Computer Networks 219 (2022) 109451

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Edge intelligence for service function chain deployment in NFV-enabled networks

Mohammad Ali Khoshkholghi *, Toktam Mahmoodi

Centre for Telecommunications Research, King's College London, United Kingdom

ARTICLE INFO

Keywords:
Edge intelligence
Network function virtualization
Service chain deployment
Markov decision process
Distributed deep reinforcement learning

ABSTRACT

With evolution of network function virtualization (NFV), network services can be provided as service function chains (SCs), each consisting of multiple virtual network functions (VNFs). The deployment of SCs including placement of VNF instances and virtual links connecting these functions, onto the substrate physical network is a critical issue which significantly affects the performance of the offered network services. Due to the unpredictable traffic and network state variations, as well as diverse quality of service (QoS) requirements, an online SCs deployment approach is needed to cope with different service requests and real-time network traffics. In this paper, we employ edge intelligence using a distributed deep reinforcement learning approach to deploy SCs in order to jointly balance the load on the physical nodes and links in the edge environments. The evaluation results show that the proposed approach outperforms state-of-the-art algorithms in terms of minimizing the drop rate of the incoming service chain requests. In addition, the proposed approach is able to rapidly deploy service flows even in the large real-world network typologies.

1. Introduction

NFV is an innovational network architecture which can improve agility and flexibility of networks by decoupling network functions such as routing, firewalls and intrusion detection from physical boxes so that they can run as software-based applications. NFV is able to dynamically scale the network function instances, send the VNFs across a distributed network and upgrade the software without interrupting services. VNFs can be hosted on cloud/edge physical machines in the form of Virtual Machine (VM) or other containers such as Linux container and Docker. Each SC consists of multiple VNFs which will be processed in a predefined order to provide a specific network service. SC deployment refers to the placement of VNF instances and virtual links connecting them, into physical network nodes and links. Service flows, then, traverse through the relevant SCs (requested by users) mapped on the substrate network. Since the network traffic varies over time in the real-world networks, the SC deployment should be dynamically adjusted in response to the network load changes using online placement approaches [1].

Load balancing refers to efficiently steering network traffic across servers and links aiming to provide fault tolerance and delay guarantee as two vital needs for the recent critical-mission applications. Based on [2], about 20% packet loss occurs in the case of VNF placement with 100% CPU utilization since the high server and link utilization leads to server failure and link bottleneck. Balancing the load of servers and links can reduce the queuing delay as well as the risk of congestion in the network and consequently decreases the service drop rate allowing the service providers to admit more user requests. Hence, considering an efficient load balancing strategy is highly required to deploy an effective SCs placement for the NFV Infrastructures (NFVI) to deal successfully with real-time network traffic in a short time especially for the mission-critical applications such as 5G and IoT applications (AR/VR, online gaming and smart cities) as well as time-sensitive traffic flows (such as in VoIP and Video Conferencing), where delay guarantee and fault-tolerance are the critical requirements.

Due to the importance of SCs deployment problem, researchers have addressed a variety of topics in the recent years, such as minimizing energy consumption [3,4], monetary cost [5,6], end-to-end latency [7], maximizing resource utilization [8], profit [9] as well as a trade-off between multiple objectives such as joint latency and cost [10] and so on. However, despite the importance of the subject, less effort has been made about jointly balancing the network and server load. In addition, the existing load balancing approaches in the literature [11,12], have focused on the offline model-based heuristics or using optimization solvers that are limited to particular system models assumed by authors and they are not applicable for different scenarios or other real-time networks. Moreover, these models need prior information about the network traffic which is not available in practice. However, in this

* Corresponding author.
E-mail addresses: ali.khoshkholghi@kcl.ac.uk (M.A. Khoshkholghi), toktam.mahmoodi@kcl.ac.uk (T. Mahmoodi).

https://doi.org/10.1016/j.comnet.2022.109451
Received 22 April 2022; Received in revised form 11 September 2022; Accepted 28 October 2022
Available online 4 November 2022

research, we aim to fill these research gaps proposing an online and autonomous data-driven approach which is able to deal with different scenarios and unforeseen network traffics in a short time.

Edge computing is a paradigm to shift the services from cloud to the network edge. In edge computing, service provisioning is a major challenge because of the distributed nature of these environments. Compute nodes are geographically distributed as well as resources are constrained and network delay is limited. Moreover, the user requests arrive across the network with variation over time. Therefore, a centralized approach where one component (e.g. a remote cloud) is in charge of deploying SCs on whole network using up-to-date global knowledge of resource utilization and service demands is unrealistic in real large-environments. Indeed, the SCs deployment approaches must consider scalability issue to be properly applied to the edge environments enabling rapid deployment of service flows. To overcome this challenge, edge intelligence can be considered as a great solution due to its advantages of decreasing network traffic between edge devices and cloud, improving service latency and response time as well as scalability. Edge intelligence as a combination of edge computing and artificial intelligence (AI) is an emerging paradigm running machine learning (ML) algorithms at the network edge where data is generated. In this paper, to employ the edge intelligence for the sake of efficient SCs deployment in the edge environments, we propose a distributed deep reinforcement learning (DRL) based approach which is able to deploy the incoming flows in a distributed fashion instead of a centralized approach. The further details on the proposed approach is provided in Section 4.

The key contributions of this paper include:

- Focusing on service chaining, we formulate a Mixed-Integer Linear Programming problem with the objectives of load balancing as well as reducing drop rate.
- An online solution using distributed DRL, and relying on the actor–critic method, is devised to solve the above mentioned problem.
- We develop an evaluation platform using real-world network topologies and traffic patterns [13], and benchmark our distributed solution with centralized solution as well as two baseline algorithms [14,15]; and
- Finally, we demonstrate the benefits of edge intelligence using distributed DRL on the SCs placement problem and we show that our solution is able to respond very fast to arriving service flows even for larger network scales while reducing the service drop rate.

The rest of this paper is organized as follows. The related works are discussed in Section 2. In Section 3, we formulate the problem of SCs deployment in order to jointly balance the servers and links load. In Section 4, we illustrate our proposed online and distributed deep reinforcement learning approach to solve the formulated problem in details. Section 5 provides the performance evaluation of the proposed approach compared with the baseline algorithms. Finally, we conclude the paper in Section 6.

2. Related works

In this section, we investigate the SCs deployment problem and existing solutions in the literature. The related works can be categorized in terms of problem modeling, objectives and solution approaches. Many existing studies have modeled the SCs deployment problem into Binary Integer Programming (BIP) [16–18], Integer Linear Programming (ILP) [8,19–21], Mixed Integer Linear Programming (MILP) [22–24] and solving the problem using optimization solvers (e.g CPLEX and Gurobi). Given the NP-hard nature of many of such formulated problems, various heuristic and meta-heuristic methods are also presented to solve the service chaining problem [25,26]. Although these approaches are efficient in the given use cases, however, applying them to different scenarios may significantly decrease their performance due to the specific assumptions considered in these works.

In terms of objectives, four metrics are mainly used in the literature, including energy consumption, latency, resource utilization, and cost. The first is network energy consumption as a result of deployed SCs. Authors in [4] have proposed a VNF chaining and placement strategy using game theory to minimize energy consumption of NFVIs. Another work [3] presented an optimal VNF placement with minimum energy consumption using CPLEX, however, it works well only for small scale networks. A data-intensive service deployment scheme using genetic algorithm has been proposed in [27] for edge environments. In [7], an ILP model has been formulated to minimize end-to-end latency, and to solve the problem, they leveraged a stable matching-based algorithm for NFV-enabled edge environments. In [28], authors have proposed heuristics to maximize the resource utilization. Also in [8], a polynomial-time heuristic has been developed to efficiently solve the ILP model formulated to maximize the resource utilization. The cost of communication, computing resources and instances required to process the service flows is defined as deployment cost. Some placement approaches aim to map the VNFs onto physical nodes and route the traffic through the paths in the network in order to reduce the deployment cost of service providers while satisfying the required quality of service (QoS). Authors in [5] proposed a genetic-based algorithm to reduce the monetary cost of SCs deployment on cloud/edge environment. Another study [6], addresses the problem of VNF placement to minimize the deployment cost by finding the minimum number of servers required to host all VNFs in the network.

In some of the state of the art on the SC deployment, combination of the above four metrics are considered as the objective function. In [10], the authors formulated the problem as a MILP to jointly minimize monetary deployment cost and end-to-end latency regarding several constraints for edge environments. They have developed a genetic-based algorithm and a bee-colony based algorithm to solve the problem and compared the results with optimal solution obtained through an optimization solver. In order to joint minimize the energy consumption and traffic cost, the authors in [29] proposed a two-step algorithm combining a matching approach and Markov approximation technique to solve the problem. The first step of the algorithm aims to find an appropriate subset of nodes to decrease the large solution space, and the second step is to deploy SCs on the selected nodes to minimize the network cost.

