A Reinforcement Learning Approach for Virtual Network Function Chaining and Sharing in Softwarized Networks

Godfrey Kibalya, Joan serrat, Juan-Luis Gorricho, and Peiying Zhang,

***Abstract*—Cognizant of the ease with which softwarized functions can be dynamically scaled according to real time resource requirements, and the fact that multiple services can have common VNFs in their chaining, this paper tackles the problem of cost effective deployment of online services from the perspective of sharing their VNF instances. First, we formally formulate the deployment problem under VNFs sharing. Secondly, given the NP-hard nature of the above problem, we propose a reinforcement learning (RL) algorithm capable of making intelligent placement decisions while considering multiple conflicting costs. Costs of transmission, VNF instantiation or energy consumption, among others. Thanks to the intelligence of the RL algorithm, simulation results show that the performance of the proposed algorithm is within a 14% margin and similar to an optimal solution in terms of request provisioning cost and acceptance ratio, respectively. Moreover, the algorithm results in more than a 20% and a 70% improvement in terms of request deployment cost and time compared to a state-of-the-art algorithm, and up to more than a 40% improvement in terms of cost compared to an algorithm that greedily minimizes the transmission or VNF activation costs.**

***Index Terms*—VNF Sharing, Reinforcement Learning, Virtual Network Function Chaining placement, Resource allocation.**

I. Introduction

FUTURE networks, including 5G, are expected to leverage the flexibility introduced by the Network Function Virtualization (NFV) paradigm in order to cope with the stringent requirements of future services. The main target behind NFV comes from decoupling complex network functions (e.g. Firewalls, Proxies or Load Balancers) from dedicated hardware appliances, implementing those services in the form of chained Virtual Network Functions (VNFs) [1], [2]. In this regard, future services will be instantiated as customized software to suit the particular requirements of different applications [3], [4]. If well implemented, the above approach has high prospects of reducing its network deployment footprint, as well as its associated operational cost. This is based on the ease with which softwarized functions can be activated, scaled, migrated or shutdown, allowing a more efficient use of the network resources [5].

With a myriad of network service requests envisaged to share the scarce substrate resources, innovative approaches for the deployment of services, in a cost-effective and resource-efficient manner, without degrading the Quality-of-Service, are of utmost necessity. Although not well explored, sharing VNF instances among multiple service requests provides a good alternative for reducing service deployment costs [6]. In the majority of existing works addressing Service Function Chaining (SFC) deployments, such as [7]–[9], each instance of a VNF only processes the input traffic coming from a single service request. However, such an approach, although easy to implement, can result in a low resource utilization, specially when the input traffic may experience severe fluctuations [1], [6]. In addition, dedicated VNF assignments may result in an excessive resource fragmentation, hence, leading to lower acceptance ratios of service requests, and higher overall service deployment costs. On the other hand, optimizing the service deployment cost using VNF sharing is a complex task due to the need to intelligently trade-off the involved cost components. Cost components like the transmission cost along the traversed links, the processing cost at the servers, the energy cost from both active and idle servers, or the deployment cost of instantiating new VNFs. This trade-off appears due to the fact that those costs may conflict in such a way that lowering one cost component will lead to raising another one. For instance, minimizing the transmission cost by naively deploying the service on the shortest path between the ingress and egress nodes may require the activation of new servers and VNF instances, resulting in an increase in VNF instantiation and energy costs, aside from promoting link bottlenecks. Conversely, minimizing the energy and VNF instantiation costs by reusing already deployed VNF instances and servers may result in an increased transmission cost and a poor node-load balancing, hence, affecting the long term performance of the acceptance ratio and the fault tolerance. Fig. 1 shows an example of service deployments with VNF sharing, considering online service request arrivals. At time $t1$, the placement agent receives request SFC 1 and places the whole SFC instance on Data Center (DC) D. Then, at time $t2$, the agent receives another request SFC 2 with ingress and egress nodes $s2$ and $s5$ respectively, and requesting the same sequence of VNFs as in SFC 1. Considering VNF sharing, the SFC 2 would have to be placed on DC D to reuse the VNF instances from SFC 1, as shown by the blue dotted line in the figure. This would result in the allocation of three inter-DC links for SFC 2, but with the advantage of not having to activate any new VNF instance. On the other hand, considering the shortest path routing, SFC 2 would have to be placed on DC B, activating the corresponding new VNF instances, and using only two inter-DC edges for this request as shown by the red dotted line. This example illustrates the need for an approach that intelligently trades off the different

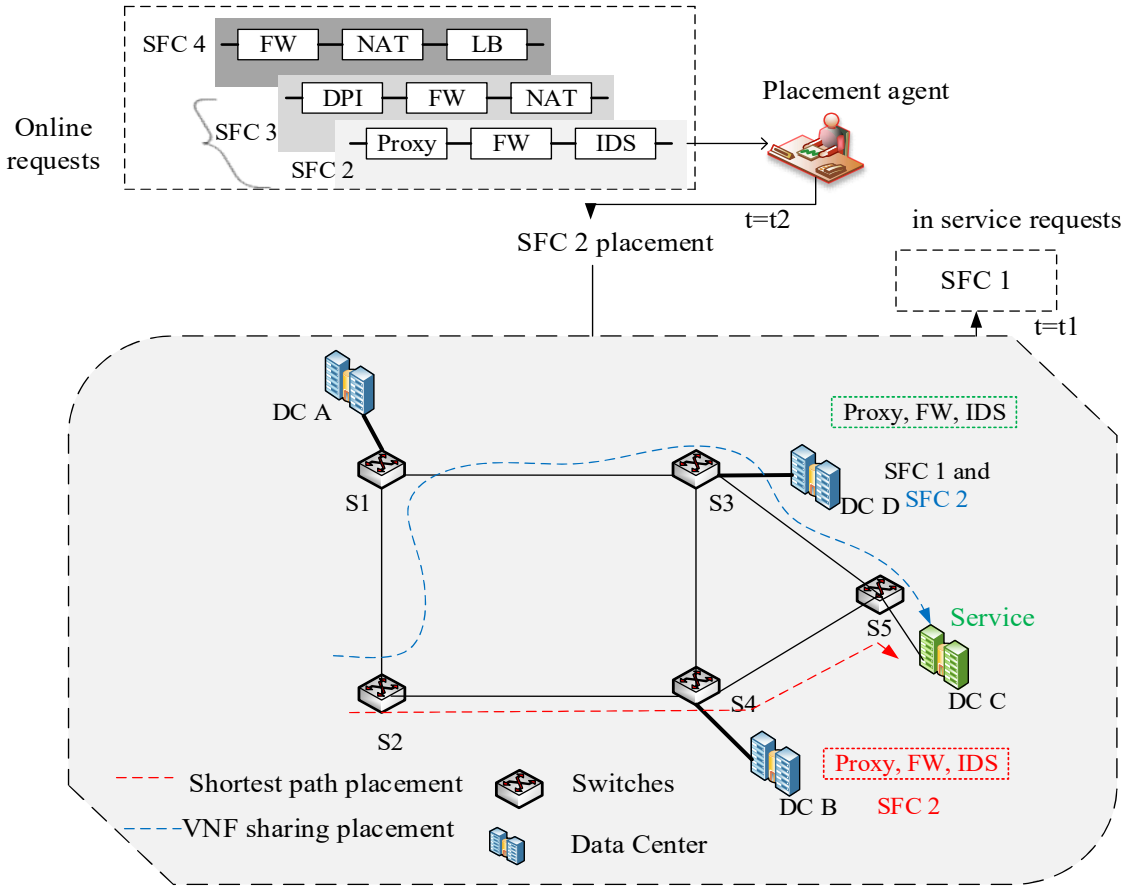


Fig. 1. An illustration of SFC deployment with VNF instance sharing

cost components if we want to optimize the overall placement objective. As demonstrated by the results of our simulations, conventional approaches that greedily target to minimize only one cost component are less effective in general, since any request deployment is always conditioned by multiple costs. With this motivation, this paper argues for a reinforcement learning approach looking for a cost effective and resource efficient service deployment, due to its ability to intelligently infer the effect of each placement decision on the long-term performance of the network. To the best of the authors' knowledge, this is the first work adopting reinforcement learning to the problem of online SFC placement with VNFs sharing. Our simulation results reveal that the adopted RL approach obtains near optimal solutions and requires even less execution time than other greedy approaches. Our contributions in this paper can be summarized as follows:

1) A formal definition and formulation of the SFC placement problem considering VNF instance sharing among different service requests.
2) The proposal of a RL algorithm using the policy gradient technique and targeting a cost-effective and resource-efficient deployment of SFCs in a feasible execution time and considering multiple cost components. Simulation results reveal that the proposed algorithm is able to find near-optimal solutions while executing in just a few millisecond per request.
3) The consideration of four different greedy heuristics as benchmark algorithms for the comparison with our RL proposal.
4) Extensive simulations considering both online and offline scenarios in order to assess the performance of the proposed algorithm against the benchmark algorithms. From the simulation results, the proposed algorithm is found to be scalable when considering an increasing size of the tested substrate networks and number of requests.

The rest of the paper is organized as follows: Section II presents the related work. The network model and problem description is presented in section III. The proposed RL based service deployment algorithm is described in section IV. The performance evaluation of the proposed algorithm, including a description of the simulation scenarios, and benchmark algorithms, is presented in section V. The paper is concluded in section VI.

II. Related work

There are a number of works in the literature addressing the SFC problem from the perspective of VNF placement, VNF scheduling, traffic steering or a joint consideration of these sub-problems. However, most of the existing works do not consider VNF sharing, which is pertinent to reduce service deployment costs and to achieve an efficient utilization of

resources. The works in [10]–[13] adopt exact approaches for the VNF orchestration problem by formulating and solving it as an Integer Linear Programming problem. However, such approaches are not well suited for practical delay sensitive applications due to their high run-time. As a result, most of the recent proposals in literature are based on heuristic or meta-heuristic approaches [9], [10], [14]–[19]. For-example, in [10], a genetic algorithm is proposed to solve the problem of Joint VNF scheduling and traffic steering with the goal of minimizing the overall VNF schedule, while [9], [17] and [18] address the problem of fault-tolerant orchestration of stateful VNFs. The work in [19] adopts a graph neural network architecture for SFC orchestration with the objective of minimising the end-to-end delay of the service request. In [14], the focus is on the problem of scheduling micro-services across multiple clouds, including micro-clouds, with the objective of reducing the overall turnaround time of complete end-to-end service of SFCs. However, these works do not incorporate the VNF sharing, which is a key component of our manuscript.

Works using VNF sharing are presented in [1], [2], [6], [20]–[26]. Considering the case where the VNFs have already been placed, the works in [1] and [2] focus on the VNF scheduling problem in such a way that, services deployed on the same node, not only share the computing resources of that node, but also their common VNF instances, if possible. In [2] the scheduling problem is formulated as an ILP problem with the goal of maximizing the accepted requests, and then, a genetic algorithm is proposed to solve that ILP problem. In [1] a VNF scheduling model based on min-plus algebra theory is proposed. However, all these works do not address directly the service deployment problem, in spite of the fact that the cost or the performance resulting from the scheduling stage is greatly tied to the previous deployment stage.

In [6] the focus is set on investigating the gain that results from sharing VNFs among different service requests, concluding that a lower resource consumption and a higher acceptance ratio is achieved in comparison with the case where each request is provisioned with dedicated VNF instances. Moreover, the sharing of VNFs is shown to promote the server consolidation, resulting in an energy saving at both software and hardware level. The work in [22] considers the deployment of premium and best-effort services with the support of VNFs sharing and migration. The service deployment problem is formulated as an integer quadratically constrained programming (IQCP) problem. But, aside from not incorporating energy or VNF deployment costs, the exact approach adopted to solve the IQCP model is not well suited for delay sensitive applications due to its high run time complexity. The work in [23] addresses the VNF sharing problem focusing on the assignment of priorities to the services sharing VNFs. In [24] a greedy algorithm based on a goodness factor for the deployment of end-to-end slices is proposed with the goal of minimising the amount of bandwidth and computation resources to be used. In all these works, however, the energy and VNF deployment costs have not been considered, simplifying the placement decision. Considering a scenario in which VNFs can be shared across multiple requests, the works in [25] and [26] focus on the problem of the VNF migration and the network service (NS) scheduling, respectively. The VNF migration problem in [25] is solved using a heuristic approach based on the Technique for Order Preference by Similarity to an Ideal Solution (TOPSIS), while the NS scheduling problem in [26] is formulated and solved as an ILP problem with the objective of maximising the number of admissible network service requests. The work in [27] formulates the joint problem of network function mapping, scheduling and routing, while considering deadline constrained services. This is done with a Tabu-search based approach being proposed for solving these sequential problems. In [20] the problem is generalized as a facility location problem and the authors propose a Mixed Integer Linear Programming (MILP) model for this problem. Given the time complexity for solving the MILP problem, the authors propose a heuristic approach, which is based on the Steiner tree problem and Markov decision processes, to choose the nodes for the placement of the SFC requests, based on the number of shareable VNFs and the forwarding costs across all requests. However, such an approach is not well suited for the online scenario in which the requests are not known ahead of time, since it is not possible to compute the shareable VNFs of all such requests in advance. Moreover, the remapping of requests upon the arrival of new requests would result in service disruptions. The work in [21] addresses the VNF placement problem with VNF instance sharing. The authors propose Flex-share, a heuristic approach that exploits the Hungarian algorithm for the convex optimization of the VNF placement using priority assignments to the shared VNFs. This work, however, focuses on scheduling and priority assignments at shared VNFs without considering service implementation costs such as energy, traffic forwarding or fragmentation costs. Similar to our work, the work in [28] considers a detailed cost modeling incorporating the VNF deployment cost, the energy cost and the traffic forwarding cost to the problem of SFC deployment with VNF sharing. First, the problem is formulated as an Integer Liner Programming (ILP) problem. Then, a heuristic approach based on a multi-stage graph, including the Viterbi algorithm, is proposed to overcome the time complexity of the ILP problem. For the proposed heuristic, a mapping decision is made considering one request at a time, hence, it is well suited for both offline and online scenarios. Moreover, the algorithm is shown to provide near optimal solutions. For this reason, we use this work as one of the benchmark algorithms for our proposed algorithm while adopting a similar cost function and components. However, the number of layers of the graph, and the number of nodes per layer, grow with the amount of required VNFs per request and the number of candidate nodes for each VNF. This significantly affects the scalability of the approach, especially when considering large networks. Different from this and other existing works, we adopt a reinforcement learning approach oriented to solve the cost-effective and resource-efficient problem while incorporating multiple concurrent costs such as energy consumption, data processing, VNF activation and data transmission. Our approach is shown to be scalable when increasing both the substrate network size and the number of required VNFs by the arriving requests.

