Oneshot Deep Reinforcement Learning Approach to Network Slicing for Autonomous IoT Systems

Abdel Kader Chabi Sika Boni, Hassan Hassan, Khalil Drira

Oneshot Deep Reinforcement Learning approach to network slicing for autonomous IoT systems

Abdel Kader Chabi Sika Boni[1], Hassan Hassan[1] and Khalil Drira[1]

***Abstract*—With the emergence of the Internet of Things (IoT) services, meeting multiple and diverse Quality of Service (QoS) requirements in networks has become a crucial issue. In the new 5G networks, network slicing is presented as the solution to provide a tailored QoS for different network services. This new technology offers better prospects for IoT services and applications. In fact, in modern IoT systems, the number of IoT devices increases, and these systems evolve to be autonomous IoT systems. QoS management must be done without human intervention, making conventional QoS management mechanisms unsuitable. In this paper, we introduce an oneshot Deep Reinforcement Learning (DRL) agent capable of autonomously receiving requests for slices and proposing a placement on the physical infrastructure that maximizes the total number of accepted requests while guaranteeing load balancing at the infrastructure resources level. By adopting a new paradigm located at the crossroads between the single DRL agent and the multi-agent DRL, our agent manages to generate the placement decision of a slice request in one step, which makes it compatible with the European Telecommunications Standards Institute (ETSI) standard. Numerous simulations and comparisons with six other algorithms allowed us to validate its effectiveness in real-time scenarios where learning from previous placements is required to improve future slice provisioning.**

***Index Terms*—Autonomous IoT systems, QoS, Network Slicing, Deep Reinforcement Learning.**

I. Introduction

In the context of fully automated networks, i.e., zero-touch networks, and particularly in autonomous Internet of Things (IoT) systems, efficient and automatic placement of network services is considered as one of the most important technological building blocks [31]. The proliferation of IoT devices in recent years has participated in the emergence of new network services. The number of IoT devices in use is estimated to be between 75 and 100 billion by 2025 [38] [3]. As about 127 new IoT devices per second will establish a connection with current devices [38], this estimate will be revised upward. These IoT devices are used in many IoT use cases i.e., smart cities, smart transportation networks, smart healthcare facilities, smart agriculture systems, etc. These use cases involve services with often conflicting Quality of Service (QoS) requirements. The coexistence of IoT services on the same physical infrastructure makes it difficult to implement tailored QoS solutions. Hopefully, NGMN (Next Generation Mobile Network) has introduced the concept of Network Slicing in fifth-generation networks (5G) [4]. It advocates the satisfaction of diverse QoS requirements by dividing the common physical infrastructure into logical networks i.e., network slices, each specialized to provide specific network capabilities and characteristics for a particular use case. This technology offers new possibilities to IoT services as one IoT service can be therefore supported by a network slice that guarantees its QoS requirements.

Network slicing is enabled by two key technologies i.e., Network Function Virtualization (NFV) and Software Defined Network (SDN) [16]. NFV enables the transformation of hardware-based network functions into software-based functions called VNFs (Virtual Network Functions). A network slice is composed of these VNFs while dynamic traffic steering between them is provided by SDN. The challenge of efficient placement and chaining of a slice request's VNFs is called the Network Slice Placement problem (NSP). It can be viewed as a special case of Virtual Network Embedding (VNE), VNF Forwarding Graph Embedding (VNF-FGE), Service Functions Chaining/Placement (SFC-P), VNF-PC (Virtual Network Function Placement and Chaining) [11]. All of these problems are NP-hard [7]. While network slicing intends to solve conflicting QoS requirements in 5G networks, the placement of network slices is a hard problem and becomes more difficult in the context of IoT systems. In fact, IoT services have the particularity of being made up of thousands of middle boxes [24]. Their slice equivalents are therefore likely to contain an important number of VNFs. This represents a strong scalability challenge compared to non-IoT slices. Moreover, since an IoT system may provide several IoT services, it becomes crucial to optimize the use of network resources in order to deploy the largest number of IoT slices. Particularly, in the context of autonomous IoT systems, these IoT slices must be created, managed, and destroyed automatically without human intervention using adaptive, reactive, reliable, and autonomous algorithms. Deep Reinforcement Learning (DRL) algorithms are among the most powerful in this category.

DRL is a category of artificial intelligence algorithms where an agent continuously interacts with an environment based on the trial and error principle. The autonomous agents created with DRL have enabled great advances in the fields of video games, self-driving cars, object detection etc. [8]. Recently, this technique has been used to solve the Network Slice Placement problem [31] [42] [41] [10] with many limitations and assumptions affecting its efficiency and making its deployment in real environments not straightforward as we highlight it in Section II.

In order to illustrate the issues inherent to NSP, let us consider

[1]LAAS-CNRS, University of Toulouse, CNRS, UPS, Toulouse, France
{akchabisik, hhassan, khalil}@laas.fr

the infrastructure shown in Figure 1. This infrastructure is composed of six NFV-capable nodes (Computing Nodes) connected to SDN-capable switches which are in turn connected by a transport network of 4 routers. The infrastructure provider receives the request for NSPR (Network Slice Placement Request) x from an IoT service provider. Depending on the intended goal, the infrastructure provider has several possible placement configurations. Each configuration corresponds to the deployment of NSPR x's VNFs (1, 2, and 3) at computing nodes from 1 to 6. The table in Figure 2 shows five configurations among the 120 possible placement configurations. Moreover, for each possible configuration, the infrastructure provider chooses the physical path associated with each virtual link (VL) of NSPR x. Thus, Figure 1 shows a possible path for $5th$ configuration in the table where virtual links VL1 and VL2 of NSPR x are associated with physical links PL15 $\rightarrow$ PL11 $\rightarrow$ PL5 $\rightarrow$ PL8 $\rightarrow$ PL10 $\rightarrow$ PL13 and PL13 $\rightarrow$ PL10 $\rightarrow$ PL7 $\rightarrow$ PL3 $\rightarrow$ PL1 respectively. The number of possible configurations depends on infrastructure's size but also on each request to satisfy. Therefore, there are 720 placement configurations if the received NSPR includes 5 VNFs ($6P5 = 720$) or 426957024 configurations when considering the infrastructure with 754 nodes from [30] ($754P3 = 426957024$). In either case, the number of possible paths for each configuration also increases substantially. Added to this are the Central Processing Unit (CPU), Random Access Memory (RAM), and storage space requirements of each VNF as well as the bandwidth requirements of the NSPR's virtual links. The infrastructure provider receives many NSPRs in one time unit.

Although DRL algorithms can offer an adaptable and reliable solution to this problem, the number of possible actions of the DRL agent must be equal to the number of possible configurations (e.g., 426957024 actions considering the request in Figure 1 and the 754-node infrastructure in [30]). But it has been shown that when the action space is too large, a DRL agent experiences great difficulties in convergence [36] [35]. Most of the existing works have therefore adopted a sequential DRL where NSPR's VNFs are processed one after the other; this is equivalent to asking, for example, on which of the 754 nodes to place the current VNF. The action space is then reduced to 754, but it now takes as many DRL iterations as there are VNFs in each request. Moreover, such an iterative VNF placement is not compatible with the European Telecommunications Standards Institute (ETSI) standards where a NSPR along with its specified VNFs placements must be addressed to the Network Function Virtualization Orchestrator (NFVO) in a single instruction [14].

In this work, we adopt the term Network Slice Placement as in [11] and use NSPRs to designate incoming slice requests. We introduce an algorithm for Network Slicing into autonomous IoT systems by proposing a deep reinforcement learning agent capable of autonomously receiving and placing NSPRs in a oneshot manner on an IoT infrastructure. Based on [20]'s classification, we position our oneshot DRL approach as a resource management technique in Network Slicing and more precisely as a Resource Allocation approach. While comparing our approach to others in the literature, we have shown through the results of numerous simulations conducted, its effectiveness and relevance in the context of autonomous IoT systems.

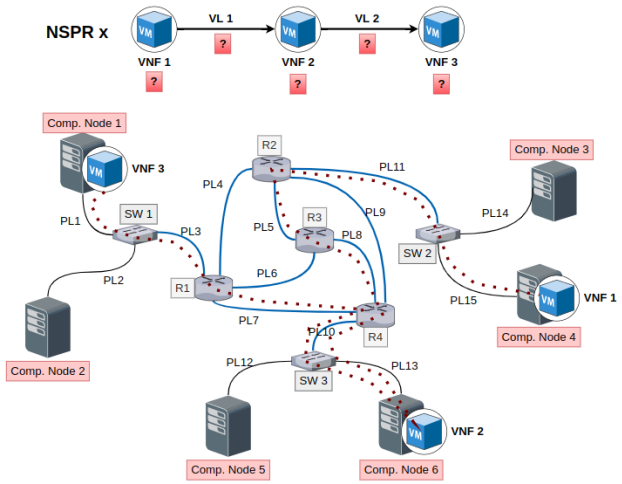


Fig. 1. Example of NSPR deployment on a 6 nodes infrastructure

The contributions of this paper are as follows:

- we propose a new reinforcement learning framework at crossroad between single agent and multi-agents paradigms
- we introduce the Extensible Reinforcement Learning Observation (ERLO) concept to support our proposed framework
- with the new framework, we propose a centralized microagents-based agent to efficiently solve NSP problem in an oneshot manner allowing compatibility with ETSI NFV MANO (MANagement and Orchestration) framework
- we introduce a new learning mechanism which efficiency grows as NSPR number of VNFs grows
- we take into account the VNF-PC DRL Loop Latency (VDLL) explained in Section II-F when designing our agent as to enable its future real deployment in real scenarios
- we design a new reward function able to stimulate the agent to generate efficient placement decisions
- we compare our solution to many existing state of the art algorithms to prove its efficiency under a such uncontrollable parameter (i.e., VDLL)

The rest of the paper is organized as follows: In Section II, we present a detailed discussion of related studies in which we show the different existing approaches to solve the Network Slice Placement problem. The formulation of the problem as well as different notations used throughout this paper are available in Section III. Global and internal functioning of our proposed oneshot DRL agent is described in Section IV. The comparison and performance analysis of this algorithm with six other algorithms is done in Section V. Within the latter, we also detail the comparison metrics and the simulation environment. Finally, a conclusion and future perspectives are discussed in Section VI while highlighting few limitations of our algorithm in Section VII.

		CNode 1	Cnode 2	CNode 3	Cnode 4	CNode 5	Cnode 6
Case 1	VNF 1	V	X	X	X	X	X
	VNF 2	V	X	X	X	X	X
	VNF 3	V	X	X	X	X	X
Case 2	VNF 1	V	X	X	X	X	X
	VNF 2	X	X	V	X	X	X
	VNF 3	X	X	X	X	V	X
Case 3	VNF 1	V	X	X	X	X	X
	VNF 2	X	V	X	X	X	X
	VNF 3	V	X	X	X	X	X
Case 4	VNF 1	V	X	X	X	X	X
	VNF 2	V	X	X	X	X	X
	VNF 3	X	V	X	X	X	X
Case 5	VNF 1	X	X	X	V	X	X
	VNF 2	X	X	X	X	X	V
	VNF 3	V	X	X	X	X	X

Fig. 2. Five possible placement configurations for NSPR x in Figure 1

II. Related work

Many existing works tried to provide solutions for VNFs placement and chaining problems as well as its variants. We can class these solutions in the following categories: Exact solution approaches, Heuristics approaches, Meta-heuristics approaches, DRL approaches and combined approaches.

