UNIVERSITY OF TECHNOLOGY SYDNEY
Faculty of Engineering and Information Technology

Network Function Virtualization for 5G Network Slicing

by

Da Xiao

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Sydney, Australia

2023

Certificate of Authorship/Originality

I, Da Xiao, declare that this thesis is submitted in fulfillment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature: Da Xiao Production Note:
Signature removed prior to publication.

Date: 14/06/2023

ABSTRACT

Network Function Virtualization for 5G Network Slicing

by

Da Xiao

Network slicing is a core technique of the 5G system and beyond. To meet the varied industrial demands, the International Telecommunication Union (ITU) has classified the current-generation mobile networks (5G) into three main categories: Ultra-Reliable Low Latency Communications (URLLC), enhanced Mobile broadband (eMBB), and massive Machine-Type Communications (mMTC). Packets belonging to the same category will be aggregated and then travel through the corresponding network slice, which comprises one or more network services. A network service can be symbolized as a service function chain (SFC) or a virtual network functions forwarding graph (VNF-FG), which is composed of a sequence of Virtual Network Functions (VNFs) and Virtual Links (VLs) connecting them. To accommodate as many network slices as possible with limited hardware resources, service providers need to refrain from over-provisioning resources, which may hinder them from cutting capital expenditures (CAPEX)/operating expenses (OPEX) for 5G infrastructure. Therefore, efficient and automatic placement of VNFs and VLs becomes one of the most critical technologies for meeting such requirements. This thesis aims to develop SFC placement algorithms in three scenarios: single data center (DC), multiple DCs (MDC) and Fog networks.

In the first scenario, we aim to improve cost efficiency while giving priority to higher-priority latency-sensitive services and maximizing the acceptance ratio of service requests in a single DC. To effectively progress towards our goal, we thoroughly investigated the current SFC embedding algorithms and developed two algorithms. In the first proposed algorithm, under the classical virtual network em-

bedding (VNE) model, we first used the DQN algorithm to maximize the acceptance ratio of service requests while ensuring prior allocation for higher-priority latency-sensitive services in a base environment. Then, we designed a binary search assisted transfer learning algorithm to realize this service in an ever-changing environment to minimize the cost. In the second proposed algorithm, we designed an algorithm under a new model where we can flexibly allocate resources to each virtual element. We used the Double DQN to choose the VNF placement (main action), employed the Dijkstra algorithm (first-phase sub-action) to derive the shortest path for each pair of adjacent VNFs in the given VNF chain, and devised a binary search assisted gradient descent algorithm (second-phase sub-action) to realize this service with the minimum cost. The simulation results show superior performance compared to the current SFC placement algorithms for latency-sensitive services in a single DC.

The dynamics and diversity of SFC requests in MDC networks, which might be neglected by most existing literature, pose a significant challenge in embedding SFCs. To overcome this challenge, in the second scenario, we proposed a two-stage GCN-based deep reinforcement learning framework for the placement of SFC in an MDC scenario, where the requests load may vary from DC to DC. In the first stage, we proposed a GCN-based DRL algorithm as a coarse granularity solution to the SFC embedding problem from the macro perspective. This solution outlines a local observation scope (LOS) for each agent in the multi-agent system of the second stage, where all agents simultaneously handle SFC requests from their respective DCs using a multi-agent framework from the micro perspective. Numerical evaluations show that, compared to state-of-the-art methods, the proposed scheme can improve the overall acceptance ratio of SFC requests and cost-effectiveness.

Fog computing, which brings computing/storage resources closer to terminal devices such as laptops, vehicles, and IoT devices, is an essential supplement to cloud data centers for supporting low-latency applications. However, the diversity of SFC request loads in different locations of a Fog network and the security issue of SFC requests have not been researched deeply in the existing literature. Therefore, in the third scenario, we designed a two-agent reinforcement learning algorithm for

SFC embedding in a Fog network. The SFC requests load may vary from location to location, and some requests have 'hard isolation' requirements. The proposed algorithm features a two-agent reinforcement learning model evolved from the multi-agent proximal policy optimization (MAPPO) model. The new model uses one agent (actor-network) to place VNFs and the other (actor-network) to chain the placed VNFs. This innovative architecture, in which the first agent communicates with the second, sufficiently explores the joint action space of the VNF placement and the chaining given the placed VNFs. Thus, it achieves better long-term accumulated rewards than existing schemes. Through experimental results, the proposed solution manifests better performance than state-of-the-art algorithms regarding the overall acceptance ratio of SFC requests and cost-effectiveness in the dynamic load scenario.

Dissertation directed by Prof. Ren Ping Liu and Prof. J. Andrew Zhang
School of Electrical and Data Engineering, University of Technology Sydney, Sydney, Australia

Acknowledgements

The accomplishment of my PhD thesis is owed to the contributions and support of many people. First of all, I would like to express my deepest thanks to my principal supervisor Prof. Ren Ping Liu for his guidance and support throughout my PhD study period at UTS. Professor Liu provided me with a precious opportunity to participate in a joint-project with high-tech giant Ericsson. During my cooperation with Ericsson, He taught me how to apply my research idea to industry projects, communicate with our industry partner, and write a research paper to summarize the results. His abundant experience, creative ideas, and rigorous attitude toward technical details set a good example for me. Professor Liu is also a kind person. During the covid-19 period, I had not met my family members for two years, and thus I applied for a six-month leave of absence to visit my parents and my wife. He understood my situation and approved my leave request quickly. The experience with Ren has taught me how to progress into a good researcher.

I also want to thank my cosupervisor Prof. Andrew Zhang, who held regular meetings fortnightly with me, for his great support to my work and life. Professor Andrew is a considerate person who can put himself in others' shoes. He always devoted his time and effort to answering my questions in detail and with great patience. He has polished all my research papers and suggested valuable suggestions for each of them. I feel lucky to be his student and grateful to him for his help and guidance.

I would like to express my gratitude to Prof. Wei Ni from CSIRO, who discussed the details of my research papers with me and guided me forward very often. We communicated in a relaxed atmosphere, which always brought new ideas. Whenever I had an idea or a question, I felt free to contact him immediately, and he always replied very quickly. He has extensive and abundant knowledge and a broad vision

for research work. I admire professor Wei Ni very much and will consult him when I encounter new problems in my research work in the future.

I also feel lucky to know three experienced researchers from world-famous universities, Prof. Xin Liu from UC davis and Prof. Jie Zhang and Dr. Shuo Chen from NTU. I am grateful to them for guiding me in my research work.

I am also thankful to Yiwen Qu, who usually discussed the details of the reinforcement learning algorithms in my papers with me. Finally, I would like to thank Dr. Shenghong Lee, Dr. Xu Wang, Dr. Ying He, Dr. Kai Wu, Dr. Zhipeng Lin, Dr. Shuo He, Dr. Xin Yuan, and Baoling Shan. They were always generous when I turned to them for help.

Finally, I want to thank CSC and UTS for their financial support during my PhD period.

Da Xiao

Sydney, Australia, 2023.

List of Publications

Journal Papers

J-1. D. Xiao, S. Chen, W. Ni, J. Zhang, J. A. Zhang, R. P. Liu, "A sub-action aided deep reinforcement learning framework for latency-sensitive network slicing," *Computer Networks*, vol. 217, pp. 109279, 2022.

J-2. D. Xiao, J. A. Zhang, X. Liu, Y. Qu, W. Ni, R. P. Liu, "A Two-Stage GCN-Based Deep Reinforcement Learning Framework for SFC Embedding in Multi-Datacenter Networks," *IEEE Transactions on Network and Service Management (TNSM)*. Status: Accepted on June 4, 2023.
https://ieeexplore.ieee.org/document/10146517

J-3. D. Xiao, W. Ni, J. A. Zhang, Y. Qu, R. P. Liu, "A GCN-Based Two-Agent Reinforcement Learning Framework for SFC Embedding in Fog Computing," *IEEE Communications Letters*. Status: Under the second review.

Conference Papers

C-1. D. Xiao, W. Ni, J. A. Zhang, R. Liu, S. Chen and Y. Qu, "AI-Enabled Automated and Closed-Loop Optimization Algorithms for Delay-Aware Network," *Proc. 2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 806-811, Apr. 10-13, 2022.

Contents

List of Figures

Abbreviation

NFV - Network functions virtualization

MANO - Management and Orchestration

VNF - Virtual Network Functions

VL - Virtual link

VNF-FG - Virtual Network Functions Forwarding Graph

SFC - Service Function Chaining

VM - Virtual Machine

DC - Data center

MDC - Multiple Data centers

MDP - Markov Decision Process

DRL - Deep reinforcement learning

DQN - Deep Q-Network

DDPG - Deep Deterministic Policy Gradient

PPO - Proximal Policy Optimization

MARL - Multi-agent reinforcement learning

LOS - Local Observation Scope

GCN - Graph Convolutional Network

AC: Actor Critic

Nomenclature and Notation

Boldface uppercase letters denote matrices.

Boldface lowercase letters denote vectors.

I_n is the identity matrix of dimension $n \times n$.

0_n is the zero matrix of dimension $n \times n$.

$\mathbb{R}$, $\mathbb{R}^+$ denote the field of real numbers, and the set of positive reals, respectively.

Chapter 1

Introduction

1.1 Background

Network Function Virtualization (NFV), which decouples network functions (NFs) from hardware and transforms these NFs into Virtual Network Functions (VNFs), is a crucial technology in 5G networks. The current 5G networks have been classified into three main categories: URLLC, eMBB, and mMTC. A network slice from any category can be symbolized by a Service Function Chain (SFC) or a VNF-Forwarding Graph (VNF-FG). 5G imposes more stringent QoS (quality of service) requirements on payload traffic than its predecessor, 4G systems, in support of 5G applications, such as remote surgery, IoT applications, and self-driving vehicles [1]. Therefore, efficient and automatic placement of VNFs and Virtual Links (VLs) becomes one of the most critical technologies for meeting such requirements.

Datacenters (DCs) have become a typical infrastructure for realizing NFV since we walked into the virtualization era. Over the years, many researchers have aimed to address the SFCs (the predefined VNFs and the VLs) placement problem in a single DC. Recently, to provide high-quality and non-disruptive network services to end-users, large enterprises such as Google, IBM, and Amazon have created multiple DCs in different geographic regions and built inter-DC networks to interconnect them. Therefore, multiple DC (MDC) has become a new research focus. Some paid close attention to Ethernet-based MDC; others focused on the SFC embedding issue in an Elastic Optical Networks (EON) based MDC.

Today, we have an enormous amount of client terminal devices such as laptops,

vehicles, and IoT devices that require access to the Internet; if they try to send all the generated data to a central cloud server in a DC for processing, there would be no enough resources (computing, storage, and bandwidth) to meet the QoS requirement of these applications, especially the latency-sensitive services. To resolve this issue, fog computing, which moves resources closer to the user or IoT device where the data needs processing, has been introduced to assist cloud computing. There are only a handful of studies on SFC embedding in fog computing, where cloud nodes and fog nodes cooperate to satisfy all SFC requests.

1.2 Motivation and Objectives

Existing SFC embedding algorithms fall into two main categories for all three scenarios: single DC, MDC, and fog networks. Some of them are heuristic-based methods that are good for stationary systems but could have degraded performance for dynamic systems. Others are Deep Reinforcement Learning (DRL)-based approaches, which are efficient when properly designed but inefficient when their action spaces become enormous. Considering the network dynamics such as the arrival rate of SFC requests and the requested resources of VNFs and VLs, we aim to design proper DRL-based algorithms, whose agents interact with the environment during the training phase, to learn and adapt to environmental changes in all three scenarios.

1.2.1 Prioritize Latency-sensitive Services and Allocate Resources to VNF Flexibly for the Single DC Scenario

In terms of the single DC scenario, there are two main open issues in all the existing works. First, only a few of them focus on the sub-slices issue, let alone latency-sensitive sub-slices. Second, when deploying network slices, most works adopt the classical Virtual Network Embedding (VNE) model, which assumes that

the required resources of VNFs are already specified in SFC requests, and thus do not need to consider the resource allocation issue of VNFs. However, in some real cases, customers might not be knowledgeable enough to configure VNFs. Hence, service providers need to help to allocate resources to VNFs. We designed two algorithms to tackle these issues step by step. In the first work, we focused on the sub-slices issue. We adopted the classical VNE model and designed a binary search assisted transfer learning algorithm to maximize the acceptance ratio while ensuring prior placement of higher-priority services for latency-aware services with the minimum cost. In the second work, we designed a sub-action-aided reinforcement learning algorithm to resolve the aforementioned two issues together.

1.2.2 Unbalanced SFC Requests Load Scenario of the MDC Network

Research on this problem in MDC networks is limited. Most of such works consider simple models by treating the MDC as a whole and assuming that the arrival of SFC requests follows the Poisson traffic model; thus, the load diversity of different DCs is generally neglected. In practice, the SFC requests load in an MDC may vary from DC to DC; e.g., some DCs may suffer traffic congestion while others are under-utilized. Hence, it is essential to design a load-balancing algorithm to resolve the SFC requests load diversity issue in MDC networks. In our work, we devised a Multi-Agent Reinforcement Learning (MARL) algorithm to maximize the overall acceptance ratio of SFC requests while minimizing the total cost in an MDC network.

1.2.3 Unbalanced SFC Requests Load Scenario and Security Issue of the Fog Networks

The studies on SFC embedding in NFV-based fog computing are insufficient. Two main open issues exist in most of such works. First, most of such works assumed that the SFC requests are uniformly distributed across the terminal layer

and proposed SFC embedding approaches accordingly. Actually, the SFC requests load may vary from location to location. They neglect this scenario in which their approaches may degrade. Second, only a few works have considered the security issue in SFC placement in fog networks. Some SFCs for cryptography services may require being physically isolated from others. To resolve these issues, we came up with a GCN-assisted two-agent DRL algorithm to maximize the acceptance ratio of SFC requests while minimizing the total cost in a fog network.

1.3 Approach and Contribution

1.3.1 The Single DC Scenario

We have designed two approaches for this scenario. In the first work, we model the VNF placement for all requests as a Markov Decision Process (MDP) and represent the MDP action space as the possible VNF placements of a single request. We further prioritize those requests based on their latency requirements and define the reward function of the MDP based on priority. For every incoming request, the DQN chooses an MDP action to determine the VNF placement. In response to the VNF placement, an MDP reward is returned from the environment to train our DQN. Once trained, the DQN approximates the optimal solution of the MDP that maximizes the acceptance ratio and ensures prior placement of higher-priority services. Then, to minimize the cost, we propose a Binary Search Assisted Transfer Learning algorithm (BSATL), in which the available hardware resources are scaled down/up and the knowledge learned from the source task is transferred to the target task in each iteration, to achieve automated and closed-loop optimization for a 6G Event Defined URLLC (EDuRLLC) [2] application.

The main contributions of this work can be summarized as follows:

1. We formulate the VNF placement problem as an MDP with an appropriate

state set, by including the real incoming traffic into the state set, rather than by using VNF capacity (CPU, memory, ...) to symbolize incoming traffic as in [3]. This is essential because, in some cases, VNF capacity could not accurately reflect real network traffic, which is critical to latency-sensitive services.

2. We prioritize service requests based on latency requirements (the lower latency threshold a service requests, the higher priority it gets) and propose a DQN framework to maximize the service acceptance ratio while ensuring prior placement of higher-priority requests.

3. We design an automated and closed-loop optimization algorithm (BSATL) for a 6G EDuRLLC application.

In the second work, We model the network slicing for all requests as a Markov Decision Process (MDP) and represent the MDP action space as the possible VNF placements of a single request. We prioritize those requests based on their latency requirements and define the reward function of the MDP based on priority and resource cost. For every incoming request, the DDQN first chooses an MDP action to determine the VNF placement. Given the VNF placement, the Dijkstra algorithm is then employed to embed the VLs. Finally, based on the placement of VNFs and VLs, the latency requirement, and the requested traffic of this service, we propose a resource allocation algorithm – Binary Search Assisted Gradient Descent (BSAGD) to provide the request with the minimum resource such that its latency requirement is satisfied. In response to the joint action (SFC placement and resource allocation), an MDP reward is returned to train our DDQN. Once trained, the SADDQN model approximates the optimal solution of ensuring priority allocation for higher-priority services and maximizing the acceptance ratio while minimizing the total cost.

The main contributions of our work can be summarized as follows:

1. Given a VNF placement (main action) and the optimal path traversing the VNF chain (the first-phase sub-action), we propose a resource allocation algorithm (the second-phase sub-action) to realize the network slice with the minimum cost. In other words, instead of assuming that the required resources of VNFs are known in advance, we optimally allocate resources to all VNFs in a given VNF chain based on the requested traffic and latency requirement of this service. In this way, for any given VNF placement, we only need to take the main action into the action space, because we find the optimal first-phase and second-phase sub-action for the main action. Otherwise, we would have to enumerate chaining and resource allocation options for each VNF placement, and thus the action space would be too huge for the DDQN algorithm to converge.

2. Most QoS-related works fail to slice a network slice into sub-slices. We prioritize services (the lower latency threshold a service requests, the higher priority it gets), define a reward function based on priority and cost, and propose a SADDQN framework to ensure priority allocation for higher-priority services and maximize the acceptance ratio while minimizing the total cost.

1.3.2 The MDC Scenario

We propose a two-stage graph convolutional network (GCN)-based DRL framework to maximize the overall acceptance ratio of SFC requests while minimizing the total cost in an MDC network.

In the first stage, we propose a DRL algorithm – GCN-based Proximal Policy Optimization (PPO) to derive a coarse granularity solution to SFC embedding from the macro perspective. Specifically, in each DC, we treat the IT resources requested by all SFCs in the waiting queue of this DC as a whole; thus, we can abstract the placement of all the SFCs received by the MDC into a load balance problem. To

balance loads in the MDC, we transfer some SFCs accumulated in high-load DCs to low-load DCs. We model the load transfer process as a Markov decision process (MDP) and represent the features of the MDC network as the MDP states and the possible DC-to-DC transfer options as the MDP actions. We use the reward function to capture the success/failure of an SFC transfer and the cost of network traffic caused by this transfer. At each time step of the MDP, the agent chooses an action based on our GCN-based PPO model and performs the chosen action in the MDC environment. In response to the action, an MDP reward, which will be used to train our model, is returned to the agent. Once trained, the model approximates the optimal solution for maximizing the overall acceptance ratio of SFC requests while minimizing the total cost. This stage aims to set a local observation scope (LOS) for each agent of the multi-agent framework in the second stage.

The second stage consists of two phases. During the first phase, to accommodate as many SFC requests as possible with limited resources, each agent embeds SFCs in its affiliated DC. The second phase commences once a high-load DC embeds SFCs in low-load DCs. In this phase, multiple agents of high-load DCs handle SFC requests simultaneously, with each embedding VNFs and VLs within its LOS step by step. Thus, we model the SFC embedding problem in this phase as a multi-agent MDP [4] and solve it using a MARL approach, GCN-based multi-agent PPO (MAPPO). A multi-agent MDP is defined by a set of states $\mathcal{S}$ that describes the possible global environmental features as well as a set of actions $\mathcal{A}_i$ $(i = 1, 2, \ldots, N)$ and a set of local observations $\mathcal{O}_i$ $(i = 1, 2, \ldots, N)$ for each agent. To choose actions, agent i feeds its observation o_i into the policy π_{θ_i} and gets a_i. The joint action of all agents generates the next global state using the state transition function $\mathrm{T}(s; a_1, a_2, \ldots, a_N) \rightarrow s'$, where N is the number of agents. Meanwhile, agent i obtains rewards based on a function of the global state and the agent's action $\mathrm{R}_i(s, a_i) \rightarrow r_i$ and receives a private observation correlated with the state $s' \rightarrow o_i$. Each agent dedicates itself to

maximizing its own total expected return $\sum_{t=0}^{T} \gamma^t r_{i,t}$ $(i = 1, 2, \ldots, N)$, where γ is a discount factor and T is the time horizon. In our multi-agent MDP, for agent i, we represent the features of the local network that agent i can sense, the position of the VNF embedded at the previous time step, and the index of the agent's affiliated DC as the local observations. In addition, the possible placements of the current VNF and the VL connecting this VNF and the previous constitute the action space. The reward function R_i consists of an external part and an internal part. The former is designed to achieve the common goal of all agents – maximizing the overall acceptance ratio of SFC requests while minimizing the total cost in an MDC environment. The latter is designed to accelerate the training. With training, each agent of the MAPPO model knows how to embed SFCs within its LOS to approximate the optimal solution.

The main contributions can be summarized as follows:

1. We propose a two-stage GCN-based DRL framework to handle service requests simultaneously from all DCs with the aim of maximizing the overall acceptance ratio of SFC requests while minimizing the total cost in an MDC network. This work effectively addresses the limitation in existing research on embedding SFCs in MDC networks by considering SFC requests load diversity of different DCs.

2. We abstract the placement of all SFCs within a time slot into a load balance problem from the macro perspective in the first stage. To balance loads in the MDC environment, we model the load transfer process as an MDP and develop a coarse granularity solution to the placement of SFCs such that the acceptance ratio of SFC requests is maximized while the total cost is minimized. The solution provides an appropriate LOS for each agent in the multi-agent framework; thus, we can tailor the action space of each agent to a proper size,

which speeds up the convergence rate of the algorithm in the second stage.

3. In the second stage, all agents handle SFCs received by their respective DCs simultaneously. In the initial phase, each agent places SFCs in its affiliated DC. In the second phase, we model the SFC embedding problem as a multi-agent MDP from the micro perspective and solve it using a MARL approach, GCN-based MAPPO. This multi-agent framework achieves better efficiency than single-agent RL algorithms in resolving the SFC requests load diversity issue in MDC networks.

1.3.3 The Fog Networks

We propose a GCN-assisted two-agent PPO framework to maximize the acceptance ratio of SFC requests while minimizing the total cost in a fog network.

We model the embedding of all SFC requests as a two-agent MDP, in which we accommodate SFC requests in sequence. These two agents both have their respective action space and local observation scope. The N-agent, which can only observe the node resources of each node in the fog network, is in charge of placing VNFs. The R-agent, which can only observe the link resources and the topology of the fog network, embeds the VLs between any two adjacent VNFs. For every SFC request, with the state of the node resources, the N-agent first chooses the locations for all VNFs based on its corresponding actor network. The locations of all VNFs, which are indicated in the output of the N-agent's actor network, will be fed to the actor network of the R-agent. With the input from the first actor network and the state of the link resources and the network topology symbolized by the graph embedding, the R-agent embeds all VLs to chain all adjacent VNFs pairs. The joint action, the placement of VNFs and VLs in the SFC request, generates the next global state with the state-transition function $\mathrm{T}(s; a_1, a_2) \rightarrow s'$. In addition, each agent obtains a reward based on a function of the global state and the agent's action

$\mathrm{R}_i(s, a_i) \rightarrow r_i$ $(i = 1, 2)$, and receives a local observation correlated with the state $s' \rightarrow o_i$ $(i = 1, 2)$. The reward function is defined based on the success/failure of the SFC request and resource cost. Both agents aim to maximize a shared total expected return $\sum_{t=0}^{T} \gamma^t r_{i,t}$ $(i = 1, 2)$, where γ is a discount factor and T is the time horizon. Once trained, the GCN-assisted two-agent PPO framework model approximates the optimal solution of maximizing the acceptance ratio while minimizing the total cost in the fog network.

The main contributions of our work are summarized as follows:

1. We propose a DRL-based framework for embedding SFCs, which resolves the dynamic SFC requests load issue and provides physical isolation for the SFC requests that require encryption computing.

2. We devise a two-agent reinforcement learning model based on the typical multi-agent actor-critic model [5]. The new model detaches the SFC embedding action into two actions, the VNFs placement and the VLs placement given the placed VNFs. It uses one agent (actor-network) to place VNFs and the other (actor-network) to chain the placed VNFs. This innovative architecture, in which the first agent communicates with the second, sufficiently explores the joint action space of the two agents. Meanwhile, it resolves the huge action-space issue for SFC embedding.

3. We adopt GCN with the attention mechanism to capture the topology features while excluding the node features for the R-agent.

1.4 Thesis Organization

This thesis is organized as follows:

- *Chapter 2*: This chapter is a survey of SFC embedding algorithms in all three scenarios.

- *Chapter 3*: Designed for the classical VNE model, our innovative BSATL algorithm, which ensures priority allocation for higher-priority services and maximizes the acceptance ratio of service requests while minimizing the total cost in a single DC, is described in this chapter.

- *Chapter 4*: Based on a more flexible model, our SADDQN algorithm achieves the same objective as the BSATL. Details are provided in this chapter.

- *Chapter 5*: This chapter presents our two-stage GCN-based DRL framework to maximize the overall acceptance ratio of SFC requests while minimizing the total cost in an MDC network, where SFC requests load varies from DC to DC.

- *Chapter 6*: This chapter presents our GCN-based two-agent PPO framework to maximize the overall acceptance ratio of SFC requests while minimizing the total cost in a fog network, where SFC requests load varies from location to location, and some SFC requests have hard isolation requirements.

- *Chapter 7*: A summary of this thesis is provided in this chapter. Recommendation for future works is also mentioned.

Chapter 2

Literature Review

NFV, with which we can build multiple virtual network services on top of a common hardware platform, is a key technology in 5G networks. In [6], the authors illustrated research challenges and state-of-the-art of NFV comprehensively, and the placement of network services is a critical aspect. In 5G networks, our objective is to accommodate as many network services as possible with limited hardware resources. Furthermore, the successful placement of a network service requires that both the resources and the QoS constraints be satisfied. Hence, efficient network services placement algorithms become essential. Many surveys have been devoted to the placement of network services techniques [7, 8, 9, 10]. In this chapter, we pay close attention to those works studying SFC embedding approaches in a single DC, MDC, and fog networks. In general, they fall into the following two categories.

2.1 Heuristic-based Approaches

In a single DC scenario, most techniques were proposed to formulate and solve optimization problems [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]. Some works [13] and [14] are based on strong assumptions. For example, the authors in [13] assumed that the delay of each link is 30ms. In [14], the authors used packet loss ratio to reflect network congestion in simulation. But they assumed that the packet loss ratio is a fixed number, like 0.02 or 0.03.

Some works first make predictions about network parameters and then determine solutions for VNF. The authors in [15] first proposed a traffic forecasting method.

Then, they devised two VNF placement algorithms, which are based on the forecast traffic, to guide online VNF scaling. While the forecast traffic curve is very close to the real-time traffic curve in most cases, its variation trend is sometimes one time step later than that of the real-time traffic curve. In our work, we aim to accommodate as many latency-sensitive services as possible with the minimum cost. For a deployed service, if the predicted traffic rate is lower than the real-time traffic rate, then the latency requirement of this service might not be satisfied. Hence, we cannot tolerate prediction error and thus design our algorithms in an environment where the traffic characteristics that we care about can be learned from traffic history.

Some works [16, 17], and [18] focus on the convergence rate of their algorithms. The authors of [16] designed an eigendecomposition-based approach to deal with VNF Forwarding Graph (VNF-FG) placement. This matrix-based method reduces the complexity and speeds up the convergence. However, it does not explore the immense space of possible actions. A novel approach that combines Markov approximation with matching theory, Sampling-based Markov Approximation (SAMA), was proposed in [17] to minimize the joint operational and traffic cost. In [18], the authors formulated the VNF placement and flow routing problems as Integer Linear Programming (ILP) optimization problems and designed a set of heuristics to find near-optimal solutions. Our first proposed approach for the single DC scenario speeds up the convergence rate by combining binary search with transfer learning. The second proposed approach speeds up the convergence rate by compressing the action space using our resource allocation algorithm (BSAGD). Although the convergence rate is important, cost-effectiveness is more important regarding our objective. Therefore, we focus more on the optimal acceptance ratio and cost-utility than on the convergence rate.

Others [20, 12], and [19] are based on delay models. In [12], the authors proposed a fine-grained delay model and extended the VNF placement optimization in [11]

with delay constraints. As another work, the authors in [19] formulated the problem of finding the optimal number of VNFs and their locations as an ILP. Then they proposed a cost-efficient proactive VNF placement and chaining algorithm to resolve it. The algorithm aims at accepting each request with the minimum cost while meeting its latency requirement. However, as in [12], it does not prioritize those value-added services based on their latency requirements. In that case, an infrastructure with insufficient resources might not ensure priority allocation of high-priority services. Recently, the authors in[20] first formulated the delay-constrained and power-aware joint VNF placement and routing problem as an ILP. Then they proposed a fast heuristic algorithm – Holu to resolve it. The authors calculated the end-to-end delay of a VNF chain by adding up the propagation delay of each link and the processing delay of each VNF. However, queuing delay and propagation delay might also be considered in some real cases. Therefore, creating or choosing a delay model that achieves an accurate estimation is critical for the VNF placement of latency-sensitive services. In our work, to ensure priority allocation of services with more strict latency requirements, we prioritize requests based on their latency requirements. To ensure an accurate estimation of the end-to-end delay, we design our frameworks based on a mature 5G end-to-end delay model, whose accuracy has been verified in [21].

In the MDC networks scenario, most techniques were proposed to formulate and solve optimization problems [22, 23, 24, 25]. Some works, [22, 23], and [24], are based on a strong assumption. In [22], the authors proposed a heuristic algorithm named LBA (LCS Based Algorithm) to realize efficient VNF service chaining in inter-DC elastic optical networks (EONs). Based on the Longest Common Subsequence (LCS) principle, this algorithm reuses the VNFs that have been deployed previously in multiple DCs to save IT cost. To reduce the inter-DC traffic cost, it places new instances of VNFs in DCs based on the nearby principle; thus, VNFs

may concentrate in some DCs, while others are under-utilized. Consequently, some DCs might suffer traffic congestion or performance degradation. To resolve this issue, the authors of [23] proposed a resource balancing algorithm (RBA) for joint placement of service chains and routing/spectrum assignment (RSA) in inter-DC EONs. Also, in [24], a joint-optimization selection (JOS) algorithm was designed to select VNFs to achieve joint load balancing of IT and spectrum resources. In [25], the authors formulated the VNF placement as a binary integer programming model and proposed the Service Function Chains eMbedding APproach (SFC-MAP) algorithm, in which the shortest path algorithm is iterated based on a multi-layer cost graph, to acquire the optimal placement solution. In all these works, the VNF reuse solution improves resource utilization. This solution is based on the assumption that the capacity of a VNF is fixed, which is reasonable during the early stage of virtualization [26]. Nevertheless, when we walk into the Kubernetes era, VNFs are embedded in containers, for which CPU allocation can be as small as 1/1000 of the total CPU resources of a server [27]. Hence, the resource allocation for a VNF can be flexible, and we can initiate VNFs instances based on the number of virtual resources requested in the corresponding service requests. Furthermore, for security purposes, many customers are unwilling to share VNFs with others. Therefore, in our work, we consider dynamic and unshared VNFs.

The study on SFC embedding in fog computing is limited. The authors in [28] studied the SFC embedding issue in a two-layer fog network. They proposed a hybrid algorithm in which Tabu searches locations for VNFs and a graph-based shortest-path module embeds VLs. This approach designs "sub-regions" to partition the search area and restricts the VNF mappings into a small scope close to a primary node. It really places the SFC near the source and thus reduces latency and conserves bandwidth. However, looking at the whole picture, this scheme would lead to a low acceptance ratio of SFC requests once the SFC requests load varies from location to

location. In [29], for a latency-sensitive SFC, the authors first found several shortest paths between the source and destination pair. Then, they used two heuristics, delay minimum (DM) and load minimum (LM), to derive a VNF placement solution, respectively. Unlike most existing SFC embedding approaches, this algorithm selects several candidate paths for VL embedding, then obtains a VNF placement option based on DM or LM principle, making it attractive. There exist two main drawbacks to this work. First, for a latency-sensitive service, we do not need to realize it with the minimum delay. Over-provisioning of resources may result in a low acceptance ratio of SFC requests. In contrast, we just need to satisfy the delay requirement of each service and try to accommodate all services. Second, the LM tries to balance the load of all fog nodes across the network. Actually, when SFC requests loads are different in different locations, this scheme might also unreasonably reject SFC requests. These two issues also have not been addressed in [30], the authors of which designed a hierarchy Descending SFC embedding scheme with load balancing. This algorithm places VNFs on higher-layer nodes first. Then, they place SFC on the lower layer with load balancing to maximize the acceptance ratio of SFC requests. Nevertheless, they assume the bandwidth is abundant, and thus they only consider the load balance of node resources but not bandwidth.

In summary, heuristics are efficient in stationary systems. Nevertheless, in scenarios where environmental characteristics, such as the arrival rate and the required resources of service requests, change dynamically, these approaches must be frequently re-triggered to obtain optimal solutions for new environments. In contrast, our proposed algorithms, whose agent interacts with the environment during the training phase, can learn the environment changes in all scenarios before it converges.

2.2 Reinforcement Learning-Based Approaches

These years, techniques based on reinforcement learning (RL) were developed [3, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42]. The valuable ability to learn from past experiences makes these approaches noteworthy. The authors in [32] devised a distributed reinforcement learning algorithm for SFC embedding, which speeds up the convergence. But they did not consider whether they should design a dedicated network for communication between the agents. In [33], the authors made VNF placement decisions based on the end-to-end performance predictions. Although the prediction algorithm performs very well in dynamic conditions, as in [15], it would need to reserve resources for VNFs in case of prediction error. Therefore, making prediction-based VNF placement solutions could not be very cost-efficient. Instead of learning from scratch, the authors in [34] trained a DRL agent to learn how to reduce the optimality gap of heuristic-based solutions. For the latency issue, [3] is the first paper that proposes to extract latency in real time, making it very attractive. Adopting the real-time model can greatly enhance the accuracy of the latency. However, there exist several limits in [3]. First, they used VNF-FGs requested by clients to symbolize traffic in the deep reinforcement learning state. It is important to take VNF-FGs into the state, but it is equally important to include the incoming traffic of each service, an indispensable feature of the environment. This is because, for a given VNF-FG configuration, different incoming traffic may result in different real latencies. Second, the authors assumed that the requested resources of VNFs are normalized and uniformly distributed, which could be impractical in some real cases. For instance, in an OpenStack (an open-source cloud computing infrastructure software project) supported cloud environment, there are several available 'flavors'[26] for resources like VCPU, RAM, and Disk. Therefore, these required resources are discrete random variables rather than continuous random variables.

As for MDC networks, the authors in [35] presented a DRL-based SFC embedding algorithm in inter-DC EONs. To solve the problem that the size of the state vector is not fixed because of the varied length of SFCs and the ever-changing number of available DCs, a feature matrix-based encoding solution was put forward. This innovation makes RL-based algorithms feasible in network environments whose state sizes are prone to change. In [36], the authors proposed DeepRMSA, a DRL-based routing, modulation, and spectrum assignment (RMSA) framework for learning the optimal online RMSA policies in EONs. The authors in [37] developed a hybrid DRL-based framework for SFC embedding across geo-distributed DCs. In [35, 36] and [37], the authors used deep neural networks (DNN) to extract the features of the complex EON states. However, for an environment that can be represented by a graph, Graph neural networks (GNN)-based encoding schemes might be more efficient in extracting its features. The authors in [38] designed a GNN-based hierarchical DRL algorithm for the VNF placement and RSA in EONs. They proposed a GNN-based encoder to abstract the features of the EON network states and the context of the pending VNFs. This GNN-based encoder advances the VNF-FG placement in network topologies that a graph can normally represent. However, as the authors in [37] and [39] did, they decoupled the placement of VNFs and VLs and designed a routing module to select the optimal route given a VNF placement. This scheme reduces the action-space size and speeds up the convergence rate. Nevertheless, the greed for a high immediate reward might result in a disappointing long-term cumulative reward.

