeated. For simplicity, in the following of the paper we assume agents observations at each state is the same as the state (i.e. a fully observed MDP). The agent's goal is to maximize its expected future discounted reward $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where r_t is the reward at the time step t, and γ is the discount factor.

B. *Q-learning*

Q-learning is an approach in reinforcement learning that learns a state-action value function. The optimal state-action value function $Q^*(s, a)$ is defined as maximum expected future reward achieved by any action policy chosen by the agent. More formally, for $(s, a) \in S \times A$, we have $Q^*s, a = \mathbb{E}_\pi[r_t|s_t = s, at = a]$, where π is the agents policy over its actions given a state (i.e. distribution over actions). The optimal state-action value function holds *bellman equation*:

$$Q^*(s, a) = \mathop{\mathbb{E}}_{s' \sim T}[r + \gamma \max_a' Q^*(s', a')|s, a]$$

Intuitively, bellman equation states that if the optimal $Q^*(s', a')$ for all the possibilities of the next state-action pair (s', a') is known, then the optimal policy at the state s' is to choose an action a' such that $r + \gamma \max_a' Q^*(s', a')$ is maximized. The intrinsic of many algorithms in reinforcement learning is to use the bellman equation as an iterative update $Q_{i+1}(s, a) = \mathbb{E}_{s' \sim T}[r + \gamma \max_a' Q_i(s', a')|s, a]$, where Q_{i+1} will be an estimation of the optimal state-action value function at the time $i + 1$ obtained from the estimation obtained from the current iteration. This *value iteration* algorithm converges to the optimal state-action value function $Q*$ as $t \to \infty$ [11], [12].

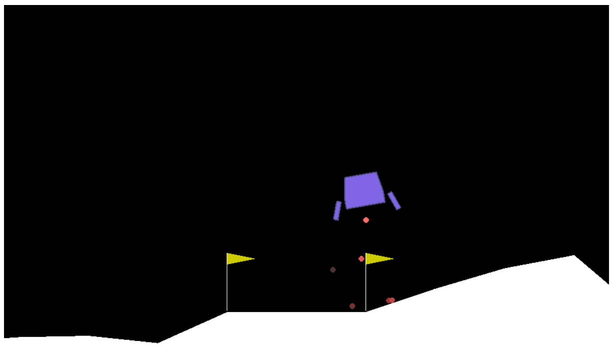

Fig. 1: LunarLander environment

C. *Deep Q-learning*

In many cases the Q function is modeled with a neural network which is referred as deep Q-learning. In this work, we train the agent using a variant of deep Q-learning with experience replay in which the agent's past experiences are stored in a buffer and is used for the future Q-function updates [11].

III. Environments

In this section, we describe the environment that we chose for our reinforcement learning problem and also explain the network structure that we used in our experiments.

A. *Environment*

One of the OpenAI gym's environments [13], LunarLander, simulates an agent in a aerial environment. In this environment, aim of the agent is to land between two specified flags on the ground. Agent's correct landing results a high reward. As the correct landing is faster, the agents gets more reward. The agent has three engines and uses them for its movement and balance keeping. Crashing with the ground results a high negative reward. Figure 2 shows a general view of this environment. At each step, the agent's state is an 8-dimensional vector, consisting of the agent's coordinate and other factors of the agent and the environment. At each step the agent can decide to turn on one of its engines or do nothing, resulting a 4-dimensional discrete action space.

B. *Nework Architeture*

We employed our simulated drone environments in a two-tier network architecture. The top tier nodes are the network core nodes, and cloud servers are chosen as a subset of core nodes. Second tier contains the edge nodes. And finally at the lowest level, the clients. in our experiments clients are drones. A network bandwidth of 10Mbps is used for this simulated network. To model drones, one simulated environment is created for each drone, and drones act independently of the other drones in their own environment. In our experiments, the number of drones in the aerial space is set to 25, the number of edge nodes is 5, and there is a node specified for the cloud services.

