Accepted to IEEE Transactions on Cognitive Communications and Networking

Dynamic Topology Design of NFV-enabled Services using Deep Reinforcement Learning

Omar Alhussein, *Member, IEEE,* and Weihua Zhuang, *Fellow, IEEE*

***Abstract*—Next-generation networks are endowed with enhanced capabilities thanks to software-defined networking and network function virtualization (NFV). There is a radical shift from device-centric to experience-driven environments of which data is the primary driver behind its running engines. In this paper, we consider joint topology design, traffic routing and NF placement for unicast NFV-enabled services. We develop an end-to-end model-free deep reinforcement learning (RL) framework to dynamically allocate processing and transmission resources, while considering time-varying network traffic patterns. First, we provide a flexible pre-processing technique that represents and reduces the state space and action space of the considered joint problem for the deep RL algorithm. Second, we present a deep deterministic policy gradient (DDPG) algorithm that is enhanced with a model-assisted exploration procedure. Due to the multiple resource types with strongly adverse effects, the existing vanilla DDPG algorithm cannot achieve consistent performance. The model-assisted exploration procedure, which utilizes a perturbed step-wise sub-optimal integer linear program, bootstraps and stabilizes the vanilla DDPG algorithm and finds optimal solutions efficiently.**

***Index Terms*—Beyond 5G networks, DRL, NFV, NF chain embedding, service composition.**

I. Introduction

Network function virtualization (NFV) and software defined networking (SDN) have paved the road for the use of powerful and sophisticated service provisioning algorithms to cater towards the increasing demand and ubiquity in modern-day communication networks. Thanks to NFV and SDN, service providers possess a (partially) centralized view of the network substrate, whereby network elements (i.e., routers and servers) are programmable and virtualizable.

In an NFV-enabled environment, a service request specifies how a certain traffic class should be processed prior to arriving at the destination(s). That is, a network service manipulates the packet header and context of traffic flows in the data plane using the so-called (virtual) network functions. Consider for instance a firewall-protected cache dissemination service, where a source sends a signal to induce the cache to disseminate some information to a destination, as shown in Fig. 1-(a). The illustrated network service is comprised of a firewall and a cache. Traditionally, NFs are implemented at the end-users or at fixed middlewares in the network hardware. In NFV, such

Omar Alhussein was with the Department of of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, N2L 3G1, and is now with Huawei Technologies Canada Inc., Ottawa, ON, Canada, K2K 3J1, email: omar.alhussein@huawei.com.

Weihua Zhuang is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, N2L 3G1, email: wzhuang@uwaterloo.ca.

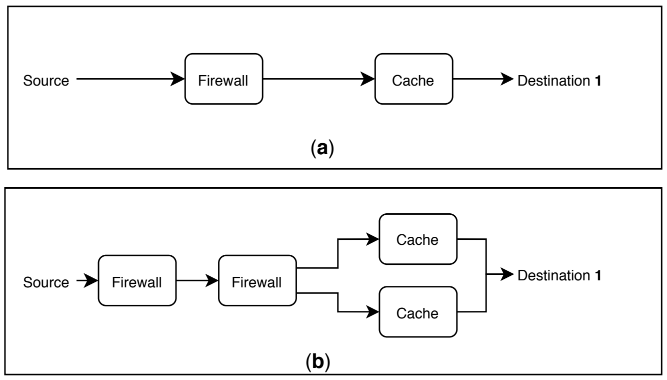


Fig. 1. Two NF chain topologies that realize the same network service request. The network service is firewall-protected and disseminates traffic to several destinations from a web-based cache which can be replicated.

NFs are virtualized, and can be deployed at several NFV nodes (commodity servers). Here the service request is represented by a directed acyclic graph, called an NF chain, whereby each vertex represents an NF or a terminal node. It is important to regard the NFs as logical nodes rather than as fixed nodes (as in the virtual network embedding literature). Therefore, in many network scenarios, semantically several NF instances can be deployed in parallel or in a distributive manner. There exist multiple NF chain topologies that represent the same service request, which gives rise to the problem of topology design (or service composition) [1]–[3]. For instance, Fig. 1-(b) illustrates another valid NF chain topology in which the firewall is implemented in a distributive manner and the cache is parallelized. Deploying the firewall in a distributed manner may be needed when the resources for such NF type is low on each NFV node. Moreover, the parallelization of the cache can be more processing-efficient when the processing resources are particularly scarce close to destination 1.

In addition to topology design, the logical NF chain needs to be embedded onto the network substrate, which gives rise to the NF placement and traffic routing problem. That is, we need to decide where to place the NF instances and how to route the traffic between the source and destinations to traverse the embedded NF instances [4]. The three aforementioned problems, namely the topology design, traffic routing, and NF placement, are essentially joint problems with correlated and conflicting effects. Designing a logical NF chain can be ineffective if not considered jointly with the traffic routing and NF placement.

In the core network, traffic patterns exhibit time-varying characteristics, which need not be periodic nor follow tractable traffic models. In addition, the scale of demands (in core networks) can be large relative to the granularity at which it is managed/routed [5]. Therefore, there is a need for dynamic provisioning that tackles time-varying characteristics of the service request.

So far, most existing approaches that address the joint composition, routing and NF placement (with fixed and varying rate requirements) are model-based or optimization-based. Model-based approaches place strong and often impractical assumptions on the traffic model and the environment constraints for tractability. In addition, communication networks are becoming increasingly time-varying, dynamic, and more difficult to model. In practice, optimization-based methods produce rigid solutions with arguably small degrees of freedom. A deviation from an assumed model can lead to a significant degradation in the performance of the underlying solution (such as in the probabilistic routing literature [6]). To this end, an alternative solution is to explore model-free data-driven provisioning methods that are powered by contemporary machine learning [7]–[9].

From a design perspective, NF (and/or link) migration is studied in existing works to deal with the time-varying traffic patterns and to alleviate network bottlenecks. In this paper, instead of considering complete NF instance and link migrations, we attempt to enforce the dynamic scaling of the network service by initializing (and tearing down) new NF instances along with already placed NF instances, to accomodate time-varying traffic characteristics. For instance, the topology of the network service in Fig. 1 can alternate between those in Fig. 1-(a), Fig. 1-(b) and other topologies based on characteristics of the network substrate and the time-varying traffic demand. Although theoretically the topology design can ameliorate the efficiency for some types of resources (e.g., function and link provisioning costs), it can exacerbate other types of resources (e.g., the routing cost and signaling overhead). Therefore, three aspects of resources, namely routing overhead, function provisioning cost, and link provisioning cost should be taken into consideration.

In light of the discussion above, this work offers the following contributions:

- We develop an end-to-end deep RL-based provisioning mechanism to dynamically allocate network resources based on the traffic pattern of the network service and the network substrate;
- We propose a pre-processing technique that represents the state space and action space of the considered problem. The pre-processing technique provides an auxiliary network transformation to extract end-to-end routing and NF placement configurations for the action space;
- We consider a deep deterministic policy gradient (DDPG) algorithm. Due to the multiple resource types with strongly adverse effects and a sparse reward function, the existing vanilla DDPG algorithm is shown to be inefficient and unreliable. Thus, we enhance the DDPG algorithm with an exploration procedure that utilizes experiences learned from a perturbed optimal step-wise solution.

The rest of this paper is organized as follows. Section II gives an overview of related works. Section III describes the system model under consideration. Section IV presents the research problem formulation. Section V discusses the deep RL framework for the proposed problem, which includes the pre-processing stage and the model-assisted deep RL algorithm. In Section VI, we conduct numerical performance evaluation of the proposed deep RL scheme, followed by concluding remarks of this study in Section VII.

II. Related Works

A. *Joint Composition, Traffic Routing and NF Placement*

Although several NF chain topologies express the same network policy semantically, the embedding process for each logical topology can yield different provisioning costs and long-run characteristics in a non-trivial manner. Generally, there are two approaches for the joint composition, NF placement and traffic routing. The first approach is to decouple the service composition (or topology design) from the NF placement and routing process [1], [10]. However, such an approach can be far from the globally optimal solution. Mehraghdam et al. propose a language model that formalizes an NFV-enabled multicast service [10]. Such language model is general as it allows for service composition. However, the addressed topological design aspect reduces to re-ordering the NFs from lowest to highest traffic inflation factors to minimize the required data rate of the NF chain. Mehraghdam et al. extend their work to account for the service composition, where a heuristic approach yeilds an extensive Pareto set of the possible composition configurations of a service as well as possible combinations of different services with respect to different optimization objectives [1].