For fog networks, the authors in [40] proposed a DRL for SFC allocation. The DRL-agent learns the optimal resource allocation decisions to reduce costs from a previously presented Mixed-integer linear programming (MILP) formulation. Based on the IMPortance weighted Actor-Learner Architectures (IMPALA), the authors in [41] proposed a distributed placement technique, which employs an adaptive off-

policy correction method for faster convergence. In [42], the authors proposed an Intelligent Fog and Service Placement (IFSP) to perform instantaneous placement decisions proactively. This algorithm considers dynamics in both the customers' requests and the available infrastructure resources, which is a pioneering 6G technology.

For all scenarios, most works adopted the classical VNE model, which symbolizes the characteristics of SFC requests. They directly treated the assumed values of required resources as the input of their approaches. However, in some real cases, customers are familiar with the QoS parameters but not the VNF size. Therefore, we designed a resource allocation algorithm to allocate resources flexibly to an SFC based on its incoming traffic and latency requirement in a single DC scenario. The resources allocation module works as a sub-action of a given SFC placement in each step of the reinforcement learning process.

Most works on the MDC issue [22, 23, 24, 35, 36, 37, 25, 38] treated the MDC network as a whole and assumed that the arrival of SFC requests follows the Poisson distribution. However, in some real cases, the diversity of SFC requests in an MDC may pose an important challenge. Thus, we designed a two-stage GCN-based RL algorithm for the SFC embedding problem in an MDC network, where the SFC requests load varies from DC to DC.

As in the MDC scenario, the diversity of SFC request loads in different locations of a fog network has not been researched deeply in the existing literature. Also, the security issue in SFCs embedding is paid little attention to. Last but not least, for the SFC embedding issue, to compress the action-space and thus accelerate convergence, most DRL-based algorithms, including the one we designed for the single DC scenario, only use the DRL-agent to explore the action-space of the VNFs placement. Typically, they designed a routing module to embed VLs given the

embedded VNFs. In this way, a VNFs placement option has only one corresponding VLs placement option. This scheme really can compress the action-space, but it neglects the action-space of the VLs embedding. One-step greedy algorithms will always lead to an unfavorable global optimum solution. Hence, we devised a two-agent GCN-assisted DRL algorithm to resolve all these issues in a fog network.

2.3 Summary

In this chapter, we have analyzed existing SFC embedding approaches in different scenarios (single DC, MDC, and fog networks) and summarized their strengths and weaknesses. To resolve the issues that have been neglected in existing literature, we have proposed SFC placement schemes for all scenarios. In the next chapter, we will introduce our SFC embedding algorithms in a single DC.

Chapter 3

SFC Embedding in an NFV-based Single DC under the Classic VNE Model

In this chapter, we will introduce a DQN-based algorithm to maximize the acceptance ratio of all SFC requests and ensure prior placement of higher-priority SFCs in a single DC environment. In this algorithm, we first train a standard DQN model in a base environment to get the optimal SFC placement solution. Then, we propose a Binary Search Assisted Transfer Learning algorithm (BSATL), in which the available hardware resources are scaled down/up and the knowledge learned from the source task is transferred to the target task in each iteration, to achieve automated and closed-loop SFC placement optimization for the ever-changing infrastructure.

3.1 System Model and Problem Formulation

3.1.1 System Architecture

In the NFV architecture, Network Slice Management Function (NSMF) receives SFC requests. Accordingly, it interfaces with Management and Orchestration (MANO) to plan for SFC placement. In our work, the DQN-agent, which takes the role of MANO, is responsible for choosing the location for VNFs in each SFC. As for the infrastructure, we adopt a widely used non-blocking fat-tree topology, depicted in Fig. 3.1. Let $\mathbb{D}$ denote the set of edge clouds and $\mathbb{E}_i$ denote the set of servers in the i^{th} edge cloud. For simplification, we assume that the number of servers in all edge clouds are the same. Hence, the number of edge clouds is $|\mathbb{D}|$, and the number of servers in each edge cloud is $|\mathbb{E}|$. Without loss of generality, we consider the following example.

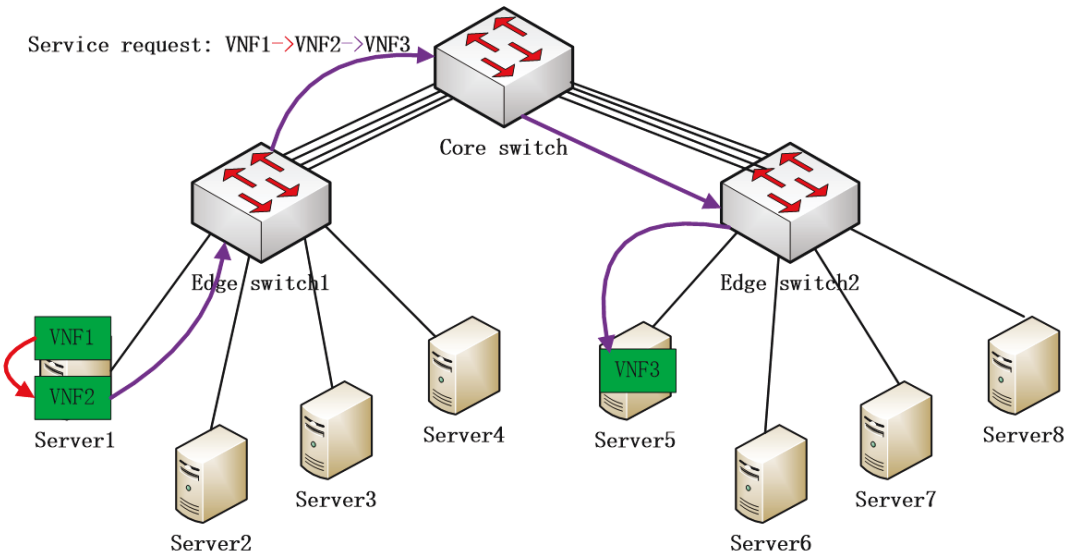

Figure 3.1 : Network infrastructure considered in this paper.

When DQN-agent handles an SFC request VNF1→VNF2→VNF3, we suppose that it chooses server1 to place VNF1 and VNF2, and server5 to place VNF3. After the incoming traffic travels through VNF1→VNF2→edge switch1→core switch→edge switch2→VNF3, the agent extracts the real latency to see whether the corresponding latency requirement is satisfied.

3.1.2 Characteristics of SFC Requests

Let K represent the number of SFC requests. In addition, the number of VNFs for a URLLC SFC is assumed to be V. Unlike the authors in [20], who assume that all SFCs have the same latency requirement, we assume that, within the URLLC category, each SFC has its distinctive latency requirement. Hence, we represent the latency threshold of the i^{th} SFC with L_i.

3.1.3 Problem Formulation

In this part, we formulate the objective as an optimization problem. Our purpose is to maximize the acceptance ratio while ensuring prior placement of higher-priority

SFCs, under the infrastructure resource constraints. Thus, the optimization problem can be written as

$$(P1): \max_{\boldsymbol{A}} C(\boldsymbol{A}) = \sum_{i=1}^{K} \rho_i \times f(L_i - l_i)$$

$$\text{s.t.} \begin{cases} \sum_{i=1}^{K} \sum_{j=1}^{V} x_{i,j}^{n} \times c_{i,j} \leq C_{\text{tot}}, \ n = 1, 2, ..., |\mathbb{D}| \times |\mathbb{E}| & \text{(3.1a)} \\ \sum_{i=1}^{K} \sum_{j=1}^{V} x_{i,j}^{n} \times w_{i,j} \leq W_{\text{tot}}, \ n = 1, 2, ..., |\mathbb{D}| \times |\mathbb{E}|. & \text{(3.1b)} \end{cases}$$

In the objective function above,

$$f(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0, \end{cases} \tag{3.2}$$

$$\rho_i = \frac{\frac{1}{L_i}}{\sum_{i=1}^{K} \frac{1}{L_i}}, \ i = 1, 2, ..., K, \tag{3.3}$$

ρ_i denotes the priority of SFC i, element $a_{i,j}$ in matrix $\boldsymbol{A}$ indicates on which server is the j^{th} VNF of the i^{th} SFC located, and L_i and l_i respectively symbolize the latency requirement and the real latency of the i^{th} SFC.

In the constraints above, the binary variable $x_{i,j}^{n}$ indicates whether the j^{th} VNF of the i^{th} SFC nests on the n^{th} server, $c_{i,j}$ and $w_{i,j}$ denote the required CPU and bandwidth resources of the j^{th} VNF of the i^{th} SFC, and C_{tot} and W_{tot} denote the CPU and bandwidth capacity of any server.

The constraints need to be modified if the infrastructure availability changes. Learning-based techniques can be utilized to learn such network dynamics and solve the problem. The mission of the learning technique is to learn a policy that generates the optimal action under each environment state. In the next section, we propose a DQN-based model for the SFC placement problem for latency-sensitive services.

3.2 Our Proposed SFC Placement Scheme

We model the VNF placement for all SFCs as an MDP, which can be resolved by a DQN framework, and represent the MDP action space as the possible VNF placements of a single SFC request. A state can be defined as a vector consisting of two parts: the available resources of the infrastructure and the characteristics of an SFC request. Once the VNF placement action of the first request is decided, the state is updated. Then the action for the second one will be determined, so on and so forth. Since our aim is to maximize the acceptance ratio while ensuring prior placement of higher-priority SFCs, we prioritize those requests based on their latency requirements and define the reward function of the MDP based on SFC priority. The details of the state, action, and reward function in the context of our problem will be provided in Sections 3.2.2, 3.2.3, and 3.2.4.

We first propose a framework based on DQN [43] to maximize the acceptance ratio and ensure prior placement of higher-priority SFCs. Then, to resolve the over-provisioning of resources, we propose the BSATL algorithm, in which the minimum number of online servers for maintaining the maximum acceptance ratio is iteratively searched for, to achieve automated and closed-loop optimization for the 6G Event Defined uRLLC (EDuRLLC) scenario. The details are presented below.

3.2.1 Overview of the Proposed DQN

A general overview of the DQN used in our scheme is shown in Fig. 3.2. The state, indicated by a vector, is fed into the neural network, which outputs a vector of Q-values, with each indicating the expected discounted cumulative reward of a corresponding action (VNF placement). At time step t, the Q-value of taking an action a_t in state s_t based on policy π is given by

$$Q^{\pi}(s_t, a_t) = E(\sum_{i=t}^{K} \gamma^{(i-t)} R(s_i, a_i) | s_t, a_t). \tag{3.4}$$

The objective of the agent is to learn a policy that maximizes the expected return $Q^{\pi}(s_t, a_t)$. At the initial stage of the training, the weights of the evaluation neural network are random, and thus the policy is poor. Given a state, the maximum Q-value may not account for the optimal policy. Hence, we train the framework episode by episode. When an episode starts, we re-initialize the state of the environment and feed the agent K requests, which will be handled one by one in K time steps. In each time step, the agent serves a request and updates the neural networks. When the agent finishes the placement of K requests, this episode ends, and the next one starts. The loop ends as the weights of the neural networks converge. Then, given a state, the agent is able to select the best action by taking the largest Q-value from the output vector. The workflow for each time step can be summarized as follows:

1. Step 1: The agent observes the state (s_t) of the environment.

2. Step 2: The agent performs an action (a_t), i.e., the VNF placement, randomly with probability ϵ or according to the evaluation network with probability $1 - \epsilon$, and thus gets a reward (r_t).

3. Step 3: The experience tuple (s_t, a_t, r_t, s_{t+1}) is stored into an experience replay buffer.

4. Step 4: To train the DQN framework, a mini-batch of N tuples are uniformly sampled from the experience replay buffer.

5. Step 5: For tuple i, s_i and a_i are fed into the evaluation network (θ), while s_{i+1} is fed into the target network (θ^{-}).

6. Step 6: The weights of the evaluation network are updated by minimizing a loss function:

$$L = \frac{1}{N} \sum_{i=1}^{N} (y_i - Q(s_i, a_i|\theta))^2, \tag{3.5}$$

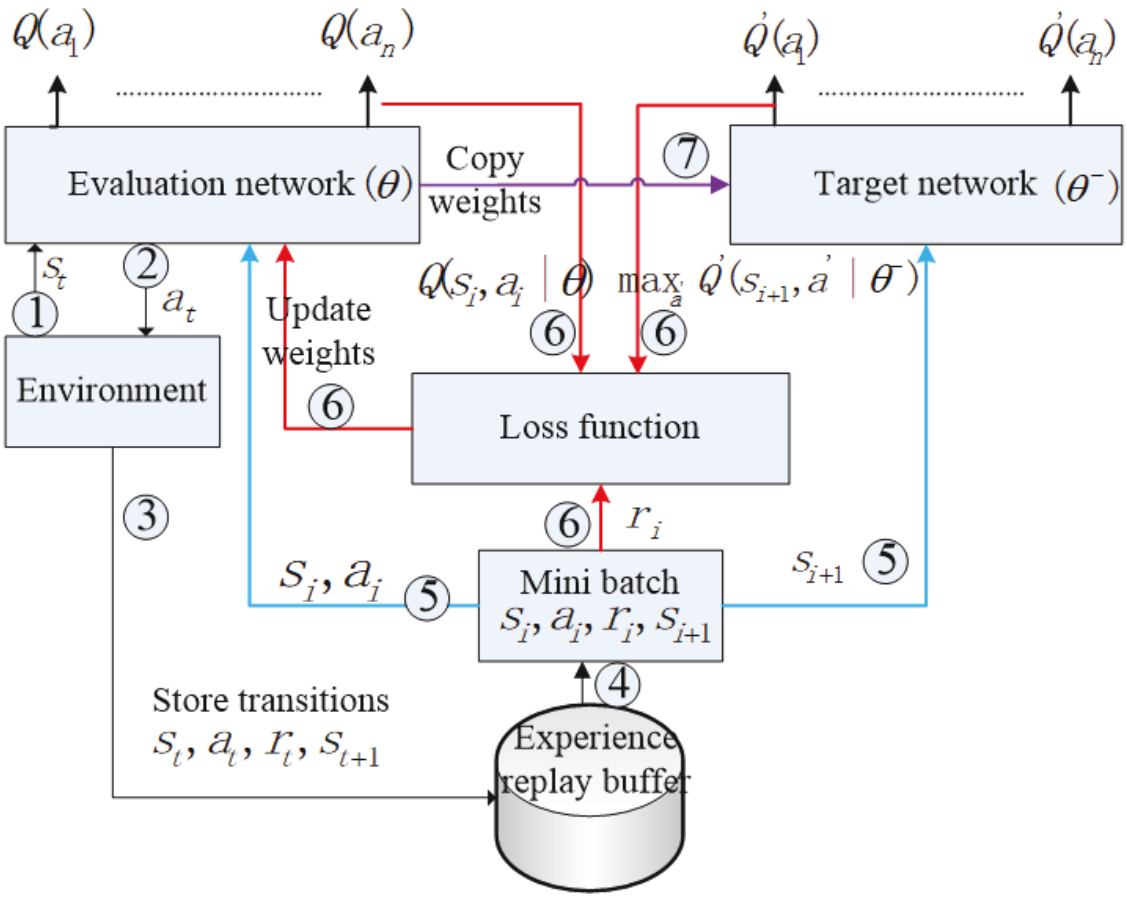

Figure 3.2 : Flowchart of the Deep Q learning network.

where

$$y_i = r_i + \gamma max_{a'} Q'(s_{i+1}, a' | \theta^-). \tag{3.6}$$

7. Step 7: The weights of the target network are updated by copying the weights of the evaluation network every C time steps.

3.2.2 State

We define the state as a vector consisting of two parts: the remaining resources of the infrastructure and the characteristics of an SFC request. Subsequently, the

state set of the environment can be given as

$$\begin{aligned} &\boldsymbol{S} = \{c_n, w_n, c_{i,j}, w_{i,j}, L_i, \lambda_i\}, \\ &n = 1, 2, ..., |\mathbb{D}| * |\mathbb{E}|, j = 1, 2, ..., V, i \in \{1, 2, ..., K\}, \end{aligned} \tag{3.7}$$

where c_n and w_n respectively represent the remaining CPU and bandwidth resources of the n^{th} server, $c_{i,j}$ and $w_{i,j}$, which can be selected from VM 'flavors', denote the required CPU and bandwidth resources of the j^{th} VNF of the i^{th} SFC, and L_i and λ_i respectively denote the latency requirement and the incoming traffic of the i^{th} SFC.

3.2.3 Action

We define the action as the possible placement for an incoming SFC i:

$$\boldsymbol{A} = \{a_{i,j}\}, j = 1, 2, ..., V, i \in \{1, 2, ..., K\}, \tag{3.8}$$

where $a_{i,j}$ indicates on which server does the j^{th} VNF of the i^{th} SFC nest.

3.2.4 Reward

We leverage a simulation tool CloudSimSDN-NFV [44] to obtain the real latency of an embedded SFC. If the SFC's latency requirement is not satisfied, the agent will receive a penalty. Otherwise, it will be granted a reward. For the i^{th} SFC, the reward function regarding delay is defined as

$$R_{\text{delay}}(i) = \begin{cases} 1 & l_i \leq L_i \\ \\ -1 & l_i > L_i, \end{cases} \tag{3.9}$$

where l_i and L_i denote the real latency and the latency requirement of the i^{th} SFC.

We suppose that the j^{th} VNF is planned to be placed on the n^{th} server. If the remaining bandwidth or CPU of server n is insufficient to accommodate VNF j, then the placement for this VNF is a failure. When DQN-agent performs an action

for SFC i, the failure placement for any VNF can lead to a penalty for this action. Therefore, the reward function regarding resources constraint is defined as

$$\mathrm{R}_{\mathrm{res}}(i) = \begin{cases} -1 \\ if\ \exists j \in [1, V],\ c_{i,j} > c_n \text{ or } w_{i,j} > w_n \\ 0 \text{ otherwise,} \end{cases} \tag{3.10}$$

where $c_{i,j}$ and $w_{i,j}$ denote the required CPU and bandwidth resources of the j^{th} VNF of the i^{th} SFC, and c_n and w_n respectively represent the remaining CPU and bandwidth resources of the n^{th} server.

We place VNFs for all SFCs such that a global utility function expressed below is maximized.

$$U = \max \sum_{i=1}^{K} (\eta_1 \times \rho_i \times R_{\mathrm{delay}}(i) + \eta_2 \times R_{\mathrm{res}}(i)), \tag{3.11}$$

where η_1 and η_2 are the weights of two reward functions and ρ_i indicates the priority of SFC i. For any SFC placement, satisfying the resources constraint is the prerequisite for being measured the delay. Thus, η_2 is greater than η_1.

3.2.5 The Binary Search Assisted Transfer Learning Algorithm

Our BSATL algorithm is summarized in Algorithm 1. The main workflow is shown in Fig. 3.3. The algorithm starts from a DQN learning process in an initial environment with the maximum number of servers (S_{max}). S_{max} and 0 are considered the two ends of the binary search (BS). A new iteration of transfer learning is triggered once the BS creates a new environment, indicated by the tentative maximum number of servers (t_{max}). In each iteration, the source task and target task [45] share the same action space and reward function, and the only difference between the two tasks is the DQN state. Hence, it is reasonable to reuse the neural network model developed in the source task as the starting point of the target task. When

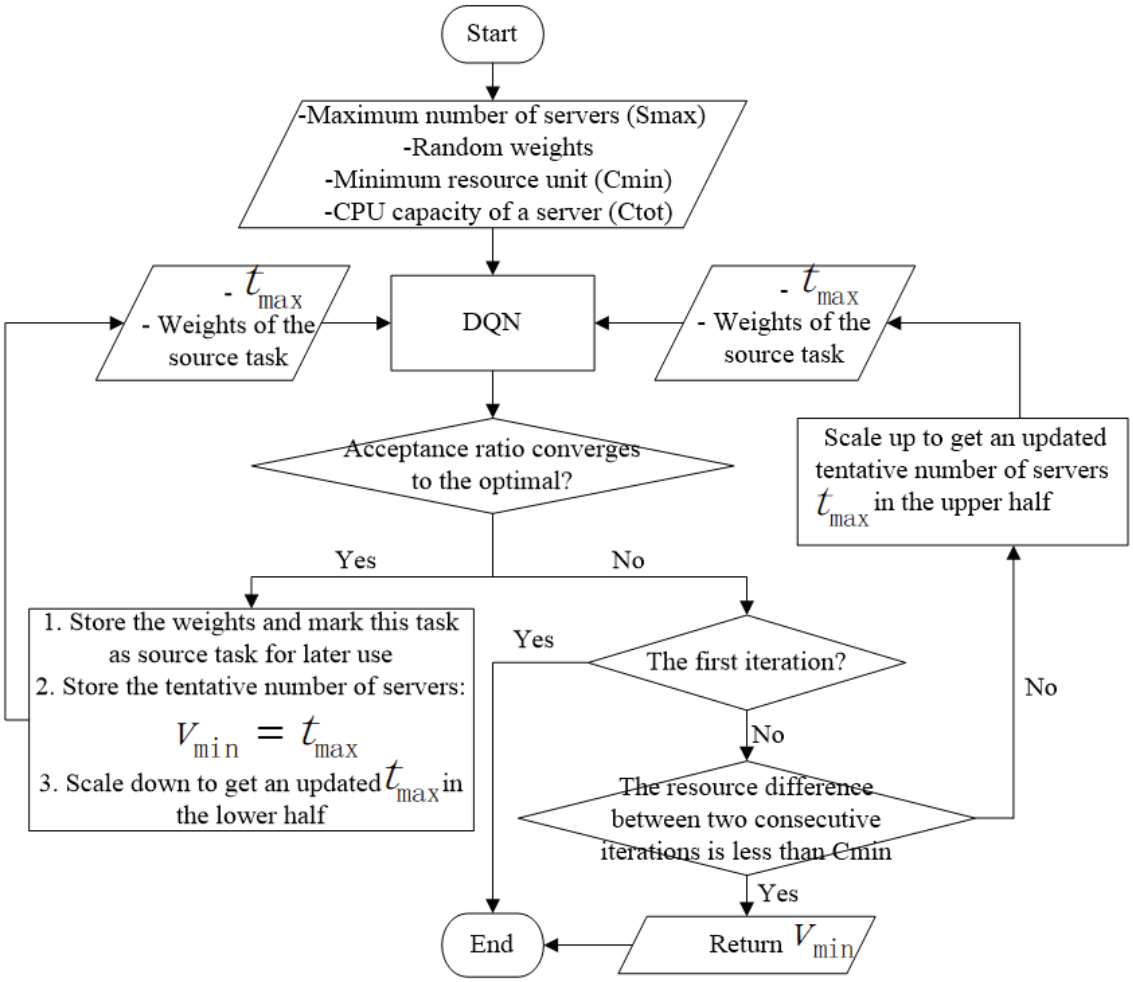

Figure 3.3 : Workflow of the BSATL algorithm.

the resource gap between the most recently verified environment and the current tentative environment is less than the minimum resource unit (C_{min}), the loop ends, and the minimum number of servers (v_{min}) for maintaining the maximum acceptance ratio is returned.

3.3 Simulation

3.3.1 Simulation Setup

In this section, we evaluate the performance of the proposed DQN algorithm regarding the acceptance ratio in phase one and the cost-utility of the proposed

Algorithm 1 BSATL Algorithm

Input: $S_{\max}, C_{\text{tot}}, C_{\min}$

Output: $v_{\min}$

$t_{\min} = 0, t_{\max} = S_{\max}, v_{\min} = 0$

$index = 1$

while $(v_{\min} - t_{\max})C_{\text{tot}} > C_{\min}$ or $index = 1$ **do**

  Substituting $t_{\max}$ into DQN state and training or

  retraining the network to see whether the DQN converges

  if the acceptance ratio converges to the optimal **then**

    $v_{\min} = t_{\max}, t_{\max} = \frac{t_{\max}+t_{\min}}{2}$

  else

    $t_{\min} = t_{\max}, t_{\max} = \frac{t_{\min}+v_{\min}}{2}$

  end if

  $index = index + 1$

end while

return $v_{\min}$

BSATL algorithm in phase two. We use the CloudSimSDN-NFV as our simulation framework. We can see that in CISCO's typical data center infrastructures [46], the bandwidth of a physical link is either 1 Gbps or 10 Gbps. Thus, we set the bandwidth of each link to 1 Gbps for our infrastructure (Fig. 3.1), and the MIPS of each server is 1000. In our experiment, we assume that there are five categories of latency-sensitive SFCs, each of which consists of a chain of three VNFs and has a distinctive latency threshold. In terms of priority, flow1>flow2>flow3>flow4>flow5. For security purposes, we assume that they are isolated networks, and thus they do not share VNFs.

The DQN network is designed with the following parameters. Adam is employed to learn the neural network hyperparameters. The learning rate is 0.005, and the discount factor is 0.99. The weights of the target network are updated every 100 episodes. The batch size is 64. We set two hidden layers for the fully connected network; the number of units is 4096 for the first hidden layer and 1024 for the second. We choose the Rectified Linear Unit (ReLU) activation for hidden layers.

Our proposed algorithms both consist of an offline training phase and an online testing phase. In the training phase, we create SFC requests according to their traffic and arrival pattern history and continuously train the framework until the weights of neural networks converge. During the online testing phase, we compare the performance of our proposed algorithms with two state-of-the-art algorithms.

3.3.2 Algorithms to Compare

Holu

The Holu algorithm [20] addresses the VNF placement and routing problem with the objective of minimizing the number of online physical machines and network switches, under the end-to-end delay and resource constraints. This fast heuristic framework efficiently solves the delay-constrained and power-aware joint VNF

placement and routing (PD-VPR) problem in an online mode. In detail, the Holu detaches the joint problem into two sub-problems and solves them in sequence: i) mapping VNFs to PMs based on a centrality-based PM ranking strategy, and ii) obtaining a Delay-Constrained Least-Cost (DCLC) shortest path through the selected PMs using the Lagrange Relaxation-based Aggregated Cost (LARAC) algorithm.

Best-Fit

The Best-Fit algorithm deploys VNFs one after another to the smallest free partition that meets the resource requirements of the VNFs.

3.3.3 Simulation Results

In phase one, we compare our DQN algorithm with the Holu and the Best-Fit regarding the acceptance ratio. From Fig. 3.4, we can see that when we restrict the number of online servers to four, the DQN and the Holu accept four requests, while the Best-Fit accepts three. Furthermore, only our DQN algorithm accepts four higher-priority SFCs and discards the lowest-priority SFC. This is because we assign greater reward/penalty values to the success/failure of higher-priority SFC requests, and the DQN algorithm achieves the maximization of the total discounted cumulative reward at the sacrifice of some instantaneous reward. However, the other two algorithms, which do not consider the priority issue, handle requests based on their arrival sequence. As a result, with the same number of online servers, our DQN algorithm achieves the same acceptance ratio as the Holu. Meanwhile, it ensures prior placement of higher-priority SFCs.

In phase two, to create an over-provisioning of resources scenario, we first turn on all eight servers to pre-train a DQN framework that can accept all five requests. From Fig. 3.5, we can see that, before episode 10000, DQN-agent explores the environment. As the agent becomes smarter and smarter, the acceptance ratio increases

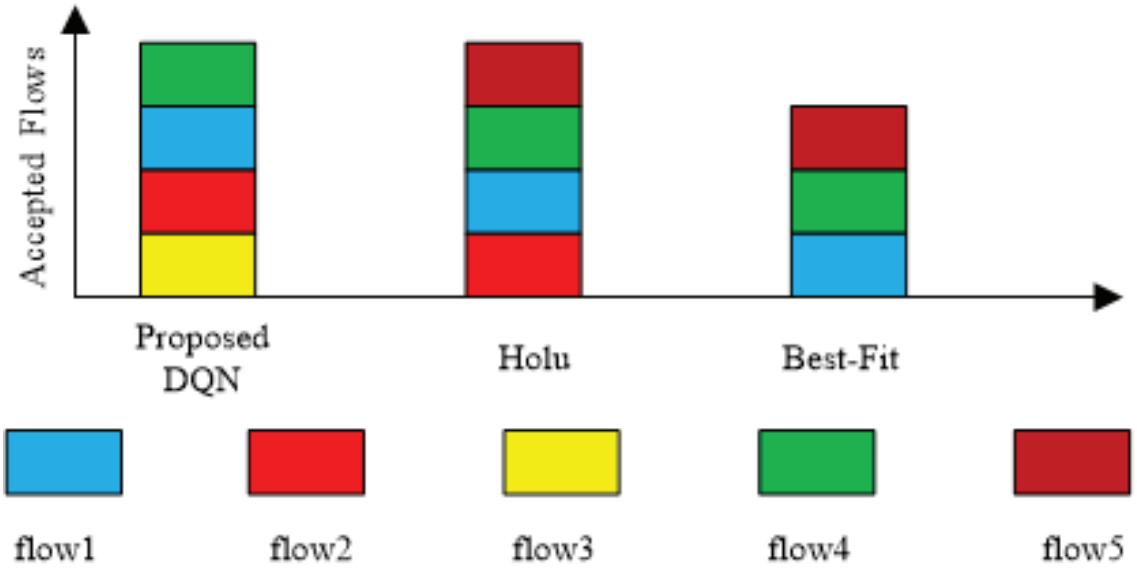

Figure 3.4 : Accepted SFCs with different priorities.

gradually. After we finish the training at episode 10000, our DQN algorithm always achieves the maximum acceptance ratio.

Then, we run our BSATL algorithm to see whether the over-provisioning problem can be resolved. From Fig. 3.6, we can see the transfer learning process during the loop of Algorithm 1. To maintain the maximum acceptance ratio, the minimum number of online servers should be five. The value 4.5 means we only utilize half of its resources for the fifth server. Compared to the initial environment with 8 servers, we significantly improve the cost-utility while maintaining the acceptance ratio. Regarding the common characteristics, we can see that all curves start from a value no less than 40% and finally converge within 1200 episodes, much faster than the rate of convergence in the initial task. This is because, in any new environment, the DQN-agent benefits from the knowledge learned in the source task. Regarding the differences, we can notice that, although the agent inherits knowledge from the initial task for both the environment with 6 servers and that with 4 servers, the curve of 6 servers starts from a higher acceptance ratio and converges faster than that of

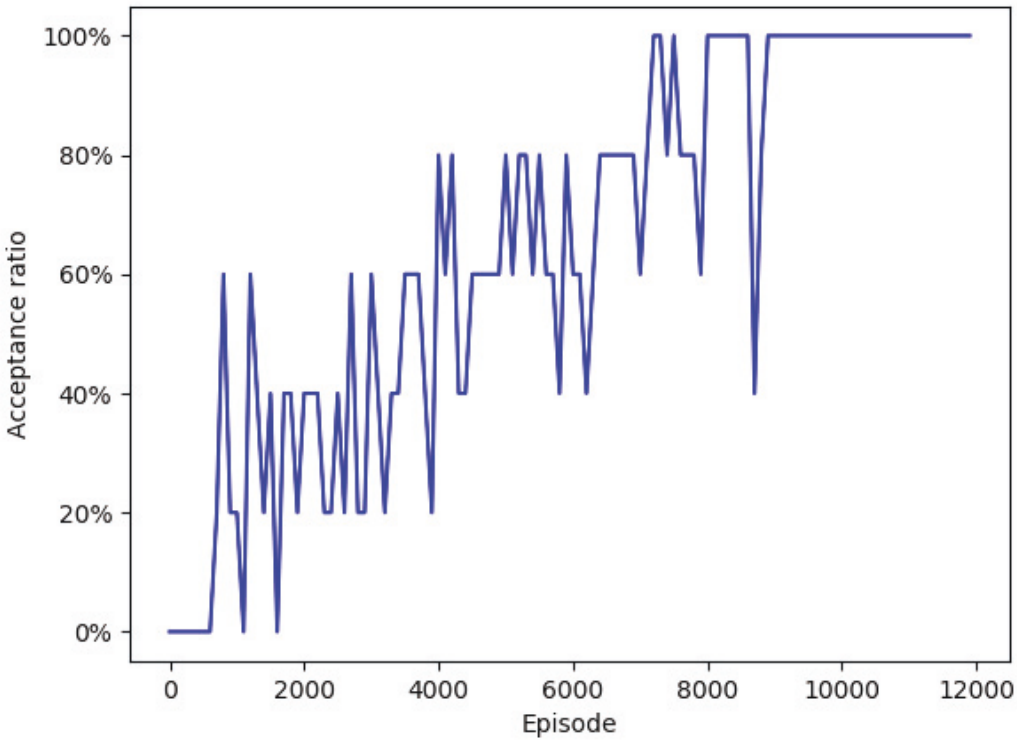

Figure 3.5 : DQN learning process of the initial task.

4 servers. This is because the environment with 6 servers shares more similarities with the initial environment. Another finding is that the acceptance ratio for the environment with 5 servers falls to 80% and then climbs to 100% very soon. The reason is that the environment of this task is highly similar to that of its source task, the environment with 6 servers.

Finally, we compare our BSATL algorithm with the Holu and the Best-Fit in terms of cost-utility under different network environments. From Fig. 3.7(a), we can see that our BSATL and the Holu occupy the same number of servers, while the Best-Fit requires more to accept all incoming requests. The reason is that the Best-Fit, which places VNFs in the smallest free partition one by one but does not consider the closeness between adjacent VNFs, needs more resources to meet the

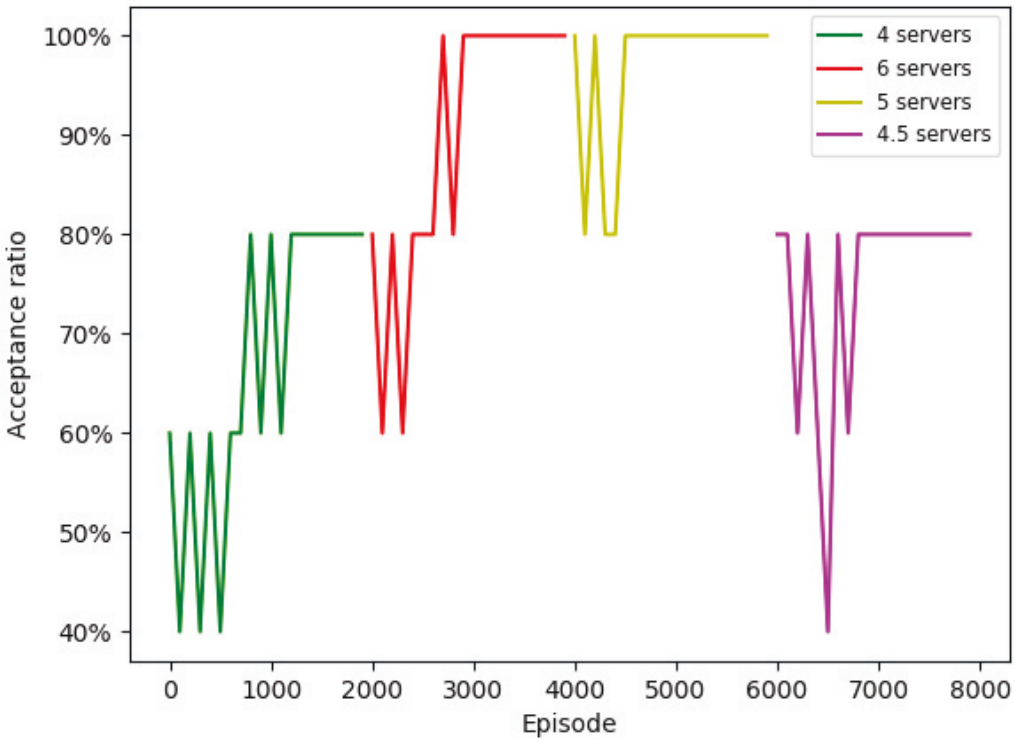

Figure 3.6 : The binary search assisted transfer learning process.

delay requirement when adjacent VNFs are far away from each other. Unlike the Best-Fit, the Holu takes the closeness between node pairs into consideration when placing VNFs, and our BSATL optimizes the VNF placement during the learning phase. From Fig. 3.7(b), we can see that the average CPU utilization of our BSATL is lower than that of the Holu. That is, our BSATL requires fewer CPU resources to accept the same number of requests. The first reason is that our BSATL flexibly reduces the VNFs size by selecting templates from 'flavors' to enhance the cost-utility. In contrast, the Holu fixes the capacity of VNFs, inevitably falling into the over-provisioning mire. The second reason is that the BSATL minimizes the cost from a global perspective, while the Holu only minimizes the cost instantaneously rather than farsightedly. In addition, the Best-Fit algorithm has the highest average CPU

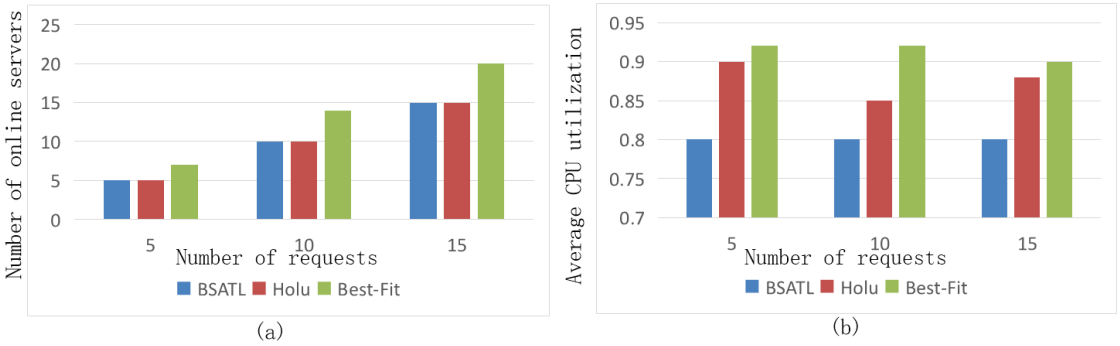

Figure 3.7 : Cost-utility comparison.

utilization and thus consumes the most resources for the aforementioned reason. In conclusion, our BSATL outperforms the other two in terms of cost-utility.

