A Multi-Agent Reinforcement Learning Architecture for Network Slicing Orchestration

Federico Mason
Dep. of Information Engineering
University of Padova
Padova, Italy
masonfed@dei.unipd.it

Gianfranco Nencioni
Dep. of Electrical Engineering and Computer Science
University of Stavanger
Stavanger, Norway
gianfranco.nencioni@uis.no

Andrea Zanella
Dep. of Information Engineering
University of Padova
Padova, Italy
zanella@dei.unipd.it

2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet) | 978-1-6654-3590-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/MEDCOMNET52149.2021.9501279

***Abstract*—The Network Slicing (NS) paradigm is one of the pillars of the future 5G networks and is gathering great attention from both industry and scientific communities. In a NS scenario, physical and virtual resources are partitioned among multiple logical networks, named slices, with specific characteristics. The challenge consists in finding efficient strategies to dynamically allocate the network resources among the different slices according to the user requirements. In this paper, we tackle the target problem by exploiting a Deep Reinforcement Learning approach. Our framework is based on a distributed architecture, where multiple agents cooperate towards a common goal. The agent training is carried out following the Advantage Actor Critic algorithm, which makes it possible to handle continuous action spaces. By means of extensive simulations, we show that our strategy yields better performance than an efficient empirical algorithm, while ensuring high adaptability to different scenarios without the need for additional training.**

***Index Terms*—Network slicing; resource allocation; distributed machine learning; deep reinforcement learning.**

I. Introduction

THE fifth generation of cellular networks (5G) aims at supporting multiple applications with different characteristics. In this perspective, the 3GPP consortium has identified three main service classes, namely enhanced Mobile BroadBand (eMBB), Ultra Reliable Low Latency Communication (URLLC) and massive Machine Type Communication (mMTC), each with specific performance requirements [1].

Traditional telecommunication networks are often based on a rigid architecture and are not apt to support such services. Hence, the research community has been looking towards the Network Slicing (NS) paradigm, which makes it possible to define multiple virtual networks, named slices, over the same physical infrastructure. Each slice has to support communication services with similar characteristics [2], enabling a better orchestration of the system resources. For instance, an eMBB slice (supporting, e.g., video streaming) is characterized by large throughput, while a URLLC slice (supporting, e.g., telesurgery) guarantees extremely high reliability and low latency.

If defined over the same infrastructures, different slices will contend for the same network resources, which can be both physical (e.g., the optical links in the backhaul networks) and virtual (e.g., virtual baseband units running in a data center). In general, such resources are acquired by the *slice manager* (i.e., the body in charge of initializing and orchestrating slices) from the *infrastructure providers* (i.e., the owners of the physical elements of the network). Then, slices are assigned to the *slice tenants* (e.g., virtual network operators), which offer slice services to the end-users. The amount of resources that are needed to support the slice services is determined by the so-called Service Level Agreement (SLA) between the slice tenant and manager [3].

A fundamental challenge in a NS scenario is how to distribute resources among the different slices in an efficient way, ensuring that all the SLAs are satisfied [4]. A *naive* approach consists in statically allocating all the network resources to the different slices. However, this solution does not exploit the statistical multiplexing of the information flows and, consequently, leads to greater over-provisioning costs. On the other hand, conventional resource allocation strategies are often unsuitable because of the high complexity of NS systems.

In this work, we attack the problem by exploiting the Deep Reinforcement Learning (DRL) paradigm, which combines Reinforcement Learning [5] and Neural Networks (NNs) to find strategies for the management of complex systems [6]. In particular, we design a distributed DRL system, where multiple agents, placed in different network locations, collaborate to maximize the system utility, adapting dynamically their behavior to the evolution of the network status. The training is based on the Advantage Actor Critic (A2C) algorithm, which allows the agents to operate in a continuous action space and constantly improve the target policy by an online learning approach [7].

The main contributions of our work are the followings:

- We design a theoretical NS model that makes it possible to assess the performance of heterogeneous telecommunication services in different scenarios. Hence, we formulate an optimization problem, where edge and core network resources have to be allocated among multiple information flows.

This work was supported by Consortium GARR through the "Orio Carlini" scholarship 2019.

- We develop a DRL strategy to solve the target problem and dynamically allocate network resources (i.e., bandwidth, computational power, and storage) to different slices. The proposed approach is characterized by high flexibility and can be implemented in many network topologies without the need for additional training.
- We evaluate the performance of our proposal against an efficient empirical strategy and under different working conditions.

The remainder of the work is organized as follows. Sec. II discusses the most relevant works in the considered topic. Sec. III describes the system model used for our analysis. Sec. IV presents our DRL strategy for resource allocation. In Sec. V, we describe the simulation scenario and present the results of our research. Finally, in Sec. VI, we draw the conclusions.

II. Related Work

The orchestration of network resources is one of the most critical aspects of NS. In this respect, the authors of [8] analyze a 5G scenario with end-to-end slices contending for the virtual resources offered by data centers, proposing a fully distributed algorithm to maximize system performance. In [9], Leconte *et al.* design a NS model where multiple traffic flows share network bandwidth and cloud processing units; hence, they implement the Alternating Direction Method of Multipliers [10] to determine the best resource allocation scheme. The authors of [11] adopt a similar approach, considering a system where multiple network operators share both licensed and unlicensed spectrum. Besides, Fossati *et al.* propose a framework to generalize multi-resource allocation techniques according to different fairness goals, considering also the case where resources are not sufficient to satisfy all the slice demands [12].

Because of the high complexity of slice orchestration, the scientific community has shown great interest in leveraging Machine Learning (ML) techniques in this field. For instance, in [13], a NN system makes it possible to predict the traffic evolution of a mobile network, thus optimizing the routing and the wavelength assignment of data flows. Another example can be found in [14], where the authors exploit generative adversarial NNs to minimize the noise in the measurement of SLA satisfaction.

