Service Function Chain Placement in Cloud Data Center Networks: a Cooperative Multi-Agent Reinforcement Learning Approach

Lynn Gao[1], Yutian Chen[2], and Bin Tang[3]

[1] Data Science, University of California Irvine, CA 92697, USA
Lmgao@uci.edu
[2] Economics Department, California State University Long Beach, CA 90840, USA
Yutian.Chen@csulb.edu
[3] Computer Science Department, California State University Dominguez Hills, Carson, CA 90747, USA
btang@csudh.edu

Abstract. Service function chaining (SFC), consisting of a sequence of virtual network functions (VNFs) (i.e., firewalls and load balancers), is an effective service provision technique in modern data center networks. By requiring cloud user traffic to traverse the VNFs in order, SFC improves the security and performance of the cloud user applications. In this paper, we study how to place an SFC inside a data center to minimize the network traffic of the virtual machine (VM) communication. We take a cooperative multi-agent reinforcement learning approach, wherein multiple agents collaboratively figure out the traffic-efficient route for the VM communication.

Underlying the SFC placement is a fundamental graph-theoretical problem called the k-stroll problem. Given a weighted graph $G(V, E)$, two nodes s, $t \in V$, and an integer k, the k-stroll problem is to find the shortest path from s to t that visits at least k other nodes in the graph. Our work is the first to take a multi-agent learning approach to solve k-stroll problem. We compare our learning algorithm with an optimal and exhaustive algorithm and an existing dynamic programming(DP)-based heuristic algorithm. We show that our learning algorithm, although lacking the complete knowledge of the network assumed by existing research, delivers comparable or even better VM communication time while taking two orders of magnitude of less execution time.

Keywords: Service Function Chaining · Data Centers · Reinforcement Learning · k-stroll Problem

1 Introduction

Background and Motivation. Middleboxes (MBs) [9], also known as "network appliances" or "network functions", are network devices that inspect, filter,

or transform network traffic for purposes of improving network security or performance. Typical examples of MBs include firewalls, intrusion detection systems, load balancers, and network address translators. MBs are an important component of modern enterprise networks to deliver services to user traffic - it has been shown that the number of MBs is comparable with the number of routers in enterprise networks and data centers [33].

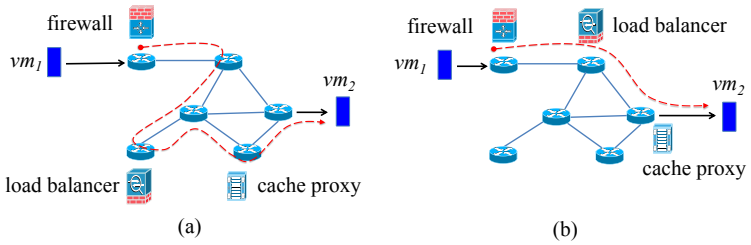


Fig. 1: Illustrating SFC placement in cloud data centers. The SFC consists of three VNFs viz. a firewall, a load balancer, and a cache proxy. The SFC placement in (b) is more network-efficient than the one in (a).

In recent years, with the advancement of *network function virtualization* (NFV) [3], MBs can now be implemented as virtual network functions (VNFs) running as virtual machines (VMs) [30] or containers [34] on commodity hardware platforms. Compared to MBs, which are mostly proprietary and dedicated purpose-built hardware devices, VNFs are software implementations that are cost effective and flexible for deployment. As such, deploying VNFs has become an effective technique in cloud data centers to achieve flexible service management and reduce capital and operational expenditures.

In particular, *service function chaining (SFC)* is established in cloud data centers to require virtual machine (VM) cloud traffic to traverse a chain of VNFs [23, 27, 43]. Fig. 1(a) shows an example of SFC, wherein cloud traffic generated at VM vm_1 traverses a sequence of VNFs including a firewall, a load balancer, and a cache proxy, to arrive at VM vm_2. With this traversal, the SFC blocks malicious traffic detected by the firewall, then diverts the credible VM traffic using the load balancer to avoid network congestion, and finally caches the network packets at the proxy server for quick data access by other cloud users.

SFC Placement Problem. As cloud network resources such as bandwidth and energy are limited in a cloud data center network, one important task for cloud operators is to install the VNFs at the right locations inside the network to optimize the cloud traffic or user-perceived VM communication delay. In this paper, we study the *SFC placement problem*. Given a source and a destination of a VM communication flow inside a cloud data center, and an SFC consisting of k VNFs of different service functions, it studies where to install the k VNFs

inside the network to minimize the communication traffic or delay of the VM flow. In fact, the SFC placement in Fig. 1(a) is not traffic- or delay-optimal for the communicating VM pair (vm_1, vm_2). By placing three VNFs as shown in Fig. 1(b), the resulting cloud traffic and user-perceived delay are dramatically reduced, as now the traffic only traverses three switches (with two network hops) compared to six switches (with six network hops) in Fig. 1(a).

State-of-the-Art. Tran et al. [37] showed that the SFC placement problem is equivalent to a fundamental graph-theoretical problem called the k-stroll problem [6, 10]. Given a weighted graph $G(V, E)$ and two nodes s, $t \in V$, and an integer k, k-stroll problem is to find a shortest path from s to t that visits at least k other nodes in the graph. As k-stroll is NP-hard, Chaudhuri et al. [10] presented a primal-dual-based $2 + \epsilon$ approximation algorithm. That is, it yields a solution with cost that is at most $2 + \epsilon$ times of the optimal cost. As the primal-dual algorithm is complicated and difficult to implement for a large-scale cloud data center, Tran et al. [37] designed a dynamic programming (DP)-based heuristic algorithm to solve the SFC. They showed that their approach constantly outperforms the performance guarantee of $2 + \epsilon$ provided by Chaudhuri et al. [10]. However, the DP algorithm needs the full knowledge of the network and a complete graph of the network as its input.

Our Contributions. In this paper, we design a cooperative multi-agent reinforcement learning (MARL) algorithm to solve the SFC placement problem. Reinforcement learning (RL) [35] refers to the use of autonomous agents to learn to perform a task by trial and error without human intervention. Unlike traditional computer algorithms, RL executes through the iterative interaction of the agent with the environment to learn about the environment, thus being more adaptive and robust to the dynamic network environment. In addition, as many network-related combinatorial problems are NP-hard and it is time-consuming to find the exact solutions, RL becomes a time-efficient alternative to solve these problems. As such, RL has been well utilized to solve network-related combinatorial optimization problems [2, 29].