Another approach is to design a heuristic algorithm such as a breadth-first search, where some composition operator is included to minimize the function and link provisioning costs [2], [3]. More specifically, Ye et al. jointly consider the topology design and embedding, where they formulate an integer linear program and develop a heuristic algorithm under the assumption that NFs in a service chain can be combined together [3]. The heuristic algorithm resembles an iterative two-step search, which iteratively combines two NF, and then performs embedding onto the physical substrate. The algorithm eventually selects an NF chain topology that minimizes the placement and routing cost. Beck and Botero present a coordinated breadth-first algorithm that minimizes the function and link provisioning costs, where the algorithm attempts to find a path from the source to the destination while being able to replicate NFs if needed [2]. Li et al. develop a two-stage heuristic algorithm that addresses the composition and embedding, where the location and functionality of the substrate nodes and the inflation factor of network functions are taken into consideration [11].

In the aformentioned works, the topology design and the NF placement and routing processes are decoupled. More recently, Coa et al. consider a genetic framework that designs an overall NF forwarding graph for multiple traffic flows to minimize the function and link provisioning costs and improve the acceptance ratio of services [12]. Jalalitabar et al. design a heuristic algorithm for the NF placement and routing that takes the dependence relationship between the NFs into account [13].

The aformentioned works present model- or optimization-based frameworks in which long-run provisioning costs and considerations can be difficult to incorporate. Moreover, the aformentioned works consider the joint composition, routing and NF placement under the assumption of a fixed data rate requirement for a service request.

B. Deep Reinforcement Learning based Dynamic Provisioning

Thanks to the recent success in deep RL, there is a renewed interest in the application of (deep) RL to networking problems. More specifically, RL has been used to tackle various traffic engineering problems that span routing, NF placement/migration, and NF scheduling [14]–[21]. One of the early works aims to maximize a network utility for a general communication network with K end-to-end communication sessions [14]. A traffic engineering-aware exploration is proposed which utilizes the DDPG algorithm with a prioritized experience replay technique. Pei et al. study the NF placement (or NF migration) problem in an NFV-enabled network under a time-varying traffic pattern [22]. Due to the large solution space, the authors divide the network into smaller regions and use a double deep Q-network algorithm to select a small number of potential NFV nodes. Troia et al. consider an NFV-enabled metro-core optical network, represented as a multi-layer network [17]. The goal is to maximize the number of successfully routed NF chains, while minimizing the reconfiguration penalty, blocking probability and power consumption. Gu et al. study the VNF orchestration and flow scheduling for NFV-enabled network services, whereby a model-assisted DDPG algorithm is used to aid in the convergence speed [16]. Qu et al. consider joint VNF migration and resource scaling in the presence of non-stationary traffic [20]. First, they propose a traffic parameter learning method based on change point detection and Gaussian process regression to detect changes in the stationarity of the traffic. Then, they propose an efficient modified Q-learning algorithm with penalty-aware prioritized experience replay to incorporate awareness of resource overloading penalty.

III. System Model

A. Network Functions and the Network Service

Time is partitioned into constant durations referred to as timesteps, indexed by τ. Let T denote the total number time durations under considered time. We consider a unicast service with time-varying traffic demand, expressed as

$$S = (u, t, \mathcal{V}, d(\tau)), \qquad \tau \in (0, T] \tag{1}$$

where u and t are the source and destination nodes, respectively; $\mathcal{V} = \{f_1, f_2, \dots, f_{|\mathcal{V}|}\}$ represents the set of NFs that need to be traversed in an ascending order for the source-destination pair; $d(\tau)$ denotes the required transmission rate at index τ in packet/s; Each NF $f_i, i = 1, 2, \dots, |\mathcal{V}|$, requires an amount of processing resources of $C(f_i(\tau))$ at index τ in packet/s. Depending on the operation semantics (which can be specified a priori), some NF types can be deployed in a parallel or sequential manner in geo-distributed NFV nodes.

B. Network Substrate

We are given a capacitated network substrate, $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ and $\mathcal{L}$ are the sets of nodes and links, respectively. Each physical link l ($\in \mathcal{L}$) has a residual transmission resource at time τ, $B_l(\tau)$, $\tau \in (0, T]$, in packet/s. Each node n ($\in \mathcal{N}$) has a residual processing resource, $C_n(\tau)$, $\tau \in (0, T]$, in packet/s. Here, $B_l(\tau)$ and $C_n(\tau)$ represent the residual resources while taking into account the background traffic and the embedded NF chain at time τ. Nodes can be either (i) switches that are capable of forwarding traffic only (with $C_n(\tau) = 0$), or NFV nodes (e.g., commodity servers) that are capable of both forwarding traffic and operating a set of NF instances. An NFV node is capable of provisioning a number of NF instances simultaneously as long as the available processing resources satisfy the deployed NF processing requirements.

IV. Problem Formulation

Given network substrate $\mathcal{G}$ and network service S with a time-varying data rate requirement, we want to develop a dynamic joint composition, routing and NF placement framework to minimize the function, link, and routing provisioning costs. The system operates in a time-slotted fashion over a potentially large time span, where certain dynamic decisions are taken at each timestep. Due to the bursty and time-varying nature of the data rate, altering the routing and NF placement configuration at each timestep should be discouraged due to the cost of setting up new NF instances.

The topology of an embedded NFV-enabled network service can be modeled as a composition of several paths. Each path emanates from the source to the destination, and traverses all the required NF instances. Let all the possible paths for unicast service S be given by $\mathcal{P}$. Let P ($\in \mathcal{P}$) be the P-th path on the network substrate that is activated to (partially) host the service request.

Let $x_{li}^P(\tau) \in \{0, 1\}$ be a decision variable at time indexed by duration τ, where $x_{l0}^P(\tau) = 1$ indicates that link l is used to direct traffic from u to the P-th instance of the first NF (f_1^P) for path P ($\in \mathcal{P}$), and $x_{li}^P(\tau) = 1$ indicates that link l is used to direct traffic from f_i^P to f_{i+1}^P. Correspondingly, define $\kappa_{li}^P(\tau) \in [0, 1]$ as a continuous flow variable that represents the fraction of flow used in link l to direct traffic from f_i^P to f_{i+1}^P such that

$$\begin{aligned}\kappa_{li}^P(\tau) =& d^P(\tau) x_{li}^P(\tau), \\ & \tau \in (0, T],\ P \in \mathcal{P},\ l \in \mathcal{L},\ i \in \Omega_0^{|\mathcal{V}|}\end{aligned} \tag{2}$$

where $d^P(\tau) \in [0, 1]$ is a continuous decision variable that represents the fraction of flow assigned for path P, and Ω_m^n denotes the set of integers from m to n ($> m$), i.e., $\Omega_m^n \triangleq \{m, m+1, \dots, n\}$. To meet the total required transmission rate, we impose constraint

$$\sum_{P \in \mathcal{P}} d^P(\tau) = 1, \quad \tau \in (0, T]. \tag{3}$$

Define binary decision variable $z_{ni}^P(\tau) \in \{0,1\}$, where $z_{ni}^P(\tau) = 1$ indicates that node n hosts f_i^P with an assigned fractional resources of $C(f_i^P(\tau)) \in [0,1]$. For each NF, to conserve the distributive processing resource, we impose

$$\sum_{P\in\mathcal{P}} C(f_i^P(\tau)) = C(f_i), \quad \tau \in (0,T],\ i \in \Omega_1^{|\mathcal{V}|}. \tag{4}$$

The overall data rate from all activated paths should not exceed the link transmission rate, $B_l(\tau)$, i.e.,

$$\sum_{P\in\mathcal{P}} \sum_{i=0}^{|\mathcal{V}|} \kappa_{li}^P(\tau) \le B_l(\tau), \quad l \in \mathcal{L},\ \tau \in (0,T]. \tag{5}$$

Moreover, the overall rate of the NF instances that are placed on an NFV node should not exceed the node processing rate $C_n(\tau)$, i.e.,

$$\sum_{i=1}^{|\mathcal{V}|} \sum_{P\in\mathcal{P}} C(f_i^P(\tau)) \le C_n(\tau), \quad n \in \mathcal{N},\ \tau \in (0,T]. \tag{6}$$