Finally, few papers considered load balancing as the objective function in the SCs deployment problem. In [2] and [30], authors have proposed an optimized load balancing approach for NFVIs aiming to balance the load on servers. Another study investigated the SCs mapping problem focusing on the network load balancing through minimizing the cost of all links in the network [31]. Another research [11], proposed a solution for balancing the load among VNFs by making a tree of SCs and then developing a flow-hash technique to distribute load among service chains. Finally, authors in [12], proposed an offline VNF placement algorithm which aims to jointly balance the network and server load. They leveraged the water filling algorithm, for this purpose. However, unlike the offline approach proposed in this paper assuming a full prior knowledge of network traffic, due to the unpredictable traffic and network state variations, as well as diverse QoS requirements, an online SCs deployment approach is needed to cope with different service requests and real-time network traffics.

As we discussed above, many studies have investigated the SCs deployment problem regarding various objectives, and also few works addressed the load balancing as their objective, however they ignore joint balancing the load on servers and network using an online strategy. Hence, there is a lack of online SCs deployment approach to balance the load on nodes and links simultaneously while satisfying delay constraints taking into account the real-time network variations. In our work, we aim to fill this research gap.

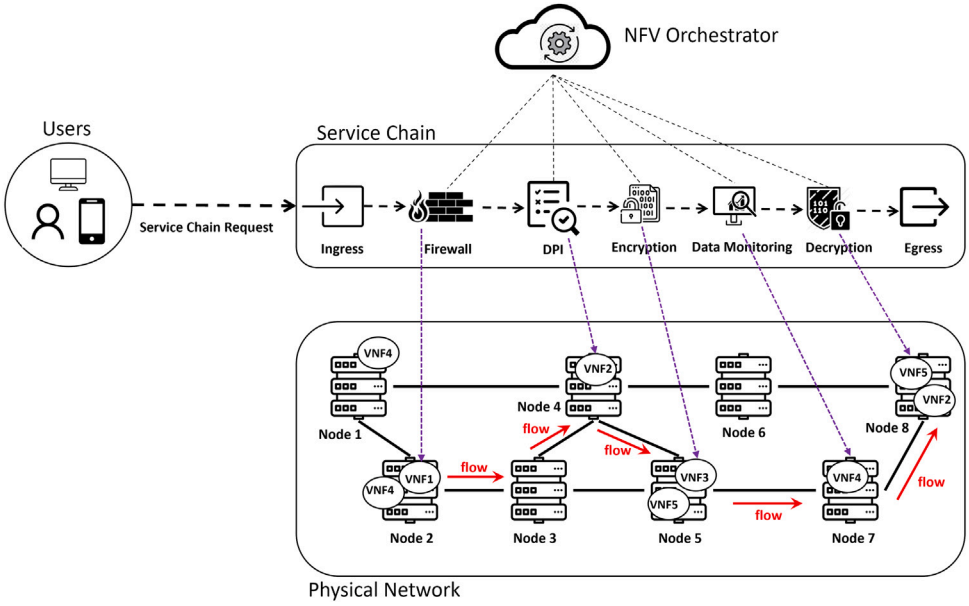

Fig. 1. An example of service chain deployment.

In addition, all the aforementioned studies, proposed mathematical programming approaches or model-based heuristics and meta-heuristics which are designed for certain network topologies and application scenarios. Scenario-customized solutions regardless of their efficiency, are difficult to adopt practically to other type of networks and applications. Recently, some researches have been conducted on the goal of SCs deployment using machining learning techniques such as reinforcement learning. These model-free techniques aim to develop data-driven solutions which can be flexibly applied to different network and applications scenarios. In [1], authors proposed a centralized deep reinforcement learning approach for SDN/NFV-enabled networks. They proposed an offline double deep Q network-based VNF placement algorithm to minimize the deployment cost of services. Authors in [32] proposed a centralized policy gradient based deep reinforcement learning approach to minimize the operational cost of the system. Another work [15] tries to maximize the accepted service requests using a centralized deep deterministic policy gradient approach. However, these centralized approaches may not be efficient for the edge environments where resources are constrained and geographically distributed, as well as the user requests arrive across the network with variation over time while network delay is limited. Different from these studies, in this paper we aim to employ edge intelligence for online SCs deployment to jointly balance the network and server load using a distributed deep reinforcement learning algorithm relying on an advantage actor–critic approach.

3. System model and problem formulation

In this section, first, we illustrate the problem of SCs deployment using an example and then, we formulate the problem as a MILP model. The objective of this optimization model is to balance the load of links and servers in order to decrease the number of rejected service flows demanded by users while satisfying the delay constraints. Table 1 shows the variables used in the paper.

In the example shown in Fig. 1, a network consists of 8 physical nodes connected by 10 physical links is presented. The physical nodes represent edge servers and the network orchestrator located at a remote cloud is in charge of SCs deployment. Some nodes are able to run specific VNFs: node 1 = [VNF_4], node 2 = [VNF_1, VNF_4], node 4 = [VNF_2], node 5 = [VNF_3, VNF_5], node 7 = [VNF_4] and node 8 = [VNF_2, VNF_5]. Users send their service requests to the network where a service function chain including 5 VNFs is provided. A service flow r_i corresponding to a given request enters the network through the ingress node and traverses the VNFs in a specific order as: VNF_1 (Firewall) → VNF_2 (Deep Packet Inspection) → VNF_3 (Encryption) → VNF_4 (Data monitoring) → VNF_4 (Decryption), and finally exits the network via egress node. Based on a placement scheme, the NFV orchestrator deploys the VNFs on the physical nodes as: node 2 hosts VNF_1, node 4 hosts VNF_2, node 5 hosts VNF_3, node 7 hosts VNF_4 and node 8 hosts VNF_5. Finally, flow r_i is passed through the nodes over the selected path.

The substrate network is represented by $G = (V, E)$, where V refers to the set of physical nodes denoted by $V = \{v_1, v_2, \dots, v_n\}$ and E is the set of K physical links between nodes indicated as $E = \{e_1, e_2, \dots, e_k\}$. Each $v \in V$ can host one or multiple VNFs. S is the set of all service chains provided by the system, each consisting of multiple VNFs with strict precedence. Each service chain $s \in S$ is defined as a directed graph denoted by $s = (F, L)$, where F is the set of VNFs and L refers to the set of virtual links between VNFs. Each $f \in F$ is characterized by its compute demands and processing delay, and each $l \in L$ is characterized by its bandwidth capacity. Each VNF instance can be hosted by a physical node that has the relevant functionality with respect to the resource constraints. Similarly, each virtual link can be mapped on a physical link that satisfies the bandwidth and delay constraints. Binary variable ($X_{sf}^{v} = 1$) indicates whether $f \in F$ from service chain s is placed on $v \in V$ or not (X_{sf}^{v}) = 0.

Given the above-mentioned parameters, we formulate an optimization model for VNF placement and virtual link embedding which aims to jointly balance the load of nodes and links while satisfying end-to-end latency requirements. Eq. (1) denotes the objective function defined in this work. We model the objective as minimizing the maximum servers' utilization and maximum links' bandwidth utilization over time, where server utilization is denoted by L_S, L_N is link bandwidth utilization and t indicates the time instants. $\alpha \in [0, 1]$ is a weight factor and can be adjusted based on the network states and service requirements. The utilization values are normalized in this equation (L_S, $L_N \in [0, 1]$).

$$\text{minimize} \quad \sum_{t \in T} \max_{v \in V, l \in L} (\alpha L_S(t) + (1 - \alpha) L_N(t)) \tag{1}$$

where:

$$L_S(t) = \sum_{s \in S} \sum_{f \in F} n_f^v(t) X_{s,f}^v, \forall t \in T, \forall v \in V \tag{2}$$

$$L_N(t) = \sum_{s\in S}\sum_{p\in P} q_s(t) H_s^p Z_p^e, \forall t \in T, \forall e \in E \quad (3)$$

In Eq. (2), $n_f^v(t)$ refers to the resource utilization of node v by VNF f at time t. In Eq. (3), $q_s(t)$ denotes the traffic load on the service chain s at time t, H_s^p is a binary variable denoting whether path p is used for service flow s ($H_s^p = 1$) or not ($H_s^p = 0$), and finally, binary variable $Z_p^e(t)$ indicates whether link e is placed on the path p (equals to 1) otherwise it equals to 0.