Machine learning techniques have already been used for

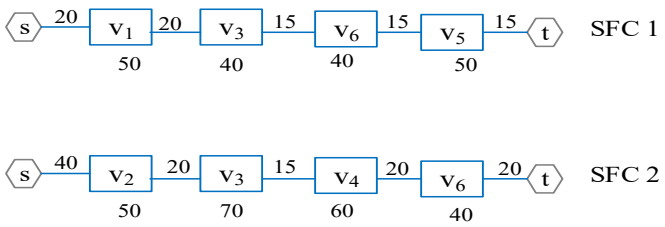


Fig. 2. An illustration of two SFC requests each with 4 traffic nodes where traffic nodes V_3 and V_6 are common between the two requests.

solving the problem of the SFC service deployment in [9], [29]–[37]. For instance, in [32], RL is adopted for VNF-SC deployment in Elastic optical networks with the objective of load balancing and minimising service delay. In [33], the focus is on minimising end-to-end delay in NFV-enabled networks. The work in [34] incorporates RL and block-chain for cost effective and secure SFC deployment, while the work in [35] focuses on traffic forwarding in SFC chains. The work in [36] adopts DQN for adaptive resource allocation with the objective of minimising cost. The work in [37] seeks to jointly maximise load balance and acceptance ratio while minimising resource consumption. In [9] the focus is set on solving the fault-tolerant placement problem of stateful VNFs with the goal of reducing the state update overhead. In [29] the authors target to exploit the inter-relation between different SFCs by aggregating multiple requests and admitting them jointly as a bunch. In [30] the authors target the problem of large network state spaces, considering a real-time online SFC orchestration under dynamic network conditions. In [31] the work focuses on an accelerated approach for learning proper VNF sizing and placement considering various network conditions. However, as opposed to the above works, the SFC placement decision in our work is influenced by multiple conflicting cost components. Moreover, different from the above works, the neural network architecture of the policy in charge of making decisions in our work includes a convolutional layer, allowing a faster training stage and providing a higher convergence, since it uses a smaller number of trainable parameters compared to conventional feed forward neural networks. Moreover, we incorporate innovative approaches which enable the same policy neural network to be used for substrate network sizes inferior to that used at the training stage and also for different cost components without the need for its retraining. In Table II we present a summary of key distinguishing features of our work in comparison with previous RL approaches applied to the problem of SFC orchestration.

III. Service and Network Modeling with Problem Description

This section describes the request and substrate network modeling, introducing the mathematical formulation of the SFC placement problem with VNF instance sharing.

A. SFC Request

Considering a set of R requests, each request $r \in R$ is modeled as a tuple $\Psi^r = < G_v^r, C_{dem}^r, \rho^r, Del_{sd}^r, \tau_s^r, \tau_d^r, \tau_f^r >$ where G_v^r is the SFC graph of the VNFs the user traffic must traverse, including the virtual links interconnecting those VNFs. We refer to each of such required VNFs as a request virtual node, denoted by $n_v^p \in N_v$, where N_v denotes the set of all such nodes and $p \in P$ denotes the function type of this node. In order to improve the readability of the document, we replace n_v^p by n_v in the rest of the document. The parameter ρ^r denotes the requested packet rate of the user input traffic from the ingress node τ_s^r to the egress node τ_d^r. C_{dem}^r is a set indicating the CPU resource requirements at the different request virtual nodes, with $C_{dem}^{n_v,r}$ denoting the CPU requirement of virtual node n_v. We assume that the CPU resources required by a node n_v are proportional to the packet rate to be processed by this node, [17], [18], i.e. $C_{dem}^{n_v,r} = \rho_{n_v}^r \times C_\rho$, where $C_{dem}^{n_v,r}$ is the amount of CPU resources required by n_v, with $\rho_{n_v}^r$ and C_ρ denoting the packet rate traversing n_v and the amount of CPU resources required to process each unit of packet rate by this node, respectively. The terms Del_{sd}^r, τ_s^r, τ_d^r, τ_f^r are used to denote the request end-to-end latency requirement, the ingress node, the egress node and the request life-time, respectively. Similarly, we denote by $l_{uv} \in L_v$ the request virtual link between virtual nodes u and v, and we denote the bandwidth requirement of such a link by Bw_{uv}^r. Figure 2 shows an example of two SFC requests. The CPU requirement of each virtual node is shown below the box, while the bandwidth requirement of each virtual link is shown on top of the link. Note that the bandwidth requirement may vary across different links since the packet rates may be altered by the traversed traffic nodes, for instance, as a result of filtering or splitting of packets due to applying some kind of networking functionality.

B. Substrate network

We model the substrate network operated by a cloud provider as an undirected graph $G_s = (N_s, E_s)$ where, N_s denotes the set of physical nodes capable of hosting VNFs, and E_s is the set of physical links interconnecting those nodes. Each node $n_s \in N_s$ is characterized by a CPU resource capacity $C_{max}^{n_s}$, its residual CPU resources $C_{res}^{n_s}$, its power consumption κ^{n_s} and its provisioning capacity for a given VNF type $\eta_p^{n_s} \in \{0, 1\}$, equal to 1 if a VNF of type p can be provisioned by n_s, zero otherwise. In a similar way, each interconnecting link $e \in E$ is characterized by its maximum bandwidth capacity B_{max}^e, a propagation delay del^e, and a cost for transmitting a packet rate unit ζ^e.

C. Virtual Network Functions

We envisage a NFV environment in which the different network functions, such as IDs and Firewalls, among others, have been virtualised and provisioned by the underlying physical infrastructure, from which they are assigned the required CPU resources upon activation. Moreover, we assume multiple instances of a given VNF type and denote by M the set of all VNFs in the system. We consider each VNF $m \in M$ of type p to be characterised by assigned resources in terms of CPU, denoted by Cpu_{vnf}^m, a deployment cost which captures the cost of the image transfer and booting of that VNF, denoted

TABLE I
RL RELATED WORK FEATURES

Considered Aspects	[9]	[29]	[30]	[31]	[32	[33]	[34]	[35]	[36]	[37]	[38]	Our work
VNF sharing consideration	x	x	x	x	x	x	x	x	x	x	x	✓
Trained Neural network applicable to different state size	x	x	x	x	x	x	x	x	x	x	x	✓
Energy cost	x	x	x	x	x	x	x	x	x	x	x	✓
VNF activation cost	x	x	x	x	x	x	x	x	x	x	x	✓
Processing cost	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Forwarding cost	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Resource fragmentation cost	x	x	x	x	x	x	x	x	x	x	✓	✓

" ✓" indicates that a given aspect is considered by the corresponding work.
"x" Indicates that a given aspect is not considered by the corresponding work

TABLE II
PARAMETERS

Notation	Description
G_s	Substrate network graph
r	SFC request
Ψ^r	SFC request tuple
G_v^r	SFC instance graph of request r
n_v^p	Request virtual node of type p
τ_s^r	Ingress node of request r
τ_d^r	Egress node of request r
τ_f^r	Request life time
ρ^r	Average packet rate of request r
$\rho_{n_v}^r$	Packet rate processed by request node n_v^p
Del_{sd}^r	End-to-end request delay
l_{uv}	Virtual link between virtual nodes u and v
Bw_{uv}^r	Bandwidth requirement for l_{uv}
$C_{dem}^{n_v,r}$	CPU demand by request node n_v^p
δ_{vnf}^p	Deployment cost for a VNF of type p
ϑ_{vnf}^m	Processing capacity of VNF m
del_{vnf}^m	Processing delay through VNF $m \in M$
ζ_{vnf}^{p,n_s}	Cost for processing a packet rate unit by a type p VNF when provisioned on substrate node $n_s \in N_s$
$C_{max}^{n_s}$	CPU capacity of substrate node $n_s \in N_s$
$C_{res}^{n_s}$	Residual CPU resources at substrate node n_s
κ^{n_s}	Power consumption at node n_s
B_{max}^e	Bandwidth capacity of substrate link $e \in E_s$
del^e	Propagation delay on link $e \in E_s$
ζ^e	Transmission cost per packet rate unit along edge $e \in E_s$
$\gamma_c^{n_v}$	Charge by NSP for processing a packet rate unit at traffic node n_v^p
$\gamma_{bw}^{l_{uv}}$	Charge by NSP for transmitting a packet rate unit through l_{uv}
β_w	Cost for each watt of power consumed
$e_{max}^{n_s}$	Maximum power consumption of n_s
α^{n_s}	penalty cost for each unit of unused node CPU
α^{vnf}	penalty cost for each unit of unused VNF CPU

TABLE III
SETS AND VARIABLES

Notation	Description
E_s	Set of all substrate edges
N_s	Set of all substrate nodes
N_V	Set of all virtual nodes of a request
R	Set of all available requests in the system
E_v	Set of all request virtual links
P	Set containing types of deployed VNFs
M	Set of all deployed VNFs
Υ_{vnf}^p	Set containing all substrate nodes on which a type p VNF can be provisioned.
C_{dem}^r	Set of CPU requirements of different virtual nodes of $r \in R$
$z_{mp} \in \{0,1\}$	z_{mp}=1 if VNF m is of type p
$\sigma_e^r \in \{0,1\}$	σ_e^r=1 if $e \in E$
$\chi_m^\tau \in \{0,1\}$	χ_m^τ=1 if the current state of VNF m is different from that at previous provisioning event
$q_{n_v^p,m}^r \in \{0,1\}$	$q_{n_v^p,m}^r$=1 if traffic node is provisioned on VNF m,zero otherwise
$f_{n_v^p}^p \in \{0,1\}$	$f_{n_v^p}^p = 1$ if n_v^p is of type p, zero otherwise.
$y_m^{n_s} \in \{0,1\}$	$y_m^{n_s} = 1$ if VNF m is provisioned on n_s
$\sigma_{uv,e}^r \in \{0,1\}$	$\sigma_{uv,e}^r = 1$ if traffic link l_{uv} is provisioned on edge $e \in E_s$
$\eta_p^{n_s} \in \{0,1\}$	$\eta_p^{n_s} = 1$ if node n_s can provision a VNF of type p
$\gamma_{n_s}^{n_v^p,r} \in \{0,1\}$	$\gamma_{n_s}^{n_v^p,r}$=1 if request node n_v^p is provisioned on n_s
$\chi^m \in \{0,1\}$	$\chi^m = 1$ if VNF m is active, zero otherwise

by δ_{vnf}^p, the processing capacity which denotes the maximum packet rate the VNF can process, denoted by ϑ_{vnf}^m, the cost for processing each unit of packet rate at this VNF if the VNF is provisioned on node $n_s \in N_s$, denoted by ζ_{vnf}^{m,n_s}, the average processing delay experienced by a packet when processed by the VNF, denoted by del_{vnf}^m, and a set of nodes on which such VNF can be provisioned, denoted by Υ_{vnf}^p.

D. Problem description and Formulation

The SFC provisioning problem involves obtaining a mapping from the SFC graph G_v^r to a subset of the substrate network graph G_s that optimizes a given provisioning objective, with the requirement that all the constraints of the request are satisfied. The goal for us will be to minimize the operational cost incurred by a NSP thanks to minimizing the implementation cost of the requests. We consider the implementation cost of a request to be influenced by the following cost components: *i*) the energy consumption cost associated with running a VNF on a given node; *ii*) the communication/forwarding/transmission cost of transferring the user traffic from the ingress node to the egress node along the intermediate links; *iii*) the processing cost incurred for processing the user traffic at the different VNFs traversed by the traffic; *iv*) the cost of deploying new VNF instances; and fi-

nally, v) the cost due to the fragmentation of substrate network resources. This work argues for a selection of substrate nodes and links provisioning the request in a manner that jointly considers all the above cost components, since in practice, the benefit obtained by greedily optimising a single component may be offset by an increased cost of another component(s). Therefore, we formulate the placement objective to minimise the average implementation cost of each request as follows:

$$Minimise: \quad \frac{1}{|R|}\sum_{r\in R} C(G_v)^r \tag{1}$$

where $C(G_v)^r$ denotes the implementation cost of request $r \in R$, and it is evaluated as:

$$C(G_v)^r = C^r_{proc} + C^r_{fwd} + C^r_{enrg} + C^r_{depl} + C^r_{frag} \tag{2}$$

where C^r_{proc}, C^r_{fwd}, C^r_{enrg}, C^r_{depl} and C^r_{frag} denote the processing cost, the traffic forwarding cost, the energy cost, the VNF deployment cost and the resource fragmentation cost, respectively. Those costs are detailed below.

