AI-Based Network-Aware Service Function Chain Migration in 5G and Beyond Networks

Rami Akrem Addad, *Graduate Student Member, IEEE*, Diego Leonel Cadette Dutra, Tarik Taleb, and Hannu Flinck

***Abstract*—While the 5G network technology is maturing and the number of commercial deployments is growing, the focus of the networking community is shifting to services and service delivery. 5G networks are designed to be a common platform for very distinct services with different characteristics. Network Slicing has been developed to offer service isolation between the different network offerings. Cloud-native services that are composed of a set of inter-dependent micro-services are assigned into their respective slices that usually span multiple service areas, network domains, and multiple data centers. Due to mobility events caused by moving end-users, slices with their assigned resources and services need to be re-scoped and re-provisioned. This leads to slice mobility whereby a slice moves between service areas and whereby the inter-dependent service and resources must be migrated to reduce system overhead and to ensure low-communication latency by following end-user mobility patterns. Recent advances in computational hardware, Artificial Intelligence, and Machine Learning have attracted interest within the communication community to study and experiment self-managed network slices. However, migrating a service instance of a slice remains an open and challenging process, given the needed co-ordination between inter-cloud resources, the dynamics, and constraints of inter-data center networks. For this purpose, we introduce a Deep Reinforcement Learning based agent that is using two different algorithms to optimize bandwidth allocations as well as to adjust the network usage to minimize slice migration overhead. We show that this approach results in significantly improved Quality of Experience. To validate our approach, we evaluate the agent under different configurations and in real-world settings and present the results.**

***Index Terms*—5G and beyond, service function chain, artificial intelligence, machine learning, multi-access edge computing, deep reinforcement learning, and management and orchestration.**

I. Introduction

THE LATEST 5G system architecture has been designed to be compatible with cloud technology. Control plane network functions follow a service-based approach that builds on Network Function Virtualization (NFV) and Software-Defined Networking (SDN) resulting in a well-scalable control plane [1]. The initial service offerings of the 5G network are classified through three major service types, namely enhanced Mobile Broadband (eMBB), Ultra-Reliable and Low-Latency Communications (URLLC), and massive Machine-Type Communications (mMTC) [2]. The services are also expected to be cloud-based, sharing the same computing and network infrastructure. To ensure isolation between these three different service types, the concept of Network Slicing (NS) [3], which is one of the key features of 5G networks, has been introduced. Furthermore, the use of Multi-Access Edge Clouds (MEC) [4] enables to bring latency-sensitive services closer to the end-users. Cloud-based services and network functions are typically implemented as Service Function Chains (SFC) [5] of dependent micro-services that collectively provide an end-to-end service. For example, one micro-service is in charge of authentication, one for data encoding, and another for the application business logic. Such SFCs are then associated with corresponding NSs that are consumed by the end-users.

From a resource point of view, a NS instance contains the end-to-end services and their respective computing, storage resources, network resources such as radio resources, and the resources connecting the computing and storage resources that may be distributed between edges and central data centers. The resource consumption demands depend on the service type, the number of service users, end-user devices, and users' locations. When the users of a service move from a place to another, the resource consumption load changes based on the length and the number of users sharing the communication path to the servers providing the service. At some point, the original resource allocation of an NS needs to be remapped to match the actual consumption and latency situation, leading to the notion of NS mobility where the network connectivity and the servers are re-provisioned.

To support NS mobility to reduce system overhead and allow low communication latency, we have proposed SFCs

Manuscript received July 7, 2020; revised October 29, 2020 and February 7, 2021; accepted February 9, 2021. Date of publication April 21, 2021; date of current version March 11, 2022. This research work is partially supported by the European Union's Horizon 2020 ICT Cloud Computing program under the ACCORDION project with grant agreement No. 871793 and by the European Union's Horizon 2020 research and innovation program under the CHARITY project with grant agreement No. 101016509. It is also partially funded by the Academy of Finland Project 6Genesis under grant agreements No. 318927. The associate editor coordinating the review of this article and approving it for publication was J. Zhao. *(Corresponding author: Rami Akrem Addad.)*

Rami Akrem Addad is with the Department of Communications and Networking, Aalto University, 02150 Espoo, Finland (e-mail: rami.addad@aalto.fi).

Diego Leonel Cadette Dutra is with the PESC, Federal University of Rio de Janeiro, Rio de Janeiro 21941-972, Brazil (e-mail: diegodutra@lcp.coppe.ufrj.br).

Tarik Taleb is with COMNET, Aalto University, Espoo 02710, Finland, also with the Centre for Wireless Communications, University of Oulu, 90570 Oulu, Finland, and also with the Computer and Information Security Department, Sejong University, Seoul 05006, South Korea.

Hannu Flinck is with Nokia Bell Labs, 02610 Espoo, Finland (e-mail: hannu.flinck@nokia-bell-labs.com).

Digital Object Identifier 10.1109/TNSM.2021.3074618

migration approaches for 5G networks [6]. SFC migration, through the usage of the SDN and NFV paradigms, steers the network traffic and flows across multiple physical and logical infrastructures while ensuring low-latency communication by following end-users' motion [5], [6]. Live service migration processes, as underlying technologies and enablers of SFC migration strategies, are known to be challenging in inter-cloud settings. The use of chained micro-services is adding its own challenges due to the dependency of the chained micro-services. A prime example is the considerable amount of network resource usage to enable the reshuffling of the virtualized instances (live/cold migration) [7]. Moreover, in 5G and beyond networks, it is expected that the number of URLLC type services that require strict delay constraints will increase as 5G networks become more widely deployed [8]. This emphasizes the importance of careful management of the end-to-end service delivery. To summarize, end-user mobility events may trigger a large number of application migrations concurrently, hence exhausting the network resources shared between the distributed edge clouds and their servers.

Recently, both academia and industry have increased their interest in Machine Learning (ML) methods, a subarea of Artificial Intelligence (AI) [9]. The use of ML in mobile networking has been also evaluated [10]. Furthermore, ML will automatically perform corrective re-configurations to the infrastructure, ensuring the availability of the network [11]. ML techniques will also apply efficient policies leading to the optimization of the system resources; hence, enabling efficient use of critical network and service resources, e.g., latency and bandwidth, and system resources, e.g., RAM, CPU, DISK, and I/O, [12]. ML techniques will be part of the networking and communication area as different research projects, proposals, and white papers have indicated [13], [14]. However, the integration of ML methods into telecommunication's standards and edge computing architecture is still embryonic [15]. To address this concern, more research on applying ML methods in 5G and beyond networks is necessary. AI/ML will act as a support for enabling smarter and more responsive generation of networks while maintaining the currently proposed architectures by the standards community, e.g., ETSI and 3GPP [1], [16].

Motivated by the limitations triggered by unpredictable end-users mobility patterns, the complexity that the current 5G service delivery faces fulfilling inter-cloud bandwidth constraints, the stochastic, i.e., non-deterministic, use of networking resources during mobility events, and by the recent advances that ML techniques bring to edge computing and next-generation networking, we propose a Deep Reinforcement Learning (DRL)-based framework that enables efficient resource allocation mechanisms. DRL methods allow complex decision-making without explicit knowledge of all underlying network elements and internal architectures, as it considers them as black boxes [17]. Moreover, an appropriate algorithm selection, hyper-parameters tuning, and well-defined neural network architecture are required for obtaining credible mapping from system states to control actions. Following the above observations and network constraints, in this work, we intend to:

- Introduce our envisioned architecture hosting the proposed network-aware agent and its constituent elements;
- Model and design a DRL-based agent capable of handling bandwidth allocation as well as refining the network usage to reduce the overhead and allow better users' QoE;
- Present the internal operational mechanisms of our network agent, neural network architectures used by the two different DRL-based algorithms constituting our network agent, and their hyper-parameters values;
- Evaluate the proposed agent under different configurations and in real-world deployments while trying to determine the most suitable DRL algorithm/approach to enable an optimized SFC migration pattern within the 5G network.

The remaining of this paper is organized as follows. Section II outlines the related work. In Section III, we present a background overview of RL and the various algorithms proposed in the preceding research. We also detail the system model and the design of our envisaged architecture used for bandwidth allocation. Section IV presents a detailed overview concerning the design of our agent as well as the different neural networks' parameters and architectures followed in both proposed algorithms. In Section V, we present and discuss the results of our experimental evaluation. Finally, we conclude the paper and introduce future research challenges in Section VI.

