Resource Provisioning in Fog Computing through Deep Reinforcement Learning

José Santos*, Tim Wauters*, Bruno Volckaert* and Filip De Turck*
* Ghent University - imec, IDLab, Department of Information Technology
Technologiepark-Zwijnaarde 126, 9052 Gent, Belgium
Email: josepedro.pereiradossantos@UGent.be

***Abstract*— The massive growth of connected devices has made traditional cloud systems inadequate to sustain the scalability, mobility, and heterogeneous nature of the Internet of Things (IoT). Distributed clouds have become a potential business opportunity for many service providers enabling the deployment of services on computational resources from the cloud up to the edge. However, challenges persist in fog-cloud infrastructures. One of them is known as Service Function Chaining (SFC), where providers benefit from network softwarization to create virtual chains of connected micro-services. Research has tackled SFC Allocation (SFCA) through theoretical modeling and heuristic algorithms, which often cannot cope with the dynamic behavior of the network. Recent works have addressed these challenges through Machine Learning (ML), which can be capable of dynamically reconfiguring cloud-native service requirements over the continuum of virtual resources in next-generation networks. Thus, in this paper, a Deep Reinforcement Learning (DRL) approach is proposed for SFCA in Fog Computing focused on energy efficiency. Our agent learns about the best resource allocation decisions, focused on reducing costs from a previously presented Mixed-integer linear programming (MILP) formulation. Results show that our agent achieves comparable performance to state-of-the-art MILP formulations during dynamic use cases, obtaining 95% of request acceptance.**

***Index Terms*—Resource Provisioning, Service Function Chaining, Fog Computing, IoT, Reinforcement Learning**

I. Introduction

The recent deployment of 5G infrastructures [1] founded on the principles of Software-Defined Networking (SDN) [2] and Network Function Virtualization (NFV) [3] has been driving the digital transformation of Internet of Things (IoT) services in multiple application domains, such as Industry 4.0, connected vehicles and smart cities. The increased flexibility and programmability has enabled the deployment of network services on computational resources from the cloud up to the edge to help bring low-latency service delivery to reality [4]. Fog Computing (FC) [5] has been introduced as an alternative to centralized cloud systems, bringing computing power, storage, and memory capacity closer to end devices by allocating services in multiple areas in the network known as Fog Nodes (FNs) or fog locations [6]. Differences distinguish cloud and FC, but both can coexist to create a continuum of virtual resources to tackle the challenges introduced by emerging use cases (e.g. high Energy Efficiency (EE), low latency). For example, FNs can be used for data filtering and pre-processing operations, while Cloud Nodes (CNs) can be reserved for heavy analytical tasks, such as Machine Learning (ML) services [7]. Next-generation networks aim to provide ultra-broadband and ultra-low latency connectivity to end-users. However, challenges persist to fully achieve a decentralized cloud-native mobile network capable of overcoming today's bottlenecks and limitations [4]. One remaining challenge is Service Function Chaining (SFC) [8], [9], where providers benefit from network softwarization to create virtual chains of connected Micro-Services (MSs). Previously, in [10], we have shown that SFC is a recent research topic in FC. Several studies have addressed SFC Allocation (SFCA) through theoretical modeling and heuristic-based algorithms, which often cannot cope with the dynamic behavior of the network and leads to poor resource usage and scalability issues. In contrast, recent research has proposed solutions through ML, which seem capable of dynamically reconfiguring cloud-native service requirements over the continuum of virtual resources in next-generation networks [11]. Thus, in this paper, a subset of ML called Reinforcement Learning (RL) [12] has been explored for SFCA in FC focused on EE. The massive number of connected devices [13] will generate large volumes of data that, if processed centrally by traditional clouds, would lead to increased power consumption and latency. Efficient collaboration between edge, fog, and cloud are crucial towards a more efficient and greener cloud-native infrastructure [14]. EE will become even more important in next-generation networks due to the increased use of data analytics and ML services [15]. Efficient resource allocation will therefore become essential since services can be requested from anywhere (i.e. cloud, fog, and edge). Resource provisioning is a difficult online decision-making problem where appropriate actions depend on fully understanding the network environment. Based on a previous Mixed-integer linear programming (MILP) formulation [16], an RL environment has been implemented where agents learn to perform SFCA in FC directly from interacting with the environment without any knowledge or information beforehand. MILP models can provide optimal schemes, however, at a cost of execution time and mostly in static use cases. Our results prove that RL techniques perform comparably to state-of-the-art ILP-based formulations but provide more scalable solutions and the ability to deal with dynamic scenarios.