1) Processing cost: The processing cost C^r_{proc} incurred by a request $r \in R$ is evaluated as:

$$C^r_{proc} = \sum_{n_v\in N_v} \gamma^{n_v,r}_{n_s} \times \zeta^{m,n_s}_{vnf} \times \rho^r_{n_v} \tag{3}$$

where the binary variable $\gamma^{n_v,r}_{n_s} \in \{0,1\}$ is equal to 1 if virtual node $n_v \in N_v$ of request $r \in R$ is provisioned on substrate node n_s, zero otherwise, with $\rho^r_{n_v}$ and ζ^{m,n_s}_{vnf} denoting the traffic rate to be processed by n_v and the processing cost for each traffic rate unit at node n_s respectively. Note that in this work, we allow the processing cost per unit of traffic rate to be different across different physical nodes, even for the same VNF type. The reason for this design choice is twofold; first, this is permissible in practice since different physical nodes could belong to different providers who may have different billing policies. Moreover, different nodes may be characterised by different QoS guarantees in terms of service reliability and holding priority, justifying the different billing rates. But most importantly, such a design choice introduces an extra degree of freedom for selecting the physical nodes on which to provision the request virtual nodes. This fact increases the complexity of the placement problem, and somehow justifies the need to use RL algorithms in front of greedy approaches in order to solve the problem in a satisfactory way.

2) VNF deployment Cost: The cost C^r_{depl} is incurred whenever new VNFs are activated in order to provision the request $r \in R$. This cost is evaluated as:

$$C^r_{depl} = \sum_{m\in M|\chi^m=1} z_{mp} \times \delta^p_{vnf} \times \chi^\tau_m \tag{4}$$

where $\chi^m \in \{0,1\}$ is a binary variable, equal to 1 if VNF $m \in M$ is active, zero otherwise; and $\chi^\tau_m \in \{0,1\}$ is a binary variable, equal to 1 if the state (i.e. active or inactive) of VNF m in the current provisioning event is different from the previous provisioning event. $z_{mp} \in \{0,1\} = 1$ if VNF m is of type $p \in P$ and δ^p_{vnf} is the deployment cost for a VNF of type p.

3) Energy cost: In order to evaluate the energy cost C^r_{enrg}, we consider the energy consumption of a node to consist of two components [28]: the active state component, which is proportional to the amount of node resources being used; and the idle state component, which is the energy consumption in the idle state. In this regard, the energy consumption of a node n_s is computed as:

$$E_{n_s} = \sum_{m\in M} y^{n_s}_m \times \chi^m \times (e_{max} - e_{id}) \times \frac{Cpu^m_{vnf}}{C^{n_s}_{max}} + e_{id} \tag{5}$$

where $y^{n_s}_m \in \{0,1\} = 1$ if VNF m is provisioned on physical node $n_s \in N_s$, zero otherwise. $C^{n_s}_{max}$ and Cpu^m_{vnf} denote the CPU capacity of node n_s and the CPU allocated to VNF m from node n_s respectively. The parameters e_{max} and e_{id} denote the energy consumption in the peak consumption state and the idle state, respectively. If we denote by $E^{'}_{n_s}$ the energy consumption at n_s prior to provisioning the request $r \in R$, and denote by $E^{''}_{n_s}$ the energy consumption at n_s after provisioning request r, then, the energy consumption at n_s due to the current request is evaluated as: $E^r_{n_s}$=$E^{''}_{n_s}$-$E^{'}_{n_s}$. Therefore, the energy cost C^r_{enrg} across all nodes is evaluated as:

$$C^r_{enrg} = \beta_w \sum_{n_s\in N_s} E^r_{n_s} \tag{6}$$

where β_w is the cost per unit of energy consumption.

4) Forwarding cost: This cost is incurred along the substrate edges traversed by the traffic from the ingress to the egress nodes of the request. Therefore, this cost component increases as the number of used substrate edges increases. This is evaluated as:

$$\sum_{l_{uv}\in L_v} \sum_{e\in E} \sigma^r_{uv,e} \times \rho^r_u \times \zeta^e \tag{7}$$

where the binary variable $\sigma^r_{uv,e} \in \{0,1\}$ is equal to 1 if substrate edge $e \in E$ is part of the substrate path provisioning the virtual link uv, zero otherwise. ρ^r_u and ζ^e denote the packet rate traversing link $e \in E$ and the cost per packet rate unit on $e \in E$.

5) Resource fragmentation cost: An envisaged key factor of the use of NFV comes from the flexibility we will have for sharing different network resources among multiple traffic flows and services. In this work, we evaluate the resource sharing capacity of the different algorithms in terms of resource fragmentation. Resource fragmentation can occur at the node level as a result of activating new nodes when there are others underutilised, or can occur at VNF level by deploying new VNF instances when there are other active instances underutilised. The fragmentation cost at node n_s can be evaluated as:

$$C_{frag} = F_{n_s} + F_{vnf} \tag{8}$$

where F_{n_s} is the fragmentation cost of the physical servers and F_{vnf} is the fragmentation cost of the VNFs (including their deployment platforms e.g. virtual machines and containers). The fragmentation cost across all nodes is evaluated as:

$$F_{n_s} = \sum_{n_s\in N_s|\lambda^{n_s}=1} \alpha^{n_s}(C^{n_s}_{max} - \sum_{m\in M} \chi^m Cpu^m_{vnf} y^{n_s}_m) \tag{9}$$

where α^{n_s} is the fragmentation penalty for each unit of unused CPU resource on that node. The binary variable $\lambda^{n_s} \in \{0,1\} =$

1 if the node n_s is active, zero otherwise. A node is considered active if there is at least one active VNF provisioned by that node, thus :

$$\sum_{m \in M} y_m^{n_s} \chi^m \geq 1 \tag{10}$$

The VNF fragmentation cost relates to the amount of CPU resources allocated to the active VNFs that is not used by the request virtual nodes. This is evaluated as below:

$$F_{vnf} = \alpha^{vnf} \sum_{m \in M} (\chi^m Cpu_{vnf}^m - \sum_{r \in R} \sum_{n_v \in N_v} q_{n_v,m}^r C_{dem} {n_v}^r) \tag{11}$$

where $q_{n_v,m}^r \in \{0,1\} = 1$ if virtual node $n_v \in N_v$ is provisioned by VNF $m \in M$. The parameters α^{vnf}, Cpu_{vnf}^m and $C_{dem}{n_v}^r$ denote the fragmentation penalty for each unit of unused VNF CPU resource, the amount of CPU allocated to VNF $m \in M$ and the amount of CPU resources consumed from VNF m by virtual node n_v of request $r \in R$.
Complementary, the optimization expressed in Eqn. 1 should adhere to a number of constraints, including the following:

• The CPU consumption by the VNFs provisioned on a physical node $n_s \in N_s$ should not exceed the resource capacity of that node.

$$\sum_{m \in M | \chi^m = 1} Cpu_{vnf}^m y_m^{n_s} \leq C_{max}^{n_s} \quad \forall n_s \in N_s \tag{12}$$

where $y_m^{n_s} \in \{0,1\} = 1$ if VNF m is provisioned on node $n_s \in N_s$

• The amount of traffic going through a given VNF $m \in M$ should not exceed the VNF processing capacity:

$$\sum_{r \in R} \sum_{n_v \in N_v} q_{n_v,m}^r \rho_{n_v}^r \leq \vartheta_{vnf}^m \quad \forall m \in M \tag{13}$$

where $q_{n_v,m}^r \in \{0,1\} = 1$ if the virtual node $n_v \in N_v$ of request r is provisioned by VNF $m \in M$.

• Each VNF of type p should be provisioned on a substrate node capable of supporting that VNF type:

$$y_m^{n_s} \times z_{mp} = 1 \; iff \; n_s \in \Upsilon_{vnf}^p \quad \forall m \in M, p \in P \tag{14}$$

where Υ_{vnf}^p is a set containing all nodes that can provision a VNF of type p.

• Similarly, each virtual node must be mapped onto a VNF of the same type:

$$f_{n_v}^p \times q_{n_v,m}^r = z_{mp} \quad \forall n_v \in N_v, r \in R, m \in M, p \in P \tag{15}$$

where $f_{n_v}^p \in \{0,1\} = 1$ if virtual node $n_v \in N_v$ is of type p, zero otherwise.

• Every virtual node $n_v \in N_v$ must be provisioned by exactly one VNF:

$$\sum_{m \in M} q_{n_v,m}^r = 1 \; \forall n_v \in N_v, r \in R, m \in M \tag{16}$$

• The total bandwidth consumption on a given edge $e \in E$ should not exceed the capacity of that edge:

$$\sum_{r \in R} \sum_{uv \in L_v} \sigma_{uv,e}^r \times Bw_{dem}^r \leq B_{max}^e \forall e \in E \tag{17}$$

where $\sigma_{uv,e}^r \in \{0,1\} = 1$ if substrate edge $e \in E$ is used to provision virtual link uv, zero otherwise.

• The end-to-end mapping delay should not exceed the acceptable delay of the request:

$$\sum_{uv \in L_v} \sum_{e \in E} \sigma_{uv,e}^r del^e + \sum_{n_v \in N_v} f_{n_v}^p \times del_{vnf}^p \leq Del_{sd}^r \quad \forall r \in R \tag{18}$$

where the first and second terms of equation 18 correspond to the propagation and processing delay, respectively.

The above problem can be solved by means of conventional solvers, such as Gurobi and CPLEX. However, given the NP-hard nature of the problem, such approaches are not well suited for delay sensitive applications envisaged in future networks due to their high run time even for medium sized networks. This motivates the use of heuristic approaches such as those based on node ranking, load balancing, and shortest paths computation, among others. Although these approaches can execute in polynomial time, they are not well suited for scenarios such as the one considered in this work where the placement objective is jointly influenced by multiple attributes. This motivates the use of machine learning approaches that are able to intelligently infer the long term influence of each cost component to the placement objective, yielding near-optimal solutions in acceptable run times.

IV. Proposed Reinforcement learning based VNF placement algorithm

This section describes the proposed SFC placement algorithm including: its MDP modeling, the neural network architecture of the policy in charge of making the VNF assignments, and the training procedure of that neural network. Everything will be discussed below.

A. MDP model

Considering a working scenario in which the state of a given system is fully observable by an RL agent, we model the system as a Markovian Decision Process (MDP) defined by the tuple (S, A, P, R), where: S denotes the set of possible states of the system; A denotes the set of possible discrete actions to be taken, actions for the selection of a physical node to host a given VNF of any request; $P = P(s_{t+1}|s_t, a_t)$ denotes the transition probabilities from state s_t to state s_{t+1} after taking action a_t; and $R = R(s_t, s_{t+1}, a_t)$ denotes the reward obtained after taking action a_t from state s_t and transiting to state s_{t+1}. Considering an entire episode as a sequence of visited states due to the corresponding sequence of actions, the return of an episode is defined as the discounted sum of all the rewards received by the agent during that episode. It is expressed as:

$$Return = \sum_{i=t}^{T} \gamma^{i-t} R(s_i, s_{i+1}, a_i) \tag{19}$$

where $R(s_i, s_{i+1}, a_i)$ is the reward received by the agent after taking action a_i in state s_i at step i. In this problem domain, the goal of the RL agent is to learn a policy $\pi : S \rightarrow A$ which maximizes the expected return, $\mathbb{E}[Return]$, over all episodes. In this work, we relate the reward signal with the placement objective expressed in equation 1. Consequently, by maximizing the reward signal, the RL agent will be able to minimise the operational expenditures of a NSP.

The parameters of the MDP tuple for our working scenario are discussed below:

1) State space: The state, at any time, is defined by relevant features of the environment, features that will be used by the agent to infer the actions that will produce high rewards. Those features will come from the substrate network state and the present incoming request, as those features will impact, directly or indirectly, the upcoming rewards. In order to conform to the neural network architecture of the policy selecting the actions, which incorporates a convolutional layer as shown in Fig. 3, the system state is modelled as an image-like $|N_s| \times K$ feature matrix, where N_s and K denote the set of substrate nodes and the number of relevant features associated with each substrate node. In order to avoid the need to retrain the proposed neural network for different test scenarios with less nodes than those used for the original training, we will use dummy nodes to assure fixed dimensions of the state matrix, since in practice, once learned, the internal structure of a neural network cannot be modified. In order to achieve this, the policy neural-network is trained using the maximum possible number of nodes. Then, for the testing phase, when the number of nodes is less than the ones used for training, we match the input matrix by appending dummy nodes with dummy feature vectors to reach the expected state size. For instance, if the policy network is trained with N nodes and the test scenario has M nodes, where $M < N$, the number of dummy nodes created in this case is N-M, with each node being assigned a vector of dummy features. The dummy features are obtained by providing the worst value of each feature, in order to make such dummy nodes less likely to be selected by the policy network. Moreover, in the worst event that a dummy node is assigned a high probability of being selected to host a given VNF, the filtering layer that we use at the output end of the architecture will be able to sieve out such a node.