A. Exact solution approaches

In literature, many previous works adopted an exact solution approach. Jose Jurandi et al. [12] formulate the NSP problem into an Integer Linear Programming (ILP) aimed at minimizing global resources consumption. Bari et al. [9] adopt a similar approach changing the objective function to minimizing VNFs deployment cost, total energy consumption, traffic forwarding cost and Service Level Objectives (SLO) cost. ILP decision variables are restricted to integers. When the mathematical formulation allows those variables to be integers as well as real numbers, it is called Mixed Integer Linear Programming (MILP). The latter is the one adopted by [34] and [19] for fulfilling Service Function Chain requests in datacenter networks and cloud environment respectively. While [34] seeks to minimize the number of Virtual Machines (VMs) set up to execute VNF instances, [19] focuses on minimizing the delay between two dependent VNFs. Dealing also with SFCs but in geo-distributed cloud, Jianing Pei et al. [28] formulate their optimization problem in Binary Integer Programming (BIP) where decision variables take two possible values (i.e., 0 or 1). They were able to minimize the number of instances required to embed SFCs with minimum embedding cost. Although exact solution approaches have merit to output optimal solutions for small instances of the problem, they take too much time when dealing with large-scale networks and are applicable only if all NSPRs are known in advance. For example, [12] provide an optimal solution for NSPRs in a network with 280 nodes after 7000 seconds ($\simeq$ 2 hours) which is impractical in real-time scenarios.

B. Heuristics approaches

Heuristics approaches differ from exact approaches in the way they trade optimality for speed, i.e., they try to find near-optimal solutions in a reasonable time. Nevertheless, they are often based on a mathematical formulation as exact approaches. Matthias and Stefan [33] introduce a LP (Linear Programming) formulation of VNE problem which they relaxed. Processing the latter, they decompose the solution they find and employ a randomized rounding heuristic to select combinations of solutions. Doing so, they avoid exploring all feasible solutions, thus reducing the solution generation for a network of 50 nodes. Similarly, [22] proposed T-SAT, a two stage heuristic solution to place SFCs in cloud datacenters while minimizing the number of activated physical machines. [29] proposes a Markov approximation technique to fulfill Service Chain Requests in NFV-enabled networks while minimizing operational and network traffic costs. Authors in [45] were able to place some VNFs chains in datacenter networks scenario using a heuristic which sorted requests in descending order according to their required resources and chose a candidate physical node randomly for each VNFs. Such a sorting was possible mainly because they considered only CPU as VNF resources. [5] employs "Power of two Choices" (P2C) to satisfy NSPRs. The principle consists of randomly choosing two physical server nodes and placing current VNF being processed on the most eligible server. They drastically reduced the execution time at the expense of optimality. Bernardetta Addis et al. [1] considered a VNE scenario in which a single VNF could have many instances set up. Then, trying to minimize the number of CPU cores used by instantiated VNFs, they formulated a MILP problem and solved it through a random values assignment to decision variables. Efficient resource allocation is a crucial challenge in NFV enabled networks [18] and all heuristics methods, due to their massive use of randomness, cannot provide such efficiency. Moreover, heuristic solutions suffer from the problem that they can get stuck in a local optimum that can be far from the real optimum [15] [18]. Other studies therefore investigated the use of meta-heuristic approaches.

C. Meta-heuristics approaches

Considering VNF-PC as a combinatorial problem, meta-heuristics sought the most optimal solution over a discrete search space [18]. Meta-heuristics go from Simulated annealing to tabu search through genetic algorithms, ant colony optimization and particle swarm optimization [15]. Jiaqiang Liu et al. [23] focuses on the Service Chaining problem by formulating it into BIP which they solve using Simulated Annealing ensuring an end-to-end delay minimization along with bandwidth consumption. Marcelo Caggiani et al. [24] proposed Fix-and-Optimize, a solution for SFCs requests. They first formulated an ILP problem and used Variational Neighborhood Search to prioritize (based on the degree of adjacency) and choose randomly subsets of Nodes Point of Presence (N-PoPs). They aim to satisfy more SFCs using fewer VNF instances. In the worst case, their solution takes 1500 seconds. [30] proposed GAVA, a genetic algorithm based solution

for jointly placing and routing VNF-FGs keeping minimum physical links resource consumption and deployment cost. Meta-heuristics methods provide much near-optimal results than heuristics as they try to mimic many real-life processes. Nevertheless, they also suffer from the sub-optimality problem which is still inconsistent in online scenarios.

D. Pure DRL approaches

Thanks to DRL advances in recent years [8], some studies investigated its use in VNE problem. DRL is an artificial intelligence technique in which an agent interacts with an environment in a trial and error manner. Some rewards are sent to the agent to adjust its actions. The agent's goal is to maximize cumulative discounted rewards. Pure DRL approaches are straightforward as they simply adapt existing DRL algorithms to VNF-PC problem without any major modification. The main challenges are: i) to set the number of actions so as to not explode the action space; ii) to provide a good reward function and; iii) to provide a well designed observation or feature matrix. The agent's convergence speed depends on all of those aspects. In [41], a VNE scenario is studied in which a DRL system is set up using Asynchronous Advantage Actor-Critic (A3C) [25] algorithm with 24 workers. Each worker interacts with its own copy of the environment. They used the Graph Convolutional Network (GCN) [21] framework to automate observation extraction while setting the number of possible actions to the number of physical nodes. They showed good performance through simulations maximizing the acceptance ratio and average Infrastructure Provider (InP) revenue. However, deploying their solution might be impossible or at least very difficult as the ability to duplicate the environment is often possible only in a simulation environment. Under the VNE scenario, [10] proposed DeepViNE for grid-like requests fulfillment on grid-like physical substrates. DeepViNE uses the Dueling Deep Q-Network (DQN) (a variant of DQN [26]) and has nine possible actions. The authors succeed in maximizing revenue while minimizing network cost. It is worth recalling that DeepViNE is applicable only on grid-like networks which drastically limits its real deployment. RDAM was proposed in [43] for the VNE problem trying to jointly maximize long-term average revenue, long-term revenue-to-cost ratio, and acceptance ratio. In RDAM, the number of actions is set as in [41] (i.e., equals the number of physical nodes). Although they succeed in their aforementioned objectives, the use of a heavy spectral method to reduce feature matrix dimension makes RDAM not suitable for large networks. NFVdeep [40] employs a DRL based on policy gradients to place SFCs with a good balance between the operating cost of the occupied servers and the total throughput of accepted requests of the NFV providers and the profits of the customers. Nevertheless, policy-gradient agents require more DRL iterations compared to value-based agents as they do not reuse all past trial and error experiences : they thus take more time to converge.

E. Combined approaches (Exact+DRL and Heuristic+DRL)

DRL can be also combined with exact and heuristic approaches. Jianing Pei et al. [27] formulated a BIP problem for SFCs requests arriving in a SDN/NFV-enabled network. Their objective was to provide good VNF placement decisions. Dividing the network into regions, they set up a Double Deep Q-Network (DDQN) [37] algorithm with the number of actions equal to the number of regions and the reward function is based on the BIP problem objective function. Their DDQN agent then interacts with the network by selecting a region to optimize which consists in increasing or decreasing VNFs instances in selected region. Except for the difficulty to define and determine network regions clearly, their approach was innovative and showed good results. Jose Jurandir et al. [11] proposed HA-DRL, an NSP solution that combines heuristic with DRL. In their approach, they inserted the "Power of two Choices" [5] as a layer in a A3C [25] agent's neural network. In each interaction, the agent is set to provide a physical node index, the one he thinks most appropriate to host current VNF of ongoing NSPR. The role of the heuristic layer is to influence the actions of the agent to make better decisions. Although their simulations showed good performance for networks with up to 1008 nodes, they were forced to provide an ILP formulation for the heuristic part. The same authors extended their work to non-stationary network conditions [6] and realistic network load conditions [13]. [32] has taken a similar approach while replacing A3C agent by a REINFORCE [39] based agent. Combining DRL with other approaches seems to be a good way to improve DRL convergence. Nevertheless, doing so, full-automation is not possible anymore as there must be an expert to design the mathematical formulation cautiously avoiding conflicts between the heuristic optimization's objective function and the agent goal.

F. Motivation

In view of the performance of pure DRL methods, they offer a good tradeoff between scalability (e.g., [40] performed well for a network with 500 nodes), automation ability and execution time among other things. However, all the aforementioned state-of-the-art works used a sequential DRL approach, i.e., a VNF placement decision is sent to the environment which tries to perform it, then compute next observation along with a reward and send them to agent for providing next VNF decision. Anyway, even in the best situations, that is, considering a container image of size 1GB, a VNF instantiation time takes on average 10 seconds [2]. Thus, a sequential deployment time is linearly proportional to the number of VNFs in NSPR. In worst-cases (e.g., when at least one decision is bad and has to be restarted or when non-container-based images are used), it could even take more time. Accordingly, along with the required time to compute an observation, we claim that a sequential DRL iteration has a latency of at least 11 seconds : we name it the VNF-PC DRL Loop Latency (VDLL). This uncontrollable parameter was neglected in previous studies. Moreover, a sequential approach is not compatible with the ETSI NFV MANO framework, as the NFVO waits for a Network Service (NS) request with its specified VNFs placements in one instruction and not iteratively [14]. Thus, the exhibited performances in the studies

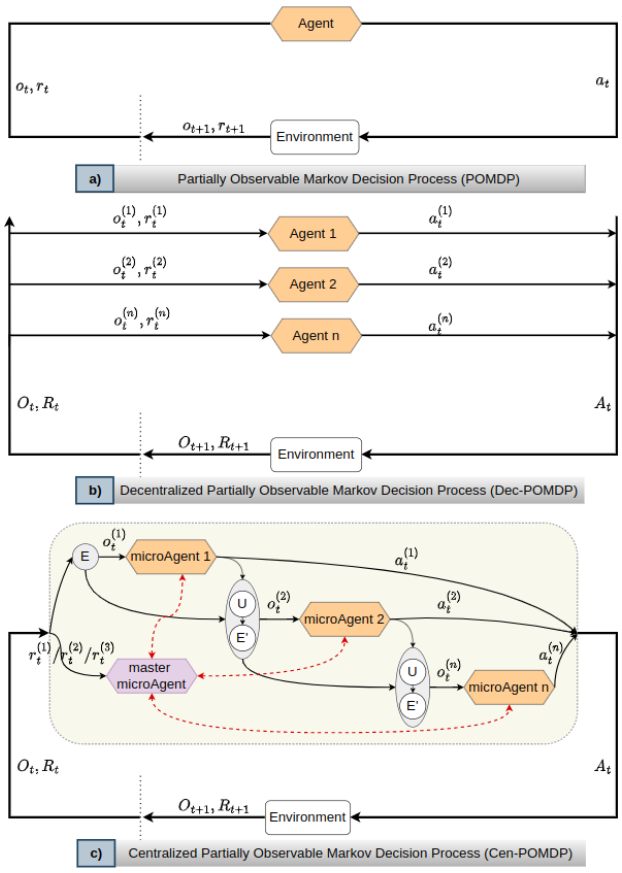


Fig. 3. Overview of existing reinforcement learning frameworks (a and b) and the one introduced in this paper (c)

are available only in simulated environments. In this work, we propose a new DRL framework to efficiently solve the Network Slice Placement problem. Different from the classic reinforcement learning framework (Figure 3-a) and the fully decentralized multi-agent framework (Figure 3-b), the framework we introduced in this paper in Figure 3-c has an internal mechanism similar to multi-agent while being externally as a single agent with centralized learning. Consequently, we overcome the limitation of sequential DRL and avoid the difficulty of fully independent decentralized agent training [44]. Under the VDLL parameter and using oneshot placement decisions, we make our framework compatible with ETSI framework.

III. Problem Statement

A. Physical Substrate Network

Let $n \in N$ be a node in the set N of nodes contained in the physical substrate network, $S \subseteq N$ the set composed of computing nodes able to execute a VNF and $s \in S$ any node of that set. For sake of simplicity, we consider each computing node to be labeled with a number in $[1, |S|]$. Note in the following sections, we use s indifferently as an entity (i.e., a computing node) or its label (i.e., $s \in [1, |S|]$) and we use physical substrate network and physical infrastructure interchangeably.