However, none of the existing research utilizes RL to solve the k-stroll problem. We show that our MARL algorithm, without knowing the complete graph of the data center network assumed by existing research, delivers comparable or even better VM communication time than existing research, while taking two orders of magnitude less in execution time.

Two characteristics of the k-stroll problem make RL a particularly good candidate to solve the problem. First, to find a shortest route from source s to destination t while visiting at least k other nodes, an agent needs to constantly make decisions along the way. Such multi-step decision making is exactly the kind of problem that RL is designed to solve. Second, the goal of k-stroll is to minimize the travel cost of the agent that constantly makes progress, which corresponds well to the goal of RL of maximizing the cumulative reward.

Paper Organization. The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 formulates the SFC placement problem. In

Section 4 we propose our MARL algorithm. Section 5 presents the existing DP algorithm as well as an optimal algorithm for SFC placement. Section 6 compares all the algorithms and discusses the results. Section 7 concludes the paper.

2 Related Work

In this section, we first review SFC placement research in general and then SFC placement research using machine learning in particular, to illustrate the contributions of our work.

SFC Placement Research. There has been extensive research for SFC placement, also referred to as VNF placement [25]. Bari et al. [5] studied a VNF orchestration problem that determined the required number and placement of VNFs to optimize network operational costs. It provided an Integer Linear Programming (ILP) solution and an efficient greedy algorithm. Bhamare et al. [7] studied the VNF placement problem that minimized inter-cloud traffic while satisfying deployment cost constraints. They mainly used queueing theories and statistical analysis, which are different from our graph-theoretical approach. Feng et al. [15] studied the NFV service distribution to minimize the overall cloud network resource cost. They formulated a multi-commodity-chain flow problem and provided a $O(\epsilon)$ fast approximation algorithm. Huin et al. [22] studied the SFC placement problem to avoid data passing through unnecessary network devices. Sang et al. [32] and Chen et al. [12] minimized the cost of VNF deployment that provides services to flows and designed optimal and approximate algorithms under different network topologies. However, they did not consider the chain-order sequence of VNFs required in SFCs. Ma et al. [28] considered the traffic changing effect of VNFs and studied the SFC deployment problems with the goal to load-balance the network. Flores et al. [16] studied policy-aware VM migration and placement problem to mitigate the dynamic network traffic in cloud data centers considering that an SFC has already been placed in the network.

There are other works that studied the admission control aspect of SFC placement when not all the user requests can be satisfied due to network resource constraints. They studied how to maximize the total utility [24] and throughput [41] of the satisfied requests, or to maximize the difference between the service provider's profit and the total deployment cost of VNFs [13]. Yang et al. [40] studied how to place VNFs on the edge and public clouds such that the maximum link load ratio is minimized and each user's requested delay is satisfied. Gu et al. [19] designed a dynamic market auction mechanism for the transaction of VNF service chains that achieves near-optimal social welfare in the NFV eco-system.

Unlike most of the above work, we take a graph-theoretical approach and model the SFC placement as a graph-theoretical problem. We uncover that the problem is equivalent to the k-stroll problem. Although k-stroll problem has been studied extensively in the theory community, it has not been studied in a concrete network context targeting a specific network problem such as the SFC problem in cloud data centers. Due to such theoretical root of our research, the

techniques developed in this paper could be utilized in any other network context where the k-stroll model is relevant and applicable.

SFC Placement Research Using Machine Learning. Recently, machine learning techniques have been utilized to solve SFC placement problems. For example, some works employed machine learning to estimate upcoming traffic rates and to adjust VNF deployment [8, 21, 14, 42, 36]. Xiao et al. [39] considered an online SFC placement problem with unpredictable real-time network variations and various service requests and introduced a Markov decision process (MDP) model to capture the dynamic network state transitions. They proposed a deep reinforcement learning (DRL) approach to jointly optimize the operation cost of NFV providers and the total throughput of requests. Using DRL techniques, Pei et al. [31] aimed to minimize the weighted costs of VNF placement cost, VNFI running cost, and penalty of rejected user requests. They proposed a Double Deep Q Network (DDQN)-based optimal solution that places or releases VNF Instances following a threshold-based policy. Recently, Wang et al. [38] extended the above DDQN approach to solving an online fault-tolerant SFC Placement.

All above works adopted the DRL approach. Utilizing neural network-based function approximation algorithms, DRL is a powerful technique that is able to handle complex states and decision-making for agents. As such, DRL is both time- and resource-consuming. Further, unlike DRL that must learn from existing data to train algorithms to find patterns, and then use that to make predictions about new data, RL uses feedback (i.e., rewards) from interacting with the environment to maximize an agent's cumulative reward. Thus the RL model is better suited to solve our SFC placement problem. Besides, as k-stroll problem has a low-dimensional and discrete setting in terms of agent's states and actions, RL is sufficient to solve the SFC placement. We show that the our RL-based solution is competitive to the optimal solution.

Our work is inspired by Ant-Q [17], a family of algorithms that combines RL algorithms with the observation of ant colony behaviors. Ant-Q designed a Q-learning-based algorithm [35] to solve the traveling salesman problem. However, like [37], it assumed that the input is a pre-processed complete graph of the studied network. Our work does not have this assumption and our designed algorithm directly works on any network topologies such as fat-tree data center networks studied in this paper.

3 Problem Formulation

System Model. We model a data center network as an undirected graph $G(V, E)$ where $V = V_h \cup V_s$ includes a set of hosts $V_h = \{h_1, h_2, ..., h_{|V_h|}\}$ and a set of switches $V_s = \{s_1, s_2, ..., s_{|V_s|}\}$. E is a set of edges, each connecting either one switch to another or a switch to a host. Fig. 2 shows a K=4 fat-tree [4] data center where K is the number of ports per switch.[4]

[4] We use fat-trees for illustration purpose. However, the problems and solutions proposed in this paper are applicable to any data center topology.