For each request $r \in R$ to be mapped, the algorithm uses the neural network to make a placement decision for each VNF of the request, one at a time, starting with the VNF closest to the ingress node. Therefore, the number of decision epochs (hence the actions taken) for each request is equal to the number of VNFs of the request. For each virtual node n_v from request $r \in \mathbb{R}$ to be scheduled for placement, the following are the features associated with each node n_s^K of the substrate network to form the state matrix:

- the residual computing resources, evaluated as:

$$CPU_{n_s}^{i} = max(0, \frac{C_{res}^{n_s} - C_{dem}^{n_v,r}}{C_{max}^{n_s}}) \tag{20}$$

  where $C_{res}^{n_s}$ denotes the residual computing resources at n_s and $C_{dem}^{n_v,r}$ denotes the required CPU resources of the current virtual node. This feature guides the RL agent in selecting nodes with sufficient CPU resources and selection of more loaded nodes when beneficial in order to reduce node resource fragmentation.
- the cost for processing each unit of packet rate through a VNF provisioned on n_s, denoted by ζ_{vnf}^{m,n_s}. This feature relates to the processing cost component of the service deployment cost.
- Deployment cost of a type p VNF denoted as δ_{vnf}^{p}. This feature guides the agent regarding the trade-off between activation of a new VNF instance and reuse of active ones in relation to other cost components.
- The cost for transmitting a packet rate unit ζ^e. This feature directly relates to the forwarding cost component.
- the number of edges of the shortest path between $n_s^{n_v-1}$ and n_s, denoted by $E_{n_s}^{n_v}$. Where $n_s^{n_v-1}$ denotes the substrate node used to map the virtual node preceding n_v. Note that $n_s^{n_v-1} = \tau_s^r$ if it is the first virtual node to be mapped. This feature relates to the suitability of node n_s in terms of traffic forwarding cost contribution to the overall cost.
- the node delay computed as the delay of the shortest path between $n_s^{n_v-1}$ and n_s, denoted by $Delay^{n_s}$. For fairness and to conform to the approach adopted in the benchmark algorithm proposed in [28], we distribute the end-to-end delay requirement of the request among the different virtual links which we denote as inter-VNF delay. In this regard, $Delay^{n_s}$ is evaluated as:

$$Delay^{n_s} = max(0, \frac{Del^r - Del_{sd}^{n_s}}{\delta_{max}}) \tag{21}$$

  where $Del_{sd}^{n_s}$ is the delay along the path from $n_s^{n_v-1}$ to n_s, while Del_{uv}^{r} is the inter-VNF delay of the request. δ_{max} is a normalisation term evaluated as the maximum delay between $n_s^{n_v-1}$ and all the alternative nodes for hosting the current request virtual node.
- the available (bottleneck) bandwidth on the shortest path from the source node to the terminal node going through n_s, denoted by $Bandwidth^{n_s}$ and computed as:

$$Bandwidth^{n_s} = max(0, \frac{Bw_{av} - Bwt_{dem}^{r}}{Bw_{max}}) \tag{22}$$

  where Bw_{av} denotes the bottleneck bandwidth along the shortest path between τ_s^r and τ_d^r. Bwt_{dem}^{r} and Bw_{max} denote the request desired bandwidth and the maximum edge bandwidth respectively. This term guides the agent in selecting nodes that result in feasible paths to the egress node and also to trade-off between link load balance and mapping cost.
- A binary variable $k_p^{n_s} \in \{0, 1\} = 1$ if there is an active VNF of type p at n_s, zero otherwise. This guides the agent about the VNF activation status at a given node.

2) Action space: Our algorithm uses the neural network to select the nodes on which to place each VNF of the service request. In order to achieve this, each substrate node is assigned a unique identifier in the range $[1, |N_s|]$, where N_s denotes the set of all substrate nodes and $|N_s|$ is the cardinality of this set. Therefore, during each decision epoch, the action of the policy implemented by the neural network is $a \in A$, where the action space $A = \{0, 1, 2, 3....|N_s|\}$. The zero value of a corresponds to the case where no substrate node is selected for hosting a given virtual node, consequently, the request will be rejected. This will occur, for example, when all the nodes have been deemed infeasible by the filtering layer of the policy neural network.

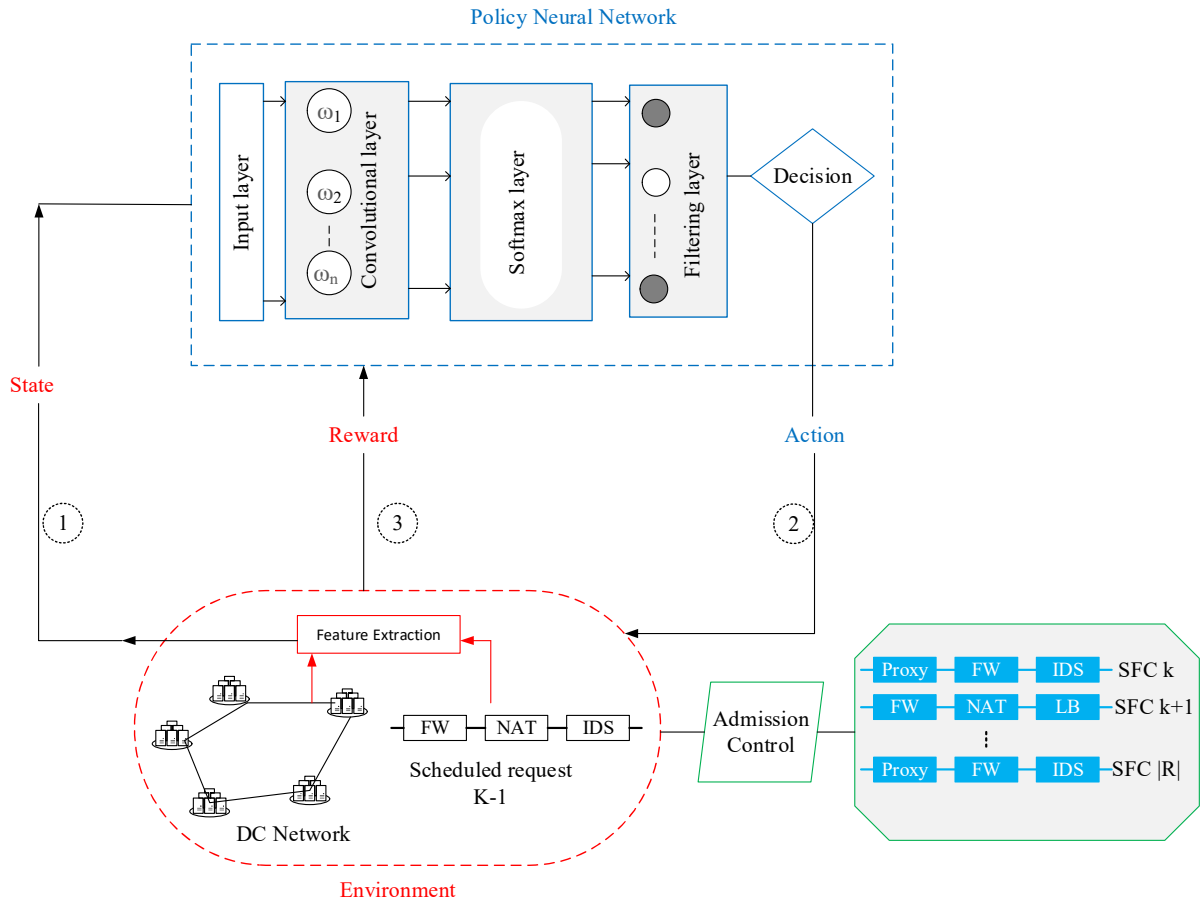


Fig. 3. Architecture of the Policy Neural Network Architecture

3) Reward: We formulate the reward signal in such a manner that by maximising the reward, the agent optimises the deployment objective given in Eqn. 1. In this regard, the reward signal is formulated as:

$$reward = \frac{Rev(G_v)^r}{C(G_v)^r} \tag{23}$$

where $Rev(G_v)^r$ denotes the request revenue as given in Eqn. 24, and $C(G_v)^r$ denotes the implementation cost of request $r \in R$ as given by Eqn. 2. Intuitively, Eqn. 23 is maximised by minimising the implementation cost of each request as desired by the deployment objective, but accepting those requests that are associated with high revenues, hence, increasing the net profit obtained by the service provider. This is different from the objective of only minimizing implementation costs. This way, the above reward formulation avoids biasing the RL agent towards requests with low resource requirements, in contrast with requests with high resource requirements, since the former are more likely to be associated with low provisioning costs, but they will also contribute less to the overall net profit. If we denote by $\gamma_c^{n_v}$ and $\gamma_{bw}^{e_v}$ the price charged by the service provider for processing and transmitting a unit of packet rate through virtual node $n_v \in N_v$ and traffic link l_{uv}, where l_{uv} is the inter-VNF path between u and v; then, the revenue $Rev(G_v)^r$ obtained from provisioning a request $r \in R$ can be defined as:

$$Rev(G_v)^r = \begin{cases} \sum_{n_v \in N^v} \gamma_c^{n_v} \rho^r + \sum_{\forall e_v \in E_v} \gamma_{bw}^{e_v} \rho^r \\ \quad \text{if } z_\tau^r = 1 \\ 0 \quad \text{otherwise} \end{cases} \tag{24}$$

where $z_\tau^r \in \{0, 1\}$ is a binary variable equal to 1 if request $r \in R$ is assigned resources, zero otherwise.

B. Architecture of the Neural Network for Policy Evaluation

Fig. 3 shows the architecture adopted for our policy neural network, which consists of four layers: the input layer, a convolutional layer, a softmax layer and a filtering layer. The convolutional layer performs a convolution between the input feature matrix and the internal weights of the layer to output a numerical vector of size $|N_s|$. The softmax layer then transforms the convolutional layer output into a vector of probabilities, where each element of the vector indicates the probability of the corresponding substrate node to be selected for hosting the VNF at hand. The filtering layer is added, at the end, to avoid infeasible nodes from being selected, for instance, the dummy nodes. Once such infeasible nodes are pruned, then, the substrate node with the highest probability is selected for hosting the corresponding VNF of the request. The processing of a request can be summarized as follows:

Upon arrival of a request to the admission control block asking for service, each virtual node of the request is processed at a time with the policy neural network deciding the substrate node on which such a node is provisioned. This way, for each arriving request all its virtual nodes are processed sequentially, one after another, starting with the virtual node next to the ingress node. This ordering ensures that the VNF order in the service chain is preserved, and also enables an early detection of infeasible virtual link mappings on executing the algorithm. The feature extraction block takes as inputs the current state of the substrate network and the requirements of the request. With all that information we compose the system state in the form of a $|N_s| \times K$ feature matrix. Then, the feature matrix is used as input to the neural network, which produces an action in the form of a node identifier for hosting the virtual node under consideration. After mapping all virtual nodes, and based on the obtained mapping solution, the resultant reward is calculated using Eqn. 23.

C. Neural network training

The training of the neural network has been done using the maximum number of foreseeable substrate nodes. As mentioned in the previous section, the placement of a request is performed virtual node by virtual node, until all the nodes of the request have been provisioned, or until a given node cannot be provisioned and the request is rejected. For each virtual node, the policy neural network is giving the substrate node with the highest probability for serving that virtual node. However, since the neural network parameters are initially assigned randomly, during the training phase we perform a trade off between exploration and exploitation to determine the substrate node on which to provision the virtual node [38].

If a virtual node cannot be provisioned, the entire request is rejected, and a new request is scheduled for placement. Otherwise, the resultant reward is obtained according to the mapping decisions made for the different virtual nodes. This reward is used to calculate all the gradients of all the internal weights of the neural network applying back propagation. The gradients from different requests are stored in a buffer until a given batch size is reached. Then, all the gradients previously stacked are jointly applied to update the internal weights of the neural network, that done, the buffer is emptied. Note that, whereas it is possible to perform a gradient update for each successfully deployed request, adopting a batch processing strategy guarantees a faster and more stable training process. In the resource management domain, attributes, such as traffic load, residual resources, among others, are usually characterized by a certain predictable temporal correlation. Those repetitive patterns enable the agent to learn online as the system executes or offline by exploiting historical information. In this manuscript, we adopted the offline option in which the neural network was trained using offline demand sets of size 600 requests per epoch for a total of 200 epochs, considering a substrate network of 60 nodes. The results of the policy neural network training are shown in Fig. 4. From Figs. 4(b) and Fig. 4(c), the acceptance ratio and execution time per epoch is seen to increase with the training time. This is because the policy network accuracy improves on increasing the training time, thus resulting in a reduction in both wrong placement decisions and bandwidth resource consumption. Therefore, the number of admitted requests increases, consequently resulting in an increase in the execution time per training epoch, since admitted requests are characterized by additional steps such as the updating of the substrate resources and the computation of the mapping cost.

The performance of the CNN and FFN has recently been investigated in [39]. With a CNN model size which is 68 times smaller than the FFN, the CNN was found to result in a similar or better performance than the FFN when applied to the problem of speech enhancement. Such a good performance, with fewer trainable parameters, makes CNN based architectures more memory efficient, making them better candidates for edge-computing and multi-agent learning scenarios in which the different service life-cycle management decisions may need to be executed by distributed resource constrained nodes [40].