3.4 Summary

In this chapter, we leveraged the DQN algorithm to maximize the acceptance ratio and ensure prior placement of higher-priority SFCs for latency-aware network slices. We considered a multi-edge cloud scenario in which several SFC requests with different priorities are fed into the DQN-agent, and developed an intelligent policy to place SFCs. First, considering the objective, we defined our new state, action, and reward function for the DQN framework. Using simulations, we showed that DQN-agent is able to maximize the acceptance ratio and ensure prior placement of higher-priority SFCs for latency-aware SFCs with different latency requirements. Then, we proposed an algorithm – BSATL, which achieves automated and closed-loop optimization for the placement of SFCs, to resolve the over-provisioning of resources. Numerical results showed that our proposed scheme can improve cost-utility while maintaining the acceptance ratio, as expected. We achieve our objective under the classical VNE model, which assumes that the resources of each VNF and VL are known in advance. In the next chapter, we will flexibly allocate resources to each VNF and VL and realize our objective in a more complex model.

Chapter 4

SFC Embedding in an NFV-based Single DC under a Flexible Resources Allocation Model

In this chapter, we will introduce a sub-action aided DDQN (SADDQN) algorithm to place SFCs in a single DC. The most prominent innovation is that we not only place SFCs, but also allocate resources to VNFs and VLs of each SFC. For every SFC request, we first use the DDQN to choose a VNF placement (main action). Next, we employ the Dijkstra algorithm (first-phase sub-action) to find the shortest path for each pair of adjacent VNFs given the VNFs' locations. Finally, we implement the BSAGD (second-phase sub-action) to realize the SFC with the minimum cost. The joint action results in a reward that can be utilized to train the DDQN. The trained SADDQN model approximates the optimal SFC placement solution.

4.1 System Overview and Problem Formulation

4.1.1 Network Infrastructure and SFC Requests

According to the 3rd Generation Partnership Project (3GPP) view [47, 48], Network Slice Management Function (NSMF) receives requests for the allocation of network slices with certain characteristics. Accordingly, it interfaces with Management and Orchestration (MANO) to plan VNFs allocation using the Network Function Virtualization (NFV) infrastructure. In this chapter, the DDQN-agent, which takes the role of MANO, is responsible for choosing the location for VNFs of an incoming SFC request. Once the VNF placement is decided, the decision will be delivered to the SDN-controller, which accordingly derives the shortest path in

terms of hop count for any two adjacent VNFs in this VNF chain and replies the hop count results back to the DDQN-agent. With the VNF location and the hop count information, the agent assigns resources to each VNF in such a way that the SFC is realized with the minimum cost.

NFV Infrastructure

As for the infrastructure, we consider a non-blocking architecture whereby each tier is connected to the next with equal aggregate bandwidth [49]. It is realized by a fat-tree topology. We adopt a widely used fat-tree topology, depicted in Fig. 4.1. Let $\mathbb{S}$ denote the set of servers, as given by

$$\mathbb{S} = \{s_n \mid n \in \{1, 2, \ldots, |\mathbb{S}|\}\},$$

where $|\mathbb{S}|$ is the number of servers and s_n indicates the n^{th} server in the infrastructure.

SFC Requests

Let L denote the number of requests, each of which is composed of a sequence of VNFs and has a distinctive delay requirement. We assume that, within the URLLC category, there are T types of SFCs. Hence, we can prioritize the T types of SFCs and give each of them a priority value. And we define SFC request i as follows:

$$r_i = (D_i^{\text{R}}, \rho_i, F_i, \lambda_i),$$

where D_i^{R} is the end-to-end delay requirement, λ_i is the requested data rate, ρ_i is the priority value, and $F_i = \{f_{i1}, f_{i2}, \cdots, f_{iK_i}\}$ indicates the components of the SFC, a sequence of VNFs through which the traffic will travel.

Without loss of generality, we consider the following example shown in Fig. 4.1. When an SFC request (ingress→VNF1→VNF2→VNF3→VNF4→VNF5→egress) arrives, we suppose that the DDQN-agent chooses server1 to place VNF1 and VNF2, server2 to place VNF3 and VNF4, and server3 to place VNF5. Once the incoming

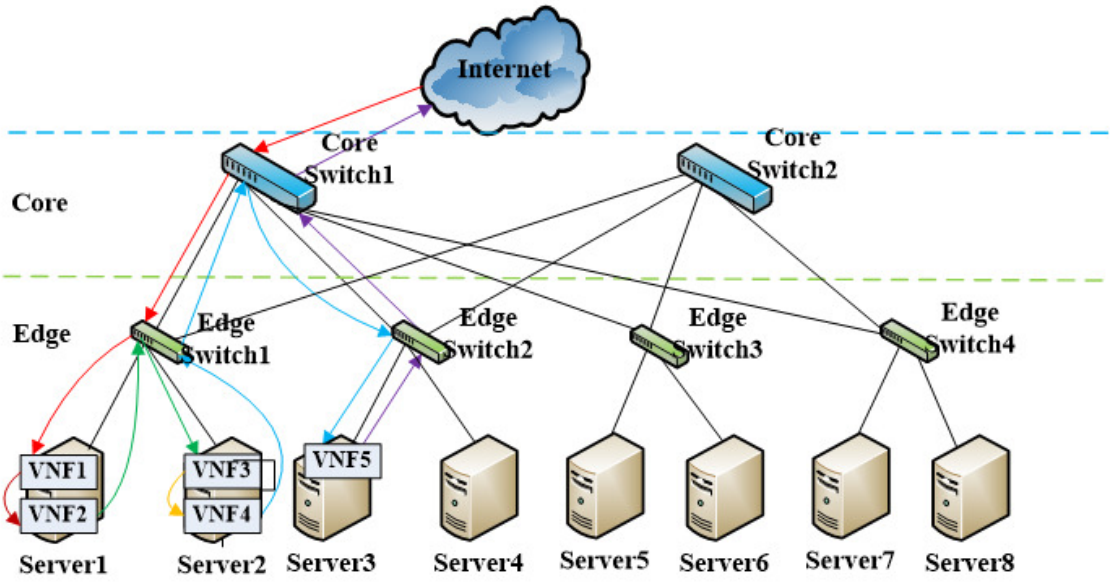

Figure 4.1 : Network infrastructure considered in this chapter.

traffic stream finishes its journey from the ingress to the egress, we obtain the real delay of this stream to see whether it meets the corresponding latency requirement.

4.1.2 Traffic Model

We model the packet arrivals of flow i at the first VNF (VNF1) as a Poisson process with arrival rate λ_i. As in [50], we define the time profile for flow i traveling through VNF1 as a two-dimensional time vector $\left[\tau_{i,1}^{\text{pro}}, \tau_{i,1}^{\text{tran}}\right]$. $\tau_{i,1}^{\text{pro}}$ denotes the CPU processing time for a packet in flow i to move through VNF1, when all CPU resources of the host server are allocated to this flow. $\tau_{i,1}^{\text{tran}}$ denotes the transmission time for a packet in flow i to go through the outgoing link of VNF1, when the whole bandwidth resources on the outgoing link of the host are allocated to flow i. Correspondingly, the rate vector $\left[C_{i,1}^{\text{pro}}, C_{i,1}^{\text{tran}}\right]$ (in the unit of packets per second) for flow i traversing VNF1 is the reciprocal of the time profile, $C_{i,1}^{\text{pro}} = 1/\tau_{i,1}^{\text{pro}}$ and $C_{i,1}^{\text{tran}} = 1/\tau_{i,1}^{\text{tran}}$. Let's consider flow i in the M/D/1 model in Fig. 4.2. If $\mu_{i,1}^{\text{tran}} > \mu_{i,1}^{\text{pro}}$, it may lead to resource waste on link transmission; If $\mu_{i,1}^{\text{tran}} < \mu_{i,1}^{\text{pro}}$, packets will accumulate in the

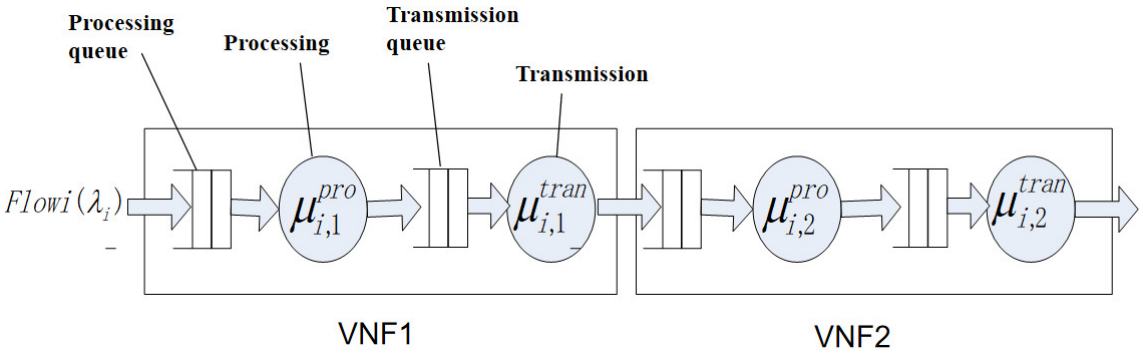

Figure 4.2 : M/D/1 queuing model.

transmission queue, resulting in an increase of the queuing delay [21]. Hence, to avoid the queuing delay of the transmission queue and bandwidth waste, we need to have

$$\frac{1}{\mu_{i,1}^{\text{pro}}} = \frac{1}{\mu_{i,1}^{\text{tran}}} = \frac{1}{\mu_{i,1}}. \tag{4.1}$$

To achieve Eq.(4.1), we have to allocate CPU and bandwidth resources to flow i passing through VNF1 according to the percentage $C_{i,1}^{\text{tran}}/C_{i,1}^{\text{pro}}$ or $C_{i,1}^{\text{pro}}/C_{i,1}^{\text{tran}}$. For simplification, we define a resources package C_{tot}, as given by

$$C_{\text{tot}} = \begin{cases} \left\{ M_{\text{tot}}, \frac{C_{i,1}^{\text{pro}}}{C_{i,1}^{\text{tran}}} W_{\text{tot}} \right\}, & C_{i,1}^{\text{tran}} > C_{i,1}^{\text{pro}} \\ \left\{ \frac{C_{i,1}^{\text{tran}}}{C_{i,1}^{\text{pro}}} M_{\text{tot}}, W_{\text{tot}} \right\}, & C_{i,1}^{\text{tran}} \leq C_{i,1}^{\text{pro}}, \end{cases} \tag{4.2}$$

where M_{tot} and W_{tot} respectively indicate the total MIPS and bandwidth of a server. When we need to allocate MIPS and bandwidth resources to a VNF, we fetch a percentage of C_{tot} for it.

4.1.3 Delay Model

When a traffic stream goes through an embedded VNF chain, the end-to-end delay is composed of the queuing delay and the processing delay on all intermediate VNFs, and the transmission delay on all links [21].

In [21], authors decoupled packet processing of all flows traveling through a VNF and thus regarded the average packet processing rate of each flow as an approximated rate. They also applied the theory to the transmission rate. Accordingly, they developed an M/D/1 queuing model for each flow to estimate the packet delay at the first VNF. Based on the analysis of packet inter-arrival time at the next VNF, they further adopted an M/D/1 queuing model to compute the average packet delay for each flow at the next VNF.

Due to the slice isolation requirements, in our work, we consider unshared VNFs [17, 51], which means two or more chains cannot share a VNF. This solution logically separates the traffic of all network sub-slices and decouples all flows. Therefore, it is reasonable for us to apply the delay model in [21] to our system.

For the first VNF, the average packet delay of flow i is expressed as

$$D_{i,1} = \frac{1}{\mu_{i,1}^{\text{pro}}} + \frac{\lambda_i}{2\left(\mu_{i,1}^{\text{pro}}\right)^2\left(1-\rho_{i,1}\right)} + \frac{1}{\mu_{i,1}^{\text{tran}}}, \tag{4.3}$$

where the first term is the average processing delay, the second indicates the average queuing delay of the processing queue [21], and the third symbolizes the average transmission delay. The utilization $\rho_{i,1}$ is given by

$$\rho_{i,1} = \frac{\lambda_i}{\mu_{i,1}^{\text{pro}}}, \tag{4.4}$$

where λ_i is the packet arrival rate and $\mu_{i,1}^{\text{pro}}$ is the processing rate.

For the $j^{\text{th}}\,(j>1)$ VNF that flow i travels through, the average packet delay is expressed as

$$D_{i,j} = \begin{cases} \frac{1}{\mu_{i,j}^{\text{pro}}} + \frac{\lambda_i}{2\left(\mu_{i,j}^{\text{pro}}\right)^2\left(1-\rho_{i,j}\right)} + \frac{1}{\mu_{i,j}^{\text{tran}}}, & \mu_{i,j-1}^{\text{tran}} > \mu_{i,j}^{\text{pro}} \\ \frac{1}{\mu_{i,j}^{\text{pro}}} + \frac{1}{\mu_{i,j}^{\text{tran}}}, & \mu_{i,j-1}^{\text{tran}} \le \mu_{i,j}^{\text{pro}}, \end{cases} \tag{4.5}$$

where $1/\mu_{i,j}^{\text{pro}}$ is the average processing delay, $\lambda_i/2\left(\mu_{i,j}^{\text{pro}}\right)^2\left(1-\rho_{i,j}\right)$ is the average queuing delay of the processing queue, and $1/\mu_{i,j}^{\text{tran}}$ is the transmission delay.

In Eq.(4.5), if the transmission rate of the $(j-1)^{\text{th}}$ VNF is lower than or equal to the processing rate of the j^{th} VNF, then there will be no queuing delay of the processing queue on the j^{th} VNF, as given in the second segment. Otherwise, in the first segment, we have the average queuing delay of the processing queue based on M/D/1 queue theory [21].

As in Eq.(4.1), we let the processing rate equal the transmission rate to eliminate the queuing delay of the transmission queue:

$$\frac{1}{\mu_{i,j}^{\text{pro}}} = \frac{1}{\mu_{i,j}^{\text{tran}}} = \frac{1}{\mu_{i,j}}, \qquad j > 1. \tag{4.6}$$

If the j^{th} and the $(j+1)^{\text{th}}$ VNFs are located on the same server, traveling from the j^{th} VNF to the $(j+1)^{\text{th}}$ VNF, the flow will suffer no forwarding delay caused by switches. Otherwise, the flow may suffer delay from traveling through links and switches. In our work, as in [21] and [52], to maximize resource utilization, we assume that the resources allocated to flow i from these switches ensure that the transmission rate of the j^{th} VNF is equal to that of any switch. Therefore, the queuing delay, either of the transmission queue or of the processing queue, does not need to be considered on these switches. Consequently, to go from the j^{th} VNF to the $(j+1)^{\text{th}}$ VNF, the total packet delay for flow i passing through n_j switches is given by

$$D_{i,j}^{\text{f}} = \frac{2n_j}{\mu_{i,j}^{\text{tran}}}, \qquad 1 \leq j \leq K_i. \tag{4.7}$$

In general, the average end-to-end delay of flow i traveling through a VNF chain, which contains K_i intermediate VNFs, is the total of the average delay for packets to move through K_i VNFs and the average delay caused by all links and switches [21], as defined by

$$D_i = \sum_{j=1}^{K_i} D_{i,j} + \sum_{j=1}^{K_i} D_{i,j}^{\text{f}}. \tag{4.8}$$

4.1.4 Cost Model

Operational Cost

We define the operational cost as in [17], and it is expressed as follows:

$$c_{\text{ope}} = \sum_{i=1}^{L} c_{\text{ope},i} = \sum_{i=1}^{L} \alpha \times 250 \times N_i, \tag{4.9}$$

where α is the parameter that transfers the power to a monetary value, N_i is the number of newly activated servers for accommodating SFC i, and 250 is the power of each active server.

Traffic Cost

In an SFC, a VNF needs to send packets to the next one through the VL, which is embedded in the physical links. So the hop count between two adjacent VNFs depends on the network topology [53]. We use $h\left(v_j, v_{j+1}\right)$ to indicate the hop count between two adjacent VNFs.

We define the network traffic cost in the same way as in [32] and [53], calculating the traffic cost of two adjacent VNFs according to the hop count between them and the bandwidth of the VL. Thus, the total traffic cost can be defined as

$$\begin{aligned} c_{\text{tran}} &= \sum_{i=1}^{L} c_{\text{tran},i} \\ &= \beta \sum_{i=1}^{L} \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\text{tran}}} \times W_{\text{tot}} \times h\left(v_j, v_{j+1}\right), \end{aligned} \tag{4.10}$$

where

$$h\left(v_j, v_{j+1}\right) = \sum_{n=1}^{|\mathbb{S}|} \sum_{n'=1}^{|\mathbb{S}|} x_{i,j}^{n} \times x_{i,j+1}^{n'} \times h\left(s_n, s_{n'}\right). \tag{4.11}$$

In Eq.(4.10), β is the parameter that transfers the traffic cost to a monetary value. In Eq.(4.11), the binary variable $x_{i,j}^{n}$ indicates whether the j^{th} VNF of the i^{th} SFC nests on the n^{th} server, $x_{i,j+1}^{n'}$ denotes whether the $(j+1)^{\text{th}}$ VNF of the i^{th} SFC nests on the n'^{th} server, and $h\left(s_n, s_{n'}\right)$ is the hop count between server n and server n'.

Server Cost

Compared with memory and capacity, CPU's computational power is much more important for latency-sensitive SFCs. Therefore, as for server cost, we only consider CPU resources in Million Instructions Per Second (MIPS), and the total server cost can be written as

$$c_{\text{ser}} = \sum_{i=1}^{L} c_{\text{ser},i} = \gamma \sum_{i=1}^{L} \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\text{pro}}} \times M_{\text{tot}}, \tag{4.12}$$

where γ is the parameter that transfers the server cost to a monetary value.

4.1.5 Problem Formulation

In this part, we formulate the objective as an optimization problem. The purpose is to maximize the number of accepted higher-priority SFC requests while minimizing the total cost, under infrastructure resources constraints.

The weighted sum of the accepted requests can be defined as:

$$u = \sum_{i=1}^{L} u_i = \sum_{i=1}^{L} \rho_i \times f(D_i^R - D_i). \tag{4.13}$$

In the function above,

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0, \end{cases} \tag{4.14}$$

D_i^{R} and D_i respectively denote the latency requirement and the real latency of the i^{th} SFC, and ρ_i indicates the priority of SFC i.

The total cost can be expressed as follows:

$$c = \sum_{i=1}^{L} c_i = \sum_{i=1}^{L} (c_{\text{ser},i} + c_{\text{tran},i} + c_{\text{ope},i}). \tag{4.15}$$

Hence, the optimization problem can be formulated as

$$(P1): \max_{x_{i,j}^n, \mu_{i,j}} \eta_1 u - \eta_2 c$$

$$
\text{s.t.}\begin{cases}
\displaystyle\sum_{i=1}^{L}\sum_{j=1}^{K_i} x_{i,j}^{n}\frac{\mu_{i,j}}{C_{i,j}^{\text{pro}}} \le 1,\ n=1,2,\ldots,|\mathbb{S}| & \text{(4.16a)}\\
\displaystyle\sum_{i=1}^{L}\sum_{j=1}^{K_i} x_{i,j}^{n}\frac{\mu_{i,j}}{C_{i,j}^{\text{tran}}} \le 1,\ n=1,2,\ldots,|\mathbb{S}| & \text{(4.16b)}\\
\displaystyle\sum_{n=1}^{|\mathbb{S}|} x_{i,j}^{n} = 1,\ i=1,2,\ldots,L, j=1,2,\ldots,K_i & \text{(4.16c)}\\
x_{i,j}^{n}\in\{0,1\}, & \\
i=1,2,\ldots,L, j=1,2,\ldots,K_i, n=1,2,\ldots,|\mathbb{S}| & \\
\mu_{i,j}\in\mathbb{N}_+, i=1,2,\ldots,L, j=1,2,\ldots,K_i. &
\end{cases}
$$

In the objective function above, η_1 and η_2 are the weights of two objectives. Since the objective of maximizing the number of accepted higher-priority SFCs is more important, we assign a higher weight to it in our paper.

In the constraints above, the binary variable $x_{i,j}^{n}$ indicates whether the j^{th} VNF of the i^{th} SFC nests on the n^{th} server. For any server in the infrastructure, the total of the MIPS allocated to the embedded VNFs cannot exceed the CPU capacity of the server, so we have Eq.(4.16a), and the total of the bandwidth allocated to the embedded VNFs cannot exceed the maximum bandwidth of the server's outgoing link, so we have Eq.(4.16b). Each VNF can be deployed at only one server, so we have Eq.(4.16c).

It can be difficult to solve this optimization problem using heuristic approaches, because heuristics can hardly trace the dynamics of the network. Alternatively, learning-based techniques can be utilized to learn the network dynamics, such as the arrival pattern of various SFCs and the changing topology of the infrastructure, and solve such a problem. The goal of the learning technique is to learn a policy that determines what action to take in each environment state. In the next section, we

will introduce a model derived from Double Deep Q-Network (DDQN) [43] for joint SFC placement and resource allocation problem regarding the latency requirement.

4.2 Proposed SFC Placement and Resource Allocation Scheme

4.2.1 Overview of Our Proposed SADDQN Algorithm

For our SFC embedding problem, an agent handles the SFC requests one after another. To achieve its objective, the agent learns how to behave in the environment, the state of which consists of the available resources provided by the infrastructure and the characteristics of the request to be processed, by performing actions and seeing the results. Specifically, for a single SFC request, the agent sequentially fulfills three tasks. First, it chooses a VNF placement (main action), the locations of all the VNFs in this SFC. Second, it employs the Dijkstra algorithm (the first-phase sub-action) to derive the optimal path traversing the VNFs placed by the first task. Third, it utilizes our proposed resource allocation algorithm – BSAGD (the second-phase sub-action) to allocate resources to all VNFs in such a way that the SFC request is realized with the minimum cost.

Performing the joint action (VNF placement, its corresponding optimal path and resource allocation solution) for the current request to the environment, the agent will receive a reward, which will be utilized to improve its action (VNF placement) for subsequent requests. Meanwhile, the state of the environment will be updated, and the action for the next request will be determined. Hence, we can model the SFC embedding for all requests as a Markov Decision Process (MDP). Because one VNF placement has only one corresponding optimal path and resource allocation solution, we represent the MDP action space as the possible VNF placements of a single SFC request. Since we aim to maximize the number of accepted higher-

priority SFCs while minimizing the total cost, we prioritize those requests based on their latency requirements and define the reward function of the MDP based on priority and resource cost. The details of the state, action, and reward function in the context of our problem will be provided in Sections 4.2.2, 4.2.3, and 4.2.4.

We propose an SADDQN algorithm to resolve our SFC embedding problem. A general sketch of the SADDQN used in our scheme is shown in Fig. 4.3. The state, indicated by a vector, is fed into the evaluation neural network, which outputs a vector of Q-values, with each indicating the expected discounted cumulative reward of a corresponding action (VNF placement). At time step t, the Q-value of performing action a_t under state s_t based on policy π is given by

$$Q^{\pi}(s_t, a_t) = E(\sum_{i=t}^{L} \gamma^{(i-t)} R(s_i, a_i) | s_t, a_t). \tag{4.17}$$

The objective of the agent is to learn a policy that maximizes the expected return $Q^{\pi}(s_t, a_t)$. Once the optimal policy is achieved, given a state, the agent can find the best action by taking the largest Q-value from the output vector. At the initial stage of the training, the weights of the evaluation neural network are random, and thus the policy is poor. For each given state, the maximum Q-value may not account for the best VNF placement. Hence, we continuously feed the framework with requests, and the agent iteratively optimizes the neural networks. Specifically, at time step t, by performing the joint action (VNF placement and its corresponding optimal sub-action) for a request to the environment, the agent receives a reward, which will be used to conduct the back-propagation process and update the weights of the evaluation network. When the next request comes, this training process will be iterated in the next time step. The loop ends until the weights of neural networks converge. Then, given a state, the agent can choose the optimal VNF placement according to the output Q-values of the evaluation network. And the corresponding optimal routing and resource allocation solution can be obtained from the Routing and BSAGD modules, given the VNF placement. The iterative training process is

summarized in Algorithm 2.

Algorithm 2 Sub-action Aided DDQN Algorithm

Input: $\lambda_1, \lambda_2, \ldots, \lambda_L, D_1^R, D_2^R, \ldots, D_L^R$

Initializing replay memory D to capacity N
Initializing estimation neural network with random weights θ
Initializing target neural network with weights θ^-
$\theta^- = \theta$
for e $= 1, 2, \ldots, I$ **do**
    Initializing state of the environment
    for $t = 1, 2, \ldots, L$ **do**
        Workflow of each learning step
    end for
end for

The workflow for each learning step can be summarized as follows:

1. Step 1: The agent observes the state (s_t) of the environment.

2. Step 2: The agent chooses a main action (a_t), i.e., the VNF placement, randomly with probability ϵ or according to the evaluation network with probability $1 - \epsilon$. Next, it delivers a_t (the chosen VNF placement) to the Routing module, which derives an optimal path ($sa_t^{'}$) traversing all the VNFs based on the VNF sequence.

3. Step 3: a_t and $sa_t^{'}$ (the chosen VNF placement and its corresponding optimal path) are fed into the BSAGD module to obtain a resource allocation solution ($sa_t^{''}$) that realizes the SFC with the minimum cost.

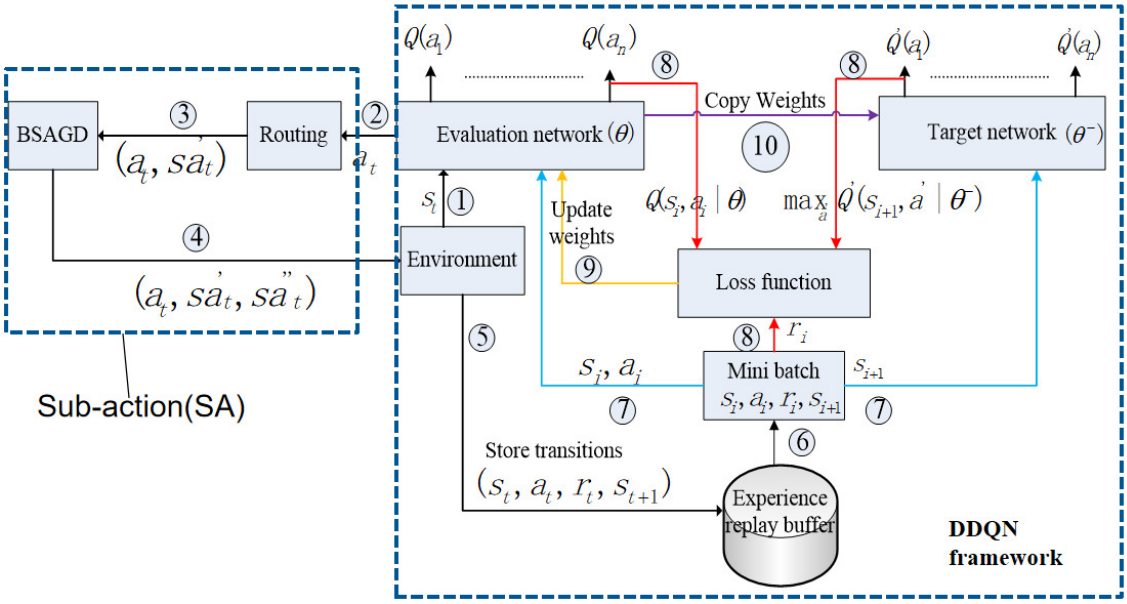

Figure 4.3 : Flowchart of the SADDQN algorithm.

4. Step 4: a_t, $sa_t^{'}$ and $sa_t^{''}$ (the chosen VNF placement, its corresponding optimal routing and resource allocation solution) are jointly performed to the environment.

5. Step 5: The agent gets a reward (r_t) from the environment, and the experience tuple (s_t, a_t, r_t, s_{t+1}) is stored into an experience replay buffer. Here, we only need to add the main action (a_t) into the experience tuple, because we have only one corresponding optimal sub-action for any main action.

6. Step 6: To train the DDQN framework, a mini-batch of N tuples are uniformly sampled from the experience replay buffer.

7. Step 7: For tuple i, s_i and a_i are fed into the evaluation network (θ), while s_{i+1} is fed into the target network (θ^-).

8. Step 8,9: A loss function is created, and the weights of the evaluation network

are updated by minimizing the loss function:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i|\theta))^2, \tag{4.18}$$

where

$$y_i = r_i + \gamma max_{a'} Q'(s_{i+1}, a'|\theta^-). \tag{4.19}$$

9. Step 10: The weights of the target network are updated by copying the weights of the evaluation network every Z time steps.

4.2.2 State

We define the state as a vector, including the latency requirement, VNF chain, requested traffic of an SFC request, and the remaining resources of each server. In our environment, the number of intermediate VNFs in SFC i is K_i. Then the characteristic vector of SFC i can be given by

$$\boldsymbol{r}_i = \begin{bmatrix} \lambda_i & D_i^R & I_1 & I_2 & \cdots & I_{K_i} \end{bmatrix},$$

where λ_i and D_i^{R} denote the requested traffic and the delay requirement of the i^{th} SFC, and I_1, I_2, ..., I_{K_i} indicate the ID of the first, the second, ..., the K_i^{th} VNF in the VNF chain. As for the network infrastructure, the characteristic vector of the n^{th} server can be expressed as

$$\boldsymbol{s}_n = \begin{bmatrix} m_n & w_n \end{bmatrix},$$

where m_n and w_n denote the remaining MIPS and bandwidth of the n^{th} server.

As a result, the state vector of the environment is written as

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{r}_i & \boldsymbol{s}_1 & \cdots & \boldsymbol{s}_{|\mathbb{S}|} \end{bmatrix}.$$

4.2.3 Action

Main Action (VNF Placement)

We use a K_i columns row vector $\boldsymbol{a}_i$ to symbolize a VNF placement for the i^{th} SFC request. Element $a_{i,j}$ symbolizes on which server the j^{th} VNF of the i^{th} SFC nests. For the SFC request in Fig. 4.1, we use a 5 columns row vector, $\boldsymbol{a}_1 = \begin{bmatrix} 1 & 1 & 2 & 2 & 3 \end{bmatrix}$, to indicate its main action. This vector indicates that the first and the second VNFs are located on server1, the third and fourth VNFs are located on server2, and the fifth VNF is located on server3.

First-Phase Sub-Action

With a given VNF placement, to minimize the resource cost, we need to find the shortest path traversing the VNF chain. Therefore, the Dijkstra algorithm, which derives the shortest path in terms of hop count, is adopted to connect adjacent VNFs. Specifically, we run the Dijkstra algorithm from the ingress node to the first VNF, from the first VNF to the second one, and so on, until the egress node.

Second-Phase Sub-Action (Binary Search Assisted Gradient Descent (BSAGD))

For an SFC, after determining the locations for all VNFs and the corresponding optimal path traversing the VNF chain, we need to allocate resources to all VNFs such that this SFC's latency requirement is satisfied with the minimum cost. In general, to find the minimum cost for realizing SFC i, there are two steps for us to take. In the first step, we prove that the minimum delay $D_i^{\min}(c_i)$ is a monotone decreasing function of variable c_i. In the second step, we use our BSAGD algorithm, a binary search algorithm in which gradient descent is iterated, to search for the minimum cost and the corresponding resources of each VNF ($\mu_{i,j}$) for meeting the delay requirement of SFC i. Because $\mu_{i,j}$ is an integer number, we adopt the relax-

and-round mechanism to get the optimal integer solution $(\widetilde{\mu}_{i,j}, j = 1, \ldots, K_i)$ for realizing SFC i.

Based on Eq.(4.1), (4.2), (4.4), (4.5), (4.6), (4.7), and (4.8), the problem of minimizing the average delay with limited cost c_i can be formulated as follows:

$$
\begin{aligned}
(P2): \min_{\boldsymbol{\mu}_i} D_i(\boldsymbol{\mu}_i) = & \frac{\lambda_i}{2\left(\mu_{i,1}\right)^2\left(1-\frac{\lambda_i}{\mu_{i,1}}\right)} + \sum_{j=1}^{K_i} \frac{2}{\mu_{i,j}} \\
& + \sum_{j=1}^{K_i} \frac{2(h(v_j, v_{j+1}) - 1)}{\mu_{i,j}} g(h(v_j, v_{j+1})) \\
& + \sum_{j=2}^{K_i} \frac{\lambda_i}{2\left(\mu_{i,j}\right)^2\left(1-\frac{\lambda_i}{\mu_{i,j}}\right)} g(\mu_{i,j-1} - \mu_{i,j})
\end{aligned}
$$

$$
\text{s.t.} \begin{cases}
\mu_{i,j} \leq \mu_{i,j+1} \text{ or } \mu_{i,j+1} \leq \mu_{i,j},\ j = 1, \ldots, K_i - 1 & \text{(4.20a)} \\
\gamma \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\text{pro}}} \times M_{\text{tot}} + \beta \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\text{tran}}} \times W_{\text{tot}} & \\
\times h\left(v_j, v_{j+1}\right) + \alpha \times 250 \times N_i = c_i & \text{(4.20b)} \\
\sum_{j=1}^{K_i} x_{i,j}^n \frac{\mu_{i,j}}{C_{i,j}^{\text{pro}}} \times M_{\text{tot}} \leq m_n, & \\
n = 1, \ldots, |\mathbb{S}| & \text{(4.20c)} \\
\sum_{j=1}^{K_i} x_{i,j}^n \frac{\mu_{i,j}}{C_{i,j}^{\text{tran}}} \times W_{\text{tot}} \leq w_n, & \\
n = 1, \ldots, |\mathbb{S}|. & \text{(4.20d)}
\end{cases}
$$

In the objective function above,

$$
g(x) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0. \end{cases} \tag{4.21}
$$

In the constraints above, Eq.(4.20a) is a possible condition of $\mu_{i,j}$ and $\mu_{i,j+1}$, $j = 1, \ldots, K_i - 1$. We provide VNF chain i with limited cost c_i, so we have Eq.(4.20b). In Eq.(4.20b), the first term is the total CPU cost of all VNFs. The second term is the traffic cost, which depends on the hop count of each pair of adjacent VNFs. The third term is the operational cost, a linear function of newly activated servers.