II. Related Work

The authors of [18] proposed a solution to overcome limitations, i.e., service disruptions and inadequate users' QoE, faced when performing live migration over Wide Area Networks (WANs). They proposed a system named "ReSeT" that predicts migration time and downtime utilizing a Linear Regression (LR) technique, thus reducing them drastically. During this work, the authors tested various metrics' combinations, e.g., CPU & number of client, CPU & network speed, and network speed & number of clients. They finally adopted the combination of network speed & number of client metrics as the ideal pair of parameters for predicting best downtime and total migration time through a series of evaluation tests. Knowing the dynamic aspect of 5G and beyond networks, considering only network speed in the prediction model while ignoring the bandwidth usage, will guarantee neither short service disruptions nor users' satisfaction.

De Vita *et al.* [19] proposed a solution for the data migration problem to improve users' QoS in a MEC environment. The authors leveraged DRL to build a self-adaptive algorithm capable of understanding the MEC nodes' status and accordingly migrating users' application data. To select the optimal policy for determining the migration time, the authors used users' positions and the current state of the network architecture. The authors validated their proposed model using OMNeT++/SimuLTE [20] integrated with the Keras ML framework to simulate the MEC environment [21]. However, they disregarded the network resources in their

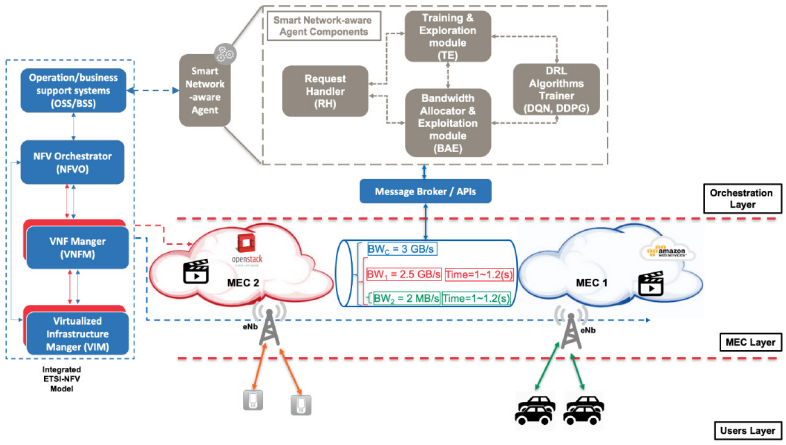


Fig. 1. Framework architecture for ML-based orchestration and allocation of bandwidth resources.

modeling, which introduces uncertainty on their solution in scenarios with a large number of mobile users.

The authors of [22] designed and implemented a network-aware Virtual Machine (VM) migration scheme in cloud data-center environments. Their proposal dynamically relocates VMs across nodes while minimizing the generated traffic flow. The proposed approach was integrated into Xen-based virtualization systems and evaluated on a practical testbed with a 78% communication cost reduction. Stage and Setzer [23] provided a comprehensive approach for scheduling VM migrations while considering both bandwidth and network topology requirements. The practical implementation is integrated with a commercial data center provider. Aiming to reduce completion time and reduce network overhead in multiple VM migrations scenarios, the authors of [24] introduced a scheduling method. Their approach starts by analyzing migration outputs, then discovers suitable shared bandwidth resources for parallel migration operations. Evaluation results showed the efficiency of the proposed scheduling scheme. Despite the efficiency of the previously cited works, 5G and beyond networks will rely on dynamic and adaptive solutions to handle networking limitations.

Addad *et al.* [6] introduced a method to handle all Virtualized Network Functions (VNFs) chain migrations, i.e., under the name of SFC migrations. The proposed approach considers both the synchronization of the VNF chain's instances and network resources consumption. The authors proposed to refine the network usage by controlling the network's bandwidth while executing SFC migrations. Although their approach offers a downtime transfer similar to when having the full utilization of the bandwidth, it was based on a brute force method. With the high dynamic workloads of 5G applications, this approach becomes rapidly inefficient.

Duggan *et al.* [25] elaborated a method to diminish network resources' consumption generated by the VM migrations. They used an RL agent to develop an autonomous network-aware VM migration strategy. The authors monitored the network's demands and let their proposed agent accordingly schedule the optimal VMs migrations' decisions. They evaluated their proposal with a simulated cloud environment. The proposed approach based on RL also has practical limits as in complex and large-scale networks, the state and action spaces are usually large. Therefore, RL may neither find the optimal policy in a reasonable time, i.e., scalability issues, nor be capable of representing the colossal number of states in a computer's memory.

The authors in [26] designed, modeled, and evaluated two DRL-based algorithms for allowing a fine-grained selection of system-based triggers regarding network slice mobility patterns. The work constituted an effort towards making their defined triggers intelligent while maintaining system resources stable. Nevertheless, the proposed solution does not ensure reduced network resource overhead. Clearly, the proposed approach requires complementary proposals guaranteeing network resource stability, as aimed for in this paper.

In comparison to the literature, this present work aims to develop a network-aware agent capable of selecting accurate bandwidth values while ensuring fast and reliable service migration to address a large number of emerging use-cases requiring strict requirements.

III. Proposed Architecture & System Model

A. Envisioned Architecture

The emergence of new vertical industries such as automotive, e-health, and public safety on top of 5G infrastructure expects low-latency communication. Besides, it is expected that mMTC applications change the network requirements in terms of the number of endpoints and the number of connections per device/user. Indeed, these stringent requirements and standards make the availability of network resources critical since all 5G services assume a reliable networking system. Hence, there is an urgent need for an ML-based agent able to refine network usage by allowing a fine-grained bandwidth allocation process for SFC or massive inter-correlated service migration workflows.

To achieve this goal, we adopt a conventional three-layer architecture widely used for representing 5G and beyond network systems. The proposed system, depicted in Fig. 1, complies with ETSI-NFV standards. In the defined system, the MEC layer is controlled through the interaction between the

components of the Orchestration layer and the elements constituting the NFV architecture [27]. We explicitly omit several components of the Orchestration layer to focus on the Smart Network-Aware (SN-A) agent that is supposed to fine-tune the bandwidth allocation process.

The Request Handler (RH) module offers to the SN-A agent a technology-agnostic abstraction to access MEC-layer entities, i.e., public or private cloud platforms. Therefore, the SN-A agent retrieves states, accordingly outputs decisions of bandwidth values, and receives rewards for each decision. The SN-A agent also receives administrative instructions from the Operation/Business Support Systems (OSS/BSS) as defined in the ETSI-NFV model. The RH module must ensure reliable communication and synchronization between the SN-A agent and the MEC layer. It can achieve this through a message broker functionality, e.g., RabbitMQ, or a standardized Application Programming Interface (API). In the NFV model, the MEC layer components are hosted on distributed NFV Infrastructure (NFVI) and would be controlled by one or more Virtualized Infrastructure Managers (VIMs). The Orchestration layer is hosted separately and communicates with the NFV domain through NFV Orchestrator (NFVO) to emit corrective decisions and actions. VNF Managers (VNFMs) manage life-cycles of the SFC services carried out on VNFs over multiple administrative domains. Furthermore, users in the users' layer benefit from the distributed aspect of computations in the MEC layer, which reduces latency while following end-users' mobility patterns.

We design our agent SN-A assuming that any process using file transfer and synchronization tools, e.g., rsync, exploits all the available bandwidth. Once other system processes start their network transfer operations, either migrations or application data traffic transfers, the bandwidth is shared among them using the best-effort policy [28]. Thus, we note that as we increase the number of concurrent migrations, the time to complete any individual migration will increase due to resource contention. Indeed, if the number of concurrent transfers is big enough, the migration times will become too large since none of them can be completed within a reasonable time.

Following these assumptions and knowing that the disk and memory pages' transfers are the main steps of any live migration process [7], [29], [30], we can derive that searching for an acceptable network bandwidth limit has to take into account:

- The heterogeneity of applications size;
- The content of the virtualized instances, i.e., containers and VMs;
- The types of migration selected, i.e., SFC or simple live migrations (not inter-correlated). Note that the authors have already proven that SFC migration data differs from single or simple live migration data [6], [30] .

It is therefore resulting in colossal action space. These conditions make the action selection a non-trivial, non-deterministic, and exhaustive procedure. Consequently, we consider employing DRL techniques to bypass the brute force search method or an uncontrolled migration process, both cases causing a detrimental impact on the QoS.

DRL techniques have gained significant attention as an enabler of RL for previously intractable problems. Indeed, DRL represents a step toward building autonomous systems with a higher-level understanding of the visual world [31]. Nevertheless, both DRL and RL techniques are based on trial and error processes and hence are unfeasible to directly integrate them with production environments as some tried actions may worsen already achieved performance. We address this issue by integrating a Training and Exploration (TE) module responsible for creating identical digital twin environments used for the training phase into the SN-A agent.