1) Physical node description: Each computing node or NFV-capable node s within an infrastructure is characterized by its available CPU resource $\mathcal{R}_s^{cpu}$, its available RAM resource $\mathcal{R}_s^{ram}$ and its available amount of storage capacity $\mathcal{R}_s^{stor}$.

2) Physical link description: In the infrastructure, a physical link is a link between a computing node and a SDN-capable switch, a link between a SDN-capable switch and a router or between two routers in the transport network. For each link, we consider in this paper the available bandwidth $\mathcal{BW}_{(m,n)}$ as a resource. The notation (m, n) is used to indicate that the link goes from the physical node $m \in N$ to $n \in N$.

3) Physical node bandwidth: We define the bandwidth of a computing or physical node as the sum of available bandwidths of its incident physical links; $\mathcal{BW}^s = \sum \mathcal{BW}_{(s',s)}$ where s' is any adjacent node of physical node s.

B. Network Slice Placement Request

We have described the entities that make up a NSPR in the following way:

1) NSPR's VNF description: An NSPR is composed of a set P of VNFs linked to each other via virtual links (example with NSPR x in Figure 1). In contrast to previous works, three requirements are considered in this paper for each VNF $v \in P$, that is, its required CPU, RAM and storage resources, respectively $\mathcal{U}_v^{cpu}$, $\mathcal{U}_v^{ram}$ and $\mathcal{U}_v^{stor}$.

2) NSPR's Virtual link description: we note $bw_{(v,w)}$ as the required bandwidth of virtual link (v, w) going from VNF $v \in P$ to VNF $w \in P$.

3) NSPR's VNF bandwidth: We introduce the notion of the bandwidth of a VNF as the sum of the required bandwidths of its associated virtual links; $bw^v = bw_{(u,v)} + bw_{(v,w)}$ where u and w are two adjacent VNFs to VNF v.

C. Remaining resource formalization

We denote by $\overline{\mathcal{R}}_{v,s}^{cpu}$, $\overline{\mathcal{R}}_{v,s}^{ram}$ and $\overline{\mathcal{R}}_{v,s}^{stor}$ respectively the remaining CPU, RAM and storage resources in computing node s after placing VNF v on it. Thus, $\overline{\mathcal{R}}_{v,s}^{cpu} = \mathcal{R}_s^{cpu} - \mathcal{U}_v^{cpu}$, $\overline{\mathcal{R}}_{v,s}^{ram} = \mathcal{R}_s^{ram} - \mathcal{U}_v^{ram}$ and $\overline{\mathcal{R}}_{v,s}^{stor} = \mathcal{R}_s^{stor} - \mathcal{U}_v^{stor}$. On the other hand, after allocating $bw_{(v,w)}$ to the virtual link (v, w), the remaining bandwidth on the physical link (m, n) is $\overline{\mathcal{BW}}_{(m,n)/(v,w)} = \mathcal{BW}_{(m,n)} - bw_{(v,w)}$.

D. Network Slice Placement formalization

NSP problem can be formulated as follows: i) given a network slice placement request with its VNFs and VLs requirements; ii) given a physical infrastructure with resource constraints on its computing nodes and physical links; iii) find for each VNF (respectively VL) in NSPR, an optimal computing node (respectively physical link) in physical infrastructure to place it on (respectively match it with). This kind of formalization was widely used in previous works as in [11].

IV. ONESHOT DRL AGENT FOR NSPR

A. Oneshot DRL agent

An overview of our proposed DRL agent interacting with its environment is provided in Figure 4. The operation of the oneshot DRL agent begins with the reception of an Extendable Reinforcement Learning Observation (ERLO), as illustrated in Figure 6. This constitutes a real-time description of the environment and comprises two parts: 1) $\mathcal{P}_1$ describes the physical infrastructure, specifically detailing the available CPU, RAM, storage space, and bandwidth resources of computing nodes, and 2) $\mathcal{P}_2$ describes the NSPR to be processed, which includes the required CPU, RAM, storage space, and bandwidth resources of the VNFs. For a given physical infrastructure, the size of $\mathcal{P}_1$ remains constant. The variability in the size of the observation is therefore contingent upon that of $\mathcal{P}_2$, which changes according to the number of VNFs contained within the NSPR being processed : this justifies the term "Extendable Reinforcement Learning Observation."

The first mechanism (mechanism 1 Figure 4) following the reception of an ERLO involves the separation of $\mathcal{P}_1$ from $\mathcal{P}_2$, based on two key parameters: the number $|N|$ of computing nodes and the fixed number of features per computing node set at 4 in this paper (CPU, RAM, storage space, and bandwidth).

The second mechanism (mechanism 2 Figure 4) involves extracting information about the number of VNFs contained within the NSPR to be processed, i.e., the parameter $|P|$ from $\mathcal{P}_2$, and instantiating or updating $|P|$ microagents, which are replicas of Master microagent. The latter model is provided to the oneshot DRL agent at runtime. The size of the action space of each microagent is equal to the number of computing nodes $|N|$ in the physical infrastructure, and is, therefore, unaffected by the variations in the number of VNFs $|P|$ within the processed NSPR (described by $\mathcal{P}_2$).

In the third mechanism (mechanism 3 Figure 4), each of the $|P|$ microagents respectively generates placement decisions for each of the $|P|$ VNFs within the NSPR. To do this, the first microagent is provided with an observation resulting from the concatenation of $\mathcal{P}_1$ and the information about the first VNF (extracted from $\mathcal{P}_2$). The first microagent proposes the placement decision for the first VNF, which is simply the index of the computing node where the VNF 1 should be placed. An update, denoted as $\mathcal{P}_1'$, is carried out using the module (U), the operation of which is detailed in Algorithm 1. The observation to be provided to the second microagent is then obtained by concatenating $\mathcal{P}_1'$ with the information regarding the second VNF (extracted from $\mathcal{P}_2$). This process continues to generate placement decisions for each subsequent VNF until the last VNF in the NSPR. The set of placements proposed by the microagents constitutes the placement solution for the entire NSPR. The complete solution is sent to the environment at the same time for implementation and evaluation. The term "oneshot DRL agent" derives precisely from the agent's ability to suggest the placements of all the VNFs within NSPR in a single DRL iteration. Thus, "oneshot DRL agent" should not be confused with "oneshot learning", which is instead employed in supervised classification tasks. It is important to note that in Deep Reinforcement Learning (DRL), only the environment has the capability to execute actions and measure their optimality. For each of the VNFs within the NSPR, the environment applies the proposed placement decision and generates an evaluation in the form of a reward, the function of which is detailed in Section IV-B. These rewards are then transmitted to the agent.

Following the receipt of rewards, the fourth and final mechanism (mechanism 4 Figure 4) involves the construction of experiences. Each microagent has undergone a different experience $e = (s_t, a_t, r_t, s_{t+1})$, which can be summarized as follows: 1) insertion of an observation s_t in its input; 2) generation of a placement decision a_t; 3) receipt of a reward r_t associated with the placement decision and 4) creation of the subsequent observation s_{t+1} resulting from the update performed by the module (U). These experiences are collected by the oneshot DRL agent, which inserts them into its unique replay memory. Periodically, samples of these experiences are extracted to train the master microagent according to equation 1.

$$Y^{DDQN} = r_t + \gamma Q_{target}(s_{t+1}, \underset{a}{\operatorname{argmax}} Q_{eval}(s_{t+1}, a)) \quad (1)$$

where γ in $[0, 1]$ is the discount factor determining how much future rewards estimations are taken into account for current target's compute.

The enhancement of Master microagent's policy is subsequently made available to the microagents through mechanism 2. In Figure 5, we provide an image-based explanation of the process for training the Master microagent's neural networks using experiences sampled from the replay memory.

Algorithm 1 pseudo-code of Update module

INPUTS :
$\mathcal{P}_1^i$ ($\mathcal{P}_1$ provided to ith microagent)
$\{\mathcal{U}_v^{cpu}, \mathcal{U}_v^{ram}, \mathcal{U}_v^{stor}, bw^v\}$ (ith VNF required resources)
a_i (ith microagent placement decision)
OUTPUT :
$\mathcal{P}_1^{i'}$ (updated $\mathcal{P}_1^i$)

Set $\mathcal{P}_1^{i'} \leftarrow \mathcal{P}_1^i$
On computing node at index a_i in $\mathcal{P}_1^{i'}$ **do**
  Set $\mathcal{R}_s^{cpu} \leftarrow \mathcal{R}_s^{cpu} - \mathcal{U}_v^{cpu}$
  Set $\mathcal{R}_s^{ram} \leftarrow \mathcal{R}_s^{ram} - \mathcal{U}_v^{ram}$
  Set $\mathcal{R}_s^{stor} \leftarrow \mathcal{R}_s^{stor} - \mathcal{U}_v^{stor}$
  Set $\mathcal{BW}^s \leftarrow \mathcal{BW}^s - bw^v$

return $\mathcal{P}_1^{i'}$

B. Reward function

We design the reward function below based on whether a VNF was successfully placed or not.

$$\begin{cases} r_{successful} = -\sum \frac{1}{\overline{\mathcal{R}}_{v,s}^{res} + \eta} \\ r_{unsuccessful} = \max(-100, -\frac{3}{\eta} + \sum \mathbb{I}[\mathcal{R}_s^{res} < \mathcal{U}_v^{res}] \frac{\overline{\mathcal{R}}_{v,s}^{res}}{\eta} + \\ \mathbb{I}[\mathcal{BW}_{(m,n)} < bw_{(v,w)}](\overline{\mathcal{BW}}_{(m,n)/(v,w)})) \end{cases}$$