Objective function: Different logical service topologies and embedding configurations can incur different routing and NF setup costs. To route traffic that belongs to a service request, a switch needs to add one routing entry in the forwarding table that specifies the header and the next port (or physical link). Therefore, for the routing overhead, we consider that each physical link hosting the service request incurs an additional cost of η_1 [23]. The routing cost is expressed as

$$c_1(\tau) = \sum_{i=0}^{|\mathcal{V}|} \sum_{l\in\mathcal{L}} \eta_1 x_{li}(\tau),\ \tau \in (0,T] \tag{7}$$

where $x_{li}(\tau) \in \{0,1\}$ is a decision variable at time τ, with $x_{li}(\tau) = 1$ indicating that $x_{li}^P(\tau) = 1$ for some path $P\,(\in \mathcal{P})$, and

$$x_{li}^P(\tau) \le x_{li}(\tau), \quad \tau \in (0,T],\ P \in \mathcal{P}. \tag{8}$$

The link and function provisioning costs can be expressed as

$$c_2(\tau) = \sum_{i=0}^{|\mathcal{V}|} \sum_{l\in\mathcal{L}} \sum_{P\in\mathcal{P}} d(\tau)\frac{\kappa_{li}^P(\tau)}{B_l(\tau)},\ \tau \in (0,T] \tag{9}$$

and

$$c_3(\tau) = \sum_{i=1}^{|\mathcal{V}|} \sum_{n\in\mathcal{N}} \sum_{P\in\mathcal{P}} d(\tau)\frac{C(f_i^P(\tau))}{C_n(\tau)} z_{ni}^P(\tau), \quad \tau \in (0,T] \tag{10}$$

respectively. We also consider the NF instance setup cost, which is incurred only when an NF instance is initialized at an NFV node. Let $z_{ni}(\tau) \in \{0,1\}$ be a decision variable, where $z_{ni}(\tau) = 1$ indicates that function f_i is initialized on node n. Then, the NF setup cost can be expressed as

$$c_4(\tau) = \sum_{n\in\mathcal{N}} \sum_{i=1}^{|\mathcal{V}|} \eta_2 z_{ni}(\tau),\ \tau \in (0,T] \tag{11}$$

where η_2 is the NF setup cost which can include the installation cost and the respective signalling overhead from the controller to the node, and

$$\sum_{P\in\mathcal{P}} z_{ni}^P(\tau) - \sum_{P\in\mathcal{P}} z_{ni}^P(\tau-1) \le z_{ni}(\tau),\ \tau \in [0,T]. \tag{12}$$

In summary, the optimization problem for the dynamic joint composition, routing and NF placement problem with time-varying data rate is expressed as

$$\text{minimize} \quad \sum_{\tau=0}^{T} \sum_{k=1}^{4} \omega_k c_k(\tau) \tag{13a}$$

subject to:

$$\forall \tau \in (0,T],\ l \in \mathcal{L}: \sum_{i=0}^{|\mathcal{V}|} \sum_{P\in\mathcal{P}} \kappa_{li}^P(\tau) \le B_l(\tau) \tag{13b}$$

$$\forall \tau \in (0,T],\ n \in \mathcal{N}: \sum_{i=1}^{|\mathcal{V}|} \sum_{P\in\mathcal{P}} C(f_i^P(\tau)) \le C_n(\tau) \tag{13c}$$

$$(2), (3), (4), (8), (12). \tag{13d}$$

The problem in (13) is an online constrained problem since the data rate, $d(\tau)$, is revealed in an online manner, where the objective is to minimize the long-run provisioning cost of the time-varying service request. Knowledge of the traffic pattern of service request and the background traffic is particularly crucial for the NF setup cost ($c_4(\tau)$).

V. Deep Reinforcement Learning Framework

A. Reinforcement Learning Background

We consider a standard RL setting consisting of an agent interacting with an environment in the discrete timesteps indexed by τ, referred to as learning steps. At each learning step, the agent observes a set of states, produces a set of actions to affect the states, and consequently receives a reward. Different from other learning paradigms (e.g., supervised learning), RL addresses a *sequential* decision making problem in a holistic manner, whereby an agent needs to find a desired behaviour (or policy) that maps the set of states to actions to maximize both immediate and *future* rewards. Here we consider a fully-observable environment where a single RL agent observes all the states from the network substrate and network service. Therefore, it is modeled as a Markov decision process with state space $\mathcal{S}$, action space $\mathcal{A}$, initial state distribution $\Pr(s_1)$, transition probabilities $\Pr(s_{\tau+1}|s_\tau, a_\tau)$, and reward function r_τ $(: \mathcal{S} \times \mathcal{A} \to \mathbb{R})$. Let the (discounted) accumulation of future rewards be the return (R_τ), defined as

$$R_\tau = \sum_{i=\tau}^{\infty} \vartheta^{i-\tau} r_i \tag{14}$$

where $\vartheta \in (0,1]$ is a discount factor that represents the present value of future rewards. The larger ϑ, the more farsighted the agent is, i.e., the more valued future rewards are. The agent aims to find a policy mapping that maximizes $\mathbb{E}_{s_\tau}[R_\tau|s_\tau]$, where $\mathbb{E}[\cdot]$ is the mathematical expectation. At learning step τ, the so-called action-value function (or Q-function) resembles the expected return after taking action a_τ while following policy λ, defined as $Q^\lambda(s_\tau, a_\tau) = \mathbb{E}[R_\tau|s_\tau, a_\tau]$. If policy λ is deterministic (i.e., $\lambda : \mathcal{S} \to \mathcal{A}$), the recursive Bellman equation can be used to learn the action-value function as follows,

$$Q^\lambda(s_\tau, a_\tau) = \mathbb{E}[r_\tau + \vartheta Q^\mu(s_{\tau+1}, \pi(s_{\tau+1}))]. \tag{15}$$

In large-scale topologies, probing all the network elements can be impractical, and one agent can impose a signaling bottleneck. Thus, a practical approach is to re-formulate the problem as a partially-observable MDP with single or multiple DRL agents. In such settings, an agent does not view the whole network substrate at one time. Therefore, it needs to be equipped with an internal memory representation (often implemented via a recurrent neural network) to combat the partial observability effect (which induces a pronounced non-Markovian impact on the problem setup). Such problem settings require further studies.

B. Pre-processing Stage

The described problem requires an end-to-end solution, whereby the design of service topology, placement of NFs, and routing configuration on the network substrate should be considered. Therefore, if approached naively, this problem can lead to an explosion in the number of states (and actions) for network substrates of a moderate size. A large number of empirical studies on the properties of real network substrates reveal that the average path length for a source-destination pair is very small. This is due to the observation that real networks tend to have a degree distribution with a power-law tail, which is known as the small-world phenomena [24], [25]. In such scale-free networks, the average path length is asymptotically logarithmic in the number of nodes of the network substrate (i.e., $= \mathcal{O}(\log |\mathcal{N}|)$). Therefore, in practice, the average number of edge-disjointed paths is small. Based on the aforementioned observations and to have a compact learning representation, we model an embedded service topology as a composition of several NF placement and routing configurations from the source to the destination.

To generate the routing and NF placement configurations, we utilize an auxiliary multilayer transformation of the network substrate [26]. We model the transformation as a directed graph, $\mathcal{G}_M = (\mathcal{N}_M, \mathcal{L}_M)$, where $\mathcal{N}_M \subseteq \mathcal{N} \times \mathcal{X}$ is the set that contains all nodes, in which node n ($\in \mathcal{N}$) is present in a corresponding layer x ($\in \mathcal{X}$); denote such node by n^x. Set $\mathcal{L}_M \subseteq \mathcal{N}_M \times \mathcal{N}_M$ comprises all intra-layer and inter-layer edges. Intra-layer edges, $\mathcal{L}_A = \{(n^x, v^y) \in \mathcal{L}_M | x = y \in \mathcal{X}\}$, represent the connections between the network elements in each layer. Inter-layer edges, $\mathcal{L}_I = \{(n^x, n^y) \in \mathcal{L}_M | x \neq y \in \mathcal{X}\}$, are used to encode the placement decisions in which a traversal of an edge from one layer x ($\in \mathcal{X}$) to another layer y ($\in \mathcal{X}$) maps to the processing of the xth NF instance in a network service.

The auxiliary multilayer transformation is constructed as follows.