In Fig. 5, we compare the training performance of a convolutional neural network (CNN) with a feed forward network (FFN) architecture considering 50 substrate nodes and 400 offline demands per epoch. From the obtained results, both architectures converge to a similar performance in terms of training time, acceptance rate and reward value. Fig. 5(a) reveals that CNN converges earlier than FFN by approx. 5 epochs. This explains the slightly higher running time for the initial epochs in Fig. 5(c), since the number of requests admitted by CNN is higher than those of FFN during this stage.

V. Performance evaluation

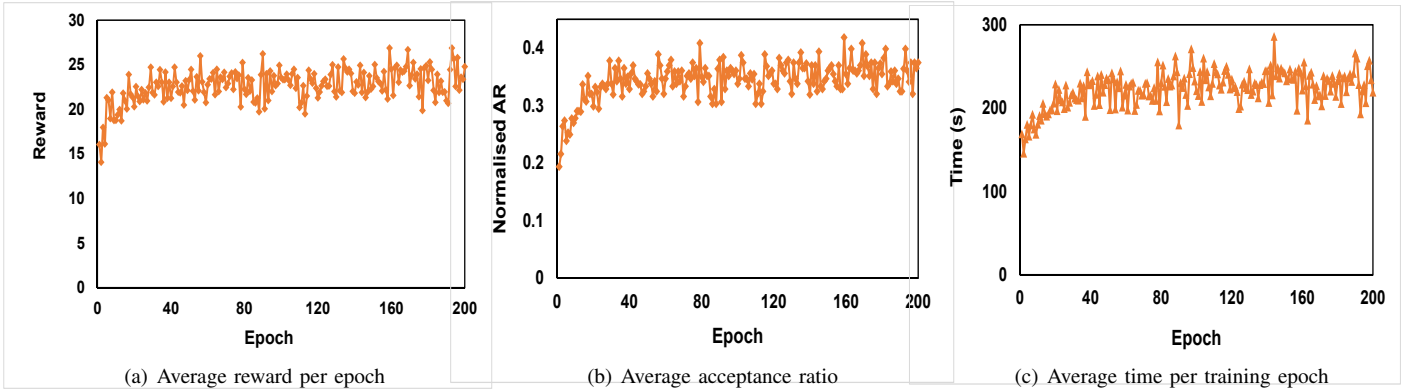
This section first explains the metrics considered for the evaluation of the performance of our proposal. It is followed by the description of the simulation settings and the introduction of four benchmark algorithms used for comparison with the RL algorithm. Finally, we include a discussion of the obtained results.

A. Performance Metrics

The proposed algorithm will be compared against several benchmark algorithms considering different performance metrics. Those metrics include the average acceptance ratio of requests (AR), the average deployment cost per accepted request, the average processing time of a request, and the average bandwidth utilisation, among others. Some of these are explained below:

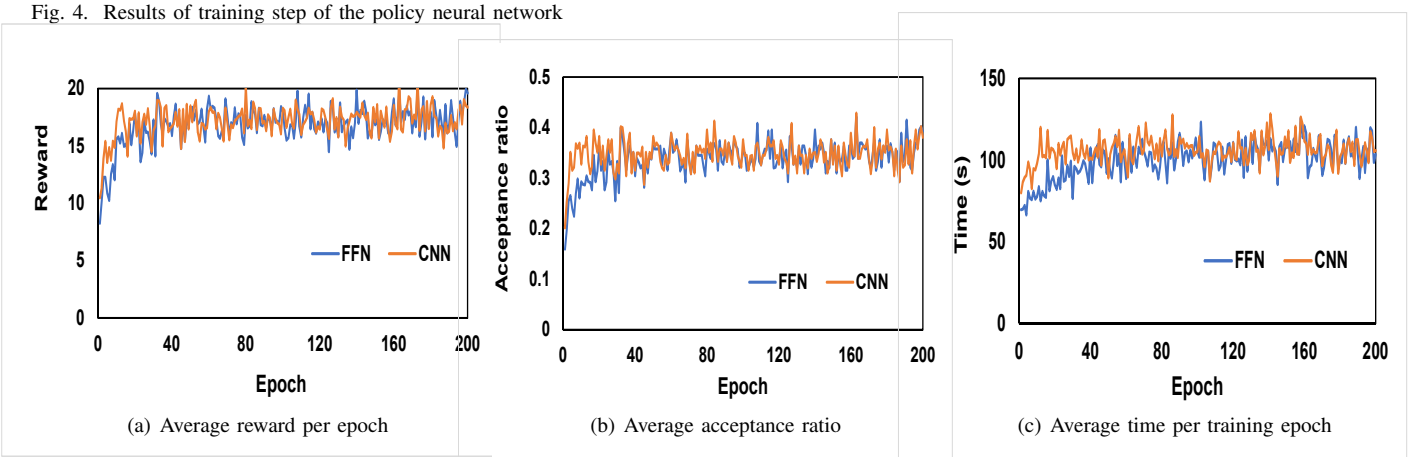
1) Average acceptance ratio, AR: This is computed as the ratio of the number of successfully accepted requests to the total number of arriving requests, i.e., the sum of both accepted and rejected. The AR metric is a direct indicator of the algorithms efficiency in using the underlying resources. Therefore, a service deployment algorithm should target to achieve a high AR performance in order to maximise the revenue received by a NSP. This is computed as follows:

$$AR = \frac{No.\ of\ embedded\ requests}{Total\ number\ of\ requests} \tag{25}$$



(a) Average reward per epoch (b) Average acceptance ratio (c) Average time per training epoch

Fig. 4. Results of training step of the policy neural network



(a) Average reward per epoch (b) Average acceptance ratio (c) Average time per training epoch

Fig. 5. Results of the training performance of the Feed-forward and convolutional neural networks

2) Average deployment cost, $Cost_{Dep}$: This denotes the cost of deploying each SFC request on average, and it is evaluated as:

$$Cost_{Dep} = \frac{1}{|R|}\sum_{r\in R} C(G_v)^r \tag{26}$$

where $C(G_v)^r$ denotes the implementation cost of request $r \in R$ as given in Eqn. 2.

3) Average bandwidth utilisation, BW_{util}: This metric quantifies the average substrate edge bandwidth resource utilization as the ratio of consumed bandwidth resources to total bandwidth resources averaged over all substrate edges. In order to reduce network congestion and service deployment costs, a deployment algorithm should target to achieve a low BW_{util} value. This is evaluated as follows:

$$Bw_{util} = \frac{1}{|E_s|}\sum_{e\in E_s} \frac{Bw^e}{B^e_{max}} \tag{27}$$

where Bw^e is the total bandwidth consumed on edge $e \in E_s$ and B^e_{max} is the bandwidth capacity of this edge.

4) Average request provisioning time, Avg_T: This is the time taken by the service deployment algorithm to compute a deployment solution for each admitted request on average. Aware that future services will have stringent latency requirements, a service deployment algorithm should have a low value of Avg_T. This is computed as:

$$Avg\_T = \frac{1}{|R_A|}|\sum_{r\in R_A} tim^r_{prov} \tag{28}$$

where $R_A \in R$ denotes a set of all admitted requests and tim^r_{prov} denotes the time taken by the algorithm to obtain a deployment solution for request $r \in R$.

B. Simulation environment and settings

For the evaluation of the proposed algorithms, we consider both real network topologies, in particular, the Abilene and BIC networks [9], with 10 and 33 nodes respectively, and synthetic topologies as adopted in [17], [18]. For the synthetic topologies, the number of substrate nodes are varied from 30 to 60, depending on the scenario under consideration, with an inter-node connection probability of 0.2. We consider the computing resources of each node to range from 60,000 units to 80,000 units, and the bandwidth capacity of each edge to be in the range of 400 $Mbps$ to 800 $Mbps$. The propagation delay on each substrate edge is in the range of 2 milliseconds to 5 milliseconds. The above settings are similar to those adopted in [9]. The cost of processing and transmitting 1GB of data at each node and link follows a uniform distribution $U(\$0.15, \$0.22)$ and $U(\$0.05, \$0.12)$ respectively. The processing delay of a packet at each VNF follows a uniform distribution $U(0.045ms, 0.3ms)$, with the

processing delay of a service chain being the sum of the processing delay of the constituent VNFs.

Each request $r \in R$ is generated with a random source τ_s^r and a random destination τ_d^r from G_s, with $\tau_s^r \neq \tau_d^r$, and with a packet rate ρ measured in packets/s following a uniform distribution $U(400, 4000)$. The delay requirement of each request follows a uniform distribution $U(10ms, 30ms)$. We consider 5 categories of network functions: Firewalls, Proxies, NATs, DPIs and Load Balancers, with their computing resource demands adopted from [41]. Similar to [18], the number of VNFs constituting each SFC instance is varied depending on the scenario under consideration. The specific values of the different simulation parameters are shown in Table IV.

In this work, we consider 2 major kinds of request behaviour scenarios i.e. offline and online scenarios. In the offline case, all the requests to be served, including their attributes, are known in advance, and these, once admitted, do not leave the system for the entire simulation window time. Therefore, the resources allocated to these requests cannot be reused by other demands. The offline consideration gives a clearer insight into the algorithm's ability to deal with permanent loading stress [42]. In the online case, the demands continuously arrive to the system with a given arrival distribution and with a finite life-time. In this case, the resources assigned to an admitted request are reclaimed upon expiry of this demand. We consider the arrival of such requests to follow a Poisson distribution with a mean value chosen according to the scenario and experiment under consideration. The life-time of each online request is exponentially distributed with a mean value of 500 units of time.

Moreover, all simulations were conducted on a desktop computer running the Windows Operating System, and with the following features: Intel(R) Core(TM) i7-8700K CPU @ 3.70GHZ and 64GB of RAM. In order to obtain stable performance values, we consider 10 trials for each arrival rate / substrate network size / request size. For each trial, a new set of requests and network topology are generated.

TABLE IV
SIMULATION PARAMETERS

Substrate Network:

Parameter	Value
No. substrate nodes	11-60
Node CPU	$U(60000, 80000)$
Edge bandwidth	$U(400, 800)$ Mbps
Edge propagation delay	$U(2, 5)$ milliseconds
Processing cost per 1 GB	$U(\$0.15, \$0.22)$
Transmission cost per 1 GB	$U(\$0.05, \$0.12)$
processing delay at each VNF	$U(0.0045, 0.3)$ milliseconds
β_w	\$ 0.01
e_{max}	2735
e_{idle}	80.5
$\gamma_c^{n_v^p}$	\$ 0.22 and
$\gamma_{bw}^{e_v}$	\$ 0.12
Fragmentation penalty, α^{n_s}	\$0.01
Packet rate	$U(400, 4000)$
No.VNFs	$U(2, 10)$
Mean arrival rate	20
Life-time	exponential with 500 (mean)

C. Benchmark Algorithms

The proposed RL-based service deployment algorithm is compared with a state-of-the-art multi-stage graph based algorithm (denoted as graph-based in this section) proposed in [28] whose operation was described in section II. In our view, the choice of this work is justified since the work is recent, and according to the authors of that work, it was found to give near-optimal results. In addition, we evaluate our algorithm against a brute-force algorithm and three greedy algorithms which are well known in literature. These are discussed below:

1) Brute-force algorithm (Brut): For a given request to be provisioned, the brute-force algorithm obtains all possible mapping combinations for that request. This is then followed by extracting all feasible solutions (i.e. solutions that do not violate the request and substrate network constraints). Then, from all feasible solutions, the solution that results in the least deployment cost is selected for provisioning that service request. Since such an approach explores all mapping possibilities, it results in the optimal solution, albeit with the penalty of the excessive run-time employed to obtain that one.

2) Bandwidth Greedy Algorithm (BwGA): This algorithm targets to jointly minimise: the amount of bandwidth resources used to provision each request, the node resource fragmentation and the VNF deployment cost. This will be done by provisioning each request on the shortest feasible path between the ingress and the egress nodes and using the minimum number of nodes. The pseudo-code of BwGA is shown in Algorithm 1. First, the algorithm computes the set $Path_K^{\tau_s^r, \tau_d^r}$ that contains the K shortest paths between the ingress and egress nodes, with the paths being sorted in increasing length order. Starting from the first path in the sorted list, the algorithm checks for the validity of that path in terms of end-to-end delay, available bandwidth and availability of CPU resources along the path to map the different request virtual nodes. For a given valid path, the physical nodes along that path are sorted in increasing order of their residual resources. Then, the minimum number of nodes with the least amount of residual resources that are required to support the entire SFC instance are selected. Then, the VNFs of the request are placed on the deduced nodes along that path. The choice of the nodes with the least amount of residual resources for provisioning the virtual nodes targets to minimise the energy and resource fragmentation costs, since such nodes are more likely to have VNFs already active on them, preventing the activation of new nodes and VNFs.

