Dynamic Resource Aware VNF Placement with Deep Reinforcement Learning for 5G Networks

Anestis Dalgkitsis*, Prodromos-Vasileios Mekikis*, Angelos Antonopoulos†
Georgios Kormentzas§ and Christos Verikoukis†
*Iquadrat Informatica S.L., Barcelona, Spain
†Telecommunications Technological Centre of Catalonia (CTTC/CERCA), Castelldefels, Barcelona, Spain
§Information and Communication Systems Engineering Dept., University of the Aegean, Samos, Greece
Email: {a.dalgkitsis, vmekikis}@iquadrat.com, {aantonopoulos, cveri}@cttc.es and gkorm@aegean.gr

***Abstract*—The increasing demand for fast, reliable, and robust network services has driven the telecommunications industry to design novel network architectures that employ Network Functions Virtualization and Software Defined Networking. Despite the advancements in cellular networks, there is a need for an automatic, self-adapting orchestrating mechanism that can manage the placement of resources. Deep Reinforcement Learning can perform such tasks dynamically, without any prior knowledge. In this work, we leverage a Deep Deterministic Policy Gradient Reinforcement Learning algorithm, to fully automate the Virtual Network Functions deployment process between edge and cloud network nodes. We evaluate the performance of our implementation and compare it with alternative solutions to prove its superiority while demonstrating results that pave the way for Experiential Network Intelligence and fully automated, Zero touch network Service Management.**

***Index Terms*—Deep Reinforcement Learning, Software Defined Networking, Virtual Network Functions, Live Migration**

I. Introduction

The explosive growth of the Fifth Generation (5G) networks and their complexity surpasses the limits of manual administration. The increasing demand for agile and robust network services has driven the telecommunication industry to design innovative network architectures that employ Network Functions Virtualization (NFV) and Software Defined Networking (SDN). NFV is the core enabling concept that proposes to decouple network functions from proprietary hardware appliances and emulates them in virtualized containers. Compared to traditional network functions implemented by dedicated hardware appliances, NFV has the potential to significantly reduce the operating and capital expenses and improve service agility. SDN aims to enable intelligent and centralized network control using software that is operating in consumer-grade hardware and not proprietary hardware appliances. According to the academia and industry, the combination of both concepts is expected to revolutionize network operations, not only by substantially reducing the cost [1], but also by introducing new possibilities for enterprises, carriers, and service providers.

The advances in NFV and cloud technologies have led to the emerging paradigm of Multi-access Edge Computing (MEC), which brings cloud computing capabilities and services close to the 5G base stations. Hence, it creates a new ecosystem that supports the flexible deployment of new applications and services close to the end-users, thus significantly reducing the overall delay. This opens up new possibilities in advanced use case scenarios such as ultra-Reliable Low-Latency Communication (uRLLC) and Enhanced Mobile Broadband (eMBB).

The strategic placement of Virtual Network Functions (VNFs), which can be composed of one or more Virtual Machines (VMs), to the most suitable location is directly tied to the availability of physical resources. It can greatly affect network performance, service latency, overall expenses, power consumption, and even carbon emissions. Furthermore, advanced VNF migration techniques support the spatial relocation of VMs from one Data Center (DC) to another with virtually no downtime [2]. This new feature, known as Live Migration, provides unique opportunities for the dynamic placement and life-cycle management of VNFs [3] [4].

The wide variety of challenges introduced by the disruptive deployment of 5G trigger the need for a drastic transformation in the way services are managed and orchestrated. It can become time-consuming to orchestrate services due to manual configurations around VNF integration. The intention is to have all operational processes and tasks, such as delivery, deployment, configuration, and optimization, executed automatically [5]. Since the world is already moving towards data-driven automation, technologies like Machine Learning (ML) can be employed to tackle part of the workload. New advancements in Deep Reinforcement Learning (DRL), will pave the way for a more intelligent and self-organizing networks [6].

DRL has provided an important breakthrough recently, not only by defeating human players in a plethora of board [7] and video games [8] but also numerous applications in industry, such as self-driving cars [9].