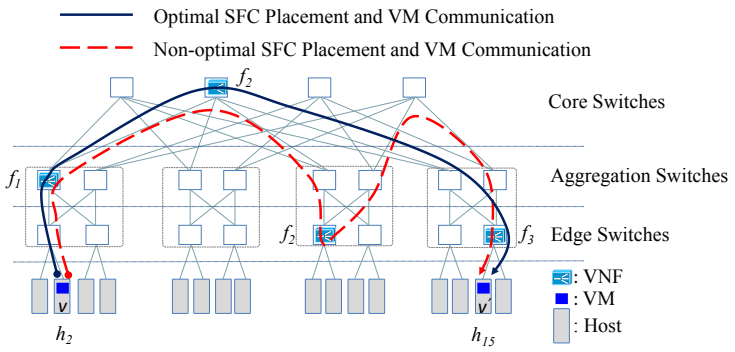


Fig. 2: A K-ary fat-tree data center with K=4. It has16 hosts: h_1, h_2, ..., and h_{16}, an SFC with 3 VNFs: f_1, f_2, and f_3, and a VM flow: (v, v'). One optimal and one non-optimal SFC placement and communication are shown in the blue solid line and the red dashed line, respectively.

There is an SFC consisting of n VNFs $\mathcal{F} = \{f_1, f_2, ..., f_n\}$, each of which needs to be installed (i.e., placed) on a different switch in the data center. Once the VNFs are installed, it requires that the VM traffic to go through the VNFs in the order of f_1, f_2, ..., and f_n. We refer to f_1 (and f_n) as the *ingress* (and *egress*) VNF, and the switch where the ingress (and egress) VNF is installed as the *ingress* (and *egress*) switch. There is one pair of communicating VMs (v, v') already in the data center, where v is located at host $s(v)$ and v' at $s(v')$. v and v' are referred to as the *source* and *destination VM*, and $s(v)$ and $s(v')$ as *source* and *destination host* respectively.

Each edge $(u, v) \in E$ has a weight $w(u, v)$, indicating either the network delay or energy cost on this edge caused by the VM communication. Given any host or switch u and v, let $c(u, v)$ denote the total cost of all the edges traversed by VM communication from u to v. Thus the *communication cost* of the VM flow (v, v') is $c\big(s(v), s(v')\big)$. Note that $c\big(s(v), s(v')\big)$ is not necessarily the cost of the shortest path between $s(v)$ and $s(v')$, as the VM communication must traverse a sequence of VNFs.

Fig. 2 shows the VM flow (v, v'), where v is located at host h_2 and v' at h_{15}, and one SFC consisting of three VNFs f_1, f_2 and f_3. The blue solid line shows one optimal SFC placement for this VM flow, which results in 6 hops of VM communication between v and v'. However, if we instead place f_2 at one of the edge switches, the VM communication between v and v' becomes 10 hops, as shown in the red dashed line. Here, we use unweighted costs (i.e., number of edges) only for purpose of illustration, as the problem and its solution target weighted graphs. Table 1 shows all the notations.

Table 1: Notation Summary

Notation	Description
$G(V, E)$	A data center graph, where $V = V_h \cup V_s$
$w(u, v)$	Weight of an edge $(u, v) \in E$
$c(u, v)$	Cost between hosts (or switches) $u, v \in V$
V_h	$V_h = \{h_1, h_2, ..., h_{\lvert V_h \rvert}\}$ is the set of $\lvert V_h \rvert$ hosts
V_s	$V_s = \{s_1, s_2, ..., s_{\lvert V_s \rvert}\}$ is the set of $\lvert V_s \rvert$ switches
$\mathcal{F}$	$\mathcal{F} = \{f_1, f_2, ..., f_k\}$ is the set of k VNFs of an SFC
(v, v')	The source and destination communicating VMs
$s(v)$	The source host where source VM v is stored
$s(v')$	The destination host where destination VM v' is stored
K	K-ary fat-tree data center wherein each switch has K ports
$p(j)$	SFC placement function p, f_j is placed at switch $p(j) \in V_s$
$C_c(p)$	Total VM communication cost with VNF placement p
α	The learning rate of each agent, $0 \leq \alpha \leq 1$
γ	The discount rate of each agent, $0 \leq \gamma \leq 1$
δ, β	Parameters weighing the relative importance of the Q-value and the edge length in the agent's action selection rule

Problem Formulation We define a *SFC placement function* as $p : \mathcal{F} \rightarrow V_s$, which places VNF $f_j \in \mathcal{F}$ at switch $p(j) \in V_s$. Given any SFC placement p, denote the communication cost of VM flow v, v' under p as $C_c(p)$. Therefore,

$$C_c(p) = \sum_{j=1}^{n-1} c\big(p(j), p(j+1)\big) + \Big(c\big(s(v_i), p(1)\big) + c\big(p(n), s(v_i')\big)\Big). \tag{1}$$

Note that for (v, v'), the ingress switch is always $p(1)$ and the egress switch is always $p(n)$. The objective of the SFC placement problem is to find a p to minimize $C_c(p)$.

k-stroll problem. Previous work [37] has shown that the SFC placement problem is equivalent to *k-stroll problem* [10, 6], which is NP-hard. Given a weighted graph $G{=}(V, E)$ with nonnegative length w_e on edge $e \in E$, two special nodes s and t, and an integer k, the k-stroll problem finds an s-t path or walk (i.e., a stroll) of minimum length that visits at least k distinct nodes excluding s and t. When $s{=}t$, it is called the *k-tour problem*. The triangle inequality holds for all edges: for $(x, y), (y, z), (z, x) \in E$, $w(x, y) + w(y, z) \geq w(z, x)$.

Fig. 3(a) shows a graph of six nodes and six edges with the weight of each edge shown as well. An optimal 2-stroll between s and t is s, D, t, C, and t, with a cost of 6. While most of the works solving k-stroll problem assume the graph is non-complete graph [18, 10, 11], two works [6, 37] assume its input is a complete graph. In particular, Tran el al. [37] showed that by converting the data center graph into a complete graph, it is able to design an efficient dynamic programming (DP) based heuristic. In this paper, however, we show that our reinforcement learning-based algorithm can relax this assumption while

still achieving comparable or even better performance compared to the DP-based algorithm. Next we propose a multi-agent reinforcement learning algorithm to solve the SFC placement problem.

4 Multi-Agent Reinforcement Learning Algorithm for SFC Placement

In this section, we first present the basics of RL and then our cooperative multi-agent reinforcement learning (MARL) framework for SFC placement.

Reinforcement Learning (RL). In a RL system [35], an agent's decision making is described by a 4-tuple (S, A, t, r) wherein,

- S is a finite set of states,
- A is a finite set of actions,
- $t : S \times A \to S$ is a state transition function, and
- $r : S \times A \to R$ is a reward function, where R is a real value reward.