1) Create $|\mathcal{X}|$ copies of network substrate $\mathcal{G}$. Each copy $(\mathcal{G}^i(\mathcal{N}^i, \mathcal{L}^i))$ represents one layer, where $i = \{0, \ldots, |\mathcal{V}|\}$.
2) Assign the source node to its corresponding node at layer 0 ($= u^0$, $u^0 \in \mathcal{N}^0$);
3) Assign the destination node to its corresponding node at the last layer ($= t^{|\mathcal{V}|}$, $t^{|\mathcal{V}|} \in \mathcal{N}^{|\mathcal{V}|}$);
4) For the first $|\mathcal{V}|$ layers, construct inter-layer edges from layer i to layer $i+1$ ($l \leftarrow (n^i, n^{i+1})$) for each NFV node that can host f_i (i.e., if $n^i \in \mathcal{F}_i$). The processing resources of each inter-layer edge, $l \leftarrow (n^i, n^{i+1})$, is equivalent to that of the corresponding NFV node n^i.

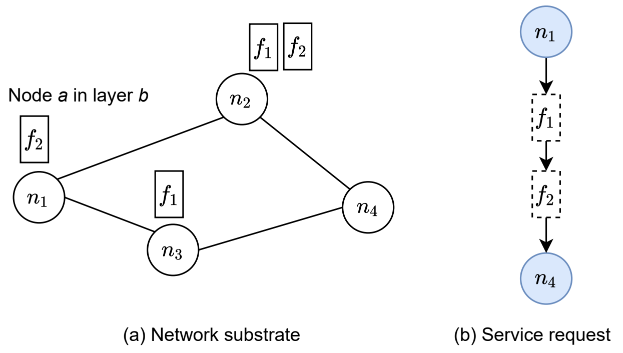


Fig. 2. A problem input: (a) A network substrate along with the permissible NFs on each network element, and (b) the logical topology of a unicast service request.

Algorithm 1: Construction of multilayer network transformation and generation of several NF placement and routing configurations.

```
1  Procedure multipleJRP($\mathcal{G}, S$);
   Input  : $\mathcal{G}$, $S = (u, \mathcal{D}, f_1, f_2, \ldots, f_{|\mathcal{V}|}, d)$
   Output: $\mathcal{P}$
2              ▷ Construction of multilayer transformation $\mathcal{G}_M$
3  $\{\mathcal{G}^k(N^k, L^k)\}_{k=0}^{|\mathcal{V}|+1} \leftarrow \mathcal{G}(N, L)$;
4  $u^0 \leftarrow u$ (source node at layer 0 as source $u$);
5  $t^{|\mathcal{V}|} \leftarrow t$ (destination node at layer $|\mathcal{V}|$ as destination $t$);
6  $\mathcal{L}_I \leftarrow \{\}$;
7  for $k = 0 : (|\mathcal{V}| - 1)$ do
8      for $n^k \in \mathcal{F}_k$ do
9          Add $l \leftarrow (n^k, n^{k+1})$ to $\mathcal{L}_I$;
10         $C(l) = C(n^k)$;
11     end
12 end
13             ▷ Populating $\mathcal{P}$ with virtual segment-disjoint
   configurations
14 $\mathcal{P} \leftarrow \{\}$;
15 for $i = 1 : \mathcal{V}$ do
16     $\mathcal{G}_M^{\text{tmp}} \leftarrow \mathcal{G}_M$;
17     while $\exists$ ShortestPath($\mathcal{G}_M^{\text{tmp}}$; $u^0$, $t^{|\mathcal{V}+1|}$) do
18         $\mathcal{P} \overset{+}{\leftarrow}$ ShortestPath($\mathcal{G}_M^{\text{tmp}}$; $u^0$, $t^{|\mathcal{V}+1|}$);
19         Remove path for virtual segment $f_i$–$f_{i+1}$ from
             $\mathcal{G}_M^{\text{tmp}}$;
20     end
21 end
22 Remove replicated path (or tree) configurations;
23 return $\mathcal{P}$;
```

We provide an illustrative example of constructing the auxiliary graph transformation and how service composition is incorporated in Figs. 2 and 3. Fig. 2 illustrates a service

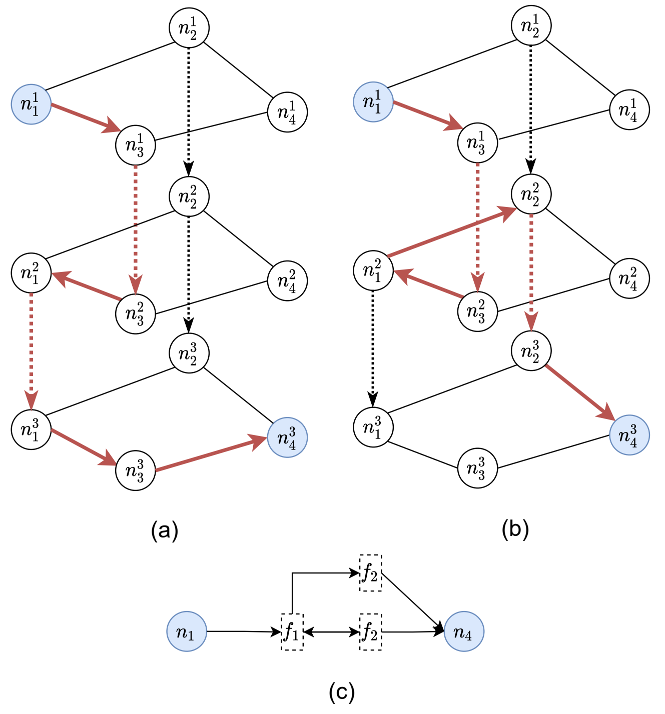


Fig. 3. The auxiliary network transformation for the problem input in Fig. 2 with a different path traversals from source to destinations in (a) and (b). Activating both trees results in the logical topology shown in (c).

request of two NFs (f_1 and f_2), where the source is n_1 and the destination node is n_4. We also have a network substrate of 4 nodes that can host either NF type, both, or neither. Fig. 3 illustrates the construction of the auxiliary graph transformation. Since we have two NFs, the auxiliary transformation has three layers. As NFV nodes n_2 and n_3 can host f_1, we construct the inter-layer edges, $n_2^1 \to n_2^2$ and $n_3^1 \to n_3^2$. Similarly, n_1 and n_2 can host f_2. Thus, we construct the inter-layer edges, $n_1^2 \to n_1^3$ and $n_2^2 \to n_2^3$.

Algorithm 1 summarizes the construction of the auxiliary transformation and the generation of routing and NF placement configurations. With the auxiliary transformation, a path traversal (while considering only the edge costs) from the source to the destination represents a routing and NF placement solution for the service in the original network substrate graph. Here, we have an illustration of two different path traversals in Figs. 3-(a) and 3-(b). Let a path in the network substrate from f_i to f_{i+1} be called a virtual segment. Using the multilayer transformation, we find all the virtual segment-disjoint paths as shown in lines 13-22 in Algorithm 1.

Combining several paths yields a variety of service topologies and embedding configurations. The desired behavior is to have a dynamic service topology that changes depending on the traffic pattern of the service request to optimize objective function (13a). That is, the service topology should grow and shrink in size (i.e., activate and tear down temporary NF instances), while taking the model-free traffic patterns into account. For example, activating both paths in Figs. 3-(a) and 3-(b) results in the service topology in Fig. 3-(c).

C. States, Actions, and Reward Function

Critical to the success of any RL framework is the design of accurate yet compact state space and action space that capture the relevant features in the environment and accurate reward structure of the problem. In what follows, we describe the state space ($\mathcal{S}$), action space ($\mathcal{A}$), and reward function.

Action space: Given the output of Algorithm 1, the action space is to choose which paths to activate and what the assigned proportion for each activated path from $\mathcal{P}$, i.e., $\boldsymbol{a}_\tau = \{d^P(\tau)\}_{P\in\mathcal{P}}$, where $\sum_{P\in\mathcal{P}} d^P(\tau) = 1$ and $\boldsymbol{a}_\tau \in \mathcal{A}$.

State space: At the beginning of timestep τ, the agent should proactively adjust the service topology and the embedding solution of the service by relying on the available states. Therefore, the state space is comprised of

1) the data rate of the previous timestep ($d(\tau-1)$);
2) the data rate for the current timestep ($\hat{d}(\tau)$);
3) the utilization ratio of the network elements of all concerned paths in the previous timestep, i.e.,

$$\sum_{P\in\mathcal{P}} \frac{d^{P|l\in P}(\tau-1)}{B_l(\tau-1)},\ \forall l \in \mathcal{L}\cap\mathcal{P} \tag{16}$$

and

$$\sum_{P\in\mathcal{P}} \sum_{i=1}^{\mathcal{V}} \frac{C(f_i^{P|n\in P}(\tau-1))}{C_n(\tau-1)},\ \forall n \in \mathcal{N}\cap\mathcal{P}; \tag{17}$$

4) the path allocation of the previous timestep ($\boldsymbol{a}_{\tau-1}$).