Despite the significant advancements in networking thanks to the introduction of 5G technologies, the deployment and orchestration tools still require additional research and development, to reach an acceptable level of automation. Since inefficient resource utilization might lead to severe performance degradation [10], there is a need for an automated, self-adapting orchestrating instrument that can manage resources wisely, taking into account as many relevant variables as possible [11]. Static scaling and migration, based solely on host system resources, overlook important factors such as End-to-End (E2E) latency and network performance that are

crucial for uRLLC services. An auspicious solution is the use of traditional Deep Learning (DL) algorithms for the prediction of future network states. However, due to the high complexity, the need for offline training and the increased power consumption, operators may face increased operational expenses and network performance issues. Thus, there is a need for a novel and flexible mechanism for intelligent VNF placement [12] to fully exploit the use of MEC for uRLLC services.

In this paper, we leverage a Deep Deterministic Policy Gradient (DDPG) Reinforcement Learning (RL) algorithm to solve the NFV placement problem between MEC DCs to minimize latency for uRLLC services. Specifically, we follow the definition of ETSI Experiential Network Intelligence (ENI) and Zero-touch network Service Management (ZSM) standards [13]. Our contribution is threefold: (i) we automate live migration with DRL, (ii) we introduce a system capable of adapting to the current network conditions, and (iii) we manage edge computing and link resources in a way beneficial to both the vendors and the end-users.

The remainder of this paper is organized as follows. Section II provides a discussion of the related work. Section III presents the network architecture of our implementation. Section IV introduces the theoretical background of DDPG and defines in detail the solution that we have developed. Section V showcases the experimental setup and the results produced. Finally, Section VI provides a conclusion of this work and our future intentions.

II. Related Work

Although a plethora of research works have addressed the dynamic VM migration between DCs for orchestration purposes, only a handful of them studied the use of DRL algorithms as a potential solution to the problem. In the following, we explore relevant works in the literature in order to provide a wider view of the subject.

Cziva et al. [14] utilize Integer Linear Programming and Gurobi Solver to minimize the E2E latency between users and their VNFs. The results seem encouraging, but the complexity of the problem requires to take into consideration additional factors. In [15], Zhang et al. propose an online adaptive control mechanism that aims to reduce operating costs and enable resource allocation through reconfiguration. The main objective is the minimization of the latency by taking into account Service Level Agreements (SLAs) and the resource cost, avoiding any resource optimization technique. Although their work provides considerable value for the research community, there is a need for a solution that incorporates resource provisioning to foresee and adapt in advance to the forthcoming network traffic. L. Yala et al. [16] propose a Genetic Algorithm to solve the VNF placement problem between MEC servers for uRLLC services. Their objective is to minimize latency and maximize service availability by solving a multi-objective optimization problem that captures the trade-off between service access latency and availability. Their experimental results reveal that the solution is close to the optimal. Ben Jemaa et al. [17] introduce VNF placement and provisioning optimization strategies, taking into account Quality of Service (QoS) requirements. However, additional metrics besides Central Processing Unit (CPU) utilization and link capacity, should be taken thoughtfully into consideration. By exclusively employing VMs, they do not fully exploit the potential of MEC, as with container technology, more VNFs can share the same resources.

III. System Model

We consider a 5G capable network consisting of a set of DCs in a star topology in a star topology [18]. The DCs can be located either at the network edge, close to the end users, or in a remote cloud service. The topology consists of (i) a single Cloud DC denoted as C, that hosts numerous VNFs, denoted as V, (ii) multiple MEC DCs M, that offer computational power near the edge, (iii) network switches W, and (iv) User Equipment (UE) devices denoted as U respectively. The links that interconnect all entities in the network are denoted as L and defined with physical limitations $l = \{delay, bandwidth\}$.