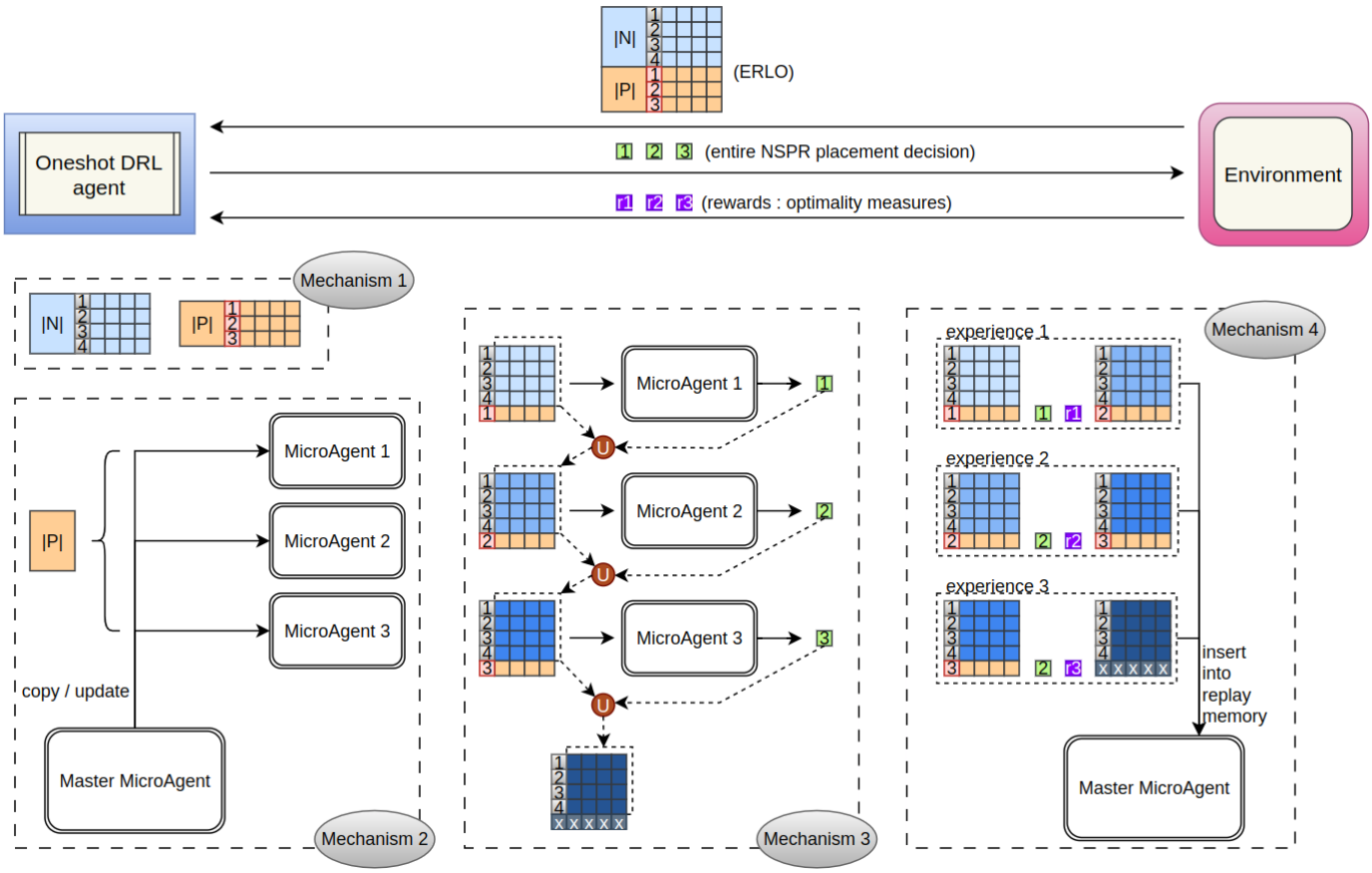


Fig. 4. Overview of the internal mechanisms of the oneshot DRL agent during a DRL iteration with the environment. Physical infrastructure is contained within the environment, where the generation/reception of NSPRs also takes place.

where res $\in$ {CPU, RAM, STOR}, η is a small value set to 0.01 avoiding division by zero in $r_{successful}$ while increasing penalty in $r_{unsuccessful}$ and $\mathbb{I}[condition]$ a binary value equals to 1 when $[condition]$ is satisfied, 0 otherwise.

In this reward function, $r_{successful}$ is intended to induce the agent to choose a computing node where once the VNF placement is done, there will be enough resources left. In this case, the difference $\mathcal{R}_s^{res} - \mathcal{U}_v^{res} = \overline{\mathcal{R}}_{v,s}^{res}$ will be positive and large, making the ratio $-\frac{1}{\overline{\mathcal{R}}_{v,s}^{res}+\eta}$ minimal, which represents a smaller negative reward for the agent.

In contrast, in the case of unsuccessful placement, the difference $\mathcal{R}_s^{res} - \mathcal{U}_v^{res} = \overline{\mathcal{R}}_{v,s}^{res}$ will be negative. The agent is then penalized proportionally to the amount of resources missing to have a successful placement. This penalty is also accentuated by division by η and addition with $-\frac{3}{\eta}$. Finally, we bounded $r_{unsuccessful}$ setting its low bound at -100 through the use of max function.

Note that the reward function has been made negative because a positive value could influence a microagent to always choose the same action that would generate a positive reward even if the overall outcome of the placement of the whole NSPR turned out to be unsatisfactory. With this negativity, each microagent will therefore always strive to do better.

V. Evaluation Results

A. Comparison algorithms

We compared our oneshot DRL agent with six other approaches.

- Algorithm 1 (sequential DRL): where a DRL agent places the VNFs contained in a NSPR one after another during successive DRL iterations. We have implemented the reward function proposed in [11].
- Algorithm 2 (ILP): where a mathematical formulation of the network slice placement problem is proposed with binary decision variables and constraints on the resources available on computing nodes and those required by the VNFs and VLs of all NSPRs. We adopt the ILP formulation of [12].
- Algorithm 3 (GAVA [30]): where a genetic approach is adopted to solve the network slice placement problem. The algorithm is based on the notion of individuals. Each of them represents the possibility of placing a request on the physical infrastructure. As an example, in Figure 9 we have provided an illustration of two possible individuals for placing NSPR x in Figure 1 on the physical infrastructure contained in the same figure. The algorithm starts with a random initialization of a number of individuals before passing them several times

	Infrastructure 1	Infrastructure 2
Computing node's initial CPU — RAM — STORAGE	randomly in $[800, 1000]$	
Computing node's physical link initial bandwidth	500	
Transport network physical link initial bandwidth	700	
Number of VNFs per NSPR	10	
VNF's initial CPU — RAM — STORAGE	randomly in $[2, 5]$	
Number of VLs per NSPR	#(number of VNFs) - 1	
VL's intial bandwidth	randomly in $[2, 5]$	
Number of NSPRs	200	600
VNF instanciation latency — Observation computation latency	10 seconds — 1 second	
Neural Network structure	Hidden layers: 2 — Neurons per hidden layer: 600	
Microagents exploration method during training	ϵ-greedy (start=1.0 ; end=0.1 ; decaying step=3x10^{-6})	
Microagents' Local State size	28	72
Optimizer used	Adam(learning rate=2.5x10^{-4})	
γ value for equation 1 computation	0.99	
Iterations between target network's updates	100	
Master microagent Replay Memory's size — Minibatch size	10^6 experiences — 128 experiences	
ns3gym simulator — Operating System	version 3.35 — Ubuntu 20.04.5 LTS	
Computer used for simulations	DELL Precision 7560 i9 with 64Gb RAM and 1To disk space	

TABLE I
SIMULATION SETTINGS

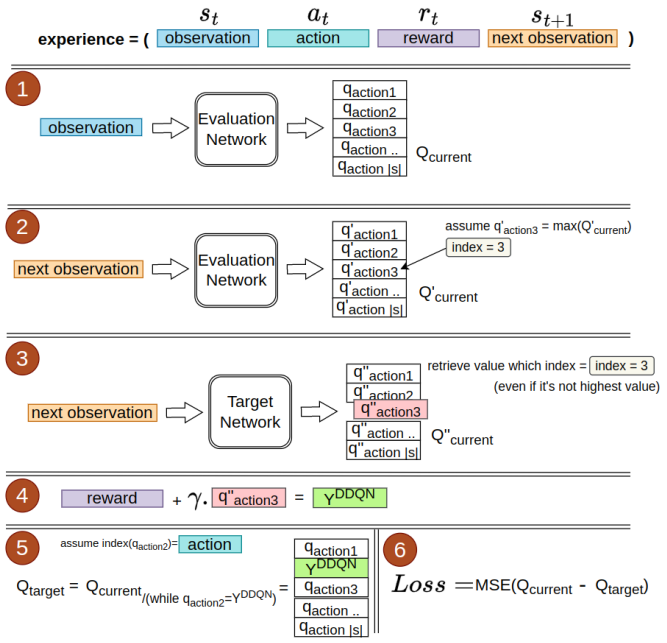


Fig. 5. Master microagent training process based on DDQN algorithm. Observation s_t from experience is an input evaluation network that outputs current estimated discounted cumulative rewards (Q-values) [step 1]. To update a neural network, there must be a current output and a desired output. Already having current output, the next goal is to search for the desired output (target). Next observation s_{t+1} of experience is input into 1) policy network and retrieve index of its maximal Q-value and 2) target network and retrieve Q-value at the index got at 1) [steps 2 and 3]. The Q-value recovered from the target network is finally multiplied by the discount factor γ (an hyperparameter) and added to the reward r_t [step 4] and substituted at the index a_t of a copy of current output (got at step 1) to obtain the desired output [steps 5 and 6]. A loss function (e.g., Mean Squared Error) is then used along with computed current output and desired output to generate the loss. Finally, an optimizer (e.g., Adam) is used to back-propagate the loss and update evaluation network's weights. Note that figure this is an illustration with a single experience and that in practice a random batch of experiences is instead taken from Replay Memory to proceed evaluation network update.

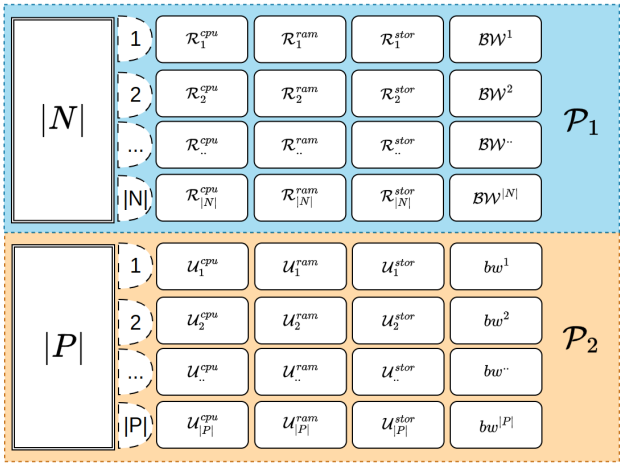


Fig. 6. Overview on introduced Extendable Reinforcement Learning Observation. The part $\mathcal{P}_1$ highlighted in blue represents a real-time description of the physical infrastructure contained within the DRL environment. The parameter $|N|$ indicates the number of computing nodes in the infrastructure, and each line of 4 values ($\mathcal{R}_s^{cpu}$, $\mathcal{R}_s^{ram}$, $\mathcal{R}_s^{stor}$ and $\mathcal{BW}^s$) provides information about the remaining resources of each computing node in terms of CPU, RAM, storage space, and bandwidth. Part $\mathcal{P}_2$ highlighted in orange serves as a real-time description of the NSPR to be processed. It informs about the number $|P|$ of VNFs within the request, along with the CPU ($\mathcal{U}_v^{cpu}$), RAM ($\mathcal{U}_v^{ram}$), storage space ($\mathcal{U}_v^{stor}$), and bandwidth (bw^v) requirements of each VNF. For a fixed infrastructure, the size of ERLO, denoted by the parameter $(|N| * 4) + (|P| * 4) + 2$, can vary depending on the characteristics of the NSPRs being processed. Hence, the term "Extendable Reinforcement Learning Observation" is used, as the size of ERLO may change from one DRL iteration to another based on the specific NSPR's size $|P|$.

through successive steps (repairer - evaluation fitness - selection - crossover - mutation) to improve their quality.