That is, at a specific state $s \in S$, the agent takes an action $a \in A$ to transition to state $t(s, a) \in S$ while receiving a reward $r(s, a) \in R$. The agent maintains a policy $\pi(s) : S \to A$ that maps its current state $s \in S$ into the desirable action $a \in A$. We consider a deterministic policy where given the state, the policy outputs a specific action for the agent to take to go to the next step. Deterministic policy suites the SFC placement well as an agent always attempts to make progress finding a k-stroll from host $s(v)$ to host $s(v')$; that is, it hopes to get closer to $s(v')$ each time it moves to the next node.

A widely used class of RL algorithms are value-based methods [35, 26]. These algorithms try to extract the near-optimal policy based on the value function $V_s^\pi = E\{\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t))|s_0 = s\}$, which is the expected value of a discounted future reward sum with the policy π at stage s. Here, γ $(1 \leq \gamma \leq 1)$ is a discounted rate and $r(s_t, \pi(s_t))$ is the reward received by the agent at state s_t at time slot t following policy π.

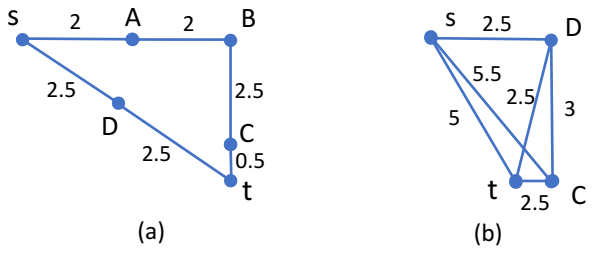


Fig. 3: (a) An example of k-stroll problem. (b) The complete graph for the DP-based Algo. 2 (only the relevant part for the DP computation is shown).

Q-Learning. Q-learning is a family of value-based algorithms [35]. It learns how to optimize the quality of the actions in terms of the Q-value $Q(s, a)$. $Q(s, a)$ is defined as the expected discounted sum of future rewards obtained by taking action a from state s following an optimal policy. The optimal action at any state is the action that gives the maximum Q-value. For an agent at state s, when it takes action a and transitions to the next state t, $Q(s, a)$ is updated as

$$Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot [r(s, a) + \gamma \cdot \max_b Q(t, b)], \tag{2}$$

where α and γ ($1 \leq \alpha, \gamma \leq 1$) are the learning rate and discount rate respectively. In Eqn. 2, $r(s, a)$ is the reward obtained if action a is taken at the current state s and $\max_b Q(t, b)$ is the maximum reward that can be obtained from the next state t.

Multi-agent Reinforcement Learning (MARL) Algorithm. In our MARL framework for SFC placement, there are multiple agents that all start from the source host $s(v)$. They work synchronously and cooperatively to learn the state-action Q-table and the reward table and take action accordingly in any of the states. In the context of the SFC placement, the *states* are the nodes (switches or hosts) where agents are located and *actions* are the nodes they move to next. The common task of all the agents is to learn and find a k-stroll: starting from the source host $s(v)$, each visiting at least k other switches, and ending at the destination host $s(v')$. The k distinct switches found in the k-stroll are where the k VNFs will be placed.

One important component of our MARL algorithm is the *action selection rule*; an agent follows such rule to select the node to move to during its k-stroll learning process. It combines exploration and exploitation of an agent; that is, an agent can reinforce the good evaluations it already knows as well as explore new actions. We define the action selection rule in SFC placement as below.

Definition 1. Action Selection Rule of SFC Placement. *The* **action selection rule** *specifies, for an agent located at node s, which node t it moves to next. When $q \leq q_0$, where q is a random value in [0, 1] and q_0 ($0 \leq q_0 \leq 1$) is a preset value, it always chooses the node $t = argmax_{u \in U}\{\frac{[Q(s,u)]^\delta}{[w(s,u)]^\beta}\}$ to move to (i.e., the exploitation). Here δ and β are parameters weighing the relative importance of the Q-value and the edge length while U is the set of nodes not visited yet by the agent. Otherwise, the agent chooses a node $t \in U$ to move to by the following distribution: $p(s, t) = \frac{[Q(s,t)]^\delta / [w(s,t)]^\beta}{\sum_{u \in U} [Q(s,u)]^\delta / [w(s,u)]^\beta}$ (i.e., the exploration).* □

In the above action selection rule, by exploitation, an agent, located at node s, always moves to a node t that maximizes the learned Q-value weighted by the length of the edge (s, t). By exploration, it chooses the next node to move to according to the distribution $p(s, t)$, which characterizes how good the nodes are in terms of learned Q-values and the edge lengths. The higher the Q-value and the shorter the edge length, the more desirable the node to move to.

The above action rule is based on ϵ-greedy exploration [35], wherein an agent selects a random action with probability ϵ and selects the best action, which

corresponds to the highest Q-value, with probability $1-\epsilon$. Moreover, our action rule augments ϵ-greedy exploration by taking into account specific features of cloud data center networks (i.e., edge lengths).

MARL Algorithm. Next, we present our MARL algorithm viz. Algo. 1. There are m agents initially located at the source host $s(v)$ (line 1). Their k-stroll learning takes place in iterations. Each iteration consists of two stages.

The first stage consists of k steps (lines 3-18). In each step, each agent independently takes actions following Definition 1 to move to the next node. At the end of each step, each updates the Q-value of the involved edge. This continues until each agent finds its k-stroll and arrives at destination host $s(v')$. Following [17], each agent maintains a list of visited switches in its memory, so that it knows how to select an unvisited switch to visit while visiting at least k switches before arriving at $s(v')$.

In the second stage (line 19-23), it finds among the m k-strolls the one with the smallest length, and updates the reward value of the edges that belong to this shortest k-stroll as well as the Q-values according to Eqn. 2. Finally, it checks if the termination condition is met. If not, it goes to the next iteration and repeats the above two stages. Here, the termination condition is either a specified number of iterations or within some proximity to the costs of the compared DP and optimal algorithms.

