Algorithmic Solutions for Virtual Network Function Migration in Cloud

Seyedeh Negar Afrasiabi

A Thesis

in

The Department

of

Information and Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Information and Systems Engineering) at

Concordia University

Montréal, Québec, Canada

December 2023

© Seyedeh Negar Afrasiabi, 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Seyedeh Negar Afrasiabi**

Entitled: **Algorithmic Solutions for Virtual Network Function Migration in Cloud**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Information and Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

______________________________ Chair
Dr. Rodolfo Coutinho

______________________________ External Examiner
Dr. Soumaya Cherkaoui

______________________________ External to Program
Dr. Ferhat Khendek

______________________________ Examiner
Dr. Chadi Assi

______________________________ Examiner
Dr. Jamal Bentahar

______________________________ Thesis Supervisor
Dr. Roch Glitho

Approved by ______________________________
Dr. Jun Yan, Graduate Program Director

Date of Defence: December 20, 2023 ______________________________
Dr. Mourad Debbabi, Dean, Gina Cody School of Engineering and Computer Science

Abstract

Algorithmic Solutions for Virtual Network Function Migration in Cloud

Seyedeh Negar Afrasiabi, Ph.D.

Concordia University, 2023

Network Function Virtualization (NFV) is a network architecture that separates network functions from dedicated hardware, implementing them as software modules known as Virtual Network Functions (VNFs), which are executed in virtual machines or containers. A Network Service (NS) consists of a chain of VNFs known as a VNF Forwarding Graph (VNF-FG). NFV increases deployment flexibility and agility within operator networks and reduces operating and capital expenditures significantly. Deploying an NS requires solving the NFV resource allocation (NFV-RA) problem, which involves the three stages of (i) VNF-FG composition, (ii) VNF-FG embedding, and (iii) VNF scheduling. Resource allocation in NFV requires efficient algorithms to determine on which physical node VNFs are embedded and to be able to migrate VNFs from one node to another. A major challenge in NFV is how to maintain reasonable VNF embedding to adapt to the changes in the network. As the VNF embedding stage may also be dynamic; it brings an additional dimension of complexity in terms of keeping track of where a given VNF is running. In other words, the VNF migration is responsible for where, when, and how to transfer the VNFs from source to destination in response to the variation in service requests. The VNF Migration problem generally refers to the process of migrating VNFs from one node to another due to specific requirements such as reduction of cost, energy saving, recovery from failures, etc. However, VNF migration faces several challenges. The first challenge arises from the mobility of end-users and the fog nodes, along with limited fog node coverage, resulting in service discontinuity and increasing application delay. A second challenge presents when there are stringent latency requirements between VNFs and can make them tightly coupled, thus hindering each VNF from being migrated individually, and resulting in poor performance. The third challenge is when we have a limitation of resources

in the network. The overloaded node can significantly impact the determination of the best VNF decomposition option among all possible choices, potentially leading to a degradation in Quality of Service (QoS). VNF migration can offer great potential to address these challenges. However, the challenge remains: where, when, and which VNF should migrate to improve performance.

In this Ph.D. thesis, we aim to address the challenges in the VNF migration problem mentioned above. Firstly, we introduce a reinforcement learning-based optimization framework for application component migration in NFV cloud-fog environments where both fog nodes and end-users are mobile. More specifically, our main objective is to efficiently migrate the VNFs of a request such that the total delay and cost are minimized. Secondly, we introduce a cost-efficient solution for solving the problem of cluster migration of VNFs for VNF-FG embedding by taking into account the latency requirement between VNFs and reusing the already deployed VNFs. The objective is to migrate the cluster of VNF so that the total embedding cost, including resource, instantiation, reuse, and transmission cost, is minimized. Lastly, when considering VNF migration in the case of VNF decomposition, we investigate how VNF migration and VNF decomposition can be mutually beneficial. We achieve this by designing a joint VNF decomposition and migration approach to minimize the embedding cost of network services (NS) and promote VNF reusability. To accomplish this, we propose two efficient heuristics for identifying the best decomposition options and facilitating the migration of previously deployed VNFs across the network.

Acknowledgments

A dissertation is not the sole outcome of an individual's efforts. Many people have contributed to its development, and I would like to take this opportunity to acknowledge those who have made a significant impact on my doctoral journey and accomplishment.

First and foremost, I extend my heartfelt gratitude to my supervisor, Dr. Roch Glitho, for his continuous support throughout my Ph.D., his patience, kindness, and for providing me with the invaluable opportunity to work in his lab. Each day spent in his lab has been a rich learning experience, not only in academia but also in fostering positive behavior. The diverse interactions with individuals from various backgrounds have broadened my perspective, and I am grateful for the wealth of knowledge gained from these encounters. I would like to express my gratitude to my committee members, Dr. Chadi Assi, Dr. Jamal Bentahar, and Dr. Ferhat Khendek, for their time, valuable feedback, and constructive comments. I also extend my appreciation to Dr. Soumay Cherkaoui for accepting to serve as my external examiner.

I am also thankful for the collaboration with Ericsson researchers Dr. Carla Mouradian, Dr. Wubin Li, and Dr. Róbert Szabó, with whom I had the opportunity to collaborate on my Ph.D. projects. It was a great pleasure working with them for all the enlightening discussions, comments, and collaborations on the projects that I completed throughout my Ph.D.