- Algorithm 4 (RandomLogicAgent): a heuristic consisting of placing each NSPR's VNF randomly on a computing node of the target physical infrastructure.
- Algorithm 5 (MaxOperatorAgent): a heuristic consisting, for each NSPR's VNF, in determining the computing nodes having 1) the most CPU resources; 2) the most RAM resources; 3) the most storage space resources; and in placing said VNF on the first of these nodes having

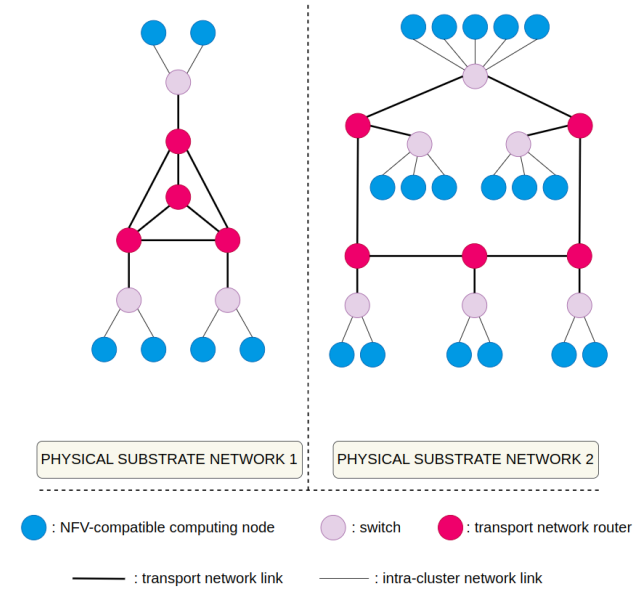


Fig. 7. Physical substrate networks used for experiments

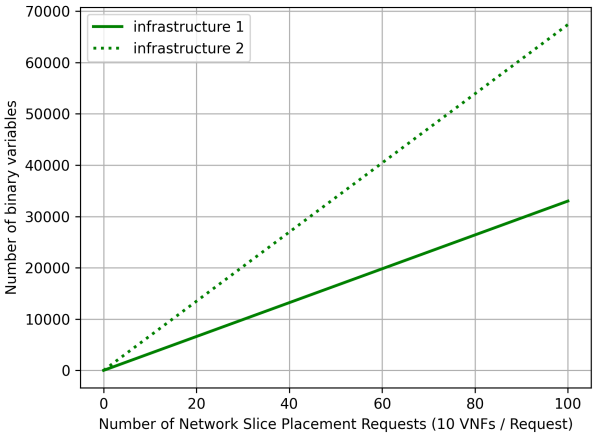


Fig. 8. Evolution of the number of binary variables in the ILP optimization problem for each of the infrastructures shown in Figure 7. For 100 requests, there is respectively 33000 and 67400 variables to manage for Infrastructures 1 and 2 respectively. It is possible to exploit the observed linear relationship to estimate the number of variables for 200 requests.

	V1/S1	V1/S2	V1/S3	V1/S4	V1/S5	V1/S6	V2/S1	V2/S2	V2/S3	V2/S4	V2/S5	V2/S6	V3/S1	V3/S2	V3/S3	V3/S4	V3/S5	V3/S6
Individual 1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0

	V1/S1	V1/S2	V1/S3	V1/S4	V1/S5	V1/S6	V2/S1	V2/S2	V2/S3	V2/S4	V2/S5	V2/S6	V3/S1	V3/S2	V3/S3	V3/S4	V3/S5	V3/S6
Individual 2	0	1	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0

Fig. 9. Example of two possible individuals during GAVA's [30] execution while placing NSPR x of Figure 1 on its physical infrastructure. Individual 1 means to place VNF 1, 2 and 3 on computing nodes 1, 2 and 4 respectively. Individual 2 means to place VNF 1 simulatenously on computing nodes 2 and 4, VNF 2 on computing node 1 and VNF 3 simultaneously on computing nodes 1, 2 and 3. Therefore, individual 2 is considered as non-viable individual and should pass through a repairing step to correct its inconsistencies.

enough resources with respect to the VNF's requirements.

- Algorithm 6 (FirstFitAgent): a heuristic consisting, for each NSPR's VNF, in placing it on the first computing node having enough resources. It should be noted that a fixed order of the computing nodes is required at the beginning of the algorithm. In this paper, we have used the indexes of computing nodes for the traversal.

The second part of the network slice placement problem is the chaining of placed VNFs. Algorithms 1, 4, 5 and 6 use the Breadth First Search (BFS) for this purpose as done in [42]. Instead, a modified version of BFS (mBFS) is used by the oneshot DRL agent. mBFS is an extension we made of BFS algorithm in which a criteria of minimum bandwidth was added enhancing it to provide a path between two nodes while satisfying that minimum bandwidth constraint. We provide in 2 a pseudocode of mBFS. As for ILP and GAVA, these two algorithms generate both placements and chaining and do not require chaining algorithms.

Algorithm 2 mBFS's pseudo-code used at VNFs chaining step

INPUTS : start_node, end_node, minimum_bw
OUTPUT : a path from start_node to end_node verifying minimum_bw

Initialize **paths** (to store computed paths)
Initialize **path** (an empty path)
Initialize **neighbors** (to store a node's neighbors)
Initialize **node** (to store current node)

Add path {start_node} into **paths**
While **paths** not empty **do**
    get and remove first path from **paths** → **path**
    get last node of **path** → **node**
    If **node** equals end_node **then**
        return **path**
    get **node**'s neighbors verifying (1) and (2) → **neighbors**

    (1) neighbor not in **path**
    (2) bandwidth(**node**, neighbor) >= minimum_bw
    For each neighbor in **neighbors** **do**
        insert new path {**path**+neighbor} into **paths**
    empty **neighbors**
return empty path

B. Comparison metrics

We compared the performance of our oneshot DRL agent with these algorithms based on three metrics.

- average execution time: this is an approximation of the time it takes for concerned algorithm to process and generate whole NSPR's decision then sends it to environment
- the acceptance ratio: it reflects the ability of concerned algorithm to successfully process a large number of received NSPRs. Its calculation is based on the formula $\frac{\#Number\ of\ successful\ NSPRs}{\#Total\ number\ of\ NSPRs}100$.
- load balancing: which proves the ability of the algorithm to fairly exploit the resources of all the computing nodes

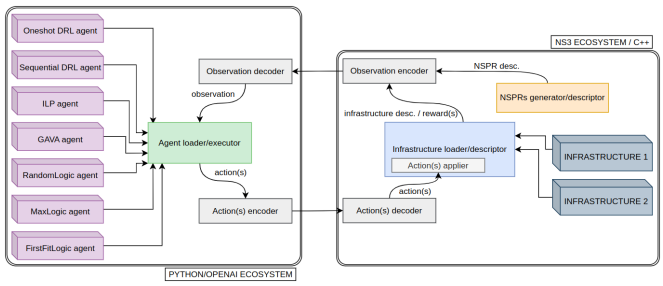


Fig. 10. Used simulator : based on ns3-gym framework

in target physical infrastructure. This metric characterizes the imbalance between the resources of computing nodes after the algorithm has processed all NSPRs addressed to it.

- adaptability: this characterizes the ability of the algorithm to process a NSPR while taking into account events that occurred during the processing of previous requests

C. Physical infrastructures in comparison

Our performance metrics were measured in simulations performed on two physical infrastructures visible in Figure 7. Infrastructure 1 is inspired by the one used in [12] and is made up of 6 computing nodes, 3 SDN-enabled switches, and a transport network of 4 routers. Infrastructure 2 is made up of 17 computing nodes, 6 SDN-compliant switches, and a transport network of 5 routers. It is based on the one provided in [11]. We have recorded in table I how we defined the resources of each infrastructure, as well as other information useful for the reproducibility of all simulations conducted in this paper.

D. Simulator

To conduct our simulations, we employed an extended version of the ns3-gym simulator [17], which is a framework that connects ns3 to OpenAI-Gym. This choice was made to facilitate the reproducibility of the results and allow for the modular implementation of different entities in the simulation, as depicted in Figure 10. All algorithms presented in Section V-A were implemented as agents that interact with the environment in a seamless manner. This approach simplifies the testing of algorithms and makes it easy to compare with other algorithms present in literature. On the environment side, we implemented two infrastructures as shown in Figure 7. To begin a simulation, we simply choose the infrastructure and the agent to load. An automatic infrastructure descriptor, along with an applicator for actions from agents, and a configurable NSPRs generator were also implemented.

E. Fine-tuning the hyperparameters of the oneshot DRL agent

The oneshot DRL agent has hyperparameters that influence its performance. These hyperparameters are specific to the DDQN algorithm used by agent. We carried out fine-tuning in order to choose the right values of hyperparameters.

We considered four high-impact hyperparameters: learning rate, gamma, target network update interval and mini-batch size. The authors of the DDQN propose some default values which are respectively 0.001; 0.99; 10000 and 32. We used these values as a starting point by i) maintaining values of hyperparameters already fine-tuned if any; ii) varying value of hyperparameter being fine-tuned and; iii) keeping default values of hyperparameters not yet fine-tuned.

To monitor performance improvement, agents are trained on 1000 different (varying seed) training episodes with an evaluation of the number of successful NSPRs per interval of 10 training episodes. This evaluation takes place in a reference environment that remains identical (i.e., with fixed seed), so that at each evaluation comparison is possible with the previous evaluations. Results are expressed in relative percentages where 100% is the percentage of the best performing agent and remaining agents' percentages are determined based on formula $\frac{\#Number\ of\ successful\ NSPRs\ achieved}{\#Number\ of\ successful\ NSPRs\ achieved\ by\ best\ agent}100$

1) Learning rate fine-tuning: Learning rate determines how fast agent neural networks' weights are updated. A too high value could lead to instability as agent erase previously learned policy while a too small value leads to a slower agent's convergence which may cause difficulty in adapting to environment. We tested 4 values ($2.5 * 10^{-2}$, $2.5 * 10^{-3}$, $2.5{*}10^{-4}$ and $2.5{*}10^{-5}$) while keeping gamma, target network update interval, and mini-batch size equal, respectively, to 0.99, 10000 and 32. Figure 11-a shows the performance of each agent on Infrastructure 1. We can see that agents with learning rates of $2.5 * 10^{-2}$ and $2.5 * 10^{-3}$ have the worst performance. This means that these values are too high. On average, maximum performance is achieved by $2.5 * 10^{-4}$ while $2.5{*}10^{-5}$ shows intermediate performance. This trend is confirmed in Infrastructure 2 (Figure 11-e) where, as training episodes progress, agents with learning rates $2.5 * 10^{-4}$ and $2.5 * 10^{-5}$ maintain the best performances, with $2.5 * 10^{-4}$ being much better at the end of training. In view of Figures 11-a and 11-e, we therefore retained $2.5{*}10^{-4}$ as the learning rate value for the remainder of the fine-tuning process.

2) Gamma fine-tuning: This hyperparameter is the one in equation 1 and is known as the discount factor. It is a value between 0 and 1 that determines how much importance the agent places on future rewards compared to immediate rewards. A value of 0 means only immediate reward matters. In other words, the discount factor allows the agent to make decisions that balance short-term rewards with long-term rewards. We set fine-tuned learning rate i.e., $2.5 * 10^{-4}$, default values of target network update interval and mini-batch size i.e., 10000 and 32, then we vary γ's value. Achieved performance in Infrastructures 1 and 2 are visible respectively in Figures 11-b and 11-f. It appears that there is no linear relationship between the γ' value and the performance of the associated agent. Thus, in both infrastructures, the agent with $\gamma = 0.75$ presents the weakest performance compared to the one with $\gamma = 0.25$. The agent with $\gamma = 0.5$ performs well in Infrastructure 1 but poorly in Infrastructure 2. On average, best performances are exhibited by agents with $\gamma = 0.25$ and $\gamma = 0.99$. However, as it is difficult to distinguish between them in view of their alternating performances, we preferred to keep $\gamma = 0.99$,

especially since it is the default value suggested by the authors of DDQN.

3) Target network update interval fine-tuning: The target network is a separate neural network that is periodically updated to create a stable target for the Q-value predictions during training. It serves as a "frozen" copy of the main Q-network, and its parameters are updated periodically to match those of the latter. Thus, this hyperparameter determines the update frequency. Its value can be any positive integer and depends strongly on task being performed by agent. We set a fine-tuned learning rate ($2.5 * 10^{-4}$) and γ (0.99), keep the default value of the mini-batch size (32) and vary the update interval. Observed performances are showed in Figures 11-c and 11-g. On Infrastructure 1 (Figure 11-c), highest performance is achieved by agent which updates target network each 100 iterations. The second-best performance is shared by the agents operating updates each 1000 and 10000 iteration. It is worth noting that updating the target network less frequently (20000) shows the worst performance. Nevertheless, only agents updating the target network at frequency of 100 and 10000 performed well on Infrastructure 2 (Figure 11-g). Specifically, the one with 100 have a better mean performance. Consequently, we set the target update interval equal to 100 for the remaining fine-tuning.