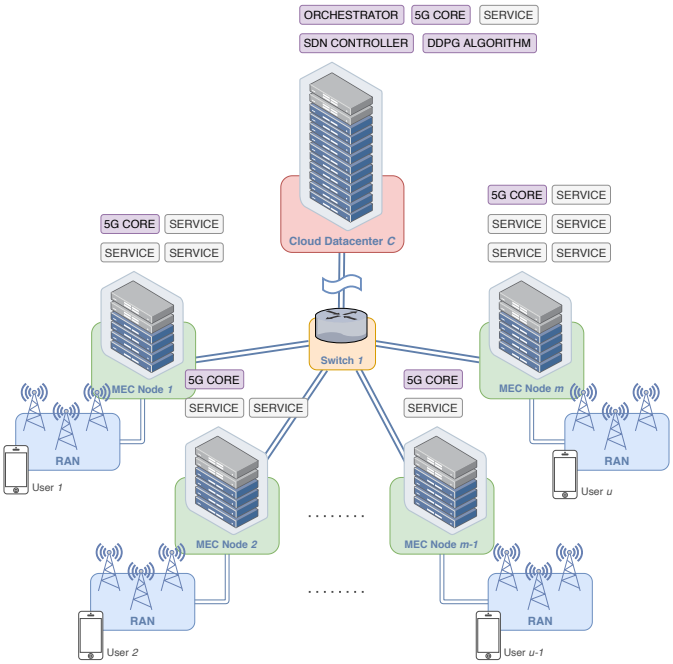

Figure 1. 5G Edge-cloud topology with a Cloud DC and multiple MEC nodes.

The network offers a uRLLC service that is composed of multiple VNFs, that are made of multiple VMs in the DCs. We assume that all MEC DCs have the same capacity $c = \{cpu, ram, storage\}$, which limit the number of VNFs that the node can host. All the entities are connected to an overlaying controller node that orchestrates the entire SDN and hosts the proposed algorithm. The topology is described as an undirected graph $G = (C, L, W, M, U)$. Both C and M DCs can host multiple VNFs at any given time.

We define as VNF profile a set of variables that declare the total computational resources, the SLA, and the average service round-trip latency needed by the VNFs of the service. It is denoted with $P = \{cpu, memory, storage, latency, sla\}$.

The objective is to calculate an appropriate placement of the VNFs on the available MEC DCs, at any given time. The placement should minimize the average E2E latency between the users and the uRLLC service VNFs while considering the distribution of available computational resources at the network edge.

The overall procedure repeats indefinitely and will be analyzed extensively in the following sections.

IV. Deep Reinforcement Learning Approach

In this section, we will make a brief introduction to the background of DRL, DDPG and the relation between them. Later on, we will focus on Actor-Critic architecture and how it applies to our proposed architecture.

A. Markov Decision Process

First, we formalize the VNF placement problem as a Markov Decision Process (MDP). MDP is a decision-making process that allow us to mathematically represent an environment through *Actions*, *States* and *Rewards*.

In this problem, as *State* we define: (i) the computational resource load of all MEC DCs M, (ii) the load of all links L, and (iii) the VNF profile P. As *Action* we define an integer that indicates the MEC id where the VNF should be placed, and as *Reward* the feedback value that precisely defines the change that the given *Action* produced to the environment of the algorithm, which is the service latency.

To reduce the large discrete action space, we devise a serialization method. This notably decreases the time complexity of the algorithm and the total training time, since it is faster to propagate information through all NNs with less neurons and hidden layers and thus increase training efficiency. Given that, the action is a single integer that corresponds to the ID of the cloud DC C or the MEC node M.

B. DRL & DDPG Background

RL is a specific type of ML method, concerned with how a software agent acts and receives a *Reward* by its environment to evaluate its *Actions*, can maximize this *Reward*. DRL is a subset of RL methods that exploit the use of multi-layer NNs instead of the design of a state-action table. NNs are composed of interconnected memory units, termed neurons. A set of neurons that perform a similar task is called a layer. Each neuron is followed by an Activation Function that decides, whether it should be activated or not. DRL algorithms, instead of employing labelled input and output data tuples like supervised learning algorithms, they focus on finding the balance between exploration of all possible actions and their outcome, and exploitation of current actions to maximize the cumulative *Reward*.

DDPG is a DRL algorithm that belongs to the family of Policy-Gradient algorithms. It is an off-policy, model-free algorithm meaning that it learns directly from raw observed data extracted from its environment with no *a priori* knowledge of the domain dynamics. This advantage makes them a versatile option for automation across a wide variety of environments compared to static planning algorithms.

C. Actor-Critic Architecture

There are two main types of RL algorithms: (i) Value-Based and (ii) Policy-Based. Value-Based algorithms map the input to real numbers, where their value represents the long-term reward achieved, given a state and an action. On the other hand, Policy-Based algorithms instead of calculating a value function of expected rewards, they learn directly the policy function that maps state to action. The DDPG algorithm is the amalgamation of both methods aiming to tackle the drawbacks of each one simultaneously. It is composed of two entirely separate NNs, as depicted in Fig. 2, that share the same input layer. (i) The *Actor* and (ii) the *Critic* NNs.