Reward function: The reward is to first maximize the additive inverse of the cost function in (13a). Second, to incorporate the constraints in (13b) and (13c), we penalize the reward function when a resource violation occurs in the physical links or NFV nodes. That is,

$$r_\tau = -(\sum_{k=1}^{4} \omega_k c_k(\tau) + \sum_{l\in\mathcal{L}} \psi_l' + \sum_{n\in\mathcal{N}} \psi_n'), \quad \tau \in [0, T] \tag{18}$$

where $\psi_l' \in \{0, \psi_l\}$ and $\psi_n' \in \{0, \psi_n\}$ are penalty factors, such that $\psi_l' = \psi_l$ and $\psi_n' = \psi_n$ are associated with violating constraints (13b) and (13c), respectively.

The number of network elements of all concerned paths dominates the size of the state space, i.e., $|\mathcal{S}| = \mathbb{R}^{(2+|\mathcal{L}\cap\mathcal{P}|+|\mathcal{N}\cap\mathcal{P}|+|\mathcal{P}|)}$. The size of the action space is the number of concerned paths, i.e., $|\mathcal{A}| = \mathbb{R}^{|\mathcal{P}|}$. Without proper generalization, the sample complexity is primarily dependent on the cardinality of the state-action space [27]. Therefore, the training can become cumbersome even for moderate-sized networks, especially without generalization capability and enhanced exploration procedure.

The reward structure exhibits strongly conflicting objectives, which can hinder an efficient exploration in the learning algorithm. Moreover, the rewards have varying degrees of sparsity. For example, the reward due to the routing cost (7) is only observed when a path activates/deactivates, which occur when $d^P(\tau) \in \{0, 1\}$, $P \in \mathcal{P}$. A more sparse event is when a new NF instance is initialized at an NFV node. Therefore, we have a reward structure that is sparse and more sensitive to certain changes in the action space. One potential approach to tackle the challenges is to perform reward engineering, i.e., to smooth and shape the reward function. In our case, shaping the reward can further hinder the exploration. Moreover, a shaped reward does not accurately reflect the intended behavior, e.g., a smooth penalty for the resource violation can push the utilization ratio of the physical links and NFV nodes to certain inadvertent levels. Therefore, to tackle the aforementioned challenges, we propose a model-assisted deep RL algorithm as follows.

D. Deep RL Algorithm (DDPG)

For RL problems with a large state space, the use of non-linear function approximators is needed. Prior to the work of Mnih et al. [28], the use of deep neural networks (or generally non-linear functions) as parameterized function approximators for learning the action-value function was avoided due to the instability in the learning process. Mnih et al. proposed the deep Q-network (DQN), which can learn the action-value function with large state space and small discrete action space. This success was achieved with two techniques, namely through (i) the use of an off-policy replay buffer to break the correlations between the sequential samples in the learning process, and (ii) the use of an earlier delayed replica of the primary Q-network (called the target network) to stabilize the learning process [28]. The replay buffer provides the ability to visit previous experiences more than once, rendering such an approach more sample-efficient than other on-policy algorithms.

Our proposed problem is online control with a continuous action space. In continuous action spaces, we need to find a policy that optimizes the action space at every timestep through an iterative optimization process. Discrete learning algorithms, such as Q-learning and DQN, cannot be applied directly. In our context, a discretization of the action space is not practical as the action space can grow exponentially for networks of moderate size. Recently, Lillicrap et al. adapt the DQN to the continuous action domain through the use of an off-policy actor-critic architecture with DDPG learning algorithm [29].

Consider an approximate action-value function parameterized by $\boldsymbol{\theta}^Q$ $(Q(\boldsymbol{s}_\tau, \boldsymbol{a}_\tau|\boldsymbol{\theta}^Q))$. Such function can be optimized by minimizing the loss function,

$$L(\boldsymbol{\theta}^Q) = \mathbb{E}[(Q(\boldsymbol{s}_\tau, \boldsymbol{a}_\tau|\boldsymbol{\theta}^Q) - y_\tau)^2] \tag{19}$$

where $y_\tau = r_\tau + \vartheta Q(\boldsymbol{s}_{\tau+1}, \boldsymbol{a}_{\tau+1}|\boldsymbol{\theta}^Q)$ represents the expected return.

The DDPG algorithm maintains a parameterized actor policy $(\mu(\boldsymbol{s}|\boldsymbol{\theta}^\mu))$ and a parameterized critic $(Q(\boldsymbol{s}_\tau, \boldsymbol{a}_\tau|\boldsymbol{\theta}^Q))$ in a primary network. The actor policy maps the states to a continuous action space. The parameterized critic function is learned using the Bellman equation as in (19). The actor policy is updated by optimizing the expected return from the start through the chain rule [29],

$$\begin{aligned}\nabla_{\boldsymbol{\theta}^\mu} H &\approx \mathbb{E}\big[\nabla_{\boldsymbol{\theta}^\mu} Q(\boldsymbol{s}, \boldsymbol{a})|\boldsymbol{\theta}^Q)|_{\boldsymbol{s}=\boldsymbol{s}_\tau, \boldsymbol{a}=\mu(\boldsymbol{s}_\tau|\boldsymbol{\theta}^\mu)}\big] \\ &= \mathbb{E}\big[\nabla_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a})|\boldsymbol{\theta}^Q))|_{\boldsymbol{s}=\boldsymbol{s}_\tau, \boldsymbol{a}=\mu(\boldsymbol{s}_\tau)} \nabla_{\boldsymbol{\theta}^\mu} \mu(\boldsymbol{s}|\boldsymbol{\theta}^\mu)|_{\boldsymbol{s}=\boldsymbol{s}_\tau}\big].\end{aligned} \tag{20}$$

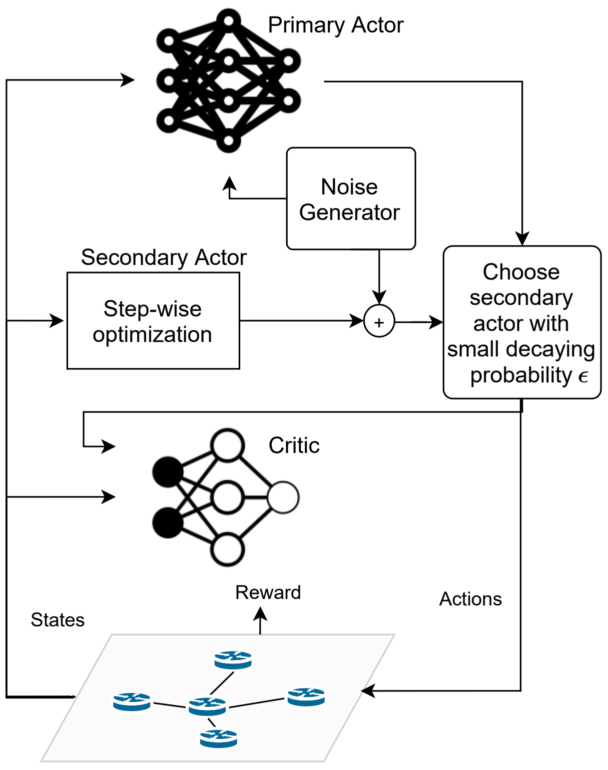


Fig. 4. The modified model-assisted DDPG algorithm.

For stability, the DDPG algorithm maintains a target network which is a replica of the primary network with actor policy $\mu'(\boldsymbol{s}|\boldsymbol{\theta}^{\mu'})$ and critic $Q'(\boldsymbol{s}_\tau, \boldsymbol{a}_\tau|\boldsymbol{\theta}^{Q'})$. The weights of the target network are set to slowly track the primary network's weights such that $\theta^{Q'} = \upsilon\theta^Q + (1-\upsilon)\theta^{Q'}$, and $\theta^{\mu'} = \upsilon\theta^\mu + (1-\upsilon)\theta^{\mu'}$, where $\upsilon \ll 1$. Parameter υ represents a tradeoff between the stability and the rate of the learning process. A very small υ greatly stabilizes the learning process at the expense of slowing the learning process. It merits to mention that the use of the action-value function (as a proxy to learn the optimal policy) provides a generalization capability where the agent can generalize new state-action pairs based on previously experienced states.