The constraints of the problem are defined as follows. Eq. (4), is the latency constraint and ensures that the end-to-end delay experienced by each service flow must not exceed the maximum tolerable latency (D_{max}^s) for the given service chain. In fact, this constraint guarantees that all service demands will be served within the expected time considered in SLA between service providers and users.

$$\sum_{p\in P}\sum_{e\in E} H_s^p Z_p^e d_e + \sum_{v\in V}\sum_{f\in F} X_f^v d_f \le D_{max}^s, \forall s \in S \quad (4)$$

where d_e is the propagation delay of each physical link determined by the distance between source and destination over link transmission rate, and d_f is the processing delay of VNF f placed on a given physical node v.

Eq. (5) represents the resource capacity constraint of physical nodes which guarantees that all processing demands of VNFs in set S mapped on nodes will not exceed the remained processing capacity for any nodes in order to avoid resource overutilization. Similarly, Eq. (6) ensures that the total bandwidth demands of virtual links in all service chains mapped on physical links will not exceed the bandwidth capacity for any physical links.

$$\sum_{v\in V}\sum_{f\in F} X_{s,f}^v \partial_{v,f} \le Cap^v, \forall s \in S \quad (5)$$

$$\sum_{e\in E}\sum_{l\in L} Y_{s,l}^e \partial_{e,l} \le Cap^e, \forall s \in S \quad (6)$$

where $\partial_{v,f}$ and $\partial_{e,l}$ denote the processing demand of VNF f from physical node v, and the bandwidth demand of virtual link l from physical link e, respectively.

Eq. (7), enforces that all VNFs of a given service chain s will be mapped on the available physical nodes. The constraint defined in Eq. (8) ensures that each VNF instance of a specific service chain will be instantiated on only one physical node. In addition, all the virtual links of each service chain, must be placed on the available physical links which is guaranteed by the Eq. (9).

$$\sum_{v\in V}\sum_{f\in F} X_{s,f}^v = |F|, \forall s \in S \quad (7)$$

$$\sum_{v\in V} X_{s,f}^v = 1, \forall f \in F, \forall s \in S \quad (8)$$

$$\sum_{e\in E}\sum_{l\in L} Y_{s,l}^e = |L|, \forall s \in S \quad (9)$$

where |F| and |L| refer to the number of VNFs and virtual links in a given service chain s. Eq. (10), guarantees that all VNFs will be placed and processed in a predefined order for a given service chain s mapped on a selected path p. It should be mentioned that if $X_{s,a}^u = 1$ and $X_{s,b}^v = 1$ and $O_s^{a,b} = 1$, then we conclude that $W_{s,a,b}^{u,v} = 1$, otherwise it equals to 0.

$$\sum_{u,v\in p} w_{s,a,b}^{u,v} = \sum_{a,b\in F} O_s^{a,b}, \forall s \in S, \forall p \in P \quad (10)$$

The constraints defined in Eqs. (11) and (12), enforce that each service flow uses only one path to carry its traffic, and all VNFs of a given service chain will be placed only on the selected path, respectively.

$$\sum_{p\in P} H_s^p = 1, \forall s \in S \quad (11)$$

$$\sum_{v\in V}\sum_{f\in F} H_s^p X_{s,f}^{p,v} = |F|, \forall p \in P, \forall s \in S \quad (12)$$

Table 1
Notations used in this paper.

Symbol	Description
G	Physical network
V	Set of physical nodes
S	Set of service chains
F	Set of VNFs in a service chain s
L	Set of virtual links
E	Set of physical links
P	Set of paths
α	Weight factor $\alpha \in [0,1]$
$L_S(t)$	Server utilization at time t
$L_N(t)$	Link utilization at time t
$n_f^v(t)$	Resource utilization of node v by VNF f at time t
$q_s(t)$	Traffic load on the service chain s at time t
d_e	Propagation delay of physical link e
d_f	Processing delay of VNF f
D_{max}^s	Maximum tolerable delay for a service chain s
$\partial_{v,f}$	Processing demand of VNF f from physical node v
$\partial_{e,l}$	Bandwidth demand of virtual link l from physical link e
Cap^v	Processing capacity of node v
Cap^e	Bandwidth capacity of link e
$X_{s,f}^v$	is 1, if VNF f from service chain s is mapped on physical node v, otherwise 0
H_s^p	is 1, if path p is used for service flow s, otherwise 0
Z_p^e	is 1, if link e is placed on the path p
$Y_{s,l}^e$	is 1, if virtual link l from service chain s is mapped on physical link e, otherwise 0
$O_s^{a,b}$	is 1, if VNF b is the successor of VNF a in the service chain s
$W_{s,a,b}^{u,v}$	is 1, if service chain s traverses from VNF a on node u to VNF b on node v, otherwise 0
$X_{s,f}^{p,v}$	is 1, if vnf f from service chain s is placed on node v from path p
τ	A given time slot
δ	The number of time steps in a given time slot
R_τ	The set of arriving flows at a given time slot
rC^v	The remaining capacity of node v
rW^e	The remaining bandwidth of link e
A_τ^r	The arrival time of flow r
ttl_r	Time to live of flow r
$\mathcal{S}$	Observation space
$\mathcal{A}$	Action space
$\mathcal{P}$	The probability of state transitions
$\mathcal{R}$	Reward Function
$\mathcal{Q}_t$	The feature set of flow r at time step t
$\mathcal{X}_t$	The availability of a given VNF instance on server v
r_c^d	Compute demand of flow r
r_w^d	Bandwidth demand of flow r
N_f	The number of unplaced VNFs
D_r	The remained tolerable delay
Re_t	Discounted cumulative reward
re_t	Reward obtained by action a at time step t
γ	Discount factor in the range [0, 1]
θ	Actor update parameter
θ_c	Critic update parameter
σ	Actor learning rate
β	Critic learning rate

4. Distributed deep reinforcement leaning approach

In this section, the proposed distributed deep reinforcement learning approach to optimize the deployment of service chains is discussed. Machine learning based solutions are able to outperform the state-of-the-art in diverse applications. In the problem of service chain deployment, ML techniques can be very useful by learning network dynamics using network statistics, traffic variations and past experience. Reinforcement learning based solutions, are alternatives for the traditional optimization techniques especially for the environments with high degree of variations. As a major problem, the conventional techniques use algorithms with a high complexity because they need to exchange information repeatedly to handle the dynamic changes and it may increase the overhead. However, reinforcement learning leverages a state–action mapping so that for each state as an input, RL agent creates an action. In addition, unlike the conventional techniques which

need certain models predefined by experts, RL agent trains the model via interactions with the environment. Instead of using a look up table as in RL, deep reinforcement learning use deep neural network (DNN) as an approximation function in order to deal with very large state space. Therefore, DRL is able to serve many incoming service demands even for the real-time environments with frequent network transitions.

In the following sub-sections, we specify observation space, action space and reward function using Markov Decision Process (MDP) for training DRL agents. Next, we explain the strategy used by agents to place the service flows. Then, we introduce the machine learning technique to create the agents, and finally illustrate the training and inference framework.

4.1. Markov Decision Process

MDP model can be well adapted to the SCs deployment problem to capture the dynamic network state transitions. The network state space is incorporated with the number of network links and servers, and the number of arriving service flows is associated with the frequency of state transitions. To handle the dynamic network load changes, we consider time slots τ including multiple time steps t, from t to $t+\delta$ ($\delta > 1$). In each τ, system controller monitors all the network links and servers, receives new service requests, eliminates timeout requests, makes the decision for each SCs and finally update the network states.

We define R as the set of all arriving service flows, $R_\tau \in R$ as the set of arriving flows at τ, rC^v as the remaining capacity of node $v \in V$ and rW^e as the remaining bandwidth of link $e \in E$. Arriving flows will be served based on their arrival times. The arrival time of a given flow $r \in R_\tau$ is defined as A^r_τ. Each flow requesting a service chain is dedicated a time to live (TTL) defined as ttl_r. At each given time, if $A^r_\tau \le t \le (A^r_\tau + ttl_r)$ then the flow is still alive, otherwise it is timed out and will be removed from the system. The MDP model is proposed as $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} >$, where $\mathcal{S}$ refers to the observation space, $\mathcal{A}$, is the action space, $\mathcal{P}$ denotes the probability of state transitions, and $\mathcal{R}$ is the reward function.