4) Mini-batch size fine-tuning: As the oneshot DRL agent keeps its experiences in a replay memory, it is essential to set how many samples will be taken when training the main neural network. To measure the effect of mini-batch size, we set all previously fine-tuned hyperparameters i.e., learning rate to $2.5 * 10^{-4}$, γ to 0.99, target network update interval to 100 and then we test four values of mini-batch size mainly 32, 64, 92 and 128. Through performance results on Infrastructure 1 (Figure 11-d), it seems there is a linear relationship between mini-batch size and achieved performance : higher is mini-batch size, better is performance. Thus, we set the mini-batch size equal to 128 and that value appears to also be the one that provides far better performance on Infrastructure 2 (Figure 11-h). Note that during our simulations, we noticed a slowness in training of agents as the mini-batch size increased. Thus, by keeping value 128 and not going beyond, we aim at finding a trade-off between performance and speed.

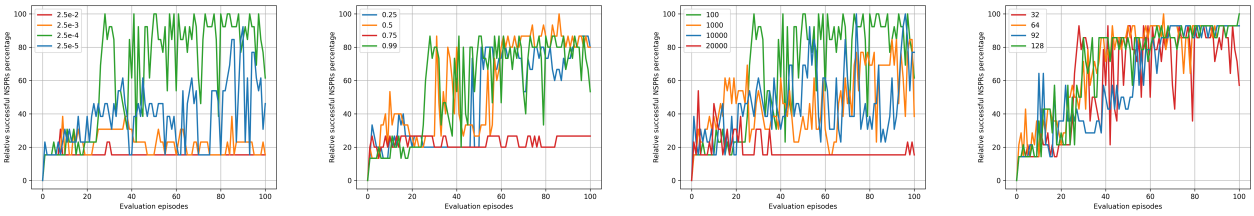
F. Analysis

1) Execution time performance: In this first batch of simulations, we measured the execution time of each algorithm. We set the number of VNFs for each NSPR to its maximum (i.e., 10) and considered an abundance of resources in CPU, RAM, storage space and bandwidth. In such a configuration, all the placement decisions of an NSPR always result in success, allowing us to measure the execution time if all the algorithms were at their peak performance.

Figure 12-a shows for Infrastructure 1 in Figure 7 the average processing time of an NSPR by each of the algorithms as the total number of requests varies. The oneshot DRL agent has an average processing time of about 13 seconds as do all heuristics (Algorithms 4 to 6). This time includes the observation latency of 1 seconds, and the instantiation time of a VNF (10 seconds). It is necessary to specify that the oneshot DRL agent and the heuristics generate the decisions of all NSPR's VNFs at once, which makes it possible to instantiate them in parallel on their respective node inside the environment, reducing the instantiation time to 10 seconds for all the 10 VNFs. However, the sequential DRL agent needs one DRL iteration per each VNF of NSPR. The VNF is instantiated on the concerned node in 10 seconds and only then the next VNF of NSPR is taken care of. The sequential DRL agent thus takes an average of 121 seconds to process a NSPR, which is 9.3 times the time required by the oneshot DRL agent, RandomLogicAgent, MaxOperatorAgent, and FirstFitAgent. The lowest average processing time, that is 2.3 seconds, is achieved by ILP. As for GAVA, it presents a very variable average processing time in $[42; 161.5]$ seconds depending on the number of NSPRs to be processed. This is due to the fact that GAVA is a genetic algorithm based on randomness in several of its steps i.e., initialization, crossover and mutation. As a consequence, individuals obtained at each generation have a low level of viability which must be remedied in the repairer step. The time required for this last step is unpredictable as it depends on the repairs to be done on the individuals at each generation.

In order to evaluate the impact of infrastructure's size on the average execution time, we conducted the same simulation considering Infrastructure 2 in Figure 7 which contains twice as many computing nodes compared to Infrastructure 1. It appears from these simulation results in Figure 12-b that the trends in terms of the average request processing time for the oneshot DRL agent, RandomLogicAgent, MaxOperatorAgent, FirstFitAgent, and sequential DRL agents remain approximately the same. As for GAVA, its average processing time varies in $[108.3; 278.9]$ seconds unpredictably. For example, it has an average time of 175.6 seconds for 60 NSPRs and 273.2 seconds for 30 NSPRs. Although this is surprising, it is perfectly explained by the use of randomness in its key steps and by the fact that each individual has a size of 170 (10 VNFs $*$ 17 computing nodes): the repair of an individual thus takes into account more values and more time than in Infrastructure 1 where each individual had a size of 60.

On the other hand, we notice a degradation of ILP performance starting from 60 requests. Its average processing time increased from 2.5 seconds (number of requests between 2 and 50) to 8.9 seconds (number of requests between 55 and 80). This is due to the fact that ILP takes into account all the requirements of all the NSPRs in order to propose the most optimal way to place them on the infrastructure. Therefore, the number of variables and constraints in the formulated optimization problem increases as the number of NSPRs increases. We confirm this analysis by the Figure 8 showing the number of binary variables in formulated optimization problem as a function of the number of NSPRs: even at 100 requests, we observe 67400 binary variables for Infrastructure 2 against 33000 for Infrastructure 1. This proves that even for small infrastructures, the performance of ILP is highly dependent on the number of NSPRs it receives. Recall that oneshot DRL, sequential DRL, RandomLogicAgent, MaxOperatorAgent and FirstFitAgent algorithms are not impacted by the number of

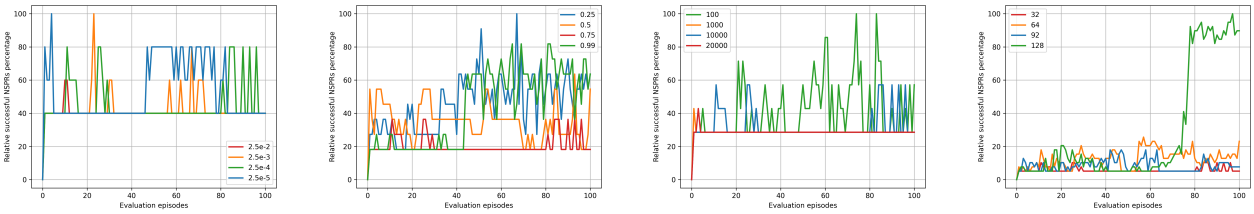


(a) Learning rate variation (Infrastructure 1)

(b) Gamma variation (Infrastructure 1)

(c) Target update interval variation (Infrastructure 1)

(d) Minibatch size variation (Infrastructure 1)



(e) Learning rate variation (Infrastructure 2)

(f) Gamma variation (Infrastructure 2)

(g) Target update interval variation (Infrastructure 2)

(h) Minibatch size variation (Infrastructure 2)

Fig. 11. Fine-tuning of the oneshot DRL agent's hyperparameters. **Top :** on Infrastructure 1; **Down :** on Infrastructure 2

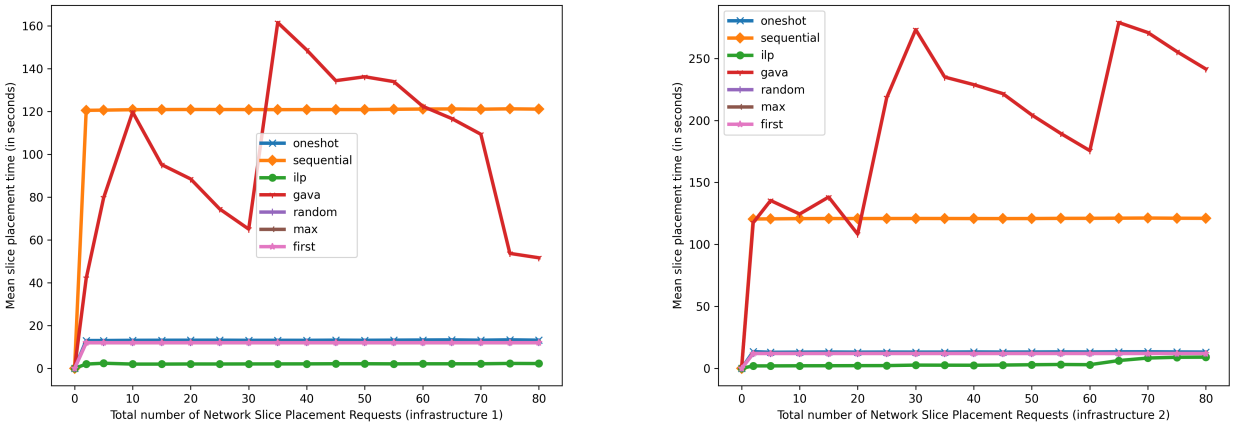


Fig. 12. Mean slice placement time vs total number of Network Slice Placement Requests. From left to right: (a) Infrastructure 1, (b) Infrastructure 2

requests because they process requests individually one after another.

2) Acceptance ratio and load balancing performances: In this second batch of simulations, we tested the efficiency of each algorithm. We quantified this efficiency by i) the percentage of success on all the NSPRs received by concerned algorithm and ii) the load balancing observed on infrastructure's computing nodes. The calculation of success percentage is presented in Section V-B. As for the calculation of load balancing, we defined the load of a computing node by $l = \frac{100(3 - \frac{\#remaining\ CPU}{\#initial\ CPU} - \frac{\#remaining\ RAM}{\#initial\ RAM} - \frac{\#remaining\ STOR}{\#initial\ STOR})}{3}$ and the load balancing of the whole infrastructure by the difference $\mathcal{G}$ between loads of the most overloaded computing node and the least overloaded one $\mathcal{G} = l^{most\ overloaded\ cnode} - l^{least\ overloaded\ cnode}$.

It is essential to note that $\mathcal{G}$ is in $[0; 100]$ and is expressed in percentage. The smaller $\mathcal{G}$, the better the load balancing on the computing nodes.

The way in which we initialize the resources of the computing nodes in each infrastructure, as well as the requirements of each NSPR and other parameters/information can be seen in table I. Also, for fairness comparison and because the sequential and the oneshot DRL agents are learning-based, we trained them on 500 different episodes than those used to evaluate all algorithms.

The results of this second batch of simulations for Infrastructure 1 are shown in Figure 13. We organized this figure in such a way that columns 1, 2 and 3 represent the results for $\mathcal{G} = 10\%$, $\mathcal{G} = 50\%$ and $\mathcal{G} = 70\%$, respectively.