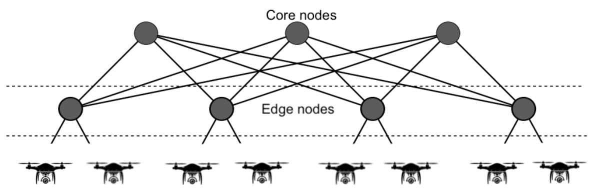

Fig. 2: Network architecture

IV. Experimental Results

We experimented several aspects of the possibility of putting drone's on the network edge. In our simulated network, the expected reward of these agents, and the rate of successful and fail landings are measured in different cases. We also tested our results with a simulated mobility pattern that drones can have, and different policies that they can have during the times that they are disconnected from their decision making system due to migration, or get late response from servers.

A. Expected Rewards

In this section, we compare the average rewards the agents can get in the landing task in cases that they send their state to an edge or a cloud node in order to get their current action. The case that the decision making system is in the agent's physical body is also considered. Since each agent must decide in real time, it should have a policy for situations that the edge or cloud server doesn't answer its query on time. We considered three simple policies for these cases: 1) agent keeps doing what it was doing before 2) agent selects a random action from its action space 3) agent do nothing and turn off the engines. The result for each of these policies is shown in the fig 3. It can be seen that in all three cases, the average reward on the cloud is negative. A negative reward in this environment means an that the agent will have an unsuccessful landing. The average reward on the edge is higher that the cloud in all three cases.

B. Successful Landing

According to the results of the previous section, it can be seen that the fixed policy for agent in the late responses or disconnected times can have effects on its expected future reward. Note that this policy is different from the policy π introduced in the section 2 which selects an action using a neural network. In this section, we compare the successful landing ratios. From figure 4, it is clear that implementing the neural network (i.e. agent's decision system) in the drone itself result highest successful landing probability due to absence of network conditions. Also, it is expected that in the edge case the success probability is higher than the cloud case. Moreover, the agent's fix policy has bigger successful rate changes on the cloud, which means implementing the agent's neural network on the cloud may be more challenging than implementing it on the edge.

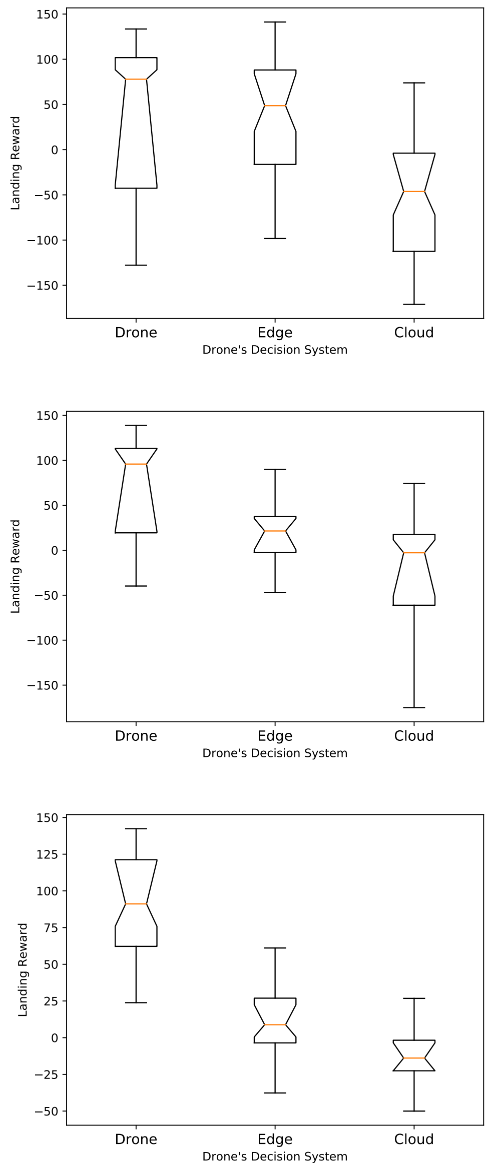

Fig. 3: Average reward for agents with the decision implemented on the drone, edge, or cloud with three different policies during no query response times: **top:** Agent keeps doing what it was doing. **mid:** Agent selects a random action. **bottom:** Agent does nothing.