Algorithm 1 MARL Algorithm for SFC Placement.
Input: A data center graph $G(V = V_s \cup V_h, E)$, $s(v_1)$, $s(v_1')$, and an SFC $(f_1, f_2, ..., f_n)$.
Output: A k-stroll from $s(v_1)$ to $s(v_1')$; that is, a switch $p(j) \in V_s$ to place each of the k VNFs $f_j \in \mathcal{F}$ and the cost $C_c(p))$ of the k-stroll.
Notations: i: index for switches; j: index for agents;
U_j: the set of nodes unvisited by agent j, initially $U_j = V_s$, the set of switches;
L_j: the path taken by agent j, initially empty;
l_j: the length of L_j, initially zero;
r_j: the node where agent j is located currently;
$Q(u, v)$: Q-value of edge (u, v), initially $\frac{|E|}{|V| \cdot \sum_{(u,v) \in E} w(u,v)}$;
p: an array storing the distinct switches on $s(v_1)$-$s(v_1')$ stroll;
α: learning rate, $\alpha = 0.1$;
γ: discount factor, $\gamma = 0.3$;
W: a constant value of 10 following [17];

1. Initially all the m agents are at host $s(v)$, i.e., $r_j = s(v), 1 \leq j \leq m$;
2. **while** (termination condition is not met)
3. **for** $(i = 1; i <= k; i{+}{+})$ // Finding the k switches to place VNFs
4. **for** $(j = 1; j \leq m; j{+}{+})$ // Agent j
5. Agent j decides the next node s_j to move to following action rule in Definition 1;
6. $L_j = L_j \cup \{s_j\}$;

$l_j = l_j + w(r_j, s_j)$;
$Q(r_j, s_j) = (1-\alpha) \cdot Q(r_j, s_j) + \alpha \cdot \gamma \cdot \max_{z \in U_j} Q(s_j, z)$; // Q-value
$r_j = s_j$; // Agent j moves to switch s_j;
$U_j = U_j - \{s_j\}$; // Switches not yet visited by agent j
end for;
end for;
for $(j = 1; j \leq m; j{+}{+})$ // Agent j ends at destination host $s(v')$
$L_j = L_j \cup \{s(v')\}$;
$l_j = l_j + w(r_j, s(v'))$;
$Q(r_j, s_j) = (1-\alpha) \cdot Q(r_j, s_j) + \alpha \cdot \gamma \cdot \max_{z \in U_k} Q(s_j, z)$; // Q-value
$r_j = s(v')$;
end for;
Let $j* = \text{argmin}_{1 \leq j \leq m} l_j$ be the agent with a k-stroll of smallest length;
for (each edge $(u, v) \in L_{j*}$)
$r(u, v) = \frac{W}{l_{j*}}$; // Update reward value $r(u, v)$;
$Q(u, v) \leftarrow (1-\alpha) \cdot Q(u, v) + \alpha \cdot [r(u, v) + \gamma \cdot \max_b Q(v, b)]$; // Q-value
end for;
end while;
RETURN The switch $p(j) \in V_s$ to place VNF $f_j \in \mathcal{F}$ and the cost $C_c(p)$.

Discussion. In each iteration, the first stage takes $m \cdot k$, the second stage takes $m + k$. Assume N iterations take place, then the time complexity of Algo. 1 is $O(N \cdot m \cdot k)$. A key question for Algo. 1 is whether it is convergent (i.e., it is able to find the global optimum k-stroll in finite time). Gutjahr [20] gave a graph-based general framework to study convergences of ant systems. It shows that under certain conditions, the solutions generated can converge with a high probability to be arbitrarily close to the optimal solution for a given problem instance. However, as it is a general framework, it does not tackle specific combinatorial problems including the k-stroll. Considering the simple definition and elegant discrete structure inherent in k-stroll problem, studying the convergence as well as estimating the theoretical speed of convergence of applying RL to solve the k-stroll problem is promising future research.

5 Existing Algorithms for SFC Placement

We compare our MARL algorithm with existing work and a naive exhaustive optimal algorithm. We present them below to be self-contained for the paper.

Dynamic Programming (DP) Algorithm. Tran el al. [37] designed a DP-based heuristic algorithm viz. Algo. 2 to solve SFC placement. It is based on the observation that although finding the shortest stroll visiting k distinct nodes is NP-hard, finding the shortest stroll of k edges can be solved optimally and efficiently using DP.

Algo. 2 takes input a complete graph $G'(V', E')$ converted from the data center graph $G(V, E)$ as follows. $V' = \{s(v), s(v')\} \cup V_s$; for an edge $(u, v) \in E'$,

its cost $c_{(u,v)}$ is $c(u, v)$, the communication cost of (v, v') between u and v in G. Algo. 2 finds a shortest $s(v)$-$s(v')$ stroll with $k+1$ edges (lines 4-10) and checks if it traverses k distinct switches (lines 11-19). If not, it finds a stroll with $k+2$ edges, so on and so forth, until k distinct switches are found (lines 20-21). It finally places f_1, ..., f_k on the first k switches and returns the cost of the k-stroll (lines 23-24). Its time complexity is $O(k \cdot |V|^4)$. Note that Algo. 2 also works for k-tour problem where $s(v)$=$s(v')$ and the special case that k distinct switches are already on the shortest path between $s(v)$ and $s(v')$.

Algorithm 2 A DP Algorithm for SFC Placement Problem.
Input: A complete graph $G'(V', E')$, $s(v)$, $s(v')$, and an SFC $(f_1, f_2, ..., f_k)$.
Output: cost of an $s(v)$-$s(v')$ stroll in G' visiting at least k distinct switches.
Notations: e: index for edges; i: index for switches;
$c(u, s(v'), e)$: cost of a u-$s(v')$ stroll with e edges, initially $+\infty$;
$successor(u, s(v'), e)$: u's successor in a u-$s(v')$ stroll with e edges, initially -1 ;
r: number of edges needed on $s(v)$-$s(v')$ stroll, initially $k+1$;
p: an array storing distinct switches on $s(v)$-$s(v')$ stroll;
num: the number of distinct switches in p;
$found$: true if it has found a $s(v)$-$s(v')$ stroll with at least k distinct switches, initially false;