For any server in the infrastructure, the total of MIPS allocated to the embedded VNFs cannot exceed the remaining CPU MIPS of the server, so we have Eq.(4.20c), and the total of the bandwidth allocated to the embedded VNFs cannot exceed the remaining bandwidth of the server's outgoing link, so we have Eq.(4.20d).

Proposition 1. *The minimum end-to-end delay expression in terms of variable c_i is a monotone decreasing function of c_i.*

Proof. The details can be seen in Appendix A.

Proposition 2. *P2 is a convex optimization problem.*

Proof. The details can be seen in Appendix B.

Since P2 is a convex optimization problem, given a constant cost C_i, we can use gradient descent to find the minimum average delay $D_i^{\min}(C_i)$. Furthermore, $D_i^{\min}(c_i)$ is a monotone decreasing function of variable c_i. Therefore, for realizing SFC i, we can use a binary search algorithm in which gradient descent is iterated to find the minimum cost and the corresponding optimal resource allocation for all VNFs.

Our BSAGD algorithm is summarized in Algorithm 3.

Given a VNF placement and a condition of $\mu_{i,j}$ and $\mu_{i,j+1}$, $j = 1, \ldots, K_i - 1$, we can obtain the unit cost C_{unit}, the maximum cost ($C_i^{\max}$), and the minimum cost ($C_i^{\min}$). $C_i^{\max}$ and $C_i^{\min}$ are used as the initial two ends of the binary search algorithm. Let tentative maximum cost $t_i^{\max} = C_i^{\max}$ and tentative minimum cost $t_i^{\min} = C_i^{\min}$. If $C_i^{\max}$ cannot obtain a delay shorter than or equal to requirement D_i^{R}, there is no solution for this placement under this condition. Otherwise, we store $C_i^{\max}$ in verified minimum cost $v_i^{\min}$ and update the rate vector $\boldsymbol{\mu}_i$. Then, we go to the middle point if $C_i^{\max}$ has a delay shorter than D_i^{R}. If the middle point cannot obtain a delay shorter than or equal to D_i^{R}, we update $t_i^{\min}$ and $t_i^{\max}$ accordingly and continue searching in the upper half. Otherwise, we store this middle point in $v_i^{\min}$

Algorithm 3 BSAGD Algorithm

Input: $C_i^{\min}, C_i^{\max}, C_{\text{unit}}$

Output: $v_i^{\min}$, $\boldsymbol{\mu}_i$

$t_i^{\min} = C_i^{\min}, v_i^{\min} = 0, t_i^{\max} = C_i^{\max}$

$i = 1, \boldsymbol{\mu}_i = \mathbf{0}$

while $v_i^{\min} - t_i^{\max} > C_{\text{unit}}$ or $i = 1$ **do**

  $i = i + 1$

  Substituting $t_i^{\max}$ into P2 and using gradient descent to find

  the minimum delay and the corresponding $\boldsymbol{\mu}_i$

  if $D_i^{\min}(t_i^{\max}) < D_i^{\text{R}}$ **then**

    $v_i^{\min} = t_i^{\max}$

    $t_i^{\max} = \frac{t_i^{\max} + t_i^{\min}}{2}$

    update $\boldsymbol{\mu}_i$

  else if $D_i^{\min}(t_i^{\max}) > D_i^{R}$ **then**

    $t_i^{\min} = t_i^{\max}$

    $t_i^{\max} = \frac{t_i^{\min} + v_i^{\min}}{2}$

  else

    $v_i^{\min} = t_i^{\max}$

    update $\boldsymbol{\mu}_i$

  end if

end while

return $v_i^{\min}$, $\boldsymbol{\mu}_i$

and update $\boldsymbol{\mu}_i$. If the delay of this middle point is shorter than D_i^{R}, we update $t_i^{\max}$ accordingly and continue searching in the lower half. When the difference between $v_i^{\min}$ and $t_i^{\max}$ is less than the unit cost C_{unit} or the current $v_i^{\min}$ has a delay equal to D_i^{R}, there is no need to continue looping, and we return $v_i^{\min}$ and $\boldsymbol{\mu}_i$ for realizing SFC i. For any possible condition of $\mu_{i,j}$ and $\mu_{i,j+1}$, $j = 1, \ldots, K_i - 1$, we can obtain a $v_i^{\min}$ and the corresponding $\boldsymbol{\mu}_i$. The minimum of these verified minimum cost values is the minimum cost for realizing SFC i, given a VNF placement.

4.2.4 Reward

A performance model needs to be created to assess whether the latency requirement has been achieved with the given resources (MIPS, bandwidth) and the current workload. In our work, we first leverage the simulation tool CloudSimSDN-NFV [44] to obtain the latency of each SFC. Then, we issue a penalty for an SFC whose latency requirement is not satisfied and a reward for a successful accommodation. For the i^{th} SFC, the reward function regarding delay is defined as

$$R_{\mathrm{delay},i} = \begin{cases} \rho_i, & D_i \leq D_i^R \\ 0, & D_i > D_i^R, \end{cases} \tag{4.22}$$

where D_i is the real delay of the i^{th} SFC we obtain from CloudSimSDN-NFV.

Regarding the cost, we define reward functions for operational cost, server cost, and traffic cost of SFC i as follows:

$$R_{\mathrm{ope},i} = \alpha \times 250 \times N_i, \tag{4.23}$$

$$R_{\mathrm{ser},i} = \gamma \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\mathrm{pro}}} \times M_{\mathrm{tot}}, \tag{4.24}$$

$$R_{\mathrm{tran},i} = \beta \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\mathrm{tran}}} \times W_{\mathrm{tot}} \times h\left(v_j, v_{j+1}\right). \tag{4.25}$$

As in [17], α, β, and γ can be adjusted to change the impact factor of each category.

Since our goal is to maximize the number of accepted higher-priority SFCs while minimizing the total cost, the reward function of SFC i is expressed as a weighted sum of delay reward and cost reward functions

$$R_i = \eta_1 R_{\text{delay},i} - \eta_2(R_{\text{ope},i} + R_{\text{ser},i} + R_{\text{tran},i}). \tag{4.26}$$

Maximizing the acceptance ratio is our top priority; therefore, we give a greater weight to delay rewards.

4.3 Simulation

4.3.1 Simulation Setup

In this section, we evaluate the performance of the proposed SADDQN algorithm regarding the admission ratio and cost efficiency under different network sizes. We use CloudSimSDN-NFV, an NFV environment simulation tool extended from CloudSimSDN [54] and CloudSim [55], as our simulation framework. To merge the DDQN algorithm into CloudSimSDN-NFV, we import deeplearning4j and nd4j written in JAVA. As for the infrastructure, we consider fat-tree, a widely used network topology for data centers. Two scenarios, i.e., 8-Node Fat-Tree and 16-Node Fat-Tree, are utilized to assess our algorithms. As in [46], we set the bandwidth of each link to 1Gbps. The total MIPS of each server is 3000. As for the cost model, we set $\alpha = 1$ \$/W, $\gamma = 0.1$ \$/MIPS, $\beta = 0.1$ \$/Mbps.

In our experiment, we assume that there are three types of latency-sensitive SFCs. By referring to the characteristics of SFCs in [21], we provide the characteristics of our SFCs in Table 4.1. Additionally, the rate vectors of all VNFs for three different packet types are listed in Table 4.2.

The neural network is designed with the following parameters. Adam [56] is employed to learn the neural network hyperparameters. The learning rate is 0.005, and

Table 4.1 : Overview of three SFC types.

SFC type	Delay requirement	SFC	Packet size (bits)	Data rate (packets/s)
SFC1	20 ms	VNF1→VNF2→VNF3	4000	[100-900]
SFC2	30 ms	VNF1→VNF2→VNF4 →VNF6	16000	[100-200]
SFC3	40 ms	VNF1→VNF2→VNF5 →VNF2→VNF1	20000	[1000-2000]

Table 4.2 : Rate vectors (packets/s) of all VNFs for three packet types.

Packet size / VNF type	4000 bits	16000 bits	20000 bits
VNF1	[125000,250000]	[125000,62500]	[80000,50000]
VNF2	[125000,250000]	[125000,62500]	[80000,50000]
VNF3	[125000,250000]	–	–
VNF4	–	[125000,62500]	–
VNF5	–	–	[80000,50000]
VNF6	–	[125000,62500]	–

the discount factor is 0.99. The parameters of the target network are updated every 100 episodes. The batch size is 64. We utilize a fully connected Deep Neural Network (DNN), which adopts the Rectified Linear Unit (ReLu) [57] as the activation function in hidden layers, while the output layer is connected to a linear activation function [58].

Regarding the traffic, we assume that, for any SFC, the packet arrival process can be modeled as a Poisson process, with parameter λ indicating the average arrival rate. According to traffic history, λ of an SFC request is uniformly distributed in the data rate interval of this SFC type in Table 4.1.

Our proposed SADDQN algorithm consists of an offline training phase and an online testing phase. In the training phase, we randomly choose an SFC type and its requested data rate from Table 4.1 to create an SFC request. We continuously feed our SADDQN framework with the SFC requests until the weights of neural networks converge.

During the online testing phase, we can run the trained SADDQN model online to optimize the VNF placement, as well as its corresponding route and resource allocation, for any given state.

4.3.2 Algorithms to Compare

Standard DDQN

For comparison purposes, we have designed a standard DDQN algorithm based on a popular assumption: the required resources of each VNF have been specified in SFC requests. The only difference between the standard DDQN and our proposed SADDQN is that the standard one assumes that the capacity of each VNF has been specified by customers, but our proposed SADDQN allocates resources to each VNF for customers. For the sake of fairness, for any algorithm, we assume that VNFs are embedded in containers, for which CPU requests can be as small as 1/1000 of the total CPU resources. For the standard DDQN, according to the number of vCPUs of those 'flavors', we have $\frac{1}{64}$, $\frac{1}{32}$, $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$, and the whole of the CPU resources of a server for any container/VNF to choose in our simulation.

Holu

The Holu algorithm has been illustrated in the previous chapter. Although the power consumption model of Holu is different from the cost model of our proposed algorithm, the objectives of the two algorithms (saving the cost while satisfying the delay constraint of each SFC) are the same. Therefore, it is reasonable to run

the Holu in our system model and compare it with our proposed SADDQN. In our simulation, as in [20], we fix the CPU capacity for each VNF for the Holu algorithm. Based on the composition of three SFCs, we set the CPU capacity of VNF1 and VNF2 to $\frac{1}{4}$ of a server's CPU capacity and that of the other VNFs to $\frac{1}{64}$.

4.3.3 Simulation Results

We compare our proposed SADDQN, the standard DDQN and the Holu with respect to the acceptance ratio, cost-utility and average end-to-end delay. To see how granularity in resource allocation will impact the performance of the standard DDQN, we set the CPU parameter of the minimum 'flavor' to $\frac{1}{32}$ (DDQN1) and $\frac{1}{64}$ (DDQN2). In other words, on a server, the minimum percentage of CPU resource that DDQN1 can allocate to a VNF is $\frac{1}{32}$ while it is $\frac{1}{64}$ for DDQN2.

From Fig. 4.4, we can see that, with the increasing number of requests, SADDQN always obtains the highest acceptance ratio among all algorithms in both topologies. That is because our proposed SADDQN algorithm uses the resource optimization algorithm (BSAGD) to allocate resources to VNFs, achieving finer granularity than any other algorithm. The standard DDQN, which can only choose the CPU capacity for VNFs based on 'flavors', excessively provisions higher-priority SFCs and thus does not have enough resources to accommodate lower-priority SFCs when we increase the number of SFC requests. DDQN2 outperforms DDQN1 because of the CPU parameter of the minimum 'flavor': the smaller it is, the more requests the standard DDQN accepts. The Holu algorithm fixes the CPU capacity for each type of VNF and allows different SFCs to share VNFs. To some extent, the sharing policy alleviates the over-provisioning issue. However, the problem still exists, and the severity level depends on the sharing percentage of each VNF. For instance, if we initiate a new instance of some VNF for an SFC request and no subsequent SFCs share this VNF instance, its resource utility will degrade.

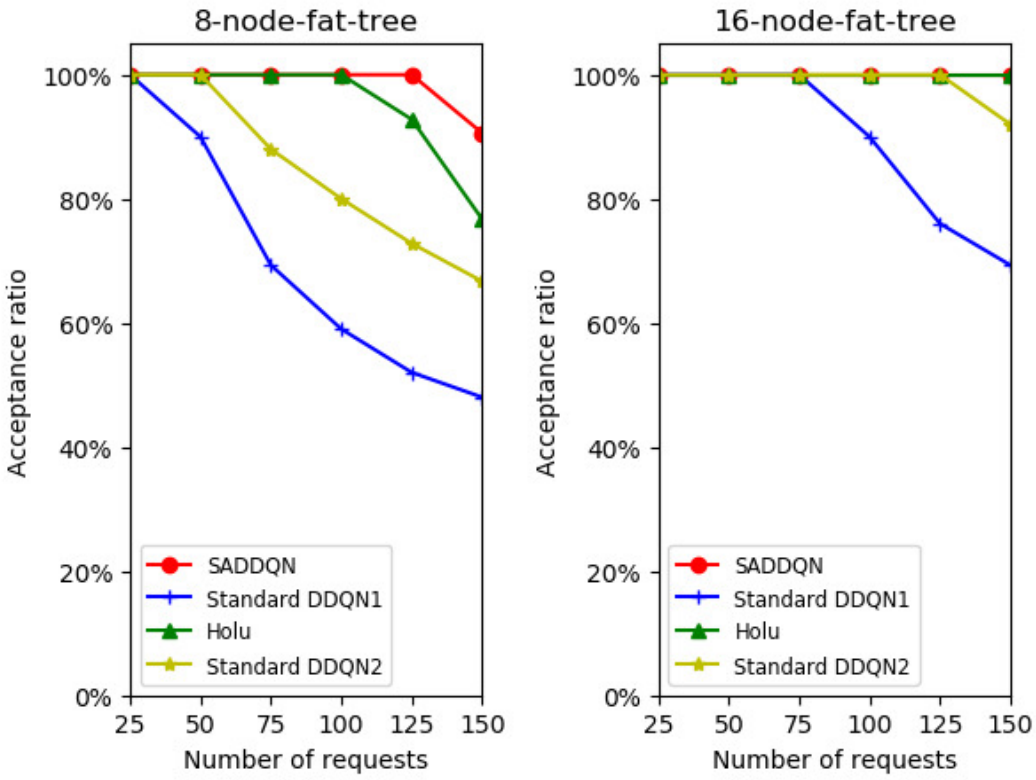

Figure 4.4 : Acceptance ratio comparison.

Fig. 4.5 indicates the accommodation results of different SFCs. It depicts the situation of 8-node fat-tree topology with the number of requests being 150. We can see that DDQN-based algorithms all ensure priority allocation for higher-priority SFCs when the resource is insufficient to accept all requests. Of all the requests accepted by SADDQN, 50 are of the highest priority, 50 are of the second highest priority, and only 36 have the lowest priority. That is to say, 14 lowest-priority requests are discarded because of a lack of resources. DDQN1 accepts 50 highest-priority requests and 22 second-highest-priority requests, while DDQN2 does 50 for both priorities. Meanwhile, these two standard DDQN algorithms drop all the lowest-priority requests. There are two reasons for the above results. First, we assign greater reward values to the successful placement of higher-priority SFC requests.

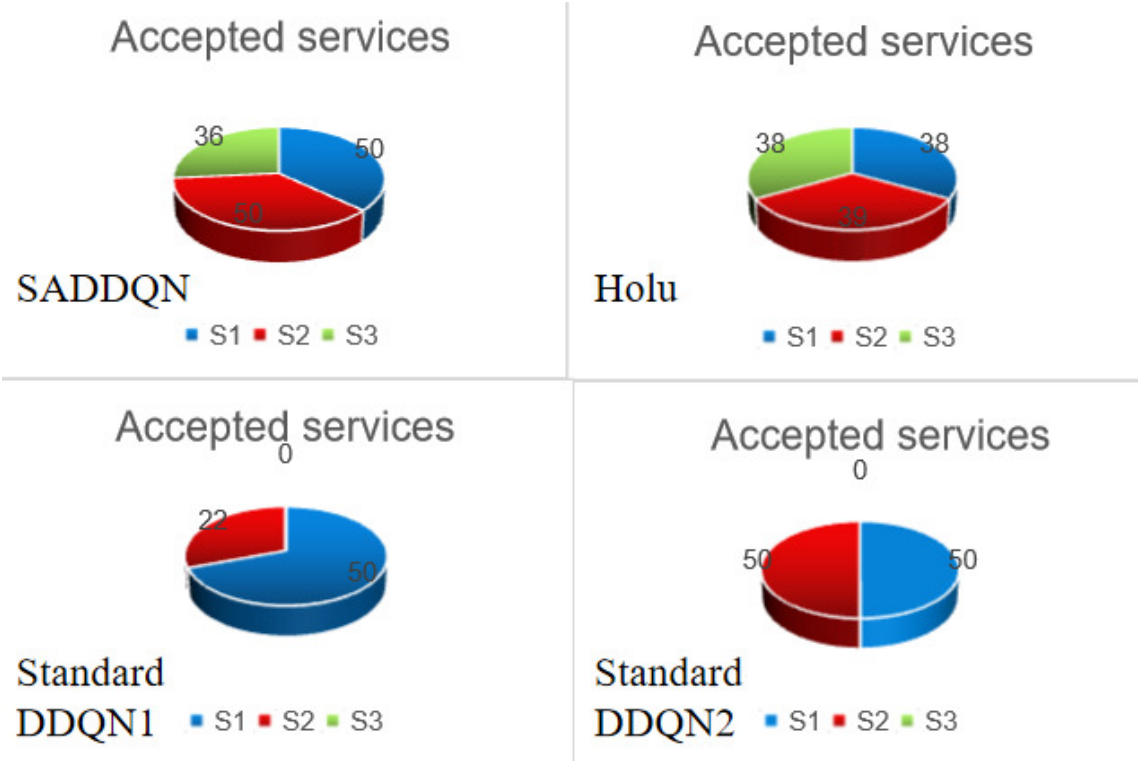

Figure 4.5 : Acceptance of different SFCs.

Second, the learning-based algorithms can learn the arrival distribution of different SFCs. With the object of maximizing the total discounted cumulative reward, the DDQN-based models discard the lower-priority SFCs when there are insufficient resources to accommodate all. However, the Holu algorithm, which does not consider SFC priority, tries to accommodate SFCs in sequence and thus cannot ensure priority allocation for higher-priority SFCs. Hence, it treats every request equally and accommodates nearly the same number of requests for three categories. In conclusion, our proposed SADDQN outperforms the other algorithms regarding the acceptance ratio. Meanwhile, it ensures priority accommodation for higher-priority SFCs.

We further compare our proposed SADDQN with the standard DDQN and the Holu algorithm in terms of the average end-to-end delay. In Fig. 4.6, we can see

that, compared to the other algorithms, our proposed SADDQN obtains an average end-to-end delay closer to the requirement for any type of SFC. The reason is that we optimize the resource allocation for VNFs of each SFC. Specifically, we approach the latency requirement of an SFC using the proposed BSAGD algorithm, which iteratively searches the minimum cost for realizing the SFC. On the contrary, without the proposed sub-action concept or the resource optimization algorithm, the standard DDQN always overly provisions these flows. The Holu algorithm first proposes a Physical Machine (PM) ranking mechanism, which is based on power consumption, to resolve the VNF placement. Then, it employs a DCLC shortest path algorithm to obtain the optimal path between the selected VNFs. Although the Holu algorithm aims to meet the end-to-end delay of each SFC with the minimum cost, it does not take the over-provisioning of resources into consideration. Therefore, from the bar chart for the standard DDQN and the Holu, we can see that each flow suffers a delay shorter than the required delay by at least a few milliseconds. The results indicate that the standard DDQN and the Holu consume extra resources to accommodate SFCs, which can be observed in Figs. 4.7 and 4.8.

We can see that from both Figs. 4.8 and 4.7, of all the algorithms, our proposed SADDQN achieves the highest cost-utility. This is because our BSAGD algorithm flexibly allocates resources to VNFs and thus provides a finer granularity in resource allocation. In contrast, the others use fixed capacity VNFs, inevitably falling into the over-provisioning mire. Furthermore, DDQN-based algorithms minimize the cost from the global perspective, while the Holu minimizes the cost instantaneously rather than farsightedly. Nevertheless, one of the Holu's advantages over the standard DDQN is its VNF sharing policy, which mitigates the overconsumption of resources. For this reason, in Figs. 4.7 and 4.8, with the increasing number of SFC requests, the Holu algorithm is second only to SADDQN in cost-utility. In Fig. 4.7, another finding is that the average cost per request does not change much for the

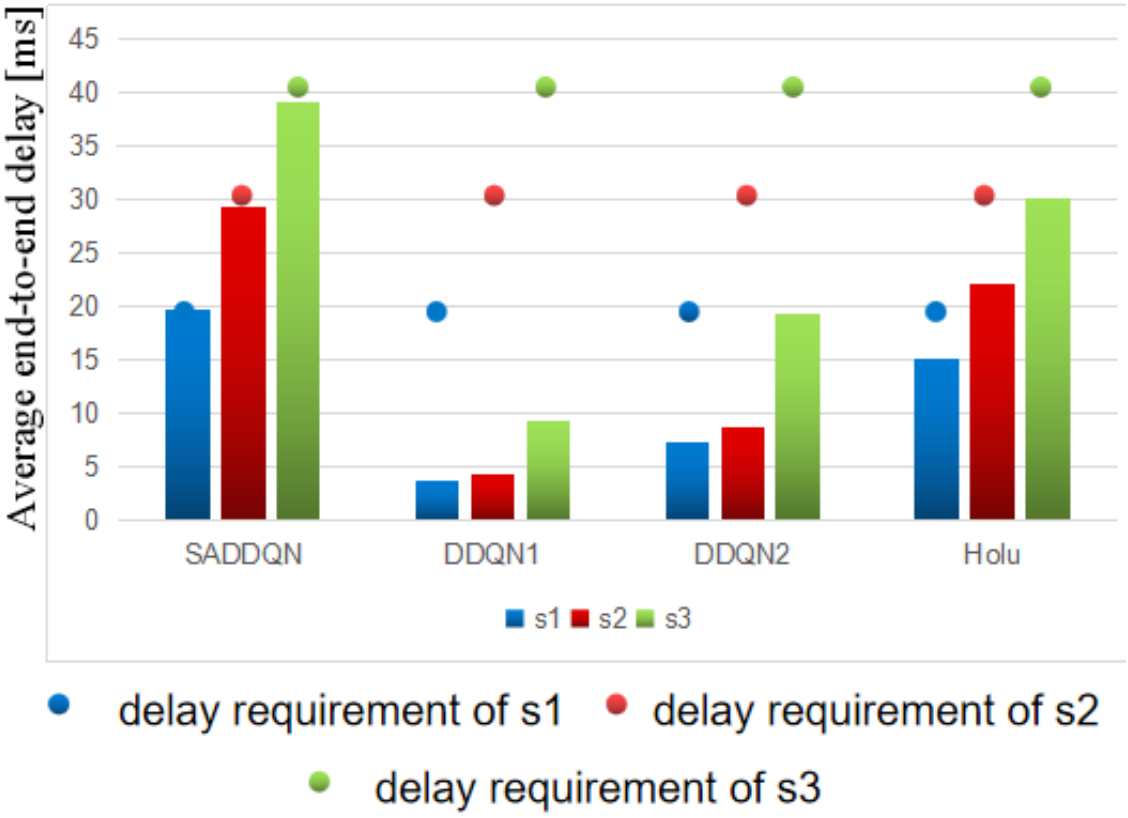

Figure 4.6 : Average end-to-end delay comparison.

SADDQN algorithm, while the cost for the Holu algorithm fluctuates. The reason is that because of the VNF sharing policy, the curve of Holu may ascend when new VNF instances need to be initiated and descend when the current running VNFs can be utilized by new SFCs. The standard DDQN algorithm does not accept VNF sharing for security reasons. Hence, the average cost per request of DDQN1 and DDQN2 keeps stable when all requests can be accepted. However, as the number of requests increases, the two curves start to descend because the lowest priority SFCs, which have a higher cost than SFCs from the other two categories, are gradually discarded. In conclusion, our proposed SADDQN is superior to all the others in terms of cost-utility.

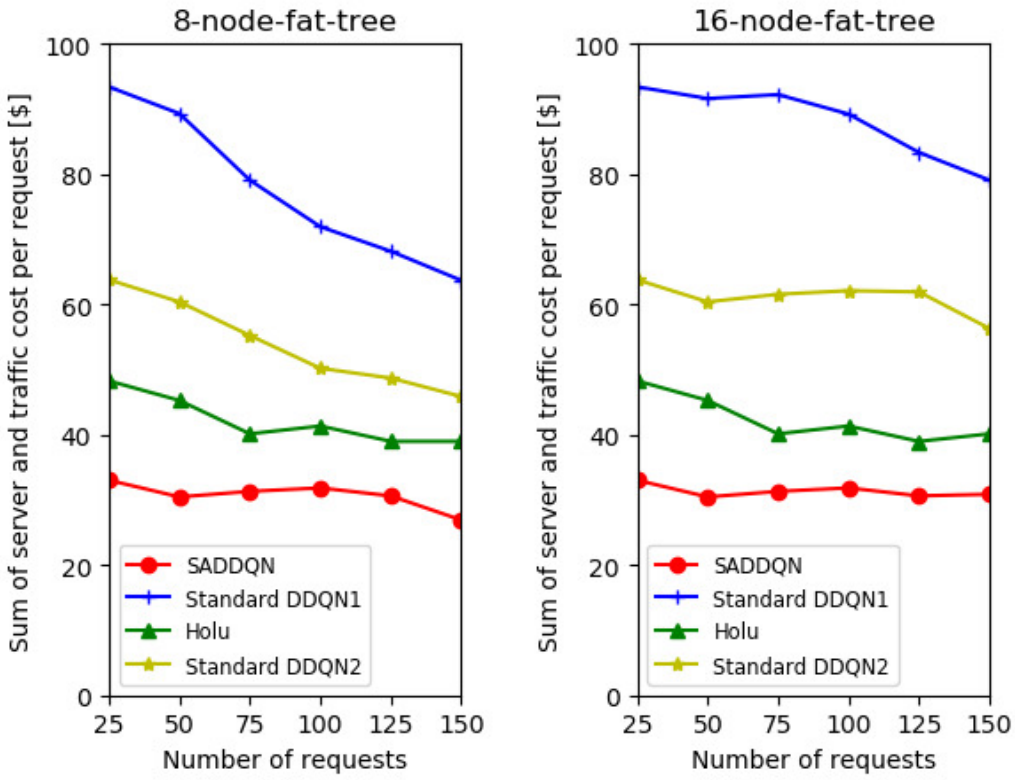

Figure 4.7 : Joint traffic and CPU cost comparison.

4.4 Summary

In this chapter, we proposed a Sub-action Aided DDQN algorithm to maximize the acceptance ratio and ensure priority allocation for higher-priority SFC requests while minimizing the total cost for latency-aware services. We considered a multi-edge cloud scenario where SFC requests with different priorities are fed into the DDQN-agent, and developed an intelligent policy to place the SFCs and allocate resources. Considering the objective, we defined our new state, main action and sub-action, and reward function for the DDQN framework and proposed an algorithm (sub-action) for allocating resources. Simulations showed that our proposed SADDQN can maximize the acceptance ratio and ensure priority allocation for higher-priority SFCs while minimizing the total cost for latency-aware SFCs

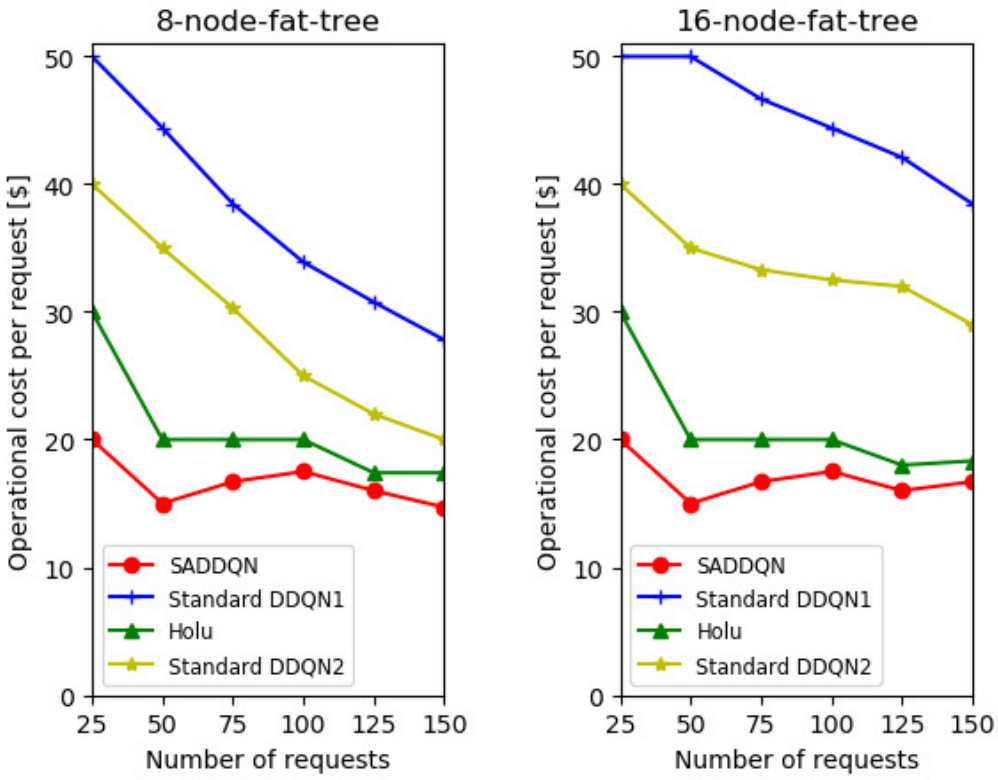

Figure 4.8 : Operational cost comparison.

with different latency requirements. Numerical results showed that, compared with the other two popular algorithms, our proposed scheme performs better in minimizing the cost and maximizing the acceptance ratio; thus, it resolves the QoS over-provisioning issue as expected. We have showcased two algorithms for SFC embedding in a single DC so far. In the next chapter, we will propose an algorithm for SFC embedding in an MDC scenario.

Chapter 5

SFC Embedding Approach in an MDC Network

In this chapter, we will introduce a two-stage GCN-based DRL algorithm to resolve the unbalanced loads in an MDC environment. This framework aims to maximize the overall acceptance ratio of SFC requests while minimizing the total cost in an MDC network. In the first stage, we propose a GCN-based DRL algorithm as a coarse granularity solution to the SFC embedding problem from the macro perspective. This solution outlines a local observation scope (LOS) for each agent in the multi-agent system of the second stage, where all agents simultaneously handle SFC requests from their respective DCs using a multi-agent framework from the micro perspective.

5.1 System Model and Problem Formulation

5.1.1 System Architecture

As we have mentioned in the previous chapter, the NFV framework receives requests for allocating network services with specific characteristics. Accordingly, at the core of the NFV architecture, the management and orchestration (MANO) framework plans SFCs allocation using the NFV infrastructure.

Inspired by the multi-tier SDN-controller system proposed in [59], we design a multi-tier MANO system depicted in Fig. 5.1. In the first stage, the upper-tier MANO cooperates with all lower-tier MANOs in setting a LOS for each of them. In the second stage, we consider a discrete time-slot system as in [3]. Once a time slot starts, the resources occupied by expired SFCs are released, and all lower-tier

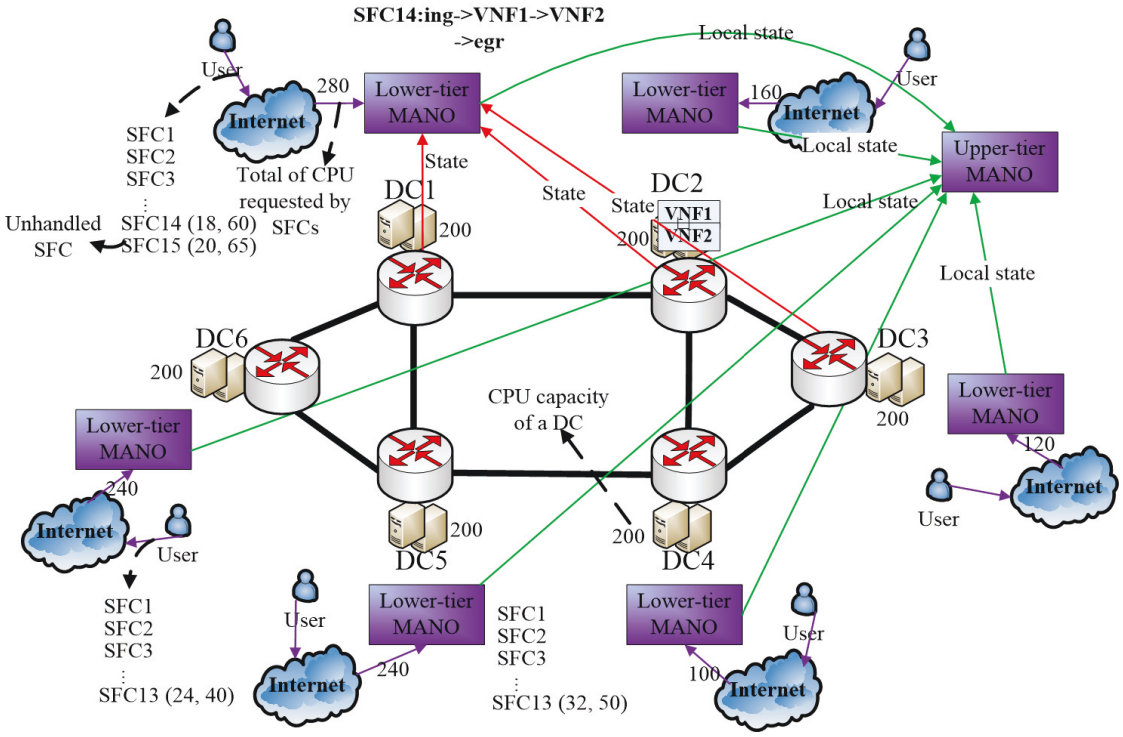

Figure 5.1 : Network infrastructure considered in this chapter.

MANOs start to embed SFCs simultaneously. For each DC, based on its current state, the corresponding lower-tier MANO handles the SFCs received by this DC one after another. Once a DC has insufficient resources to accommodate the SFCs waiting in its queue, the lower-tier MANO in charge of this DC will place SFCs in low-load DCs within its LOS.

5.1.2 NFV Infrastructure

We model the MDC network as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of DC nodes and $\mathcal{E}$ is the set of transmission links that interconnect these DC nodes. Hence, the numbers of DC nodes and physical links are $|\mathcal{V}|$ and $|\mathcal{E}|$, respectively. Furthermore, the CPU capacity and the remaining CPU resources of DC v are denoted by $\{C_v | C_v \geq 0; v \in \mathcal{V}\}$ and $\{c_v | c_v \geq 0; v \in \mathcal{V}\}$, respectively. The BW capacity and the remaining BW of link (v, u) are denoted by $\{B_{v,u} | B_{v,u} \geq$

$0, (v,u) \in \mathcal{E}\}$ and $\{b_{v,u} | b_{v,u} \geq 0, (v,u) \in \mathcal{E}\}$, respectively.