The remainder of the paper is organized as follows. In the next section, related work is discussed. Section III introduces the proposed Deep RL (DRL) approach for SFCA in FC.

ISBN 978-3-903176-32-4 © 2021 IFIP

In Section IV, the evaluation setup is described, followed by the results in Section V. Finally, conclusions are presented in Section VI.

II. Related Work

In recent years, resource provisioning gained significant attention in FC. In [17], a provisioning algorithm focused on optimizing service elasticity in FC has been proposed. Results have shown that their algorithm can efficiently allocate resources while minimizing response time and maximizing throughput. In [18], the authors have proposed an FC scheme for the support of IoT crowdsensing applications, which has then formulated as a MILP model focused on cost-efficient provisioning and task distribution. Results have confirmed that their approach can outperform traditional clouds. In [19], a particle swarm optimization algorithm has been presented for resource allocation in FC focused on smart building use cases. Results have demonstrated that their algorithm can reduce the response time, the amount of transferred data, and allocation costs. In [20], the trade-off between maximizing the reliability and minimizing the overall system cost has been studied for the resource allocation problem in FC. A highly complex ILP model has been described followed by a heuristic algorithm able to find suboptimal solutions, albeit achieving better time efficiency. In [21], two service placement strategies for FC have been introduced based on matching game algorithms. The first strategy focuses on SFC concepts by considering an ordered sequence of services, while the second procedure overlooks the chain structure to lower the computation complexity without compromising performance. Recently, in [22], both the scalability and the volatility of an FC infrastructure have been studied. The authors have proposed a scheduling algorithm to allocate resources in a large-scale deployment capable of reacting to the network demand. In contrast to [23], this paper focuses on maximizing EE and reducing costs while their approach minimizes the time consumption of safety-related applications in vehicular FC. Results have confirmed that their allocation schemes reduce time consumption compared to traditional clouds.

Although most of the cited research has dealt with provisioning issues in FC, most works have only considered theoretical modeling and simulation studies, which limit their practical implementation. These approaches can only be applied when the total network demand is known, which does not happen in practice. Service providers will only benefit from solutions that can react to sudden network changes and adapt the service allocation scheme accordingly. Previously, in [24], we have presented an early approach for SFCA in FC based on a standard RL algorithm called Q-Learning. Results have shown the potential of RL in resource allocation problems, but the approach still could not fully address dynamic use cases due to the reduced complexity of the evaluated RL environment since to ease the learning process of the Q-learning algorithm, the Observation Space (OS) has been limited. Thus, the present work builds further on our previous one since the RL environment has been extended to include further information about the SFCA problem. The OS has been increased to include information about the available resources in the infrastructure. Furthermore, a different RL algorithm has been implemented based on a DRL technique known as Deep Q-Network (DQN), where its full applicability has been shown in the evaluation section based on the reimplemented RL environment.

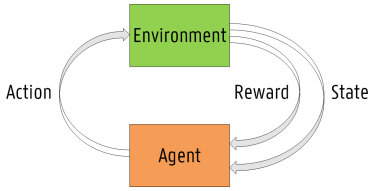


Fig. 1: The representation schema of most RL scenarios [24].

III. Towards Deep Reinforcement Learning (DRL) in Fog Computing (FC)

This section introduces the DRL approach for SFCA in FC. The RL concept is explained, followed by the presentation of the SFCA problem. Next, the OS and the Action Space (AS) are described. Lastly, the Reward Function (RF) and the agent are introduced.

A. Reinforcement Learning (RL)

Lately, RL methods have become an important area in ML research [25], [26]. Fig. 1 represents a typical scenario in RL. RL is often used to solve sequential decision-making problems where agents learn to perform actions directly from experience by interacting with an environment. The environment represents the problem to be solved. At first, the agent knows nothing about the problem at hand and learns by performing predefined actions in an environment. For each chosen action, the agent collects a reward, and a new observation of the environment is returned, describing the new state of the environment after applying the agent's selected action. Depending on the purpose and how well the agent is performing on the given assignment, the reward can be positive or negative. The agent learns to be successful by repeated interactions with the environment, by determining the inherent synergies between states, actions, and subsequent rewards. Agents attempt to maximize the total collected reward during multiple problem executions. For instance, an agent is allocating resources in a fog-cloud infrastructure, and for each applied action, it receives a reward. If the action translates into an appropriate allocation scheme, the agent receives a positive reward. Otherwise, if the performed action is inadequate (e.g. terminate a service when needed), the returned reward is negative. To maximize the collected reward, the agent needs to apply actions that translate into proper allocation schemes at all times. The goal in this scenario is to train an agent capable of selecting adequate actions to maximize performance and minimize costs. Based on our expertise, RL is well-suited for resource allocation problems. By continuously receiving