1. $V' = \{u_1, ..., u_{|V'|}\}$, let $u_a = s(v)$ and $u_{|V'|} = s(v')$;
2. $\forall u_i, u_j \in V'$ with $i \neq j$, $c(u_i, u_j, 1) = c_{u_i,u_j}$, $successor(u_i, u_j, 1) = u_j$, $successor(u_j, u_i, 1) = u_i$; $\forall u_i \in V'$, $c(u_i, u_i, 1) = +\infty$, $successor(u_i, u_i, 1) = -1$;
3. **while** $(\neg found)$
4. **for** $(e = 2; e <= r; e{+}{+})$ // edges in u_i-$s(v')$ stroll
5. **for** $(i = 1; i \leq |V'| - 1; i{+}{+})$ // node u_i
6. **for** $\big($each u, $u \neq u_i \wedge u \neq s(v') \wedge u_i \neq successor(u, s(v'), e-1)\big)$
7. **if** $\big(c(u_i, s(v'), e) > c_{u_i,u} + c(u, s(v'), e-1)\big)$
8. $c(u_i, s(v'), e) = c_{u_i,u} + c(u, s(v'), e-1)$;
9. $successor(u_i, s(v'), e) = u$;
10. **end if;**
11. $num = 0$; $p = \phi$ (empty set), $e{-}{-}$;
12. $b = successor(s(v), s(v'), e)$;
13. **while** $(e > 1)$
14. **if** $(b \neq s(v) \wedge b \neq s(v') \wedge b \notin p)$
15. $p[num] = b; num{+}{+}$;
16. **end if;**
17. $e - -$;
18. $b = successor(b, s(v'), e)$;
19. **end while;**
20. **if** $(num < k)$ $r{+}{+}$; // less than k distinct switches
21. **else** $found$ = true;
22. **end while;**
23. Place f_1, ..., f_k on the first k switches stored in p;
24. **RETURN** $c(s(v), s(v'), r)$.

Example 1. Fig. 3(b) shows the complete graph of Fig. 3(a) that is used for above DP computation. In this example, the Algo. 2 is able to find the optimal 2-stroll between s and t as s, D, t, C, and t. □

Optimal Algorithm. Below we present Algo. 3, an exhaustive algorithm that enumerates all the SFC placements and finds the one with minimum cost, thus solving the SFC placement problem optimally. It takes $O(|V|^k)$. Although it is not time-efficient, it can be implemented easily, and we compare it with other algorithms as a benchmark in small cases.

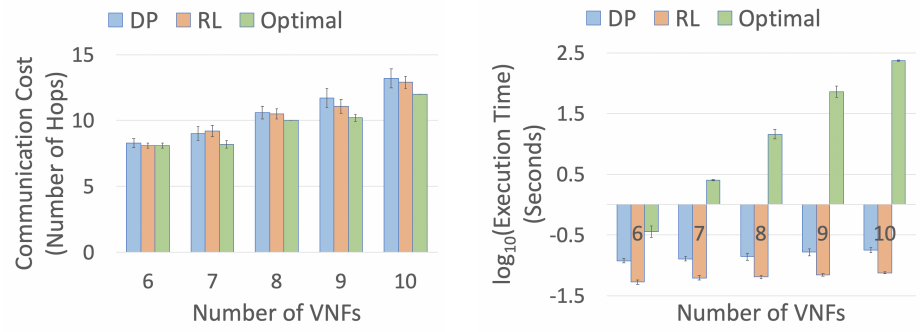
Algorithm 3 Exhaustive and Optimal SFC Placement.
Input: A data center graph $G(V, E)$, $s(v)$ and $s(v')$, and an SFC $(f_1, f_2, ..., f_k)$.
Output: A VNF placement p and the total cost $C_c(p)$.
1. $C_c(p) = +\infty$;
2. Among all $|V_s| \cdot (|V_s| - 1) \cdot ..., \cdot (|V_s| - k + 1)$ SFC placements, find p that gives the minimum cost $C_c(p)$;
3. **RETURN** p and $C_c(p)$.

6 Performance Evaluation

Experiment Setup. We compare our RL-based learning algorithm viz. Algo. 1 (referred to **RL**) with DP-based algorithm Algo. 2 (referred to as **DP**) and exhaustive optimal Algo. 3 (referred to as **Optimal**). We write our own simulator in Python on a MacBook Pro (Big Sur 11.5.1) with Intel Processor (2.7GHz Quad-Core Intel Core i7) and 16GB of memory. As Optimal takes a long time to execute, we first compare these three algorithms in small $K = 4$ fat-tree cloud data centers of 16 hosts. We then compare RL with DP in $K = 8$ data centers of 128 hosts. As RL is a multi-agent cooperative learning algorithm, we also investigate the effects of the number of agents m in a $K = 6$ fat-tree data center of 54 hosts. Unless otherwise mentioned, m is set as $\frac{K^3}{4}$, the number of hosts in the data center.

In the plots, each data point is an average of 20 runs with 95% confidence interval. For a fair comparison, in each run instance, the source VM v and destination VM v' are first randomly placed on the hosts then we compare the algorithms on the same VM placement. The SFCs in real-world cases are broadly categorized into two types viz. access SFCs and application SFCs [1]. As it shows that a typical SFC could have 5 to 6 access functions and 4 to 5 application functions, we consider up to 11 VNFs in an SFC.

Simulation Parameters. Following [17], we set all the RL related parameters as follows. The learning rate $\alpha = 0.1$, the discount factor $\gamma = 0.3$, the original Q-values for all the edges is $\frac{|E|}{|V| \cdot \sum_{(u,v) \in E} w(u,v)}$; that is, 1 divided by the multiplication of average distance between all nodes with number of nodes. For the



(a) Communication Cost. (b) Execution Time.

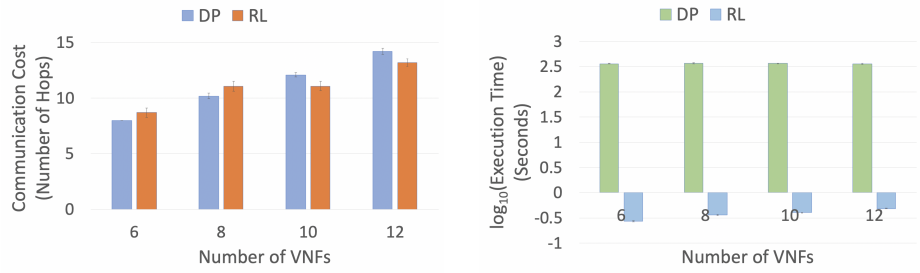
Fig. 4: Comparing RL, DP, and Optimal in $K = 4$ fat-tree data centers. Here, $m = 16$.

parameters used in action selection rule, $\delta = 1$ and $\beta = 2$. The constant number W used in Algo. 1 is set as 10.

Comparing RL, DP, and Optimal in $K = 4$ fat-trees. Fig. 4 compares all three algorithms in $K = 4$ fat-trees while varying the number of VNFs k. Fig. 4(a) compares the VM communication costs in terms of the number of hops yielded by all three algorithms. We have several observations. First, Optimal performs the best by giving the smallest communication cost. Second, RL performs better than DP in most cases although it does not have the complete knowledge of the data center network as DP does. This demonstrates that RL is indeed an effective SFC placement algorithm. Fig. 4 show the execution time of the three algorithms. While both RL and DP are time-efficient, incurring less than one second for all the cases, RL takes around half of the time compared to DP (note the logarithmic scale of the y values). In contrast, Optimal takes an enormous amount of time, in the order of hundreds of seconds when k gets large. This shows that our RL algorithm achieves comparable communication cost as the existing SFC placement algorithms while using smaller amount of execution time.