Among all the ML frameworks, DRL is particularly appreciated because of its ability to learn complex strategies by trial and error. In [15], the authors define a novel slice admission policy, based on Q-Learning [16], that considers both the bandwidth and computational requirements of the system services. Instead, Ayala-Romero *et al.* orchestrate virtualized radio resources by employing a DRL algorithm that encodes traffic data into resource allocation decisions [17]. A similar approach is considered in [18], where the authors reconfigure processing power and storage capacity in order to dynamically adapt virtual network functions to different services. Besides, Abiko *et al.* develop a multi-agent system to distribute radio resource blocks and prove its adaptability to a variable number of slices [19]. Finally, the authors of [20] adopt a DRL architecture to balance the communication requirements of eMBB and URLLC services; particularly, an Actor-Critic algorithm is used to schedule URLLC transmissions without degrading the eMBB reliability.

Despite the growing interest in this domain, many open questions still need investigation. Most of the aforementioned approaches, indeed, focus on the management of local functionalities, without addressing the problem of jointly optimizing all the available resources in a wide telecommunication system including multiple network elements. Besides, the great heterogeneity of NS scenarios requires the implementation of more flexible strategies, that can promptly react to changes in the service requirements. A very promising solution is to exploit hierarchical reinforcement learning architectures, which is an approach that has not yet been fully investigated in this context. Our work develops along with this direction, with the final aim of designing a fully scalable DRL system that can be separated into smaller units, capable of both acting autonomously and cooperating to orchestrate network resources under multiple working conditions.

III. System Model

In this section, we model a NS environment where multiple information flows contend for the same physical and virtual network resources. We adopt a fluid traffic model, where the traffic through a link is viewed as a fluid stream of data with a certain flow rate. The generality and flexibility thus promoted allow our system to model the behavior of many of the actors involved in a NS scenario. For the reader convenience, we report the main parameters of our model in Tab. I.

A. Slice Model

In our system, we define a *network slice* as an aggregation of information flows with similar requirements. We denote by Σ the set of slice classes, and by Φ the set of information flows. Given a class $\sigma \in \Sigma$, Φ_σ indicates the set of the information flows belonging to σ. Each information flow $\phi \in \Phi$ is characterized by a tuple of parameters, namely:

- the *flow endpoints* $\mathcal{E}_\phi = (\epsilon^i_\phi, \epsilon^e_\phi)$, i.e., the network nodes where the users' data enter/exit the slice, which usually correspond to base stations, edge routers of autonomous systems, or servers;
- the *resource demand vector* $\mathbf{r}_\phi = [\eta_\phi, c_\phi, m_\phi, \delta_\phi]$, whose elements are the requirements in terms of throughput (η), computational power (c), memory capacity (m), and delay (δ) of the flow;
- the *performance function* $F_\phi(\cdot)$, which describes the performance of the considered information flow according to the level the SLA is fulfilled.

We assume that $\mathcal{E}_\phi$, $F_\phi(\cdot)$ and δ_ϕ are fixed, while η_ϕ, c_ϕ and m_ϕ can change in time. In particular, the time is discretized in timeslots of T seconds, and the information flow parameters can change only slot by slot. The symbol $\mathbf{r}_\phi(t)$ indicates the resource *demanded* by ϕ during timeslot t, while $\hat{\mathbf{r}}_\phi(t) = [\hat{\eta}_\phi(t), \hat{c}_\phi(t), \hat{m}_\phi(t), \hat{\delta}_\phi(t)]$ indicates the resources *assigned*

TABLE I: Model parameters.

Parameter	Description	Parameter	Description	Parameter	Description
$\phi \in \Phi$	Information flow	$\epsilon^i_\phi, \epsilon^e_\phi$	Flow endpoints	B_l	Link rate capacity [bps]
$\sigma \in \Sigma$	Slice class	$\mathbf{r}_\phi$	Flow demand vector	C^c_n	Node computational capacity [bps]
$l \in \mathcal{L}$	Link	ρ_ϕ	Resource required by ϕ	C^m_n	Node memory capacity [b]
$n \in \mathcal{N}$	Node	$\hat{\rho}_\phi$	Resource assigned to ϕ	$b^i_{l,\phi}$	Input flow rate [bps]
η	Throughput [bps]	$f_\sigma(\cdot)$	Resource performance function	$b^o_{l,\phi}$	Output flow rate [bps]
c	Computational power [bps]	$F_\sigma(\cdot)$	Flow performance function	τ_n	Node routing delay [s]
m	Memory capacity [b]	Ω	System utility	$D_{l,\phi}$	Data of ϕ queued in l [b]
δ	Delay [s]	$b_{l,\phi}$	Bit rate assigned by l to ϕ [bps]	$\tau^q_{l,\phi}$	Queuing delay of ϕ in l [s]
t	Discrete time	$c_{n,\phi}$	Computation assigned by n to ϕ [bps]	$\tau^\tau_{l,\phi}$	Transmission delay of ϕ in l [s]
T	Timeslot duration [s]	$m_{n,\phi}$	Memory assigned by n to ϕ [b]	τ^p_l	Link propagation delay [s]

to ϕ during timeslot t. Note that $\mathbf{r}_\phi(t)$ is determined by the slice class σ that ϕ belongs to, while $\hat{\mathbf{r}}_\phi(t)$ is determined by the resource allocation strategy.

All the information flows belonging to the same class σ share the same performance function $F_\sigma(\cdot)$, i.e., $\forall \phi \in \Phi_\sigma$, $F_\phi(\cdot) = F_\sigma(\cdot)$. In general, $F_\sigma(\cdot)$ takes $\mathbf{r}$ and $\hat{\mathbf{r}}$ as input, and returns a value in $[0, 1]$, where 1 means that the Quality of Experience (QoE) of the slice users is maximized. For the sake of simplicity, in this work, we consider only two slice classes, namely eMBB (e) and URLLC (u), which have almost complementary characteristics.