Initially, the TE module, through the RH module, gathers all the bandwidth capacity and latency information between each pair of MEC nodes to obtain a global knowledge of the distributed infrastructure. We use a client/server-based IPerf test integrated with the TE module in this scouting stage. This step is a reconnaissance phase that generates most of the network information we use as an upper-bound for selecting bandwidth actions [32]. Then, after each migration decision in the test environment, the TE module reserves the network resources to successfully complete the SFC migration operations while improving the global bandwidth utilization. Finally, we release the used resources whenever migrations are completed. Note that we use a practical implementation of the SFC migration schemes presented in [6], which, in addition to ensuring service migration, guarantees predetermined order of SFC components and their respective network and system dependencies. The presented process allows the SN-A agent to learn how to attribute optimal/near-optimal bandwidth values over time through the TE module. It should be also noted that we can replicate these offline trial and error achievements in other environments, e.g., 5G networks, as the training and testing phases share the same input features and output decisions.

Once obtaining preliminary results, the TE module shares its learned model with the Bandwidth Allocator and Exploitation (BAE) module to minimize network resource utilization. Therefore, we can validate the results' usability by comparing them to their handcrafted counterpart, defined in [6]. The SN-A agent compares the learned policies against the handcrafted values; if both downtime and total migration time of the SFC migration increase, the TE module will continue the learning process without reporting its current findings to the BAE module. Reversely, if the TE finished learning a fully working model, the SN-A agent will use BAE to forward the accurate decisions to the MEC layer. Finally, both TE and BAE use the "DRL Algorithms Trainer (DAT)" module, which trains DRL algorithms based on the received inputs and delivers adequate bandwidth values. Furthermore, in Sections IV-A and V, we detail and analyze the proposed comparison method of the SN-A agent.

B. Reinforcement Learning Background

We start by presenting a brief introduction to RL and DRL, given their importance to this research work. RL is one of the most important research directions of ML, which has significant impact on the development of AI/ML over the last

twenty years [33]. RL is a learning process wherein an agent can periodically interact with an environment E, where the period, by convention, is considered discrete. Particularly, the agent observes a current state s_t, then executes an action a_t, and observes a new state s_{t+1} along with receiving results in the form of a reward r_{t+1}, i.e., sometimes referred to as punishment, to automatically adjust the strategy/policy $\pi(s_t, a_t)$ for carrying out an optimal behavior [34]. The policy π is the process of mapping states to actions, i.e., $\pi : S \rightarrow A$, while maximizing the discounted reward over the discrete-time steps. The discounted reward G_t is defined by:

$$G_t = \sum_{m=0}^{\infty} \gamma^m r_{t+m+1} \tag{1}$$

where γ is the discounting factor defined between [0-1]. The discount factor helps determine the importance of future rewards.

Notwithstanding their proof of convergence, RL methods have limited application in practice [35]. Admittedly, in complex and large-scale networks, state-action spaces became usually large, and RL struggles to represent them in current memory architectures. Hence, it may fail to find an optimal policy in a reasonable time. However, the emergence of the Deep Learning (DL) paradigm has caused a breakthrough in the ML area [36]. Therefore, DRL approaches combine basic RL methods with Deep Neural Networks (DNNs) to effectively handle scalability issues [37].

RL/DRL is composed of value-based and policy-based methods, each of which has its advantages and inconveniences regarding the applicability, feasibility, and computation requirements. Among them, Q-learning, which is part of the value-based family, is one of the most prominent RL algorithms [38]. Q-learning uses a simple structure represented by a table dubbed Q-table, however, this algorithm is, in practice, limited and inefficient. Consequently, Deep Q-Network (DQN) replaces the static Q-table with a DNN. The DNN computes values of $Q(s_t, a_t)$ or Q-values, i.e., the quality of the selected action a_t in the state s_t, and realizes an acceptable mapping from states to actions. Albeit showing good scalability with regards to the number of states, DQN was often unstable and divergent. Hence, there is a need to add experience replay memory to break the correlation between subsequent time-steps and allowing a stable learning curve [39].

Policy-based methods directly learn the policy function that maps states to actions instead of computing value functions to each approximated state. Policy Gradients (PG) algorithms, such as REINFORCE and its variants [40], in addition to Asynchronous Actor-Critic (A2C) [41] and Asynchronous Advantage Actor-Critic (A3C) [42], i.e., A2C and A3C are hybrid but are often classified as policy-based algorithms. They are efficient in high dimensional action spaces as well as continuous spaces compared to value-based algorithms [43]. Besides, policy-based methods can also learn from stochastic policies by outputting probabilities for each action. Therefore, handling the exploration/exploitation trade-off as well as getting rid of the problem of perceptual aliasing state where identical states require different actions [44]. Although policy-based methods can solve problems that value-based methods cannot, they usually converge on a local maximum rather than on the global optimum [45]. Consequently, the selection of RL/DRL algorithms heavily depends on the type of the problem, the desired accuracy, and the computational time/resource trade-off.

C. System Model

Before going deep into the implementation of the SN-A agent, we first define the used state, i.e., problem's inputs, and action, i.e., problem's outputs, spaces as well as the reward function guiding the agent's decisions.

1) State Space: As the bandwidth selection problem always occurs between two MEC nodes, only the source and destination nodes will be considered when defining state space or the problem inputs. We can model this problem using, as a state or problem inputs, the size of the last iteration of a given SFC migration process together with the number of memory pages written in that iteration. The dump size is the memory size of the last iteration in an iterative live migration process. The memory pages are the number of written pages by a live migration process. Those two input parameters are crucial as the number of memory pages, the dump size, and the available bandwidth are directly correlated with the instance's downtime duration, which is the key factor determining users' QoE and satisfaction [6], [29].

$$\mathcal{S} = (d_s, p_r) \tag{2}$$

where d_s denotes the dump size and p_r denotes the number of memory pages.

2) Action Space: The action space, also known as problem outputs, is represented as allowed bandwidth allocations for migrations. The DRL agent selects a given bandwidth value at each time-step, offering by the same time the possibility to test actions as much as possible.

$$\mathcal{A} = \{bw_1, bw_2, bw_3, \ldots, bw_n\} \tag{3}$$

where $\mathcal{A}$ represents the set of all possible bandwidth values in case of discretized values. However, most of the time, $\mathcal{A}$ tends to infinity. Therefore, we must consider both the continuous and discrete action spaces in our problem.

3) Reward Function: By using a reward function that covers the required metrics, an agent maximizes profits, thereby optimally performing and selecting the right actions from within all defined states. As the live migration process uses both system and network resources, the adequate reward function must cover those resources. Moreover, our modeling assumes a direct relation between the bandwidth used and the transmission delay, as well as the propagation delay, thus covering the network resources part. Regarding the system part, the processing delay can be considered as the synchronization time. By measuring the required time for copying memory pages/file system, i.e., rootfs, in all live migration's actions, the coverage of those three-time delays is guaranteed. Besides, by inverting the obtained time, we ensure that the longer the migration time is, the lower the reward $\mathcal{R}$ will be.

$$\mathcal{R} = 1/\mathcal{T} \tag{4}$$

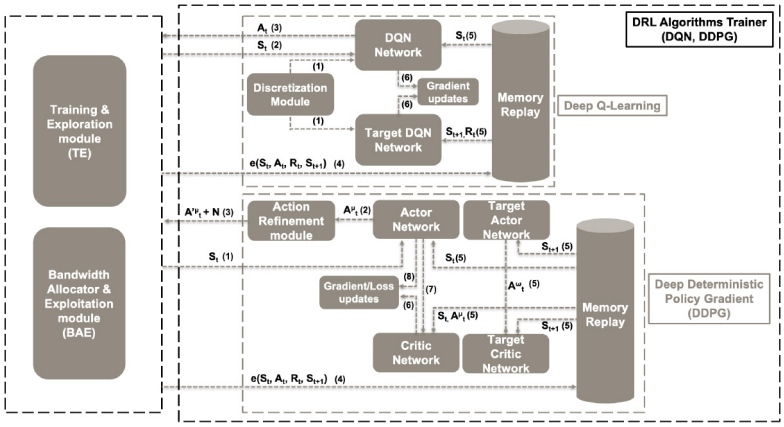


Fig. 2. Design and principles of the DRL Algorithms Trainer (DAT) module.

where $\mathcal{T} = \mathcal{T}_{transmission} + \mathcal{T}_{propagation} + \mathcal{T}_{processing} + \Delta\mathcal{T}$. $\Delta\mathcal{T}$ represents a constant related to the queuing delay as well as to the Kernel/Userspace transitions.

However, as we have several possible bandwidth values, this reward function becomes inefficient. For instance, if we consider a bandwidth equal to 3 GBps and a second one equal to 2 GBps, both provide similar times for the SFC migration metrics, which prevents the agent from determining the best action to select. Thus, to increase the accuracy of our reward function, the addition of the numerical value of the selected bandwidth is mandatory. The new reward function $\mathcal{R}$ is then expressed as:

$$\mathcal{R} = (1/\mathcal{T}) + (1/bw_i) \tag{5}$$

where bw_i denotes the current selected bandwidth value, $bw_i \in \mathcal{A}$.