Model-assisted exploration – The exploration procedure can be treated independently from the learning algorithm because the DDPG is an off-policy algorithm. In the vanilla DDPG algorithm [29], a perturbed exploration policy, $\tilde{\mu}(\boldsymbol{s}_\tau)$, is constructed by adding a temporally correlated noise process directly to the action space, i.e., $\tilde{\mu}(\boldsymbol{s}_\tau) = \mu(\boldsymbol{s}_\tau) + \boldsymbol{n}$, where $\boldsymbol{n} \sim OU(0, \sigma^2)$ is the Ornstein-Uhlenbeck process. In each episode, the exploration process will produce a different action for the same state since the (correlated) noise is independent of the current state, $\boldsymbol{s}_\tau$. Such behavior is detrimental to our problem. A properly structured, state-dependent exploration methodology is to inject some noise directly to the parameters of the actor's deep neural network at the beginning of an episode, such that $\tilde{\theta}^{\tilde{\mu}} = \theta^\mu + n$, where $n \sim \mathcal{N}(0, \sigma_n^2)$ and $\tilde{\theta}^{\tilde{\mu}}$ is

Algorithm 2: Model-assisted DDPG-based learning algorithm

```
1  Generate set of routing and NF placement configurations
     for $S^r$ (using Algorithm 1);
2  Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor
     network $\mu(s|\theta^\mu)$;
3  Initialize target network $Q'$ and $\mu'$ with weights
     $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$;
4  Initialize replay buffer $R$;
5  for e = 1 : $M$ do
6      Receive initial observation states $\boldsymbol{s}_1$;
7      Create a perturbed actor network such that
         $\tilde{\theta}^{\tilde{\mu}} = \theta^\mu + \mathcal{N}(0, \sigma_n^2)$;
8      for t = 1 : $T$ do
9          if $\mathcal{U}(0,1) \leq \epsilon$ then
10             Select actions $\boldsymbol{a}_t$ by solving (13);
11             Add noise to action ($\boldsymbol{a}_t = \boldsymbol{a}_t + \mathcal{N}(0, \sigma_n^e)$);
12         end
13         else
14             Select actions $\boldsymbol{a}_t$ from perturbed actor
                 network $\tilde{\mu}$;
15         end
16         Execute actions $\boldsymbol{a}_t$, and observe reward $r_t$ and
             new states $\boldsymbol{s}_t$;
17         Store transitions $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ in $R$;
18         Sample a random mini-batch of $N$ transitions
             $(\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}_{i+1})$ from $R$;
19         Set $y_i = r_i + \gamma Q'(\boldsymbol{s}_{i+1}, \mu'(\boldsymbol{s}_{i+1}|\theta^{\mu'})|\theta^{Q'})$;
20         Update critic by minimizing loss function using
             (19);
21         Update the actor policy using the sampled policy
             gradient using (20);
22         Update the target networks:
               $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
               $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
23     end
24     Decrease $\epsilon$ if bigger than zero ($\epsilon \leftarrow \epsilon^\Delta$ if $\epsilon > 0$);
25 end
```

the perturbed actor policy [30]. Parameter σ_n can be controlled by measuring its induced variance on the action space [30]. Moreover, to aid the exploration process, we guide the learning algorithm with some domain-knowledge as follows. With a small decaying probability (ϵ), choose a perturbed action by invoking a step-wise version of the defined optimization problem in (13), which can be solved by Gurobi solver.

Algorithm 2 summarizes the model-assisted DDPG-based learning algorithm. First, using the proposed pre-processing stage, we generate a set of routing and NF placement configurations for the service request (in line 1). Then, we randomly initialize the primary and secondary actor and critic deep neural networks, and we initialize the replay buffer (lines 2-4). At the beginning of each episode, we create a perturbed actor network by injecting Gaussian noise to the parameters of respective deep neural network (in line 7). Then, for each timestep, we select an action vector from either the perturbed actor policy or the perturbed step-wise solution with probability $1-\epsilon$ and ϵ, respectively (in lines 9-15). The selected action vector is executed in the network substrate, for which a reward is observed, and new states are produced (in line 16). Then, we store the previous transition tuple (or experience), namely $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$, in the replay buffer (in line 18). Here, we sample several transitions at random from the replay buffer to break the correlations caused by sequential experiences, and update the critic network by computing the loss function using (19) (in line 20), and we update the actor network using 20 (in line 21). Finally, we slowly update the target network parameters to track the primary network (in line 22). At the end of each episode, we decrease step-wise exploration factor ϵ (in line 24).

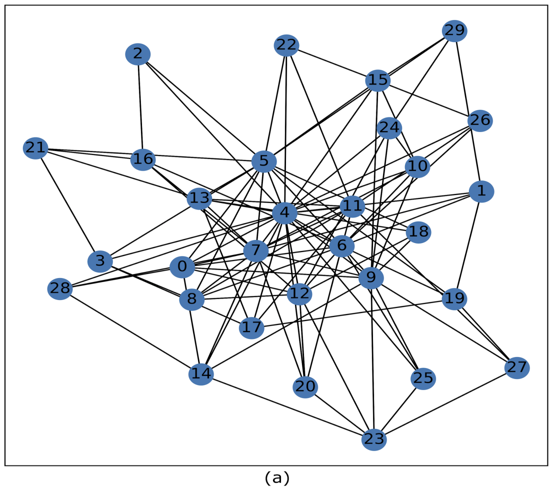
Fig. 4 provides a pictorial representation of the main components in the model-assisted DDPG framework. Here, we have two deep neural networks that correspond to the actor and critic, respectively. The actor network takes the states as input and produces an action vector. For the actor network, we apply a Softmax layer at the output layer to produce an action vector that sums up to unity. The critic network takes the states and actions as input and produces one output that corresponds to the action-value function ($Q(\boldsymbol{s}_\tau, \boldsymbol{a}_\tau)$). We also have a temporary actor that produces the step-wise solution and is selected to act with small decaying probability ϵ.

VI. Performance Evaluation

In this section, we numerically evaluate and analyze the performance of the proposed model-assisted deep RL algorithm. To evaluate the efficiency of the proposed approach, we use two benchmarks, namely step-wise optimization and vanilla DDPG algorithm. In the step-wise optimization, we run an instantaneous version of (13) at each decision epoch to maximize the instantaneous reward. The vanilla DDPG algorithm does not utilize the domain-based exploration method as described in Subsection V-D. That is, for the vanilla DDPG algorithm, we omit lines 9-12 from Algorithm 2.

Both learning algorithms employ 2 hidden feed-forward neural network layers with 32 nodes for both the actor and the critic. Also, we use a leaky rectifier (ReLU) as activation functions in the hidden layers. The learning rates of the actor and critic are 10^{-4} and 10^{-3}, respectively. The discount factor (ϑ) and the learning rate (υ) are set to 0.99 and 0.001, respectively. The memory and batch sizes are set to 10^6 and 64, respectively, and the variance of the action noise (σ_n^2) is set to 0.1. For the model-assisted DDPG algorithm, we set the probability of invoking the model-based solution to $\epsilon = 0.08$ with a decaying factor of $\Delta = 1.001$.

For the traffic pattern of network services, we utilize an open-source traffic trace that is collected and maintained by the Widely Integrated Distributed Environment research group [31]. The traffic trace is collected from a 96-hour traffic on four consecutive days in 2019. We extract the HTTP traffic (port 443) from the raw data packet trace. Based on the timestamp of each packet arrival, we sample the number of packets every


(a)

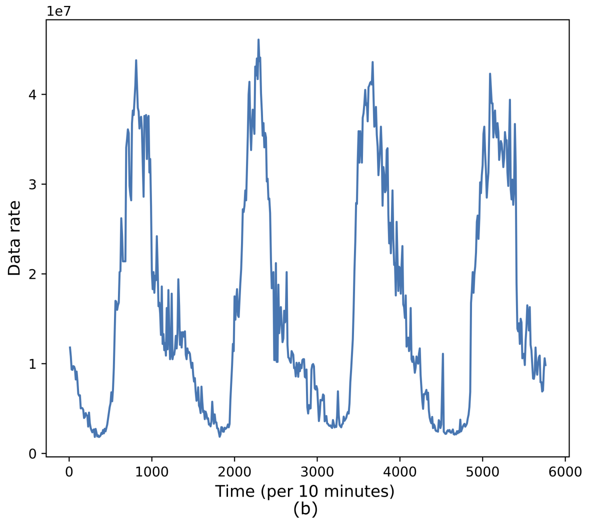
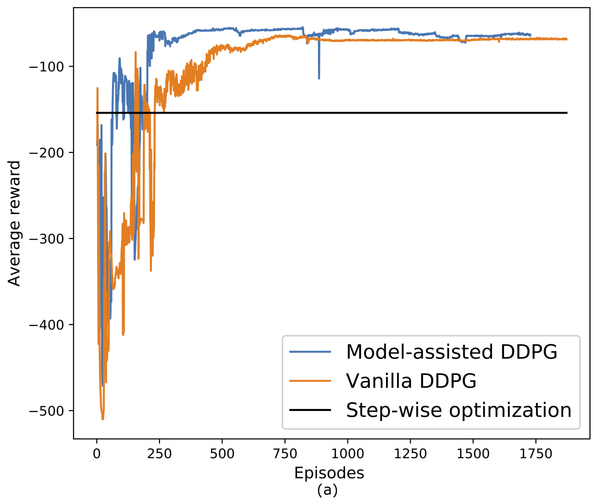


Fig. 5. (a) A random topology with 30 nodes that is generated using the Barbási–Albert preferential attachment model; (b) HTTP-type traffic trace.