For what concerns the eMBB slice, we assume

$$F_e(\mathbf{r}, \hat{\mathbf{r}}) = \alpha_\delta f_e\left(\frac{\delta}{\hat{\delta}}\right) + \sum_{\rho \in \{\eta, c, m\}} \alpha_\rho f_e\left(\frac{\hat{\rho}}{\rho}\right), \quad (1)$$

where $\alpha_\eta, \alpha_c, \alpha_m$, and α_δ are non negative and add up to 1. Hence, $F_e(\cdot)$ is the convex sum of $f_e(x_\rho)$, where x_ρ is the level of fulfillment of the request, for $\rho \in \{\delta, \eta, c, m\}$.

Generally, $f_e(\cdot)$ can take a different shape for each type of resource as long as $f_e(x) = 1, \forall\, x \geq 1$, i.e., the performance is maximized anytime the allocated resources equals or exceeds the request. In this work, we consider

$$f_e(x) = \begin{cases} \beta_1 x + \beta_2 x^2 + \beta_3 x^3, & x \in [0, 1); \\ 1, & x \geq 1; \end{cases} \quad (2)$$

where, β_1, β_2 and β_3 are scalar parameters ensuring that $f_e(\cdot)$ is concave and monotonic increasing for $x \in [0, 1]$. The smooth shape of $f_e(\cdot)$ embodies the flexibility of the eMBB services, whose QoE degrades smoothly when the SLA is violated.

Conversely, URLLC flows have very strict requirements that, if infringed, cause the sudden degradation of the QoE. For this reason, to model the performance of this class of services, we consider the product of step functions:

$$F_u(\mathbf{r}, \hat{\mathbf{r}}) = f_u\left(\frac{\delta}{\hat{\delta}}\right) \times \prod_{\rho \in \{\eta, c, m\}} f_u\left(\frac{\hat{\rho}}{\rho}\right), \quad (3)$$

where

$$f_u(x) = \begin{cases} 0, & x \in [0, 1); \\ 1, & x \geq 1. \end{cases} \quad (4)$$

Given the functions $\{F_\sigma(\cdot)\}$ for all the slice classes $\sigma \in \Sigma$, the system utility is obtained as

$$\Omega = \frac{|\Phi_e|}{|\Phi|}\Omega_e + \frac{|\Phi_u|}{|\Phi|}\Omega_u, \quad (5)$$

where

$$\Omega_\sigma = \frac{1}{|\Phi_\sigma|} \sum_{\phi \in \Phi_\sigma} F_\sigma(\mathbf{r}_\phi, \hat{\mathbf{r}}_\phi), \ \sigma \in \Sigma, \quad (6)$$

and $|\cdot|$ is the cardinality operator. We observe that both $\{\Omega_\sigma\}$ and, consequently, Ω always take values in $[0, 1]$.

B. Network Model

In our model, we consider two different network elements, namely *node* and *link*, as detailed below.

- Nodes can be of two types: edge or core. *Edge nodes* are located at the network edge and connect users with the rest of the network. *Core nodes* are located in the core of the network and forward the aggregated data flows coming from the edge nodes. Each node n is equipped with a certain amount of computational (C^c_n) and memory (C^m_n) resources.
- A *link* is any connection l between two different nodes. This element is provided with a certain bit rate (B_l) to support the communications between the connected nodes.

From now on, we denote by $\mathcal{N}$ and $\mathcal{L}$ the set of network nodes and links, respectively.

Let Φ_l be the set of information flows that cross a certain link $l \in \mathcal{L}$. Each flow $\phi \in \Phi_l$ gets assigned a portion $b_{l,\phi}$ of the link bit rate B_l. Similarly, for each node $n \in \mathcal{N}$, let Φ_n be the set of information flows crossing the node, and let $c_{n,\phi}$ and $m_{n,\phi}$ be the amount of computational and storage resources assigned to flow $\phi \in \Phi_n$. Consequently, any resource allocation pattern must comply with the following feasibility conditions:

$$\sum_{\phi \in \Phi_l} b_{l,\phi} \leq B_l, \qquad \forall\, l \in \mathcal{L}; \quad (7)$$

$$\sum_{\phi \in \Phi_n} \rho_{n,\phi} \leq C^\rho_n, \qquad \rho \in \{c, m\}, \forall\, n \in \mathcal{N}. \quad (8)$$

The performance of a flow ϕ depends on the resource allocation vector $\hat{\mathbf{r}}_\phi$, which is the result of the allocations in

the network elements. Let $\mathcal{N}_\phi$ and $\mathcal{L}_\phi$ be the sets of network nodes and links crossed by ϕ, respectively. We assume that the computational and memory requests of a flow $\phi \in \Phi$ can be distributed among all the nodes in $\mathcal{N}_\phi$, so that

$$\hat{\rho}_\phi = \sum_{n \in \mathcal{N}_\phi} \rho_{n,\phi}, \quad \forall\ \rho \in \{c, m\}. \tag{9}$$

Hence, each slice can build its own virtual network functions using the resources provided by any of the servers with which it is associated.