5.1.3 Characteristics of SFC Requests

We denote M as the number of SFC requests to be handled within a time slot. SFC request i can be defined as $\mathrm{SR}_i = \{F_i, C_i, B_i\}$, where B_i indicates the BW requested by SFC i, $F_i = \{F_{i0}, F_{i1}, F_{i2}, ..., F_{iK_i}, F_{iK_{i+1}}\}$ (K_i is the number of VNFs in SFC i) denotes the ingress (F_{i0}), egress ($F_{iK_{i+1}}$), and VNFs ($\{F_{i1}, F_{i2}, ..., F_{iK_i}\}$) that are used to compose SFC i, and $C_i = \{C_{i1}, C_{i2}, ..., C_{iK_i}\}$ indicates the CPU requested by each VNF in SFC i. The VL between the j^{th} and $(j+1)^{\text{th}}$ nodes of SFC i is denoted by $(F_{ij}, F_{i(j+1)})$.

5.1.4 Cost Model

Traffic Cost

In an SFC, one VNF must forward packets to the next through the VL connecting them. The VL can be embedded in one or multiple physical links, each of which might have a different hop distance depending on the network topology. As in [17], we define the traffic cost caused by a VL connecting two adjacent VNFs as the product of the overall distance between these two VNFs and the BW of the VL. Since the distance between two DCs is far longer than that between two servers within a DC, we only consider the traffic in the inter-DC network as in [35] and [38]. As a result, the total traffic cost can be written as

$$C_{\text{tr}} = \beta \sum_{i=1}^{M} \sum_{j=0}^{K_i} \sum_{(v,u)\in\mathcal{E}} z^{v,u}_{F_{ij},F_{i(j+1)}} \cdot B_i \cdot \mathrm{Dis}\,(v,u)\,, \tag{5.1}$$

where β is the price that transfers the traffic cost to a monetary value, binary $z^{v,u}_{F_{ij},F_{i(j+1)}}$ indicates whether the VL connecting the j^{th} and $(j+1)^{\text{th}}$ nodes of SFC i is embedded in physical link (v,u), and $\mathrm{Dis}\,(v,u)$ is the distance between DC v and DC u.

Server Cost

Compared with memory and capacity, CPU's computational power is much more important. Therefore, as for server cost, we only consider CPU, and the extension to multiple types of resources is simple. The total server cost can be written as

$$C_{\text{se}} = \gamma \sum_{i=1}^{M} \sum_{j=1}^{K_i} \sum_{v \in \mathcal{V}} x_{i,j}^{v} \cdot C_{ij}, \tag{5.2}$$

where γ is the price that transfers the server cost to a monetary value, and binary $x_{i,j}^{v}$ indicates whether the j^{th} VNF of SFC i nests in DC v.

5.1.5 Problem Formulation

In this part, we formulate the objective as an optimization problem. Our purpose is to maximize the overall acceptance ratio of SFC requests while minimizing the total cost regarding the infrastructure resource constraints. We first focus on the constraints of the SFC embedding problem. For any DC node, the computing resource constraint must be satisfied; thus, we have

$$\sum_{i=1}^{M} \sum_{j=1}^{K_i} x_{i,j}^{v} \cdot C_{ij} \leq C_v, \qquad \forall v \in \mathcal{V}. \tag{5.3}$$

For any physical link, the BW constraint must be satisfied; therefore, we have

$$\sum_{i=1}^{M} B_i \sum_{j=0}^{K_i} z_{F_{ij},F_{i(j+1)}}^{v,u} \leq B_{v,u}, \qquad \forall (v,u) \in \mathcal{E}. \tag{5.4}$$

Each VNF can be placed on only one DC node; hence, we have

$$\sum_{v \in \mathcal{V}} x_{i,j}^{v} \leq 1, \qquad (1 \leq i \leq M, 1 \leq j \leq K_i). \tag{5.5}$$

To ensure that the j^{th} and $(j+1)^{\text{th}}$ nodes of SFC i are connected by a continuous path, we have

$$\begin{gathered} \sum_{(v,u) \in \mathcal{O}(v)} z_{F_{ij},F_{i(j+1)}}^{v,u} - \sum_{(u',v) \in \mathcal{I}(v)} z_{F_{ij},F_{i(j+1)}}^{u',v} = x_{i,j}^{v} - x_{i,j+1}^{v}, \\ \forall v \in \mathcal{V}, 1 \leq i \leq M, 0 \leq j \leq K_i, \end{gathered} \tag{5.6}$$

where $\mathcal{I}(v)$ and $\mathcal{O}(v)$ denote the sets of incoming links and outgoing links of DC v, respectively.

As in [60], we use a constraint to avoid a loop as follows:

$$\begin{aligned} \sum_{(v,u)\in\mathcal{O}(v)} z^{v,u}_{F_{ij},F_{i(j+1)}} + \sum_{(u',v)\in\mathcal{I}(v)} z^{u',v}_{F_{ij},F_{i(j+1)}} \leq 1, \\ \forall v \in \mathcal{V}, 1 \leq i \leq M, 0 \leq j \leq K_i. \end{aligned} \tag{5.7}$$

An SFC request is accommodated if and only if all virtual elements in this SFC are successfully deployed; hence, we can denote the number of accepted requests as

$$\begin{aligned} A = \sum_{i=1}^{M} f((\sum_{j=1}^{K_i}\sum_{v\in\mathcal{V}} x^v_{i,j} - K_i) + \\ (\sum_{j=0}^{K_i}\sum_{(v,u)\in\mathcal{E}} z^{v,u}_{F_{ij},F_{i(j+1)}} - K_{i+1})), \end{aligned} \tag{5.8}$$

where

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0. \end{cases} \tag{5.9}$$

Since our object is to maximize the overall acceptance ratio of SFC requests while minimizing the total cost, the optimization problem can be written as

$$\begin{aligned} (P1): \max_{x^v_{i,j}, z^{v,u}_{F_{ij},F_{i(j+1)}}} \quad & U(x^v_{i,j}, z^{v,u}_{F_{ij},F_{i(j+1)}}) = \\ & \alpha_1 \cdot A - \alpha_2 \cdot (C_{\text{tr}} + C_{\text{se}}) \\ s.t. & (5.3)(5.4)(5.5)(5.6)(5.7). \end{aligned}$$

In the objective function above, α_1 and α_2 are the weights of two objectives. Since the objective of maximizing the acceptance ratio of SFC requests is more important, we assign a higher weight to it.

It can be challenging to solve this optimization problem using heuristic approaches because it is difficult to track the dynamics of the network. Alternatively,

learning-based techniques can be utilized to learn the network dynamics, such as the arrival patterns and the resource requirements of various services, and solve such a problem. The learning technique aims to learn a policy that determines what action to take in each state. In the next section, we detail a two-stage GCN-based DRL for the SFC embedding problem in our MDC environment.

5.2 Proposed Two-stage SFC Embedding Scheme

In this section, we present our proposed algorithm. First, the background of the proposed algorithm is provided. Then, we detail our two-stage GCN-based DRL algorithm for solving P1.

5.2.1 Background of the Proposed Algorithm

Model Our Environment as a Multi-agent MDP

The SFC embedding problem has been modeled as an MDP in many existing works. In our MDC environment, if we model this problem as an MDP, the agent needs to handle SFC requests received by all DCs; thus, the action space would be too large for the DRL algorithm to converge, especially in a large-scale MDC. Alternatively, we study a multi-agent MDP [4]. The details of the state representation, action space, and reward function for each agent in the context of our problem are provided in Section 5.2.3.

Overview of Multi-agent DRL and Motivation for Using MAPPO

MARL algorithms have been widely adopted to solve the multi-agent MDP models. The authors in [5] proposed the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm for mixed cooperative-competitive environments. It is a simple extension of actor-critic policy gradient methods. In this model, the critic has a global view to know the policies of other agents, while the actor can only acquire

local information. In the MADDPG algorithm, each agent learns a policy network $\pi_i(a_i|o_i)$ and a centralized critic network $Q_i^{\pi}(\mathbf{x}, a_1, \ldots, a_N)$, where $\mathbf{x}$ consists of the observations of all agents: $\mathbf{x} = (o_1, \ldots, o_N)$. This algorithm follows the framework of centralized training with decentralized execution (CTDE). Although the DDPG, which follows the policy gradient to find the optimal actions, is more efficient than the value-based DRL algorithms, it does not consider the step size of each policy update. To resolve this issue, OpenAI released the PPO [61] algorithm in 2017. Its core idea is that the new policy should not be too far from the old one after an update. To achieve that, it first uses a ratio to symbolize the difference between the new policy and the old one. Then, it clips the ratio within $[1 - \epsilon, 1 + \epsilon]$, where ϵ is a hyperparameter. The authors in [62] investigated the MAPPO algorithm in three popular multi-agent testbeds: the particle-world environments, the Starcraft multi-agent challenge, and the Hanabi challenge. They found that the MAPPO achieves surprisingly strong performance while exhibiting comparable sample efficiency with the MADDPG.

Motivation for Using GCN

Our MDC network topology is modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ indicate the sets of DC nodes and links, respectively. Since we model our environment as a multi-agent MDP, it is essential to design an MDP state space that comprehensively represents the features of the environment. Feeding the traditional DNN or Convolutional Neural Networks (CNN) with a flat feature vector containing all the state information is the simplest option. Nevertheless, this approach cannot scale well to complex topology structures, because it would be difficult for DNN or CNN to symbolize the relations between every pair of nodes in a graph with handcrafted feature engineering. Therefore, we design an encoder for analyzing $\mathcal{G}$ based on GCN [63], which has been widely exploited in [64, 65, 66] to extract the

features of undirected graphs through aggregating the characteristics of nodes and topologies.

Two-stage Design

In our MAPPO framework, each agent learns a local policy network and a centralized critic network. As in [5], for each agent, the critic has a global perspective to obtain the policies of other agents, while the actor can only acquire local information. Here comes a question: how shall we set an observation scope for each agent? Let's consider the state of the environment depicted in Fig. 5.1. If each agent (lower-tier MANO) can observe the information of all DCs, the MAPPO could converge to the optimum, but the convergence rate might be slow as each agent must cooperate and compete with all the other agents during the training process; If we restrict the LOS of each agent within its one-hop neighbors' area, the MAPPO could converge much faster but might not be able to find the global optimum because of the restricted view of each agent. Therefore, to reach an ideal optimum as quickly as possible, we must design an appropriate LOS for each agent before running our multi-agent algorithm.

5.2.2 The First Stage

In this stage, we first model the load transfer process as an MDP. Then, we propose a GCN-based PPO to solve the MDP.

Model the Load Transfer Process as an MDP

The authors in [59] proposed a multi-tier SDN-controller system for the space-air-ground integrated network. Similarly, we propose a multi-tier MANO system in which an upper-tier MANO assembles local network states from all lower-tier MANOs and schedules resources for SFC requests globally. In this stage, the upper-tier MANO balances the SFC requests loads of multiple DCs from a macro perspec-

tive, aiming to define a LOS for each lower-tier MANO. In that case, each agent of the multi-agent framework in the second stage can only embed SFCs within its LOS; thus, unnecessary competition among multiple agents will be avoided. As a result, we speed up the convergence rate of the multi-agent algorithm in the second stage. Specifically, we define a DC node that has insufficient CPU resources to accommodate the SFC requests received by this DC as a 'poor' DC, and a DC node that has more CPU resources than those requested by all SFCs waiting in its queue as a 'rich' DC. During a data preprocessing phase, each lower-tier MANO fully exploits the CPU resources of its affiliated DC to accommodate SFCs received by this DC. When there are no SFCs in 'rich' DCs or resources in 'poor' DCs, the upper-level MANO transfers the unhandled SFCs from 'poor' DCs to 'rich' DCs step by step to maximize the overall acceptance ratio of SFC requests while minimizing the total cost. Thus, it is reasonable for us to model the load transfer process, which follows the data preprocessing phase, as an MDP, whose state representation, action space, state transition dynamics, and reward function are defined below.

State Representation: The state is represented by graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The features of a DC node $v \in \mathcal{V}$ include four parts: (i) the available CPU resources (c_v), (ii) the sum of BW defined as the total available BW of links associated with DC v, (iii) the requested CPU resources of each unhandled SFC, and (iv) the requested BW resources of each unhandled SFC. Considering the MDC topology depicted in Fig. 5.1, the initial state is shown in Table 5.1.

The features of the graph, including the nodes and the topology, need to be represented by a complex data structure and thus cannot be directly fed into a fully connected DNN. Hence, to feed the DNN with a real-valued vector, we apply the encoding architecture proposed in [67], which involves two main steps shown below, to the state encoding process of our problem.

Table 5.1 : Initial state.

DC ID	Remaining CPU	Sum of BW	Req CPU of the first unhandled SFC	Req BW of the first unhandled SFC	Req CPU of the second unhandled SFC	Req BW of the second unhandled SFC
1	0	$b_{1,2}+b_{1,5}$ $+b_{1,6}$	18	60	20	65
2	$200-160$	$b_{2,1}+b_{2,3}$ $+b_{2,4}$	0	0	0	0
3	$200-120$	$b_{3,2}+b_{3,4}$	0	0	0	0
4	$200-100$	$b_{4,2}+b_{4,3}$ $+b_{4,5}$	0	0	0	0
5	0	$b_{5,1}+b_{5,4}$ $+b_{5,6}$	32	50	0	0
6	0	$b_{6,1}+b_{6,5}$	24	40	0	0

- Node-level encoding using GCN: On the basis of the natural features of each DC in graph $\mathcal{G}$, we employ GCN to aggregate information from all neighbours for each DC. Feeding the raw features of the graph into a GCN with l layers, we create a matrix $\mathbf{N} \in \mathbb{R}^{|\mathcal{V}|\times D}$. In matrix $\mathbf{N}$, row vector $\mathbf{n}_v$ $(1 \leq v \leq |\mathcal{V}|)$ indicates the encoding of DC v and D is a hyperparameter symbolizing the dimension of $\mathbf{n}_v$. We consider a multi-layer GCN with the typical layer-wise propagation rule [63]:

$$\mathbf{H}^{(l+1)} = \sigma(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}). \tag{5.10}$$

Here, $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_{|\mathcal{V}|}$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections, where $\mathbf{I}_{|\mathcal{V}|}$ is the identity matrix. $\mathbf{W}^{(l)}$ is a layer-specific trainable weight matrix, and $\sigma(\cdot)$ denotes an activation function such as the ReLU$(\cdot)$. $\mathbf{H}^{(l)}$ is the matrix of activations in the l^{th} layer; $\mathbf{H}^{(0)} = \mathbf{X}$, where

$\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times Y}$ is the natural feature matrix, with each row indicating the Y-dimensional feature vector for a DC node. In our case, $Y = 2+2\max(u(v), v = 1, \ldots, |\mathcal{V}|)$, where $u(v)$ indicates the number of unhandled SFCs in the waiting queue of DC v.

- Graph level encoding using attention layers: As in [67], we define the context of the MDC network as follows:

$$\mathbf{ct} = g((\frac{\sum_{v=1}^{|\mathcal{V}|} \mathbf{n}_v}{|\mathcal{V}|})\mathbf{W}), \tag{5.11}$$

where $\mathbf{W}$ is a learnable weight matrix, $\mathbf{n}_v$ is the encoding of DC v, and $g(\cdot)$ is a nonlinear function. Similarly, we define the encoding of the MDC as a weighted sum of DC node encoding. As a consequence, the MDC network encoding $\mathbf{h}$, a real-valued vector, can be given by

$$\mathbf{h} = \sum_{v=1}^{|\mathcal{V}|} a_v \mathbf{n}_v, \tag{5.12}$$

where a_v, the weight of DC v, equals the inner product of its node encoding and the context:

$$a_v = \mathbf{n}_v^T \mathbf{ct}. \tag{5.13}$$

Action Space: At time step t, the agent sequentially selects (i) a 'poor' DC and a 'rich' DC (ii) and the route from the 'poor' DC to the 'rich' DC. Thus, the action at time step t can be defined as follows:

$$\boldsymbol{a}_t = (s, d, i_{s,d}), \tag{5.14}$$

where s and d indicate the indexes of the 'poor' DC and the 'rich' DC, respectively, and $i_{s,d}$ indicates the index of the route from the 'poor' DC to the 'rich' DC. Considering the MDC topology depicted in Fig. 5.1, the action space for the 'poor' DC (DC 1) and 'rich' DC (DC 2) pair is shown in Table 5.2.

Table 5.2 : Action space for pair (DC 1, DC 2).

Source	Destination	Route
1	2	1→2
1	2	1→5→4→2
1	2	1→6→5→4→2
1	2	1→6→5→4→3→2

We denote the sets of 'poor' DCs and 'rich' DCs by $\mathcal{P}$ and $\mathcal{R}$, respectively. Therefore, the numbers of 'poor' DCs and 'rich' DCs are $|\mathcal{P}|$ and $|\mathcal{R}|$, respectively. Besides, we denote the number of routes between DC p and DC r as $K_{p,r}$. As in [36], rather than enumerating all possible candidate routes for each node pair, we set a small value of K (e.g., $K = 5$) to limit the number of routes because the gain from considering long-distance paths is limited. Consequently, the action-space size is $\sum_{p\in\mathcal{P}} \sum_{r\in\mathcal{R}} \min(K_{p,r}, K)$.

Action $\boldsymbol{a}_0 = [1, 2, 1]$ indicates that at time step 1, the first unhandled SFC in the waiting queue of DC 1 will be placed in DC 2, and the agent will choose the first route between these two DCs (DC 1 → DC 2) to transfer this SFC.

State Transition Dynamics: At time step t, the agent observes the state s_t, chooses an action $\boldsymbol{a}_t = (s, d, i_{s,d})$, and performs the action in the MDC environment. Then, the MDP transits to the next state s_{t+1}, with its node features updated as follows:

- If neither the BW constraint of any link along the chosen route nor the CPU constraint of the destination DC is violated, then the action is successful; thus, several related fields need to be updated. First, the remaining CPU of DC d is updated by subtracting the total requested CPU of the first unhandled SFC

in the waiting queue of DC s. Next, the BW of any link along route $i_{s,d}$ is updated by subtracting the requested BW of this SFC; thus, the BW feature of any DC will be updated if the DC is associated with an updated link. Last, for DC s, the first unhandled SFC has been settled, and the features of the $(n+1)^{\text{th}}$ unhandled SFC will be moved to the corresponding fields of the n^{th} unhandled SFC.

- Otherwise, this is a bad action, and thus the state remains unchanged.

Considering the case depicted in Fig. 5.1. Under the initial state, the agent performs $\boldsymbol{a}_0$ in the environment. Consequently, the MDP transits to s_1, which can be symbolized by Table 5.3.

Table 5.3 : State s_1.

DC ID	Remaining CPU	Sum of BW	Req CPU of the first unhandled SFC	Req BW of the first unhandled SFC
1	0	$b_{1,2}+b_{1,5}+b_{1,6}$ -60	20	65
2	$200-160$ -18	$b_{2,1}+b_{2,3}+b_{2,4}$ -60	0	0
3	$200-120$	$b_{3,2}+b_{3,4}$	0	0
4	$200-100$	$b_{4,2}+b_{4,3}+b_{4,5}$	0	0
5	0	$b_{5,1}+b_{5,4}+b_{5,6}$	32	50
6	0	$b_{6,1}+b_{6,5}$	24	40

Reward Design: We aim to maximize the overall acceptance ratio of SFC requests while minimizing the total cost. For an SFC transfer, if no BW or CPU

constraints violation exists, the agent receives a reward consisting of two parts. The first part indicates a successful SFC transfer, and the second is the network traffic cost caused by the transferred SFC. Otherwise, it gets a penalty. Accordingly, the reward function is given by

$$R_t = \begin{cases} 1 - \delta \cdot B_t \cdot \mathrm{Dis}(\boldsymbol{a}_t[0], \boldsymbol{a}_t[1], \boldsymbol{a}_t[2]), \\ \text{if no BW or CPU constraints violation} \\ -1, \text{ otherwise,} \end{cases}$$

where B_t is the requested bandwidth of the SFC to be handled at time step t, $\mathrm{Dis}(\boldsymbol{a}_t[0], \boldsymbol{a}_t[1], \boldsymbol{a}_t[2])$ is the distance of the $(\boldsymbol{a}_t[2])^{\text{th}}$ route between $\boldsymbol{a}_t[0]$ and $\boldsymbol{a}_t[1]$, and δ is the weight of the cost. Since maximizing the acceptance ratio of SFC requests is our top priority, we assign a value to δ such that the second term in the first segment is always smaller than 1.

GCN-based PPO

The PPO algorithm [61] is a policy gradient DRL algorithm that comprises two components: (i) sampling data through interactions with the environment (ii) and optimizing a 'surrogate' objective function using stochastic gradient ascent. Many existing works have shown it to be much more straightforward to implement and general than other policy-based DRL algorithms.

The policy gradient objective function of typical policy gradient algorithms is defined as follows [68]:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t], \tag{5.15}$$

where $\hat{A}_t$ is an estimator of the advantage function at time step t and is given by

$$\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t). \tag{5.16}$$

If $\hat{A}_t > 0$, action a_t is better than the average of all the actions under state s_t. Hence, by taking gradient ascent steps on the policy loss function $L^{PG}(\theta)$, we can teach the agent to choose better actions. However, it is challenging to select the step size. An inadequate small step size will result in a too-slow training process, while with too large a step size, the algorithm may go astray and converge to a local maximum. Hence, OpenAI proposed the PPO algorithm, which optimizes the actor training by clipping the policy update of each step. The clipped 'surrogate' objective of the PPO algorithm is the following [61]:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \tag{5.17}$$

where

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}. \tag{5.18}$$

In Eq.(5.18), $r_t(\theta)$ denotes the ratio between the probability of action a_t in state s_t under the current policy and that under the previous policy. If $r_t(\theta) > 1$, in state s_t, the action is more probable under the current policy than under the previous; If $1 > r_t(\theta) > 0$, in state s_t, the action is less probable under the current policy than under the previous. In Eq.(5.17), we can see two ratios: one non-clipped and the other clipped within $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter (set to 0.2 in [62]) that defines the clip range. The minimum of these two ratios ensures a modest policy update, as the new policy is not allowed to be too far away from the old one.

A general overview of our proposed GCN-based PPO algorithm is shown in Fig. 5.2. Its workflow is summarized below. The sample collection process is described in steps 1-5. The training procedures for the policy network and the critic network are described in steps 6-9 and step 10, respectively.

1. Step 1: The agent observes the current state (s_t) of the MDC network, represented by a graph.

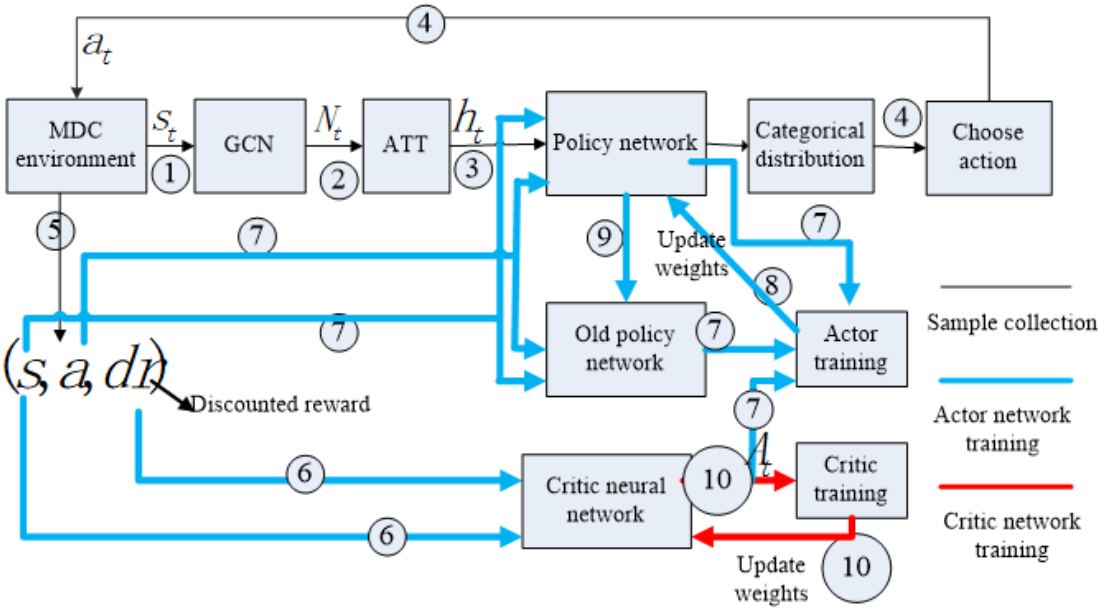

Figure 5.2 : A brief signal flow of our proposed PPO.

2. Step 2: State s_t is fed into a GCN. Subsequently, a matrix $\mathbf{N}_t$, which represents the node-level encoding of the MDC network, is output after multiple layers of convolution.

3. Step 3: An attention mechanism is utilized to learn the weight of each DC node encoding, and the encoding of the MDC is defined as a weighted sum of DC node encoding. In this way, we abstract the graph-structured features of the environment into a real-valued vector state $\mathbf{h}_t$, which is then fed into the current policy network.

4. Step 4: The outputs of the current policy network are fed into the SoftMax function to generate a Categorical distribution. Then, the agent samples data from the distribution to choose a corresponding action a_t, which will be performed in the environment.

5. Step 5: In response to action a_t, a reward r_t is returned to the agent. If time step t is the last step of an episode, the discounted reward for each step

is calculated, and the batch of samples collected in this episode (s_t, dr_t, a_t) $(t = 1, 2, \ldots, T)$ is stored into a buffer.

6. Step 6: Encoded s_t $(t = 1, 2, \ldots, T)$ is fed into the critic network $V_\phi(s_t)$. Then, the value $V_\phi(s_t)$ $(t = 1, 2, \ldots, T)$ and dr_t $(t = 1, 2, \ldots, T)$ are used to calculate the advantage $\hat{A}_t$ $(t = 1, 2, \ldots, T)$ based on Eq.(5.16).

7. Step 7: Encoded s_t $(t = 1, 2, \ldots, T)$ and a_t $(t = 1, 2, \ldots, T)$ are fed into the current policy network and the old one to get $\pi_\theta(a_t|s_t)$ and $\pi_{\theta_{\text{old}}}(a_t|s_t)$, respectively.

8. Step 8: The weights of the current policy network are updated by maximizing the policy loss function given by Eq.(5.17).

9. Step 9: Copying the weights of the current policy network to the old one.

10. Step 10: The weights of the critic network are updated by minimizing the value loss function given in line 14 of Algorithm 4.

The GCN-based PPO algorithm is summarized in Algorithm 4. At the beginning of each episode, we need to fulfill two tasks: (i) releasing the resources occupied by expired SFCs (line 3) (ii) and exploiting the resources thoroughly in all DCs to accommodate the SFCs waiting in their respective queues until there are no SFCs in 'rich' DCs or resources in 'poor' DCs (line 4). After that, the agent starts to transfer unhandled SFCs step by step (line 5) from 'poor' DCs to 'rich' DCs, aiming to maximize the overall acceptance ratio of SFC requests while minimizing the total cost. When no SFCs left, the end of an episode is reached (line 6). Then, a batch of samples (s_t, dr_t, a_t) $(t = 1, 2, \ldots, T)$, which will be used to train the critic and the policy networks, is stored into a buffer. The batch of samples will be reused to train the critic and the actor for L epochs (lines 9-16). We train the neural networks episode by episode until their weights converge. Once the algorithm converges, for

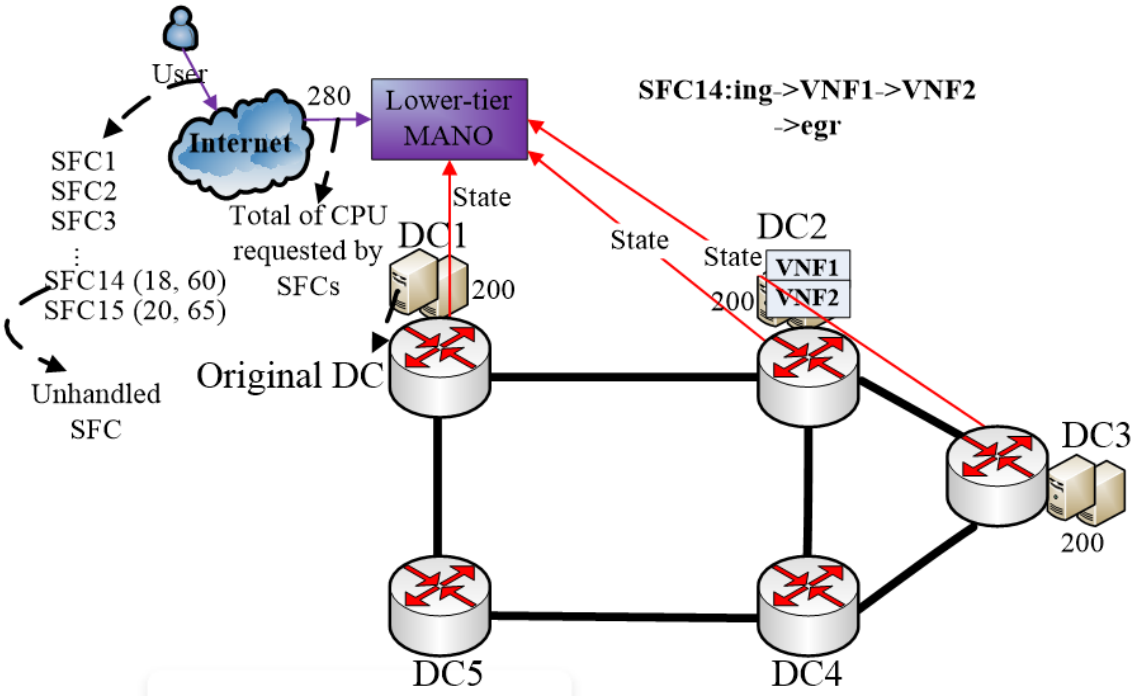

Figure 5.3 : The local observation scope of agent 1.

each 'poor' DC, we know to which 'rich' DCs and via which routes shall we transfer its unhandled SFCs; thus, we set an appropriate LOS for each agent in the multi-agent framework of the next stage. For instance, in the LOS of agent 1 depicted in Fig. 5.3, agent 1 has exhausted the IT resources in its affiliated (original) DC (DC 1); thus, it needs to take advantage of the IT resources of DC 2 and DC 3 to embed its unhandled SFCs, and the data streams of these SFCs might travel through the border router of DC 4 or DC 5. For the agent of each 'rich' DC, the LOS only includes its affiliated DC.

5.2.3 The Second Stage

In the first stage, we set a LOS for each lower-tier MANO; thus, we can efficiently design a multi-agent framework in this stage such that needless competition among multiple agents can be prevented. The multi-agent framework is composed of two phases. During the first phase, all agents simultaneously embed SFCs in their

Algorithm 4 GCN-based PPO Algorithm

Initialize neural network parameters θ and ϕ
for *episode* $= 1, 2, \ldots$ **do**
  Release the resources occupied by expired SFCs
  Preprocess SFCs received by each DC
  Run policy π_θ for T timesteps, collecting $\{s_t, a_t, dr_t\}$
  Break the timesteps loop if no requests left
  Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)$
  $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$
  for $l = 1, \ldots, L$ **do**
    $L^{CLIP}(\theta) = \sum_{t=1}^{T} \min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t, \text{clip}(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\hat{A}_t)$
    Update θ via a gradient method w.r.t. $L^{CLIP}(\theta)$
  end for
  for $l = 1, \ldots, L$ **do**
    $L^{VL}(\phi) = \sum_{t=1}^{T} (\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t))^2$
    Update ϕ via a gradient method w.r.t. $L^{VL}(\phi)$
  end for
end for

respective DCs. The second phase is initiated once the agent of a 'poor' DC starts to embed its unhandled SFCs in 'rich' DCs. We model the SFC embedding problem in phase two as a multi-agent MDP [4] and solve it using a MARL approach, GCN-based MAPPO.

The Components of the Multi-agent MDP

First, we define the observation representation, action space, and reward function for agent m at time step t as follows:

Observation Representation: The LOS of agent m can be modeled as a graph $\mathcal{G}_m = (\mathcal{V}_m, \mathcal{E}_m)$, a sub-graph of $\mathcal{G}$. The observation of agent m at time step t ($o_{m,t}$) consists of three parts: the graph-structured information of $\mathcal{G}_m$, the location (DC index) of the VNF embedded at the previous time step, and the index of the original DC. In $\mathcal{G}_m$, each DC node $v \in \mathcal{V}_m$ has four features: (i) the remaining CPU resources (c_v), (ii) the sum of BW defined as the total available BW of links associated with v, (iii) the requested CPU of the current VNF to be processed ($C_{r,m,t}$), (iv) and the requested BW of the VL to be processed ($B_{r,m,t}$).

To feed the graph-structured information into the policy network of agent m, we first reuse the two-step state encoding process as mentioned earlier to encode the sub-graph into a real-valued vector. Then, we concatenate the location of the VNF embedded at the previous time step, the index of the original DC, and the encoded vector to form a new vector $\boldsymbol{h}_{m,t}$, which will be fed into the policy network.

Action Space: Designing an actor network for the SFC placement issue is challenging because the dimension of the output layer of a neural network is fixed, with each neuron symbolizing a possible action. Nevertheless, if we couple the placement of a VNF and a VL, the number of possible placements changes as the location of the previously embedded VNF changes. That is, the action-space size is not a constant under different states. Most existing works avoided this tricky

problem by decoupling the placement of VNFs and VLs. They utilized the actor network to choose a VNF placement and adopted the Dijkstra algorithm to select the 'optimal' route given the VNF placement. This method reduces the action-space size and thus speeds up the convergence rate; however, given a VNF placement, the Dijkstra algorithm chooses the 'optimal' route and thus neglects all the other options. The greed for immediate rewards might result in unfavorable long-term cumulative rewards. To resolve this issue, we use a 3-column row vector $\boldsymbol{a}_{m,t}$ to symbolize the placement of the current VNF and the VL connecting it and its previous VNF. In $\boldsymbol{a}_{m,t}$, the first element indicates the source DC of the VL, the second indicates in which DC will the current VNF be embedded, and the third denotes the route index between the source DC and the DC that will accommodate the current VNF. We do not require that the source DC accommodate the previous VNF. Thus, the source DC can be any DC $v \in \mathcal{V}_m$, and the action-space size is fixed. We denote the number of routes between DC i and DC j by $K_{i,j}$. Because the benefit from enumerating long-distance routes is limited, as in [36] and [35], we restrict the number of routes between DC i and DC j and define it as $K^r_{i,j} = \min(K, K_{i,j})$, where K (e.g., $K = 5$) is the number of shortest routes between two DCs. Thus, the action-space size is $\sum_{i=1}^{|\mathcal{V}_m|} \sum_{j=1}^{|\mathcal{V}_m|} K^r_{i,j}$. Considering the sub-graph depicted in Fig. 5.3, a small part of the action space is shown in Table 5.4.

Action $\boldsymbol{a}_{m,t} = [1, 2, 1]$ indicates that the current VNF is placed in DC 2, the source DC of the VL is DC 1, and the VL connecting DC 2 and DC 1 is embedded in the first route (DC 1 $\rightarrow$ DC 2) between these two DCs.

Reward: The reward function consists of an internal function and two external functions. The interval reward function, which only regulates the behavior of agent

Table 5.4 : Action space.

Source	Destination	Route
1	2	1→2
1	2	1→5→4→2
1	2	1→5→4→3→2

m and has no correlation with other agents, can be defined as

$$R^{\text{in}}(m,t) = \begin{cases} -1, & \text{if } (\boldsymbol{a}_{m,t}[0] \neq \boldsymbol{h}_{m,t}[0]) \\ & \text{or (egress and } \boldsymbol{a}_{m,t}[1] \neq \boldsymbol{h}_{m,t}[1]) \\ 1, & \text{otherwise,} \end{cases}$$

where $\boldsymbol{a}_{m,t}[0]$ and $\boldsymbol{a}_{m,t}[1]$ are the indexes of the source and destination DCs of the VL, $\boldsymbol{h}_{m,t}[0]$ is the index of the DC that the previous VNF is located in, and $\boldsymbol{h}_{m,t}[1]$ is the index of the original DC of agent m, at time step t. If the VL does not start at the previous VNF or end at the original DC for the last pair of VNF (egress) and VL in an SFC, the action is a bad action, and agent m receives a penalty.