Comparing RL and DP in $K = 8$ fat-trees. Next, we compare RL and DP in a larger scale of $K = 8$ fat-tree data centers while varying the number of VNFs k from 6 to 12. Fig. 5(a) shows that DP performs better than RL when k is relatively small and RL performs better than DP when k gets large. However, Fig. 5(b) shows that our RL algorithm has a much smaller amount of execution time than DP (again, note the logarithmic scale of the y values). In particular, while RL takes less than one second to find the SFC placement, the DP takes hundreds of seconds to do so.

This is in sharp contrast to Fig. 4(b), which shows that DP only takes twice as much time as RL does. As the time complexity of DP is $O(k \cdot |V|^4)$ and $|V| = \frac{5}{4} \cdot K^2$, the number of switches in a K-ary fat-tree, the time complexity of DP in a K-ary fat-tree is thus $O(k \cdot K^8)$. On the other hand, the time complexity



(a) Communication Cost. (b) Communication Cost.

Fig. 5: Comparing RL and DP in $K = 8$ fat-tree data centers, $m = 128$ and $k = 10$.

of RL is $O(N \cdot m \cdot k) = O(N \cdot K^3 \cdot k)$, which is less dependent on the size of the fat-tree than DP is. The two orders of magnitude difference in execution times show that the RL algorithm is a promising technique for the SFC placement problem. Another reason why RL performs better can be attributed to the large number of agents (i.e., 128) that participate in the RL algorithm. As all the agents work cooperatively and synchronously to learn the state-action Q-table and make progress towards finding the k-stroll, the algorithm takes less time.

Effects of Number of Agents m in RL. Finally, we take a close look at the RL algorithm, and investigate the effects of the number of agents m on the performance of RL. We place an SFC of 10 VNFs (i.e., $k = 10$) in a $K = 6$ data center. Table. 2 shows that with the increase of m, the VM communication costs found by RL, the execution time, and the number of training iterations for RL all decrease. As more agents participate in the cooperative learning process, its time efficiency increases dramatically. We also observe that the VM communication cost seems to stabilize when m reaches 20. One possible reason could be that when $m = 20$, the RL is able to find the optimal VM communication cost.

7 Conclusions and Future Work

For data center operators, figuring out how to place SFC with different VNFs inside a data center network while minimizing the communication cost of VMs is a critical task. In this paper, we proposed a multi-agent reinforcement learning algorithm to solve the SFC placement problem. Although the SFC placement

Table 2: Varying number of agents m in RL. Here, $k = 10$ and $K = 6$.

Number of Agents m	1	5	10	15	20
Communication Cost (number of hops)	23.3	17.6	14.7	13.7	13.4
Execution Time (seconds)	65.54	27.72	9.07	2.70	0.22
Number of Iterations	169.6	75.05	22.25	10.7	3.1

problem has been solved extensively with and without resorting to machine learning techniques, our work has two novelties. First, we discovered that at the core of the SFC placement problem is the k-stroll problem, which is only studied in the theory community and has not been identified by any of the existing research on SFC placement. We hope our work can shed some light on how k-stroll can not only advance SFC placement research but also model an even wider range of network applications. Second, all the existing work utilizing machine learning techniques adopts a deep reinforcement learning approach. We showed that under k-stroll modeling of SFC placement, this is not necessary as there are a limited number of states and actions available for the agents. We designed a multi-agent reinforcement learning algorithm where multiple agents cooperatively and synchronously solve k-stroll in order to place SFC. We showed via simulations that our algorithm's performance is comparable with or even better than the existing algorithms, however, with execution time that is up to two orders of magnitude smaller than that of the existing work.

We have three research directions in the future. First, we will investigate the convergence and convergence speed of our RL algorithm by studying the unique discrete structure of k-stroll. For example, the existing DP-based algorithm provides some insights into how visiting k distinct nodes is related to visiting $k+1$ distinct edges, which can be solved optimally and efficiently. Second, in our current multi-agent scenario, all the agents cooperate with each other by sharing the same reward and updating the same Q-table. In a rational and game-theoretical environment wherein different agents have different goals and rewards, how agents balance cooperation with competition to find a near-optimal SFC placement efficiently remains largely unexplored. Third, currently, we only consider SFC placement for one VM flow. When there exist multiple VM flows with highly diverse and dynamic traffic rates (e.g., transmission rates and bandwidth demands), how to apply RL techniques to achieve optimal network traffic and communication delay in cloud data centers becomes a challenging problem.

Acknowledgment

This work was supported by NSF Grants CNS-1911191 and CNS-2131309.

References

1. Service function chaining use cases in data centers (ietf), https://tools.ietf.org/html/draft-ietf-sfc-dc-use-cases-06section-3.3.1
2. Reinforcement learning for combinatorial optimization: A survey. Computers & Operations Research **134** (2021)
3. et al., R.G.: Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper. In: SDN and OpenFlow World Congress (2012)
4. Al-Fares, M., Loukissas, A., Vahdat, A.: A scalable, commodity data center network architecture. SIGCOMM Comput. Commun. Rev. **38**(4), 63–74 (2008)