The throughput $\hat{\eta}_\phi$, instead, is equal to the output rate from the last link along the routing path of ϕ. Let $b^o_{l,\phi}(t)$ be the output rate at time t from link $l \in \mathcal{L}$. The value of $b^o_{l,\phi}(t)$ is the minimum between the allocated rate $b_{l,\phi}(t)$ and the sum of the incoming and queued traffic, i.e.,

$$b^o_{l,\phi}(t) = \min\left\{ b_{l,\phi}(t), \frac{D_{l,\phi}(t-1)}{T} + b^i_{l,\phi}(t) \right\}, \tag{10}$$

where $b^i_{l,\phi}(t)$ is the input rate from the upstream link and $D_{l,\phi}(t-1)$ is the amount of data of ϕ queued at node n at the end of the previous slot. In particular, $b^i_{l,\phi}(t)$ is given by

$$b^i_{l,\phi}(t) = \begin{cases} b^o_{\ell,\phi}(t), & \text{if } \ell \text{ is the upstream link of } l \text{ in } \mathcal{L}_\phi; \\ \eta_\phi(t), & \text{if } l \text{ is the first link in } \mathcal{L}_\phi. \end{cases} \tag{11}$$

Instead, $D_{l,\phi}(t)$ is set to 0 for any time t before the initialization of the flow, and then it is updated as

$$D_{l,\phi}(t) = \max\left\{0, D_{l,\phi}(t-1) + T(b^i_{l,\phi}(t) - b_{l,\phi}(t))\right\}. \tag{12}$$

Note that the value of $D_{l,\phi}$ increases as the assigned rate $b_{l,\phi}$ is lower than the input rate $b^i_{l,\phi}$.

For what concerns the delay experienced by ϕ, we have

$$\hat{\delta}_\phi(t) = \sum_{n \in \mathcal{N}_\phi} \tau_n + \sum_{l \in \mathcal{L}_\phi} \tau_{l,\phi}(t), \tag{13}$$

where τ_n is a positive value representing the delay due to routing operations at node n, which is assumed constant over time. Instead, $\tau_{l,\phi}(t)$ is computed as

$$\tau_{l,\phi}(t) = \tau^q_{l,\phi}(t) + \tau^\tau_{l,\phi}(t) + \tau^p_l, \tag{14}$$

where τ^p_l is the constant propagation delay of ϕ through link l, $\tau^q_{l,\phi}(t)$ is the average queuing time in l during t, given by

$$\tau^q_{l,\phi}(t) = \frac{2D_{l,\phi}(t-1) - T(b^o_{l,\phi}(t) - b^i_{l,\phi}(t))}{2b_{l,\phi}(t)}, \tag{15}$$

and $\tau^\tau_{l,\phi}(t)$ is the transmission delay, which is inversely proportional to the output rate, i.e.,

$$\tau^t_{l,\phi}(t) = \frac{1}{b^o_{l,\phi}(t)}. \tag{16}$$

We highlight that, despite we consider a discrete time-frame, $\hat{\delta}_\phi$ is a continuous value.

Our aim is to determine the resource allocation to maximize the system utility as given in (5). Mathematically, we want to determine $\hat{\mathbf{r}}_\phi$, $b_{l,\phi}$, $c_{n,\phi}$, $m_{n,\phi}$, $b^i_{l,\phi}$, $b^o_{l,\phi}$, $\tau^q_{l,\phi}$, and $\tau^\tau_{l,\phi}$, $\forall\ \phi \in \Phi$, $n \in \mathcal{N}$, $l \in \mathcal{L}$, that maximize Ω, under the constraints given in (7)-(16). If information flow demands evolve too quickly, conventional techniques are unable to find a solution within a reasonable time frame. Instead, DRL approaches can overcome such problem, and provide high-performance solutions in a relatively short time.

IV. Learning Strategy

In order to efficiently orchestrate communication resources in the NS scenario described in Sec. III, we develop a distributed DRL strategy. The designed system is based on multiple learning units, named *local controllers*, that collaborate to maximize the overall utility, given by (5). To evaluate the performance of the proposed approach, we design an empirical resource allocation algorithm, whose behavior is described at the end of this section.

A. Learning Architecture

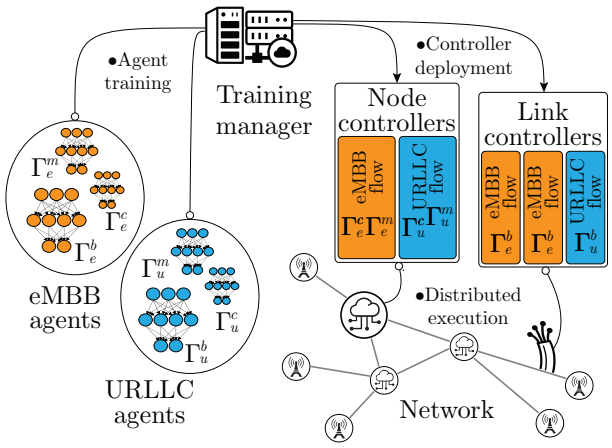


Fig. 1: Learning Architecture.

Our learning architecture (shown in Fig. 1) entails a different controller for each information flow and network element. From an operational point of view, the total number of local controllers depends on the network topology and the cardinality of Φ. Practically, the local controllers are all replicas of $3 \times |\Sigma|$ learning agents. Indeed, we train a different agent for each of the three type of network resources, namely bit rate, computational power and memory, and for each of the $|\Sigma|$ different slice classes. Given a slice class $\sigma \in \Sigma$, we denote by $(\Gamma^b_\sigma, \Gamma^c_\sigma, \Gamma^m_\sigma)$ the tuple of agents that σ is associates with. The agent Γ^b_σ orchestrates the bit rate assignments for each information flow $\phi \in \Phi_\sigma$ in each link $l \in \mathcal{L}$; instead, Γ^c_σ and Γ^m_σ orchestrate the computation and memory resources for each information flow $\phi \in \Phi_\sigma$ in each node $n \in \mathcal{N}$. Practically, the agent training is performed by a central entity, named *training manager*, which collect the system information and update the learning architecture accordingly. Then, copies of Γ^b_σ, Γ^c_σ, and Γ^m_σ will be deployed in each network element crossed by any flow $\phi \in \Phi_\sigma$.