Observation Space. A vector is defined for each observation state $_t \in \mathcal{S}$ as $_t = (rC_t, rW_t, \mathcal{Q}_t, \mathcal{X}_t)$. The vector indicates that the observation space includes four different parts. First, rC_t denotes the remaining capacity of all nodes at time step t and is defined as $rC_t = (rC^1_t, rC^2_t, \dots, rC^{|V|}_t)$. Second, rW_t represents the remaining capacity of all links at time step t and is defined as $rW_t = (rW^1_t, rW^2_t, \dots, rC^{|E|}_t)$. Third, $\mathcal{Q}_t$ denotes the features of each service flow defined as $\mathcal{Q}_t = (r^d_c, r^d_{tw}, ttl_r, N_f, D_r)$ consisting of compute demand of the flow, bandwidth demand of the flow, remained TTL, the number of unplaced VNFs and the remained acceptable delay for the given flow, respectively. Finally, $\mathcal{X}_t$ is the availability of the requested VNF instance in a given server v.

Action Space. Each physical node in the network is assigned a unique index j in a range from 1 to the number of nodes in the network $|V|$, $j = 1, 2, \dots, |V|$. Action space, then, is determined as a set of node indices $\mathcal{A} = \{0, 1, 2, \dots, |V|\}$, where each integer denotes an action $a \in \mathcal{A}$. If a given VNF $f \in F$ cannot be allocated to any available servers, then $a = 0$. Otherwise, the $a = j$ denotes that the VNF f is successfully placed on the server $j \in V$.

Reward Function. The goal of the RL agent is to maximize the expected cumulative rewards that it receives in the long-term utility. As we described in Section 3, the objective of the optimization model is to jointly balance the network and server load upon which the objective function is formulated in Eq. (1). Based on the objective function, the reward function can be defined as follows.

$$R(s_t, a) = \begin{cases} \frac{1}{\alpha(L^t_S)+(1-\alpha)L^t_N} & \text{if flow } r_i \text{ is accepted,} \\ 0 & \text{if flow } r_i \text{ is rejected} \\ & \text{or incomplete,} \end{cases} \tag{13}$$

In Eq. (13), since the agent aims to increase the cumulative reward and the values of L^t_S and L^t_N are normalized to the range [0,1], we defined the reward as a fraction in case the flow r_i is accepted, otherwise the reward is 0. The less maximum load on nodes and links, the more reward will be gained for a given action.

4.2. Service chain placement strategy

In this sub-section, the strategy used for the placement of service chains is presented. As we can see in Fig. 2, at each time slot (or episode), a number of service flows can arrive in the system that should be placed on the network. At the beginning of each time slot, the controller eliminates timeout requests and releases their occupied resources. Given the time slot τ, two service requests $r_1, r_2 \in R$ arrive in the system. The request r_1 consists of 3 VNFs as $(f_{1,1}, f_{1,2}, f_{1,3})$ and r_2 includes 3 VNFs as $(f_{2,1}, f_{2,2}, f_{2,3})$. We assume r_1 has an earlier arriving time, hence, it should be placed first.

At each state transition, only one VNF will be placed. Therefore, starting from the state transition s_t, the first VNF f_1 is supposed to be mapped on a server. The agent collects the network statistics such as remaining servers' resources and links' bandwidth, and also extract the flow features, to make a list of candidate servers with enough resources to host f_1. Then, based on its policy, the agent takes an action from the feasible actions, places the VNF on the selected server and allocates required resources. Then, reward $r(a_i)$ will be calculated by the agent based on the Eq. (13) and the state moves to s_{t+1}. If there is no available server with sufficient resources and link bandwidth to host the VNF or delay constraint is not satisfied, the VNF and consequently the flow will be rejected as it happened to f_2 and r_1. In this case, the allocated resources are taken back and the second flow r_2 will be started to deploy in the same way. As it can be observed at s_{t+4}, r_2 is successfully deployed and the corresponding reward is calculated. This procedure will be continued for all the service flows arriving at time slot τ.

4.3. Advantage actor–critic reinforcement learning

An advantage actor–critic reinforcement learning algorithm is developed to train the DRL agents in our work. At each time step t, the DRL agent observes s_t, selects action a_t from the feasible actions and calculates the reward re_t. The agent aims to maximize the discounted cumulative reward that it receives in the long-run as follows.

$$J(\theta) = \mathbb{E}(\sum_{t=0}^{T-1} Re_t) \tag{14}$$

where:

$$Re_t = re_{t+1} + \gamma re_{t+2} + \gamma^2 re_{t+3} + \cdots = \sum_{k=0}^{T-1} \gamma^k re_{t+k+1} \tag{15}$$

In Eq. (15), $\gamma \in [0, 1]$ is a discount factor to trade off between the importance of immediate and future rewards. As the DRL agent, we develop an advantage actor–critic reinforcement learning algorithm, which combines a police-based and a value-based DRL algorithm. Each agent includes an actor network to generate a policy according to the observation states and the probability distribution of the agent, and a critic network to evaluate the policy based on a Temporal Difference [TD] error [33] as shown in Fig. 3. Since these networks perform different tasks, their architectures are different. The output layer of the actor network is a set of probability distributions corresponding to the feasible actions, and the output of critic network is only one value to evaluate the actor policy. In fact, the critic network estimates the advantage function whereas the actor network creates a policy represented by $\pi(s_t, a_t)$. Advantage function can be defined as follows.

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{16}$$

where $Q(s_t, a_t)$ refers to the Q-value of conducting action a_t at state s_t and $V(s_t)$ is the value function at state s_t. Advantage function determines the improvement obtained by action a_t in comparison with

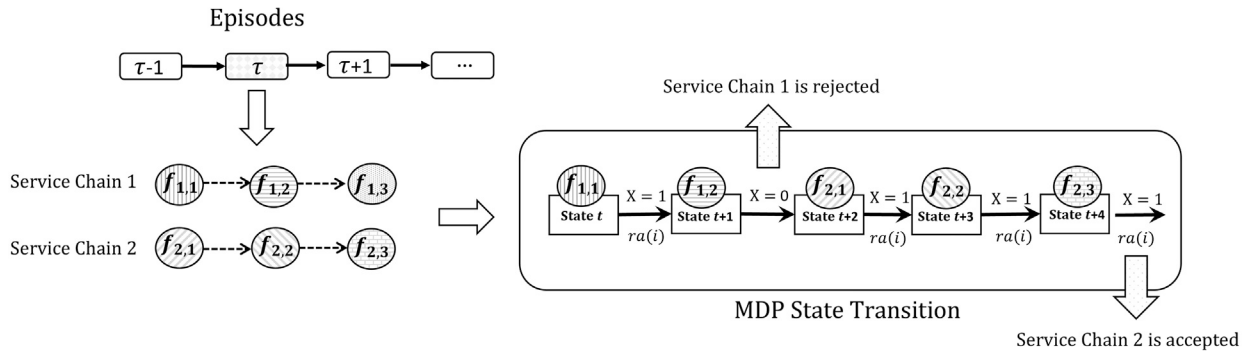

Fig. 2. MDP model for service chain deployment.

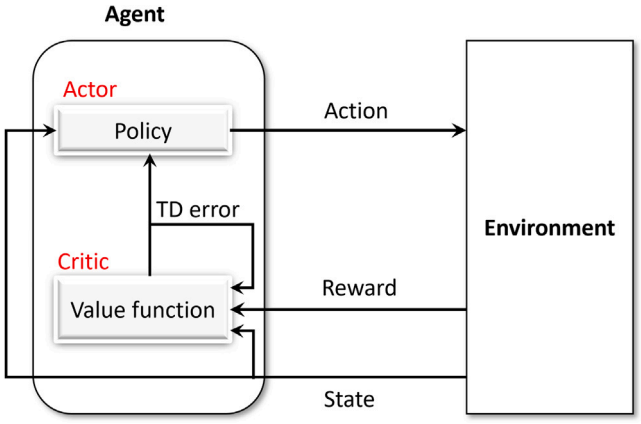

Fig. 3. Advantage actor–critic reinforcement learning framework.

an optimal action at state s_t. Advantage actor–critic algorithm uses the stochastic gradient decent method to update the parameters of neural networks denoted by θ for the actor network and θ_c for the critic network. The parameters are updated by the agent after reaching the terminal state. From the bellman equation [34] formulated in Eq. (17):

$$Q(s_t, a_t) = \mathbb{E}[re_{t+1} + \gamma V(s_{t+1})] \tag{17}$$

we can rewrite the advantage function for the critic network as follows.

$$A(s_t, a_t; \theta_c) = re_{t+1} + \gamma V_{\theta_c}(s_{t+1}) - V_{\theta_c}(s_t) \tag{18}$$

Using the policy gradient method, the gradient of accumulated reward is computed by Eq. (19). The advantage function $A(s_t, a_t; \theta_c)$ estimates the difference between expected reward using policy π compared with the expected reward using a deterministic policy.