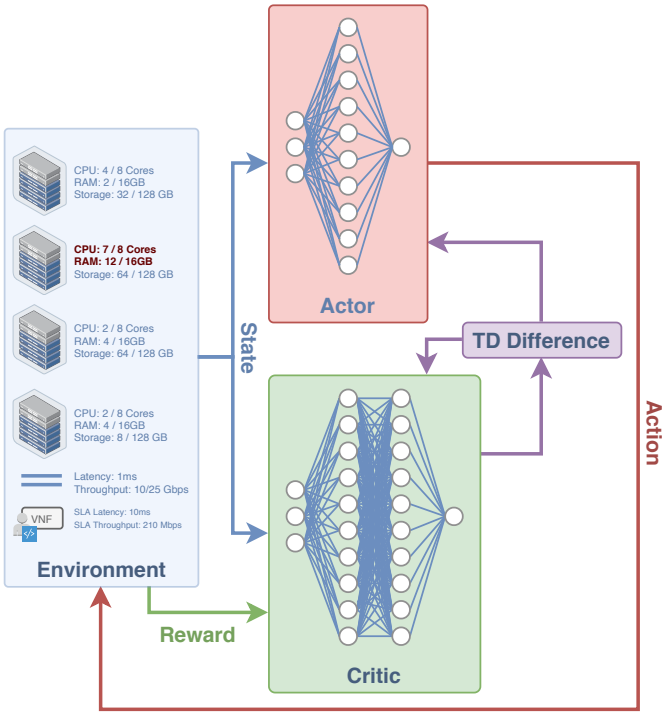

Figure 2. Actor-Critic Architecture.

1) Actor: The Actor NN directly maps the *States* to the VNF placement decisions, creating a *Policy* between the input and the an output. We use an *One-vs-Rest* in the output of Actor NN that selects the decision with the maximum probability to return higher *Reward*, to point at the MEC id that the VNF should be migrated to.

2) Critic: The Critic NN, on the other hand, is responsible for the approximation of the *Q-value* function of the Actor NN to determine the inclination of the Actor's policy base model, similar to the *Deep Q-network* algorithm [19]. The *Q-value* is essentially an approximation of the maximum future *Reward*. The Critic NN is similar to the Actor NN, except for the final

layer. The final layer is a fully connected layer that maps the states and actions with the *Q-value* of the Actor NN.

The *Reward* is an injection of positive or negative values in the weights of both Actor and Critic NNs, to point them towards making decisions that minimize the service latency.

D. TD-Error, Loss & Mini-batch

The NNs of both the Actor and the Critic generate a Temporal-Difference (TD) error signal in each time step. The TD error is computed by:

$$y_t = r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'}), \tag{1}$$

where r_t expresses the immediate reward and $t \epsilon \mathbb{N}$. $\theta^{\mu'}$ and $\theta^{Q'}$ are the outputs of the Actor and Critic NNs accordingly. The weights of Critic NN $\theta^{Q'}$ are updated with the gradients obtained from the loss function:

$$L = \frac{1}{N}\sum_t (y_t - Q(s_t, a_t|\theta^Q)^2), \tag{2}$$

where N is the size of a mini-batch, sampled from the Replay Buffer (RB), which is a list of all the actions and rewards gained from the environment so far by the actions taken. The RB allows the software agent to learn by sampling experiences from the environment. Mini-batch is a smaller list holding experiences from the RB for processing. Random mini-batches from the RB are used to improve stability [20]. The $Q(s_t, a_t|\theta^Q)$ is the loss of the Critic NN for every sample with index t.

E. Ornstein-Uhlenbeck Process

We also employ the Ornstein-Uhlenbeck process to generate random decisions that are temporally correlated. This stochastic process is called *Exploration* and allows the algorithm to explore the network behavior in the current network traffic. According to DeepMind [19], forcing *Exploration* during the first stages of operation can improve convergence speed and boost the frequency of higher rewards in RL algorithms. It is described by:

$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t, \tag{3}$$

where θ is the speed the variable reverts towards to the mean value, μ represents the equilibrium value and σ represents the volatility of the process.

