A Heuristically Assisted Deep Reinforcement Learning Approach for Network Slice Placement

Jose Jurandir Alves Esteves, Amina Boubendir, Fabrice Guillemin, Pierre Sens

A Heuristically Assisted Deep Reinforcement Learning Approach for Network Slice Placement

José Jurandir Alves Esteves, *Student Member, IEEE,* Amina Boubendir, *Member, IEEE,* Fabrice Guillemin, and Pierre Sens

***Abstract*—Network Slice placement with the problem of allocation of resources from a virtualized substrate network is an optimization problem which can be formulated as a multi-objective Integer Linear Programming (ILP) problem. However, to cope with the complexity of such a continuous task and seeking for optimality and automation, the use of Machine Learning (ML) techniques appear as a promising approach. We introduce a hybrid placement solution based on Deep Reinforcement Learning (DRL) and a dedicated optimization heuristic based on the "Power of Two Choices" principle. The DRL algorithm uses the so-called Asynchronous Advantage Actor Critic (A3C) algorithm for fast learning, and Graph Convolutional Networks (GCN) to automate feature extraction from the physical substrate network. The proposed Heuristically-Assisted DRL (HA-DRL) allows for the acceleration of the learning process and substantial gain in resource usage when compared against other state-of-the-art approaches, as evidenced by evaluation results.**

***Index Terms*—Network Slicing, Optimization, Automation, Deep Reinforcement Learning, Placement, Large Scale.**

I. Introduction

NETWORK Slicing is a major stake in 5G networks, which is notably enabled by virtualization techniques applied to Network Functions, a.k.a. Network Function Virtualization (NFV), and by Software Defined Network (SDN) techniques [1]. Thanks to these technical enablers now widely used in the telecom industry, a telecommunications network becomes a programmable platform, which can offer virtual networks enriched by Virtual Network Functions (VNFs) and IT resources, and can be tailored to the specific needs of certain customers (e.g., companies) or vertical markets (automotive, e-health, etc.). These augmented virtual networks give rise to the concept of Network Slicing, which is specified by standardization bodies [2], [3]. In this paper, we shall consider a Network Slice as a chain of VNFs interconnected by a transport network and with networking (bandwidth) and computing (CPU, RAM) requirements.

Network Slice Placement problem is close to the Virtual Network Embedding (VNE) problem but there are at least two outstanding differences: 1) VNE consists of placing virtual nodes onto physical ones subject to bandwidth constraints, whereas Network Slice Placement takes into account several constraints such as RAM, CPU, disk, bandwidth and latency; 2) VNE consists of a 1:1 node mapping, whereas Network Slice Placement can place multiple "virtual nodes" (namely, VNFs) on the same physical node. There is a huge amount of literature on VNF placement [4]–[8] and associated optimization problems are usually modeled as Integer Linear Programming (ILP) problems, which turn out to be $\mathcal{NP}$-hard [9] with very long convergence time.

Heuristics have then been developed, see [10] for an extensive list of heuristics for slice placement and in particular the one based on the "Power of two Choices" principle (for short, P2C), which gives satisfactory results, both in terms of convergence time and slice acceptance ratio. From an operational perspective, heuristic approaches are more suitable than ILP as they yield faster placement results. This is very important for operational networks because traffic conditions are fluctuating and placement response time is an important performance indicator in the customer relationship. The drawback of heuristic approaches is that they give sub-optimal solutions. To address this issue, Machine Learning (ML) offer a corpus of methods able to overcome the convergence issues of ILPs while being more accurate than heuristics. Deep Reinforcement Learning (DRL) has recently been used in the context of VNE and VNF-FGE [11]–[13].

A DRL agent is theoretically capable of learning an optimal decision policy only based on its own experience; this property eliminates the need for an accurate training data set that may not be available. However, from a practical point of view, ensuring that the DRL agent converges to an optimal policy is a challenge since the agent acts as a self-controlled black box. In addition, there are a large number of hyper-parameters to fine-tune in order to ensure an adequate equilibrium between exploring solutions and exploiting the knowledge acquired via training. While there are techniques to improve the efficiency of the solution exploration process (e.g., ϵ-greedy, entropy regularization), their use may also lead to situations of instability, where the algorithm may diverge from the optimal point.

To overcome this unsuitable behavior of DRL agents, we introduce in the present paper the concept of Heuristically Assisted DRL (HA-DRL) to accelerate and stabilize the convergence of DRL techniques when applied to the Network Slice Placement. The proposed contributions are twofold: i) We propose a DRL algorithm combining Advantage Actor Critic and a Graph Convolutional Network (GCN) to solve Network Slice Placement optimization problem [11]; ii) We reinforce the DRL learning process by using the P2C based heuristic [10] to control the DRL convergence.

The organization of this paper is as follows: In Section II, we review the related work on slice placement and more generally

J.J. Alves Esteves is with Orange Labs, 92320 Chatilon, France and also with LIP6 – Inria, Sorbonne Univ., 75005 Paris, France (e-mail: josejurandir.alvesesteves@orange.com).

A. Boubendir and F. Guillemin are with Orange Labs, 92320 Chatillon, France (e-mail: firstname.name@orange.com).

P. Sens is with LIP6 – Inria, Sorbonne Univ., CNRS, 75005 Paris, France (e-mail: pierre.sens@lip6.fr).

on VNE by paying special attention to ML techniques. In Section III, we describe the Network Slice Placement problem and introduce the various elements of the model. The multi-objective optimization problem of Network Slice Placement is formulated in Section IV. A DRL approach to solving the multi-objective optimization problem is described in Section V. The control of the DRL convergence by using the P2C heuristic is introduced in Section VI. The experiments and evaluation results are presented in Section VII, while conclusions and perspectives are presented in Section VIII.

II. Related Work Analysis

We review, in this section, recent studies on the network slice placement problem. We consider comprehensive surveys such as [4]–[8], [11] and analyse existing works along two lines: i) ML-based approaches for slice placement optimization (Section II-A), and ii) hybrid approaches combining both heuristics and ML for slice placement (Section II-B). There are many heuristics for the placement of chains of VNFs [8], [10], [14]–[16]. In the following, we focus on ML methods.

A. On ML Approaches for Slice Placement Optimization

The most relevant ML-based approaches for network slice placement optimization are DRL-based solutions. We analyze: i) the RL setup elements of each solution (i.e., state, action, reward); and ii) the modeling aspects of DRL (i.e., type of convolutions, training algorithms).

1) RL Setup Elements:

a) State representation: The state representation is almost similar in all analyzed solutions. In [11]–[13], [17], only resource-related features are used to represent the state: i) the number of resources available on physical nodes (CPU and RAM) and links (bandwidth) and ii) the number of these resources required by the VNFs and Virtual Links (VLs) of the network slice to be placed. Latency-related features are also considered in [18]–[21]. However, adding such features to the state brings additional complexity to their models. In particular, the model in [18] requires additional chaining information and performance indexes for the VNFs. In [21], the model requires an explicit VL representation. We adopt a resource oriented state representation. A simpler model considering latency remains an open issue.

b) Action representation: In [11]–[13], [17], [19], the problem is modeled by considering finite action spaces. The action in [11], [13], [19] is the index of the physical node in which to place a specific VNF of the slice. This representation of the action requires breaking the process of placing one slice in a sequence of placement actions and has the advantage of reducing the size of the action space to the number of physical nodes. In [17], the action is represented as a binary variable used to modify or accept the placement of a specific VNF of the slice previously computed by a heuristic. The action in [12] is either to mark a VNF to be placed on a physical node or to place a VNF on the previously marked physical node. The placement of the entire slice is then iteratively constructed and the algorithm stops when all the VNFs of the slice are placed or when a maximal number of iterations is reached. In [18], [20], [21], infinite action spaces are adopted, i.e., the actions are real numbers. In [18] the action is defined as an instruction to a scheduler to adjust the resources allocated to a VNF by a certain percentage whereas in [21], the action is the placement price returned to the client.

In [20], the action is represented by two sets of weights: i) the placement priority of each VNF in the slice on each physical node and ii) the placement priority of each VL in the slice on each physical link. To reduce complexity, we adopt a finite action space represented as in [11].

c) Reward function: Most of analyzed solutions adopt placement cost or revenue in their reward function [11]–[13], [17], [19], [21]. Cost and revenue are mainly calculated in terms of resource consumption. Some solutions adopt a reward associated to acceptance ratio [11], [19], [20]. A penalty for SLA violations is considered only in [18]. The most complete reward function is the one proposed in [11] combining acceptance ratio and placement cost with load balance. We leverage on [11] reward function criteria but propose a formulation that reduces bias during training.