3) Greedy activation algorithm (GAA): This algorithm targets to minimize the deployment cost by greedily minimizing the VNF deployment cost thanks to reusing as much as possible already active VNF instances. If an instance of a given VNF is not active or the node resources are not sufficient to enable its sharing, then, the new virtual node is placed on a substrate node that is closest to the host of the preceding virtual node (or the ingress node in case of mapping the first virtual node). The pseudo-code of this algorithm is shown in

Algorithm 1: BwGA Algorithm

Input: G_s, Ψ^r
Initialise: Deployment_solution =*None*
Compute $Path_K^{\tau_s^r, \tau_d^r}$.
if $|Path_K^{\tau_s^r, \tau_d^r}|$=0 **then**
  reject request
  return
end
for $path_k \in Path_K^{\tau_s^r, \tau_d^r}$ **do**
  $check\_path\_validity(path_k)$
  if $path_k$ *is valid* **then**
    Extract_minimum_host_nodes($path_k, \Psi^r$)
    Map the request
    if *mapping is feasible* **then**
      Deployment solution=$path_k$
      Terminate for loop
    end
  end
end
if $Deployment\_solution$=*None* **then**
  Reject request
end
else
  Return Deployment_solution
end

Algorithm 2: GAA Algorithm

Input: G_s, Ψ^r
Output: $Deployment\_solution$
Initialise: $prev\_host$=τ_s^r; $node\_solution$ = []
for $n_v^p \in N_v$ **do**
  $Cands_{n_v^p}$=$Compute\_candidates(G_s, \Psi^r)$
  if $Cands_{n_v^p}$=$\emptyset$ **then**
    Reject request
    break
  end
  else
    Extract $Cands_{n_v^p}^{act}$ from $Cands_{n_v^p}$
    if $Cands_{n_v^p}^{act} \neq \emptyset$ **then**
      $Cands$=$Sort\_dist(Cands_{n_v^p}^{act}, prev\_host)$
      $n_v^p \leftarrow$Cands[0]
      $node\_solution.append(Cands[0])$
      $prev\_host$=Cands[0]
    end
    else
      Cands=$Sort\_dist(Cands_{n_v^p}, prev\_host)$
      $n_v \leftarrow$Cands[0]
      $node\_solution.append(Cands[0])$
      $prev\_host$=Cands[0]
    end
  end
end
Run $link\_mapping(node\_solution, G_s, \Psi^r)$
if *Sucessful* **then**
  Deploy request
end
else
  Reject request
end

Algorithm 2. For each traffic node to be mapped, the function $compute\_candidates()$ computes a set $Cands_{n_v^p}$ consisting of the candidate substrate nodes for virtual node $n_v^p \in N_v$. Then, if there are substrate nodes with already active VNFs to support n_v^p, among these, the node closest to the host of the preceding virtual node is chosen to provision the current virtual node. Otherwise, all candidates are sorted according to the distance to the host of the preceding virtual node, with the closest candidate being chosen to provision the current traffic node. Once all the virtual nodes are provisioned, then the host for the virtual links are computed using the shortest available paths between adjacent nodes of the selected substrate nodes.

4) Load balance based algorithm (LBA): For each virtual node of the request to be provisioned, this algorithm ranks all candidate nodes according to their resource score, which is calculated as the product of the node's computational resources by the accumulative residual bandwidth of the node's inbound links. Then, the candidate node with the highest score is selected for hosting the current virtual node. In this way, the algorithm targets to minimise node and link resource bottlenecks, guaranteeing a good long term acceptance ratio performance. The score of a given node $n_s \in N_s$ is computed as:

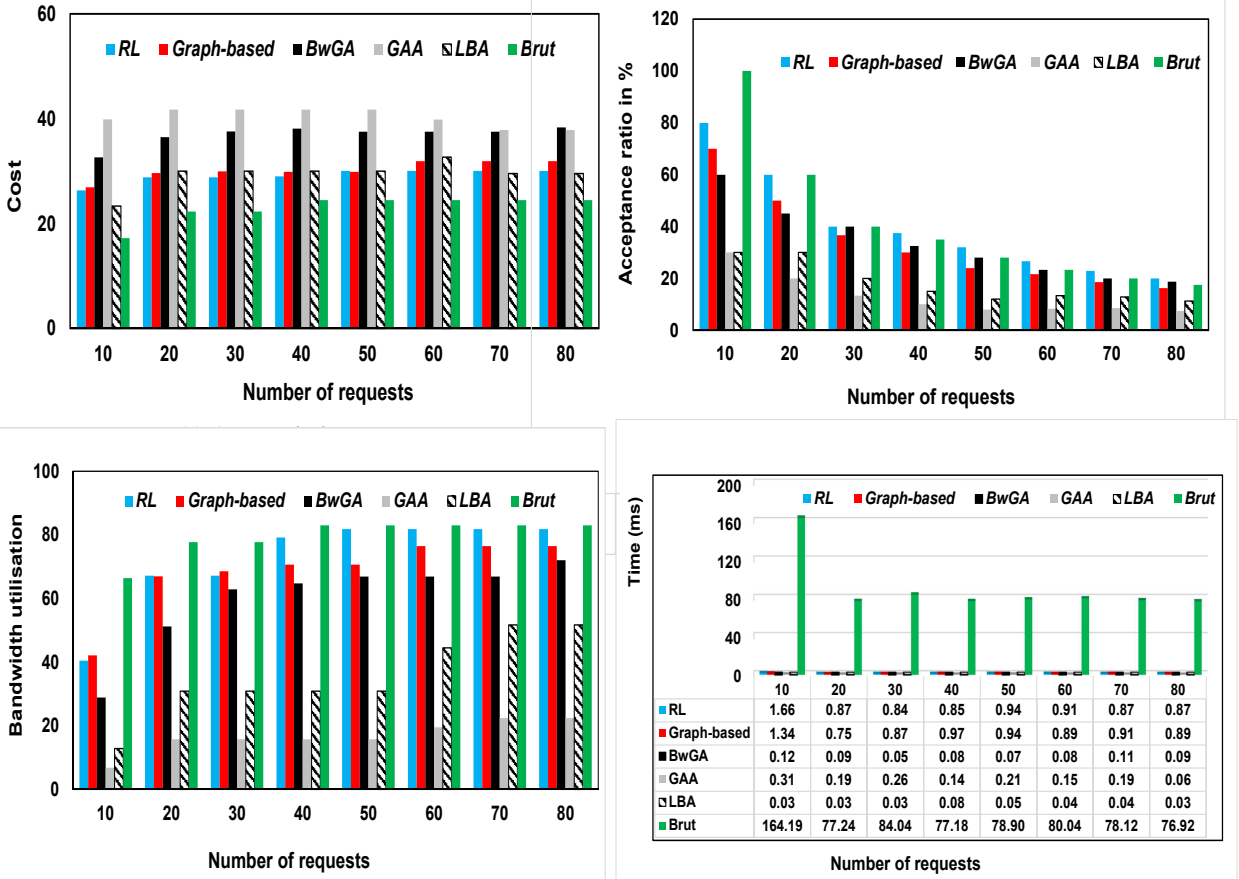
$$Score^{n_s} = C_{res}^{n_s} \times \sum_{e \in E_{adj}^{n_s}} B_{res}^{e} \tag{29}$$

where $C_{res}^{n_s}$ denotes the available computational resources at the node and B_{res}^{e} denotes the residual bandwidth resources at an inbound edge $e \in E$. The pseudo-code of this algorithm is shown in Algorithm 3. Once all virtual nodes have been provisioned, then, the end-to-end traffic link is obtained by running the Dijkstra algorithm in between each pair of the hosting nodes, while updating the available edge resources.

D. Results and discussion

In this section, we analyze the performance of the proposed algorithm against the benchmark algorithms discussed in section V-C considering a number of scenarios as discussed below:

1) Performance with respect to the optimal solution: In experiment 1, whose results are shown in Fig. 6, we analyse the performance of the proposed algorithm in comparison with the optimal (brute-force) algorithm considering an offline case while varying the number of requests. Due to the high time consuming of the brute-force algorithm, the comparison has been done using the Abilene topology with 11 nodes. The results in Fig. 6(a) show that *Brut* results in the best performance in terms of deployment cost per request with an average value of \$23, averaged over all request numbers. This is followed by RL, LBA, $Graph-based$, $BwGA$ and GAA with average values of \$27, \$29.38, \$30.25, \$37.0 and \$40.31, respectively. Therefore, RL is within a 14% margin of



(c) Average Bandwidth utilisation

(d) Average request provisioning time

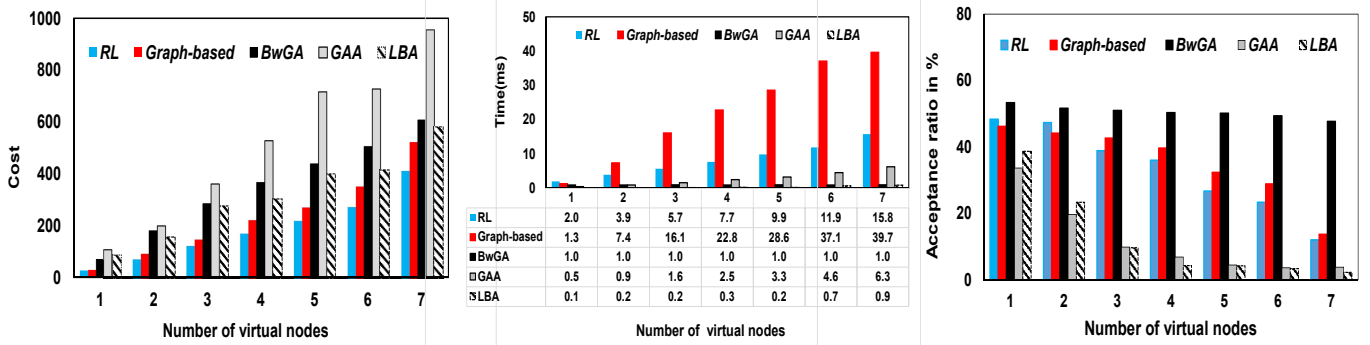
Fig. 6. Results of experiment 1 considering the Abilene topology in which the algorithm performance is evaluated for different numbers of offline requests

the optimal solution and results in more than approx. an 8% improvement compared to $Graph-based$ and LBA, 27% compared to $BwGA$ and 33% compared to GAA.

Moreover, the RL performance in terms of AR, as shown in Fig. 6(b), is within a 1% margin of $Brut$, with an average value of 39.88% compared to 40.48% from $Brut$. This becomes a performance improvement of approximately 6% compared to $BwGA$ and $Graph-based$, 21.8% compared to LBA, and 26.66% compared to GAA, whose average AR values are 33.45%, 33.40%, 18.06% and 13.22%. $BwGA$ emerges competitive with respect to $Graph-based$, since it maps requests on the shortest possible paths, resulting in a low bandwidth utilization, which was in fact the bottleneck resource, as reflected from the high bandwidth consumption shown in Fig. 6(c). However, even with this desirable behaviour, it remains inferior to RL since it may greedily reuse the shortest paths between any source and destination, which may result in resource bottlenecks in the long term, yet, RL is able to intelligently trade-off the resource consumption and cost in order to guarantee a good long term performance. The greedy nature of GAA and LBA may result in virtual nodes being mapped far from each other, which may result in a high consumption of link resources, leading to link resource bottlenecks. Moreover, this also increases the likelihood of failure to obtain a feasible substrate path for hosting the corresponding virtual links due to delay constraints. This is evident from Fig. 6(c) where GAA and LBA result in the lowest values of bandwidth utilisation with average values of 16.8% and 35.54% respectively, demonstrating that most of the link resources remain unused due to link bottlenecks and the failure to meet delay constraints.

From Fig. 6(d), $Brut$ results in more than a 98.9% overhead in terms of the average time for provisioning each request compared to the other algorithms, with an average value of 89.58 milliseconds for the considered topology. The rest of the algorithms are provisioning each request in a fraction of a millisecond with average values of 0.98, 0.94, 0.10, 0.20 and 0.04 milliseconds for RL, $Graph-based$, $BwGA$, LBA and GAA respectively, further demonstrating that the proposed RL-based algorithm is well suited for provisioning delay sensitive applications in resource constrained networks.

2) Impact of request size: In experiment 2, whose results are shown in Fig. 7, we analyse the impact of the request size by varying the number of requested virtual nodes from 1 to 7, and considering BIC network topologies for each request size. The experiment targets to demonstrate the scalability of the proposed algorithm as the size of the requests increases. From the results in Fig. 7(a), RL results in an average value of $186.8 in terms of service deployment cost for each request, resulting in a 20.72%, 46.93%, 63.71% and



	1	2	3	4	5	6	7
RL	2.0	3.9	5.7	7.7	9.9	11.9	15.8
Graph-based	1.3	7.4	16.1	22.8	28.6	37.1	39.7
BwGA	1.0	1.0	1.0	1.0	1.0	1.0	1.0
GAA	0.5	0.9	1.6	2.5	3.3	4.6	6.3
LBA	0.1	0.2	0.2	0.3	0.2	0.7	0.9

(a) Average deployment cost (b) Average request provisioning time (c) Average acceptance ratio

Fig. 7. Experiment 2 in which the impact of request size is analysed by varying the number of virtual nodes per request from 1 to 7 considering BIC topology and 30 offline requests for each number of nodes.