In the training phase, we leverage the A2C algorithm, which makes it possible to handle a continuous state space, providing

stable DRL solutions in very complex scenarios [21]. According to this approach, each learning agent is composed of two units, the *actor* and the *critic*, which are implemented by means of NNs. Particularly, we consider an architecture with two hidden layers and the Rectifier Linear Unit (ReLU) as activation function [22]. The output of the actor is the amount ρ^* of resources demanded by the local controller, while that of the critic is the expected future reward. The size of the NN input varies according to the type of resources that has to be managed, as explained next.

B. State Space

In our system, each local controller has full knowledge of the element where it runs, while it has a limited view on the rest of the network, which implies that the system state is only partially observable [23]. Let us consider a local controller managing the rate resources of a flow ϕ in a link l. At the beginning of each timeslot, such a controller is provided with two vectors representing the status of the information flow and the network element that is associated to.

The first vector gives the state of ϕ at the beginning of timeslot t:

$$\mathbf{s}_\phi(t) = [\mathbf{r}_\phi(t), \hat{\mathbf{r}}_\phi(t-1)], \tag{17}$$

where $\mathbf{r}_\phi(t)$ and $\hat{\mathbf{r}}_\phi(t-1)$, defined in Sec. III, are the resources requested and granted by ϕ at the beginning of timeslot t and $t-1$, respectively. We observe that $\hat{\mathbf{r}}_\phi(t-1)$ can be computed only knowing the aggregate amount of network resources assigned to ϕ by the network elements of its routing path. Therefore, our architecture can properly operate only if $\mathbf{s}_\phi(t)$ is shared among all the controllers assigned to ϕ at the beginning of each timeslot t. However, the size of $\mathbf{s}_\phi(t)$ is negligible with respect to the rate requirements of the slices, and, therefore, can be transmitted within the user data plane of ϕ, without degrading the performance of our system.

The second vector is

$$\begin{aligned}\mathbf{s}_{l,\phi}(t) = [&B_l, \tau_{l,\phi}(t-1), D_{l,\phi}(t-1), b^*_{l,\phi}(t-1),\\ &b_{l,\phi}(t-1), b^e_l(t-1), b^u_l(t-1)],\end{aligned} \tag{18}$$

which provides the state of the rate resources of ϕ in l at the beginning of timeslot t. More specifically, in (18), $b^\sigma_l(t-1)$ is the aggregate rate demanded in l by all the flows of class σ during timeslot $t-1$, while the other parameters were defined in Sec. III. Hence, the link controller takes $\mathbf{s}_\phi(t)$ and $\mathbf{s}_{l,\phi}(t)$ as input and returns $b^*_{l,\phi}(t)$, which is the bit rate demanded by ϕ in l during t.

When considering a controller associated to a node n and a flow ϕ we use the same approach and, depending on the resource $\rho \in \{c, m\}$ to be allocated, the input (18) is replaced with the states of the computation and memory resources assigned to ϕ by node n:

$$\begin{aligned}\mathbf{s}^\rho_{n,\phi}(t) = [&C^\rho_n, \rho^*_{n,\phi}(t-1), \rho_{n,\phi}(t-1),\\ &\rho^e_n(t-1), \rho^u_n(t-1)],\end{aligned} \tag{19}$$

where $\rho^\sigma_n(t-1)$ is the aggregate amount of resource ρ demanded in node n by all the flows of class σ during timeslot $t-1$. At the beginning of timeslot t, the node controller takes $\mathbf{s}_\phi(t)$ and $\mathbf{s}^\rho_{n,\phi}(t)$ as input and returns $\rho^*_{n,\phi}(t)$, with $\rho \in \{c, m\}$; this latter is the computational (or memory) capacity demanded by ϕ in n during t.

C. Reward Function

In accordance to the RL paradigm, we need to define a reward function $r(\cdot)$ that represents the benefit generated by each possible state-action pair of the policy. In particular, to maximize the overall utility, each local controller should demand enough resources to maximize the performance of the flow ϕ it is in charge of, without subtracting too many resources to the other flows. In our system, given a controller associated to a link $l \in \mathcal{L}$ and a flow $\phi \in \Phi_l$, the reward is given by

$$\begin{aligned}r_{l,\phi}(t) = &\gamma_0 \left(f_{\sigma_\phi}\left(\frac{\eta_\phi(t)}{\hat{\eta}_\phi(t)}\right) + f_{\sigma_\phi}\left(\frac{\hat{\delta}_\phi(t)}{\delta_\phi(t)}\right)\right) +\\ &\frac{\gamma_1}{|\Phi_l|}\sum_{\psi\in\Phi_l}\left(f_{\sigma_\psi}\left(\frac{\eta_\psi(t)}{\hat{\eta}_\psi(t)}\right) + f_{\sigma_\psi}\left(\frac{\hat{\delta}_\psi(t)}{\delta_\psi(t)}\right)\right),\end{aligned} \tag{20}$$

where σ_ϕ is the slice class that ϕ belongs to, $\gamma_0 > 0$ weights the performance of flow ϕ, while $\gamma_1 > 0$ weights the average performance of all the flows crossing link l.

Similarly, a controller assigned to a node n and a flow ϕ is rewarded according to

$$r^\rho_{n,\phi}(t) = \gamma_0 f_{\sigma_\phi}\left(\frac{\rho_\phi(t)}{\hat{\rho}_\phi(t)}\right) + \frac{\gamma_1}{|\Phi_n|}\sum_{\psi\in\Phi_n} f_{\sigma_\psi}\left(\frac{\rho_\psi(t)}{\hat{\rho}_\psi(t)}\right), \tag{21}$$

where $\rho \in \{c, m\}$.

Under such constraints, if a link controller uses all the bit rate available in l for its own flow ϕ, all the flows in $\Phi_l \backslash \{\phi\}$ experience degraded performance and, consequently, the reward $r_{l,\phi}(t)$ decreases. Particularly, a local controller receives the highest reward when all the competing information flows experience the maximum performance.