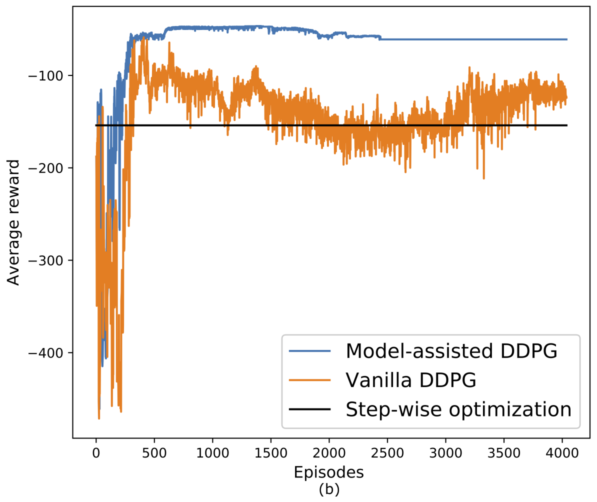


Fig. 6. Average reward per episode of the proposed model-assisted DDPG algorithm compared to the step-wise optimization and the vanilla DDPG algorithm while varying the randomization seeds in (a) and (b) over a random topology with 30 nodes.

10 minutes as shown in Fig. 5-(b). Here, the traffic rate varies from 5 to 65 Giga packet/s.

For the network substrate, we use the Barbási–Albert preferential attachment model to generate scale-free random networks [25] (see Fig. 5-(a)). The transmission and processing resources are randomly distributed between 25 and 55 Giga packet/s. We consider that all nodes can be NFV nodes, where each NFV node can host 2/3 of the possible NF types at random. The transmission and processing resources are randomly distributed between 25 and 55 Giga packet/s.

First, we conduct a case study on the performance of the proposed algorithm and the two benchmarks over a random network substrate with 30 nodes ($|\mathcal{N}| = 30$). We generate a service request with one NF from node 23 to node 15. Fig. 6 shows the average reward per episode for the considered problem setup with two different randomization seeds in Figs. 6-(a) and 6-(a), respectively. As shown, the proposed model-assisted DDPG algorithm converges to a solution that outperforms the step-wise optimization, implying that the temporal traffic patterns are learned and harnessed to minimize the NF setup cost as the traffic rate varies. More specifically, it is observed that the proposed algorithm learns to activate a new path (or install a new NF instance) only due to the long-run time-varying characteristics rather than any short-term increase in the traffic demand.

Comparing the learning algorithms, we observe that the model-assisted algorithm is more robust and is faster to converge and stabilize. In Fig. 6-(a), the model-assisted algorithm stabilized after episode 400, whereas the vanilla algorithm stabilized after episode 750. In Fig. 6-(b), where only the randomization seed is changed, the performance of the vanilla DDPG kept fluctuating and did not converge until episode 4000. This experiment demonstrates how brittle is the Vanilla DDPG algorithm, as it has been observed to be very sensitive to hyperparameters. The brittleness and hyper-sensitivity of the vanilla DDPG algorithm have been reported in other studies [32], [33].

Next, we analyze the performance of the trained agent, i.e., the proposed model-assisted algorithm after convergence.

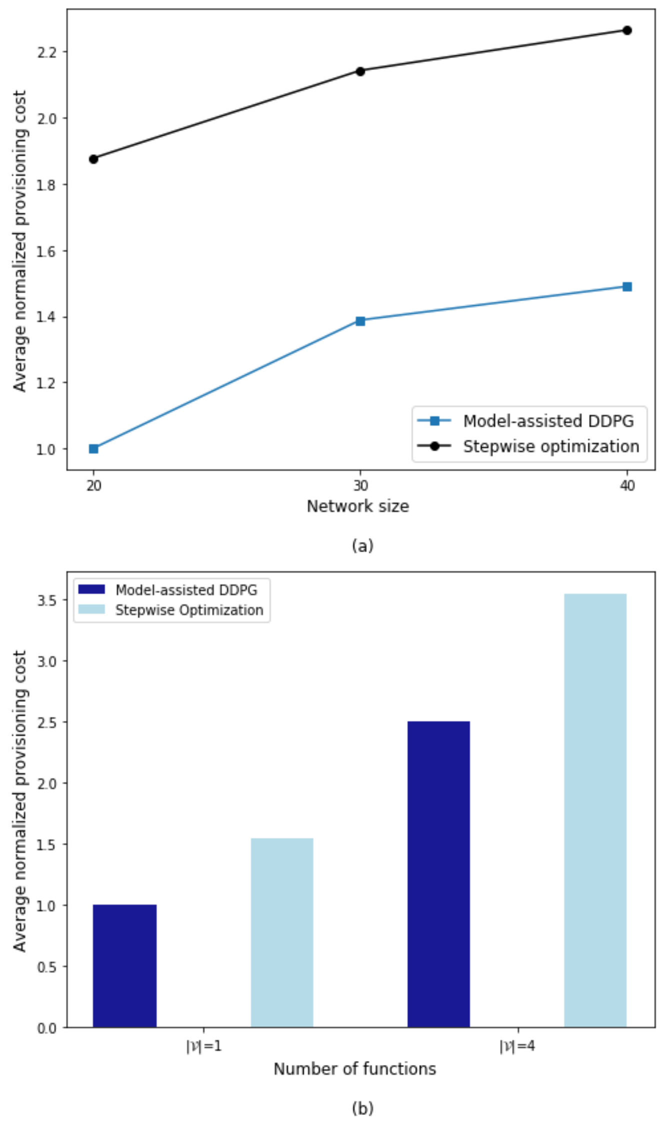


Fig. 7. Performance of the proposed algorithm and the stepwise optimization over a random network substrate as the network size and the number of NFs vary in (a) and (b), respectively.

Similarly, we use the Barbási–Albert method to generate random network substrates. We repeat the experiment over different instantiations of the network substrate to measure the average performance. We generate a service request from node 23 to node 15 with varying number of NFs. Due to the extreme brittleness of the Vanilla DDPG algorithm, hereafter we only compare the trained model-assisted agent with the step-wise optimization.

Fig. 7-(a) shows the average provisioning cost of the trained agent and the step-wise optimization, normalized by the peak value, as the network size varies, while the number of NFs is set to 1. We observe an increasing trend in the provisioning cost for both algorithms due to the increase in the number of path lengths for the service request. We also observe an approximately constant performance gap between the model-assisted algorithm and the step-wise benchmark since the number of NFs does not vary for the same service request.

Fig. 7-(b) shows the average normalized provisioning cost of the trained agent and the step-wise optimization as the number of NFs varies. Here, the gap between the model-assisted algorithm and the step-wise optimization widens as the number of NFs increases. This is because the event of setting up and tearing down NF instances becomes more frequent for the step-wise optimization as the number of NFs increases, thereby the inefficiency of the step-wise benchmark becomes more pronounced.

VII. Conclusions

In this paper, we develop a deep RL based dynamic provisioning mechanism for NFV-enabled services. We take into consideration the transmission and processing resources, the NF setup cost, and the routing overhead. The deep RL approach relies on an actor-critic architecture with DDPG algorithm. However, incorporating such considerations renders the vanilla DDPG algorithm incapable of consistently achieving the desired behavior due to its notorious sensitivity to hyperparameters. Therefore, to aid with the exploration process and to speed up the convergence of the learning algorithm, we propose to leverage and integrate domain-based knowledge obtained from a step-wise formulation with the deep RL algorithm. Numerical results demonstrate that the deep RL approach outperforms the step-wise formulation, which is expected since deep RL algorithms find ways to optimize the long-run objective. Moreover, results show the modified DDPG algorithm reduces the sensitivity and brittleness observed in the vanilla DDPG algorithm.