The Ornstein-Uhlenbeck process offers a way to prevent the algorithm from converging at the local minimum states of the problem. This also reduces the training time required by the algorithm to converge.

F. Algorithm Flow & Procedures

As shown in Algorithm 1, the training procedure of our algorithm begins by initializing all main variables.

Algorithm 1 DDPG Training Procedure

```
Build and Initialization neural network
while episodes < episodes number do
    Initialize network environment
    Reset reward
    while steps < steps number do
        Generate state
        Compute placement action
        Perform placement and calculate reward
        Store transition
        if memory >= memory capacity then
            Decrease exploration by exploration rate
            Learn the transition
        end if
    end while
end while
```

The first loop iterates through the *Episodes* and the enclosed loop through the *Steps*. Every *Step* begins by requesting a new *State* from the controller. Then, we use *State* as an input to the Actor and Critic NNs. Based on the current conditions described in the *State*, the *Actor* NN generates the number of the corresponding MEC or Cloud to migrate the VNF, while the Critic NN estimates output that will return the maximum *Reward*. Finally, the algorithm instructs the orchestrator about the migration decision. The migration procedure begins by copying and initiating the VNFs in their new hosts while shutting them down in the old ones. After the migration procedure is completed, the algorithm requests another *State* to calculate the impact of the action in the network. The impact shapes the *Reward* value and is calculated by comparing the *State* before and after the migration.

In addition, the algorithm has to comply with a high-level set of instructions and rules set by the operator. This is called the *Operator Rules* Set and describes with high-level language a target policy, basic instructions that the algorithm uses to evaluate its actions. This approach offers a fail-safe way to adjust the algorithm to gradually meet the requirements of the network operator and prevent potential bottlenecks, packet loss or performance drops [6].

V. Experimental Evaluation

In this section, we conduct several simulation experiments to study the performance of the proposed algorithm in a realistic network topology.

A. Actor-Critic Configuration

The *Actor* NN has one hidden layer with 30 neurons and *ReLu* activation functions, as this design returned the highest reward during testing. The output layer uses *tanh* activation function that outputs the final MEC id where the VNF should be migrated. *ReLu* activation functions form a ramp function and offer a threshold that needs to be passed for the neuron to update, whereas *tanh* forms its naming function, offering a smoother value update.

The *Critic* NN is built with 30 nodes in its first hidden layer and 30 more in its second hidden layer. It also has a *ReLU* activation function that estimates the Q-value.

B. Hyper-parameter Configuration

In ML terminology, hyper-parameters are the adjustable values that are defined before the learning process and are directly linked to the performance of the algorithm. The hyper-parameters selected for this simulation study are shown in Table I. In our case, manual hyper-parameter tuning through trial-and-error was the key to find the optimal balance.

Table I
HYPER-PARAMETERS TABLE

Hyper-parameter	Value
Episodes	2000
Steps	200
Actor Network Learning Rate	0.001
Critic Network Learning Rate	0.002
Discount Factor	0.9
Soft Replacement Tau	0.01
Memory Replay Capacity Transitions	10000
Exploration Descent Rate	0.9995

C. Simulation Setup

In our experimental topology, we employed Containernet [21], an advanced branch of Mininet [22] network emulator. It enables us to divide the physical resources into individual Containers with Docker and build an SDN layer on top. We utilised the VideoLAN media player for real-time video streaming between the Containers and Wireshark to inspect the network traffic. Also, a Network Level POX controller with in-house built migration functionality is used to interact with the underlying SDN OpenFlow switches. The DDPG algorithm is implemented with a Python-based framework called TensorFlow [23].

D. Topology

In our performance evaluation, we consider a network topology that consists of a Cloud DC and 5 MEC DCs. The Cloud DC is considered to posses abundant computational resources. It also acts as a host for both the proposed DDPG algorithm orchestrator and the SDN controller. All MEC DCs have a finite amount of resources, 8 CPU cores, and 8 GB of RAM each.

In the following, we assume that VNFs $V \epsilon [25, 250]$ and each one of them serves one user. Each VNF is broadcasting a 1280×720 pixel, highly compressed H.264 video stream with Real-Time Transfer Protocol to its assigned user. Traffic distribution is designed considering that 90% of users generate less than 5% of total MEC node resources combined, whereas the remaining 10% produces more than 25%. SLA latency requirements were assigned randomly following a normal distribution, with values ranging between 5 and 30 milliseconds, to simulate the nature of different services.