2) Modeling Aspects of DRL:

a) Use of convolutions: Most of the analyzed papers employ a Convolutional Neural Network (CNN) to perform automatic feature extraction except the approach proposed in [19], which uses regular Deep Neural Network (DNN). Classical CNNs are limited by the fact that they only work on Euclidean objects (e.g., images, grids). DRL algorithms for network slice placement built upon this technique have reduced real-life applicability because they can not work on unstructured networks. To overcome this issue, Graph Convolutional Networks (GCN) have been used in [22], [23] in order to work on arbitrarily structured graphs. GCNs are used in [11] to automatically extract features from the physical network when solving a VNE problem. A type of GCN adapted to hetero-graphs called Relational GCN is used by [17] to automatically learn how to improve the quality of an initial placement computed by a heuristic. We also use the power of GCN in our proposal.

b) Training algorithms: All the analyzed solutions use different types of Policy Gradient (PG) training algorithms, except the solution in [12], which uses the well-known value-based algorithm "Deep Q-learning". We rely on the Asynchronous Advantage Actor Critic (A3C) approach introduced in [24] and also used by [11] owing to its robustness and improved performance.

B. On Hybrid ML Approaches for Slice Placement

Although ML-based techniques, such as DRL, have been shown to be robust when applied to solving various problems, these approaches also exhibit some drawbacks as they: i) are difficult to run due to a variety of hyper parameters to tune; ii) have convergence times difficult to control since they act as self-controlled black boxes; and iii) take too much time to start finding good solutions because their performance depends on the exploration of a huge number of states and actions. This makes pure DRL approaches uncertain in real online optimization scenarios. Researchers then focused on developing

hybrid methods that combine ML-based optimization with safer but less scalable techniques such as heuristics. The concept of Heuristically Accelerated Reinforcement Learning (HARL) is introduced in [25] as a way to solve RL problems with the explicit use of a heuristic function to influence the choice of actions by a learning agent.

As shown in [26], HA- versions of well-known RL algorithms such as Q-learning, SARSA(λ) and TD(λ) have lower convergence times than those of their classical versions when applied to a variety of RL problems. Although HARL has shown relevant results in different fields of applications such as video streaming [27], multi-agent systems [28] and robotics [29], to the best of our knowledge, it has only been used in [30] in the networking domain, namely, autonomous spectrum management. We build on HARL to propose HA-DRL and we apply this algorithm in the present paper to slice placement. To the best of our knowledge, this is the first work on HA-DRL. We rely on the formulation of heuristic function proposed in [25]. We adapt this method to DRL and combine it with an efficient placement heuristic proposed in [10] to strengthen and accelerate the DRL learning process. This improves performance and safety when compared with state-of-the-art DRL placement approaches [11].

Other hybrid approaches for placement optimization combining DRL and heuristics were proposed recently. In [20], a HFA-DDPG algorithm combining Deep Deterministic Policy Gradient (DDPG) with a Heuristic Fitting Algorithm (HFA) to solve the VNF-FGE problem is proposed. Evaluation results show that HFA-DDPG converges much faster than DDPG. However, the approach reveals to be more complex than needed due to the infinite action space formulation that adds an unnecessary complexity, thus reducing the applicability of the algorithm. Our approach, with a bounded action space, limits the complexity and seems to be more appropriated for large-scale scenarios. In [17], the REINFORCE algorithm is used to learn and to improve a VNE solution previously computed by a heuristic. In spite of the significant improvement of the initial heuristic solution obtained by this approach, its effectiveness is highly dependent on the quality of the initial solution provided by the heuristic. In addition, both algorithms proposed in [20] and [17] are based on a DRL agent that relies heavily on the heuristic algorithm to compute the placement decisions. In our case, we only use the heuristic to improve the DRL exploration process. As a result, our DRL agent can scale and be used after training even without the support of the heuristic.

III. Network Slice Placement Optimization Problem Modeling

We present, in this section, the two main elements of the Network Slice Placement Optimization problem: the Physical Substrate Network in Section III-A, and the Network Slice Placement Requests in Section III-B.

A. Physical Substrate Network Modeling

The Physical Substrate Network (for short, PSN) is composed of the infrastructure resources, namely servers with IT resources (CPU, RAM, disk, etc.) and the transport network, in particular VLs interconnecting servers . As illustrated in Figure 1, the PSN is divided into three components: the Virtualized Infrastructure (VI) corresponding to IT resources, the Access Network (AN), and the Transport Network (TN).

1) The Virtualized Infrastructure (VI): This component is the set of Data Centers (DCs) interconnected by network elements (switches and routers) and either distributed in Points of Presence (PoP) or centralized (e.g., in a big cloud platform).

Following the reference architecture presented in [31] and describing the possible of a network operator, we define three types of DCs with different capacities: Edge Data Centers (EDCs) as local DCs with small resources capacities, Core Data Centers (CDCs) as regional DCs with medium resource capacities, and Central Cloud Platforms (CCPs) as national DCs with big resource capacities.

2) The Access Network (AN): This set represents User Access Points (UAPs) (Wi-Fi APs, antennas of cellular networks, etc.) and Access Links. Users access slices via one UAP, which may change during the life time of a communication by a user (e.g., because of mobility).

3) The Transport Network (TN): This is the set of routers and transmission links needed to interconnect the different DCs and the UAPs.

The complete PSN is modeled as a weighted undirected graph $G_s = (N, L)$ where N is the set of physical nodes in the PSN, and $L \subset \{(a, b) \in N \times N \wedge a \neq b\}$ refers to a set of substrate links. Each node has a type in the set {UAP, router, switch, server}. The available CPU and RAM capacities on each node are defined respectively as $cap_n^{cpu} \in \mathbb{R}$, $cap_n^{ram} \in \mathbb{R}$ for all $n \in N$. The available bandwidth on the links are defined as $cap_{(a,b)}^{bw} \in \mathbb{R}, \forall (a, b) \in L$.

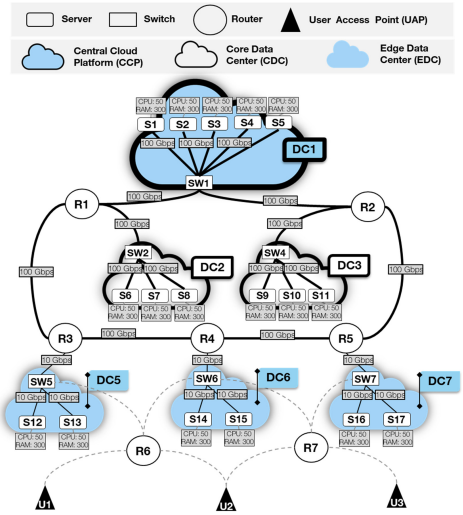


Fig. 1. Physical Substrate Network.

B. Network Slice Placement Requests Modeling

We consider each slice as a finite number of VNFs to be placed and chained on the PSN. VNFs are batched and introduced in the network as Network Slice Placement Requests (NSPRs). The NSPRs are similarly represented as a weighted undirected graph $G_v = (V, E)$, where V is the set of VNFs in the NSPR, and $E \subset \{(\bar{a}, \bar{b}) \in V \times V \wedge \bar{a} \neq \bar{b}\}$ is a set of VLs.

The CPU and RAM requirements of each VNF of an NSPR are defined as $req_v^{cpu} \in \mathbb{R}$ and $req_v^{ram} \in \mathbb{R}$ for all $v \in V$, respectively. The bandwidth required by each VL in an NSPR is given by $req^{bw}_{(\bar{a},\bar{b})} \in \mathbb{R}$ for all $(\bar{a},\bar{b}) \in E$. The end-to-end (E2E) latency is denoted by δ. An example of NPSR is represented in Figure 2.

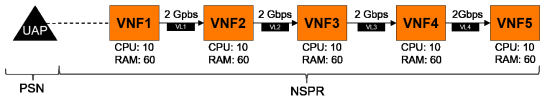


Fig. 2. Example of an NSPR.

IV. Multi-objective Optimization Problem for Network Slice Placement

In this section, we formulate the multi-objective optimization problem for network slice placement. The parameters related to PSN and NSPR are described in Tables I and II, respectively.

A. Network Slice Placement Optimization Problem Statement

- *Given:* an NSPR graph $G_v = (V, E)$ and a PSN graph $G_s = (N, L)$,
- *Find:* a mapping $G_v \rightarrow \bar{G}_s = (\bar{N}, \bar{L})$, $\bar{N} \subset N$, $\bar{L} \subset L$,
- *Subject to:* the VNF CPU requirements $req_v^{cpu}, \forall v \in V$, the VNF RAM requirements $req_v^{ram}, \forall v \in V$, the VLs bandwidth requirements $req^{bw}_{(\bar{a},\bar{b})}, \forall (\bar{a},\bar{b}) \in E$, the server CPU available capacity $cap_s^{cpu}, \forall s \in S$, the server RAM available capacity $cap_s^{ram}, \forall s \in S$, the physical link bandwidth available capacity $cap^{bw}_{(a,b)}, \forall (a,b) \in L$.
- *Objective:* maximize the network slice placement request acceptance ratio, minimize the total resource consumption, meet E2E latency requirement, and maximize load balancing.