TABLE I: A sample fraction of the gym-fog OS.

Metric	Description
ur	The number of user requests at the given moment.
ar	The percentage of accepted requests at the given step.
rl_i/mp_i	The number of s_i instances allocated by the agent / MILP.
rlC/mpC	The allocation scheme cost provided by the agent / MILP.
s_in_i	The binary flag of s_i in n_i.
C_{n_i}	The available CPU of n_i.
M_{n_i}	The available memory of n_i.
B_{n_i}	The available bandwidth of n_i.

feedback from the environment, agents can adjust their action selection and achieve long-term objectives in complex situations, such as the considered SFCA problem in FC.

B. Problem Formulation - SFC allocation in Fog Computing

The SFCA problem in FC has been modeled as a MILP formulation previously presented in [16]. A set of IoT applications A composed of MSs S are allocated on nodes $n \,\varepsilon\, N$. Each application a has a given SFC identifier $id \,\varepsilon\, ID$. All MSs have a maximum number of replicas given by β. The replication factor for a particular MS $s \,\varepsilon\, S$ for the application a with the SFC identifier id is given by $\beta_{a,id,s}$. Thus, the model determines the exact number of replicas for each MS based on the considered objective. Each MS s has a CPU and a memory requirement represented by ω_s (in cpu) and γ_s (in GB) respectively. Also, each MS s has a minimum bandwidth requirement represented by δ_s (in Mbit/s). A binary placement matrix P is used to represent in which node n, the replica β_i of an MS s has been allocated. Additionally, ϖ_n corresponds to the associated node weight (e.g. FNs have a lower weight than CNs). An RL environment called gym-fog [1] has been previously developed based on the MILP model where RL agents learn how to perform SFCA in FC depending on the current status of the network infrastructure [24]. The OS and the AS of the gym-fog environment have been extended to provide further information to the agent. The purpose of the agent is to minimize the overall allocation cost, which translates into increased EE. This objective can be expressed as shown in Eq. 1.

$$\sum_{a \,\varepsilon\, A} \sum_{id \,\varepsilon\, ID} \sum_{s \,\varepsilon\, S} \sum_{\beta_i \,\varepsilon\, \beta} \sum_{n \,\varepsilon\, N} P^{a,id}_{s,\beta_i}(n) \times \varpi_n \times \omega_s \times \gamma_s \times \delta_s \quad (1)$$

C. Observation Space (OS)

The OS corresponds to the state representing the environment at a given step. For example, considering an agent performing MS allocations in a fog-cloud infrastructure, the OS should include information about the specific services previously allocated and the available amount of resources at the given moment. The OS has been designed as shown in Tab. I. For an easier understanding of our methodology, a small infrastructure scenario has been considered with two cluster nodes (n_1, n_2) where all User Requests (URs) coming to our system are made based on a single application decomposed in two individual MSs (s_1, s_2). The OS is then constituted by 18 metrics. First, two metrics (ur and ar) represent the total number of URs and the percentage of acceptance at a given step, respectively. Thus, the agent perceives the current request acceptance based on the current allocation scheme. Then, two metrics (rl_1 and rl_2) represent the total number of allocated MS instances by the agent for s_1 and s_2. The same procedure applies for the MILP model thus two more metrics have been added (mp_1 and mp_2). Furthermore, two metrics represent the overall cost of both the agent and the MILP model (rlC and mpC). The agent can then compare its performance to the MILP formulation since it recognizes the optimal cost at each step. Additionally, four binary flag metrics (s_in_i) have been included to express the exact allocation of each MS instance in the infrastructure. If any of these metrics are equal to 1, at least one MS instance has been allocated on that node. These metrics help the agent in finding adequate actions in terms of MS reallocation since the exact allocation of MS instances is known. Therefore, the agent only terminates MS instances when the correspondent flag is equal to 1. Finally, six metrics (C_{n_1}, ..., B_{n_2}) have been added regarding the available resources in the infrastructure to guarantee that the agent can learn adequate actions based on capacity constraints included in the MILP model. Otherwise, the agent cannot learn when it can deploy or terminate an MS instance accordingly. By adding this deployment information, the agent acknowledges when a given action would compromise a node since it knows if the selected node has enough capacity to support the given MS. As noted, the OS increases linearly with the number of MSs considered in the service chain and the number of nodes available in the FC infrastructure.