D. Empirical Algorithm

Besides the DRL strategy, we introduce an empirical algorithm that does not require any training and, therefore, can potentially be more efficient than the learning approach, though less flexible. With the empirical algorithm, each flow tries to acquire a sufficient amount of rate to both satisfy the current throughput requirement and empty the buffer of any previous data. Therefore, at the beginning of timeslot t, each flow ϕ crossing link l demands an amount of rate equal to $b^*_{l,\phi}(t) = \eta_\phi(t) + D_{l,\phi}(t-1)/T$.

The computation and memory resources of a flow ϕ, instead, are distributed among all the nodes $\{n\}$ crossed by ϕ, proportionally to the element capacities. In particular, each flow ϕ crossing node n demands an amount of resources equal to $\rho^*_{n,\phi}(t) = \chi_{n,\phi}\rho_\phi(t)$, where $\chi_{n,\phi} = C^\rho_n / \left(\sum_{k\in\mathcal{N}_\phi} C^\rho_k\right)$ and $\rho \in \{c, m\}$. We highlight that, to compute $\chi_{n,\phi}$, it is necessary to known the computation and storage capacities of each node $n \in \mathcal{N}_\phi$, an information that is not available to the

local controllers of the DRL strategy and, hence, might give some advantage to the empirical algorithm.

At this stage of the work, we have not implemented any admission strategy. Therefore, both using the empirical and the DRL approaches, some network elements may not be able to satisfy all the requests they receives. Practically, the resources assigned to any flow ϕ in a link $l \in \mathcal{L}_\phi$ or a node $n \in \mathcal{N}_\phi$ are computed as

$$b_{l,\phi}(t) = b^*_{l,\phi}(t) \min\left\{1, \frac{B_l}{\sum_{\psi \in \Phi_l} b^*_{l,\psi}(t)}\right\}, \quad (22)$$

$$\rho_{n,\phi}(t) = \rho^*_{n,\phi}(t) \min\left\{1, \frac{C^\rho_n}{\sum_{\psi \in \Phi_n} \rho^*_{n,\psi}(t)}\right\}, \quad (23)$$

where $\rho \in \{c, m\}$.

V. Scenarios and Results

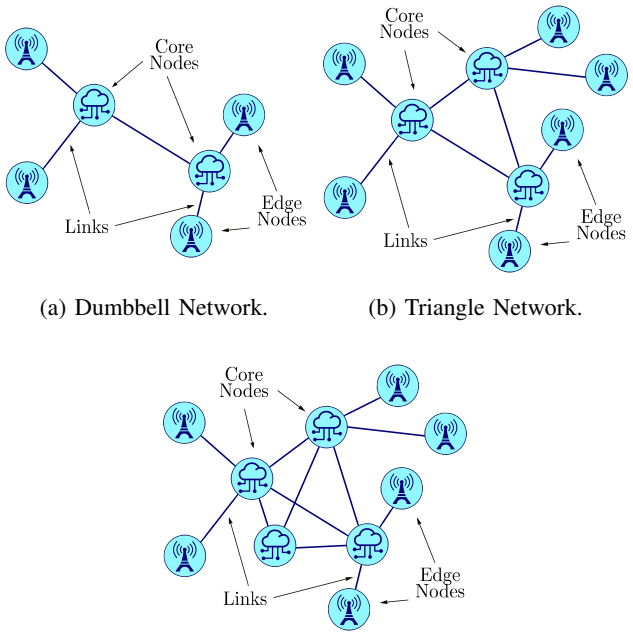
In this section, we first describe the settings of our simulations as well as the testing scenarios where our model is implemented. Then, we investigate the performance of the proposed DRL strategy under different working conditions.

A. Scenarios

We consider three different scenarios, named *Dumbbell*, *Triangle* and *Pyramid Networks*, whose topologies are reported in Fig. 2. In all the cases, the number of information flows in the network (i.e., the cardinality of Φ) is in $\{2, ..., 6\}$. The capacities of each network element are fixed; particularly, we set $C^c_n = 60$ Gbps and $C^m_n = 60$ Gb for core nodes, $C^c_n = 20$ Gbps and $C^m_n = 20$ Gb for edge nodes, $B_l = 50$ Gbps for links. Finally, we assume that $\tau^p_l = 0.1$ ms $\forall\ l \in \mathcal{L}$, and $\tau_n = 0.001$ ms $\forall\ n \in \mathcal{N}$. Although arbitrary, these values are well-aligned to the features of modern network elements.

To model the information flow requirements, we consider a different Markov Model (MM) [24] M_σ with transition probability matrix $\mathbf{P}_\sigma$ for each class $\sigma \in \Sigma$. All the models include $N_{\text{state}} = 10$ states that represent combinations of resource requirements, i.e., different realizations of the vector $\mathbf{r}$. Hence, each information flow $\phi \in \Phi_\sigma$ is associated to an independent copy of M_σ that changes state at each timeslot t, thus varying $\mathbf{r}_\phi$. In particular, the matrices $\{\mathbf{P}_\sigma\}$ are designed in such a way that transitions can only occur between adjacent states. The minimum and maximum value of the resource requirements are summarized in Tab. II; a reference for the considered values can be found in [1].

To train the learning agents, we generate $N_{\text{train}} = 5 \cdot 10^4$ independent episodes using the same network topology. Each episode lasts $N_{\text{slot}} = 50$ timeslots of $T = 0.1$ seconds. At the beginning of each episode, a random number of information flows is generated. Hence, each flow ϕ is associated with a static route, which is randomly selected from the possible direct paths between the flow endpoints Then, the agents Γ^b_σ, Γ^c_σ, Γ^m_σ, $\forall\ \sigma \in \Sigma$, distribute network resources according to the slice requests, while the A2C algorithm is used to train the system. In particular, we exploit the Adaptive Moment Estimation algorithm to optimize the NN weights, considering $\zeta_a = 3 \cdot 10^{-5}$ and $\zeta_c = 3 \cdot 10^{-5}$ as learning rates of the actor and the critic, respectively [26]. The main settings of the agent architectures are summarized in Tab. III.