TABLE I
PSN parameters

Parameter	*Description*
$G_s = (N, L)$	PSN graph
N	Network nodes
$S \subset N$	Set of servers
$L = \{(a,b) \in N \times N \wedge a \neq b\}$	Set of physical links
$cap^{bw}_{(a,b)} \in \mathbb{R}, \forall (a,b) \in L$	Bandwidth capacity of link (a,b)
$cap_s^{cpu} \in \mathbb{R}, \forall s \in S$	available CPU capacity on server s
$M_s^{cpu} \in \mathbb{R}, \forall s \in S$	maximum CPU capacity of server s
$cap_s^{ram} \in \mathbb{R}, \forall s \in S$	available RAM capacity on server s
$M_s^{ram} \in \mathbb{R}, \forall s \in S$	maximum RAM capacity of server s
$M_s^{bw} \in \mathbb{R}, \forall s \in S$	maximum outgoing bandwidth from s
$\delta_{(a,b)} \in \mathbb{R}, \forall (a,b) \in L$	latency induced by physical link (a,b)

B. Problem Formulation

To formulate the optimization problem, we introduce the decision variables and we identify the constraints, which have to be satisfied by the placement algorithm.

TABLE II
NSPR parameters

Parameter	*Description*
$G_v = (V, E)$	NSPR graph
V	Set of VNFs of the NSPR
$E = \{(\bar{a},\bar{b}) \in N \times N \wedge \bar{a} \neq \bar{b}\}$	Set of VLs of the NSPR
$v_{root} \in V$	Root VNF of the NSPR
$req_v^{cpu} \in \mathbb{R}$	CPU requirement of VNF v
$req_v^{ram} \in \mathbb{R}$	RAM requirement of VNF v
$req^{bw}_{(\bar{a},\bar{b})} \in \mathbb{R}$	Bandwidth requirement of VL $(\bar{a},\bar{b})$
$req^{\delta}_{(\bar{a},\bar{b})} \in \mathbb{R}$	Latency requirement of VL $(\bar{a},\bar{b})$
$\delta \in \mathbb{R}$	E2E latency requirement of the NSPR
$\alpha_{max} \in \mathbb{R}$	Access latency requirement of the NSPR
$\alpha_s \in \mathbb{R}$	Access latency to server s

1) Decision Variables: We use the two following binary decision variables:

- $x_s^v \in \{0,1\}$ for $v \in V$ and $s \in S$ is equal to 1 if the VNF v is placed onto server s and 0 otherwise,
- $y^{(\bar{a},\bar{b})}_{(a,b)} \in \{0,1\}$ for $(\bar{a},\bar{b}) \in E$ and $(a,b) \in L$ is equal to 1 if the virtual link $(\bar{a},\bar{b})$ is mapped onto physical link (a,b) and 0 otherwise.

2) Problem Constraints:

a) VNF placement: The following constraint ensures that: i) all VNFs of the NSPR must be placed; and ii) each VNF must be placed in only one server:

$$\forall v \in V, \quad \sum_{s \in S} x_s^v = 1. \tag{1}$$

b) Network resource capacities constraints: Eqs. (2) and (3) below ensure that the resource capacities of each server (for CPU and RAM, respectively) are not exceeded; the subsequent Eq. (4) guarantees that the bandwidth capacity of each physical link is not exceeded:

$$\forall s \in S, \quad \sum_{v \in V} req_v^{cpu} x_s^v \leq cap_s^{cpu}, \tag{2}$$

$$\forall s \in S, \quad \sum_{v \in V} req_v^{ram} x_s^v \leq cap_s^{ram}, \tag{3}$$

$$\forall (a,b) \in L, \quad \sum_{(\bar{a},\bar{b}) \in E} req^{bw}_{(\bar{a},\bar{b})} y^{(\bar{a},\bar{b})}_{(a,b)} \leq cap^{bw}_{(a,b)}. \tag{4}$$

c) Eligible physical path calculation: We use flow conservation constraints [32] formulated by Eq. (5), (6), and (7) for the definition of the eligible physical paths in which to map every VL $(\bar{a},\bar{b}) \in E$ as : for all $a \in S$ and $(\bar{a},\bar{b}) \in E$,

$$\sum_{\substack{b \in N: \\ (a,b) \in L}} y^{(\bar{a},\bar{b})}_{(a,b)} - \sum_{\substack{b \in N: \\ (b,a) \in L}} y^{(\bar{a},\bar{b})}_{(b,a)} = x_a^{\bar{b}} - x_a^{\bar{a}}, \tag{5}$$

and

$$\forall a \in N \setminus S, \forall (\bar{a},\bar{b}) \in E, \quad \sum_{\substack{b \in N: \\ (a,b) \in L}} y^{(\bar{a},\bar{b})}_{(a,b)} - \sum_{\substack{b \in N \\ (b,a) \in L}} y^{(\bar{a},\bar{b})}_{(b,a)} = 0, \tag{6}$$

$$\forall (\bar{a},\bar{b}) \in E, \forall (a,b) \in L, \; y^{(\bar{a},\bar{b})}_{(a,b)} + y^{(\bar{a},\bar{b})}_{(b,a)} \leq 1. \tag{7}$$

The left-hand sides of Eq. (5) and (6) compute the difference between the activated links outgoing and incoming from/to

each node $a \in N$. One computation is done for each VL $(\bar{a}, \bar{b}) \in E$. If $a \in S$, the right-hand side of Eq. (5) ensures that the computed difference must be equal to $x_a^{\bar{b}} - x_a^{\bar{a}}$. That is: 0 if server a is used to place both VNFs $\bar{a}$ and $\bar{b}$ or if its not used to place neither of them; -1 if only the source VNF $\bar{a}$ is placed on a; 1 if only the destination VNF $\bar{b}$ is placed on a. If $a \in N \setminus S$, Eq. (6) ensures that this difference will always be 0. Eq. (7) imposes that each link must be used only in one direction when mapping a specific VL.

d) Network Slice Latency Requirements Constraints: The equations below ensure the respect of latency requirements for each VL and the access together with the E2E latency:

$$\forall (\bar{a}, \bar{b}) \in E, \sum_{(a,b) \in L} \delta_{(a,b)} y_{(a,b)}^{(\bar{a},\bar{b})} \leq req_{(\bar{a},\bar{b})}^{\delta}, \tag{8}$$

$$\sum_{s \in S} \alpha_s^r x_s^{v_{root}} \leq \alpha_{max}, \tag{9}$$

$$\sum_{s \in S} \alpha_s x_s^{v_{root}} + \sum_{(a,b) \in L} \sum_{(\bar{a},\bar{b}) \in c} \delta_{(a,b)} y_{(a,b)}^{(\bar{a},\bar{b})} \leq \delta. \tag{10}$$

The above constraints have been considered in [10]; for the sake of simplicity, they will be not be included in the DRL algorithm proposed in the present paper.

3) Objective Function: We consider three objective functions: a) minimization of resource consumption; b) maximization of slice acceptance; and c) maximization of node load balance.

a) Minimization of resource consumption: The placement of all VNFs of an NSPR is mandatory otherwise the solution would violate Eq. (1). The optimization objective is then to minimize the bandwidth consumption given by

$$\min_{x,y} \sum_{(\bar{a},\bar{b}) \in E} \sum_{(a,b) \in L} y_{(a,b)}^{(\bar{a},\bar{b})} req_{(\bar{a},\bar{b})}^{bw}. \tag{11}$$

This objective can be seen as finding the shortest path to map each VL $(a, b) \in E$ of an NSPR. It is worth noting that Eqs. (5), (6), and (7) control the value of y in such a way that $y_{(a,b)}^{(\bar{a},\bar{b})} = 1$ for all links $(a, b) \in L$ included in the path used to map VL $(\bar{a}, \bar{b}) \in E$.

b) Maximization of slice requests acceptance: The maximization of slice request acceptance objective function is given by Equation (12). Auxiliary variable $z \in \{0, 1\}$ representing whether the NSPR is accepted ($z = 1$) or not ($z = 0$) is used in this case.

$$\max_{x,y} z. \tag{12}$$

In this case, Eq. (1) is relaxed and additional Eqs. (13) and (14) below need to be inserted in the model:

$$\forall v \in V, z \leq \sum_{s \in S} x_s^v, \tag{13}$$

$$z \geq \sum_{s \in S} \sum_{v \in V} x_s^v - |V - 1|. \tag{14}$$

Eq. (13) ensures that $z = 0$ if there is a VNF $v \in V$ that cannot be placed (i.e., there is a $v \in V$ such that $\sum_{s \in S} x_s^v = 0$). Eq. (14) ensures that $z = 1$ if and only if all VNFs $v \in V$ are placed (i.e., $\sum_{s \in S} \sum_{v \in V} x_s^v = |V|$).

c) Maximization of node load balance: Eq. (15) gives the maximization of node load balance objective function.

$$\max_{x,y} \sum_{s \in S} \sum_{v \in V} x_s^v \left(\frac{cap_s^{cpu}}{M_s^{cpu}} + \frac{cap_s^{ram}}{M_s^{ram}} \right). \tag{15}$$

C. P2C Heuristic Principles

We have proposed in [10] a heuristic to solve the ILP problems introduced above. This heuristic is based on the P2C principle [33], which states in the present context that considering 2 possible data centers chosen "randomly" instead of only 1 brings exponential improvement of the solution quality.