The external reward functions guide all agents toward the common objective of maximizing the overall acceptance ratio of SFC requests while minimizing the total cost. The reward function regarding the overall acceptance ratio of SFC requests is defined as

$$R_{\text{a}}^{\text{ex}}(m,t) = \begin{cases} N_{a,t} - N_{c,t}, \text{ if } \exists v \in \mathcal{V},\ c_v \leq \sum_{m=1}^{|\mathcal{P}|} x_{m,t}^{v} C_{r,m,t} \\ \qquad \text{or } \exists (v,u) \in \mathcal{E},\ b_{v,u} \leq \sum_{m=1}^{|\mathcal{P}|} z_{m,t}^{v,u} B_{r,m,t} \\ N_{a,t}, \qquad \text{otherwise,} \end{cases}$$

where binary variable $x_{m,t}^{v}$ indicates whether the VNF to be processed at time step t by agent m nests in DC v, binary variable $z_{m,t}^{v,u}$ indicates whether the VL to be

processed at time step t by agent m is embedded in link (v, u), $N_{a,t}$ is the number of agents that have SFCs to handle at time step t, and $N_{c,t}$ is the number of agents that fail to accommodate their VNFs or VLs at time step t.

We define the reward function for the total cost as

$$\begin{aligned} R_{\mathrm{c}}^{\mathrm{ex}}(m,t) =& \theta_1 \sum_{v\in\mathcal{V}} \sum_{m\in\mathcal{P}} x_{m,t}^{v} \cdot C_{r,m,t} \\ &+ \theta_2 \sum_{(v,u)\in\mathcal{E}} \sum_{m\in\mathcal{P}} z_{m,t}^{v,u} \cdot B_{r,m,t} \cdot \mathrm{Dis}\,(v,u)\,, \end{aligned}$$

where θ_1 and θ_2 are the weights of the CPU cost and the traffic cost, respectively.

Considering our objective, the reward function $R(m,t)$ is expressed as a weighted sum of the internal and external reward functions.

$$R(m,t) = \eta_1 \cdot R^{\mathrm{in}}(m,t) + \eta_2 \cdot R_{\mathrm{a}}^{\mathrm{ex}}(m,t) - \eta_3 \cdot R_{\mathrm{c}}^{\mathrm{ex}}(m,t).$$

Regulating the interval behavior of an agent is an essential prerequisite for achieving the common objective; thus, η_1 is the greatest of the three weights. Maximizing the acceptance ratio is more important than minimizing the total cost; therefore, η_2 is greater than η_3.

Then, we define the global state s_t and action a_t as follows:

State: State (s_t) is the union of the local observation of each agent: $s_t = \{o_{1,t} \cup o_{2,t} \ldots \cup o_{|\mathcal{P}|,t}\}$.

Joint Action: We concatenate the action of each agent to generate the joint action: $a_t = \{a_{1,t}, \ldots, a_{|\mathcal{P}|,t}\}$.

The Proposed Multi-agent Framework

As in [5], the MAPPO used in our scheme is based on the framework of CTDE. Its general overview is shown in Fig. 5.4, where we detail the workflow of agent 1, which is the same as that of all the other agents of 'poor' DCs. Agent m ($1 \leq$

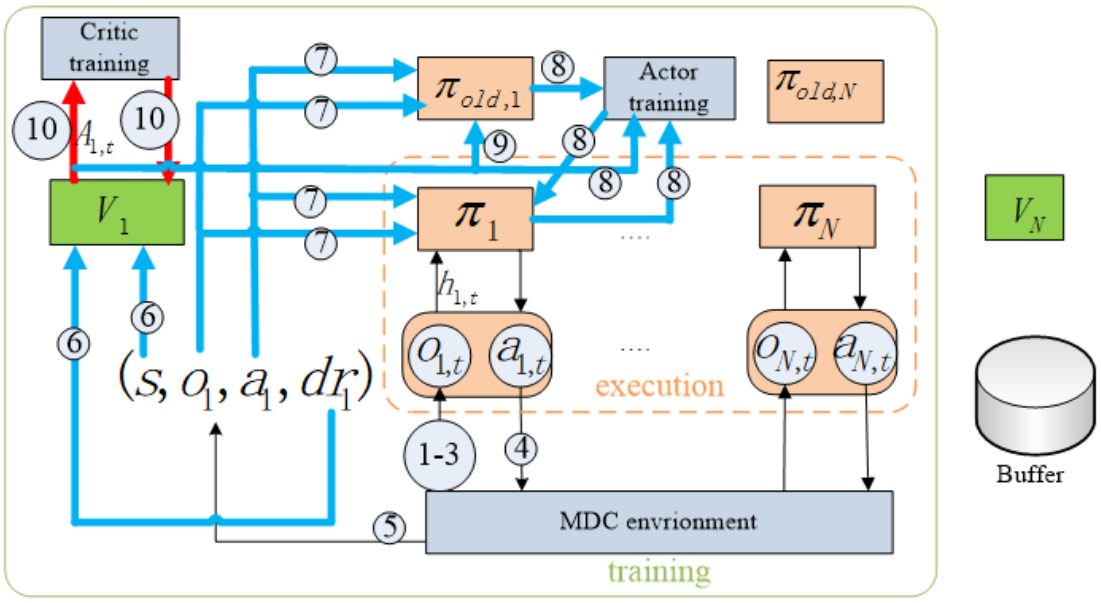

Figure 5.4 : A brief signal flow of our proposed MAPPO.

$m \leq |\mathcal{P}|$) is equipped with a current policy network $\pi_\theta(a_{m,t}|o_{m,t})$, an old policy network $\pi_{\theta_{\text{old}}}(a_{m,t}|o_{m,t})$, and a critic network $V_{\phi_m}(s_t)$, where s_t is composed of the local observations of all agents: $s_t = \{o_{1,t} \cup o_{2,t} \ldots \cup o_{|\mathcal{P}|,t}\}$. The workflow of MAPPO from the perspective of agent m is summarized below.

1. Step 1: At time step t, agent m obtains its local observation $o_{m,t}$, which consists of the graph-structured information, the position (DC index) of the VNF embedded at time step $t-1$, and the index of the agent's original DC.

2. Steps 2-3: The graph-structured features of the environment are encoded into a real-valued vector, which is concatenated with the position of the previous embedded VNF and the index of the original DC to form a new vector $\boldsymbol{h}_{m,t}$. Then, $\boldsymbol{h}_{m,t}$, which can be viewed as an encoded $o_{m,t}$, is fed into the policy network $\pi_{\theta_m}(a_{m,t}|o_{m,t})$.

3. Step 4: An action $a_{m,t}$, the placement of the current VNF and the VL connecting this VNF and the previous, is chosen and performed in the environment.

4. Step 5: In response to the joint action of all agents, a reward $r_{m,t}$ and the global state s_t, which consists of the observations of all agents, are returned to agent m. If time step t is the last step of an episode, the discounted reward for each step is calculated and the batch of samples collected in this episode $(s_t, o_{m,t}, dr_{m,t}, a_{m,t})$ $(t = 1, 2, \ldots, T)$ is stored into buffer m.

5. Step 6: Encoded s_t $(t = 1, 2, \ldots, T)$ is fed into the critic network $V_{\phi_m}(s_t)$. Then, the value $V_{\phi_m}(s_t)$ $(t = 1, 2, \ldots, T)$ and $dr_{m,t}$ $(t = 1, 2, \ldots, T)$ are used to calculate the advantage $\hat{A}_{m,t}$ $(t = 1, 2, \ldots, T)$ based on the equation in line 15 of Algorithm 5.

6. Step 7: Encoded $o_{m,t}$ $(t = 1, 2, \ldots, T)$ and $a_{m,t}$ $(t = 1, 2, \ldots, T)$ are fed into the current policy network and the old one to get $\pi_\theta(a_{m,t}|o_{m,t})$ and $\pi_{\theta_{\text{old}}}(a_{m,t}|o_{m,t})$, respectively.

7. Steps 8-10: The same as the corresponding steps in PPO.

The proposed multi-agent algorithm is summarized in Algorithm 5. The agents of 'rich' DCs embed SFCs in their respective original DCs; thus, they do not need to optimize actions by training neural networks. In contrast, each agent of a 'poor' DC needs to embed SFCs in other DCs within its LOS when it exhausts the resources in its original DC; hence, each agent $m \in \mathcal{P}$ learns a local policy network and a centralized critic network. At the beginning of each episode, we release the resources occupied by expired SFCs (line 4). During the first phase (lines 5-9), all agents place SFCs in their respective DCs. During the second phase, multiple agents of 'poor' DCs simultaneously place their VNFs and VLs in 'rich' DCs (line 11), creating a multi-agent MDP environment. From the perspective of agent m, it embeds VNFs and VLs step by step in 'rich' DCs, aiming to maximize the overall acceptance ratio of SFC requests while minimizing the total cost. When no SFCs left, the end of an episode is reached (line 14). Then, a batch of samples $(s_{m,t}, a_{m,t}, o_{m,t}, dr_{m,t})$ $(t =$

$1, 2, \ldots, T$), which will be used to train the critic network ($V_{\phi_m}(s_t)$) and the policy network ($\pi_{\theta_m}(a_{m,t}|o_{m,t})$), is stored into buffer m. The batch of samples will be reused to train the critic network and the policy network for L epochs (lines 17-24). We train the neural networks episode by episode until their weights converge. Once the algorithm converges, all agents of 'poor' DCs know how to cooperate and compete with each other to maximize the overall acceptance ratio of SFC requests while minimizing the total cost.

5.3 Simulation

5.3.1 Simulation Setup

In this Section, we evaluate the performance of the proposed two-stage GCN-based DRL algorithm regarding the admission ratio and the cost-effectiveness under different network sizes. Two scenarios, i.e., the 14-node NSFNET and the 11-node COST239 topologies[36] depicted in Figs. 5.5 and 5.6, are utilized to assess our algorithm. Cloud service providers such as Amazon, Facebook, and Microsoft are now deploying 100 Gbps data centers. Furthermore, 100 Gbps data centers are expected to become prevalent within the next two years [69] and then be upgraded to 400 Gbps and 800 Gbps. Thus, in our infrastructure, to guarantee the allocation for the vast majority of all SFCs, the BW between each DC pair is set to 1Tbps. The capacity of each DC node is set to 600 units. For the cost model, we set $\beta = 0.1$ \$/Tbps $\cdot$ Km and $\gamma = 0.1$ \$/unit. As for the weights of the two objectives, $\alpha_1 : \alpha_2 = 10 : 1$.

In [70], the authors studied the workload collected by Google and standardized the workload to decrease the fluctuation range. Then, they plotted a figure describing the aggregated task arrival rates of different types of service per minute throughout 29 days. In our simulation, we care about the SFC requests arrival rate of each DC in the MDC environment. Thus, we intercept 24 hours of data in [70]

Algorithm 5 The Proposed Multi-agent Algorithm

Create a thread for each agent
Initialize neural network parameters θ_m and ϕ_m for each agent that manages a poor DC
for *episode* $= 1, 2, \ldots$ **do**
  Release the resources occupied by expired SFCs
  if A rich DC **then**
    Embeds SFCs in this DC
  else
    if A poor DC with sufficient resources to handle the current SFC **then**
      Embeds SFCs in this DC
    else
      Run policy π_{θ_m} for T timesteps, embedding a pair of VNF and VL in each step and collecting $\{s_{m,t},\ a_{m,t}, o_{m,t}, dr_{m,t}\}$
    end if
  end if
  Break the timesteps loop if no requests left
  Estimate advantages $\hat{A}_{m,t} = \sum_{t'>t} \gamma^{t'-t} r_{m,t'} - V_\phi(s_{m,t})$
  $\pi_{\theta_{m,\text{old}}} \leftarrow \pi_{\theta_m}$
  for $l = 1, \ldots, L$ **do**
    $L^{CLIP}(\theta_m) = \sum_{t=1}^{T} \min(\frac{\pi_\theta(a_{m,t}|o_{m,t})}{\pi_{\theta_{\text{old}}}(a_{m,t}|o_{m,t})}\hat{A}_{m,t},\ \text{clip}(\frac{\pi_{\theta_m}(a_{m,t}|o_{m,t})}{\pi_{\theta_{m,\text{old}}}(a_{m,t}|o_{m,t})}, 1-\epsilon, 1+\epsilon)\hat{A}_{m,t})$
    Update θ_m via a gradient method w.r.t. $L^{CLIP}(\theta_m)$
  end for
  for $l = 1, \ldots, L$ **do**
    $L^{VL}(\phi_m) = \sum_{t=1}^{T}(\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_{m,t}))^2$
    Update ϕ via a gradient method w.r.t. $L^{VL}(\phi_m)$
  end for
end for

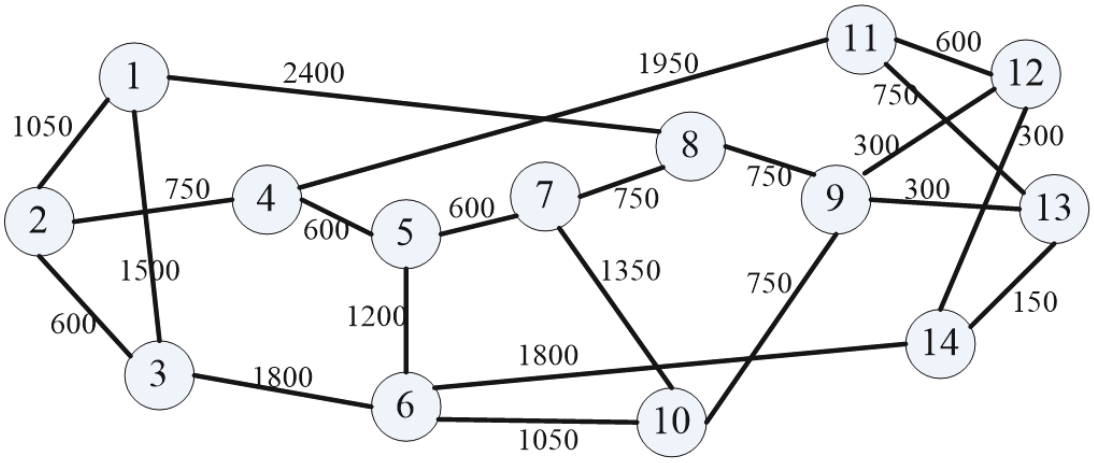

Figure 5.5 : Fourteen-node NSFNET (link length in KM).

for each DC based on its time zone and pick out five typical combinations (seeds in Figs. 5.11 and 5.12) of unbalanced traffic loads for the MDC network. We scale down the numbers of SFC requests based on the capacity of our NFV environment. As in [71] and [72], we assume that each SFC is composed of [2,4] VNFs, and its lifetime follows an exponential distribution with an average of one time slot. The requested CPU of a VNF is a discrete random variable uniformly distributed over the interval [4, 8] units, while the requested BW of an SFC is a random variable uniformly distributed over the interval [25, 100] Gbps.

Neural networks are set up with the following parameters. In the first stage, the number of GCN layers is set to 3. In the second stage, for each agent that manages a 'poor' DC, the number of GCN layers is determined by the scale of its LOS and can be set to 1 or 2 for the policy network. For the critic network, it is set to 3. And $\tanh(\cdot)$ is used as the activation function. Two fully connected layers are created to represent the policy network and the critic network, respectively. Adam [56] is adopted to learn the neural network parameters. The learning rate is 10^{-4} for the policy network and 2×10^{-4} for the critic network, and the discount factor is 0.99. The Clipping ratio ϵ is set to 0.2, and the batch size is set to 100. Besides, each

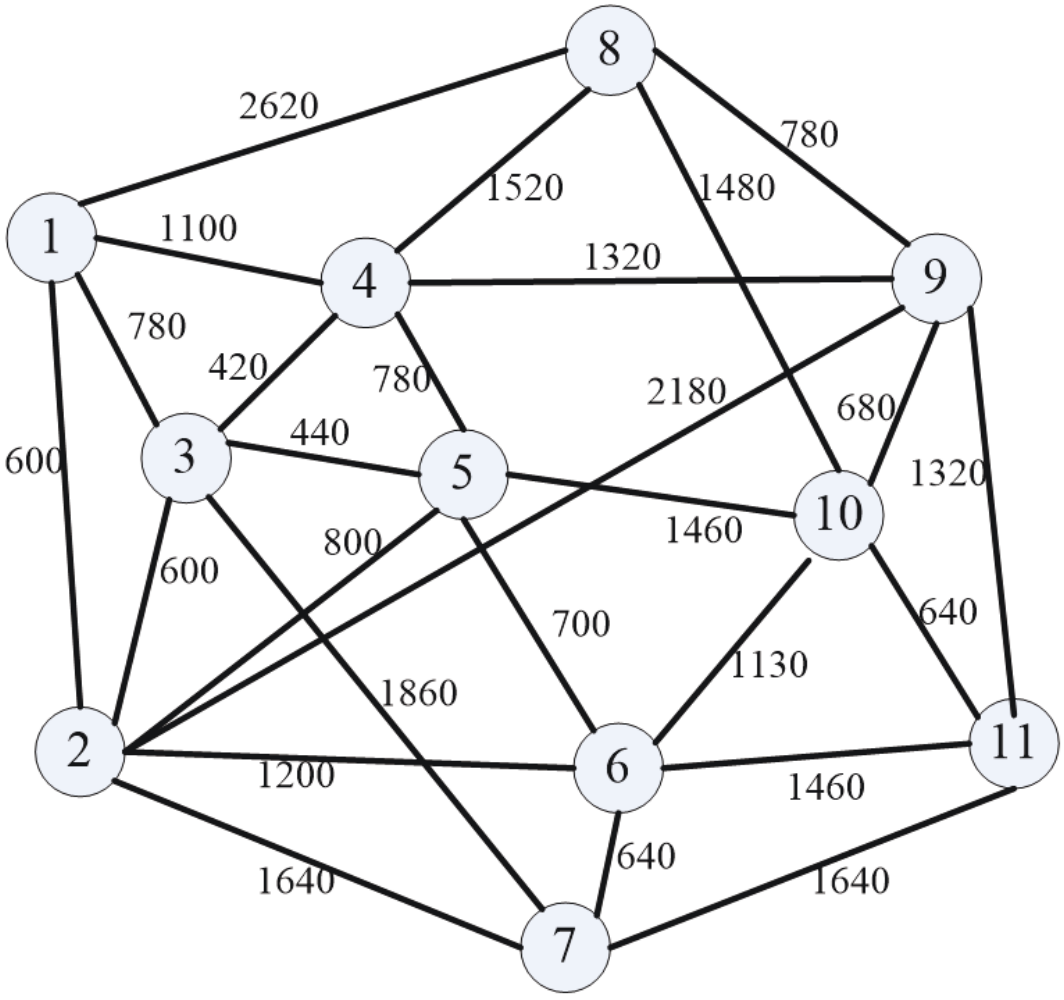

Figure 5.6 : Eleven-node COST239 (link length in KM).

batch of samples will be reused for 10 epochs for training the policy and the critic networks.

Our proposed algorithm consists of a training phase and an evaluation phase. To improve the training efficiency, we first choose one traffic combination from the five typical unbalanced loads to train a model in each scenario. Then, based on the trained model, we perform transfer learning for all the other four traffic combinations.

5.3.2 Algorithms Used for Comparison

MAPPO in a Partially Observable Environment (One-hop Neighbors)

Each agent has its own LOS and can only allocate resources within its one-hop neighbors' scope. This partially observable environment is an extreme case for the LOS of each agent.

MAPPO in a Fully Observable Environment

All agents can observe the whole environment and may cooperate or compete everywhere. This environment is another extreme case for the LOS of each agent.

Asynchronous Advantage Actor Critic (A3C)

A3C [73], one of the newest DRL algorithms, uses multiple agents, each of which has its exclusive network parameters and a copy of the fully observable environment. These agents interact with their respective environments asynchronously. Actor-Critic, which combines the strengths of the policy-based and the value-based RL algorithms, predicts both the value function $V(s)$ and the optimal policy function $\pi(s)$. The learning agent uses the outputs of the value function to refine the policies of the actor. The advantage metric is given by Eq.(5.16). Its value indicates the extent to which an action is better or worse than the average of all actions under a specific state. This approach adopts the A3C algorithm to choose VNF placement actions and the Dijkstra algorithm to embed the VLs that connect adjacent VNFs.

Deep Q-Network (DQN)

The typical DQN algorithm [43] is adopted to select VNF placement actions, and the Dijkstra algorithm is utilized to embed the VLs that connect adjacent VNFs.

First-Fit (FFT)

This approach employs the First-Fit algorithm to embed VNFs and the Dijkstra (the shortest path in terms of distance) to determine the placement of VLs. FFT allocates VNFs to a partition that is first sufficient from the top of the main memory address. In our MDC environment, the higher index a DC has, the higher priority it will be given to accommodate VNFs. Given a VNF placement, the Dijkstra algorithm is utilized to determine the route between adjacent VNFs. The FFT has been selected as one of the benchmarks to measure performance in [3] and [36].

5.3.3 Simulation Results

We compare our proposed two-stage GCN-based RL algorithm, MAPPO in a partially observable environment (One-hop MAPPO), MAPPO in a fully observable environment (Fully-observable MAPPO), A3C, DQN, and FFT with respect to the convergence rate, overall acceptance ratio, total CPU cost, and average BW cost per SFC request.

In the first stage, we need to use an M-layer GCN to extract the features of the MDC network. A GCN with a large M means an extremely long training time, while a GCN with a small M may not be able to efficiently extract all features from the graph. To acquire the best value of M, we evaluate the performance of the GCN-based PPO with different numbers of GCN layers. It is shown in Fig. 5.7 that the GCN-based PPO with 3-layer GCN achieves a better result than that with 2 layers or 4 layers. Thus, we select the 3-layer GCN for our implementation.

The clipping ratio ϵ is a critical hyperparameter that we can tune to balance training stability and fast convergence. From Fig. 5.8, we can see that the GCN-based PPO with $\epsilon = 0.2$ converges faster than that with $\epsilon = 0.15$. Meanwhile, they attain approximately the same long-term cumulative reward, which is better than that obtained by the GCN-based PPO with $\epsilon = 0.25$. Thus, we set $\epsilon = 0.2$ for our

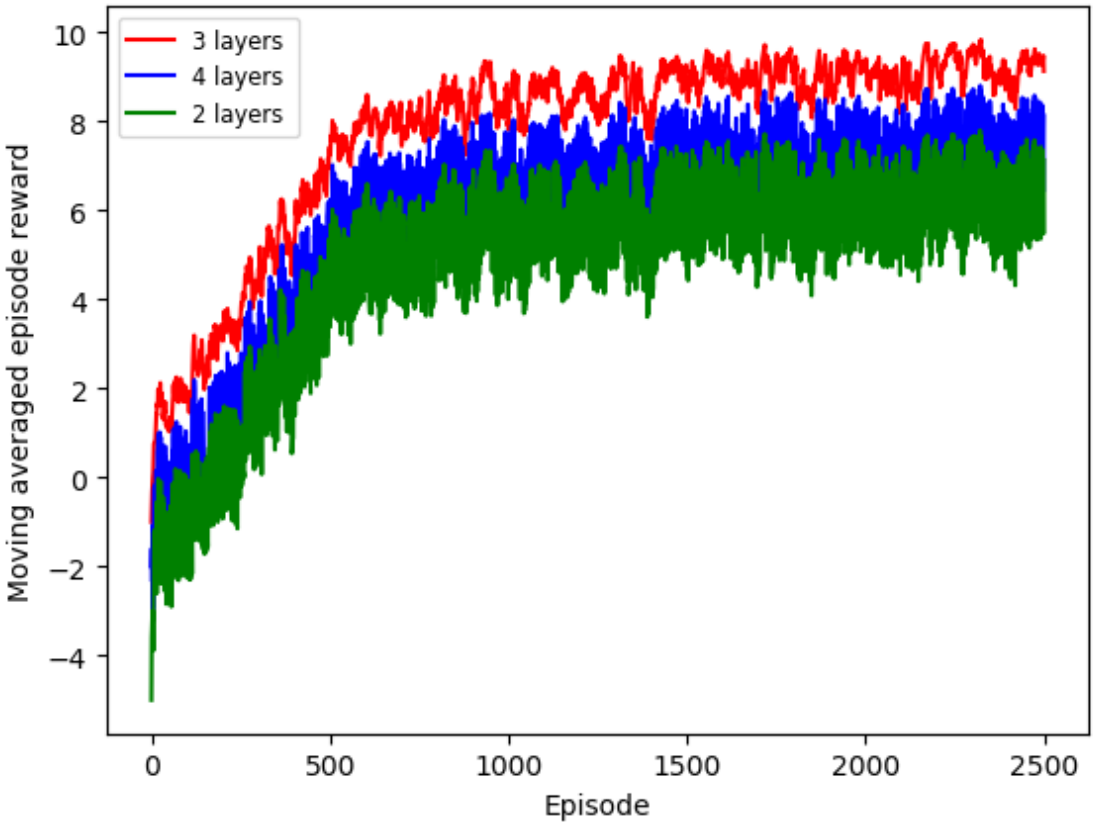

Figure 5.7 : The performance of our proposed GCN-based PPO with different numbers of GCN layers.

implementation.

In the second stage, for each agent in the MAPPO, the number of GCN layers is set to 1 or 2 for the policy network according to the scale of the agent's LOS. For the critic network, the number of GCN layers is set to 3 as we need to feed the encoded global state into it.

From Figs. 5.9 and 5.10, we can see that our proposed algorithm outperforms other baselines. It achieves similar performance to the fully-observable MAPPO while converging much faster. Our proposed algorithm sets an appropriate LOS for each agent, making it converge faster than the fully-observable MAPPO, whose agents can observe the whole environment. Specifically, each agent of the Fully-

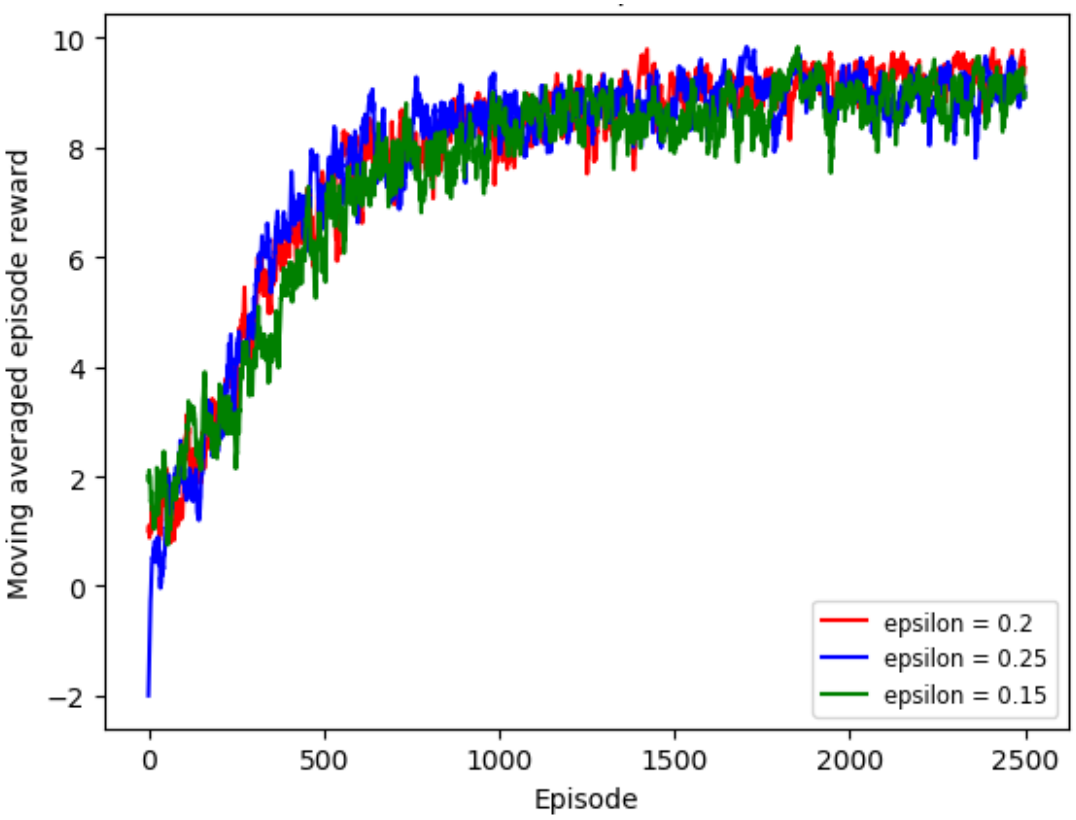

Figure 5.8 : The performance of our proposed GCN-based PPO with different clipping ratio hyperparameters.

observable MAPPO can place VNFs or VLs in every corner of the whole environment, leading to unnecessary competition that should have been avoided. In contrast, our proposed algorithm tailors an appropriate LOS for each agent, confining the competition to the smallest scope. These two algorithms showcase better performance than the others. The reason is that, given a VNF placement, they choose a route for embedding a VL from K shortest candidates. In this way, they explore the environment better and thus acquire better long-term cumulative rewards than the algorithms that adopt the Dijkstra algorithm to select a short-sighted optimal immediate reward in each time step. The One-hop MAPPO converges fastest because each agent of this algorithm can only exploit the resources within its one-hop neighbors' scope. However, it achieves an unfavorable long-term cumulative reward

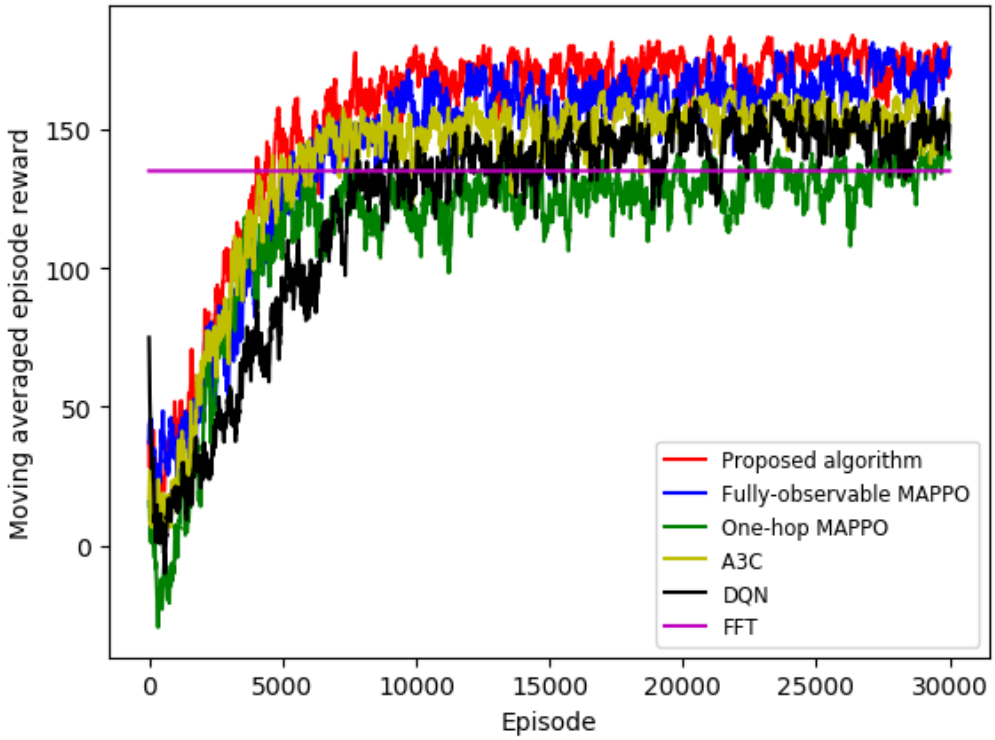

Figure 5.9 : The learning curve of six algorithms in 14-node NSFNET.

because of the restricted view of each agent.

The other three algorithms, which adopt the Dijkstra algorithm to determine the route option for a VL given a VNF placement, obtain worse accumulated rewards than our proposed algorithm and the Fully-observable MAPPO because of the greed for immediate rewards. The two DRL-based algorithms are better than the FFT. The reason is that the FFT allocates VNFs to a partition that is first sufficient from the top of the main memory address. In that case, the placement mode for VNFs is fixed. Given a VNF placement, it adopts the Dijkstra algorithm to embed a VL; thus, the placement mode for VLs is also fixed. In contrast, the A3C and DQN follow their logic to explore the environment, although they only consider the exploration of VNFs placement. Through interacting with the environment, they find the optimal

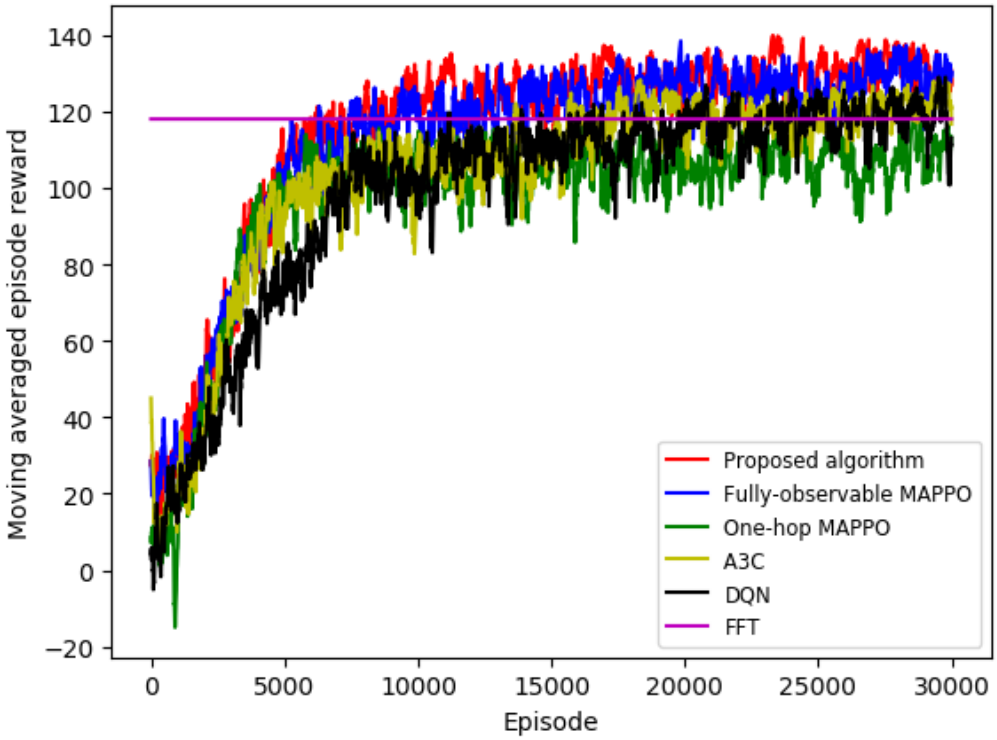

Figure 5.10 : The learning curve of six algorithms in 11-node COST239.

actions to place VNFs and VLs, obtaining better long-term cumulative rewards than the FFT. Another finding is that, of the two DRL algorithms, the A3C converges faster than the DQN. Two reasons can account for this result. First, the actor-critic framework follows the policy gradient to find the optimal action given a state. In comparison, the DQN assigns a score indicating the maximum expected future reward to each possible action, drastically increasing the time complexity. Second, the A3C asynchronously executes multiple agents in parallel on multiple instances of the environment. All agents work on their respective networks and update the global network asynchronously, accelerating the convergence rate.

During the evaluation phase, from Figs. 5.11(a) and 5.12(a), we can see that our proposed algorithm achieves approximately the same acceptance ratio of SFC

requests as the Fully-observable MAPPO. These two algorithms outperform all the other algorithms in terms of the acceptance ratio of SFC requests for the aforementioned reasons.

From Figs. 5.11(b) and 5.12(b), we can see that the outlines are similar to those of the overall acceptance ratio. The reason is that, in our environment, the more SFC requests an algorithm accepts, the more CPU resources it allocates to accommodate these SFC requests. Since the total CPU cost is proportional to the overall acceptance ratio for each algorithm and the outlines for these two parameters are approximately the same, the average CPU costs per SFC request for all algorithms have no significant difference.