$$\nabla_\theta j(\theta) = \mathbb{E}^{\pi}_{\theta}[\nabla_\theta \log \pi(s_t, a_t; \theta) \times A(s_t, a_t; \theta_c)] \tag{19}$$

To maximize the long-term discounted reward following a selected action, the policy parameter θ can be updated as follows.

$$\theta^{t+1} = \theta^t + \sigma \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(s_t, a_t) \times A(s_t, a_t; \theta) \tag{20}$$

where σ denotes the learning rate of the actor network. In addition, the critic parameter θ_c can be updated as:

$$\theta_c^{t+1} = \theta_c^t + \beta \sum_{t=0}^{T-1} \nabla_{\theta_c}(re_{t+1} + \gamma V_{\theta_c}(s_{t+1}) - V_{\theta_c}(s_t)) \tag{21}$$

where β is the learning rate of the critic network and γ is the discount factor defined to give weights to the rewards of different time steps. According to [35] and [36], in order to avoid premature convergence to a sub-optimal policy and to encourage the policy to exploration, an entropy regularization H can be added to Eq. (20) as follows.

$$\theta^{t+1} = \theta^t + \sigma \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(s_t, a_t) \times A(s_t, a_t; \theta)$$
$$+\eta \nabla_\theta H(\pi_\theta(s_t)) \tag{22}$$

where η is a weight factor to control the strength of entropy so that the larger value of η encourages more exploration.

4.4. Training and inference procedure

The service chain deployment approach proposed in this work, first, trains a DRL agent offline using a centralized algorithm to reduce computation overhead and avoid complexity of online training. In this, a controller (at cloud) monitors and manages whole network and trains the neural networks based on the observations of all nodes and for all service flows in the network to ensure creating an optimal policy. However, the centralized approach cannot be scaled online for distributed and large-size networks with fast arriving service requests, where hundreds of decisions should be made per just a few milliseconds. In addition, collecting data from whole network and sending them to the central controller to make a global knowledge is costly and even infeasible. To overcome this problem, a distributed approach (DDRL) could be deployed in order to fast and consistent service chains deployment. After offline training convergence, the trained model will be sent to the nodes (edge servers) across the network, so that each local node including an agent can decide for the arriving flows based on its own training model. For each agent, training will be continued online for the set of locally arriving flows. Fig. 4 shows the training and inference procedure.

Algorithm 1 shows the offline training process of the proposed DDRL approach. To alleviate the high variance of training caused by random parameters, we train k number of agents in parallel in a centralized fashion. Then for each agent, the actor neural network (π_θ) and critic neural network (V_{θ_c}) are initiated randomly (lines 2–4). Then, in each parallel environment and for different episodes, the agent observes current state s_t in terms of observation space described in Section 4-A. Based on the agent's policy $\pi_\theta(s_t, a)$, an action is selected and performed on the environment and the corresponding reward is calculated using Eq. (13). As we discussed in Section 4-B, then, the state moves to the next state s_{t+1}, where the agent observes the new state, as well as the next VNF is considered for placement. The state transition including current state, performed action, obtained reward and the new state are stored in the memory (lines 5–13). At the end of each episode, the policy parameter θ and critic parameter θ_c are updated using Eq. (21) and (22), respectively (lines 14–15). This procedure will be repeated for M number of episodes until the learning model reaches convergence. Finally, the best agent with the highest obtained reward among k parallel agents is selected as the trained model to be used for inference in run-time process.

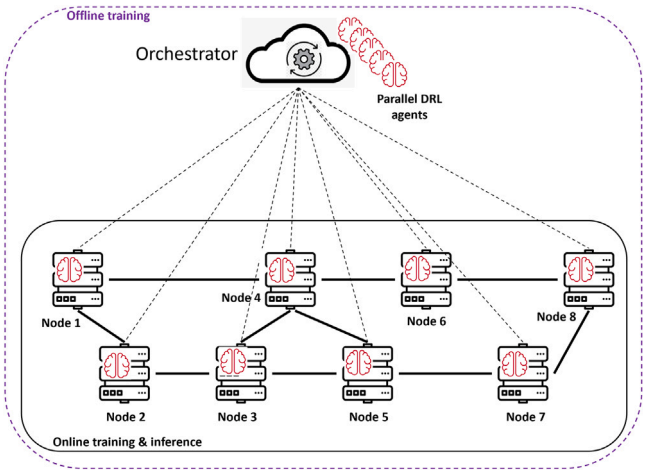

Fig. 4. Training and inference procedure.

Algorithm 1: Offline Training Process of DDRL

```
1  begin
2      k ← number of parallel agents;
3      for k agents in parallel do
4          Initialize the parameters of actor and critic neural
             networks θ and θ_c;
5          for Episode 1, ... , M do
6              Capture initial state s_t;
7              for time step t = 1, ... , T do
8                  Agent selects action a_t according to policy
                     π_θ(s_t, a);
9                  Perform action a_t to place VNF f_t;
10                 Calculate reward r(a_t) using Eq. (13);
11                 Observe new state s_{t+1};
12                 Transfer s_t to s_{t+1} and get the next VNF f_{t+1};
13                 Store transition (s_t, a_t, r(a_t), s_{t+1});
14             Update policy parameter θ using Eq. (22);
15             Update critic parameter θ_c using Eq. (21);
16     Select the agent with the best policy π_θ(s_t, a) achieving the
         highest reward;
```

Algorithm 2 illustrates the online inference and training process of DDRL algorithm. As we can see in line 2, the trained model provided offline by Algorithm 1, is deployed on the nodes across the network to create an online and distributed approach for the SCs deployment problem. Given a set of arriving service requests R_τ at each time slot, the request with the earliest arrival time is considered to be mapped on the physical network. Then, for a number of time steps, if a service flow arrives at node v, the node's agent starts the inference process using its own policy. First, the agent observes the current state s_t, and performs the action selected by the agent policy $\pi_\theta(s_t, a)$. Next, the agent calculates the reward and shifts s_t to s_{t+1} and store the transition states in memory. If the current state is the terminal state, then, the agent trains both actor and critic networks based on the state transitions using the defined equations. This procedure will be repeated for all the service requests in R_τ.

5. Performance evaluation

In this section, we explain the performance evaluation of the proposed DDRL approach. The efficiency of our approach is evaluated through simulation using real-traced network traffic patterns along with real-world network topologies. We have implemented the proposed approach in Python and developed DRL agents using Stable-Baselines3 [37]. All the experiments are conducted on a Dell machine with an Intel(R) core(TM) i7-10750H CPU @ 2.6 GHz, 6 cores and 12 threads, 16 GB of RAM and a Nvidia GeForce RTX 2070 GPU. We run each algorithm 10 times and the best value is shown in the measurement. In the fallowing sub-sections, we describe the evaluation setting and then, present the numerical results obtained by DDRL algorithm compared with the baselines in terms of several performance metrics and for different scenarios.

Algorithm 2: Online Inference and Training Process of DDRL

```
1  begin
2      Deploy the best trained model to every node v ∈ V;
3      Select a request r from a set of arriving request R_τ based on
         their arrival time;
4      for time step t = 1, ... , T do
5          if Service flow r arrives at node v then
6              Capture initial state s_t;
7              Perform action a to place VNF f_t according to policy
                 π_θ(s_t, a);
8              Calculate reward r(a_t) using Eq. (13);
9              Observe new state s_{t+1};
10             Transfer s_t to s_{t+1} and get the next VNF f_{t+1};
11             Store transition (s_t, a_t, r(a_t), s_{t+1});
12             if terminal state then
13                 Update policy parameter θ using Eq. (22) at node
                     v;
14                 Update critic parameter θ_c using Eq. (21) at node
                     v;
15         Continue for the next service request;
```

5.1. Experimental setup

5.1.1. Network topologies and simulation settings

In this work, we have used the real-world network topologies extracted from SNDlib [13]. SNDlib's network instances are widely used in the previous studies as substrate networks to simulate the SCs placement approaches (e.g. see [38]). We have selected four network scales with 12, 22, 50 and 161 physical nodes to evaluate the scalability of the DDRL approach. Since the topologies determine the location of nodes and their interconnections, the link delay can be calculated according to the distance between nodes. We set randomly and uniformly a processing unit (CPU cores) to each physical node in the range [0, 2] and the capacity of bandwidth of each physical link in the range [1, 10] Gbps. We defined a set of VNFs, each with a given processing delay (5 or 10 ms). Each service chain contains 1 to 3 VNFs with data rate 1 Gbps, TTL 100 ms and various end-to-end delay deadlines (30, 40, 50, 60 and 100 ms).