TABLE II: A sample fraction of the gym-fog AS.

Action Label	Description
$DoNothing$	The RL agent does nothing.
$Add - s_in_i$	Allocate an instance s_i on node n_i.
$Stop - s_in_i$	Terminate the s_i instance deployed on node n_i.

D. Action Space (AS)

The AS corresponds to all actions that the agent can perform in the environment. The AS of the gym-fog environment has been designed as shown in Tab. II assuming the small infrastructure previously presented. The AS is composed of 9 discrete actions. The AS also increases linearly with the number of MSs and the number of nodes available in the infrastructure. The first action is called $DoNothing$ as the agent performs no allocation or termination actions. The agent should only select this action when the current allocation scheme meets the current network demand. The second set of actions corresponds to the allocation of MS instances ($Add - s_in_i$). The agent selects the MS to be allocated and on which node the instance should be deployed. For example, an ML service should be allocated in a CN where more computing resources are available, despite their higher weight

[1] https://github.com/jpedro1992/gym-fog

compared to edge and FNs. Agents can select the same action several times to guarantee that the deployed number of MS instances meet the current network demand. Finally, the last set of actions corresponds to the termination of MS instances ($Stop - s_i n_i$). As in deploying actions, the agent chooses the MS to be terminated based on a particular cluster node thus the MS instance allocated on the selected node is terminated.

E. Reward Function (RF)

Algorithm 1 Reward Function of the gym-fog environment

Input: Observation state after action step in
Output: Reward out

// Return the reward for the given state
getReward(obs):
  $r = 0$
  $ur, ar, rl1, rl2, mp1, mp2, rlC, mpC = obs.get()$
  // Reward based on Keywords for MILP constraints
  if $MaxInstancesOnNode == True$ **then return** 0
  if $StopWithoutDeploy == True$ **then return** 0
  if $MaxInstancesReached == True$ **then return** 0
  if $MaxNodeCapacity == True$ **then return** 0

  $r = r + getM(rl1, mp1)$ *// MS1 calculation*
  $r = r + getM(rl2, mp2)$ *// MS2 calculation*
  if $ar == 0$ **then return** r
  $r = r + ar$ *// Request Calculation*
  $r = r + getC(rlC, mpC)$ *// Cost Calculation*
  if $rl1 == mp1$ **and** $rl2 == mp2$ **then**
    if $rlC == mpC$ **then**
      $r = r + 100$
  return r

The purpose of the RF is to teach the agent how to maximize the accumulated reward by selecting appropriate actions depending on the observation provided by the environment. For each action, a particular reward is retrieved. Thus, the agent learns if its chosen action has been appropriate based on the received reward. The design of a proper RF through the manual tuning of ML parameters is needed to ensure the agent learns what it is supposed to. The implemented RF is shown in Alg. 1. The objective is to lead the agent to allocate MSs in a fog-cloud infrastructure according to the MILP formulation. The MILP model provisions MSs by minimizing the overall cost, as shown previously. Therefore, the closer the agent is to achieve the MILP's solution, the higher the reward it receives. First, rewards are returned based on constraints added to the MILP model. For example, capacity constraints have been added to the environment. If an allocation action is selected, and the chosen node does not have enough resources to allocate the MS instance, the deployment is revoked, and a null reward is returned. The agent learns what forced this constraint due to the extension of the OS, meaning that the particular action would not be selected again if the same state occurs since a null reward would be expected by the agent. Then, MS rewards have been calculated as shown in (2).

$$getM(rl, mp) = \begin{cases} 5.0 & \text{if } rl == mp \\ 0 & \text{Otherwise} \end{cases} \tag{2}$$