From Figs. 5.11(c) and 5.12(c), we can see that the One-hop MAPPO has the least average traffic cost because each agent can only exploit resources within its one-hop neighbors' scope. Nevertheless, it consumes the least average BW per SFC request at the sacrifice of the most important objective: the overall acceptance ratio of the SFC requests. Thus, its great performance in saving BW cost is meaningless. Of all the other algorithms, our proposed algorithm and the Fully-observable MAPPO use the minimum average traffic cost for accommodating an SFC request. The aforementioned VLs embedding scheme plays a major role in accounting for this result. Furthermore, PPO-based algorithms tune a clip ratio, which controls the step size of policy updates, balancing training stability and fast convergence. This innovation ensures that PPO-based algorithms have better long-term cumulative rewards than the A3C and DQN.

5.4 Summary

In this chapter, we proposed a two-stage GCN-based DRL algorithm to maximize the overall acceptance ratio of SFC requests while minimizing the total cost in an MDC environment. We considered an MDC scenario where the arrival rates of SFC

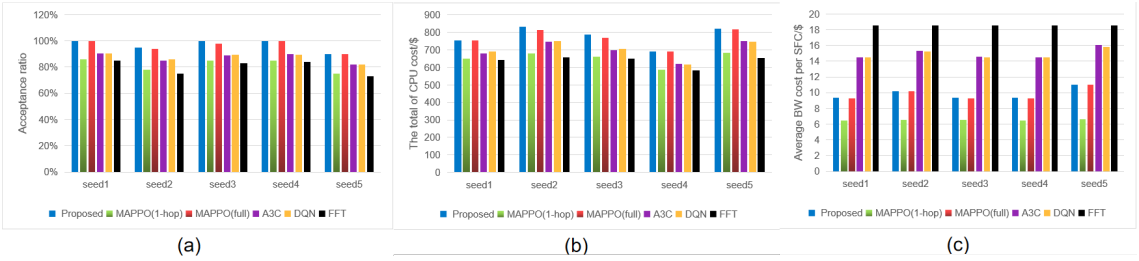

Figure 5.11 : Performance evaluation in 14-node NSFNET.

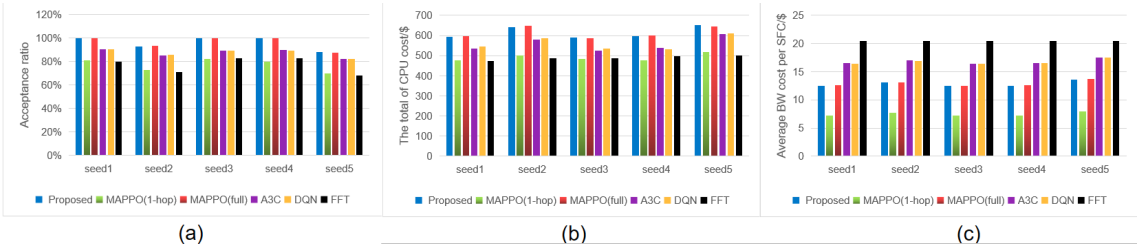

Figure 5.12 : Performance evaluation in 11-node COST239.

requests vary from DC to DC and developed an intelligent policy to place the VNFs and VLs requested by SFCs. To achieve the objective, we designed the GCN-based PPO framework in the first stage and the multi-agent framework in the second stage. Simulation results demonstrate that our proposed algorithm outperforms other baselines. It achieves similar performance to the fully-observable MAPPO while converging much faster. In summary, compared to state-of-the-art methods, our proposed scheme improves the overall acceptance ratio of SFC requests and cost-effectiveness. In the next chapter, we will propose an innovative approach for accommodating SFC in a fog environment.

Chapter 6

SFC Embedding Algorithm in a Fog Network

In this chapter, we present a graph convolutional networks (GCN)-based two-agent reinforcement learning (RL) SFC embedding scheme for a fog/cloud environment. We detach the placement of SFCs into two sub-actions. The N-agent chooses the locations for all VNFs, which will be notified to the R-agent. Then, the R-agent connects adjacent VNFs using VLs. Based on the workflow of our algorithm, we modify the typical multi-agent RL model. The new model can be applied to similar scenarios where the action includes two steps and the second sub-action depends on the first.

6.1 System Model and Problem Formulation

6.1.1 NFV Infrastructure

We consider a system architecture, as depicted in Fig. 6.1 and detailed in [28], consisting of the terminal, fog and cloud layers.

Terminal Layer

This layer contains terminal users, such as vehicles, laptops, and IoT devices, from which SFC requests are generated. The end devices in this layer connect fog nodes over wireless access links.

Fog Layer

This layer consists of fog nodes with limited resources. They process SFC requests from end-users at the network edge. Each fog node manages a group of

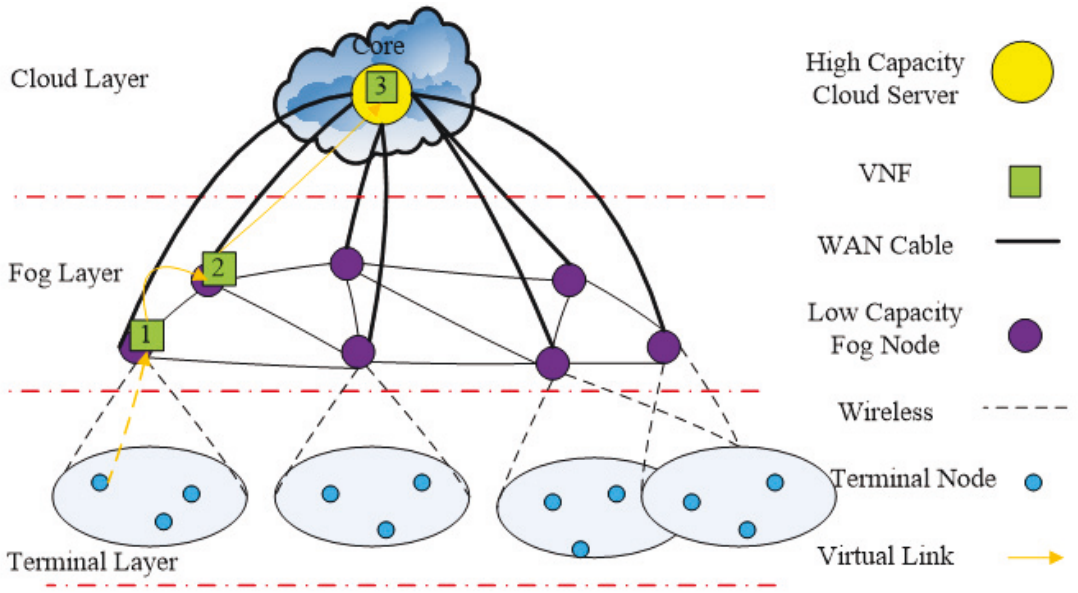

Figure 6.1 : Fog/cloud network architecture.

terminals and interconnects with its peers in this layer. Also, they connect cloud nodes over a wired link system.

Cloud Layer

This layer comprises cloud data center nodes with abundant computation and storage resources. These nodes are interconnected by high-bandwidth wide area network (WAN) backbone links. Cloud datacenters can process and store data at much lower costs than fog nodes. Nevertheless, this layer is located far from the network edge.

We model the fog/cloud network as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes from fog and cloud layer and $\mathcal{E}$ is the set of transmission links that interconnect these nodes. Hence, the numbers of nodes and physical links are $|\mathcal{V}|$ and $|\mathcal{E}|$, respectively. Furthermore, the CPU capacity and the remaining CPU resources of node v are denoted by $\{C_v | C_v \geq 0; v \in \mathcal{V}\}$ and $\{c_v | c_v \geq 0; v \in \mathcal{V}\}$, respectively. The storage capacity and the remaining storage resources of node v are denoted

by $\{M_v|M_v \geq 0; v \in \mathcal{V}\}$ and $\{m_v|m_v \geq 0; v \in \mathcal{V}\}$, respectively. The BW capacity and the remaining BW of link (v, u) are denoted by $\{B_{v,u}|B_{v,u} \geq 0, (v, u) \in \mathcal{E}\}$ and $\{b_{v,u}|b_{v,u} \geq 0, (v, u) \in \mathcal{E}\}$, respectively.

6.1.2 Characteristics of SFC Requests

SFC request i can be defined as $\mathrm{SR}_i = \{F_i, C_i, M_i, B_i, D_i, h_i\}$, where B_i symbolizes the BW requested by SFC i, $F_i = \{F_{i0}, F_{i1}, F_{i2}, ..., F_{iK_i}, F_{iK_{i+1}}\}$ (K_i is the number of VNFs in SFC i) denotes the source (F_{i0}), destination ($F_{iK_{i+1}}$), and VNFs $\{F_{i1}, F_{i2}, ..., F_{iK_i}\}$ that are used to compose SFC i, $C_i = \{C_{i1}, C_{i2}, ..., C_{iK_i}\}$ and $M_i = \{M_{i1}, M_{i2}, ..., M_{iK_i}\}$ indicate the CPU and storage resources requested by every VNF in SFC i, D_i is the delay requirement of SFC i, and h_i indicates whether SFC i requires physical isolation.

6.1.3 Cost Model

We adopt the typical cost model in [17] and [53] we have used in Chapter 5. As for the server node, we consider both CPU and storage resources.

6.1.4 Delay Model

We adopt the delay model in [28, 29] and [30].

6.1.5 Problem Formulation

In this part, we formulate the objective as an optimization problem. The purpose is to maximize the number of accepted SFC requests while minimizing the total cost, under infrastructure resources constraints.

The weighted sum of the accepted requests can be defined as:

$$u = \sum_{i=1}^{L} u_i = \sum_{i=1}^{L} f(D_i^R - D_i) * g(h_i, \boldsymbol{a}_i). \tag{6.1}$$

In the function above,

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0, \end{cases} \tag{6.2}$$

D_i^{R} and D_i respectively denote the latency requirement and the real latency of the i^{th} SFC,

$$g(x, \boldsymbol{y}) = \begin{cases} 1, & (x = 1 \text{ and } y_1 = \cdots = y_{K_i}) \\ & \text{or } (x = 0) \\ 0, & \text{otherwise}, \end{cases} \tag{6.3}$$

h_i denotes whether SFC i requires hard isolation, $\boldsymbol{a}_i$ symbolizes the locations of all VNFs in SFC i, and L is the number of SFC requests.

The total cost can be calculated as follows:

$$c = \sum_{i=1}^{L} c_i = \sum_{i=1}^{L} (c_{\mathrm{ser},i} + c_{\mathrm{tran},i}). \tag{6.4}$$

Thus, the optimization problem can be written as

$$(P1): \max_{x_{i,j}^v, z_{F_{ij},F_{i(j+1)}}^{v,u}} \quad \eta_1 u - \eta_2 c$$

$$\text{s.t.} \begin{cases} \sum_{i=1}^{L} \sum_{j=1}^{K_i} x_{i,j}^v C_{i,j} \leq C_v, \ v = 1, 2, \ldots, |\mathbb{V}| & (6.5a) \\ \sum_{i=1}^{L} \sum_{j=1}^{K_i} x_{i,j}^v M_{i,j} \leq M_v, \ v = 1, 2, \ldots, |\mathbb{V}| & (6.5b) \\ \sum_{v=1}^{|\mathbb{V}|} x_{i,j}^v = 1, \ i = 1, 2, \ldots, L, j = 1, 2, \ldots, K_i & (6.5c) \\ \sum_{i=1}^{L} \sum_{j=1}^{K_i} z_{F_{ij},F_{i(j+1)}}^{v,u} B_i \leq B_{v,u} \ \forall (v,u) \in \mathcal{E} & (6.5d) \\ x_{i,j}^v, z_{F_{ij},F_{i(j+1)}}^{v,u} \in \{0,1\}, \\ i = 1, 2, \ldots, L, j = 1, 2, \ldots, K_i, v = 1, 2, \ldots, |\mathbb{V}| \end{cases}$$

In the objective function above, η_1 and η_2 are the weights of two objectives. Since the objective of maximizing the number of accepted SFCs is more important, we assign a higher weight to it in our work.

In the constraints above, the binary variable $x_{i,j}^{v}$ indicates whether the j^{th} VNF of the i^{th} SFC nests on the v^{th} node, and the binary variable $z_{F_{ij},F_{i(j+1)}}^{v,u}$ indicates whether the VL between the j^{th} and $j+1^{\text{th}}$ VNF of the i^{th} SFC nests on the physical link (v,u). For any node in the infrastructure, the total of the required CPU resources of the embedded VNFs on it cannot exceed its CPU capacity, so we have Eq.(6.5a), and the total of the required storage of its embedded VNFs cannot exceed its storage capacity, so we have Eq.(6.5b). Each VNF can be deployed at only one server, so we have Eq.(6.5c). For any physical link, the total of the required bandwidth of its embedded VLs cannot exceed its link capacity, so we have Eq.(6.5d).

6.2 Proposed SFC Embedding Scheme

In this Section, we present our proposed algorithm. First, the background of the proposed algorithm is provided. Then, we detail our GCN-based two-agent PPO algorithm.

6.2.1 Background of the Proposed Algorithm

Model Our Environment as a Two-agent MDP

We model the SFC embedding problem as a two-agent MDP, which has been elaborated in Section 1.3.3 of Chapter 1. The details of the state representation, action space, and reward function for each agent in the context of our problem are provided in Section 6.2.2.

Overview of Multi-agent DRL and Motivation for Using two-agent PPO

The overview of MARL has been detailed in Section 5.2.1 of Chapter 5. For the SFC embedding issue, to compress the action-space and thus accelerate the convergence rate, most DRL-based algorithms use a single DRL-agent to explore the action-space of the VNFs placement. Typically, they chose the locations for all VNFs based on the neural network and designed a routing module to embed VLs given the embedded VNFs. In this way, a VNF placement action has only one corresponding VLs placement action. This scheme really compresses the action-space, but it neglects the action-space of the VLs embedding. One-step greedy algorithms for placing VLs may exhaust resources of some critical physical links and will always lead to an unfavorable global optimum solution. Hence, we detached the SFC embedding action into two sub-actions and devised a two-agent GCN-assisted DRL algorithm to sufficiently explore the joint action space of the VNFs and VLs placement.

Motivation for Using GCN

We have mentioned in Section 5.2.1 of Chapter 5.

6.2.2 The Components of the Two-agent MDP

Observation Representation

N-agent: The observation of N-agent ($\boldsymbol{o}_{1,t}$) includes three parts: (i) the requested CPU and storage resources of all VNFs of the current SFC to be processed, (ii) the physical isolation flag of the SFC, (iii) and the remaining CPU and storage resources of all physical nodes in the topology.

R-agent: The observation of R-agent ($\boldsymbol{o}_{2,t}$) includes three parts: (i) the locations of the VNFs embedded by the N-agent, (ii) the graph-structured information of $\mathcal{G}$

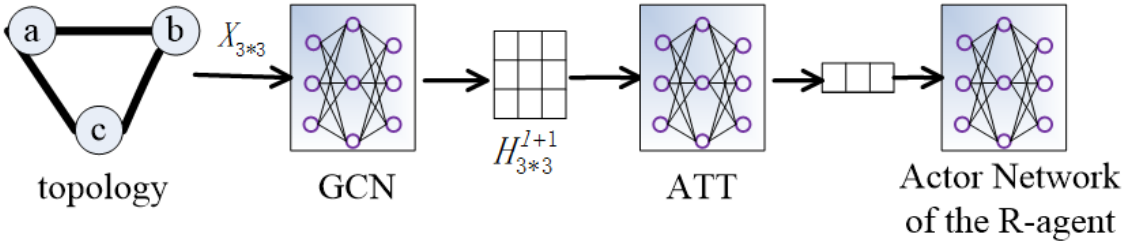

Figure 6.2 : Graph embedding process.

(excluding the node features), (iii) and the required BW and delay of the current SFC.

To feed the graph-structured information into the actor-network of the R-agent, we employ the GCN [63] to perform node-level encoding and the attention mechanism [67] to realize graph-level encoding. As a result, the graph-structured information is transformed into a real-valued vector. This process is described in Fig. 6.2. The typical multi-layer GCN with the layer-wise propagation rule is expressed as follows [63]:

$$\mathbf{H}^{(l+1)} = \sigma(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}).$$

Here, $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_{|\mathcal{V}|}$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections, where $\mathbf{I}_{|\mathcal{V}|}$ is the identity matrix. $\mathbf{W}^{(l)}$ is a layer-specific trainable weight matrix, and $\sigma(\cdot)$ denotes an activation function such as the ReLU$(\cdot)$. $\mathbf{H}^{(l)}$ is the matrix of activations in the l^{th} layer; $\mathbf{H}^{(0)} = \mathbf{X}$, where $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times Y}$ is the natural feature matrix, with each row indicating the Y-dimensional feature vector of a node.

In this work, the R-agent only extracts the features of the topology structure but not the nodes; thus, we set $\mathbf{X} = \mathbf{I}_{|\mathcal{V}|}$. Each element in the adjacency matrix $\mathbf{A}$ is weighted by the BW of the corresponding link.

Action Space

N-agent: N-agent is in charge of choosing locations for all VNFs of the current SFC. Hence, the action for VNFs placement is a multi-dimensional discrete action. Action $\boldsymbol{a}_{1,t} = [1, 5, 6]$ indicates that the current SFC comprises three VNFs, the first is placed on node 1, the second is on node 5, and the third is on node 6.

R-agent: R-agent is responsible for embedding VLs to connect all adjacent VNF pairs of the current SFC. The action for VLs placement is also a multi-dimensional discrete action. If action $\boldsymbol{a}_{2,t} = [2, 7, 8]$, then the first element 2 indicates that the VL between the first VNF and the second is embedded on the second route between these two VNFs.

Reward

The reward function consists of two sub-functions. Two agents cooperate to achieve a common goal; thus, they share a joint reward function.

The sub-function regarding the success/failure of an SFC request is defined as

$$R_a(t) = \begin{cases} g(h_t, \boldsymbol{a}_{1,t}), & D_t \geq d_t \\ 0, & D_t < d_t \\ -1, & \text{violating any resources constraint.} \end{cases}$$

In the function above,

$$g(h_t, \boldsymbol{a}_{1,t}) = \begin{cases} 1, & (h_t = 1 \text{ and } a_{1,1,t} = \cdots = a_{1,K_t,t}) \\ & \text{or } (h_t = 0) \\ 0, & \text{otherwise,} \end{cases}$$

$\boldsymbol{a}_{1,t}$ symbolizes the locations of all VNFs, h_t denotes whether the current SFC requires physical isolation, and D_t and d_t denote the delay requirement and the real

delay of the SFC, respectively. This sub-function resolves the physical isolation issue for security-related SFCs.

The sub-function for the total cost is defined as

$$\begin{aligned} R_c(t) =& \theta_1 \sum_{v \in \mathcal{V}} \sum_{j=1}^{K_t} x_{t,j}^{v} \cdot (C_{t,j} + M_{t,j}) \\ &+ \theta_2 \sum_{(v,u) \in \mathcal{E}} \sum_{j=1}^{K_t} z_{t,j,j+1}^{v,u} \cdot B_t \cdot \mathrm{Dis}\,(v,u)\,, \end{aligned}$$

where $x_{t,j}^{v}$ indicates whether the j^{th} VNF of the current SFC is placed on node v, $z_{t,j,j+1}^{v,u}$ indicates whether the VL between the j^{th} and $(j+1)^{\text{th}}$ VNFs nests on physical link (v,u), $\mathrm{Dis}\,(v,u)$ is the length of link (v,u), and θ_1 and θ_2 are the weights of the node cost and the traffic cost, respectively.

Considering our objective, the reward function $R(t)$ is expressed as a weighted sum of the two sub-functions:

$$R(t) = \eta_1 \cdot R_{\mathrm{a}}(t) - \eta_2 \cdot R_c(t).$$

Maximizing the acceptance ratio is a prerequisite for minimizing the total cost; thus, η_1 is greater than η_2.

State

We concatenate local observations of both agents to generate the global state: $\boldsymbol{s}_t = \{\boldsymbol{o}_{1,t}, \boldsymbol{o}_{2,t}\}$.

Joint Action

We concatenate the actions of both agents to generate the joint action: $\boldsymbol{a}_t = \{\boldsymbol{a}_{1,t}, \boldsymbol{a}_{2,t}\}$.

6.2.3 The Proposed Framework

The RL algorithm – GCN-based two-agent proximal policy optimization (PPO) is based on the framework of centralized training and decentralized execution (CTDE).

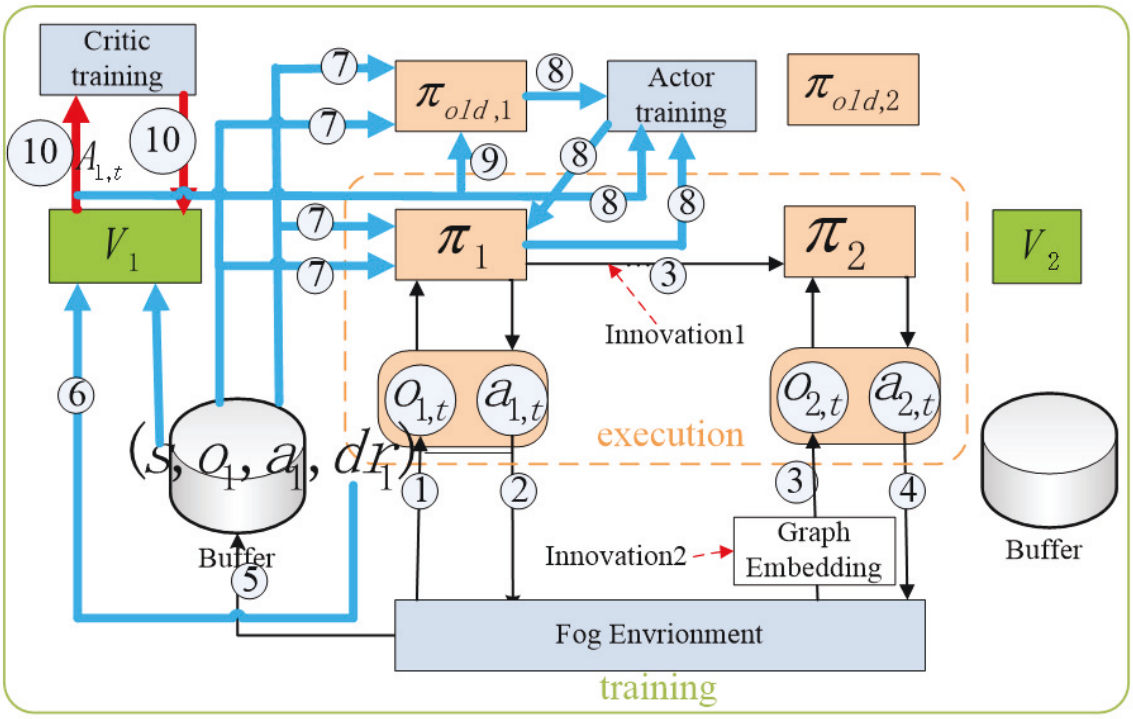

Figure 6.3 : A brief signal flow of our proposed scheme.

Its general overview is shown in Fig. 6.3. Each agent is equipped with a current actor network $\pi_\theta(\boldsymbol{a}_{m,t}|\boldsymbol{o}_{m,t})$, an old actor network $\pi_{\theta_{\text{old}}}(\boldsymbol{a}_{m,t}|\boldsymbol{o}_{m,t})$, and a critic network $V_{\phi_m}(\boldsymbol{s}_t)$, where $\boldsymbol{s}_t$ is composed of the local observations of both agents: $\boldsymbol{s}_t = \{\boldsymbol{o}_{1,t}, \boldsymbol{o}_{2,t}\}$. Steps 5-10 are the same for both agents; thus, we only plot the corresponding workflow for the first agent in the figure. The complete workflow for time step t of our proposed framework is summarized below. For any variable t or m in this framework, $t = 1, 2, \ldots, T$, and $m = 1, 2$.

1. Step 1: At time step t, N-agent obtains its local observation $\boldsymbol{o}_{1,t}$ and feeds it into actor network $\pi_{\theta_1}(\boldsymbol{a}_{1,t}|\boldsymbol{o}_{1,t})$.

2. Step 2: The output of actor network $\pi_{\theta_1}(\boldsymbol{a}_{1,t}|\boldsymbol{o}_{1,t})$, action $\boldsymbol{a}_{1,t}$, which indicates the placement of all VNFs, is performed in the environment.

3. Step 3: Action $\boldsymbol{a}_{1,t}$ is also forwarded to the R-agent and then concatenated

with the graph-structured information of the topology and the VL-related requirements of the current SFC to form $\boldsymbol{o}_{2,t}$, which is fed into actor network $\pi_{\theta_2}(\boldsymbol{a}_{2,t}|\boldsymbol{o}_{2,t})$.

4. Step 4: Action $\boldsymbol{a}_{2,t}$, the placement of all VLs connecting all adjacent VNFs pairs, is chosen and performed in the environment.

5. Step 5: In response to the joint action of both agents, a reward r_t and the global state $\boldsymbol{s}_t$, which consists of the observations of both agents, are returned to them. If time step t is the last step of an episode, the discounted reward for each step is calculated, the batch of samples collected in this episode $(\mathbf{s}, \mathbf{o}_m, \boldsymbol{dr}_m, \mathbf{a}_m)$ is stored into buffer m, and the flow proceeds to Step 6. Otherwise, it goes back to Step 1.

6. Step 6: Vector $\boldsymbol{s}_t$ is fed into the critic network $V_{\phi_m}(\boldsymbol{s}_t)$. Then, the output of $V_{\phi_m}(\boldsymbol{s}_t)$ and $dr_{m,t}$, are used to calculate the advantage $\hat{A}_{m,t}$ based on the equation

$$\hat{A}_{m,t} = \sum\nolimits_{t'>t} \gamma^{t'-t} r_{m,t'} - V_{\phi_m}(\boldsymbol{s}_t).$$

7. Step 7: Vectors $\boldsymbol{o}_{m,t}$ and $\boldsymbol{a}_{m,t}$ are fed into the current actor network and the old one to get $\pi_{\theta}(\boldsymbol{a}_{m,t}|\boldsymbol{o}_{m,t})$ and $\pi_{\theta_{\text{old}}}(\boldsymbol{a}_{m,t}|\boldsymbol{o}_{m,t})$, respectively.

8. Step 8: The weights of the current actor network are updated by maximizing the policy loss function

$$\begin{aligned} L^{\text{CLIP}}(\theta_m) = \sum_{t=1}^{T} \min\Big(&\frac{\pi_{\theta}(\boldsymbol{a}_{m,t}|\boldsymbol{o}_{m,t})}{\pi_{\theta_{\text{old}}}(\boldsymbol{a}_{m,t}|\boldsymbol{o}_{m,t})}\hat{A}_{m,t}, \\ &\text{clip}\Big(\frac{\pi_{\theta_m}(\boldsymbol{a}_{m,t}|\boldsymbol{o}_{m,t})}{\pi_{\theta_{m,\text{old}}}(\boldsymbol{a}_{m,t}|\boldsymbol{o}_{m,t})}, 1-\epsilon, 1+\epsilon\Big)\hat{A}_{m,t}\Big). \end{aligned}$$

9. Step 9: Copying the weights of the current actor network to the old one.

10. Step 10: The weights of the critic network are updated by minimizing the value loss function

$$L^{\mathrm{VL}}(\phi_m) = \sum_{t=1}^{T}(\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_{\phi_m}(\boldsymbol{s}_t))^2.$$

The proposed multi-agent algorithm is summarized in Algorithm 6. At the beginning of each episode, we release all the resources (line 4). At each time step, the N-agent places all the VNFs for the current SFC. After acquiring the location information of all VNFs from the N-agent, the R-agent embeds VLs to connect all adjacent VNFs. With the common objective of maximizing the acceptance ratio of SFC requests while minimizing the total cost, they cooperate with each other to accommodate SFCs step by step, creating a two-agent MDP environment. When no SFC comes, the end of an episode is reached (line 9). Then, a batch of samples $(s_{m,t}, a_{m,t}, o_{m,t}, dr_{m,t})$ $(t = 1, 2, \ldots, T.\ m = 1, 2)$, which will be used to train the critic network $(V_{\phi_m}(s_t))$ and the policy network $(\pi_{\theta_m}(a_{m,t}|o_{m,t}))$, is stored into buffer m for each of them. The batch of samples will be reused to train the critic network and the policy network for L epochs (lines 13-20). We train the neural networks episode by episode until their weights converge. Once the algorithm converges, both agents know how to cooperate with each other to maximize the acceptance ratio of SFC requests while minimizing the total cost.

6.3 Simulation

6.3.1 Simulation Setup

In this Section, we evaluate the performance of the proposed two-stage GCN-based DRL algorithm regarding the admission ratio and the cost-effectiveness under a multi-layer fog architecture in [28]. As in [28], the related node and link resource levels are also shown in Table 6.1.

Algorithm 6 The Proposed GCN-based two-agent PPO Algorithm

Create a thread for each agent
Initialize neural network parameters θ_m and ϕ_m for each agent
for $episode = 1, 2, \ldots$ **do**
  Release all the resources
  for $step = 1, 2, \ldots$ **do**
    The N-agent embeds VNFs of the current SFC
    The R-agent embeds VLs to connect VNFs
    Run policy π_{θ_m} for T timesteps, and collecting $\{s_{m,t},\ a_{m,t}, o_{m,t}, dr_{m,t}\}$
    Break the timesteps loop if no requests
  end for
  Estimate advantages $\hat{A}_{m,t} = \sum_{t'>t} \gamma^{t'-t} r_{m,t'} - V_\phi(s_{m,t})$
  $\pi_{\theta_{m,\text{old}}} \leftarrow \pi_{\theta_m}$
  for $l = 1, \ldots, L$ **do**
    $L^{CLIP}(\theta_m) = \sum_{t=1}^{T} \min(\frac{\pi_\theta(a_{m,t}|o_{m,t})}{\pi_{\theta_{\text{old}}}(a_{m,t}|o_{m,t})}\hat{A}_{m,t},\ \text{clip}(\frac{\pi_{\theta_m}(a_{m,t}|o_{m,t})}{\pi_{\theta_{m,\text{old}}}(a_{m,t}|o_{m,t})}, 1-\epsilon, 1+\epsilon)\hat{A}_{m,t})$
    Update θ_m via a gradient method w.r.t. $L^{CLIP}(\theta_m)$
  end for
  for $l = 1, \ldots, L$ **do**
    $L^{VL}(\phi_m) = \sum_{t=1}^{T}(\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_{m,t}))^2$
    Update ϕ via a gradient method w.r.t. $L^{VL}(\phi_m)$
  end for
end for

Table 6.1 : Network Parameters.

Parameter	Value
Number of nodes (fog, cloud):	(20,8)
Number of terminals:	23
Node CPU capacity (fog, cloud):	(15,100)
Node storage capacity (GB) (fog, cloud):	(32,128)
VNF required CPU, storage:	Rand(1,3), Rand(1,5)
Request bandwidth:	Rand(1,10) Mbps
Number of VNFs in a request:	Rand(3,5)
Link bandwidth: fog-fog, cloud-fog, cloud-cloud:	0.3,10,20 Gbps
Link delay(ms): terminal-fog, fog-fog, fog-cloud, cloud-cloud	Rand $[1-2], [0.6, 2.1]$, $[4-14], [14-34]$
VNF processing delay (ms): Fog, cloud	Rand[0.03,1], [1.03,2.03]
Delay requirement:	10ms
Cost coefficient in fog layer(θ_1, θ_2):	(0.5,0.5)
Cost coefficient in cloud layer(θ_1, θ_2):	(0.1,0.1)

Different from [28, 30] and [29], who randomly choose a terminal node to generate an SFC request. We set a probability for each node to generate an SFC. The probability varies from node to node. Hence, the SFC requests load is not uniformly distributed across the network.

Regarding the settings for neural networks, we adopt the common hyperparameters for multi-agent PPO across all domains provided by [62].

6.3.2 Algorithms Used for Comparison

Hierarchy Descending SFC Embedding Scheme With Load Balancing (HDWLB)

This algorithm consists of three phases. In the first phase, for the source node of an SFC request, the closest node in the higher layer is allocated and treated as the destination node for it. In the second phase, if there are sufficient resources in the destination node, the algorithm implements group mapping to embed all the VNFs of this SFC request onto the destination node. Otherwise, it adopts a single VNF mapping procedure to map the VNFs one after another on the nodes in the lower layer. The current VNF is placed on the closest node to its previous VNF. During the third phase, load balancing is conducted across the low layer to complete node saturation.

Single-agent PPO

The typical PPO algorithm [61] is adopted to select VNF placement actions, and the Dijkstra algorithm is utilized to embed the VLs that connect adjacent VNFs.

6.3.3 Simulation Results

We compare our proposed algorithm, the HWDLB [30], and the Single-agent PPO (SAPPO) with respect to the acceptance ratio of all SFC requests as well as the average cost and delay per accepted SFC request.

From Fig. 6.4, we can see that our proposed scheme outperforms other baselines. The HDWLB algorithm first chooses the closest cloud node and then implements group mapping if the chosen cloud node has enough resources. Hence, when the number of SFC requests is small, the HDWLB can map all SFC requests onto the cloud layer and achieve a low average cost per accepted SFC. Nevertheless, the average cost increases drastically as the number of SFC requests increases. This is

because the SFC requests are not uniformly distributed in the considered scenario. With the increasing number of SFC requests, some SFCs from 'hot' locations may not be placed on the cloud layer under the group mapping scheme because their corresponding closest cloud nodes are overloaded. Alternatively, the HDWLB conducts single mapping to embed VNFs one after another onto fog nodes, which are more expensive than cloud nodes. In comparison, the VNFs placement solutions of the two PPO-based algorithms are much more flexible. For SFCs from 'hot' locations, they can place some VNFs on fog nodes and others on cloud nodes. In this way, more cloud nodes, which are cheaper than fog nodes, can be exploited; thus, the average cost per accepted SFC will be lower than that of the HDWLB. Our proposed algorithm performs better than the SAPPO. The reason is that, given a VNF placement of an SFC, the SAPPO algorithm adopts the Dijkstra algorithm to choose an 'optimal' solution to embed VLs. Consequently, it neglects to explore the action space of the VLs placement. In contrast, our proposed algorithm employs two agents to explore the action space of both the VNFs placement and the VLs placement. Therefore, it sufficiently explores the action space of the SFC embedding and thus acquires a better global optimum than the SAPPO that adopts the Dijkstra algorithm to greedily select a short-sighted immediate optimum in each time step.