To demonstrate the importance of introducing the selected bandwidth's value into the reward function, we present a detailed example in Fig. 1. Let us assume that a group of users is moving to a different location, e.g., connected cars in Fig. 1 moving from the service area of MEC 1 to the service area of MEC2. Moreover, those connected cars are consuming a video streaming service hosted initially in MEC1. Therefore, intuitively, we must ensure a minimum bandwidth allocation to support the virtualized instances' migration, i.e., SFC, composing the video application, and serving the connected cars while ensuring users' QoE. We represent the globally available bandwidth between the two MEC nodes in Fig. 1 using a cylinder to indicate the capacity.

Meanwhile, we assume that the SN-A agent is in the training phase, where the TE module measures the total available bandwidth bw_c in the reconnaissance phase presented earlier. The maximum bandwidth value measured is equal to 3 GBps, i.e., in blue color in the cylinder between MEC1 and MEC2. The TE module will be constituting the state "s_t," ($d_s = 1.2$ MB, $p_r = 596$ memory pages), based on the information gathered from the MEC source via the periodical sampling of the RH module.

The TE module then selects an action, i.e., bandwidth value, based on a given policy, either learned (e.g., PG) or followed (e.g., ϵ-Greedy). In the presented example, the first selected bandwidth value is equal to 2.5 GBps, i.e., illustrated in red inside the cylinder representing the available bandwidth, while the downtime was equal to $1 \approx 1.2$ (s). Once done, the SN-A agent sends the selected action to the environment, i.e., the MEC layer, for execution. Then, the environment returns a reward and a new state to the SN-A agent. After that, the SN-A agent evaluates its choices and similarly selects another action equal to 2 MBps, i.e., illustrated in green inside the cylinder representing the available bandwidth, in which the downtime was equal to $1 \approx 1.2$ (s). We must highlight that this is only a simplified example of two stages that differ from the training used by the SN-A agent, which keeps repeating this process until its convergence. This allows us to conclude that defining a reward function only based on time is not helpful in this situation. In other words, selecting action $bw_1 = 2.5$ GBps and action $bw_2 = 2$ MBps is identical for the SN-A agent since $R_1 \approx 0.83$ and $R_2 \approx 0.83$. In contrast, adding the inverse of the chosen action will undoubtedly make our SN-A agent select lower bandwidth values, i.e., $R_1 \approx 0.834$ and $R_2 \approx 1.33$; hence, selecting the second, i.e., 2 MBps, bandwidth value in the example and ensuring optimal bandwidth allocation. It is worth noticing that the performances are not affected as the first term of the reward function, i.e., $1/\mathcal{T}$, prevents this through time.

Finally, we added coefficients to both the time and bandwidth values to make one parameter more influential than the other depending on the network provider's considered objective.

$$\mathcal{R} = \theta * (1/\mathcal{T}) + \vartheta * (1/bw_i) \tag{6}$$

where $\theta \geq \vartheta$ as we emphasize the importance of the downtime.

IV. Design of the DRL Algorithms Trainer

A. Operational Mechanisms

We have separately introduced the DAT module in Fig. 2 to highlight the different components of the algorithms, i.e., Deep Deterministic Policy Gradient (DDPG) and DQN, constituting it and their working mechanisms. The DAT module aims to train two distinctive and divergent objectives through two particular types of DRL algorithms, namely DQN and

DDPG. The DQN algorithm focuses on accelerating the delivery of decisions and the learning process along with the loss of precision, while DDPG is more constrained by the learning time, but outputs effective decisions [46].

Our objective is to develop a hybrid approach capable of coping with the stringent demands of 5G and beyond networks. Leveraging the OSS/BSS components of the NFV architecture, the SN-A agent can obtain information about the type of service, i.e., URLLC, mMTC, and eMBB. Besides, we know via MEC APIs [47], [48] the number of MEC services and applications susceptible to request an SFC migration pattern. We note a trade-off between how quickly and how accurate decisions can be made, which depends on the underlying DRL algorithm. In case of a large number of users requiring low-latency communications, a massive number of mMTC services, or enhanced mobile broadband resources, the DAT module will deliver actions, i.e., bandwidth values, based on the DDPG algorithm to either the BAE module in case of exploitation or to the TE module during training. Contrarily, if the resource requirements and the number of end-users applications are reasonable, the DAT module will deliver results following the DQN algorithm as those services do not consume or require strict bandwidth values.

Before describing both the DQN and DDPG algorithms and their hyper-parameters, we provide the pseudo-code detailing the SN-A agent's functionalities in Algorithm 1. The proposed agent is divided into two distinctive steps:

- The training phase: It begins by neural network initialization. It then allows our agent, through the TE and DAT modules, to learn optimal policy by selecting appropriate actions, i.e., bandwidth values;
- The exploitation phase: By following the optimal policy, the agent delivers actions and optimized bandwidth values leveraging the BAE and the DAT modules.

Algorithm 1, called Kernel SN-A (KSN-A), serves to describe in detail the two steps constituting the SN-A agent. Initially, KSN-A initializes both "*base_bw*" and "*base_downtime*" variables with "*baseline_bw*" and "*baseline_downtime*" values, respectively. The variable "*baseline_bw*" is the baseline bandwidth value and the variable *baseline_downtime* is the optimal downtime achieved using the "*baseline_bw*" value, i.e., both defined in [6]. The variable "*trained*" is set to "*False*" to allow KSN-A to start the training phase. The initialization procedure is shown in lines 1 to 3 in KSN-A, i.e., Algorithm 1.

As long as the variable "*trained*" is equal to "*False*" and the variable "iteration" is smaller than "M," KSN-A will continue learning, and states will be fed into the training phase directly, i.e., lines 4 to 18 in KSN-A. It is noticed that "M" was introduced for reducing complexity and efficiency purposes. KSN-A gather states, i.e., "*S*Ï," through a blocking function, i.e., get_state(), using the RH module that interacts with the MEC layer, i.e., KSN-A line 7. Each state "*S*" is routed to the TE module through the RH module, i.e., KSN-A line 8. Meanwhile, we obtain the type of service requesting an SFC migration operation from the ETSI-integrated NFV domain, i.e., line 9. The TE module will then input the state "*S*" for either the DQN algorithm or the DDPG algorithm in the DAT module, depending on the criticality of the service requesting service migration, i.e., KSN-A, line 10 to 15. Note that the criticality of the service was deeply explained in the introduction of Section IV. Also, the initialization of both algorithms, i.e., DQN and DDPG, is omitted in KSN-A for the sake of simplicity. After that, in line 10 of KSN-A, the DAT module trains the selected algorithm, and the variable "iteration" is incremented by one for each new state, i.e., lines 16 and 17. Whenever the variable "iteration" is bigger than "M," we compare the learned bandwidth values and downtime to the "*base_bw*" and "*base_downtime*" of the baseline solution. This step ensures that the learned values are optimal compared to the current baseline values. If the learned values are less than the baseline values, KSN-A sets new baseline values and updates the variable "*trained*" to "*True*," i.e., lines 19 to 23; thus, switching to the exploitation phase starting from the next input states.

Once the variable "*trained*" is equal to "*True*," the SN-A agent, through its RH module and the ETSI-NFV integrated domain, gathers new states and routes the requests to the

Algorithm 1: Kernel-Smart Network-Aware (KSN-A)

```
1  base_bw ← baseline_bw;
2  base_downtime ← baseline_downtime;
3  trained ← False;
4  while trained == False do
5      iteration ← 0;
6      while iteration < M do
7          S ← RH.get_state();
8          RH.route(TE);
9          type_of_service = NFV.service_type();
10         if type_of_service == Critical then
11             TE.input(S, DDPG);
12         end
13         else
14             TE.input(S, DQN);
15         end
16         bw_value ← DAT.train();
17         iteration ← iteration + 1;
18     end
19     if bw_value < base_bw and RH.downtime() <
       base_downtime then
20         base_bw ← bw_value;
21         base_downtime ← RH.downtime();
22         trained ← True;
23     end
24 end
25 S ← RH.get_state();
26 RH.route(BAE);
27 type_of_service = NFV.service_type();
28 if type_of_service == Critical then
29     BAE.input(S, DDPG);
30 end
31 else
32     BAE.input(S, DQN);
33 end
34 bw_value ← DAT.deliver();
```

BAE module with the state "*S*" and the adequate algorithm depending on the type of service. Finally, the BAE module contacts the DAT module, which will deliver accurate bandwidth values in the context of SFC migration operations, i.e., lines 28 and 34 in KSN-A. Note that we precede each function/method with the module's name that executes it to improve the understanding of the proposed SN-A agent's core features.