The proposed heuristic is a greedy algorithm such that for each VNF $\bar{b} \in V$: i) randomly selects 2 candidate servers $s_1, s_2 \in S$; ii) evaluates the resource consumption when placing $\bar{b}$ in s_1 and s_2 and place $\bar{b}$ on the best server; iii) maps the VLs $(\bar{a}, \bar{b}) \in E$ associated to $\bar{b}$. This heuristic contains the limitations of all heuristic approaches: the lack of flexibility due to manual feature design, the difficulties to handle multiple optimization criteria, and the sub-optimality of the provided solutions. But it also yields low execution time and good load balancing; it was shown in [10] that the heuristic outperforms two ILP based algorithms. We elaborate in the present paper on these proprieties of the heuristic to propose our HA-DRL approach described in the following sections.

V. DRL for Network Slice Placement Optimization

The ILP introduced in Section IV is too difficult to solve in small configurations and all the more in near real-time under operational conditions. As an alternative, we have used DRL techniques in toy scenarios with networks with a small number of links and nodes. It turns out that DRL methods are very efficient and give results very close (up to a few percents) to those obtained when solving the corresponding ILP, which is manageable in this case. Motivated by this observation, we develop in this section a DRL approach for large scale networks, which will be augmented by a heuristic in the subsequent section.

A. DRL Framework for Network Slice Placement

Figure 3 presents the proposed DRL framework. The state contains the features of the PSN and NSPR to be placed. A valid action is, for a given NSPR graph $G_v = (V, E)$, to find sub graph $\overline{G}_s$ of the PSN graph G_s (i.e., $\overline{G}_s \subset G_s = (N, L)$) to place the NSPR that does not violate the problem constraints described in Section IV-B. The reward evaluates how good is the computed action with respect to the optimization objectives described in Section IV-B3. DNNs are trained to: i) calculate optimal actions for each state (i.e., placements with maximal rewards), ii) calculate the State-value function used in the learning process.

B. Policy Enforcement

Let $\mathcal{A}$ be the set of all possible actions that the DRL agent can take and $\mathcal{S}$ the set of all states that it can visit. We adopt a sequential placement strategy in which, at each time step,

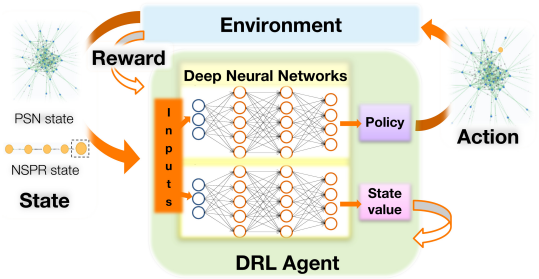


Fig. 3. DRL framework for Network Slice Placement Optimization

we choose a node $n \in N$ where to place a specific VNF $v \in \{1, ..., |V|\}$. The VNFs are placed in ascending order, which means that the placement starts with the VNF $v = 1$ and ends for the VNF $v = |V|$.

We break then the process of placing one NSPR graph $G_v = (V, E)$ into a sequence of $|V|$ actions, one for each $v \in V$, instead of considering the one shot placement of G_v. The latter strategy would require the definition of the action as a subgraph of the PSN graph $G_s = (N, L)$ what would imply $|\mathcal{A}| = |SG|$, where SG is the set of all sub graphs of G_s, that grows exponentially with size of G_s. Note that with the sequential placement strategy $\mathcal{A} = N$, thus $|\mathcal{A}| \ll |SG|$. At each time step t, the DRL agent focuses on the placement of exactly one VNF $v \in V$ of the NSPR. The mapping of the virtual links associated to the VNF v to a physical path in the PSN is done by the shortest path algorithm.

Given a state σ_t, the DRL agent uses the policy to select an action $a_t \in \mathcal{A}$ corresponding to the PSN node for placing VNF v. The policy probabilities are calculated using the Softmax distribution defined by

$$\pi_\theta(a_t = a|\sigma_t) = \frac{e^{Z_\theta(\sigma_t, a),}}{\sum_{b \in N} e^{Z_\theta(\sigma_t, b)}}, \tag{16}$$

where the function $Z_\theta : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ maps each state and action to a real value. In our formulation this function is calculated by a DNN described in Section VI-A1. The notation π_θ is used to indicate that policy depends on Z_θ. The control parameter θ represents the weights in the DNN.

C. State Representation

The state contains a compact representation of the two main elements of our model.

1) PSN State: The PSN State is the real-time representation of the PSN status, which is given by four characteristics. Three of these characteristics are related to resources (see Table I for the notation) and are defined by the following sets: $cap^{cpu} = \{cap_n^{cpu} : n \in N\}$, $cap^{ram} = \{cap_n^{ram} : n \in N\}$ and $cap^{bw} = \{cap_n^{bw} = \sum_{(n,b) \in L} cap_{(n,b)}^{bw} : n \in N\}$. We consider in addition one characteristic related to placement in order to record the actions taken by the DRL agent during the placement of the current NSPR described by the vector $\chi = \{\chi_n \in \{0, .., |V|\} : n \in N\}$, where χ_n corresponds to the number of VNFs of the current NSPR placed on node n.

2) NSPR State: The NSPR State represents a view of the current placement. It is composed of four characteristics. Three resource requirement characteristics (see Table II for the notation) associated with the current VNF v to be placed: req_v^{cpu}, req_v^{ram} and $req_v^{bw} = \sum_{(v,\bar{b}) \in E} req_{(v,\bar{b})}^{bw}$. We also consider the number $m_v = |V| - v + 1$ used to track the number of VNFs still to be placed at each time step.

D. Reward Function

The proposed Reward function contains one reward value for each optimization objective introduced in Section IV-B as detailed in the following sections.

1) Acceptance Reward: Each Action may lead to a successful or unsuccessful placement depending on whether it respects the problem constraints for the candidate VNF v and its associated VLs or not. We then define the Acceptance Reward value due to action a_t as

$$\delta_{t+1}^a = \begin{cases} 100, & \text{if } a_t \text{ is successful,} \\ -100, & \text{otherwise.} \end{cases} \tag{17}$$

2) Resource Consumption Reward: As above, we define the Resource Consumption Reward value for the placement of VNF v via action a_t as

$$\delta_{t+1}^c = \begin{cases} \frac{req_{(v-1,v)}^{bw}}{req_{(v-1,v)}^{bw}|P|} = \frac{1}{|P|}, & \text{if } |P| > 0, \\ 1, & \text{otherwise.} \end{cases} \tag{18}$$

where P is the path used to place VL $(v-1, v)$. Note that a maximum $\delta_{t+1}^c = 1$ is given when $|P| = 0$, that is, when VNFs $v-1$ and v are placed on the same server.

3) Load Balancing Reward: Finally, we define the Load Balancing Reward value for the placement of VNF v via a_t

$$\delta_{t+1}^b = \frac{cap_{a_t}^{cpu}}{M_{a_t}^{cpu}} + \frac{cap_{a_t}^{ram}}{M_{a_t}^{ram}}. \tag{19}$$

4) Global Reward: On the basis of the three reward values introduced above, we define the global as

$$r_{t+1} = \begin{cases} 0, & \text{if } t < T \text{ and } a_t \text{ is successful} \\ \sum_{i=0}^{T} \delta_{i+1}^a \delta_{i+1}^b \delta_{i+1}^c, & \text{if } t = T \text{ and } a_t \text{ is successful} \\ \delta_{t+1}^a, & \text{otherwise} \end{cases} \tag{20}$$

where T is the number of iterations of a training episode and the quantities δ_{i+1}^a, δ_{i+1}^b, and δ_{i+1}^c are defined by Equations (17), (19) and (18), respectively. The notation r_{t+1} is used to emphasize that the reward obtained via action a_t is provided by the environment after the action is performed, that is, at time step $t+1$. In the present case, we consider $T \leq |V|$. Our reward function formulation is inspired by that proposed in [11] but different. In [11], partial rewards are admitted by considering that $r_{t+1} = \delta_{i+1}^a \delta_{i+1}^b \delta_{i+1}^c$. In practice this approach can lead to ineffective learning, since the agent can learn how to obtain good partial rewards without never reaching its true goal, that is, to place the entire NSPR.

To address this issue, we propose to accumulate the intermediary rewards and return them to the agent only if all the VNFs of the NSPR are placed (second case of Eq. (20)). A 0 reward is given to the agent on the intermediary time steps,

where a VNF is successfully placed (first case of Eq. (20)). If the agent takes an unsuccessful placement action, a negative reward is given (third case of Eq.(20)).

VI. Adaptation of DRL and Introduction of a Heuristic Function

A. Proposed Deep Reinforcement Learning Algorithm

To learn the optimal policy, we use a single thread version of the A3C Algorithm introduced in [24]. This algorithm has evidenced good results when applied to the VNE problem in [11]. A3C uses two DNNs that are trained in parallel: i) the Actor Network with the parameter θ, which is used to generate the policy π_θ at each time step, and ii) the Critic Network with the parameter θ_v which generates an estimate $\nu_{\theta_v}^{\pi_\theta}(\sigma_t)$ for the State-value function defined by

$$\nu_\pi(t|\sigma) = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} | \sigma_t = \sigma \right],$$

equal to the expected return when starting on state σ and following policy π thereafter with the discount parameter γ.