(a) Dumbbell Network. (b) Triangle Network.

(c) Pyramid Network.

Fig. 2: Network topologies.

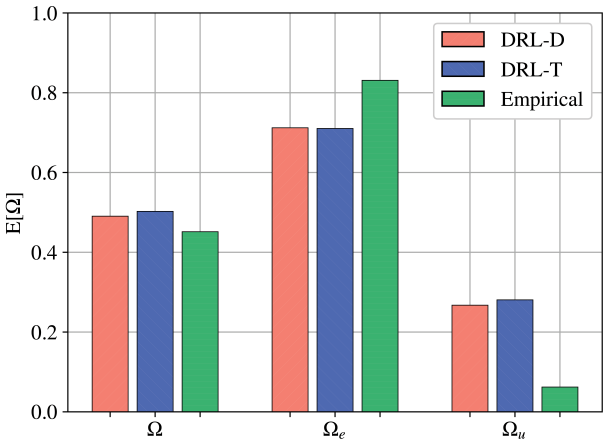
TABLE II: Resource requirements.

Service class	Parameter	Range of values	Unit
eMBB	η	$0.30 \div 42.5$	Gbps
	c	$50 \div 100$	Gbps
	m	$50 \div 100$	Gb
	δ	20	ms
URLLC	η	$2.08 \div 10$	Gbps
	c	$50 \div 100$	Gbps
	m	$50 \div 100$	Gb
	δ	1	ms

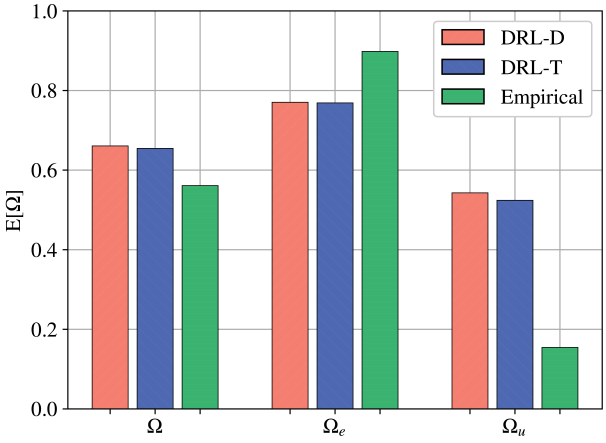
TABLE III: Local controller design.

	Γ^b_σ		Γ^c_σ		Γ^m_σ	
Parameter	Actor	Critic	Actor	Critic	Actor	Critic
Input size	11	11	7	7	7	7
Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
Hidden size	12	12	8	8	8	8
Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
Hidden size	6	6	4	4	4	4
Activation	Linear	Linear	Linear	Linear	Linear	Linear
Output size	1	1	1	1	1	1

To be noted that, in our simulations, the number of flows and their requirements change randomly, so that the aggregate resource requests can exceed the capacity of the network. In such conditions, the allocation strategy should decide which flow to penalize, in order to maximize the overall utility.



(a) Dumbbell Network.



(b) Triangle Network.

Fig. 3: Expected utility.

Therefore, our system can be exploited to handle critical scenarios where there is a lack of network resources, or to estimate the reliability of a specific set of network slices.

B. Results

We consider two versions of our learning system, one trained in the Dumbbell Network (DRL-D) and the other in the Triangle Network (DRL-T). In Fig. 3, we report the expected performance achieved by each slice class, and by the whole system (see (5) and (6)), in the Dumbbell and Triangle Network, respectively. In both the scenarios, the empirical algorithm yields worse performance since it tends to favor the eMBB class at the expenses of URLLC flows. Instead, the DRL strategies double the fraction of satisfied URLLC flows, with only a small degradation in the eMBB services.

In Fig. 4 we show the distribution of the system performance for the analyzed strategies, in all the considered network topologies. To do so, we adopt the boxplot representation, where the white line at the box center is the median, the box edges are the 25th and the 75th percentile, while the whiskers are the 5th and the 95th percentile. The DRL strategies always outperform the empirical one, both considering the median and the percentiles of Ω. We observe that, in the Pyramid Network, the lack of network resources is less striking and, consequently, the performance of all the strategies increases. In particular, using the empirical algorithm, 75% of the test episodes experience $\Omega > 0.35$; this threshold is raised to 0.5 with the DRL strategies, while the median of Ω is improved by more than 25%. We highlight that DRL-D and DRL-T yield higher performance in all the testing scenarios. In particular, the learning strategies succeed in addressing slice requirements also in the Pyramid Network, which is a different environment from those seen during the training phase.

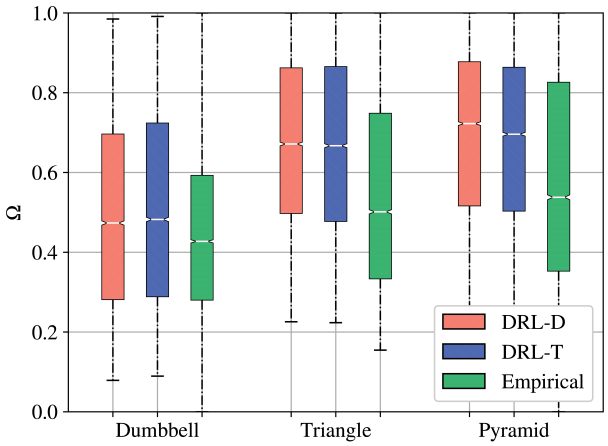


Fig. 4: Utility distribution.