To enhance the understanding of the role of each used algorithm, we will provide a brief introduction to both of them while specifying the necessity and the complementary usage in different situations.

1) Deep Q-Network: Mnih *et al.* [36] designed and introduced DQN, a value-based algorithm where the deep network takes a state s_t as an input while following policy π and produces a Q-value for every action in the action space. As shown in Fig. 2, DQN uses experience replay to break the correlation between subsequent time-steps, allowing a stable learning curve [39]. At each batch size, DQN computes the Temporal Difference (TD) error by taking the difference between target Q-values, i.e., the maximum possible value from next states s_{t+1}, shown in Equation (7) as "$r_{t+1} + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$İ," and the predicted Q-values, i.e., $Q^{\pi}(s_t, a_t)$ in Equation (7), [49]. This process results in a well-known regression problem in which we have to minimize the total error of the training data. Hence, allowing the function approximator, e.g., neural network, to learn a useful behavior by adjusting its parameters through forwarding/back propagation [50]. In short, the update of each Q-value, represented as in Equation (7), i.e., in case of Q-Learning, will be replaced by the update of weights in Equation (8). In both Equations (7) and (8), α is the learning rate used for setting errors' acceptance in which a higher value tolerates more error by adjusting aggressively while a smaller one adjusts conservatively. Whereas, in the same equations, γ is the discount rate that promotes or reduces the next action's impact according to the defined value.

$$Q^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) + \alpha\big(r_{t+1} + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q^{\pi}(s_t, a_t)\big) \tag{7}$$

$$\Delta\omega = \alpha\big[\big(r_{t+1} + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \omega) - Q^{\pi}(s_t, a_t, \omega)\big)\big]\nabla Q^{\pi}(s_t, a_t, \omega) \tag{8}$$

Although DQN has its advantages when solving problems with a small discrete action space, it fails to compute $max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \omega)$ of the target Q-value term in Equation (8) in case of continuous or pseudo-continuous action space. We can solve this issue by discretizing the action space, e.g., if our maximum available bandwidth is 3 GBps, we can use all values starting from 1 MBps to 3000 MBps, i.e., 3 GBps, generating 3000 actions. This will reduce the number of actions, but it will also neglect some bandwidth values, reducing bandwidth allocation precision. However, due to computational limitations when using ML frameworks such as Pytorch or Tensorflow, the discretization has to be more refined and smaller, i.e., 3000 actions will generate a huge computation for simple tasks [51]. Meanwhile, by leveraging a handcrafted bandwidth value equal to 2 MBps, the author of [6] achieved a downtime equivalent to when using the whole available bandwidth for both video streaming applications and blank containers. Consequently, we discretize our action space via the Discretization Module (DM in Fig. 2) to twenty different actions while centralizing the range around 2 MBps, which will give us twenty actions between 1 MBps and 3 MBps with a step of 0.1 MBps. The proposed algorithm is fast and ensures convergence to the near-optimal value while lacking precision in bandwidth value selection, albeit at the cost of less precise bandwidth reservations.

2) Deep Deterministic Policy Gradient: DQN ensures fast training and delivery of predictions [46]. However, in 5G and beyond networks, it is expected to have numerous services with different characteristics and types [4]. Consequently, a lack of precision in bandwidth action selection may increase bandwidth capacity usage without being fully exploited. This poor action selection may result in an implicit reduction of available network resources, thus impacting sensitive services such as URLLC and eMBB ones. To cope with the previously cited constraint and the continuous/pseudo-continuous action space limitation, we propose using the DDPG algorithm, which is considered a policy-based RL algorithm [52]. As illustrated in Fig. 2, DDPG is quite similar to A2C and A3C principles with a difference in the Actor's operations [42]. Note that A3C is an A2C variant that implements parallel training where multiple workers in parallel environments independently update a global value function, hence the addition of "asynchronous." The DDPG Actor maps the states to actions instead of outputting the probability distribution across action space like in A3C and A2C. The Actor will start by observing the state s_t of the environment E. It will then select a given action with the current weights of the approximator network, i.e., steps 1 and 2 in the DDPG part of Fig. 2. Besides, the DDPG Actor adds noise $\mathcal{N}$ while selecting a given action a_t to encourage exploration. Upon performing the action $a_t + \mathcal{N}$, the environment returns a reward r_t and the next state s_{t+1} and considering that we are using experience replay principals in all our proposals, a tuple containing (s_t, a_t, r_t, s_{t+1}) will be collected and stored in the experience pool, i.e., step 4, DDPG, Fig. 2. Once reaching the defined batch size, the DDPG Actor will start retrieving the next states and predicting the actions to be selected. Meanwhile, the Critic network uses the same selected batch of next states and the predicted actions from the Actor network to compute and evaluate the target Q-values, i.e., step 5, DDPG, Fig. 2. Finally, a loss function will be updated to learn how to evaluate more accurately in case of a Critic network and to increase the probability of choosing the right actions in case of an Actor network, i.e., steps 7 and 8, DDPG, Fig. 2. It should be noted that both Actor and Critic networks use target networks to prevent the optimization, i.e., prediction of next states' actions and their evaluations, from encountering tight correlation problems.

B. Design of Neural Networks

To realize an accurate mapping from states to actions, we build our neural networks following a pseudo-grid-searching mechanism, in which we take the hyper-parameters

values utilized in original papers. We then vary those hyper-parameters to obtain optimal configurations and parameters related to selecting bandwidth value in SFC migration schemes. This method is similar to grid searching developed in [53]. We selected the following specifications.

1) DQN Hyper-Parameters: Regarding the DQN approach, we use two neural networks, the main Q-Network and a target Q-Network as a replica of the main network. While the main Q-Network is used to predict the current state's Q-value, the target Q-Network is employed to predict next-state Q-values. We utilize an "ϵ-Greedy" based on the "ϵ" decay policy to allow a trade-off between exploration/exploitation dilemma. For both Q-Networks, we adopt Adam optimizer, i.e., an adaptive learning rate optimization algorithm for improving stochastic gradient descent, for adjusting the network's parameters [54]. The learning rates, i.e., α, and the discount factors, i.e., γ, parameters were equal to $5 \cdot 10^{-2}$ and 0.95, respectively, for both Q-Networks. Target Q-Network's parameters are updated every four episodes. The batch size used for updating the main Q-Network weights is 32. We also consider two fully-connected hidden layers in which the number of units, i.e., activation functions, is the mean between input and output features. For both hidden layers and the output layer, we select the Rectified Linear Unit (ReLU) as activation functions. The ReLU activation function, denoted by Equation (9), is linear for all positive values and zero for all negative values. Therefore, it offers computational simplicity and better convergence features compared to other activation functions. The number of output decisions is obtained via the DM, i.e., introduced earlier in Section IV-A1, showed in step 1, DQN part, Fig. 2.

$$ReLU(z) = \begin{cases} z & \text{if z} > 0 \\ 0 & \text{if z} \leq 0. \end{cases} \tag{9}$$

2) DDPG Hyper-Parameters: DDPG uses four neural networks. A Q-network and a target Q-network as Critic networks for evaluation of selected actions. In addition to a deterministic policy network and a target policy network as Actor networks for action prediction. For both Critic and Actor networks, we adopt the Adam optimizer for adjusting networks' parameters. The learning rates of Actor and Critic networks were $25 \cdot 10^{-5}$ and $25 \cdot 10^{-4}$, respectively. The discount factor γ was 0.99. The parameter of the target Actor and Critic networks are updated with a coefficient τ equal to $1 \cdot 10^{-2}$. The batch size used for updating the deterministic policy network weights is 8. We employ a similar representation of hidden layers for all Actor/Critic networks; mainly, we use two fully-connected hidden layers in which the number of units, i.e., activation functions, is 400 and 300, respectively. Each activation function uses the ReLU function for weights computation, introduced in Equation (9). For the output layers of Actor networks, we utilize a Tanh activation function, i.e., Equation (10). It is worth noticing that Critic networks have a unique output used to compute the value of taking a given action at a given state. By using Tanh as an output activation function for Actor networks, the selected bandwidth values are confined between the range of $[-1, 1]$, see Equation (10).

$$\tanh(z) = \frac{1 - exp(-2z)}{1 + exp(-2z)} \tag{10}$$

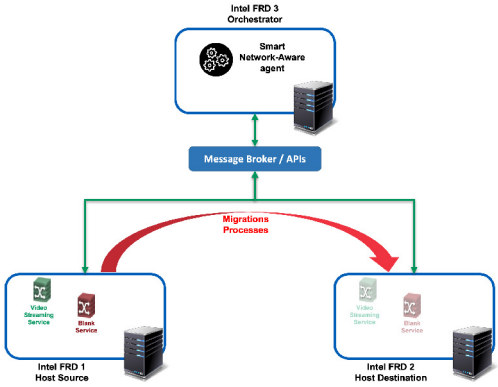


Fig. 3. Training Testbed setup.