V. Mobility simulation

In this section, we measure the effect of adding mobility to the drone's flying pattern. We fix the policy to keep the previous actions in disconnected times. The mobility is simulated with a Poisson process model. In the events resulted from the Poisson process model, a random agent changes its location from one edge node to another node, and in cases that its decision making system is not in its physical body the agent experiences a down time randomly sampled from a Gaussian distribution. Our experiments are done in a pessimistic manner. In our experiments, an agent switches its

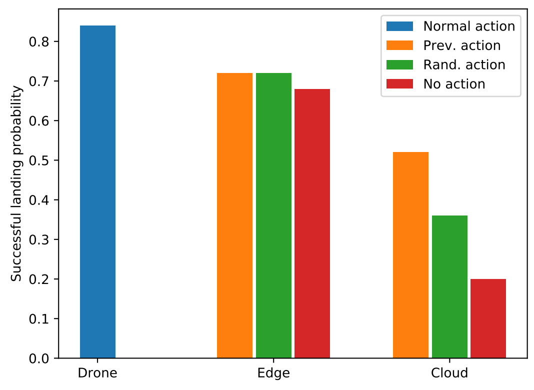

Fig. 4: Success landing probability of the agent

network with an average of less than a minute. The results are shown in the figure 5. The Poisson process mobility modeling affect the average reward for the both edge, and cloud. However, this effect is more significant for cloud and reduces the average landing reward more. It is also adds to the variance of the average landing reward which further indicates that the implementing the drone neural network on the cloud may be more challenging.

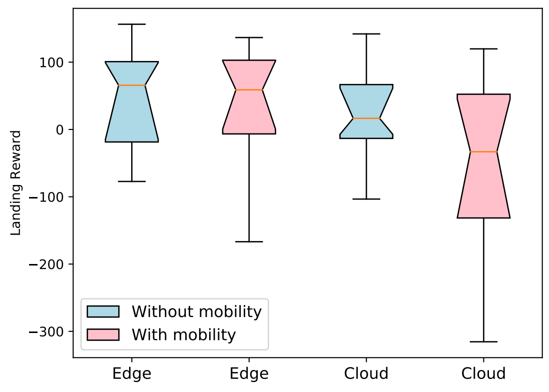

Fig. 5: Comparing the agent's average landing reward in two cases: 1) without mobility, 2) with a simulated mobility pattern

VI. Bandwidth Effects

The effect of bandwidth is also measured on the landing reward. It is clear that a lower bandwidth results higher network congestion and higher delay in responding the agent queries. We mentioned earlier that the maximum bandwidth of the network is chosen as 10Mbps. In this section, we measure the expected landing reward versus the network bandwidth. In case that the network bandwidth is close to 0Mbps, no matter where the decision making system is, on the edge or cloud, agent only has the choice to perform based on its fixed policy (e.g. keep doing a fixed action) which results a high negative reward. The effect of the bandwidth is shown in the figure 6

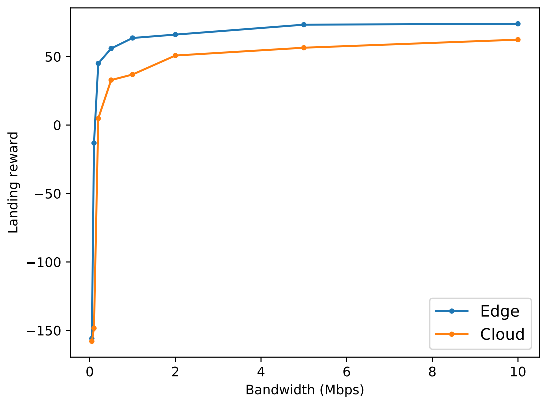

Fig. 6: The effect of the bandwidth limit on the average agent landing reward. The bandwidth range is from 0Mbps to 10Mbps

VII. Conclusion and Future Works

In this work, we evaluated the potential of placing neural networks as decision makers of reinforcement learning (RL) agents on the network edge. We focused on a specific RL environment in which a simulated drone tries to land in a specific region on the ground. We trained this agent using deep reinforcement learning techniques. Then the trained agent's decision making system were used in three different places on a two-tier network: 1) The agent itself 2) The network edge, 3) Cloud servers. We compared the average expected reward that the agent can get in different circumstances. Our experiments showed that implementing the agent's decision making system on the cloud is not optimized and may have side effects and can be challenging. Network's edge is a better option and results higher expected future reward for the agent. It is also beneficial in cases that multiple deep RL agents want to accomplish a task in collaboration with each other.