1) DNNs Structure : Both Actor and Critic Networks have identical structure except for their output layer as represented in Fig. 4. As in [11], we use the GCN formulation proposed by [23] to automatically extract advanced characteristics of the PSN. The characteristics produced by the GCN represent semantics of the PSN topology by encoding and accumulating characteristics of neighbour nodes in the PSN graph. The size of the neighbourhood is defined by the order-index parameter K. If K is too large, the computation becomes expensive.

If K is too small, the GCN will use information from a small subset of nodes and can become ineffective. The reader may refer to [22], [23] for more details. As in [11], we consider in the following $K = 3$ and perform automatic extraction of 60 characteristics per PSN node.

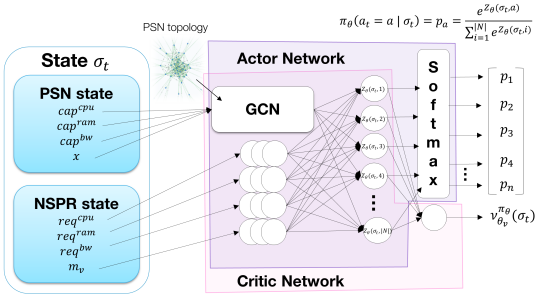


Fig. 4. Architecture of the proposed DRL algorithm

The NSPR state characteristics are separately transmitted to a fully connected layer with 4 units. The characteristics extracted by both layers are combined into a single column vector of size $60|N| + 4$ and passed through a full connection layer with $|N|$ units. In the Critic Network, the outputs of this layer are forwarded to a single neuron, which is used to calculate the state-value function estimation $\nu_{\theta_v}^{\pi_\theta}(\sigma_t)$. In the Actor Network, the outputs of this layer represent the values of the function Z_θ introduced in Section V-B. These values are injected into a Softmax layer that transforms them into a Softmax distribution that corresponds to the policy π_θ.

2) DNNs Update: During the training phase, at each time step t, the A3C algorithm uses the Actor Network to calculate the policy $\pi_\theta(.|\sigma_t)$. An action a_t is sampled using the policy and performed on the environment. The Critic Network is used to calculate the state-value function approximation $\nu_{\theta_v}^{\pi_\theta}(\sigma_t)$. The agent receives then the reward r_{t+1} and next state σ_{t+1} from the environment and the placement process continues until a terminal state is reached, that is, until the Actor Network returns an unsuccessful action or until the current NSPR is completely placed. At the end of the training episode, the A3C algorithm updates parameters θ and θ_v adopting the rules given in Table III, derived from the PG Theorem [24].

TABLE III
Updates of the control parameters θ and θ_v.

$$J(\theta) = \sum_{t=t_0}^{T} \log\left(\pi_\theta(a_t|\sigma_t)\right) A^{\pi_\theta}(\sigma_t, a_t)$$
$$\delta(\theta) = \sum_{t=t_0}^{T} H(\pi_\theta(.|\sigma_t))$$
$$\theta \leftarrow \theta + \frac{\alpha}{T - t_0 + 1} \left(\nabla_\theta J(\theta) + \phi \nabla_\theta \delta(\theta)\right) \quad (21)$$

$$J(\theta_v) = \sum_{t=t_0}^{T} \left(r_{t+1} + \nu_{\theta_v}^{\pi_\theta}(\sigma_{t+1}) - \nu_{\theta_v}^{\pi_\theta}(\sigma_t)\right)^2$$
$$\theta_v \leftarrow \theta_v + \frac{\alpha'}{T - t_0 + 1} \nabla_{\theta_v} J(\theta_v) \quad (22)$$

In Table III, the term $A^{\pi_\theta}(\sigma_t, a_t)$ represents an estimate of the Advantage function, which indicates how better is the action a_t when compared against the "average action" from the corresponding policy under a certain state σ_t. By applying the Temporal Difference (TD) method [34], $A^{\pi_\theta}(\sigma_t, a_t) = r_{t+1} + \nu_{\theta_v}^{\pi_\theta}(\sigma_{t+1}) - \nu_{\theta_v}^{\pi_\theta}(\sigma_t)$.

The $J(\theta)$ function is the performance of the Actor Network. It is given by the Log-likelihood of the policy weighted by the Advantage Function. The $\delta(\theta)$ is the sum of the policy entropy $H(\pi_\theta(.|\sigma_t)$ used as a regularization term to discourage premature convergence. The function $J(\theta_v)$ is the performance of the Critic Network. It is given by the squared Temporal Difference error of $\nu_{\theta_v}^{\pi_\theta}(\sigma_{t+1})$. The gradients ∇_θ and ∇_{θ_v} are the vectors of partial derivatives w.r.t. θ and θ_v, respectively. The hyper-parameters α, α', and ϕ are the learning rates and heuristically fixed. t_0 is the first time step of the training episode and T is the last one. Note that $t_0 \leq T < t_0 + |V|$.

B. Remarks on the Implementation of the DRL Algorithm

All resource-related characteristics are normalized to be in the $[0, 1]$ interval. This is done by dividing cap^j and req^j, $j \in \{\text{cpu, ram,bw}\}$ by $\max_{n \in N} M_n^j$. With regard to the DNNs, we have implemented the Actor and Critic as two independent Neural Networks. Each neuron has a bias assigned. We have used the hyperbolic tangent (tanh) activation for non-output layers of the Actor Network and Rectified Linear Unit (ReLU) activation for all layers of the Critic Network. We have normalized positive global rewards to be in the $[0, 10]$ interval.

During the training phase, we have considered the policy as a Categorical distribution and used it to sample the actions randomly. In the validation tests, we have always selected the action with higher probability, i.e., $a_t = \text{argmax}_{a \in N} \pi_\theta(a|\sigma_t)$. As in [11], we have taken $\phi = 0.5$. We performed hyper-parameter search to define α and α' values as described in Section VII-E.

C. Introduction of a Heuristic

1) Motivation: To learn the optimal policy π^*_θ, the proposed DRL agent needs to perform actions over various states and receives the respective rewards to improve the policy iteratively. However, this is not a straightforward process since the convergence depends on many hyper parameters. The agent may get stuck in suboptimal policies or have trouble to learn good estimations for the state-value function. Also, pure DRL approaches need to perform many actions and visit a huge number states to learn good policies.

As a consequence, these approaches take much time to start providing good solutions. We propose then to reinforce and accelerate the DRL learning process using the heuristic described in Section IV-C.

2) Modified DRL algorithm: Fig. 5 presents the architecture of the proposed HA-DRL algorithm. We modify the structure of the Actor Network by introducing a new layer, namely the Heuristic layer that calculates an Heuristic Function $H : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ based on external information provided by the heuristic method described in Section IV-C and referred to as HEU.

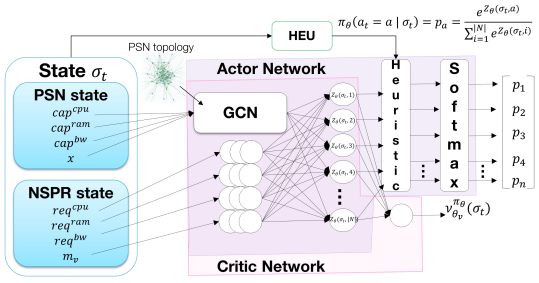


Fig. 5. Architecture of the proposed HA-DRL algorithm

As described in [26], the Heuristic Function $H(\sigma_t, a_t)$ is a policy modifier that defines the importance of executing a certain action a_t in state σ_t. In what follows, we adapt the $H(\sigma_t, a_t)$ formulation proposed in [26] to our DRL algorithm and describe how we use $H(\sigma_t, a_t)$ as a modifier of the Actor Network output to give more importance to the action selected by the HEU method.

3) Heuristic Function Formulation : Let Z_θ be the function computed by the fully connected layer of the Actor Network that maps each state and action to a real value which is after converted by the Softmax layer into the selection probability of the respective action (see Section V-B). Let $\bar{a}_t = \text{argmax}_{a \in \mathcal{A}} Z_\theta(\sigma_t, a)$ be the action with the highest Z_θ value for state σ_t. Let $a^*_t = HEU(\sigma_t)$ be the action derived by the HEU method at time step t and the preferred action to be chosen. $H(\sigma_t, a^*_t)$ is shaped to allow the value of $Z_\theta(\sigma_t, a^*_t)$ to become closer to the value of $Z_\theta(\sigma_t, \bar{a}_t)$. The aim is to turn a^*_t into one of the likeliest actions to be chosen by the policy.

The Heuristic Function is then formulated as

$$H(\sigma_t, a_t) = \begin{cases} Z_\theta(\sigma_t, \bar{a}_t) - Z_\theta(\sigma_t, a_t) + \eta, & \text{if } a_t = a^*_t \\ 0, & \text{otherwise} \end{cases} \quad (23)$$

where η parameter is a small real number. During the training process, the Heuristic layer calculates $H(\sigma_t, .)$ and updates the $Z_\theta(\sigma_t, .)$ values by using the following equation:

$$Z_\theta(\sigma_t, .) = Z_\theta(\sigma_t, .) + \xi H(\sigma_t, .)^\beta \quad (24)$$

The Softmax layer then computes the policy using the modified Z_θ. Note the action returned by a^*_t will have a higher probability to be chosen. The ξ and β are parameters used to control how much HEU influence the policy.