However, having bandwidth values limited between this small range is not conceivable. Hence, leveraging the Action Refinement (AR) module, illustrated in Fig. 2, i.e., step 3 in the DDPG section, and Equation (11), we multiply the obtained values by a given number dubbed "*X*İ," Then, we add "*X*" to the obtained number; this has the main objective to centralize the output around "*X*İ," For instance, our Actor network output 0.5, the AR module will output 750 KBps if the "*X*" = 500 KBps.

$$bw_{ddpg}(X, z) = X \times (1 + \tanh(z)) \tag{11}$$

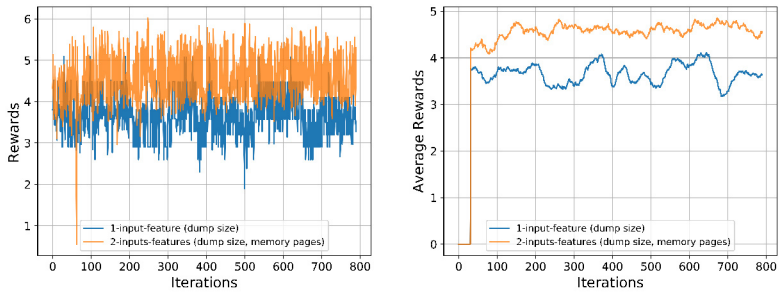
Note that "*X*" is also considered a critical hyper-parameter; thus, we vary it to obtain optimal configurations and parameters related to selecting bandwidth value in SFC migration schemes. Initially, we exploit the results in [6], where the authors achieved the best results using a handcrafted bandwidth value equal to 2 MBps. Having this initial indicator, knowing that the Tanh function varies from -1 to $+1$, and using the proposed formula developed in Equation (11), we initially set "*X*" equal to 1 MBps. This value is half of the handcrafted value and allows us to visit the range from 0 MBps to 2MBps. Then, we vary the hyper-parameter "*X*" while observing the different learning curves. This method is similar to grid searching developed in [53]. Note that we selected the value 800 for the hyper-parameter "*X*," Finally, we add the Ornstein-Uhlenbeck Noise to obtained action for encouraging exploration [55].

V. Experimental Evaluation

This section presents our preliminary training and assessments of the two DRL-based algorithms for enabling a fine-grained selection of network bandwidth values for service migration. Our focus on the networking part of migration processes arises from our perceived need to support multiple simultaneous migrations, i.e., SFC migrations, caused by user mobility across domains or resource shortages.

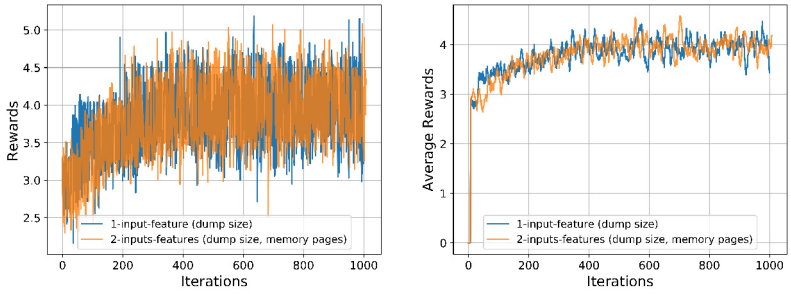
Fig. 3 describes the testbed environment used for the training phase, thus allowing our SN-A agent to refine bandwidth values selection. The testbed consists of three edge servers, Intel Fog Reference Design (FRD),[1] as depicted in Fig. 3.

[1]The Fog reference design is not a product sold by Intel and is rather a reference design offered to certain industry leaders to allow rapid development of Edge products.



(a) Rewards using DQN. (b) Average rewards using DQN.

Fig. 4. DQN-based training and comparison.



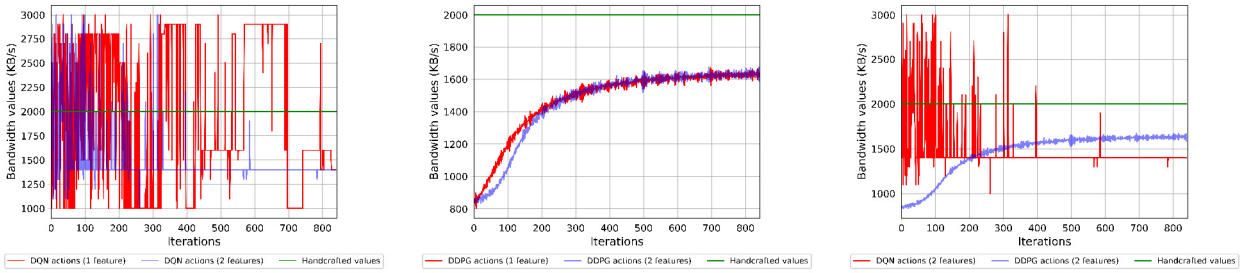
(a) Rewards using DDPG. (b) Average rewards using DDPG.

Fig. 5. DDPG-based training and comparison.

It is worth noticing that we adopted FRDs servers to match MECs' computations standards. Each FRD has 8 cores, i.e., Intel Xeon CPU E3-1275 v5 @ 3.60GHz, with VT-X support enabled, 32 GB of memory, and Ubuntu 16.04 LTS with the 4.4.0-77-generic kernel installed. FRD1 and FRD2 are acting as the host source and destination, respectively. Both of them are using LXC 2.8 as a container engine to enable container-level virtualization and CRIU 3.11 to allow service migration. We use the Synchronized Wait-For-Me SFC migration pattern developed in our previous work. This SFC migration pattern allows us to run all migration steps for each instance of the SFC in parallel except the final memory blocking action, i.e., dump, where all instances should wait for each other [6]. This approach in a basic handcrafted controlled network aims to efficiently control each step separately, thus allowing a fine-grained control and reducing the overall system and network resource consumption. Besides, we use ONOS as an SDN controller to configure OVS switches and steer network traffic between the SFC constituents. We consider an SFC for a video streaming application with three virtualized instances, i.e., length three. Each SFC contains a client, a video streaming server, and a turnaround node that analyzes and route the integrality of the traffic in both directions. FRD3 serves as a global orchestrator used for handling the life-cycle of containers and agent's training and exploitation phases. Note that FRD3 consists of the SN-A agent, the life-cycle orchestrator, i.e., managing network function creation/deletion/migration/scaling operations, and the message broker server, i.e., RabbitMQ in our case.

We start evaluating our proposed agent by showing the training phases for both DQN and DDPG while considering different input features to highlight features selection, training speed, and stability, i.e., Fig. 4 and Fig. 5. Afterward, we present a detailed comparison, in Fig. 6, regarding action selection, i.e., bandwidth values, for DDPG against DQN and handcrafted values, i.e., baseline solution. Finally, we show the efficiency and the capacity of the proposed agent, i.e., SN-A, to reduce the network consumption compared to the basic solution via a downtime comparison, i.e., Fig. 7. Note that we consider an SN-A variant that only uses DQN and another SN-A variant that only uses DDPG to properly and separately evaluate each algorithm in our current experiments. Note also that we overcome the issues of migration failures in the training of both DDPG and DQN based algorithms, i.e., Fig. 4 and Fig. 5, through the use of the retry module developed in our previous work [7]. This module was developed due to the need for high efficiency of the live migration. We designed a mechanism able to detect the failure of the last step and trigger an automatic retry to ensure a highly efficient migration that meets the 5G networks' requirements. Thus, the correlation between migration requests is respected. In the case of unknown issues or errors, the whole scenario is deleted or not used for the training to keep the results concrete.

The initial experiment is related to the SN-A agent based on the DQN algorithm. We run 800 migration operations while randomly modifying the virtualization instances' resources, i.e., CPU, RAM. Fig. 4 shows the training comparison while considering the agent based on SN-A DQN for respectively two, i.e., dump size and memory pages, and a single, i.e., dump size, input features. In Fig. 4(a), the Y-axis represents the rewards collected over time-steps while the X-axis shows the number of iterations in the training process. Still, in Fig. 4(a),



(a) Comparing using different input features in the DQN-based agent.

(b) Comparing using different input features in the DDPG-based agent.

(c) Comparing DQN to DDPG when input features = 2.

Fig. 6. Comparing bandwidth action selection for DQN against DDPG and handcrafted values.

rewards for two input features are represented with the orange color while the single input is illustrated using the blue color. Fig. 4(b) conserves an identical representation, except for the Y-axis, in which we show the average rewards for every 32 iterations. As an initial reflection, we can state that DQN using both the dump size and the memory pages features outperforms DQN using one input feature, i.e., dump size.