The figure in row 1 column 1 shows acceptance percentages of algorithms when the maximum tolerable imbalance between

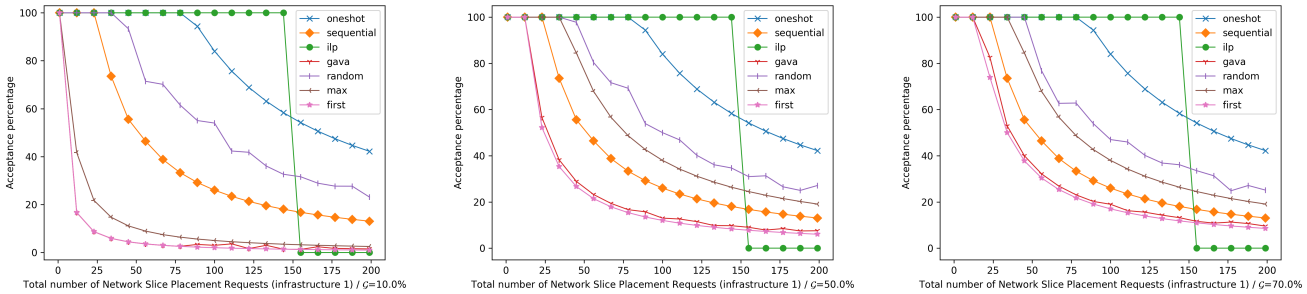


Fig. 13. Acceptance ratio vs total number of Network Slice Placement Requests, for Infrastructure 1. From left to right: $\mathcal{G} = 10\%$, $\mathcal{G} = 50\%$ and $\mathcal{G} = 70\%$

the most overloaded computing node and the least overloaded one is 10%. When there is only one NSPR to process, all algorithms achieve a percentage of 100%. This can be explained by the fact that before receiving that first request, all computing nodes had a load of 0%. Therefore, there was a large enough margin for each solution to not create an imbalance of more than 10%. At 12 NSPRs to be processed, the acceptance percentage of GAVA and FirstFitAgent drops to 16.66%, i.e., barely two requests successfully placed. For FirstFitAgent, this drop in performance is due to the fact that it places all VNFs of all NSPRs it receives on the same computing node as long as the latter has sufficient resources. As a result, the first computing node ends up with a high load, while the 5 other computing nodes of the infrastructure remain at 0% load. It quickly violates the $\mathcal{G} = 10\%$ requirement, which invalidates the proposed placement for the NSPR. Worse, FirstFitAgent proposes the same placement for each of the following NSPRs because it still trying to use resources of the same computing node regardless of load balancing requirements. Paradoxically, the same behavior is observed with the GAVA genetic algorithm. Indeed, the authors of GAVA adopted a fitness function so that the more an individual proposes to group VNFs of an NSPR on the same computing node, the better it is considered and thus will have more chance to be retained as a placement decision. With such a fitness function, GAVA's behavior is therefore equivalent to that of FirstFitAgent, hence its percentage of 16.66%. MaxOperatorAgent performs a little better (41.66%) because by placing the current VNF on the computing node with the most resources, it tends to balance the remaining resources. However, in the Network Slice Placement problem, balancing remaining resources is not enough to have a good load balancing. For example, if we consider a computing node $c1$ with 12 CPU and another $c2$ with 6 CPU and we place VNFs so that each one has 6 CPU free, we will have generated a load of 50% on computing node $c1$ while computing node $c2$ will be at 0%. This explains why MaxOperatorAgent also violated the $\mathcal{G} = 10\%$ requirement at the risk of having its placement decisions invalidated. The oneshot DRL agent and the sequential DRL agent succeeded in placing all their requests as did RandomLogicAgent. The performance of the latter may seem surprising, but it is subject to randomness. Therefore, it is not an exaggeration to say that it is a fluke. Especially since, for the same NSPRs, its decisions vary from one moment to the next randomly. At 110 NSPRs, except for the oneshot DRL agent and ILP, all algorithms are at a success percentage less than 50% : 23.42% for sequential DRL agent, 3.60% for GAVA, 42.34% for RandomLogicAgent, 4.50% for MaxOperatorAgent and 1.80% for FirstFitAgent. If the performance degradation of RandomLogicAgent is acceptable, the one of the sequential DRL agent is justified by its reward function (the one of [11]) which evaluates the performance more thoroughly on the entire request placement than on each VNF placement. Indeed, it is equal to 0 as long as the last VNF has not been processed. The sequential DRL agent thus ends up with experiences where the proposed placement for a VNF is judged both "not good" and "not bad". In contrast, the reward function proposed in this paper is placement oriented and allows the oneshot DRL agent to learn from each placement decision of each VNF. This explains its performance of 75.67%. Note that ILP keeps a performance of 100% (number of requests between 1 and 150) due to the fact that the optimization problem is formulated and solved only once all the NSPRs are received, which differs from the behavior of other algorithms that process the NSPRs one after the other as they are received. However, from 155 requests, the percentage of ILP drops drastically to 0%. In fact, as the number of NSPRs increases, there are more constraints on the resources, and as soon as one constraint is unsatisfiable in the optimization problem, no solution can be found. ILP thus has an "all or nothing" logic where even last NSPR can destabilize all the previous ones. Moreover, it remains difficult to identify that NSPR for not taking it into account, for example.

Figures in row 1, columns 2, and 3 show the performance of each algorithm when tolerating 50% and 70% imbalances, respectively. This shows a slight improvement in the performance of GAVA and FirstFitAgent and reflects more overloading of the first nodes of Infrastructure 1 when the last nodes are at 0% load. MaxOperatorAgent also improves to the point of surpassing the sequential DRL agent while still underperforming RandomLogicAgent, ILP and the oneshot DRL agent. For $\mathcal{G} = 50\%$, out of the 200 NSPRs received by each algorithm, the oneshot DRL was able to process and place successfully 84 while sequential DRL, ILP, GAVA, RandomLogicAgent, MaxOperatorAgent and FirstFitAgent placed 26, 0, 15, 54, 38 and 12 respectively i.e., a percentage of

42.21% for Oneshot DRL, 13.06% for sequential DRL, 0% for ILP, 7.53% for GAVA, 27.13% for RandonLogicAgent, 19.09% for MaxLogicAgent and 6.03% for FirstFitAgent. ILP, meanwhile, is at 0% after a run of 100% between 1 and 150 requests. It is beyond 150 requests that it could not solve the optimization problem. This proves that no matter what concessions the infrastructure provider makes on load balance, the all-or-nothing logic of ILP will remain unchanged.

The passage from $\mathcal{G} = 50\%$ to $\mathcal{G} = 70\%$ actually only results in a slight increase in the performance of GAVA and FirstFitAgent which pass, respectively, from 7.53% to 9.54% and from 6.03% to 8.54%. Interestingly, unlike the latter, RandomLogicAgent drops performance from 27.13% to 25.12%. This shows how the randomness of this algorithm does not allow it to take into account the needs expressed by the infrastructure provider.

The results of the simulation on Infrastructure 1 allowed us to see that $\mathcal{G} = 10\%$ is a little too strict as a load balance condition and that $\mathcal{G} = 50\%$ or $\mathcal{G} = 70\%$ gives approximately the same result in terms of algorithm performance. We therefore performed the simulations on Infrastructure 2 by setting only $\mathcal{G} = 50\%$.

Infrastructure 2 is made up of 17 computing nodes. The simulation results obtained on the acceptance percentage are in Figure 14. Between 1 to 12 requests, all algorithms have a success percentage of 100% because Infrastructure 2 with its 17 nodes has slightly enough resources than Infrastructure 1. For a few NSPRs, the placements proposed by each algorithm are then successful. At 23 NSPRs, the trend is still 100% except for GAVA and FirstFitAgent which drop to 60.86% and 52.17%, respectively. This is because these two algorithms, by their operation, have placed all VNFs on the first computing node of Infrastructure 2, i.e., a total of 230 VNFs (23 requests $*$ 10 VNFs per request). The difference $\mathcal{G}$ in computing loads between the first node that is very overloaded and the other nodes exceeds 50%, which is a violation of the requirement $\mathcal{G} = 50\%$ leading to the rejection of their placements. This trend is confirmed as the number of NSPRs increases. At 200 NSPRs, GAVA is at 7.53% and FirstFitAgent at 6.03%. As for sequential DRL, from 34 NSPRs, it continuously decreases in performance to end up at 12.56% at 200 requests i.e., barely 25 requests placed successfully. This confirms that even with enough resources on the infrastructure, the reward function proposed in [11] does not allow the agent to have the right feedback to correct its future placements. RandomLogicAgent and MaxLogicAgent overlap in terms of performance, with sometimes RandomLogicAgent managing to slightly outperform MaxLogicAgent (e.g., when number of requests equals 78, 100 or 133) and sometimes the opposite (e.g., when number of requests equals 56 or 133). In 200 NSPRs, RandomLogicAgent has an acceptance percentage of 29.64% and MaxLogicAgent has 28.64%. This shows that trying to equalize the remaining resources on the computing nodes without taking into account other issues (such as available bandwidth on the physical links) can generate performance as low as randomly placing VNFs on the computing nodes. ILP is still in its all-or-nothing logic, performing at 100% up to about 175 requests before collapsing to 0%. Oneshot

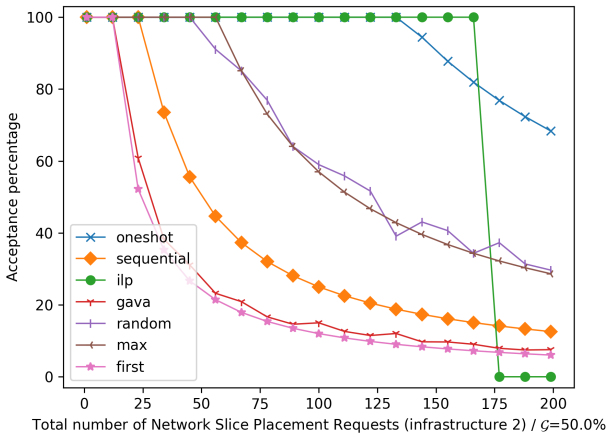


Fig. 14. Acceptance ratio vs total number of NSPRs for Infrastructure 2 and $\mathcal{G} = 50\%$

DRL held its own against ILP from 1 to 133 requests, also getting a succession of 100% before gradually dropping in performance. However, when ILP collapses from 175 requests onwards, Oneshot DRL maintained a performance of 76.83% (177 NSPRs), 72.34% (188 NSPRs) and 68.34% (200 NSPRs). This clearly shows that even when resources are very limited on the infrastructure, the oneshot DRL agent manages to place successfully a certain number of NSPRs instead of rejecting them all as ILP does. Relying on mBFS, the oneshot DRL agent also manages well to chain NSPR's VNFs, which limits the risk of placements failing due to chaining constraints. While ILP needs to know all the NSPRs before proposing placements, making it unsuitable for online scenarios, the oneshot DRL agent processes NSPRs as they arrive and relies on its past placements rewards to improve itself. Those characteristics make the oneshot DRL agent a better choice when dealing with online scenarios such as in autonomous IoT systems.

VI. Conclusion

In this paper we present a new algorithm for network slicing in IoT networks based on deep reinforcement technique. Our algorithm is fully ETSI compliant and proposes oneshot placement for slice requests. Hence, it is suitable for deployment in real-world IoT networks, which is a significant step towards enabling network operators to efficiently allocate network resources and meet the ever-increasing demands for IoT services. Our approach is hybrid between single agent and multi-agent paradigms and addresses the challenges of real scenarios without requiring prior knowledge of all slice placement requests. We implemented and tested this algorithm in the ns3-gym framework and compared its performance with six other algorithms. The results show that our algorithm is as fast as heuristics, with the extra benefit of learning from past placement decisions to improve future placements. The use of Extensible Reinforcement Learning Observation (ERLO) makes it possible to adapt to changes in the environment

without changing the algorithm. The algorithm is a promising step towards the provisioning of autonomous IoT systems with adaptable tools capable of managing service requirements without human intervention. In future work we would like to extend the optimization algorithm to include price and revenue parameters to meet the needs of both service providers and consumers.

VII. Limitations

The ability of the oneshot DRL agent to generate the placement decision for an entire NSPR in a single DRL iteration largely relies on the use of module (U) (Algorithm 1), which updates the observations provided to microagents. Consequently, each microagent operates independently while taking into account the placement decisions of previous microagents. Therefore, the oneshot DRL agent is better suited for resource allocation or management scenarios where a straightforward subtraction of allocated resources can update the remaining resources. However, we also consider that it might be feasible to employ the oneshot DRL agent in other scenarios, provided that the effect of one microagent on the input (observation) of the subsequent microagent is predictable without the need to involve the environment. While not entirely predictable, an update module based on predictive Machine Learning models could be considered. In any case, without the update of microagents' input, the oneshot DRL agent may suffer from degraded performance.

Although it includes independent microagents, the oneshot DRL agent has been proposed as a centralized solution. Thus, we think that to transform it into a distributed solution, several aspects need to be addressed: 1) the rapid supply of microagents' experiences to Master microagent; 2) the synchronization of microagents in generating placement decisions; and 3) the swift dissemination of policy improvements obtained by the Master microagent to the microagents.