VII. Implementation and Evaluation Results

A. Implementation Details & Simulator Settings

1) Experimental setting: In the absence of real data sets of slice demand and data center capacities, we simulate realistic network scenarios. We consider fixed capacities and demands and we control the "difficulty" of the problem by varying the network load. We developed a simulator in Python containing: i) the elements of the Network Slice Placement Optimization problem (i.e., PSN and NSPR); ii) the DRL and HA-DRL algorithms. We used the PyTorch framework to implement the DNNs. We consider an implementation of the HEU algorithm [10] in Julia as a benchmark in the performance evaluations. Experiments were run in a 2x6 cores @2.95Ghz 96GB machine.

2) Physical Substrate Network Settings: We consider a PSN that could reflect the infrastructure of an operator [35]. In this network, three types of DCs are introduced as in Section III. Each CDC is connected to three EDCs which are 100 km apart. CDCs are interconnected and connected to a CCP that is 300 km away. The Tables IV and V summarize the DCs and transport links properties. The CPU and RAM capacities of each server are 50 and 300 units, respectively.

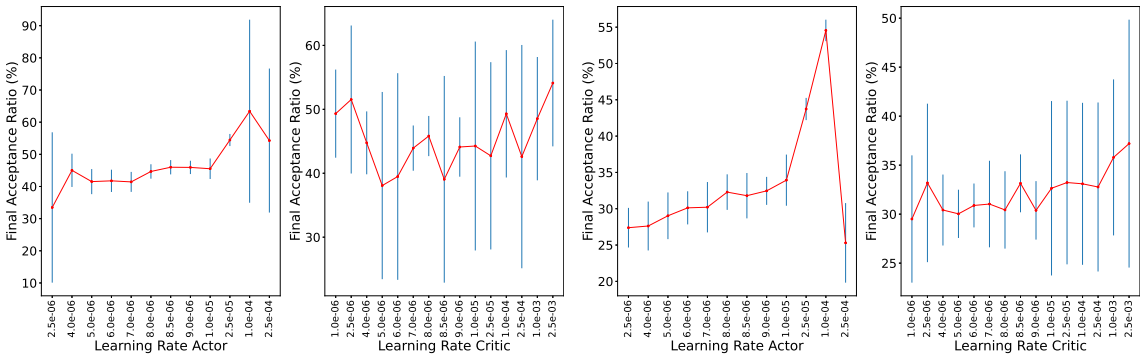
TABLE IV
Data centers description

Data center type	Number of data centers	Number of servers per data center	Intra data center links bandwith capcity
CCP	1	16	100 Gbps
CDC	5	10	100 Gbps
EDC	15	4	10 Gbps

TABLE V
Transport links capacities

	CCP	CDC	EDC
CCP	NA	100 Gbps	100 Gbps
CDC	100 Gbps	100 Gbps	100 Gbps
EDC	10 Gbps	10 Gbps	10 Gbps

3) Network Slice Placement Requests Settings : We consider NSPRs to have the Enhanced Mobile Broadband (eMBB) setting described in [10]. Each NSPR is composed of 5 VNFs. Each VNF requires 25 units of CPU and 150 units of RAM. Each VL requires 2 Gbps of bandwidth.



(a) Actor Network, $\rho = 0.50$. (b) Critic Network, $\rho = 0.50$. (c) Actor Network, $\rho = 1.0$. (d) Critic Network, $\rho = 1.0$.

Fig. 6. Hyper-parameter search results.

B. Algorithms & Experimental Setup

1) Training Process & Hyper-parameters: We consider a training process with maximum duration of 24 hours for the DRL and HA-DRL agents with learning rates for the Actor and Critic networks set to $\alpha = 10^{-4}$ and $\alpha' = 2.5 \times 10^{-3}$ (see Section VII-E about hyper-parameter tuning). We program four versions of HA-DRL agent (HA-DRL, $\beta = 0.1$; HA-DRL, $\beta = 0.5$; HA-DRL, $\beta = 1.0$; HA-DRL, $\beta = 2.0$), each with a different value for the β parameter of the heuristic function formulation (see Section VI-C3). We set the parameters $\xi = 1$ and $\eta = 0$.

2) Heuristic baseline: Since the performance of the HEU algorithm does not depend on learning, we consider the placement of 100,000 NSPRs with the HEU algorithm and use its performance in the steady state as a benchmark.

C. Network Load Calculation

We use the formula proposed in [35] to compute the NSPR arrival rates (λ^k) under the three network load conditions considered in the evaluation: underload ($\rho = 0.5$), normal load ($\rho = 0.8$ and $\rho = 0.9$), and critical load ($\rho = 1.0$). Network loads are calculated using CPU resources. In general, we set $1/\mu_k = 100$ time units for all $k \in \mathcal{K}$, where $\mathcal{K}$ is the number of slice classes. For the resource j with total capacity C_j, the load is $\rho_j = \frac{1}{C_j} \sum_{k=1}^{\mathcal{K}} \frac{\lambda^k}{\mu^k} A_j^k$, where A_j^k is the number of resource units requested by an NSPR of class k.

D. Evaluation Metrics

To characterize the performance of the placement algorithms, we consider 3 performance metrics:

1) **Average execution time:** the average execution time in seconds required to place 1 NSPR. This metric is calculated based on 100 NSPR placements;
2) **Acceptance Ratio per training phase**: the Acceptance Ratio obtained in each training phase, i.e., each part of the training process, corresponding to 1000 NSPR arrivals. It is calculated as follows: $\frac{\#\text{accepted NSPRs}}{1000}$. This metric is used to evaluate the convergence of the algorithms as it allows us to observe of the evolution of the agent's performance in time;
3) **Acceptance Ratio after training:** the Acceptance Ratio of the different tested algorithms after training computed after each arrival as follows: $\frac{\#\text{accepted NSPRs}}{\#\text{arrived NSPRs}}$. This metric is used in our validation test to compare the performance of the agents after training.

E. Hyper-parameter Tuning

To define the appropriate learning rates α and α' for training the Actor and Critic networks, respectively, we perform a classical hyper-parameter search procedure divided in 3 steps: i) we conduct several training experiments with different combinations of α and α' values, ii) we observe the final acceptance ratios, i.e., the acceptance ratios obtained in the last training phase of each experiment, and iii) we aggregate the final acceptance ratios by learning rate values.

The DNNs are trained for 4 hours and we consider α and α' to be in the following intervals: $\alpha \in [0.00001, 0.00025]$ and $\alpha' \in [0.000001, 0.0025]$, where upper bounds of the intervals are the learning rates used in [11]. Fig. 6 presents the average values and standard deviations of the final acceptance ratios aggregated by learning rate for the Actor and Critic networks considering two values for the network load parameter ρ: $\rho = 0.5$ (Fig. 6a and 6b) and $\rho = 1.0$ (Fig. 6c and 6d). As shown in Fig. 6a and 6c, under both network load conditions, the best average of final acceptance ratios was achieved when considering a learning rate $\alpha = 10^{-4}$ for the Actor Network. This fact can be observed even more clearly when $\rho = 1.0$ since we observe a peak in the curve when $\alpha = 10^{-4}$ in addition to a smaller standard deviation as shown by Fig. 6c.

Fig. 6b and 6d show that the best average of final acceptance rate was obtained when considering a learning rate $\alpha' = 2.5 \times 10^{-3}$ for the Critic Network. We also observe a higher standard deviation when aggregating the final acceptance ratios on the basis α' than on the basis of α. The parameter α thus seems to have a stronger impact on the final acceptance ratios obtained than obtained for α'.

F. Acceptance Ratio Evaluation

Fig. 7 and Tables VI-IX present the Acceptance Ratio per training phase obtained with the HA-DRL, DRL and HEU algorithms under different network loads.

HA-DRL with $\beta = 2.0$ exhibits the most robust performance with convergence after a few training phases for all values of ρ. This happens because when setting $\beta = 2.0$ the Heuristic Function computed on the basis of the HEU algorithm has

TABLE VI
ACCEPTANCE RATIO AT DIFFERENT TRAINING PHASES, $\rho = 0.5$.

Algorithm	Acceptance Ratio at different Training Phases (%)				
	25	100	200	300	400
HADRL,β=0.1	57.90	80.20	92.70	93.00	96.20
HADRL,β=0.5	58.50	83.00	90.20	94.80	96.30
HADRL,β=1.0	58.80	86.20	86.80	85.50	85.80
HADRL,β=2.0	93.50	90.30	93.10	92.20	94.80
DRL	57.10	83.60	91.20	94.80	95.70
HEU	93.50	94.00	94.00*	94.00*	94.00*

TABLE VII
ACCEPTANCE RATIO AT DIFFERENT TRAINING PHASES, $\rho = 0.8$.

Algorithm	Acceptance Ratios at different Training Phases (%)					
	25	100	200	300	400	480
HADRL,β=0.1	44.10	74.60	77.80	82.80	87.50	85.30
HADRL,β=0.5	46.30	75.00	77.40	80.90	86.90	83.60
HADRL,β=1.0	46.00	77.00	74.90	72.40	74.80	74.10
HADRL,β=2.0	87.80	88.80	85.60	86.50	84.90	85.40
DRL	46.10	71.89	75.70	78.40	83.70	77.80
HEU	79.20	79.27	79.27*	79.27*	79.27*	79.27*

TABLE VIII
ACCEPTANCE RATIO AT DIFFERENT TRAINING PHASES, $\rho = 0.9$.