If the number of allocated MS instances by the agent is equal to the ones allocated by the MILP model, 5 is returned. The agent is incentivized to provision the exact number of MS replicas as the MILP model. If the agent allocates a higher number of replicas (i.e. over-provisioning) or if the agent provisions fewer instances (i.e. under-provisioning), 0 is returned. After the MS reward calculation, the percentage of accepted requests is verified. If the acceptance is 0%, the current accumulated reward is returned. The agent learns that zero acceptance leads to low rewards. The agent receives a positive reward when it deploys the same number of allocated replicas of a given MS as the MILP model. Otherwise, a zero reward is given. If the percentage of acceptance is higher than 0%, the current percentage is added as a reward, meaning that if all requests are accepted, 100 is added to the accumulated reward. Thus, the agent learns that higher acceptance rates translate into higher rewards. A linear increase has been applied since it led the agent towards better allocation decisions rather than an exponential function in the conducted experiments. Then, a cost RF has been applied as shown in (3). If the agent's cost is equal or up to 10% higher than the MILP one, 10 is returned since the agent is performing similar to the MILP model. Then, depending on how higher the agent cost is compared to the MILP cost, a decreasing negative reward is returned, meaning that the agent is being taught that the closer it stays to the MILP cost, the higher reward it receives. Lastly, a bonus reward is given to the agent if the agent's allocation scheme is the exact one provided by the MILP model. A bonus reward of 100 has been given if both the agent's cost matches the MILP cost and the number of MS instances are equal, the ultimate purpose of our agent: learning how to allocate MSs in a fog-cloud infrastructure as a MILP formulation.

$$getC(rl, mp) = \begin{cases} 10.0 & \text{if } mp \leq rl \leq 1.10 \times mp \\ -10 & \text{if } 1.10 \times mp < rl \leq 1.25 \times mp \\ -25 & \text{if } 1.25 \times mp < rl \leq 1.75 \times mp \\ -50 & \text{if } 1.75 \times mp < rl \leq 2.0 \times mp \\ -75 & \text{if } 2.0 \times mp < rl \leq 3.0 \times mp \\ -90 & \text{if } 3.0 \times mp < rl \leq 4.0 \times mp \\ -100 & \text{Otherwise} \end{cases} \tag{3}$$

F. Agent - Deep Q-Network (DQN) methods

In [24], a Q-Learning agent based on a former version of the gym-fog environment has been evaluated. For this paper, a more complex RL algorithm based on DQNs has been assessed. The DQN algorithm has been introduced by Google DeepMind [27], [28], which combines Q-Learning theory with deep neural networks to teach agents RL tasks as a supervised learning operation. The concept behind DQN is that Q-learning is not feasible for high dimensional problems since

TABLE III: The gym-fog environment configuration.

Name	Description
Applications / Micro-services	1 / 3
Locations / nodes	9 / 45
SFC structure	$a_1 : s_1 \rightarrow s_2 \rightarrow s_3$
Max. replication factor	5
Action / Observation spaces	241 actions (l_1) / 280 states (l_5)
Episode duration	100 steps
DQN structure / layers	5 layers [280, 512, 256, 64, 241]

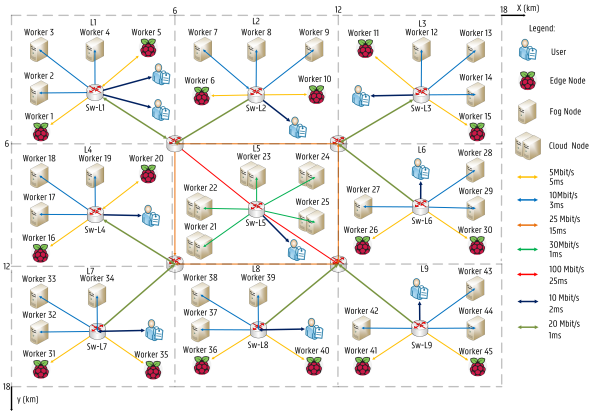


Fig. 2: The fog-cloud infrastructure for the evaluation.

TABLE IV: The hardware configuration of each node.