5.1.2. Training process and hyperparameters

In the simulation, we considered 100,000 time steps during training process and 20,000 time steps in the run-time process in which service flows arrive in a real-traced load pattern. After each 50 steps, the network states are observed and actions are performed. We set the number of parallel agents to be trained offline as $k = 5$ and the best model is selected for the online phase. Adam optimizer [39] is used to train the actor and critic neural networks each with a single hidden layer with 64 nodes (ReLU activation). The learning rates for the critic and actor networks are fixed to 0.001 and 0.0001, respectively; and the discount factor is set to 0.99.

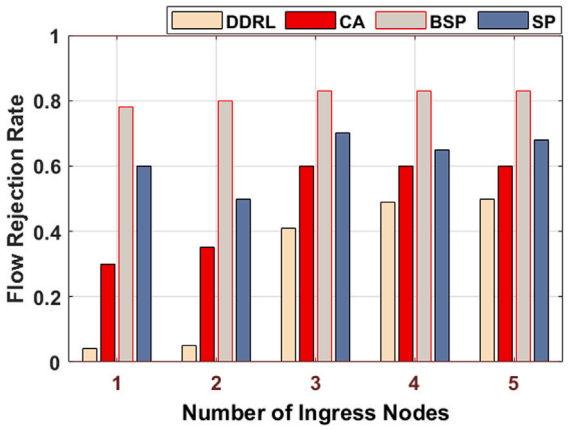

Fig. 5. Flow rejection rate for different number of ingress nodes.

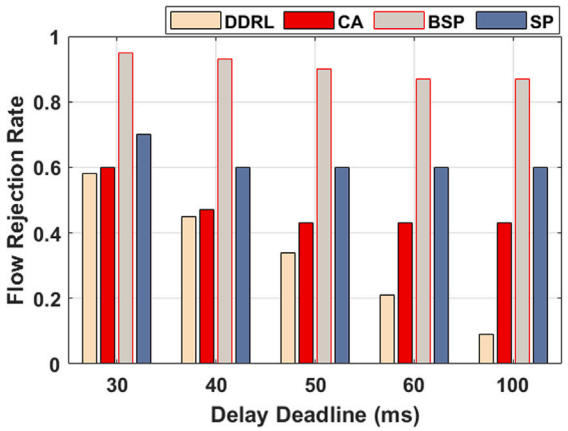

Fig. 6. Flow rejection rate with varying delay deadlines.

5.1.3. Baseline algorithms

In this sub-section, in order to evaluate the efficiency of the proposed DDRL algorithm, we compare it against the following baselines.

- A state-of-the-art SCs placement approach (CA) [15]: A central controller collects all network statistics and service flows' features and solve the placement problem using a deep deterministic policy gradient (DDPG) approach.
- A non-recursive greedy approach (SP), which deploys SCs using a shortest path algorithm. SP tries to place the first VNF instance in the ingress node and the rest of VNFs at the closest neighbors from the previous one.
- BSP [14]: A bidirectional heuristic algorithm to jointly place and scale VNFs leveraging a destroy-and-repair strategy.

5.2. Numerical results

In this section, we present and analyze the experimental results obtained by the proposed DDRL algorithm against three baseline algorithms.

5.2.1. Evaluation with different number of ingress nodes

Fig. 5 shows the evaluation results in terms of flow rejection rate against number of ingress nodes using Abilene topology with 11 nodes and 15 links and its real-traced traffic patterns extracted from SDNlib. The number of ingress nodes is increased from 1 to 5. By increasing the number of ingress nodes, the network traffic is increased while the compute resources and bandwidth capacity are fixed. Hence, with more ingress nodes we expect more flow drops caused by resource saturation. As it can be observed, DDRL outperforms all three baseline algorithms for every number of ingress nodes by decreasing the flow rejection rate. Given one and two ingress nodes, DDRL significantly reduces the number of rejected flows compared with other algorithms, so that the rejection percentage is not more than 5% while the rejection percentage is up to 35%, 80% and 60% for CA, BSP and SP, respectively. Since the network traffic is increased sharply from 3 ingress nodes, we can see an increase in the number of dropped flows although still DDRL obtains the lowest rate among others and this trend continues until 5 ingress nodes. CA algorithm works better than BSP and SP, and it can successfully deploy SCs with almost the same rate for 3 to 5 ingress nodes, although in all cases DDRL outperforms it by 19% (3 ingress nodes), 11% (4 Ingress nodes) and 10% (5 Ingress nodes). For 3 to 5 ingress nodes, DDRL also obtains better results compared with BSP and SP, up to 42% and 29%, respectively. These results show that DDRL is quite successful in placing SCs on the network so that the rejection rate is minimized even in the scenarios which the network resources are highly saturated.

5.2.2. Evaluation with different delay deadline setting

Fig. 6 shows the results obtained by the algorithms in terms of the flow rejection rate in 5 scenarios with varying delay deadlines and considering 4 ingress nodes. Delay deadline refers to a maximum tolerable end-to-end delay in which a flow should be processed successfully. We considered the deadlines as 30, 40, 50, 60 and 100 ms, by which we can explore the performance of DDRL for the flows related to various applications with different time-sensitivity and latency requirements. Given a tight delay deadline as 30 or 40 ms, the percentage of dropped flows is quite high, since the flows are dropped when they exceed the certain deadline. We can see the rejection rate of DDRL is slightly better than CA for 30 and 40 ms deadlines, however it decreases the rejection rate up to 22% and 37% compared with SP and BSP, respectively.

By increasing the delay deadline, since DDRL has more time to exploit the network to find an optimal placement for incoming flows, it is able to distribute and balance the load on more nodes and links farther away, and consequently it obtains a better rate of successful deployment of the flows. Hence, the higher delay deadline the less rejection rate can be observed by DDRL. The performance of BSP is again the worst among all algorithms and it drops many flows although for the higher deadlines it obtains less rejection rate. The rejection rate is dropped about 10 percents by SP from 30 to 40 ms and it shows the same rate afterwards, because SP simply selects the shortest path for the flows and the end-to-end delay required by SP to place the flows is less than 40 ms. The CA algorithm works better than SP and SA and the rejection rate is decreased from 30 to 50 ms but becomes constant afterwards. DDRL significantly outperforms all the baselines specially for the higher deadlines so that in the last scenario with 100 ms deadline, it is able to reduce the rejection rate by 34%, 88% and 51% compared with CA, BSP and SP, respectively.

Fig. 7 shows the average end-to-end delay obtained by 4 algorithms for different delay deadlines. The obtained results are in line with the results discussed for Fig. 6 and illustrates the reasons behind them. By increasing the deadline from 30 to 100 ms, DDRL constantly exploits higher end-to-end delays within the deadline to deploy more flows by which the number of rejected flows is decreased. CA although uses higher delay to deploy more flows from 30 to 50 ms, but does not exploit beyond 50 ms. Similarly, SP and BSP stop exploiting higher delays at some points as well (SP: 40 ms and BSP: 50 ms).

5.2.3. Evaluation with different network topology

To evaluate the scalability of the proposed DDRL algorithm in comparison to the baselines, we have used 3 network topologies as Geant, Germany50 and Brain including 22 nodes, 50 nodes and 161 nodes respectively, along with Abilene network. The results shown in Fig. 8 indicate the efficiency of DDRL for different network topologies

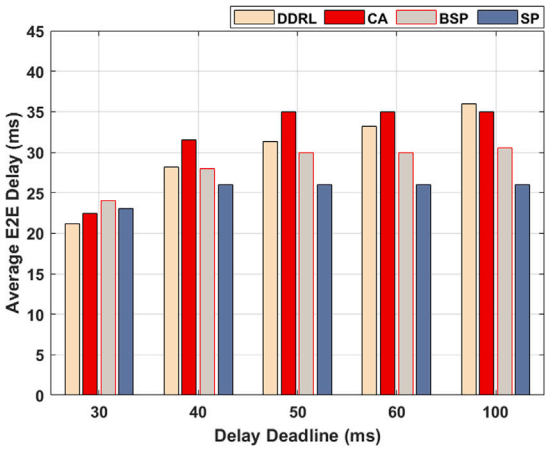

Fig. 7. Average end-to-end delay with varying delay deadlines.