From Fig. 6.5, we can see that our proposed algorithm is also superior to other baselines. The HDWLB algorithm is designed under the assumption that SFC requests are distributed uniformly. Therefore, its performance degrades in the unbalanced loads scenario. Some cloud nodes far from the 'hot' locations cannot be utilized even when some SFCs from 'hot' locations have to nest on the expensive fog layer or be rejected. Furthermore, The HDWLB algorithm assumes that the BW is abundant; thus, it only considers the load balancing of node resources but overlooks the BW. Nevertheless, in the considered scenario, the BW is also limited. Hence, the HDWLB obtains the lowest acceptance ratio. Two PPO-based algorithms perform

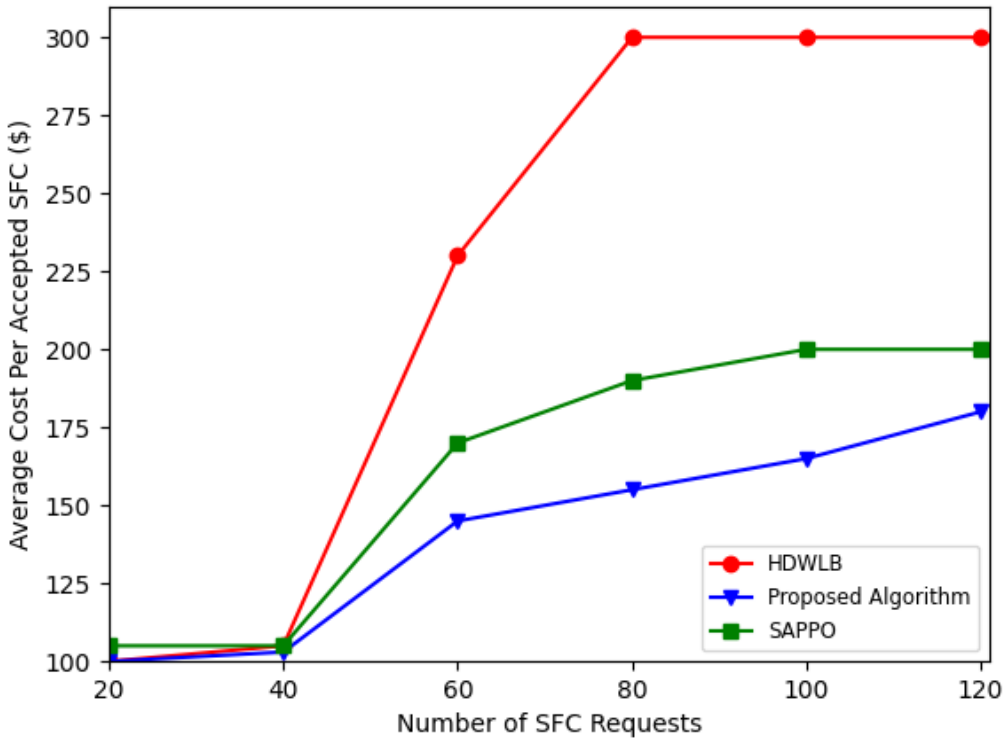

Figure 6.4 : The average cost comparison of three algorithms.

actions and acquire knowledge from the environment. The abilities to learn from the dynamic environment make them adaptive. As the number of SFC requests increases, our proposed algorithm achieves higher acceptance ratios than the SAPPO because of more extensive action-space exploration.

For a delay-sensitive service, the most critical issue is to satisfy the delay requirement. Thus, for an SFC, we are concerned about whether the real delay is shorter than the delay requirement but not how short it is. From Fig. 6.6, we can see that when the number of requests increases from 40 to 60, the average delay of the HDWLB suffers a noticeable decline, while the delays of the other two are still close to the requirement. The reason is that, with the increasing number of requests, the HDWLB embeds newly arrived SFCs on the fog layer, while the two

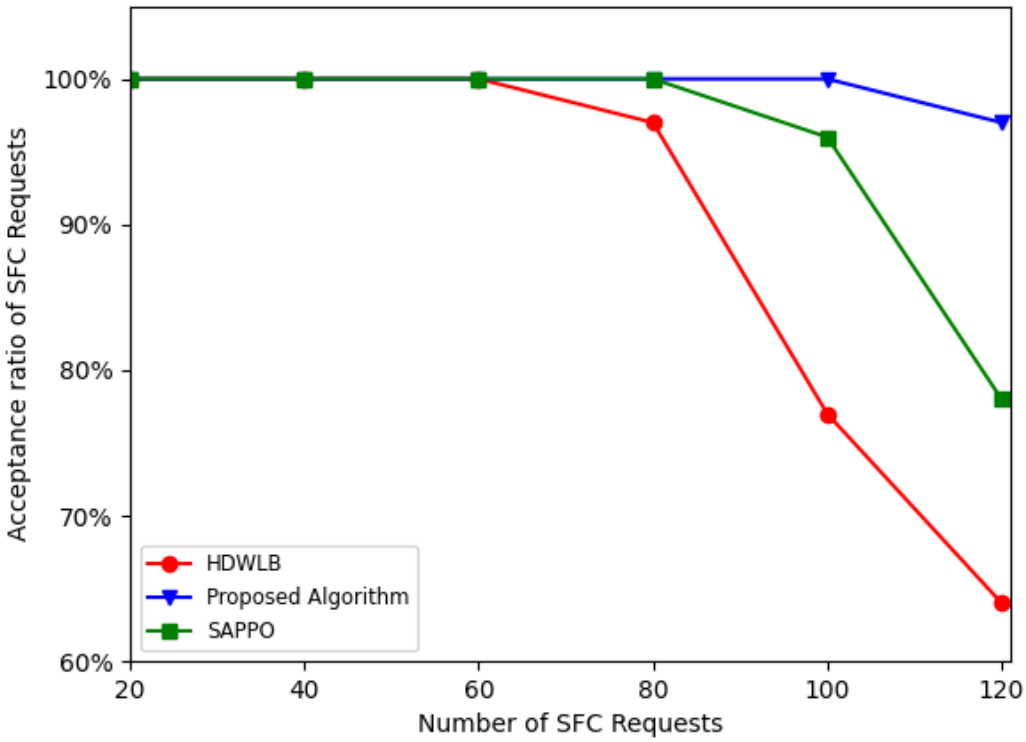

Figure 6.5 : The acceptance ratio comparison of three algorithms.

PPO-based algorithms can still place a proportion of VNFs onto the cloud layer. As the number of requests increases from 60 to 80, the figure of the HDWLB goes on falling at a little bit lower rate than in the previous interval because the percentage of short-delay SFCs increases. In comparison, the figure of the SAPPO decreases more severely. The reason is that the SAPPO embeds as many VNFs on a fog node as possible, achieving a delay much shorter than the requirement. Our proposed algorithm, which outperforms the SAPPO in exploring the action space, can still rent resources from the cloud layer to place SFCs. Thus, its average delay is still close to the requirement. Within the [80,100] interval, the HDWLB cannot accept more SFC requests; hence, its figure does not change. As for the SAPPO, the rate of descent is slightly lower than in the previous interval since the percentage of short-delay SFCs

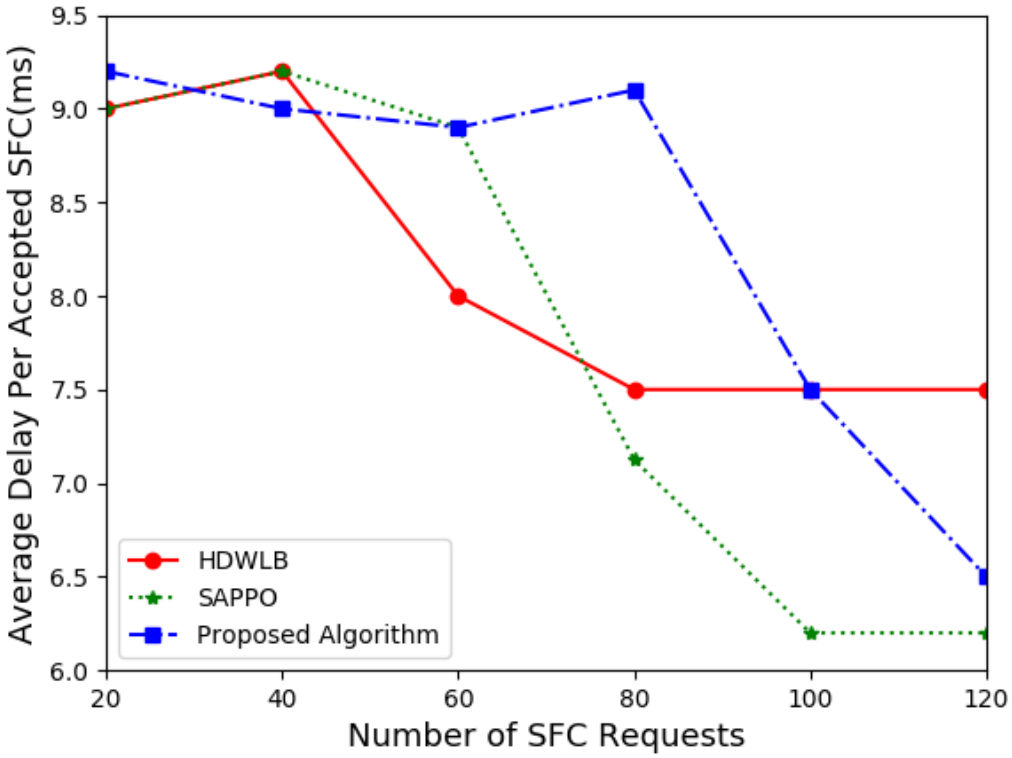

Figure 6.6 : The average delay comparison of three algorithms.

increases. Our proposed algorithm starts to embed SFCs on the fog layer, and thus its average delay starts to drop.

6.4 Summary

In this chapter, we proposed a GCN-based two-agent PPO algorithm to maximize the acceptance ratio of SFC requests while minimizing the total cost in a fog/cloud environment. We considered a fog/cloud architecture where the SFC request loads vary from location to location, and some security-related SFC requests require physical isolation from others. Accordingly, we developed an intelligent GCN-based two-agent PPO framework to place the VNFs and VLs requested by SFCs, achieving notable improvements in the acceptance ratio of SFC requests and

cost-effectiveness. Simulation results demonstrate that our proposed algorithm outperforms other state-of-the-art baselines in acceptance ratio, cost, and delay. In the next chapter, we will summarize our contributions and discuss our future work.

Chapter 7

Conclusions and Future Work

7.1 Summary of Outcomes

Considering the problems and challenges in the SFC embedding issue, we designed four DRL-based SFC embedding algorithms for three typical scenarios. With theory analysis and simulation, we have justified our algorithms and made some contributions to this field.

7.2 Recommendations & Future Work

As for future work, we will design a multi-agent DRL for the SFC embedding issue in the inter-DC EONs. Since the spectrum assignment scheme of the lightpath should comply with the spectrum contiguous, non-overlapping, and continuous constraints, the new algorithm could be different from the one we designed in Chapter 5. Another field of future interest could be the multi-agent DRL framework. We will design a distributed training framework and compare it with the CTDE framework used in this paper. Further, we plan to try to apply our MARL algorithms to address the data transfer issue in MDC in an emergency.

Appendix

APPENDIX A. PROOF OF PROPOSITION 1

Based on Eq.(4.20b), we have

$$\begin{cases} \sum_{j=1}^{K_i} a_j \times \mu_{i,j} = c_i - \alpha \times 250 \times N_i & (7.1a) \\ a_j = \gamma \frac{M_{\text{tot}}}{C_{i,j}^{\text{pro}}} + \beta \frac{W_{\text{tot}} \times h\left(v_j, v_{j+1}\right)}{C_{i,j}^{\text{tran}}}, \\ j = 1, 2, \ldots, K_i. & (7.1b) \end{cases}$$

According to Eq.(7.1a), no matter what the optimal solution for P2 in Chapter 4 is, we can conclude that the optimal solution $\widetilde{\boldsymbol{\mu}}_{\boldsymbol{i}}$ satisfies

$$\begin{cases} a_j \times \widetilde{\mu}_{i,j} = q_j \times (c_i - \alpha \times 250 \times N_i), \\ j = 1, 2, \ldots, K_i & (7.2a) \\ \sum_{j=1}^{K_i} q_j = 1 & (7.2b) \\ 0 < q_j < 1,\ j = 1, 2, \ldots, K_i. & (7.2c) \end{cases}$$

Substituting the optimal solution $\widetilde{\boldsymbol{\mu}}_{\boldsymbol{i}}$ into P2, we have

$$\begin{aligned} D_i^{min}(\widetilde{\boldsymbol{\mu}}_{\boldsymbol{i}}) &= \frac{\lambda_i}{2\left(\widetilde{\mu}_{i,1}\right)^2\left(1-\frac{\lambda_i}{\widetilde{\mu}_{i,1}}\right)} + \sum_{j=1}^{K_i}\frac{2}{\widetilde{\mu}_{i,j}} \\ &+ \sum_{j=1}^{K_i}\frac{2(h(v_j,v_{j+1})-1)}{\widetilde{\mu}_{i,j}} g(h(v_j,v_{j+1})) \\ &+ \sum_{j=2}^{K_i}\frac{\lambda_i}{2\left(\widetilde{\mu}_{i,j}\right)^2\left(1-\frac{\lambda_i}{\widetilde{\mu}_{i,j}}\right)} g(\widetilde{\mu}_{i,j-1}-\widetilde{\mu}_{i,j}) \\ &= \frac{1}{2}\left(\frac{1}{\widetilde{\mu}_{i,1}-\lambda_i}-\frac{1}{\widetilde{\mu}_{i,1}}\right) + \sum_{j=1}^{K_i}\frac{2}{\widetilde{\mu}_{i,j}} \\ &+ \sum_{j=1}^{K_i}\frac{2(h(v_j,v_{j+1})-1)}{\widetilde{\mu}_{i,j}} g(h(v_j,v_{j+1})) \\ &+ \sum_{j=2}^{K_i}\frac{1}{2}\left(\frac{1}{\widetilde{\mu}_{i,j}-\lambda_i}-\frac{1}{\widetilde{\mu}_{i,j}}\right) g(\widetilde{\mu}_{i,j-1}-\widetilde{\mu}_{i,j}). \end{aligned} \tag{7.3}$$

Let

$$D_{i1}^{min}(\widetilde{\mu}_{i,j}) = \frac{1}{\widetilde{\mu}_{i,j}-\lambda_i} - \frac{1}{\widetilde{\mu}_{i,j}},\ j=1,2,\ldots,K_i, \tag{7.4}$$

$$D_{i2}^{min}(\widetilde{\mu}_{i,j}) = \frac{2}{\widetilde{\mu}_{i,j}},\ j=1,2,\ldots,K_i. \tag{7.5}$$

Then, we have

$$\begin{aligned} D_i^{min}(\widetilde{\boldsymbol{\mu}}_{\boldsymbol{i}}) &= \frac{D_{i1}^{min}(\widetilde{\mu}_{i,j})}{2} + \sum_{j=2}^{K_i}\frac{D_{i1}^{min}(\widetilde{\mu}_{i,j})}{2} g(\widetilde{\mu}_{i,j-1}-\widetilde{\mu}_{i,j}) \\ &+ \sum_{j=1}^{K_i} D_{i2}^{min}(\widetilde{\mu}_{i,j})(h(v_j,v_{j+1})-1) g(h(v_j,v_{j+1})) \\ &+ \sum_{j=1}^{K_i} D_{i2}^{min}(\widetilde{\mu}_{i,j}),\ j=1,2,\ldots,K_i. \end{aligned} \tag{7.6}$$

Substituting Eq.(7.1b) and (7.2a) into Eq.(7.4) and (7.5) and then differentiating D_{i1}^{min} and D_{i2}^{min} with respect to c_i, we have

$$\begin{aligned} \frac{\mathrm{d}D_{i1}^{min}(c_i)}{\mathrm{d}c_i} &= \frac{a_j}{q_j}\left(\frac{1}{(c_i-250\times\alpha\times N_i)^2}\right. \\ &\left. - \frac{1}{(c_i-250\times\alpha\times N_i-\frac{a_j}{q_j}\lambda_i)^2}\right), \end{aligned} \tag{7.7}$$

$$\frac{\mathrm{d}D_{i2}^{min}(c_i)}{\mathrm{d}c_i} = \frac{a_j}{q_j} \frac{-1}{(c_i - 250 \times \alpha \times N_i)^2}. \tag{7.8}$$

Based on Eq.(7.1b) and (7.2c), we know $a_j > 0$ and $q_j > 0$. So we can conclude that $\frac{\mathrm{d}D_{i1}^{min}(c_i)}{\mathrm{d}c_i} < 0$ and $\frac{\mathrm{d}D_{i2}^{min}(c_i)}{\mathrm{d}c_i} < 0$. Furthermore, based on Eq.(4.21), we know $g(\mu_{i,j-1} - \mu_{i,j}) = 1$ or 0, and $g(h(v_j, v_{j+1})) = 1$ or 0. At this stage, we can conclude that $\frac{\mathrm{d}D_i^{min}(c_i)}{\mathrm{d}c_i} < 0$, and thus $D_i^{\min}(c_i)$ is a monotone decreasing function of c_i.

APPENDIX B. PROOF OF PROPOSITION 2

The standard form of an optimization problem can be expressed as [74]:

$$\min f_0(x)$$

$$\text{s.t.} \begin{cases} f_i(x) \leq 0, & i = 1, 2, \ldots, m \\ h_i(x) = 0, & i = 1, 2, \ldots, p. \end{cases} \tag{7.9}$$

The problem is to find an $\mathbf{x}$ that minimizes $f_0(\mathbf{x})$ among all $\mathbf{x}$ that satisfy the conditions $f_i(\mathbf{x}) \leq 0, i = 1, \ldots, m$, and $h_i(\mathbf{x}) = 0, i = 1, \ldots, p$. To be a convex optimization problem, it needs to satisfy three additional requirements [74]:

1. the objective function must be convex,
2. the inequality constraint functions must be convex,
3. the equality constraint functions must be affine.

Now let's prove that our problem satisfies these three requirements and thus is a convex optimization problem.

Objective function is convex

In our problem, the objective function shown below is twice differentiable; that is to say, its Hessian or second derivative exists at each point in dom D_i, which is

open. Then D_i is convex if and only if dom D_i is convex and its Hessian is positive semidefinite [74].

$$\begin{aligned} D_i(\boldsymbol{\mu_i}) = & \frac{\lambda_i}{2\left(\mu_{i,1}\right)^2\left(1-\frac{\lambda_i}{\mu_{i,1}}\right)} + \sum_{j=1}^{K_i}\frac{2}{\mu_{i,j}} \\ & + \sum_{j=1}^{K_i}\frac{2(h(v_j,v_{j+1})-1)}{\mu_{i,j}} g(h(v_j,v_{j+1})) \\ & + \sum_{j=2}^{K_i}\frac{\lambda_i}{2\left(\mu_{i,j}\right)^2\left(1-\frac{\lambda_i}{\mu_{i,j}}\right)} g(\mu_{i,j-1}-\mu_{i,j}). \end{aligned}$$

The hessian matrix of D_i can be expressed as

$$H(D_i) = \begin{bmatrix} \frac{\partial^2 D_i}{\partial \mu_{i,1}^2} & \frac{\partial^2 D_i}{\partial \mu_{i,1}\partial \mu_{i,2}} & \cdots & \frac{\partial^2 D_i}{\partial \mu_{i,1}\partial \mu_{i,K_i}} \\ \frac{\partial^2 D_i}{\partial \mu_{i,2}\partial \mu_{i,1}} & \frac{\partial^2 D_i}{\partial \mu_{i,2}^2} & \cdots & \frac{\partial^2 D_i}{\partial \mu_{i,2}\partial \mu_{i,K_i}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 D_i}{\partial \mu_{i,K_i}\partial \mu_{i,1}} & \frac{\partial^2 D_i}{\partial \mu_{i,K_i}\partial \mu_{i,2}} & \cdots & \frac{\partial^2 D_i}{\partial \mu_{i,K_i}^2} \end{bmatrix},$$

where

$$\begin{aligned} \frac{\partial^2 D_i}{\partial \mu_{i,1}^2} = & \frac{3+4(h\left(v_1,v_2\right)-1)\times g(h(v_1,v_2))}{\mu_{i,1}^3} \\ & + \frac{1}{\left(\mu_{i,1}-\lambda_i\right)^3}, \end{aligned}$$

when $2 \le j \le K_i$,

$$\frac{\partial^2 D_i}{\partial \mu_{i,j}^2} = \begin{cases} \frac{1}{\left(\mu_{i,j}-\lambda_i\right)^3} + \frac{3+4(h(v_j,v_{j+1})-1)\times g(h(v_j,v_{j+1}))}{\mu_{i,j}^3}, \\ \quad \mu_{i,j} < \mu_{i,j-1} \\ \frac{4+4(h(v_j,v_{j+1})-1)\times g(h(v_j,v_{j+1}))}{\mu_{i,j}^3}, \ \mu_{i,j} \ge \mu_{i,j-1}, \end{cases}$$

and

$$\frac{\partial^2 D_i}{\partial \mu_{i,j}\partial \mu_{i,j'}} = 0, \ j \ne j'.$$

In the M/D/1 model, obviously, $\mu_{i,j} > \lambda_i$ is satisfied for any j. So dom $D_i = \{\boldsymbol{\mu_i} \in R^{K_i} | \mu_{i,j} > \lambda_i, j = 1,\ldots,K_i\}$ is convex. Based on Eq.(4.21), we know

$g(h(v_j, v_{j+1})) = 1$ or 0. So we have $\frac{\partial^2 D_i}{\partial \mu_{i,j}^2} > 0, j = 1, \ldots, K_i$, and thus $H(D_i)$ is a positive-definite matrix. At this stage, we can conclude that the objective function is convex.

Inequality constraint functions are convex

The standard form of inequality constraints in our problem can be expressed as

$$f_j(\boldsymbol{\mu_i}) = \mu_{i,j} - \mu_{i,j+1} \leq 0,\ j = 1, 2, \ldots, K_i - 1, \text{ or}$$

$$f_j(\boldsymbol{\mu_i}) = \mu_{i,j+1} - \mu_{i,j} \leq 0,\ j = 1, 2, \ldots, K_i - 1,$$

$$f_{K_i - 1 + n}(\boldsymbol{\mu_i}) = \sum_{j=1}^{K_i} x_{i,j}^n \times \frac{\mu_{i,j}}{C_{i,j}^{\text{pro}}} \times M_{\text{tot}} - m_n \leq 0,$$

$$n = 1, 2, \ldots, |\mathbb{S}|,$$

$$f_{K_i + |\mathbb{S}| - 1 + n}(\boldsymbol{\mu_i}) = \sum_{j=1}^{K_i} x_{i,j}^n \times \frac{\mu_{i,j}}{C_{i,j}^{\text{tran}}} \times W_{\text{tot}} - w_n \leq 0,$$

$$n = 1, 2, \ldots, |\mathbb{S}|.$$

Evidently, for any constraint function, dom $f_j = \{\boldsymbol{\mu_i} \in R^2 | \mu_{i,j} > \lambda_i, \mu_{i,j+1} > \lambda_i\},\ j = 1, 2, \ldots, K_i - 1$ or dom $f_n = \{\boldsymbol{\mu_i} \in R^L | \mu_{i,1} > \lambda_i, \ldots, \mu_{i,L} > \lambda_i\},\ n \geq K_i$ (suppose that L VNFs nest on the n^{th} server) is convex. Furthermore, its Hessian is a zero matrix and thus positive semidefinite. Hence, inequality constraints are all convex.

Equality constraint function is affine

When service i is provided with a constant cost C_i, the standard form of the equality constraint in our problem can be expressed as

$$h_1(\boldsymbol{\mu_i}) = \gamma \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\text{pro}}} \times M_{\text{tot}} + \beta \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\text{tran}}} \times W_{\text{tot}} \times h\left(v_j, v_{j+1}\right) + \alpha \times 250 \times N_i - C_i = 0.$$

According to the definition of affine: A set C is affine if the line through any two distinct points in C lies in C[74], we know that any line or line segment is affine. Therefore, the equality constraint in our problem is affine.

Bibliography

[1] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, "The 5g-enabled tactile internet: Applications, requirements, and architecture," in *2016 IEEE Wireless Communication and Networking Conference Workshops (WCNCW)*, Doha, Qatar, Apr. 2016, pp. 1–6.

[2] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y. A. Zhang, "The roadmap to 6g - ai empowered wireless networks," Jul. 2019. [Online]. Available: https://arxiv.org/pdf/1904.11686

[3] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A deep reinforcement learning approach for vnf forwarding graph embedding," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1318–1331, Dec. 2019.

[4] Y. Yang and J. Wang, "An overview of multi-agent reinforcement learning from game theoretical perspective," Mar. 2021. [Online]. Available: https://arxiv.org/abs/2011.00583

[5] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," Mar. 2020. [Online]. Available: https://arxiv.org/abs/1706.02275

[6] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, Mar. 2016.

[7] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, third quarter 2018.

[8] J. Gil Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[9] S. Mostafavi, V. Hakami, and M. Sanaei, "Quality of service provisioning in network function virtualization: A survey," *Computing*, vol. 103, pp. 917–991, Mar. 2021.

[10] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1409–1434, Mar. 2019.

[11] A. Baumgartner, V. S. Reddy, and T. Bauschert, "Combined virtual mobile core network function placement and topology optimization with latency bounds," in *2015 Fourth European Workshop on Software Defined Networks*, Bilbao, Spain, Sep.-Oct. 2015, pp. 97–102.

[12] D. B. Oljira, K. Grinnemo, J. Taheri, and A. Brunstrom, "A model for qos-aware vnf placement and provisioning," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Berlin, Germany, Nov. 2017, pp. 1–7.

[13] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *2015 IFIP/IEEE International Symposium on*

Integrated Network Management (IM), Ottawa, ON, Canada, May 2015, pp. 98–106.

[14] Q. Zhang, Y. Xiao, F. Liu, J. C. S. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, Atlanta, GA, USA, Jun. 2017, pp. 731–741.

[15] H. Tang, D. Zhou, and D. Chen, "Dynamic network function instance scaling based on traffic forecasting and vnf placement in operator data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 530–543, Mar. 2019.

[16] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 533–546, Sep. 2016.

[17] C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach," *IEEE Transactions on Services Computing*, vol. 13, no. 1, pp. 172–185, Jan.-Feb. 2020.

[18] M. M. Tajiki, S. Salsano, L. Chiaraviglio, M. Shojafar, and B. Akbari, "Joint energy efficient and qos-aware path allocation and vnf placement for service function chaining," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 374–388, Mar. 2019.

[19] M. Dieye *et al.*, "Cpvnf: Cost-efficient proactive vnf placement and chaining for value-added services in content delivery networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 774–786, Jun. 2018.

[20] A. Varasteh, B. Madiwalar, A. V. Bemten, W. Kellerer, and C. Mas-Machuca, "Holu: Power-aware and delay-constrained vnf placement and chaining," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1524–1539, Jun. 2021.

[21] Q. Ye, W. Zhuang, X. Li, and J. Rao, "End-to-end delay modeling for embedded vnf chains in 5g core networks," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 692–704, Feb. 2019.

[22] W. Fang, M. Zeng, X. Liu, W. Lu, and Z. Zhu, "Joint spectrum and it resource allocation for efficient vnf service chaining in inter-datacenter elastic optical networks," *IEEE Communications Letters*, vol. 20, no. 8, pp. 1539–1542, Aug. 2016.

[23] A. Khatiri and G. Mirjalily, "Resource balanced service chaining in nfv-enabled inter-datacenter elastic optical networks," in *2020 12th International Conference on Knowledge and Smart Technology (KST)*, Pattaya, Thailand, Jan.-Feb. 2020, pp. 168–171.

[24] Y. Li *et al.*, "Joint balancing of it and spectrum resources for selecting virtualized network function in inter-datacenter elastic optical networks," *Opt Express*, vol. 27, no. 11, pp. 15 116–15 128, May 2019.

[25] J. Pei, P. Hong, K. Xue, and D. Li, "Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2179–2192, Oct. 2019.

[26] Open Science Data Cloud, "Virtual machines (vms)," 2014. [Online]. Available: https://www.opensciencedatacloud.org/support/instances.html

[27] Kubernetes Documentation, "Kubernetes: Resource management for pods and containers," Jun. 2022. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

[28] N. Siasi, A. Jaesim, and N. Ghani, "Tabu search for efficient service function chain provisioning in fog networks," in *2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC)*, Los Angeles, CA, USA, Dec. 2019, pp. 145–150.

[29] N. Siasi, M. Jasim, and N. Ghani, "Service function chain mapping in fog networks," *IEEE Communications Letters*, vol. 25, no. 1, pp. 99–102, Jan. 2021.

[30] M. A. Jasim, N. Siasi, and N. Ghani, "Hierarchy descending sfc provisioning scheme with load balancing in fog computing," *IEEE Communications Letters*, vol. 26, no. 9, pp. 2096–2100, Sep. 2022.

[31] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "Deep reinforcement learning based qos-aware routing in knowledge-defined networking," in *Qshine 2018 - 14th EAI International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, Ho Chi Minh City, Vietnam, Dec. 2018, pp. 1–13.

[32] R. Mijumbi, J. Gorricho, J. Serrat, M. Claeys, F. De Turck, and S. Latré, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, Krakow, Poland, May 2014, pp. 1–9.

[33] M. Bunyakitanon, X. Vasilakos, R. Nejabati, and D. Simeonidou, "End-to-end performance-based autonomous vnf placement with adopted reinforcement

learning," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 2, pp. 534–547, Jun. 2020.

[34] A. Rkhami, Y. Hadjadj-Aoul, and A. Outtagarts, "Learn to improve: A novel deep reinforcement learning approach for beyond 5g network slicing," in *IEEE Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, USA, Jan. 2021, pp. 1–9.

[35] M. Zhu, Q. Chen, J. Gu, and P. Gu, "Deep reinforcement learning for provisioning virtualized network function in inter-datacenter elastic optical networks," in ieee transactions on network and service management," *IEEE Transactions on Network and Service Management*, May 2022, early access.

[36] X. Chen, B. Li, R. Proietti, Z. Z. H. Lu, and S. J. B. Yoo, "Deeprmsa: A deep reinforcement learning framework for routing, modulation and spectrum assignment in elastic optical networks," *Journal of Lightwave Technology*, vol. 37, no. 16, pp. 4155–4163, Aug. 2019.

[37] T. Tang, B. Wu, and G. Hu, "A hybrid learning framework for service function chaining across geo-distributed data centersa hybrid learning framework for service function chaining across geo-distributed data centers," *IEEE Access*, vol. 8, pp. 170 225–170 236, Sep. 2020.

[38] B. Li and Z. Zhu, "Gnn-based hierarchical deep reinforcement learning for nfv-oriented online resource orchestration in elastic optical dcis," *Journal of Lightwave Technology*, vol. 40, no. 4, pp. 935–946, Feb. 2022.

[39] Y. Xie *et al.*, "Virtualized network function forwarding graph placing in sdn and nfv-enabled iot networks: A graph neural network assisted deep reinforcement learning method," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 524–537, Mar. 2022.

[40] B. V. J. Santos, T. Wauters and F. D. Turck, "Resource provisioning in fog computing through deep reinforcement learning," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Bordeaux, France, Jun. 2021, pp. 431–437.

[41] M. Goudarzi, M. Palaniswami, and R. Buyya, "A distributed deep reinforcement learning technique for application placement in edge and fog computing environments," Oct. 2021. [Online]. Available: https://arxiv.org/abs/2110.12415

[42] H. Sami, A. Mourad, H. Otrok, and J. Bentahar, "Demand-driven deep reinforcement learning for scalable fog and service placement," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2671–2684, Sep.-Oct. 2022.

[43] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," Sep. 2015. [Online]. Available: https://arxiv.org/abs/1509.06461

[44] J. SON, T. He, and R. Buyya, "CloudSimSDN-NFV: Modeling and simulation of network function virtualization and service function chaining in edge computing environments," *Software: Practice and Experience*, vol. 49, no. 12, pp. 1748–1764, Dec. 2019.

[45] F. Zhuang *et al.*, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, Jan. 2021.

[46] CISCO, "Enterprise data center infrastructure," 2023. [Online]. Available: https://www.ccexpert.us/network-design/enterprise-data-center-infrastructure.html

[47] T. Tovinger, "Management, orchestration and charging for 5g networks," Mar. 2018. [Online]. Available: https://www.3gpp.org/news-events/1951-sa5_5g

[48] 3GPP, "3gpp tr 28.801 v15.1.0," Oct. 2018. [Online]. Available: https://www.3gpp.org/ftp/TSG_SA/WG5_TM/TSGS5_116/SA_78/28801-f10.doc

[49] P. Goransson, C. Black, and T. Culver, *Software Defined Networks:A Comprehensive Approach*. Elsevier, 2016, ch. 7, p. 159.

[50] W. Wang, B. Liang, and B. Li, "Multi-resource generalized processor sharing for packet processing," in *2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*, Montreal, QC, Canada, Jun. 2013, pp. 1–10.

[51] S. Gu, Z. Li, C. Wu, and C. Huang, "An efficient auction mechanism for service chains in the nfv market," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, San Francisco, CA, USA, Apr. 2016, pp. 1–9.

[52] S. Larsen, P. Sarangam, and R. Huggahalli, "Architectural breakdown of end-to-end latency in a tcp/ip network," in *19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'07)*, Gramado, Brazil, Oct. 2007, pp. 195–202.

[53] A. Greenberg, J. R. Hamilton, N. Jain *et al.*, "Vl2: a scalable and flexible data center network," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 51–62, Aug. 2009.

[54] J. Son, A. V. Dastjerdi, R. N. Calheiros, Y. Y. X. Ji, and R. Buyya, "Cloudsimsdn: Modeling and simulation of software-defined cloud data centers," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Shenzhen, China, May 2015, pp. 475–484.

[55] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource

provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[56] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," Jan. 2017. [Online]. Available: https://arxiv.org/abs/1412.6980v9

[57] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15, Fort Lauderdale, FL, USA, Apr. 2011, pp. 315–323.

[58] A. L. Maas, "Rectifier nonlinearities improve neural network acoustic models," in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[59] N. Zhang, S. Zhang, P. Yang, O. Alhussein, W. Zhuang, and X. S. Shen, "Software defined space-air-ground integrated vehicular networks: Challenges and solutions," *IEEE Communications Magazine*, vol. 55, no. 7, pp. 101–109, Jul. 2017.

[60] M. M. Tajiki, S. Salsano, L. Chiaraviglio, M. Shojafar, and B. Akbari, "Joint energy efficient and qos-aware path allocation and vnf placement for service function chaining," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 374–388, Mar. 2019.

[61] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," Aug. 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[62] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative, multi-agent games," Jul. 2021. [Online]. Available: https://arxiv.org/abs/2103.01955

[63] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," Feb. 2017. [Online]. Available: https://arxiv.org/abs/1609.02907

[64] P. Pan, Q. Fan, S. Wang, X. Li, J. Li, and W. Shi, "Gcn-td: A learning-based approach for service function chain deployment on the fly," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, Taipei, Taiwan, Dec. 2020, pp. 1–6.

[65] A. Rkhami, T. A. Q. Pham, Y. Hadjadj-Aoul, A. Outtagarts, and G. Rubino, "On the use of graph neural networks for virtual network embedding," in *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, Montreal, QC, Canada, Oct. 2020, pp. 1–6.

[66] S. Qi, S. Li, S. Lin, M. Y. Saidi, and K. Chen, "Energy-efficient vnf deployment for graph-structured sfc based on graph neural network and constrained deep reinforcement learning," in *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Tainan, Taiwan, Sep. 2021, pp. 348–353.

[67] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "Simgnn: A neural network approach to fast graph similarity computation," Mar. 2020. [Online]. Available: https://arxiv.org/abs/1808.05689

[68] J. Achiam, "Openai spinning up: Vanilla policy gradient," 2018. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/vpg.html

[69] Virginia, "How 400g has transformed data centers," Jun. 2022. [Online]. Available: https://community.fs.com/blog/how-400g-has-transformed-data-centers.html

[70] J. Bai, H. Yuan, L. Zhang, and J. Zhang, "Sgw-scn: An integrated machine

learning approach for workload forecasting in geo-distributed cloud data centers," *Information Sciences*, vol. 481, pp. 57–68, May 2019.

[71] C. Hu, Y. Guo, Y. Deng, and L. Lang, "Improve the energy efficiency of datacenters with the awareness of workload variability," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1260–1273, Jun. 2022.

[72] N. Yang, Y. Ma, L. Suo, Y. Lu, S. Ren, and L. Lin, "Proximal virtual network embedding based on multi-dimensional load balancing in data centers," in *2021 IEEE/CIC International Conference on Communications in China (ICCC)*, Xiamen, China, Jul. 2021, pp. 945–950.

[73] V. M. et al., "Asynchronous methods for deep reinforcement learning," Jun. 2016. [Online]. Available: https://arxiv.org/abs/1602.01783

[74] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004, ch. 4, pp. 129–138.