Node Type	Weight	CPU (cpu)	RAM (Mi)	Band. (Mbit/s)
Edge	1.0	1.0	2.0	5.0
Fog	2.0	2.0	4.0	10.0
Cloud	3.0	8.0	16.0	30.0

thousands of state-action pairs are needed to compute the Q-value function, which makes the learning process impractical. This has been one of the main reasons why we have reduced the state-space complexity of the gym-fog environment to assess the performance of a Q-learning agent. However, the agent struggled to learn all the dynamics of our environment due to low state complexity. Thus, in this paper, a deep neural network is used to estimate the Q-value function instead of computing the Q-value for each state-action pair of our gym-fog environment. Several improvements have been made to DQN over the last few years. In the developed DQN agent, three extensions have been applied: double [29], dueling [30] and Prioritized Experience Replay (PER) [31]. A DQN method applying all these concepts is named D3QN-PER, the implemented agent for the SFCA problem.

IV. Evaluation Setup

The gym-fog environment has been developed based on the OpenAi gym [32], an open-source project for RL research written in Python. The MILP model initially developed in Java has been rewritten in Python to ease the interaction between the MILP model and the OpenAi gym. The environment configuration is shown in Tab. III based on the FC infrastructure illustrated in Fig. 2. The infrastructure considers

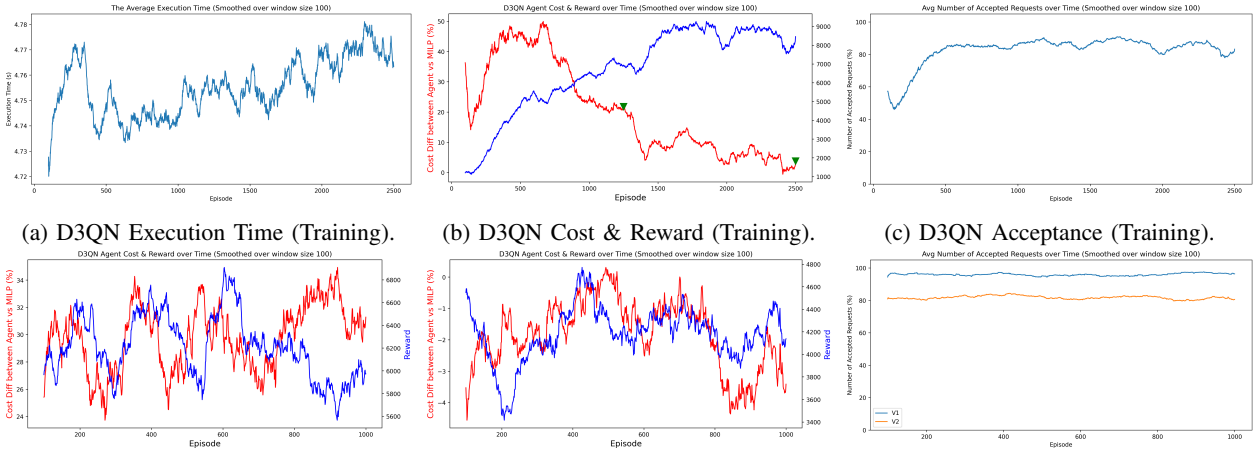
TABLE V: The MILP model execution time.

User Requests	1	5	10	20	30	40	45	47	50
Exec. Time (s)	0.2	0.6	1.1	15.1	26.5	33.8	242.7	3600.0	

a total area of 324 km^2, in which the MS allocation is possible in nine locations L, each managing a set of five nodes. Tab. IV shows the hardware configurations of each node based on the correspondent node type. Each node has a given computing capacity and a certain weight, which is the necessary data to calculate the overall system cost based on the MILP formulation. As expected, CNs have more computing power than edge or FNs, but their weight is also higher since their usage translates into more power consumption. In the evaluation, a dynamic use case has been assessed, in which the network demand is changing during the episode where users join and leave randomly. The number of URs may decrease or increase, and the D3QN agent must adapt its allocation scheme according to the user demand. The number of URs has been changed every five steps between 1 and 50 based on specific probabilities (increase: 50%, equal: 35%, decrease: 15%). The total increase or decrease is random, meaning that at each step, users may leave or stay connected while others may join, translating into several patterns occurring in different episodes. The D3QN agent and the MILP formulation have been executed on a 6-core Intel i7-9850H CPU @ 2.6 GHz processor with 16 GB of memory.