Algorithm 3: LBA Algorithm

Input: G_s, Ψ^r
Output: $Deployment\_solution$
Initialise: $prev\_host$=τ_s^r; $node\_solution$ = []
for $n_v^p \in N_v$ **do**
  $Cands_{n_v^p}$=$Compute\_candidates(G_s, \Psi^r)$
  if $Cands_{n_v^p}$=$\emptyset$ **then**
    Reject request
    break
  end
  else
    Cands=$Sort\_Score(Cands_{n_v^p}, G_s)$
    n_v^p←Cands[0]
    $node\_solution.append(Cands[0])$
  end
end
Run $link\_mapping(node\_solution, G_s, \Psi^r)$
if *Sucessful* **then**
  Deploy request
end
else
  Reject request
end

40.65% improvement compared to $Graph-based$, $BwGA$, GAA and LBA respectively, whose average cost values are \$235.62, \$352.05, \$514.74 and \$314.77 respectively. These results reveal that: *i*) the average deployment cost of each algorithm under the BIC topology is higher than under the Abilene topology. This is due to the fact that BIC has more nodes and links, hence, the probability of activating new VNFs and traversing longer substrate paths increases, resulting in increased deployment costs; *ii*) The greedy algorithms, namely LBA, $BwGA$ and GAA, result in up to a 40% overhead in terms of cost, demonstrating the inefficiency of greedy approaches for solving problems in which the objective is influenced by multiple attributes; *iii*) even for a medium size topology such as BIC, RL remains superior over the state-of-the-art multi-layer graph based approach in terms of cost, service provisioning time and AR.

From the results in Fig. 7(c), the AR performance of all algorithms tends to decrease with the increase of the request size. This is partly due to the increased resource consumption, and partly due to the increased probability of failure to satisfy the end-to-end constraints of the request. $BwGA$ results in the highest value of AR with an average value of 49.5% averaged across all request sizes. RL and $Graph-based$ result in a similar performance (within a 2% margin) with average values of 33.52% and 35.3% respectively, followed by LBA and GAA with 12.22% and 11.68% respectively. The results demonstrate that, even while achieving above a 20.72% improvement in terms of request provisioning cost over $Graph-based$, RL remains competitive in terms of AR. The $BwGA$ algorithm results in a high AR performance since it greedily targets to map requests using the least possible bandwidth resources at the expense of the request provisioning cost, the objective addressed in this paper. On the other hand, RL and $Graph-based$ may need to map virtual nodes further away from each other in order to minimise the VNF activation and the resource fragmentation costs, whenever such a decision results in a lower service deployment cost. However, this is achieved at the expense of an increased bandwidth resource consumption, especially as the network size increases. Moreover, this also explains the slightly higher performance of $Graph-based$ approach over RL in terms of AR, since RL is more strict in optimizing the placement objective of provisioning cost compared to $Graph-based$ which may provision requests on shorter paths (hence less bandwidth consumption and request rejection rate) at the expense of increased cost.

From the results in Fig. 7(b), RL provisions each request on average in 8.12 milliseconds, which is 62.81% faster than $Graph-based$, whose processing time per request is 21.86 milliseconds. LBA, GAA and $BwGA$ result in the lowest provisioning times, with average values in milliseconds of 0.36, 0.98 and 2.81 respectively, albeit at the expense of a high request deployment cost. As the number of required VNFs per request increases, the run time of $Graph-based$ experiences a drastic increase since this increases the number of layers of its multi-layer graph, increasing the computation steps of the algorithm. On the contrary, the computational complexity in RL relies on a feature matrix whose size is dependant on

the substrate network size, hence, a change in the request size only affects on the number of such computations.

3) Impact of substrate network size: In experiment 3, we analyse the impact of the substrate network size on the algorithms' performance by varying the substrate nodes from 10 to 60, with the results shown in Fig. 8. The aim of this experiment is twofold: *i*) to assess the impact of the substrate network size on the performance of the proposed algorithm regarding the aforementioned performance metrics; and *ii*) to demonstrate the generalization capability of the proposed policy neural network regarding the use of substrate networks with sizes different from the one used for training, without the need of retraining the neural network. From the results in Fig. 8(a), the average cost for the deployment of each request tends to decrease as the number of nodes increases, that applies for most of the algorithms. This is due to an increase in the availability of resources, which increases the number of available options for deploying a service request at a lower cost. Moreover, RL results in an average request deployment cost of \$82.19, which becomes a percentage improvement of 26.2%, 65.3%, 34.2% and 58.7% compared to $Graph-based$, $BwGA$, GAA and LBA, whose average values are \$111.40, \$236.87, \$124.9 and \$199.04 respectively. The results reveal that the performance gain of using RL in terms of cost tends to increase as the substrate network increases. This is because, as the network size increases, the number of alternative nodes and links for provisioning the request increases, which complicates the decision making for the other algorithms. Thanks to its intelligence, RL is able to select the optimal nodes and links for the provisioning of the request among the multiple alternatives.

Moreover, Fig.8(b) reveals that RL is only 5% inferior in terms of AR compared to $Graph-based$, with an average AR value of 55.4% compared to 60.3% of $Graph-based$. $BwGA$, GAA and LBA result in AR average values of 54.87%, 25.0% and 29.05%, demonstrating the superiority of RL due to intelligently trading-off the service deployment objective and the resource utilisation efficiency in comparison with the alternative approaches. Moreover, the policy neural network is able to make efficient placement decisions, even for substrate networks whose size is inferior to the one used at the training stage, demonstrating the generalization capability of our adopted approach using dummy nodes when needed.

In terms of execution time, $Graph-based$ results in up to a 86% overhead, with an average processing time of 53.68 milliseconds per admitted request, compared to an average value of 7.13 milliseconds for RL. As the number of substrate nodes increases, the run time of $Graph-based$ greatly increases, since this fact increases the number of possible candidates for each virtual node of the request (hence, increasing the number of nodes at each layer of the multi-layer graph), and consequently, increasing the time-complexity of the multi-layer graph.

The above results also demonstrate a good generalization capability of RL for substrate networks whose size is different from that used at the training stage. For instance, when considering 50 substrate nodes, RL outperforms $Graph-based$ by 52.7% in terms mapping cost and by 53.45% when considering 60 substrate nodes that are used for training the policy network. This translates to less than a 1% margin.

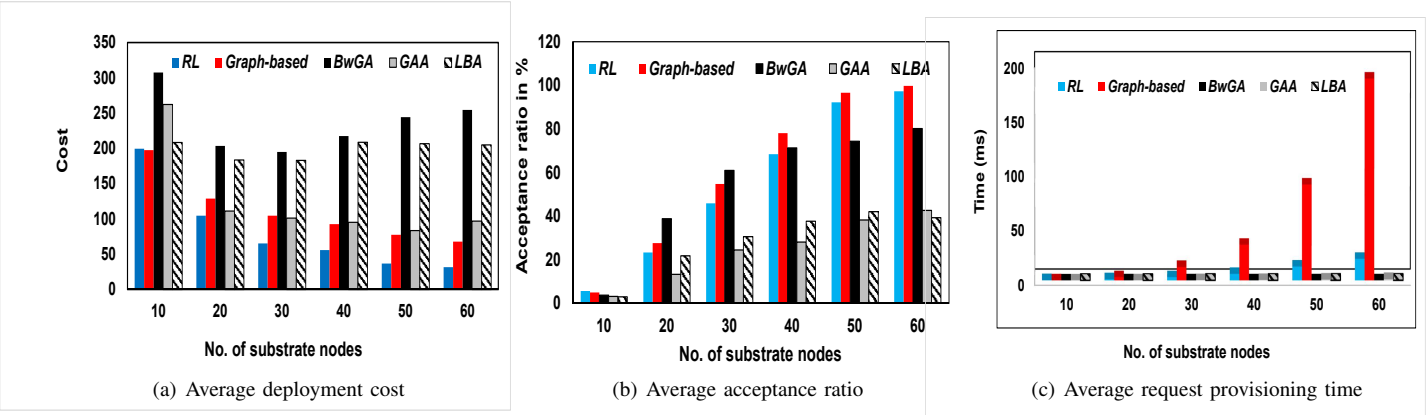
A summary of the results from the above simulation scenarios is given in Table. V where a positive("+") indicates that RL is superior to a given benchmark algorithm by the indicated percentage value while the negative ("-") indicates that RL is inferior to the corresponding algorithm by the indicated percentage value in the corresponding metric.

4) Online behaviour of the algorithms: From the results of Fig. 9, corresponding to experiment 4 of the performance analysis, we analyse the behaviour of the algorithms while considering a Poisson arrival of the requests with a mean value of 20 requests on each interval of 100 time units, for a total of 12,000 time units, which corresponds to a total of 2,400 requests. From Fig. 9(a), the deployment cost for the algorithms tends to decrease along the time, with the decrease being dominant for $BwGA$, LBA and GAA. This is because, as new requests arrive, most of the VNFs remain activated, which decreases the VNF activation cost. Moreover, the number of feasible nodes and links decreases with the arrival of new requests, which simplifies the mapping decision of the greedy algorithms. On average, RL results in the lowest mapping cost per request, with an average value of \$13.91, followed by GAA, LBA, $Graph-based$ and $BwGA$, with average values of \$14.01, \$14.61, \$15.41 and \$16.31 respectively. Moreover, from the results in Fig. 9(c), the performance gain in terms of AR for $BwGA$ and $Graph-based$ with respect to RL decreases. This is due to the fact that, for the online case, resources are returned back to the network upon expiry of admitted requests, making the bandwidth resource usage not as much constrained as the offline scenario which is characterized by a permanent loading. The average AR values of the algorithms are; 90.50%, 88.82%, 92.90%, 68.57%, and 84.82% for RL, $Graph-based$, $BwGA$, GAA,and LBA respectively.

From Fig. 9(b), RL provisions each request in less than 8.52 milliseconds, which is 70.3% faster than the alternative $Graph-based$ algorithm. The running time of $Graph-based$ decreases along the time due to a reduction in the number of feasible nodes for provisioning each virtual node of the request. This results in a reduction in the number of candidate nodes at each layer of the multi-layer graph, decreasing the time-complexity for obtaining a mapping solution.

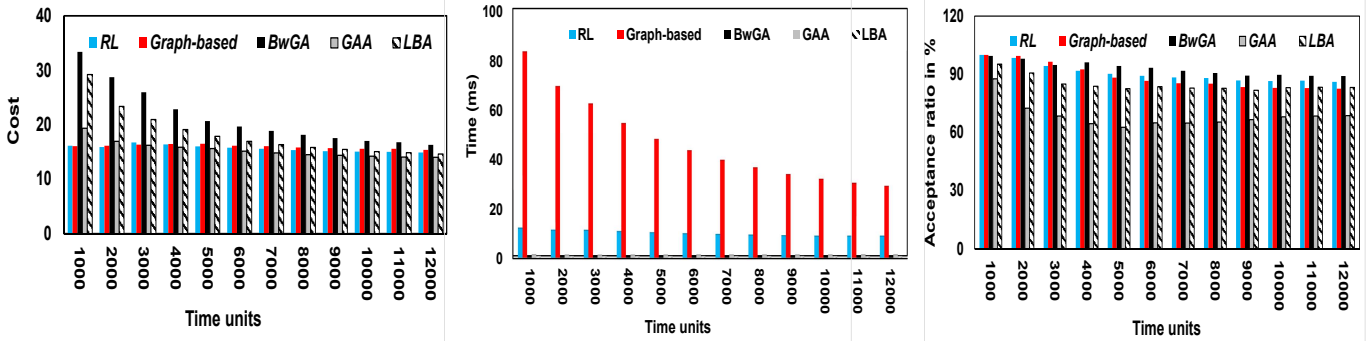
VI. Conclusion

This paper has addressed the problem of cost-effective and resource-efficient deployment of service requests with the possibility of sharing VNFs among multiple service requests, and under multiple conflicting cost components, including: resource fragmentation, VNF activation, energy consumption, packet processing and traffic forwarding. This paper has proposed a RL-based algorithm whose policy neural network can be adopted for making deployment decisions for substrate networks of different size, as long as that size is smaller than the one used for training the neural network. From the simulation results, the proposed algorithm has been found to be optimal in terms of acceptance ratio, placement cost



(a) Average deployment cost (b) Average acceptance ratio (c) Average request provisioning time

Fig. 8. Experiment 3 in which the impact of substrate network size is analysed by varying the number substrate nodes from 10 to 60 considering synthetic network topologies with the number of offline requests fixed at 200



(a) Average deployment cost (b) Average request provisioning time (c) Average acceptance ratio

Fig. 9. Experiment 4 analysing the online behaviour of algorithms considering 10 arrivals per 100 time units for a total of 12000 time units considering BIC

TABLE V
HIGHLIGHTS OF THE SIMULATION RESULTS

Simulation scenarios	Algorithm	Performance metrics		
		AR	Time	Cost
Performance with respect to optimal solution	Brut	-1%	+98.9%	-14/%
	Graph based	+6%	-4%	+8%
	BwGA	+6%	-89%	+27%
	GAA	+26.6%	-96.7%	+33%
	LBA	+21%	-79.5%	+8%
Impact of request size	Graph based	-2%	+62.8%	+20.7%,
	BwGA	-16%	-64.8%	+46.9%
	GAA	+21.9%	-87.8%	+64.7%
	LBA	+21.5%	-95.5%	+40.6%
Impact of substrate network size	Graph based	-5%	+86%	26.2%
	BwGA	+1%	-98.8%	65.3%
	GAA	+30.4%	-92.9%	34.2%
	LBA	+25.4%	-99.8%	58.7%

and request provisioning time. The algorithm results in a performance similar to the brute-force algorithm in terms of AR while executing in less than 98% of the time required by the brute-force algorithm. In terms of service deployment cost, the proposed algorithm obtains solutions which are within a 14% margin of the optimal one; and results in up to a 20% and a 40% improvement in comparison with a state-of-the-art algorithm and a set of algorithms that greedily target to minimise only one cost component, respectively. Moreover, in some scenarios, the proposed algorithm is found to provision each request within up to 70% less time compared to a state-of-the-art multi-layer graph based algorithm. Moreover, the proposed algorithm has been found to be scalable under changes in both network and request size, exhibiting good generalized properties. Thanks to the intelligence of the proposed algorithm, the above results have demonstrated that the proposed algorithm is well suited for the deployment of delay sensitive service requests under resource constrained networks, and where the placement objective is jointly influenced by multiple conflicting costs.