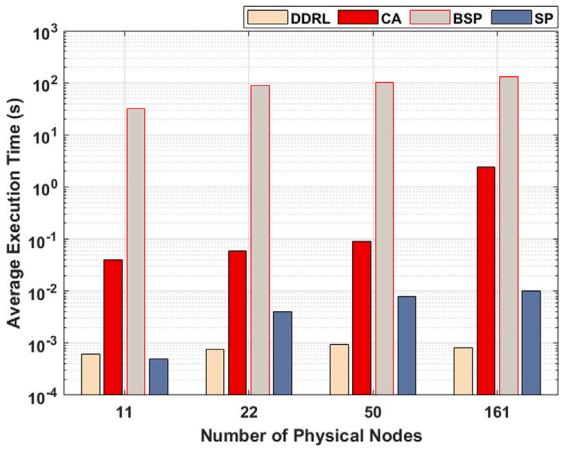

Fig. 9. Average inference execution time for different network topologies with increasing number of nodes.

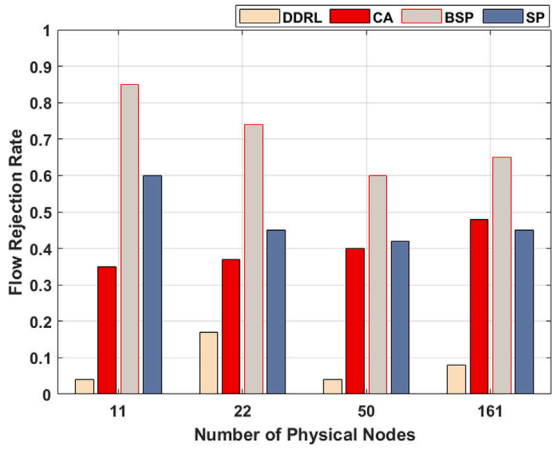

Fig. 8. Flow rejection rate for different network topologies with increasing number of nodes.

with increasing number of physical nodes. As it can be observed, due to the distributed technique used by DDRL algorithm upon which agents are spread across the network, the performance of the algorithm is independent of the network size. The rejection rate obtained by DDRL for the networks including 11 and 50 nodes is 4%, for 161-node network is 8%, while it is slightly higher for 22-node network. However, CA increases the number of dropped flows by increasing the network size since it uses a central approach where an agent is responsible for the whole network. For the larger network sizes, CA even could be worse than SP since it will increase the end-to-end delay of the flows and exceed the deadline which could consequently lead to flow rejections. SP also obtains the results regardless of the network size since it only focuses on the shortest paths in the network. DDRL outperforms other algorithms significantly in all 4 scenarios e.g. for 50-node network by 36% (SP), 56% (CA), and 61% (BSP).

Fig. 9 shows the evaluation results in algorithmic scale obtained on the average inference execution time for 4 network topologies with increasing number of nodes. As we mentioned before, an online service chain deployment algorithm should be very fast to make decisions even for large-scale networks with dynamic changes, to be applied to the real-world run-time networks. It can be observed that the average inference time for each placement decision obtained by DDRL is less than 1 millisecond even for larger networks since each agent is in charge of its own local decisions. However, the CA algorithm which uses a central agent, increases the inference time by increasing the number of nodes so that for larger networks, e.g. 161 nodes, it may takes few seconds to make decisions which consequently causes many flow rejections. The inference time results are in line with the results discussed for Fig. 8 where by increasing the network size, CA needs more inference time which leads to more flow rejection rate, however, DDRL can handle the flows very fast even for larger networks and therefore, it can successfully deploy a very high percentage of incoming flows in different scenarios. In addition, the SP algorithm is quite fast and make a decision within few milliseconds. BSP obtained the worst results and its average inference time is even more than 100 s for the larger networks. Therefore, DDRL outperforms all other algorithms in terms of the inference execution time. As a conclusion, based on the results shown in Figs. 5 to 9, the proposed online DDRL approach is quite successful to deploy SCs very fast and efficiently onto the substrate networks even for the large-scale ones with the realistic traffic patterns and dynamic changes.

5.2.4. *Evaluation of load balancing*

This subsection evaluates the efficiency of the proposed algorithm compared with the baseline algorithms in terms of load balancing as formulated in Eq. (1) in Section 3. As shown in Fig. 10, DDRL outperforms three other algorithms for every number of ingress nodes by minimizing the maximum resource utilization especially for 3 ingress nodes and afterwards where the resource demand is increased sharply while the resource capacity is fixed. The BSP algorithm obtains the worse results among all algorithms in this measurement as it tries to place VNFs into the nodes as much as they have remained capacity. Moreover, as it can be observed in Fig. 11, in terms of minimizing the maximum bandwidth utilization, DDRL obtains the lowest percentage among others whereas the SP algorithm presents a poor performance in this metric especially by increasing the number of ingress nodes and consequently network traffic since it uses a static and non-recursive shortest path approach which may lead to link overutilization. These results show that DDRL is quite successful in placing SCs on the network so that the load is balanced on joint links and nodes of the network.

6. Conclusions

In this paper, we studies the online service chain deployment problem considering dynamic network traffic in NFV-enabled networks. First, we formulate the problem as a MILP model where the objective function is jointly balancing the load on the physical nodes and

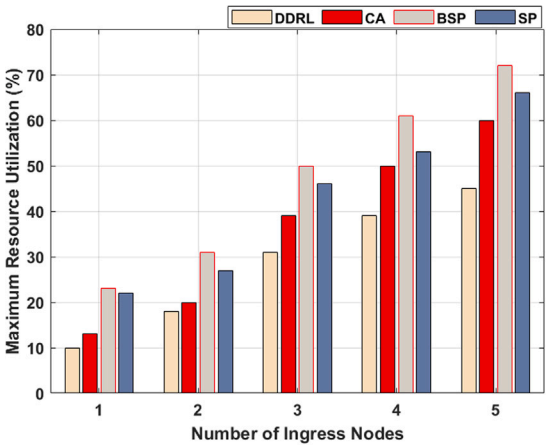

Fig. 10. Maximum resource utilization for different number of ingress nodes.

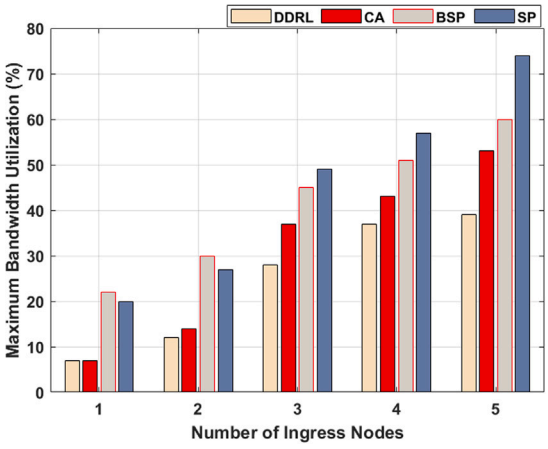

Fig. 11. Maximum bandwidth utilization for different number of ingress nodes.

links of the network. In order to solve the service chain problem, we propose a distributed and online approach using deep reinforcement learning. The goal is to devise a service chain placement policy to minimize the number of dropped service flows while balancing the load on nodes and links satisfying the end-to-end latency constraints. An on-policy advantage actor–critic reinforcement learning technique has been used to develop the DRL agents. Simulation results show that DDRL algorithm can respond very fast just in less than one millisecond to arriving service flows even for larger network scales. The results also demonstrate the performance improvement of the proposed DDRL approach in terms of the number of dropped service flows compared with the state-of-the-art baseline algorithms. In the future work, we are planing to extend our work by leveraging a federated learning technique to continue online learning of agents collaboratively using a reward aggregator.