V. Results

The execution time of the D3QN agent is shown in Fig. 3a. The D3QN agent solves a single episode in on average 4 and 5 seconds, which is significantly faster compared to MILP-based calculations since the MILP model needs to calculate the optimal scheme on each episode step. In Tab. V, the MILP execution time is presented. For example, for more than 10 URs, the MILP model takes at least 1 second to obtain the optimal scheme. For 40 URs or higher, a time limitation of 3600 seconds and a gap tolerance of 15% have been introduced in the model. Otherwise, the model could have taken hours to find the optimal solution, even though the provided solution with these limitations has the same overall value as the optimal one. These calculations represent a single episode step, which proves the drawback of MILP formulations since every change on the network would require a new calculation, making these solutions impractical. Fig. 3b illustrates both the accumulated reward and the average cost difference between the D3QN agent and the MILP model during training. The D3QN agent can reduce the overall cost reaching solutions 2% worse than the MILP model. The agent collects accumulated rewards of 8500 in a single episode, meaning that the agent is receiving on average a reward of 85 per step. Based on the designed RF, the agent cannot allocate all the required MS instances in the infrastructure based on the current demand, which affects the percentage of accepted requests as shown in Fig. 3c. Due to the dynamic behavior, the agent needs to constantly adjust the



(a) D3QN Execution Time (Training). (b) D3QN Cost & Reward (Training). (c) D3QN Acceptance (Training).

(d) V1 Cost & Reward (Testing). (e) V2 Cost & Reward (Testing). (f) V1 & V2 Acceptance (Testing).

Fig. 3: Results obtained by the D3QN agent during training and testing.

allocation scheme, translating into under-provisioning or over-provisioning schemes during some steps in a single episode. Thus, the request acceptance oscillates between 75% and 85% during training when a smoothing window of 100 is applied. Two markers (green triangles) have been added to Fig. 3b to highlight two saved configurations of the trained agent: V1 and V2. The V1 agent configuration has been saved after completing the 1250th episode, while the V2 agent configuration has been stored at the end of the training, after the 2500th episode. Agents must be trained and then tested as in supervised learning tasks. For generalization purposes, the agent has been trained to perform SFCA in FC by experiencing different demand patterns during training. Now, during testing, these patterns will also be different, meaning that the agent must be able to select adequate actions even if the current demand has never been experienced before. Both agents have been tested for 1000 episodes. Fig. 3d and Fig. 3e show the accumulated reward and the cost difference of the V1 and the V2 agent, respectively. The V1 agent reduces the overall cost reaching solutions 30% worse than the MILP model with accumulated rewards of 6200, while the V2 agent provides allocation schemes 2% cheaper with accumulated rewards of 4200. The V2 agent had more training time than the V1 agent thus the V2 agent can achieve lower costs than the V1 agent, obtaining higher EE. However, the V2 agent provides under-provisioning schemes, which do not accept all user requests, as illustrated in Fig. 3f. The V1 agent obtains 95% acceptance while the V2 agent achieves 80% on average. By further reducing costs, the allocation scheme adapts to the current demand. When the demand increases, the agent has less time to react, and for a few steps, some requests cannot be accepted since the deployed MSs are overloaded. In practice, when service providers opt for reducing costs, users may experience short service disruptions when the network demand increases since the allocation scheme cannot support all URs. In contrast, other service providers may opt for maximizing service reliability, and thus, if the demand suddenly increases, there is still enough capacity to accept the increased number of URs. This highlights the differences obtained by the two saved agents. On the one hand, the V1 agent is allocating on average 30% more resources than the MILP model, responding faster to increased demands, which translates into 95% acceptance during testing. On the other hand, the V2 agent is reducing costs allocating on average 2% fewer resources than the MILP model, which in most cases cannot react quickly to increased demands, resulting in only 80% acceptance.

VI. Conclusions

In this paper, a DRL approach for SFCA in FC has been proposed. An environment called gym-fog has been developed to bridge the gap between MILP formulations with RL algorithms. A RF has been set up to incentivize RL agents to select adequate actions for SFCA focused on reducing the overall system cost, translating into higher EE. Results have shown that RL agents can obtain comparable performance to state-of-the-art ILP formulations while providing a more scalable solution. The V2 agent training has been significantly longer than the V1 agent training, resulting in reduced costs within 30% and 0% and acceptance rates of 95% to 80% compared to the exact MILP solution. A clear trade-off between cost reduction and request acceptance has been demonstrated. Our approach has proven its full applicability to SFCA in FC. RL systems capable of reallocating services in the infrastructure by reacting to sudden network changes will be the next main topic in the resource allocation field. As future work, our RL approach will be extended to consider different objectives, such as end-to-end latency reduction in service chains, and make it less dependable on our MILP model, thus, making it more scalable.