Next, we evaluate the SN-A agent based on a DDPG algorithm approach in our second experimental scenario. We trained the model for thousands of migrations while randomly selecting application types and modifying the resources, i.e., CPU, RAM, of the virtualization instances. Fig. 5 features the comparison between the dump size, i.e., blue, as a unique input feature and both the dump size and the memory pages, i.e., orange, features when considering the DDPG-based algorithm. In Fig. 5(a), the Y-axis represents the obtained rewards over all iterations, while the X-axis shows the number of iterations or migrations we did during training in the proposed architecture. Fig. 5(b) keeps the same depiction for the X-axis, while the cumulative reward for every 8 iterations is used for Y-axis. Unlike the results in Fig. 4, findings in Fig. 5 show that our DDPG-based algorithm is immune to the number of input features.

In Fig. 6, we compare the bandwidth action selection for SN-A based on the DQN algorithm, SN-A with the DDPG algorithm, and a baseline solution. It is noticed that the baseline solution that uses handcrafted bandwidth values is an existing approach developed by [6] and serves as an upper layer for the comparison. In Fig. 6, for all sub-figures, the left Y-axis represents the bandwidth values in "KBps" while the X-axis portrays the number of iterations done in the training phase. It is noticed that for Figures 6(a), 6(b), and 6(c), the baseline solution is represented by the green color. The single input feature-based solution and two inputs features are shown with red and blue colors, respectively, in Figures 6(a) and 6(b). While in Figure 6(c), the colors red and blue portray actions based on DQN and DDPG algorithms, both for two features, respectively. Fig. 6(a) highlights the comparison between bandwidth action selection for SN-A agent based on DQN when considering one and two features, respectively. Based on the results, the agent using the DQN algorithm with one input feature, i.e., dump/memory size, failed to outperform the baseline solution, i.e., handcrafted, while the SN-A agent using the DQN algorithm with two features surpasses clearly

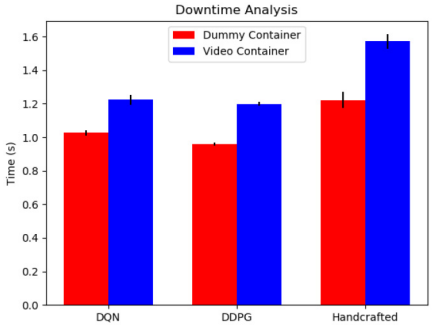


Fig. 7. Downtime comparison in case of different algorithms.

both the baseline solution and the one feature DQN algorithm. Thus, we conclude the necessity of considering memory pages as a second input feature. We also notice the high variation in action selection for both approaches, i.e., DQN with one input feature or two features. Unlike DQN, the DDPG-based agent, shown in Fig. 6(b), outputs stable and consistent training for both considerations, i.e., one/two input features. Besides, both DDPGs approaches outperform by far the baseline solution showed in green; the selected bandwidth actions were near to 1,600 KBps.

To compare DDPG and DQN agents, we plotted Fig. 6(c) while considering two input features. The preliminary results demonstrate that both DQN and DDPG achieved better results compared to the baseline solution. From Fig. 6(c), we confirm that DDPG is stable compared to DQN and explores a broader range of actions during the training phase. However, Fig. 6(c) indicates that in convergence, DQN is selecting lower bandwidth values than the DDPG-based agent, i.e., 1,400 KBps. Based only on action selection, we cannot determine the best approach in terms of resource efficiency. Thus, we extend our evaluation to cover downtime comparison.

Fig. 7 depicts the induced downtime under distinct network configurations while using DQN-based agent, DDPG-based agent, and Handcrafted bandwidth, i.e., baseline solution, respectively. The main purpose of this experiment is to compare the proposed DRL algorithms, i.e., DQN and DDPG, with each other as well as against the limited handcrafted bandwidth used in [6] in terms of induced downtime. Still, in Fig. 7, the DQN-based agent, the DDPG-based agent, and the baseline solution downtimes can be viewed in the X-axis, while the Y-axis presents the time in seconds. This experiment

TABLE I
DOWNTIME COMPARISON IN CASE OF DIFFERENT DRL APPROACHES

Algorithms	Strategies	Mean Time (s)	Std dev	CI 95%	Coef Var
DQN-based agent	Blank Container	1.026	0.208	0.0181	0.203
	Video Container	1.224	0.324	0.0286	0.265
DDPG-based agent	Blank Container	0.957	0.098	0.008	0.102
	Video Container	1.198	0.138	0.012	0.115
Handcrafted	Blank Container	1.222	0.066	0.05	0.054
	Video Container	1.571	0.056	0.042	0.036

outputs the mean downtime, standard deviation, 95% confidence interval (CI), and coefficient of variation (CV) results when using the learned policy in case of DQN and DDPG algorithms and the static handcrafted value, i.e., 2 MBps, as part of defining the most suitable DRL algorithm. Table I presents detailed downtime values. As expected, the video-streaming container results are larger when compared to the blank, i.e., turnaround or dummy, container results for all variants. The difference in these results is due to the additional copies of the network connections status. From Table I, we can observe that the agent based on the DDPG algorithm outperforms the remaining proposed approaches in terms of downtime. Indeed, the DDPG-based agent is the only DRL algorithm which reached less than one-second downtime when migrating blank, i.e., turnaround, containers.

A. Discussion

Based on the bandwidth action selection shown in Fig. 6(c) and the downtime comparison displayed in Fig. 7, we conclude that the minimal downtime is unbacked by selecting lower bandwidth values actions; this was confirmed by the downtime results in Table I. Moreover, increasing the downtime may affect end-users' QoE and break the Service Level Agreement (SLA) defining the requested QoS. Therefore, DDPG is preferred over DQN for its training stability, lower downtime achievements and user satisfaction.

VI. CONCLUSION AND FUTURE WORK

In this work, we designed, modeled, and evaluated two DRL-based algorithms for allowing a fine-grained selection and allocation of bandwidth resources. Our results show that DDPG outperforms DQN in terms of accuracy, stability, and users' QoE. Nonetheless, the proposed SN-A agent with its two DRL-based algorithms is limited to only detecting the required bandwidth for a given workflow. As a future research work, we will extend the proposed ML-based solution, presented in this paper, to cover the scheduling of workflows after detecting their bandwidth requirements. Besides, we intend to improve the downtime results by taking into account application-level indicators and decision-making optimizations. Finally, we plan to extend our work to cover extreme variations in environments by developing an efficient mechanism for coefficient selection.

REFERENCES

[1] *System Architecture for the 5G System; Stage 2*, 3GPP Standard TS 23.501, Mar. 2018.

[2] *5G White Paper*, NGMN Alliance, Geneva, Switzerland, Feb. 2015.

[3] T. Taleb, B. Mada, M.-I. Corici, A. Nakao, and H. Flinck, "PERMIT: Network slicing for personalized 5G mobile telecommunications," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 88–93, May 2017.

[4] D. M. Gutierrez-Estevez *et al.*, "Artificial intelligence for elastic management and orchestration of 5G networks," *IEEE Wireless Commun.*, vol. 26, no. 5, pp. 134–141, Oct. 2019.

[5] R. A. Addad, T. Taleb, H. Flinck, M. Bagaa, and D. Dutra, "Network slice mobility in next generation mobile systems: Challenges and potential solutions," *IEEE Netw.*, vol. 34, no. 1, pp. 84–93, Jan./Feb. 2020.

[6] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, and H. Flinck, "Towards studying service function chain migration patterns in 5G networks and beyond," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Waikoloa, HI, USA, Dec. 2019, pp. 1–6.

[7] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, and H. Flinck, "MIRA!: An SDN-based framework for cross-domain fast migration of ultra-low latency 5G services," in *Proc. IEEE Global Commun. Conf. GLOBECOM*, Abu Dhabi, UAE, Dec. 2018, pp. 1–6.

[8] *MEC in 5G Networks*, ETSI, Sophia Antipolis, France, Jun.'2018.

[9] O. Obulesu, M. Mahendra, and M. ThrilokReddy, "Machine learning techniques and tools: A survey," in *Proc. Int. Conf. Invent. Res. Comput. Appl. (ICIRCA)*, Coimbatore, India, Jul. 2018, pp. 1–9.

[10] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Netw.*, vol. 32, no. 2, pp. 92–99, Mar. 2018.

[11] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, "Wireless network intelligence at the edge," *Proc. IEEE*, vol. 107, no. 11, pp. 2204–2239, Nov. 2019.

[12] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, "Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 38–67, 1st Quart., 2020.

[13] M. Latva-Aho and K. Leppänen, "Key drivers and research challenges for 6G ubiquitous wireless intelligence," 6G Res. Vis., Univ. Oulu, Oulu, Finaland, White Paper, 2019.