5. Bari, F., Chowdhury, S.R., Ahmed, R., Boutaba, R., Duarte, O.C.M.B.: Orchestrating virtualized network functions. IEEE Transactions on Network and Service Management **13**(4), 725–739 (2016)
6. Bateni, M., Chuzhoy, J.: Approximation algorithms for the directed k-tour and k-stroll problems. In: Proc. of APPROX/RANDOM 2010
7. Bhamare, D., Samaka, M., Erbad, A., Jain, R., Gupta, L., Chan, H.A.: Optimal virtual network function placement in multi-cloud service function chaining architecture. Comput. Comm. **102**, 1 – 16 (2017)
8. Bunyakitanon, M., Vasilakos, X., Nejabati, R., Simeonidou, D.: End-to-end performance-based autonomous vnf placement with adopted reinforcement learning. IEEE Transactions on Cognitive Communications and Networking **6**(2), 534–547 (2020)
9. Carpenter, B., Brim, S.: Middleboxes: Taxonomy and issues (2002), https://tools.ietf.org/html/rfc3234
10. Chaudhuri, K., Godfrey, B., Rao, S., Talwar, K.: Paths, trees, and minimum latency tours. In: Proc. of IEEE FOCS 2003
11. Chekuri, C., Korula, N., Pál, M.: Improved algorithms for orienteering and related problems. ACM Trans. on Algorithms **8**(3) (Jul 2012)
12. Chen, Y., Wu, J., Ji, B.: Virtual network function deployment in tree-structured networks. In: Proc. of ICNP 2018
13. Eramo, V., Miucci, E., Ammar, M., Lavacca, F.G.: An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures. IEEE/ACM Transactions on Networking **25**(4), 2008–2025 (2017)
14. Fei, X., Liu, F., Xu, H., Jin, H.: Adaptive vnf scaling and flow routing with proactive demand prediction. In: Proc. of IEEE INFOCOM 2018
15. Feng, H., L., J., Tulino, A.M., Raz, D., Molisch, A.F.: Approximation algorithms for the nfv service distribution problem. In: Proc. of IEEE INFOCOM 2017
16. Flores, H., Tran, V., Tang, B.: Pam & pal: Policy-aware virtual machine migration and placement in dynamic cloud data centers. In: Proc. of IEEE INFOCOM 2020
17. Gambardella, L., Dorigo, M.: Ant-q: A reinforcement learning approach to the traveling salesman problem. In: ICML (1995)
18. Garg, N.: Saving an ϵ: A 2-approximation for the k-mst problem in graphs. In: Proc. of ACM STOC 2005
19. Gu, S., Li, Z., Wu, C., Huang, C.: An efficient auction mechanism for service chains in the nfv market. In: Proc. of IEEE INFOCOM 2016
20. Gutjahr, W.: A graph-based ant system and its convergence. Future Generation Computer Systems **16**, 873–888 (2000)
21. Huang, X., Bian, S., Gao, X., Wu, W., Shao, Z., Yang, Y., Lui, J.: Online vnf chaining and predictive scheduling: Optimality and trade-offs. IEEE/ACM Transactions on Networking **29**(4), 1867–1880 (2021)
22. Huin, N., Jaumard, B., Giroire, F.: Optimal network service chain provisioning. IEEE/ACM Trans. on Netw. **26**(3), 1320–1333 (June 2018)
23. Joseph, D.A., Tavakoli, A., Stoica, I.: A policy-aware switching layer for data centers. In: Proc. of ACM SIGCOMM 2008
24. Kuo, T., Liou, B., Lin, K.C., Tsai, M.: Deploying chains of virtual network functions: On the relation between link and server usage. IEEE/ACM Transactions on Networking **26**(4), 1562–1576 (Aug 2018)
25. Laghrissi, A., Taleb, T.: A survey on the placement of virtual resources and virtual network functions. IEEE Communications Surveys Tutorials **21**(2), 1409–1434 (2019)

26. Littman, M.L.: Value-function reinforcement learning in markov games. Cognitive Systems Research **2**(1), 55–66 (2001)
27. Liu, J., Li, Y., Zhang, Y., Su, L., Jin, D.: Improve service chaining performance with optimized middlebox placement. IEEE Transactions on Services Computing **10**(4), 560–573 (2017)
28. Ma, W., Beltran, J., Pan, D., Pissinou, N.: Traffic aware placement of interdependent nfv middleboxes. IEEE Transactions on Network and Service Management **16**(4), 1303–1317 (Dec 2019)
29. Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E.: Reinforcement learning for combinatorial optimization: A survey. CoRR (2020)
30. Mijumbi, R., Serrat, J., Gorricho, J.L., Bouten, N., Turck, F.D., Boutaba, R.: Network function virtualization: State-of-the-art and research challenges. IEEE Communications Sur. and Tut. **18**(1) (2015)
31. Pei, J., Hong, P., Pan, M., Liu, J., Zhou, J.: Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks. IEEE Journal on Selected Areas in Communications **38**(2), 263–278 (2020)
32. Sang, Y., Ji, B., Gupta, G.R., Du, X., Ye, L.: Provably efficient algorithms for joint placement and allocation of virtual network functions. In: Proc. of INFOCOM 2017
33. Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S., Sekar, V.: Making middleboxes someone else's problem: Network processing as a cloud service. In: Proc. of ACM SIGCOMM 2012
34. Soltesz, S., Pötzl, H., Fiuczynski, M.E., Bavier, A., Peterson, L.: Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. SIGOPS Oper. Syst. Rev. **41**(3), 275–287 (Mar 2007)
35. Sutton, R.S., Barto, A.G.: Reinforcement Learning, An Introduction. The MIT Press (2020)
36. Tang, L., He, X., Zhao, P., Zhao, G., Zhou, Y., Chen, Q.: Virtual network function migration based on dynamic resource requirements prediction. IEEE Access **7**, 112348–112362 (2019)
37. Tran, V., Sun, J., Tang, B., Pan, D.: Traffic-optimal virtual network function placement and migration in dynamic cloud data centers. Submitted to IEEE IPDPS 2022
38. Wang, L., Mao, W., Zhao, J., Xu, Y.: Ddqp: A double deep q-learning approach to online fault-tolerant sfc placement. IEEE Transactions on Network and Service Management **18**(1), 118–132 (2021)
39. Xiao, Y., Zhang, Q., Liu, F., Wang, J., Zhao, M., Zhang, Z., Zhang, J.: Nfvdeep: Adaptive online service function chain deployment with deep reinforcement learning. In: Proc. of the IWQoS 2019
40. Yang, S., Li, F., Trajanovski, S., Chen, X., Wang, Y., Fu, X.: Delay-aware virtual network function placement and routing in edge clouds. IEEE Transactions on Mobile Computing (2019)
41. Zhang, Q., Liu, F., Zeng, C.: Adaptive interference-aware vnf placement for service-customized 5g network slices. In: IEEE INFOCOM 2019 (2019)
42. Zhang, X., Wu, C., Li, Z., Lau, F.C.: Proactive vnf provisioning with multi-timescale cloud resources: Fusing online learning and online optimization. In: Proc. of IEEE INFOCOM 2017
43. Zhang, Y., Beheshti, N., Beliveau, L., Lefebvre, G., Manghirmalani, R., Mishra, R., Patney, R., Shirazipour, M., Subrahmaniam, R., Truchan, C., Tatipamula, M.: Steering: A software-defined networking for inline service chaining. In: Proc. of IEEE ICNP 2013