Acknowledgment

This research was performed within the project “Intelligent DEnse And Longe range IoT networks (IDEAL-IoT)” under Grant Agreement #S004017N, from the fund for Scientific Research-Flanders (FWO-V).

REFERENCES

[1] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, "5g: A tutorial overview of standards, trials, challenges, deployment, and practice," *IEEE journal on selected areas in communications*, vol. 35, no. 6, pp. 1201–1221, 2017.

[2] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[3] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Network*, vol. 28, no. 6, pp. 18–26, 2014.

[4] (2020) White paper on 6g networking. [Online]. Available: https://www.6gchannel.com/portfolio-posts/6g-white-paper-networking/

[5] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of everything*. Springer, 2018, pp. 103–130.

[6] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Fog computing: Enabling the management and orchestration of smart city applications in 5g networks," *Entropy*, vol. 20, no. 1, p. 4, 2018.

[7] J. Santos, P. Leroux, T. Wauters, B. Volckaert, and F. De Turck, "Anomaly detection for smart city applications over 5g low power wide area networks," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–9.

[8] Y. Xie, Z. Liu, S. Wang, and Y. Wang, "Service function chaining resource allocation: A survey," *arXiv preprint arXiv:1608.00095*, 2016.

[9] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, 2016.

[10] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards delay-aware container-based service function chaining in fog computing," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.

[11] S.-s. Lee and S. Lee, "Resource allocation for vehicular fog computing using reinforcement learning combined with heuristic information," *IEEE Internet of Things Journal*, 2020.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[13] G. M. D. T. Forecast, "Cisco visual networking index: global mobile data traffic forecast update, 2017–2022," *Update*, vol. 2017, p. 2022, 2019.

[14] J. Elmirghani, T. Klein, K. Hinton, L. Nonde, A. Lawey, T. El-Gorashi, M. Musa, and X. Dong, "Greentouch greenmeter core network energy-efficiency improvement measures and optimization," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 10, no. 2, pp. A250–A269, 2018.

[15] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6g networks: Use cases and technologies," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 55–61, 2020.

[16] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards end-to-end resource provisioning in fog computing over low power wide area networks," *Journal of Network and Computer Applications*, p. 102915, 2020.

[17] S. Agarwal, S. Yadav, and A. K. Yadav, "An efficient architecture and algorithm for resource provisioning in fog computing," *International Journal of Information Engineering and Electronic Business*, vol. 8, no. 1, p. 48, 2016.

[18] H. R. Arkian, A. Diyanat, and A. Pourkhalili, "Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications," *Journal of Network and Computer Applications*, vol. 82, pp. 152–165, 2017.

[19] A. Yasmeen, N. Javaid, O. U. Rehman, H. Iftikhar, M. F. Malik, and F. J. Muhammad, "Efficient resource provisioning for smart buildings utilizing fog and cloud based environment," in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE, 2018, pp. 811–816.

[20] J. Yao and N. Ansari, "Fog resource provisioning in reliability-aware iot networks," *IEEE Internet of Things Journal*, 2019.

[21] F. Chiti, R. Fantacci, F. Paganelli, and B. Picano, "Virtual functions placement with time constraints in fog computing: a matching theory perspective," *IEEE Transactions on Network and Service Management*, 2019.

[22] T. Goethals, B. Volckaert, and F. De Turck, "Adaptive fog service placement for real-time topology changes in kubernetes clusters," in *CLOSER2020, the 10th International Conference on Cloud Computing and Services Science*, 2020, pp. 161–170.

[23] X. Chen, S. Leng, K. Zhang, and K. Xiong, "A machine-learning based time constrained resource allocation scheme for vehicular fog computing," *China Communications*, vol. 16, no. 11, pp. 29–41, 2019.

[24] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Reinforcement learning for service function chain allocation in fog computing," *Book Chapter in revision, Submitted to Communications Network and Service Management In the Era of Artificial Intelligence and Machine Learning, IEEE Press*, 2020.

[25] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[26] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[29] P. A. Tsividis, T. Pouncy, J. L. Xu, J. B. Tenenbaum, and S. J. Gershman, "Human learning in atari," in *2017 AAAI Spring Symposium Series*, 2017.

[30] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1995–2003.

[31] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[32] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.