E. Baseline & Cloud Approaches

As a *Baseline*, we consider an algorithm that rejects any VNF placement request to a MEC, if the given node has reached 90% of its total utilization capacity, at any given metric. The rejected VNFs are hosted at the cloud DC instead of the network edge nodes, having their SLA requirements ignored.

As a *Cloud* approach, we consider the use of one remote Cloud DC with a fixed delay value. We consider this static value as the total delay caused by the wide-area transport network that includes multiple physical switches and links. Every possible hop and queue introduces an additional delay to the propagation of the information from the edge of the network to the cloud DC.

F. Performance Evaluation

First, we carried experiments to study the average round-trip latency from the users to their VNFs, where a different number of users considered.

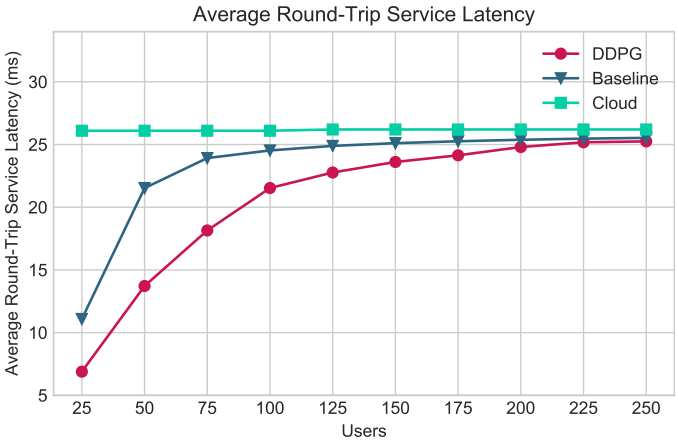

Figure 3. Comparison of the average latency from end-users to their VNF between Cloud, Baseline, and DDPG Assisted systems.

Fig. 3 outlines the average round-trip service latency that users perceive. As it is expected, increasing the number of the users in the network also increases the average service latency. This phenomenon is observed because a greater number of services have to compete for the same, insufficient MEC resources.

We can observe that our proposed algorithm, greatly outperforms both baseline and cloud options by offering considerably lower average round-trip latency, specifically at a lower number of users. All options gradually increase until they reach a threshold that indicates the saturation of MEC resources.

As presented in Fig. 4, with optimal placement in the same topology, both the average SLA violations and the VNF rejections retain lower levels than the baseline option at any traffic scale. This is an indication that our proposed algorithm can learn how to prioritize the placement of VNFs according to their latency requirements in the MEC servers and orchestrate them according to their VNF profiles.

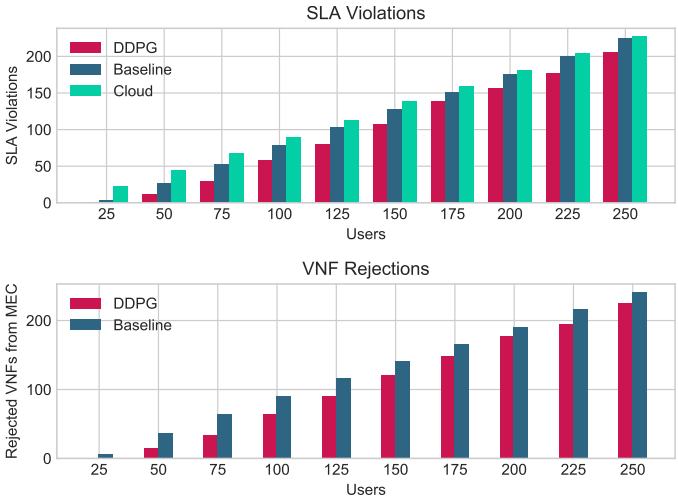

Figure 4. The top plot compares the total SLA violations between Cloud, Baseline and DDPG algorithms. The bottom plot compares the total rejected VNFs from the MEC node between Baseline and DDPG algorithms.

Compared with the rest options, our proposed solution achieves the least amount of SLA violations and reduces the number of VNF rejections by 23.4% and 31.2% respectively, at any traffic scale.