Traditional heuristic approaches target to solve optimisation problems at the current time, without regarding the impact of the taken decisions on the future network state and performance. This is because, unlike AI based techniques, such approaches are devoid of the intelligence necessary to infer the long term impact of the different parameters influencing the network performance. Aware that network and service management problems are usually constrained by multiple conflicting and sometimes dynamic parameters, artificial intelligence techniques, such as RL, are promising approaches to be applied in this problem domain.

This work has considered the various service requests to have fixed requirements in terms of node and link resources throughout their lifetime. However, in practice, the resource requirements of any service may experience temporal variations, making it necessary to scale up or down some of its shared VNFs and/or migrate the service when such scaling

can not be performed. This will result in additional costs related to VNF scaling to be considered, service migration or service level agreement violation costs, among others, which directly impact the net profit obtained by the service provider. As part of our future work, we target to address the deployment of SFCs with the appropriate elasticity behaviour by exploiting the predictive capability of machine learning techniques for making service deployment decisions that are cognizant of future scaling and migration requirements of the service requests.

REFERENCES

[1] B. Yi, X. Wang, and M. Huang, "A Generalized VNF Sharing Approach for Service Scheduling," *IEEE Communications Letters*, vol. 22, no. 1, pp. 73–76, 2018.

[2] Y. Zhang, F. He, T. Sato, and E. Oki, "Network Service Scheduling with Resource Sharing and Preemption," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 764–778, 2020.

[3] G. Kibalya, J. Serrat, J. L. Gorricho, R. Pasquini, H. Yao, and P. Zhang, "A Reinforcement Learning Based Approach for 5G Network Slicing across Multiple Domains," in *15th International Conference on Network and Service Management, CNSM 2019*, 2019.

[4] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, no. November, 2020.

[5] P. Kaliyammal Thiruvasagam, V. J. Kotagi, and C. S. R. Murthy, "The More the Merrier: Enhancing Reliability of 5G Communication Services With Guaranteed Delay," *IEEE Networking Letters*, vol. 1, no. 2, pp. 52–55, 2019.

[6] A. Mohamad, "On Demonstrating the Gain of SFC Placement with VNF Sharing at the Edge," 2019.

[7] G. Sun, Z. Xu, H. Yu, X. Chen, V. Chang, and A. V. Vasilakos, "Low-latency and Resource-efficient Service Function Chaining Orchestration in Network Function Virtualization," *IEEE Internet of Things Journal*, vol. PP, no. c, pp. 1–1, 2019.

[8] G. Sun, Y. Li, D. Liao, and V. Chang, "Service function chain orchestration across multiple domains: A full mesh aggregation approach," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1175–1191, 2018.

[9] W. Mao, L. Wang, J. Zhao, and Y. Xu, "Online Fault-tolerant VNF Chain Placement: A Deep Reinforcement Learning Approach," *IFIP Networking 2020 Conference and Workshops, Networking 2020*, pp. 163–171, 2020.

[10] L. Qu, C. Assi, and K. Shaban, "Delay-Aware Scheduling and Resource Optimization with Network Function Virtualization," *IEEE Transactions on Communications*, vol. 64, no. 9, pp. 3746–3758, 2016.

[11] D. Dietrich, A. Rizk, and P. Papadimitriou, "Multi-domain virtual network embedding with limited information disclosure," *2013 IFIP Networking Conference, IFIP Networking 2013*, vol. 12, no. 2, pp. 188–201, 2013.

[12] ——, "Multi-Provider Virtual Network Embedding With Limited Information Disclosure," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 188–201, 2015.

[13] D. Dietrich, A. Abujoda, A. Rizk, and P. Papadimitriou, "Multi-Provider Service Chain Embedding With Nestor," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 91–105, 2017.

[14] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Multi-objective scheduling of micro-services for optimal service function chains," *IEEE International Conference on Communications*, 2017.

[15] G. Kibalya, J. Serrat, J.-l. Gorricho, J. Serugunda, and P. Zhang, "A multi-stage graph based algorithm for survivable Service Function Chain orchestration with backup resource sharing," *Computer Communications*, vol. 174, no. December 2020, pp. 42–60, 2021. [Online]. Available: https://doi.org/10.1016/j.comcom.2021.04.008

[16] T. A. Q. Pham, Y. Hadjadj-aoul, and A. Outtagarts, "VNF-FG embedding: A deep reinforcement learning approach," *IEEE Transactions on Network and Service Management*, pp. 1–10, 2019.

[17] B. Yang, Z. Xu, W. K. Chai, W. Liang, D. Tuncer, A. Galis, and G. Pavlou, "Algorithms for fault-tolerant placement of stateful virtualized network functions," *IEEE International Conference on Communications*, vol. 2018-May, 2018.

[18] G. Yuan, Z. Xu, B. Yang, W. Liang, W. Koong, D. Tuncer, A. Galis, G. Pavlou, and G. Wu, "Fault tolerant placement of stateful VNFs and dynamic fault recovery in cloud networks," *Computer Networks*, vol. 166, p. 106953, 2020. [Online]. Available: https://doi.org/10.1016/j.comnet.2019.106953

[19] D. N. Heo, S. Lange, H. G. Kim, and H. Choi, "Graph neural network based service function chaining for automatic network control," *APNOMS 2020 - 2020 21st Asia-Pacific Network Operations and Management Symposium: Towards Service and Networking Intelligence for Humanity*, pp. 7–12, 2020.

[20] H. Guo, Y. Wang, Z. Li, X. Qiu, H. An, P. Yu, and N. Yuan, "Cost-aware Placement and Chaining of Service Function Chain with VNF Instance Sharing," *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2020: Management in the Age of Softwarization and Artificial Intelligence, NOMS 2020*, 2020.

[21] F. Malandrino, C. F. Chiasserini, G. Einziger, and G. Scalosub, "Reducing service deployment cost through VNF sharing," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2363–2376, 2019.

[22] A. Mohamad, "PSVShare : A Priority-based SFC placement with VNF Sharing," pp. 25–30, 2020.

[23] F. Malandrino and C.-f. Chiasserini, "Getting the Most Out of Your VNFs : Flexible Assignment of Service Priorities in 5G," pp. 1–9.

[24] T. Truong-huu, P. M. Mohan, and M. Gurusamy, "Service Chain Embedding for Diversified 5G Slices With Virtual Network Function Sharing," vol. 23, no. 5, pp. 2019–2022, 2019.

[25] B. Yi, X. Wang, M. Huang, and A. Dong, "A multi-criteria decision approach for minimizing the influence of VNF migration," vol. 159, pp. 51–62, 2019.

[26] Y. Zhang, F. He, T. Sato, and E. Oki, "Optimization of Network Service Scheduling with Resource Sharing and Preemption," 2019.

[27] H. A. Alameddine, L. Qu, and C. Assi, "Scheduling service function chains for ultra-low latency network services," *2017 13th International Conference on Network and Service Management, CNSM 2017*, vol. 2018-Janua, pp. 1–9, 2017.

[28] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, O. Carlos, and M. Bandeira, "Orchestrating Virtualized Network Functions," vol. 13, no. 4, pp. 725–739, 2016.

[29] H. Chai, J. Zhang, Z. Wang, J. Shi, and T. Huang, "A Parallel Placement Approach for Service Function Chain Using Deep Reinforcement Learning," *2019 IEEE 5th International Conference on Computer and Communications, ICCC 2019*, pp. 2123–2128, 2019.

[30] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," *Proceedings of the International Symposium on Quality of Service, IWQoS 2019*, no. 1, 2019.

[31] M. Nakanoya, Y. Sato, and H. Shimonishi, "Environment-adaptive sizing and placement of NFV service chains with accelerated reinforcement learning," *2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM 2019*, pp. 36–44, 2019.

[32] H. Xuan, X. Zhao, J. Fan, Y. Xue, F. Zhu, and Y. Li, "VNF Service Chain Deployment Algorithm," vol. 48, no. 1, 2021.

[33] R. Wang, J. Li, K. Wang, X. Liu, and X. Li, "Service function chaining in NFV-enabled edge networks with natural actor-critic deep reinforcement learning," *2021 IEEE/CIC International Conference on Communications in China, ICCC 2021*, no. Iccc, pp. 1095–1100, 2021.

[34] S. Guo, Y. Qi, Y. Jin, W. Li, X. Qiu, and L. Meng, "Endogenous Trusted DRL-Based Service Function Chain Orchestration for IoT," *IEEE Transactions on Computers*, vol. 71, no. 2, pp. 397–406, 2022.

[35] S. R. de Araújo Júnior and R. A. C. Bianchi, "A Model for Traffic Forwarding through Service Function Chaining using Deep Reinforcement Learning Techniques," pp. 619–630, 2021.

[36] A. Nouruzi, A. Zakeri, M. R. Javan, N. Mokari, R. Hussain, and A. S. Kazmi, "Online Service Provisioning in NFV-enabled Networks Using Deep Reinforcement Learning," pp. 1–13, 2021. [Online]. Available: http://arxiv.org/abs/2111.02209

[37] J. J. A. Esteves, A. Boubendir, F. Guillemin, and P. Sens, "A Heuristically Assisted Deep Reinforcement Learning Approach for Network Slice Placement," *IEEE Transactions on Network and Service Management*, vol. 4537, no. c, pp. 1–12, 2021.

[38] R. Amiri, H. Mehrpouyan, L. Fridman, R. K. Mallik, A. Nallanathan, and D. Matolak, "A Machine Learning Approach for Power Allocation in HetNets Considering QoS," *IEEE International Conference on Communications*, vol. 2018-May, 2018.

[39] S. R. Park and J. W. Lee, "A fully convolutional neural network for speech enhancement," *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, vol. 2017-Augus, no. 2, pp. 1993–1997, 2017.

[40] S. T. Arzo, D. Scotece, R. Bassoli, D. Barattini, F. Granelli, L. Foschini, and F. H. P. Fitzek, "MSN: A Playground Framework for Design and Evaluation of MicroServices-Based sdN Controller," *Journal of Network and Systems Management*, vol. 30, no. 1, pp. 1–31, 2022.

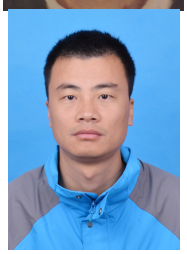
[41] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, F. Huici, and I. Nsdi, "ClickOS and the Art of Network Function Virtualization This paper is included in the Proceedings of the," 2014.

[42] G. Kibalya, J. Serrat, J. L. Gorricho, H. Yao, and P. Zhang, "A novel dynamic programming inspired algorithm for embedding of virtual networks in future networks," *Computer Networks*, vol. 179, no. May, p. 107349, 2020. [Online]. Available: https://doi.org/10.1016/j.comnet.2020.107349

Godfrey Kibalya received a BSc. degree in Telecommunications Engineering in 2010 from Makerere University Uganda and a MSc. degree in Telecommunications Engineering from the University of Trento, Italy. Currently, he is a Ph.D. candidate at the Technical University of Catalonia (UPC), Spain under the Department of Network Engineering. His research interests include Network function virtualization, and application of Artificial Intelligence in network management. He also an assistant lecturer at Kabale University, Uganda under the Department of Electrical Engineering.

Joan Serrat-Fernández received the degree of Telecommunication Engineer in 1977, and the Doctor degree in Telecommunication Engineering in 1983, both from Universitat Politècnica de Catalunya -UPC-. Currently he is Full Professor at UPC where he has been involved in several collaborative projects with different European research groups, both through bilateral agreements or through participation in European funded projects. His topics of interest are in the field of autonomic networking and service and network management.

Juan-Luis Gorricho received a Telecommunication Engineering degree in 1993, and a Ph.D. degree in 1998, both of them from the Technical University of Catalonia (UPC). Since 1994 he joined the Department of Network Engineering at the UPC, as associate professor since 2001. His most recent research interests have been focused on the management of resources for virtualized networks and functions, cloud computing and software defined networks.



Peiying Zhang PEIYING ZHANG is currently an Associate Professor with the College of Computer Science and Technology, China University of Petroleum (East China). He received his Ph.D. in the School of Information and Communication Engineering at University of Beijing University of Posts and Telecommunications in 2019. He has published multiple IEEE/ACM Trans./Journal/Magazine papers since 2016, such as IEEE TVT, IEEE TNSE, IEEE TNSM, IEEE TETC, IEEE Network, IEEE Access, IEEE IoT-J, ACM TALLIP, COMPUT COMMUN, IEEE COMMUN MAG, and etc. He served as the Technical Program Committee of ISCIT 2016, ISCIT 2017, ISCIT 2018, ISCIT 2019, Globecom 2019, COMNETSAT 2020, SoftIoT 2021, IWCMC- Satellite 2019, and IWCMC-Satellite 2020. His research interests include semantic computing, future internet architecture, network virtualization, and artificial intelligence for networking.