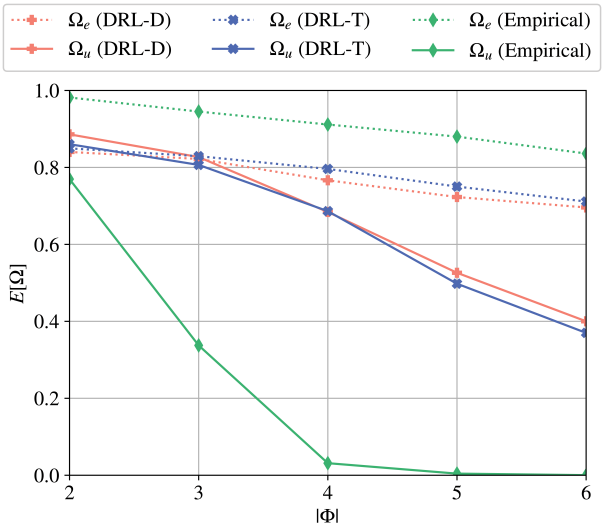


Fig. 5: Expected utility vs flow number (Pyramid Network).

Finally, in Fig. 5, we plot the utility of the eMBB and URLLC slices (i.e., Ω_e and Ω_u) in the Pyramid Network scenario as a function of the number of active information flows. When considering the eMBB service, all the allocation strategies have a similar behavior: $E[\Omega_e]$ decreases smoothly as the cardinality of Φ increases. If we use DRL-D or DRL-T,

$E[\Omega_u]$ keeps a linear trend, although its value decreases much faster than before. In contrast, using the empirical algorithm, the utility of the URLLC flows follows an exponential decay and, for $|\Phi| \geq 4$, $E[\Omega_u]$ approaches 0. Therefore, if the number of information flows is limited, it is sufficient to adopt an empirical approach to orchestrate the network resources. At the same time, DRL techniques, like the one we propose, can considerably improve the overall performance when the target scenario gets more complex.

VI. Conclusion

In this work, we investigated the potentials of DRL to orchestrate network resources in a NS scenario. Specifically, we developed a distributed DRL strategy where different learning units interact to meet the resource demands of multiple information flows. We showed that the designed system allows to better optimize the management of network resources, especially for more complex systems, both in terms of network topology and service heterogeneity. Besides, our system proved to be highly flexible since it can suit multiple network topologies without the need for additional training.

References

[1] 3GPP, "Service requirements for next generation new services and markets," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 22.261, March 2020, version 17.2.0.

[2] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi, "5G wireless network slicing for eMBB, URLLC, and mMTC: A communication-theoretic view," *IEEE Access*, vol. 6, pp. 55 765–55 779, September 2018.

[3] P. Caballero, A. Banchs, G. de Veciana, and X. Costa-Pérez, "Multi-tenant radio access network slicing: Statistical multiplexing of spatial loads," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 3044–3058, July 2017.

[4] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, March 2018.

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

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

[7] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a GPU," *arXiv e-prints*, pp. arXiv–1611, 2016.

[8] H. Halabian, "Distributed resource allocation optimization in 5G virtualized networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 627–642, 2019.

[9] M. Leconte, G. S. Paschos, P. Mertikopoulos, and U. C. Kozat, "A resource allocation framework for network slicing," in *IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2177–2185.

[10] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," 2011.

[11] Y. Xiao, M. Hirzallah, and M. Krunz, "Distributed resource allocation for network slicing over licensed and unlicensed bands," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2260–2274, 2018.

[12] F. Fossati, S. Moretti, P. Perny, and S. Secci, "Multi-resource allocation for network slicing," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1311–1324, 2020.

[13] R. Alvizu, S. Troia, G. Maier, and A. Pattavina, "Matheuristic with machine-learning-based prediction for software-defined mobile metro-core networks," *Journal of Optical Communications and Networking*, vol. 9, no. 9, pp. 19–30, 2017.

[14] Y. Hua, R. Li, Z. Zhao, X. Chen, and H. Zhang, "GAN-powered deep distributional reinforcement learning for resource management in network slicing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 334–349, 2020.

[15] N. Van Huynh, D. Thai Hoang, D. N. Nguyen, and E. Dutkiewicz, "Optimal and fast real-time resource slicing with deep dueling neural networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1455–1470, 2019.

[16] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[17] J. A. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, A. Banchs, and J. J. Alcaraz, "vrAIn: A deep learning approach tailoring computing and radio resources in virtualized RANs," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.

[18] J. S. P. Roig, D. M. Gutierrez-Estevez, and D. Gündüz, "Management and orchestration of virtual network functions via deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 304–317, 2019.

[19] Y. Abiko, T. Saito, D. Ikeda, K. Ohta, T. Mizuno, and H. Mineno, "Flexible resource block allocation to multiple slices for radio access network slicing using deep reinforcement learning," *IEEE Access*, vol. 8, pp. 68 183–68 198, 2020.

[20] M. Alsenwi, N. H. Tran, M. Bennis, S. R. Pandey, A. K. Bairagi, and C. S. Hong, "Intelligent resource slicing for eMBB and URLLC coexistence in 5G and beyond: A deep reinforcement learning based approach," *arXiv preprint arXiv:2003.07651*, 2020.

[21] 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 *International conference on machine learning*, February 2016, pp. 1928–1937.

[22] A. K. Dubey and V. Jain, "Comparative study of convolution neural network's ReLU and leaky-ReLU activation functions," in *Applications of Computing, Automation and Wireless Systems in Electrical Engineering*. Springer, 2019, pp. 873–880.

[23] T. Jaakkola, S. P. Singh, and M. I. Jordan, "Reinforcement learning algorithm for partially observable Markov decision problems," in *Advances in neural information processing systems*, 1995, pp. 345–352.

[24] C. J. Geyer, "Practical Markov chain Monte Carlo," *Statistical science*, pp. 473–483, 1992.

[25] 3GPP, "System architecture for the 5G System (5GS)," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 23.501, March 2020, version 16.4.0.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.