Figure 5. The average total rewards with different number of users.

Finally, to showcase the training efficiency, we train our algorithm in different traffic scales with a different number of users. Fig. 5 plots the reward during two training sessions for 100 and 200 users in the same topology. Our algorithm can successfully converge after 42 Episodes of training in the case of 100 users and 78 in the case of 200 users.

VI. Conclusion & Future Work

In this paper, we proved the viability of DDPG RL for automated spatial resource allocation by migrating VNFs between a cloud DC and several MEC nodes. Our work is flexible and dynamic, discovering the best trade-offs between SLA requirements, latency and network resources, which is beneficial for both operators and users. The presented results show clear benefits over alternative solutions.

As future work, we intend to build a proactive state prediction algorithm based on Long-Short Term Memory (LSTM) Recurrent Neural Networks (RNNs), that will provide better insights into the usage trend of each entity in the network.

Acknowledgment

This work was supported by the research projects AGAUR (2017-SGR-891), SPOT5G (TEC2017-87456-P) and MonB5G (871780).

References

[1] F. Z. Yousaf and T. Taleb, "Fine-grained resource-aware virtual network function management for 5g carrier cloud," *IEEE Network*, vol. 30, no. 2, pp. 110–115, March 2016.

[2] M. Nelson *et al.*, "Fast transparent migration for virtual machines." 01 2005, pp. 391–394.

[3] B. Li *et al.*, "Deep-learning-assisted network orchestration for on-demand and cost-effective vnf service chaining in inter-dc elastic optical networks," *Journal of Optical Communications and Networking*, vol. 10, 05 2018.

[4] K. Ye *et al.*, "Live migration of multiple virtual machines with resource reservation in cloud computing environments," in *2011 IEEE 4th International Conference on Cloud Computing*, July 2011, pp. 267–274.

[5] Y. Goto, "Standardization of automation technology for network slice management by etsi zero touch network and service management industry specification group (zsm isg)," *NTT Technical Review*, vol. 16, pp. 39–43, 09 2018.

[6] A. Mestres *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, 06 2016.

[7] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, 10 2017.

[8] Vinyals *et al.*, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II," https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[9] S. Wang *et al.*, "Deep reinforcement learning for autonomous driving," *CoRR*, vol. abs/1811.11329, 2018. [Online]. Available: http://arxiv.org/abs/1811.11329

[10] T. S. Buda *et al.*, "Can machine learning aid in delivering new use cases and scenarios in 5g?" in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, April 2016, pp. 1279–1284.

[11] D. Clark, C. Partridge, J. Ramming, and J. Wroclawski, "A knowledge plane for the internet," 10 2003, pp. 3–10.

[12] M. Gutierrez-Estevez *et al.*, "5g-monarch use case for etsi eni: Elastic resource management and orchestration," 10 2018, pp. 1–5.

[13] Y. Wang *et al.*, "Network management and orchestration using artificial intelligence: Overview of etsi eni," *IEEE Communications Standards Magazine*, vol. 2, no. 4, pp. 58–65, December 2018.

[14] R. Cziva and D. P. Pezaros, "On the latency benefits of edge nfv," *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 105–106, 2017.

[15] Q. Zhang *et al.*, "Dynamic service placement in geographically distributed clouds," in *2012 IEEE 32nd International Conference on Distributed Computing Systems*, June 2012, pp. 526–535.

[16] L. Yala, P. A. Frangoudis, and A. Ksentini, "Latency and availability driven vnf placement in a mec-nfv environment," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–7.

[17] F. Ben Jemaa *et al.*, "Qos-aware vnf placement optimization in edge-central carrier cloud architecture," in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–7.

[18] 5G PPP Architecture Working Group. (2020) View on 5g architecture. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2020/02/5G-PPP-5G-Architecture-White-Paper_final.pdf

[19] Silver *et al.*, "Deterministic policy gradient algorithms," *31st International Conference on Machine Learning, ICML 2014*, vol. 1, 06 2014.

[20] T. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, 09 2015.

[21] M. Peuster *et al.*, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 148–153.

[22] B. Lantz *et al.*, "A network in a laptop: Rapid prototyping for software-defined networks," 01 2010, p. 19.

[23] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/