[14] "White paper 5G evolution and 6G," NTT DOCOMO, INC, Tokyo, Japan, Rep., Jan. 2020.

[15] *Experiential Networked Intelligence (ENI); ENI Definition of Categories for AI Application to Networks*, Eur. Telecommun. Stand. Inst., Sophia Antipolis, France, Nov. 2019.

[16] *Experiential Networked Intelligence (ENI); ENI Use Cases*, Eur. Telecommun. Stand. Inst., Sophia Antipolis, France, Sep. 2019.

[17] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, May 1996.

[18] J. Jeong, J. Ha, and M. Kim, "ReSeT: Reducing the service disruption time of follow me edges over wide area networks," in *Proc. 22nd Conf. Innov. Clouds Internet Netw. Workshops (ICIN)*, Paris, France, Feb. 2019, pp. 159–166.

[19] F. De Vita, D. Bruneo, A. Puliafito, G. Nardini, A. Virdis, and G. Stea, "A deep reinforcement learning approach for data migration in multi-access edge computing," in *Proc. ITU Kaleidoscope Mach. Learn. 5G Future (ITU K)*, Santa Fe, Argentina, Nov. 2018, pp. 1–8.

[20] A. Varga and R. Hornig, "An overview of the OMNET++ simulation environment," in *Proc. 1st Int. Conf. Simulat. Tools Techn. Commun. Netw. Syst. Workshops*, Marseille, France, Mar. 2008, p. 60.

[21] F. Chollet *et al.* (2015). *Keras*. [Online]. Available: https://github.com/fchollet/keras

[22] F. P. Tso, G. Hamilton, K. Oikonomou, and D. P. Pezaros, "Implementing scalable, network-aware virtual machine migration for cloud data centers," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, Santa Clara, CA, USA, Jun. 2013, pp. 557–564.

[23] A. Stage and T. Setzer, "Network-aware migration control and scheduling of differentiated virtual machine workloads," in *Proc. ICSE Workshop Softw. Eng. Challenges Cloud Comput.*, Vancouver, BC, Canada, May 2009, pp. 9–14.

[24] H. Chen, H. Kang, G. Jiang, and Y. Zhang, "Coordinating virtual machine migrations in enterprise data centers and clouds," 2012. [Online]. Available: https://www.semanticscholar.org/paper/Coordinating-Virtual-Machine-Migrations-in-Data-and-Chen-Kang/4d328f0ae97749e4a28c5cb3ec575619ff6e4efa

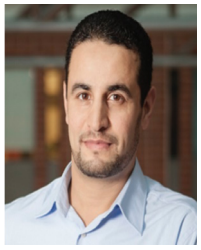
[25] M. Duggan, J. Duggan, E. Howley, and E. Barrett, "An autonomous network aware VM migration strategy in cloud data centres," in *Proc. Int. Conf. Cloud Auton. Comput. (ICCAC)*, Augsburg, Germany, Sep. 2016, pp. 24–32.

[26] R. A. Addad, D. L. C. Dutra, T. Taleb, and H. Flinck, "Toward using reinforcement learning for trigger selection in network slice mobility," *IEEE J. Selected Areas Commun.,* vol. 39, no. 7, pp. 2241–2253, Jul. 2021, doi: 10.1109/JSAC.2021.3078501.

[27] *Developing Software for Multi-Access Edge Computing*, Eur. Telecommun. Stand. Inst., Sophia Antipolis, France, Feb. 2019.
[28] Linux & Unix Tutorials. (2019). *How to Set RSYNC Speed Limit From Eating All Bandwidth With Bwlimit Option*. [Online]. Available: https://www.cyberciti.biz/faq/how-to-set-keep-rsync-from-using-all-your-bandwidth-on-linux-unix/
[29] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, and H. Flinck, "Towards a fast service migration in 5G," in *Proc. IEEE Conf. Stand. Commun. Netw. (CSCN)*, Paris, France, Oct. 2018, pp. 1–6.
[30] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, and H. Flinck, "Fast service migration in 5G trends and scenarios," *IEEE Netw.*, vol. 34, no. 2, pp. 92–98, Mar./Apr. 2020.
[31] N. C. Luong *et al.*, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, 4th Quart., 2019.
[32] A. Tirumala, F. J. Qin, J. M. Dugan, J. A. Ferguson, and K. Gibbs. (2005). *iPERF: TCP/UDP Bandwidth Measurement Tool*. [Online]. Available: https://iperf.fr/
[33] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.
[34] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.
[35] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double $Q-$learning," in *Proc. 13th AAAI Conf. Artif. Intell. (AAAI)*, Feb. 2016, pp. 2094–2100.
[36] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: arxiv.abs/1312.5602
[37] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.
[38] C. J. Watkins and P. Dayan, "*Q*-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, May 1992.
[39] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015. [Online]. Available: arxiv.abs/1511.05952
[40] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992.
[41] V. R. Konda and J. N. Tsitsiklis, "On actor–critic algorithms," *SIAM J. Control Optim.*, vol. 42, no. 4, pp. 1143–1166, Apr. 2003.
[42] Z. Wang *et al.*, "Sample efficient actor–critic with experience replay," Aug. 2016. [Online]. Available: arxiv.abs/1611.01224
[43] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn. (ICML)*, New York, NY, USA, Jun. 2016, pp. 1928–1937.
[44] L. Chrisman, "Reinforcement learning with perceptual aliasing: The perceptual distinctions approach," in *Proc. 10th Nat. Conf. Artif. Intell. (AAAI)*, San Jose, CA, USA, Jul. 1992, pp. 183–188.
[45] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Denver, CO, USA, Nov. 1999, pp. 1–9.
[46] S. Chen, "Comparing deep reinforcement learning methods for engineering applications," M.S. thesis, Fac. Comput. Sci., Otto-von-Guericke-Universität Magdeburg, Magdeburg, Germany, 2018.
[47] "Multi-access edge computing (MEC); application mobility service API," Eur. Telecommun. Stand. Inst., Sophia Antipolis, France, Rep. ETSI GS MEC 021, Jan. 2020.
[48] "Mobile edge computing (MEC); bandwidth management API," Eur. Telecommun. Stand. Inst., Sophia Antipolis, France, Rep. GS MEC-IEG 006, Oct. 2017.
[49] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, Aug. 1988.
[50] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Jul. 1989.
[51] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Vancouver, BC, Canada, Dec. 2019, p. 32.
[52] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015. [Online]. Available: arxiv.abs/1509.02971
[53] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, Feb. 2012.
[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: arxiv:abs/1412.6980
[55] L. Valdivieso, W. Schoutens, and F. Tuerlinckx, "Maximum likelihood estimation in processes of ornstein-uhlenbeck type," *Stat. Inference Stochast. Process.*, vol. 12, no. 1, pp. 1–19, Feb. 2009.

Rami Akrem Addad (Graduate Student Member, IEEE) received the Licentiate and master's degrees (High Distinction and Hons.) from USTHB, Algeria, in 2015 and 2017, respectively. He is currently pursuing the Doctoral degree with the Department of Communications and Networking, School of Electrical Engineering, Aalto University, Finland. His research interests include 5G network architecture, cloud-native technologies and approaches, network softwarization and slicing mechanisms, MEC, NFV, SDN, and distributed systems.

Diego Leonel Cadette Dutra received the B.Sc. degree in computer science, and the M.Sc. and D.Sc. degrees in systems engineering and computer science program from the Federal University of Rio de Janeiro (UFRJ), Brazil. He is currently working as a Professor with UFRJ, where he is also a Member of the COMPASS Lab. He has worked as a Postdoctoral Researcher with the COMPASS/UFRJ and MOSA!C Lab/Aalto. His research interests include computer architecture, HPC, virtualization, cloud computing, wireless networking, and SDN.



Tarik Taleb received the B.E. degree (with Distinction) in information engineering and the M.Sc. and Ph.D. degrees in information sciences from Tohoku University, Sendai, Japan, in 2001, 2003, and 2005, respectively. He is a Professor with Aalto University, Espoo, Finland and the University of Oulu, Oulu, Finland. He is also a Visiting Professor with Sejong University, Seoul, South Korea. He is the Founder and the Director of the MOSA!C Lab (www.mosaic-lab.org).

Hannu Flinck received the M.Sc. and Lic.Tech. degrees in computer science and communication systems from Aalto University in 1986 and 1993, respectively. He was with Nokia Research Center and the Technology and Innovation Unit of Nokia Networks in various positions. He is a Research Manager with Nokia Bell Labs, Espoo, Finland. He has been actively participating in a number of EU research projects. His current research interests include MEC, SDN, and content delivery in mobile networks, particularly in 5G networks.