List of Abbreviations

IoT	Internet of Things
QoS	Quality of Service
DRL	Deep Reinforcement Learning
ETSI	European Telecommunications Standards Institute
NGMN	Next Generation Mobile Network
5G	fifth-generation networks
NFV	Network Function Virtualization
SDN	Software Defined Network
VNF	Virtual Network Function
NSP	Network Slice Placement
VNE	Virtual Network Embedding
VNF-FGE	Virtual Network Function Forwarding Graph Embedding
SFC-P	Service Functions Chaining/Placement
VNF-PC	Virtual Network Function Placement and Chaining
NSPR	Network Slice Placement Request
VL	Virtual Link
CPU	Central Processing Unit
RAM	Random Access Memory
NFVO	Network Function Virtualization Orchestrator
MANO	MANagement and Orchestration
VDLL	VNF-PC DRL Loop Latency
ILP	Integer Linear Programming
SLO	Service Level Objectives
MILP	Mixed Integer Linear Programming
VM	Virtual Machine
SFC	Service Functions Chaining
BIP	Binary Integer Programming
LP	Linear Programming
P2C	Power of two Choices
N-PoP	Nodes Point of Presence
VNF-FG	Virtual Network Function Forwarding Graph
A3C	Asynchronous Advantage Actor-Critic
GCN	Graph Convolutional Network
InP	Infrastructure Provider
DQN	Deep Q-Network
DDQN	Double Deep Q-Network
HA-DRL	Heuristically Assisted Deep Reinforcement Learning
NS	Network Service
ERLO	Extendable Reinforcement Learning Observation
GAVA	Genetic Algorithm for VNF-FGs Allocation

References

[1] Bernardetta Addis, Dallal Belabed, Mathieu Bouet, and Stefano Secci. Virtual network functions placement and routing optimization. In *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, pages 171–177, 2015.

[2] Ibrahim Afolabi, Tarik Taleb, Pantelis A. Frangoudis, Miloud Bagaa, and Adlen Ksentini. Network slicing-based customization of 5g mobile services. *IEEE Network*, 33(5):134–141, 2019.

[3] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini, and Hannu Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys Tutorials*, 20(3):2429–2453, 2018.

[4] N. Alliance. "description of network slicing concept," ngmn 5g p, vol. 1, no. 1, 2016.

[5] Jose Jurandir Alves Esteves, Amina Boubendir, Fabice Guillemin, and Pierre Sens. Heuristic for Edge-enabled Network Slicing Optimization using the "Power of Two Choices". In *CNSM 2020 - 16th International Conference on Network and Service Management*, Izmir / Virtual, Turkey, November 2020.

[6] Jose Jurandir Alves Esteves, Amina Boubendir, Fabrice Guillemin, and Pierre Sens. DRL-based Slice Placement Under Non-Stationary Conditions. In *CNSM 2021 - 17th International Conference on Network and Service Management*, Izmir, Turkey, October 2021.

[7] Edoardo Amaldi, Stefano Coniglio, Arie M.C.A. Koster, and Martin Tieves. On the computational complexity of the virtual network embedding problem. *Electronic Notes in Discrete Mathematics*, 52:213–220, 2016. INOC 2015 – 7th International Network Optimization Conference.

[8] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, nov 2017.

[9] Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, Raouf Boutaba, and Otto Carlos Muniz Bandeira Duarte. Orchestrating virtualized network functions. *IEEE Transactions on Network and Service Management*, 13(4):725–739, 2016.

[10] Mahdi Dolati, Seyedeh Bahereh Hassanpour, Majid Ghaderi, and Ahmad Khonsari. Deepvine: Virtual network embedding with deep reinforcement learning. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 879–885, 2019.

[11] Jose Jurandir Alves Esteves, Amina Boubendir, Fabrice Guillemin, and Pierre Sens. A heuristically assisted deep reinforcement learning approach for network slice placement, 2021.

[12] José Jurandir Alves Esteves, Amina Boubendir, Fabrice Guillemin, and Pierre Sens. Location-based data model for optimized network slice placement. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pages 404–412, 2020.

[13] José Jurandir Alves Esteves, Amina Boubendir, Fabrice Guillemin, and Pierre Sens. Drl-based slice placement under realistic network load conditions, 2021.

[14] ETSI. Network functions virtualisation (nfv); management and orchestration, european telecommunication standard institute (etsi), group specification (gs) nfv-man 001, 12 2014, version 1.1.1., 2014.

[15] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann de Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys Tutorials*, 15(4):1888–1906, 2013.

[16] Xenofon Foukas, Georgios Patounas, Ahmed Elmokashfi, and Mahesh K. Marina. Network slicing in 5g: Survey and challenges. *IEEE Communications Magazine*, 55(5):94–100, 2017.

[17] P. Gawłowicz and A. Zubow. Ns-3 meets openai gym: The playground for machine learning in networking research. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '19, page 113–120, New York, NY, USA, 2019. Association for Computing Machinery.

[18] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in nfv: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[19] Hassan Hawilo, Manar Jammal, and Abdallah Shami. Network function virtualization-aware orchestrator for service function chaining placement in the cloud. *IEEE Journal on Selected Areas in Communications*, 37(3):643–655, 2019.

[20] Johanna Andrea Hurtado Sánchez, Katherine Casilimas, and Oscar Mauricio Caicedo Rendon. Deep reinforcement learning for resource management on network slicing: A survey. *Sensors*, 22(8), 2022.

[21] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.

[22] Defang Li, Peilin Hong, Kaiping Xue, and jianing Pei. Virtual network function placement considering resource optimization and sfc requests in cloud datacenter. *IEEE Transactions on Parallel and Distributed Systems*, 29(7):1664–1677, 2018.

[23] Jiaqiang Liu, Yong Li, Ying Zhang, Li Su, and Depeng Jin. Improve service chaining performance with optimized middlebox placement. *IEEE Transactions on Services Computing*, 10(4):560–573, 2017.

[24] Marcelo Caggiani Luizelli, Weverton Luis da Costa Cordeiro, Luciana S. Buriol, and Luciano Paschoal Gaspary. A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining. *Computer Communications*, 102:67–77, 2017.

[25] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 2016.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[27] Jianing Pei, Peilin Hong, Miao Pan, Jiangqing Liu, and Jingsong Zhou. Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks. *IEEE Journal on Selected Areas in Communications*, 38(2):263–278, 2020.

[28] Jianing Pei, Peilin Hong, Kaiping Xue, and Defang Li. Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system. *IEEE Transactions on Parallel and Distributed Systems*, 30(10):2179–2192, 2019.

[29] Chuan Pham, Nguyen H. Tran, Shaolei Ren, Walid Saad, and Choong Seon Hong. Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach. *IEEE Transactions on Services Computing*, 13(1):172–185, 2020.

[30] Tran Anh Quang Pham, Jean-Michel Sanner, Cédric Morin, and Yassine Hadjadj-Aoul. Virtual network function–forwarding graph embedding: A genetic algorithm approach. *International Journal of Communication Systems*, 33(10):e4098, 2020. e4098 0.1002/dac.4098.

[31] Pham Tran Anh Quang, Yassine Hadjadj-Aoul, and Abdelkader Outtagarts. A deep reinforcement learning approach for vnf forwarding graph embedding. *IEEE Transactions on Network and Service Management*, 16(4):1318–1331, 2019.

[32] Anouar Rkhami, Yassine Hadjadj-Aoul, and Abdelkader Outtagarts. Learn to improve: A novel deep reinforcement learning approach for beyond 5g network slicing. In *2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC)*, pages 1–6, 2021.

[33] Matthias Rost and Stefan Schmid. Virtual network embedding approximations: Leveraging randomized rounding, 2018.

[34] Hong Tang, Danny Zhou, and Duan Chen. Dynamic network function instance scaling based on traffic forecasting and vnf placement in operator data centers. *IEEE Transactions on Parallel and Distributed Systems*, 30(3):530–543, 2019.

[35] Khuong Tran, Maxwell Standen, Junae Kim, David Bowman, Toby Richer, Ashlesha Akella, and Chin-Teng Lin. Cascaded reinforcement learning agents for large action spaces in autonomous penetration testing. *Applied Sciences*, 12(21), 2022.

[36] Tom Van de Wiele, David Warde-Farley, Andriy Mnih, and Volodymyr Mnih. Q-learning in enormous action spaces via amortized approximate maximization, 2020.

[37] H. van Hasselt, A.r Guez, and D. Silver. Deep reinforcement learning with double q-learning, 2015.

[38] Shalitha Wijethilaka and Madhusanka Liyanage. Survey on network slicing for internet of things realization in 5g networks. *IEEE Communications Surveys Tutorials*, 23(2):957–994, 2021.

[39] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. 1992.

[40] Yikai Xiao, Qixia Zhang, Fangming Liu, Jia Wang, Miao Zhao, Zhongxing Zhang, and Jiaxing Zhang. Nfvdeep: Adaptive online service function chain deployment with deep reinforcement learning. In *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*, pages 1–10, 2019.

[41] Zhongxia Yan, Jingguo Ge, Yulei Wu, Liangxiong Li, and Tong Li. Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks. *IEEE Journal on Selected Areas in Communications*, 38(6):1040–1057, 2020.

[42] Haipeng Yao, Xu Chen, Maozhen Li, Peiying Zhang, and Luyao Wang. A novel reinforcement learning algorithm for virtual network embedding. *Neurocomputing*, 284:1–9, 2018.

[43] Haipeng Yao, Bo Zhang, Peiying Zhang, Sheng Wu, Chunxiao Jiang, and Song Guo. Rdam: A reinforcement learning based dynamic attribute matrix representation for virtual network embedding. *IEEE Transactions on Emerging Topics in Computing*, 9(2):901–914, 2021.

[44] Dong Yin, Zhe Zhao, Yinglong Dai, and Han Long. A novel multi-agent deep reinforcement learning approach. *Journal of Physics: Conference Series*, 1757(1):012097, jan 2021.

[45] Qixia Zhang, Yikai Xiao, Fangming Liu, John C.S. Lui, Jian Guo, and Tao Wang. Joint optimization of chain placement and request scheduling for network function virtualization. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 731–741, 2017.

Abdel Kader Chabi Sika Boni received the Engineering degree in Networks and Telecommunications from ENSA Khouribga, Morocco, in 2020 and the Master II ILoRD degree in software engineering of networks and distributed systems from INSA Toulouse, France in 2021. Since 2021, he is PhD candidate at the French National Center for Scientific Research (CNRS). His PhD topic is about providing intelligent architectures and algorithms including Machine Learning and Deep Reinforcement Learning to support autonomous IoT systems.

Hassan Hassan received the Engineering degree in Computer Science from ENSERG (INP Grenoble) in septembre 1996 and M.S. (DEA) degree in Networks and Telecommunication from ENSEEIHT (INP Toulouse) in June 2003. He obtained his Ph.D. degree in Computer Science from UPS, University Paul Sabatier Toulouse III, in December 2006. He was from January 2007 to January 2008, Postodctoral researcher at LAAS-CNRS, and from February 2008 et December 2008 R&D engineer at Ginkgo Networks, a startup of Paris 6 University. He is since January 2009 Research Engineer, a full-time position, at the French National Center for Scientific Research (CNRS). His research interests include Computer Networks and Deep Reinforcement Learning applications in IoT systems.

Khalil Drira received the Master degree in computer science from INP, Toulouse, in 1988, and the Ph.D. and HDR degrees in computer science from Université Paul Sabatier Toulouse in 1992 and 2005, respectively. Since 1992, he assumes a full-time research position in CNRS, France. His research interests include cooperative network IoT services, platforms and applications. His research activity addresses topics in this field focusing on Software architectures and communication services. He continues to be involved in national and international conferences and journals. He serves as a member of the program journals in the fields of software architecture as well as IoT and Internet networks. He has also been an Editor of several proceedings, books, and journals.