Acknowledgments

This work was supported by research grants from Huawei Technologies Canada and from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] S. Mehraghdam and H. Karl, "Placement of services with flexible structures specified by a YANG data model," in *Proc. IEEE NetSoft*, 2016, pp. 184–192.

[2] M. T. Beck and J. F. Botero, "Coordinated allocation of service function chains," in *Proc. IEEE GLOBECOM*, 2015, pp. 1–6.

[3] Z. Ye, X. Cao, J. Wang, H. Yu, and C. Qiao, "Joint topology design and mapping of service function chains for efficient, scalable, and reliable network functions virtualization," *IEEE Netw.*, vol. 30, no. 3, pp. 81–87, 2016.

[4] O. Alhussein, P. T. Do, Q. Ye, J. Li, W. Shi, W. Zhuang, X. Shen, X. Li, and J. Rao, "A virtual network customization framework for multicast services in NFV-enabled core networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1025–1039, 2020.

[5] C. Chekuri, S. Khanna, and F. B. Shepherd, "The all-or-nothing multicommodity flow problem," in *Proc. ACM STOC*, 2004, pp. 156–165.

[6] P. Marbach, O. Mihatsch, and J. N. Tsitsiklis, "Call admission control and routing in integrated services networks using neuro-dynamic programming," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 2, pp. 197–208, Feb 2000.

[7] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. IEEE INFOCOM*, Apr. 2018, pp. 1871–1879.

[8] Y. V. Kiran, T. Venkatesh, and C. S. Ram Murthy, "A reinforcement learning framework for path selection and wavelength selection in optical burst switched networks," *IEEE J. Sel. Areas Commun.*, vol. 25, no. 9, pp. 18–26, Dec. 2007.

[9] X. Huang, T. Yuan, G. Qiao, and Y. Ren, “Deep reinforcement learning for multimedia traffic control in software defined networking,” *IEEE Netw.*, vol. 32, no. 6, pp. 35–41, Nov. 2018.

[10] S. Mehraghdam, M. Keller, and H. Karl, “Specifying and placing chains of virtual network functions,” in *Proc. IEEE CloudNet*, 2014. doi: 10.1109/CloudNet.2014.6968961. ISBN 978-1-4799-2730-2 pp. 7–13.

[11] J. Li, W. Shi, Q. Ye, W. Zhuang, X. Shen, and X. Li, “Online joint VNF chain composition and embedding for 5G networks,” in *Proc. IEEE GLOBECOM*, 2018, pp. 1–6.

[12] J. Cao, Y. Zhang, W. An, X. Chen, J. Sun, and Y. Han, “VNF-FG design and VNF placement for 5G mobile networks,” *Science China Information Sciences*, vol. 60, no. 4, p. 040302, 2017.

[13] M. Jalalitabar, E. Guler, D. Zheng, G. Luo, L. Tian, and X. Cao, “Embedding dependence-aware service function chains,” *IEEE/OSA J. Optical Commun. Netw.*, vol. 10, no. 8, pp. 64–74, 2018.

[14] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, “Experience-driven networking: A deep reinforcement learning based approach,” in *Proc. IEEE INFOCOM*, 2018, pp. 1871–1879.

[15] J. Chen, P. Yang, Q. Ye, W. Zhuang, X. Shen, and X. Li, “Learning-based proactive resource allocation for delay-sensitive packet transmission,” *IEEE Trans. Cognitive Commun. Netw.*, vol. 7, no. 2, pp. 675–688, 2021.

[16] L. Gu, D. Zeng, W. Li, S. Guo, A. Y. Zomaya, and H. Jin, “Intelligent VNF orchestration and flow scheduling via model-assisted deep reinforcement learning,” *IEEE J. Selected Areas Commun.*, pp. 1–1, 2019.

[17] S. Troia, R. Alvizu, and G. Maier, “Reinforcement learning for service function chain reconfiguration in NFV-SDN metro-core optical networks,” *IEEE Access*, vol. 7, pp. 167 944–167 957, 2019.

[18] X. Fu, F. R. Yu, J. Wang, Q. Qi, and J. Liao, “Service function chain embedding for NFV-enabled IoT based on deep reinforcement learning,” *IEEE Communications Magazine*, vol. 57, no. 11, pp. 102–108, 2019.

[19] S. Yan, Q. Ye, and W. Zhuang, “Learning-based transmission protocol customization for VoD streaming in cybertwin-enabled next-generation core networks,” *IEEE Internet Things J.*, vol. 8, no. 22, pp. 16 326–16 336, 2021.

[20] K. Qu, W. Zhuang, X. Shen, X. Li, and J. Rao, “Dynamic resource scaling for VNF over nonstationary traffic: A learning approach,” *IEEE Trans. Cognitive Commun. Netw.*, pp. 1–1, 2020.

[21] J. Li, W. Shi, N. Zhang, and X. Shen, “Delay-aware VNF scheduling: A reinforcement learning approach with variable action set,” *IEEE Trans. Cognitive Commun. Netw.*, vol. 7, no. 1, pp. 304–318, 2021.

[22] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, “Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks,” *IEEE J. Sel. Areas Commun.*, pp. 1–1, 2019.

[23] S. Q. Zhang, Q. Zhang, A. Tizghadam, B. Park, H. Bannazadeh, R. Boutaba, and A. Leon-Garcia, “Sector: TCAM space aware routing on SDN,” in *Proc. ITC*, vol. 1, 2016, pp. 216–224.

[24] S. H. Strogatz, “Exploring complex networks,” *Nature*, vol. 410, no. 6825, pp. 268–276, 2001.

[25] R. Albert and A.-L. Barabási, “Statistical mechanics of complex networks,” *Rev. Mod. Phys.*, vol. 74, pp. 47–97, 2002.

[26] O. Alhussein and W. Zhuang, “Robust online composition, routing and NF placement for NFV-enabled services,” *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1089–1101, 2020.

[27] W. Mou, Z. Wen, and X. Chen, “On the sample complexity of reinforcement learning with policy space generalization,” *CoRR*, vol. abs/2008.07353, 2020.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *CoRR*, vol. abs/1312.5602, 2013.

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *CoRR*, vol. abs/1509.02971, 2015.

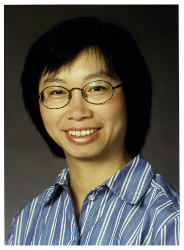
[30] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, “Parameter space noise for exploration,” *CoRR*, vol. abs/1706.01905, 2017.

[31] M. W. Group, “Packet traces from wide backbone,” 2019. [Online]. Available: http://mawi.wide.ad.jp/mawi/

[32] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep reinforcement learning that matters,” *CoRR*, vol. abs/1709.06560, 2017.

[33] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, “Benchmarking deep reinforcement learning for continuous control,” in *Proc. ICML*, 2016, pp. 1329–1338.

Omar Alhussein is currently a Senior Research Engineer with the Advanced Networking Team at Huawei Technologies Canada. He received the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Canada, in 2020, the M.A.Sc. degree in engineering science from Simon Fraser University, Canada, in 2015, and the B.Sc. degree in communications engineering from Khalifa University, United Arab Emirates, in 2013. His research interests include network resource management and optimization, AI and networking, NFV/SDN, and wireless networks.



Weihua Zhuang (Fellow, IEEE) received the Ph.D. degree from the University of New Brunswick, Canada, in electrical engineering. She has been with the Department of Electrical and Computer Engineering, University of Waterloo, Canada, since 1993, where she is a University Professor and a Tier I Canada Research Chair in Wireless Communication Networks. Dr. Zhuang is the recipient of 2021 Women’s Distinguished Career Award from IEEE Vehicular Technology Society, 2021 Technical Contribution Award in Cognitive Networks from IEEE Communications Society, and 2021 R.A. Fessenden Award from IEEE Canada. She was the Editor-in-Chief of the IEEE Transactions on Vehicular Technology from 2007 to 2013, Technical Program Chair/Co-Chair of IEEE VTC 2017/2016 Fall, Technical Program Symposia Chair of IEEE Globecom 2011, and an IEEE Communications Society Distinguished Lecturer from 2008 to 2011. She is an elected member of the Board of Governors and Executive Vice President of the IEEE Vehicular Technology Society. Dr. Zhuang is a Fellow of the Royal Society of Canada, Canadian Academy of Engineering, and Engineering Institute of Canada.