Algorithm	Acceptance Ratios at different Training Phases (%)					
	25	100	200	300	400	480
HADRL,β=0.1	42.40	66.20	75.00	73.00	81.70	78.50
HADRL,β=0.5	40.30	63.40	70.90	68.20	78.20	75.30
HADRL,β=1.0	42.69	66.00	70.90	68.40	71.10	66.90
HADRL,β=2.0	82.89	81.10	84.10	81.00	82.80	83.36
DRL	41.60	63.40	72.10	71.10	80.60	75.80
HEU	73.90	75.68	75.68*	75.68*	75.68*	75.68*

strong influence on the actions chosen by the agent. Since the HEU algorithm often indicates a good action, this strong influence of the heuristic function helps the algorithm to become stable more quickly.

Fig. 7a and Table VI reveal that DRL and HA-DRL algorithms with $\beta \in \{0.1, 0.5, 1.0\}$ converge within an interval of 200 to 300 training phases when $\rho = 0.5$ and that all algorithms except HA-DRL with $\beta = 1.0$ have an Acceptance Ratio higher than 94% in the last training phase.

Fig. 7b and Table VII show that the performance of DRL and HA-DRL algorithms with $\beta \in \{0.1, 0.5, 1.0\}$ stabilizes only after more than 400 training phases when $\rho = 0.8$. They also show that when $\rho = 0.8$, only HA-DRL with $\beta \in \{0.1, 0.5, 2.0\}$ have an Acceptance Ratio higher than 83% in the last training phase. Algorithms HA-DRL with $\beta = 0.1$ and HA-DRL with $\beta = 2.0$ have similar performance and they are about 2% better than HA-DRL for $\beta = 0.5$, 6% better than HEU, 8% better than DRL, and 11% better than HA-DRL with $\beta = 1.0$. As shown in Fig. 7c and Table VIII when $\rho = 0.9$, only HA-DRL with $\beta = 2.0$ has an Acceptance Ratio higher than 83% in the last training phase. HA-DRL with $\beta = 2.0$ is about 5% better HA-DRL with $\beta = 0.1$, 8% better than DRL, HEU and HA-DRL with $\beta = 0.5$, and 16% better than HA-DRL with $\beta = 1.0$.

Fig. 7c and Table VIII also reveal that the performance of algorithms DRL and HA-DRL with $\beta \in \{0.1, 0.5, 1.0\}$ only stabilizes after more than 400 training phases when $\rho = 0.9$. Fig. 7d and Table IX show that when $\rho = 1.0$, HA-DRL with $\beta = 2.0$ performs significantly better than the other algorithms at the end of the training. HA-DRL with $\beta = 2.0$ accepts 67.2 % of the NSPR arrivals in the last training phase, 8.34% more than HEU, 10.21% more than DRL, 14.7% more than HA-DRL with $\beta = 1.0$, 22.6% more than HA-DRL with $\beta = 0.5$, 29.3% more than HA-DRL with $\beta = 0.1$. Fig. 7d and Table IX also show that performance of algorithms DRL and HA-DRL with $\beta \in \{0.1, 0.5, 1.0\}$ is still not stable at the end of the training process when $\rho = 1.0$.

G. Execution Time Evaluation

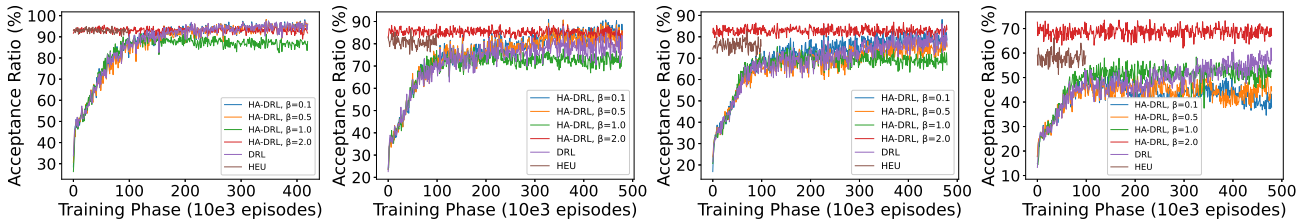
Fig. 8a and 8b present the average execution time of the HEU, DRL and HA-DRL algorithms as a function of the number of VNFs in the NSPR and the number of servers in the PSN, respectively (see Section VII-D for details on the metric calculation). In both evaluations, we start by the PSN and NSPR settings described in Sections VII-A2 and VII-A3, respectively, and generate new settings by increasing either the number of VNFs per NSPR or the number of servers in the PSN. The evaluation results confirm our expectations by showing that the average execution times increase faster for heuristics than for a pure DRL approach. However, both HEU and DRL strategies have low execution times (less than 0.6s in the largest scenarios). The number of VNFs per NSPR has more impact on the average execution times of HEU and DRL algorithms than the number of servers on the PSN. The average execution time of HEU algorithm is more impacted than DRL by the number of servers in the PSN. The HA-DRL algorithm depends on a sequential execution of DRL and HEU. Therefore, the average execution time of HA-DRL is approximately to the sum of the execution times of HEU and DRL. Since DRL and HEU have small execution times, the average execution times of HA-DRL are also small (less than 1.0s for the largest NSPR setting and about 0.6s for the largest PSN setting).

H. Validation Test

To validate the effectiveness of the different trained DRL agents, we perform a validation test. We consider the same PSN and NSPR settings described in Section VII and the arrival of 10,000 NSPRs to be placed and generating a network load $\rho = 0.8$. We run a simulation with each one of the trained

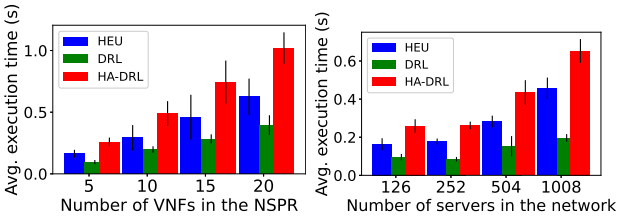
TABLE IX
ACCEPTANCE RATIO AT DIFFERENT TRAINING PHASES, $\rho = 1.0$.

Algorithm	Acceptance Ratios at different Training Phases (%)					
	25	100	200	300	400	480
HADRL,β=0.1	30.10	49.70	45.30	45.0	41.00	37.90
HADRL,β=0.5	30.80	45.90	50.80	44.5	38.90	44.60
HADRL,β=1.0	27.40	52.00	55.50	55.60	49.00	52.50
HADRL,β=2.0	67.60	67.60	70.80	69.60	66.10	67.2
DRL	29.30	49.10	46.60	50.10	53.40	56.99
HEU	60.70	58.86	58.86*	58.86*	58.86*	58.86*



(a) $\rho = 0.50$. (b) $\rho = 0.80$. (c) $\rho = 0.90$. (d) $\rho = 1.0$.

Fig. 7. Acceptance ratio evaluation results.



(a) (b)

Fig. 8. Average execution time evaluation.

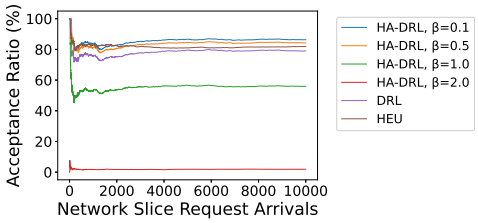


Fig. 9. Acceptance Ratio at validation test, $\rho = 0.8$

DRL agents as well with the HEU algorithm and compare the obtained Acceptance Ratios at the end of the simulations (see description of the Acceptance Ratio after training metric on Section VII-D).

Fig. 9 and Table X show that the HA-DRL agent with $\beta = 0.1$ is the one that better scales since it has the best Acceptance Ratio after training. HA-DRL agent with $\beta = 2.0$ has the best Acceptance Ratio during training as described in Section VII-F but is the one that scales the worst since it has poorest Acceptance Ratio after training performance. This is because with the use of a high β value, HA-DRL suffers from a strong dependence on the HEU algorithm. This prevents HA-DRL with $\beta = 2.0$ from being used with the HEU algorithm deactivated. Note, however, that HA-DRL remains the best solution even when HEU is disabled after 24 hours of training since HA-DRL with $\beta = 0.1$ has an Acceptance Ratio 7.34% higher than the pure DRL and 4.42% higher than the HEU algorithm. To get the best performance we can then use the HEU algorithm to help HA-DRL to learn during the training phase and then "disengage" it to avoid the overhead generated by the "node pooling" principle used by the HEU algorithm.

VIII. Conclusion

We have presented a Heuristically-assisted DRL (HA-DRL) approach to Network Slice Placement Optimization with 5 main contributions: the proposed method i) enhances the scalability of existing ILP and heuristic approaches, ii) can cope with multiple optimization criteria, iii) combines DRL with GCN to automate feature extraction, iv) strengthens and accelerates the DRL learning process using an efficient placement optimization heuristic, and v) supports multi-domain slice placement.