CRediT authorship contribution statement

Mohammad Ali Khoshkholghi: Conceptualization, Methodology, Software, Data analysis and interpretation, Writing. **Toktam Mahmoodi:** Conceptualization, Critical revision, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] J. Pei, P. Hong, M. Pan, J. Liu, J. Zhou, Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks, IEEE J. Sel. Areas Commun. 38 (2) (2019) 263–278.
[2] T. Wang, H. Xu, F. Liu, Multi-resource load balancing for virtual network functions, in: 2017 IEEE 37th International Conference on Distributed Computing Systems, ICDCS, IEEE, 2017, pp. 1322–1332.
[3] M.A. Raayatpanah, T. Weise, Virtual network function placement for service function chaining with minimum energy consumption, in: 2018 IEEE International Conference on Computer and Communication Engineering Technology, CCET, IEEE, 2018, pp. 198–202.
[4] R. Bruschi, A. Carrega, F. Davoli, A game for energy-aware allocation of virtualized network functions, J. Electr. Comput. Eng. 2016 (2016).
[5] M.A. Khoshkholghi, J. Taheri, D. Bhamare, A. Kassler, Optimized service chain placement using genetic algorithm, in: 2019 IEEE Conference on Network Softwarization (NetSoft), IEEE, 2019, pp. 472–479.
[6] M. Xia, M. Shirazipour, Y. Zhang, H. Green, A. Takacs, Network function placement for NFV chaining in packet/optical datacenters, J. Lightwave Technol. 33 (8) (2015) 1565–1570.
[7] K.S. Ghai, S. Choudhury, A. Yassine, A stable matching based algorithm to minimize the end-to-end latency of edge nfv, Procedia Comput. Sci. 151 (2019) 377–384.
[8] D. Li, P. Hong, K. Xue, J. Pei, Virtual network function placement and resource optimization in NFV and edge computing enabled networks, Comput. Netw. 152 (2019) 12–24.
[9] F. Paganelli, P. Cappanera, A. Brogi, R. Falco, Profit-aware placement of multi-flavoured VNF chains, in: 2021 IEEE 10th International Conference on Cloud Networking (CloudNet), IEEE, 2021, pp. 48–55.
[10] M.A. Khoshkholghi, M.G. Khan, K.A. Noghani, J. Taheri, D. Bhamare, A. Kassler, Z. Xiang, S. Deng, X. Yang, Service function chain placement for joint cost and latency optimization, Mob. Netw. Appl. 25 (6) (2020) 2191–2205.
[11] P.-C. Lin, Y.-D. Lin, C.-Y. Wu, Y.-C. Lai, Y.-C. Kao, Balanced service chaining in software-defined networks with network function virtualization, Computer 49 (11) (2016) 68–76.
[12] A. Zamani, B. Bakhshi, S. Sharifian, An efficient load balancing approach for service function chain mapping, Comput. Electr. Eng. 90 (2021) 106890.
[13] SNDlib URL http://sndlib.zib.de.
[14] S. Dräxler, S. Schneider, H. Karl, Scaling and placing bidirectional services with stateful virtual and physical network functions, in: 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), IEEE, 2018, pp. 123–131.
[15] S. Schneider, R. Khalili, A. Manzoor, H. Qarawlus, R. Schellenberg, H. Karl, A. Hecker, Self-learning multi-objective service coordination using deep reinforcement learning, IEEE Trans. Netw. Serv. Manag. (2021).
[16] J. Pei, P. Hong, K. Xue, D. Li, Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system, IEEE Trans. Parallel Distrib. Syst. 30 (10) (2018) 2179–2192.
[17] J. Liu, Y. Li, Y. Zhang, L. Su, D. Jin, Improve service chaining performance with optimized middlebox placement, IEEE Trans. Serv. Comput. 10 (4) (2015) 560–573.
[18] Y. Liu, J. Pei, P. Hong, D. Li, Cost-efficient virtual network function placement and traffic steering, in: ICC 2019-2019 IEEE International Conference on Communications, ICC, IEEE, 2019, pp. 1–6.
[19] F. Bari, S.R. Chowdhury, R. Ahmed, R. Boutaba, O.C.M.B. Duarte, Orchestrating virtualized network functions, IEEE Trans. Netw. Serv. Manag. 13 (4) (2016) 725–739.
[20] D. Li, P. Hong, K. Xue, et al., Virtual network function placement considering resource optimization and SFC requests in cloud datacenter, IEEE Trans. Parallel Distrib. Syst. 29 (7) (2018) 1664–1677.
[21] D. Qi, S. Shen, G. Wang, Towards an efficient VNF placement in network function virtualization, Comput. Commun. 138 (2019) 81–89.
[22] H. Tang, D. Zhou, D. Chen, Dynamic network function instance scaling based on traffic forecasting and VNF placement in operator data centers, IEEE Trans. Parallel Distrib. Syst. 30 (3) (2018) 530–543.
[23] H. Hawilo, M. Jammal, A. Shami, Network function virtualization-aware orchestrator for service function chaining placement in the cloud, IEEE J. Sel. Areas Commun. 37 (3) (2019) 643–655.
[24] M. Zeng, W. Fang, Z. Zhu, Orchestrating tree-type VNF forwarding graphs in inter-DC elastic optical networks, J. Lightwave Technol. 34 (14) (2016) 3330–3341.
[25] M.A. Khoshkholghi, Y. Sharma, M.G. Khan, A. Al-dulaimy, J. Taheri, Resource allocation models in/for edge computing, 2020.

[26] M.G. Khan, J. Taheri, M.A. Khoshkholghi, A. Kassler, C. Cartwright, M. Darula, S. Deng, A performance modelling approach for sla-aware resource recommendation in cloud native network functions, in: 2020 6th IEEE Conference on Network Softwarization, NetSoft, IEEE, 2020, pp. 292–300.
[27] S. Deng, Z. Xiang, J. Taheri, M.A. Khoshkholghi, J. Yin, A.Y. Zomaya, S. Dustdar, Optimal application deployment in resource constrained distributed edges, IEEE Trans. Mob. Comput. 20 (5) (2020) 1907–1923.
[28] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. Ramakrishnan, T. Wood, Virtual function placement and traffic steering in flexible and dynamic software defined networks, in: The 21st IEEE International Workshop on Local and Metropolitan Area Networks, IEEE, 2015, pp. 1–6.
[29] C. Pham, N.H. Tran, S. Ren, W. Saad, C.S. Hong, Traffic-aware and energy-efficient vnf placement for service chaining: joint sampling and matching approach, IEEE Trans. Serv. Comput. 13 (1) (2017) 172–185.
[30] X. Fei, F. Liu, H. Xu, H. Jin, Towards load-balanced VNF assignment in geo-distributed nfv infrastructure, in: 2017 IEEE/ACM 25th International Symposium on Quality of Service, IWQoS, IEEE, 2017, pp. 1–10.
[31] F. Carpio, S. Dhahri, A. Jukan, VNF placement with replication for loac balancing in NFV networks, in: 2017 IEEE International Conference on Communications, ICC, IEEE, 2017, pp. 1–6.
[32] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, J. Zhang, NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning, in: Proceedings of the International Symposium on Quality of Service, 2019, pp. 1–10.
[33] R.H. Crites, A.G. Barto, Elevator group control using multiple reinforcement learning agents, Mach. Learn. 33 (2) (1998) 235–262.
[34] M. Sewak, Deep Reinforcement Learning, Springer, 2019.
[35] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference on Machine Learning, PMLR, 2016, pp. 1928–1937.
[36] H. Khan, A. Elgabli, S. Samarakoon, M. Bennis, C.S. Hong, Reinforcement learning-based vehicle-cell association algorithm for highly mobile millimeter wave communication, IEEE Trans. Cogn. Commun. Netw. 5 (4) (2019) 1073–1085.
[37] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, N. Dormann, Stable Baselines3, GitHub Repository (2019) https://github.com/DLR-RM/stable-baselines3.
[38] D. Bhamare, A. Kassler, J. Vestin, M.A. Khoshkholghi, J. Taheri, IntOpt: In-band network telemetry optimization for NFV service chain monitoring, in: ICC 2019-2019 IEEE International Conference on Communications, ICC, IEEE, 2019, pp. 1–7.
[39] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

Mohammad Ali Khoshkholghi received his Ph.D. degree in Computer Science from the Faculty of Computer Science and Information Technology at the University Putra Malaysia in 2017, and Bachelors and Masters of Computer Science from Iran, in 2007 and 2011, respectively. He is currently a research associate in the Center for Telecommunication Research (CTR), Department of Engineering, King's College London (KCL), UK. Before joining KCL, he worked as a postdoctoral research fellow with the DISCO Research Group, Department of Computer science, Karlstad University, Sweden, from 2018 to 2020. He has also worked as a researcher and university lecturer within computer science in industry and academia. He serves as the referee, TPC and editorial board member for many prestigious journals and conferences. His research interests lie in the area of Edge and Cloud Computing, Network Function Virtualization and Machine Learning.

Toktam Mahmoodi (Senior Member, IEEE) received the B.Sc. degree in electrical engineering from the Sharif University of Technology, Iran, and the Ph.D. degree in telecommunications from King's College London, U.K. She was a Visiting Research Scientist with F5 Networks, San Jose, CA, USA, in 2013, a Postdoctoral Research Associate with the ISN Research Group, Electrical and Electronic Engineering Department, Imperial College, from 2010 to 2011, and a Mobile VCE Researcher, from 2006 to 2009. She has also worked in mobile and personal communications industry, from 2002 to 2006. She has worked with a Research and Development Team on developing DECT standard for WLL applications. She is currently the Head of the Centre for Telecommunications Research, Department of Informatics, King's College London. She has contributed to, and led a number of FP7, H2020, and EPSRC funded projects, advancing mobile and wireless communication networks. Her research interests include 5G communications, network virtualization, and low latency networking.