Testing the landing task with a real drone that uses a neural network decision making system is considered as a future work. Also, setting up the agent's neural network on a real edge location and controlling the agent from edge servers using mobile applications is considered as another future work. Due to the limitation of real-world experimental setup, the task of scaling to large number of agents is considered for simulated environments where a few thousands of agents are connected to the network. Although, the state space of the trained agent is not high and can be different from real world drones that should act based on a vision system.

Even though deep RL had a great progress in recent years, there are still ongoing research in stabilizing RL agents even in not complicated tasks. For example, defensing against [14], [15] crafted adversarial attacks against the agent's observation is an open problem and many works focused on solving this problem (particularly when the agant's observation is an image). There are other conditions such as the ability to perform in different environments which also make the problem of bringing these agents to perform real-world tasks challenging. This is a reason that the traditional planning algorithms are still used in practice. Bringing these reinforcement learning agents to the network edge is even more challenging due to the additional burden from the network conditions.

Even though our work is done for a specific landing task, there is no limitation in choosing drone's task. The same experiments can be done for other tasks such as picking a package up, or negotiation in air.

REFERENCES

[1] Bokhari M.U., Makki Q., Tamandani Y.K. (2018) A Survey on Cloud Computing. In: Aggarwal V., Bhatnagar V., Mishra D. (eds) Big Data Analytics. Advances in Intelligent Systems and Computing, vol 654. Springer, Singapore

[2] Schneider, Michael et al. Augmented reality based on edge computing using the example of remote live support. 2017 IEEE International Conference on Industrial Technology (ICIT) (2017): 1277-1282.

[3] Kim, Kyoung Ill, et al. "CloudBased Gaming Service Platform Supporting Multiple Devices." ETRI Journal 35.6 (2013): 960-968.

[4] Krner, Marc, et al. "Open carrier interface: an open source edge computing framework." Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies. ACM, 2018.

[5] Zhang, Tan, et al. "The design and implementation of a wireless video surveillance system." Proceedings of the 21st Annual International Conference on Mobile Computing and Networking. ACM, 2015.

[6] Tocz, Klervie, and Simin Nadjm-Tehrani. "A Taxonomy for Management and Optimization of Multiple Resources in Edge Computing." Wireless Communications and Mobile Computing 2018 (2018).

[7] Pham, Huy X., et al. "Autonomous uav navigation using reinforcement learning." arXiv preprint arXiv:1801.05086 (2018).

[8] Imanberdiyev, Nursultan, et al. "Autonomous navigation of UAV by using real-time model-based reinforcement learning." Control, Automation, Robotics and Vision (ICARCV), 2016 14th International Conference on. IEEE, 2016.

[9] Wang, Shangguang, et al. "A Survey on Service Migration in Mobile Edge Computing." IEEE Access 6 (2018): 23511-23528.

[10] Li, Dawei, et al. "Deepcham: Collaborative edge-mediated adaptive deep learning for mobile object recognition." Edge Computing (SEC), IEEE/ACM Symposium on. IEEE, 2016.

[11] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[12] Richard Sutton and Andrew Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.

[13] Brockman, Greg, et al. "Openai gym." arXiv preprint arXiv:1606.01540 (2016).

[14] Yuan, Xiaoyong, et al. "Adversarial examples: Attacks and defenses for deep learning." arXiv preprint arXiv:1712.07107 (2017).

[15] Kurakin, Alexey, Ian Goodfellow, and Samy Bengio. "Adversarial examples in the physical world." arXiv preprint arXiv:1607.02533 (2016).