TABLE X
Values of Acceptance Ratio after Training

HADRL,$\beta = 0.1$	86.31%
HADRL,$\beta = 0.5$	84.31%
HADRL,$\beta = 1.0$	55.95%
HADRL,$\beta = 2.0$	1.89%
DRL	78.97%
HEU	81.89%

Evaluation results show that the proposed HA-DRL approach yields good placement solutions in nearly real time, converges significantly faster than pure DRL approaches, and yields better performance in terms of acceptance ratio than state-of-the-art heuristics and pure DRL algorithms during and after the training phase. As a future work, we plan to explore a parallel computing implementation of HA-DRL to reduce its execution time and thus achieve the best performance in both Acceptance Ratio and Execution Time. We also plan to evaluate the HA-DRL in an online optimization scenario with fluctuating traffic demand to access the advantages of using HA-DRL in practice.

Acknowledgment

This work has been performed in the framework of 5GPPP MON-B5G project (www.monb5g.eu). The experiments were conducted using Grid'5000, a large scale testbed by Inria and Sorbonne University (www.grid5000.fr).

References

[1] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.

[2] 3GPP, "Management and orchestration; 5G Network Resource Model (NRM); Stage 2 and stage 3 (Release 17)," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 28.541, Dec. 2020, version 17.1.0.

[3] ETSI NFV ISG, "Network Functions Virtualisation (NFV); Evolution and Ecosystem; Report on Network Slicing Support, ETSI Standard GR NFV-EVE 012 V3.1.1," ETSI, Tech. Rep., 2017. [Online]. Available: https://www.etsi.org/technologies-clusters/technologies/nfv

[4] A. Fischer, J. F. Botero Vega, M. Duelli, D. Schlosser, X. Hesselbach Serra, and H. De Meer, "ALEVIN – a framework to develop, compare, and analyze virtual network embedding algorithms," *Open Access J. Electron. Commun. EASST*, pp. 1–12, Mar. 2011.

[5] H. Cao, S. Wu, Y. Hu, Y. Liu, and L. Yang, "A survey of embedding algorithm for virtual network embedding," *China Commun.*, vol. 16, no. 12, pp. 1–33, Dec. 2019.
[6] J. Gil Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.
[7] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1409–1434, 2nd. Quart., 2019.
[8] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, Feb. 2013.
[9] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," *Electron. Notes Discrete Math.*, vol. 52, pp. 213–220, Jun. 2016.
[10] J. J. Alves Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Heuristic for edge-enabled network slicing optimization using the "power of two choices"," in *Proc. 2020 IEEE 16th Int. Conf. Netw. Service Manag. (CNSM)*, 2020, pp. 1–9.
[11] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1040–1057, Jun. 2020.
[12] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "DeepViNE: Virtual network embedding with deep reinforcement learning," in *Proc. IEEE INFOCOM 2019 - IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2019, pp. 879–885.
[13] H. Yao, X. Chen, M. Li, P. Zhang, and L. Wang, "A novel reinforcement learning algorithm for virtual network embedding," *Neurocomputing*, vol. 284, pp. 1–9, Apr. 2018.
[14] M. Mechtri, C. Ghribi, and D. Zeghlache, "VNF placement and chaining in distributed cloud," in *Proc. 9th IEEE Int. Conf. Cloud Comput.*, 2016, pp. 376–383.
[15] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proc. IEEE Conf. Comput. Commun.*, 2009, pp. 783–791.
[16] A. Alleg, T. Ahmed, M. Mosbah, R. Riggio, and R. Boutaba, "Delay-aware VNF placement and chaining based on a flexible resource allocation approach," in *Proc. 13th IEEE Int. Conf. Netw. Service Manage.*, 2017, pp. 1–7.
[17] A. Rkhami, Y. Hadjadj-Aoul, and A. Outtagarts, "Learn to improve: A novel deep reinforcement learning approach for beyond 5G network slicing," in *Proc. 2021 IEEE 18th Annu. Consum. Commun. Netw. Conf. (CCNC)*, 2021, pp. 1–6.
[18] H. Wang, Y. Wu, G. Min, J. Xu, and P. Tang, "Data-driven dynamic resource scheduling for network slicing: A deep reinforcement learning approach," *Inf. Sci.*, vol. 498, pp. 106–116, Sep. 2019.
[19] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *Proc. 2019 IEEE/ACM 27th Int. Symp. Qual. Service (IWQoS)*, 2019, pp. 1–10.
[20] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A deep reinforcement learning approach for vnf forwarding graph embedding," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 4, pp. 1318–1331, Dec. 2019.
[21] P. T. A. Quang, A. Bradai, K. D. Singh, and Y. Hadjadj-Aoul, "Multi-domain non-cooperative VNF-FG embedding: A deep reinforcement learning approach," in *Proc. IEEE INFOCOM 2019 - IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2019, pp. 886–891.
[22] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. 30th Int. Conf. Neural Processing Syst. (NIPS)*, 2016, pp. 3844–3852.
[23] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Representations (ICLR)*, 2017, pp. 1–14.
[24] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Int. Conf. Mach. Learn.* PMLR, 2016, pp. 1928–1937.
[25] R. A. Bianchi, C. H. Ribeiro, and A. H. Costa, "Accelerating autonomous learning by using heuristic selection of actions," *J. Heuristics*, vol. 14, no. 2, pp. 135–168, 2008.
[26] R. A. Bianchi, C. H. Ribeiro, and A. H. R. Costa, "Heuristically accelerated reinforcement learning: Theoretical and experimental results." in *in Proc. 20th Eur. Conf. Artif. Intell. (Frontiers in Artificial Intelligence and Applications, vol. 242)*., 2012, pp. 169–174.
[27] J. Supancic and D. Ramanan, "Tracking as online decision-making: Learning a policy from streaming videos with reinforcement learning," in *Proc. 2017 IEEE Int. Conf. on Comput. Vision (ICCV)*, 2017, pp. 322–331.
[28] R. A. Bianchi, M. F. Martins, C. H. Ribeiro, and A. H. Costa, "Heuristically-accelerated multiagent reinforcement learning," *IEEE Trans. Cybern.*, vol. 44, no. 2, pp. 252–265, Feb. 2014.
[29] R. A. Bianchi, P. E. Santos, I. J. Da Silva, L. A. Celiberto, and R. L. de Mantaras, "Heuristically accelerated reinforcement learning by means of case-based reasoning and transfer learning," *J. Intelligent Robotic Syst.*, vol. 91, no. 2, pp. 301–312, Oct. 2017.
[30] N. Morozs, T. Clarke, and D. Grace, "Heuristically accelerated reinforcement learning for dynamic secondary spectrum sharing," *IEEE Access*, vol. 3, pp. 2771–2783, Dec. 2015.
[31] F. Slim, F. Guillemin, and Y. Hadjadj-Aoul, "CLOSE: A costless service offloading strategy for distributed edge cloud," in *Proc. 2018 15th IEEE Annu. Cons. Commun. Netw. Conf. (CCNC)*, 2018, pp. 1–6.
[32] L. Taccari, "Integer programming formulations for the elementary shortest path problem," *Eur. J. Oper. Res.*, vol. 252, no. 1, pp. 122–130, Jul. 2016.
[33] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, Oct. 2001.
[34] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA, USA: MIT press, 2015.
[35] F. Slim, F. Guillemin, A. Gravey, and Y. Hadjadj-Aoul, "Towards a dynamic adaptive placement of virtual network functions under ONAP," in *Proc. 2017 IEEE Conf. on Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, 2017, pp. 210–215.

José Jurandir Alves Esteves graduated from the University of Clermont Auvergne in 2017 and from the Federal University of Minas Gerais in 2019 obtaining two engineering degrees and a master's degree in computer science from the University of Clermont Auvergne. He is doing his PhD between Orange Labs and the Computer Science Laboratory of Paris 6 (LIP6), Sorbonne University. His research is related to automation and optimization models and algorithms for network slice orchestration.

Amina Boubendir is a researcher and project manager at Orange Labs in France. Her research is in the area of design, management and softwarization of networks and services. Amina received a Master degree in Network Design and Architecture from Télécom Paris in 2013, and a PhD in Networking and Computer Science from Télécom Paris in 2016. She is a member of the Orange Expert community on "Networks of Future".

Fabrice Guillemin graduated from Ecole Polytechnique in 1984 and from Telecom Paris in 1989. He received the PhD degree from the University of Rennes in 1992. He defended his "habilitation" thesis in 1999 at the University Pierre et Marie Curie (LIP6), Paris. Since 1989, he has been with Orange Labs (former CNET and France Telecom R&D). He is currently leading a project on the evolution of network control. He is a member of the Orange Expert community on "Networks of Future".

Pierre Sens received his Ph.D. in Computer Science in 1994, and the "Habilitation à diriger des recherches" in 2000 from Paris 6 University (UPMC), France. Currently, he is a full Professor at Sorbonne Université (ex-UPMC). His research interests include distributed systems and algorithms, large scale data storage, fault tolerance, and cloud computing. He is leading Delys a joint research team between LIP6 and Inria.