I dedicate this thesis to my beloved parents. I am deeply indebted to my mother, Sohila, for her endless love, care, sacrifice, and guidance. Nothing would have been possible without her support at all times. My gratitude extends to my father, Hossein, for always pushing me to advance in my education and career. To my sisters, Niloofar and Nazanin, who are my best friends, I express my heartfelt thanks. Thank you both for always cheering me up. A special acknowledgment goes to

Nazanin for being by my side since the beginning of this journey, providing unwavering support and helping me navigate through the weakest moments. Your presence and assistance have been invaluable in overcoming obstacles, and I am deeply grateful to have you as my sister. Words cannot express how grateful I am to my family for having them in my life.

Contents

List of Figures **xi**

List of Tables **xiv**

1 Introduction **1**

1.1 Overview 1

1.2 Challenges 2

1.3 Thesis Contributions 4

1.3.1 Reinforcement Learning-based Optimization Framework for Application Component Migration in NFV Cloud-Fog Environments [1, 2] 4

1.3.2 Cost-efficient Cluster Migration of VNFs for VNF Forwarding Graph Embedding [3] 5

1.3.3 Joint VNF Migration and Decomposition for Cost-efficient VNF Forwarding Graph Embedding [4, 5] 6

1.4 Background Information 6

1.4.1 Cloud, Edge and Fog Computing 6

1.4.2 NFV Resource Allocation 8

1.4.3 Reinforcement learning 12

1.5 Thesis Outline 13

2 Critical Review of the State of the Art **14**

2.1 Application Component Migration in Edge Computing 14

2.2 VNF Migration . . . 16
2.2.1 Simple VNF Migration . . . 16
2.2.2 Cluster VNF Migration . . . 17
2.3 Joint VNF Decomposition and Migration . . . 18
2.3.1 VNF Decomposition . . . 18
2.3.2 Joint Methods . . . 19
2.4 Conclusion . . . 21

3 Reinforcement Learning-based Optimization Framework for Application Component Migration in NFV Cloud-Fog Environments 22
3.1 Introduction . . . 22
3.2 Motivating Scenario . . . 23
3.3 System Model and Problem Formulation . . . 25
3.3.1 System Model . . . 26
3.3.2 Problem Formulation . . . 28
3.3.3 Problem Analysis . . . 36
3.4 RL-Based VNF Migration . . . 37
3.4.1 MDP Framework . . . 37
3.4.2 Design of the Deep RL Agent . . . 38
3.4.3 DDQN with LSTM Cells . . . 41
3.4.4 Complexity Analysis . . . 43
3.5 Results and Discussions . . . 44
3.5.1 Simulation Settings . . . 44
3.5.2 Convergence Performance . . . 46
3.5.3 Simulation Results . . . 50
3.6 Conclusions . . . 54

4 Cost-efficient Cluster Migration of VNFs for VNF forwarding graph embedding 55
4.1 Introduction . . . 55
4.2 Motivating Scenario . . . 56

4.3 System Model and Problem Formulation 60
4.3.1 System Model 60
4.3.2 Problem Formulation 62
4.4 Proposed Single-/Multi-Destination Cluster VNF Migration Algorithm 66
4.4.1 Single-Destination Cluster Migration 66
4.4.2 Multi-Destination Cluster Migration 69
4.4.3 Complexity Analysis 71
4.5 Results and Discussions 71
4.5.1 Simulation Setup 72
4.5.2 Results 74
4.6 Conclusions 79

5 Joint VNF Decomposition and Migration for Cost-efficient VNF Forwarding Graph Embedding 81
5.1 Introduction 81
5.2 Motivating Scenario 82
5.3 System Model and Problem Formulation 85
5.3.1 System Model 85
5.3.2 Problem Formulation 87
5.4 Proposed Solution 91
5.4.1 Cost Aware VNF Decomposition (CA-VNF-D) 91
5.4.2 Node Aware VNF Migration (NA-VNF-M) 96
5.4.3 Asymptotic Analysis 99
5.5 Results and Discussions 100
5.5.1 Simulation setup 101
5.5.2 Results 102
5.6 Conclusions 108

6 Conclusions and Future Works 110
6.1 Conclusions 110

6.2 Future Works . . . 111
6.2.1 Application Component Migration . . . 111
6.2.2 VNF Cluster Migration . . . 112
6.2.3 Joint VNF Decomposition and Migration . . . 112

List of Figures

Figure 1.1 Illustrative example of a VNF-FG with two different realizations of VNF B: (a) chain of required VNFs, (b) VNF-FG with decomposition option 1 for VNF B, and (c) VNF-FG with decomposition option 2 for VNF B. 9

Figure 1.2 VNF Migration by considering the mobility of fog nodes.: (a) Before Migration, (b) After Migration. 10

Figure 3.1 Example of earthquake early warning and recovery application [6]. 25

Figure 3.2 Migration of an application component for an earthquake early warning and recovery application :(a) Before Moving, (b) After Moving 26

Figure 3.3 Agent-environment interaction with DDQN strategy. 41

Figure 3.4 Network topology. 46

Figure 3.5 Convergence performance of our proposed DDQN-CM algorithm for different values of learning rate α against the traditional DDQN algorithm ($\alpha = 0.001$) [7]. 48

Figure 3.6 Objective function vs. episode for different values of fog node speed v_{max}. . 48

Figure 3.7 (a) Delay, (b) cost, and (c) objective function vs. weight ω for different algorithms. 49

Figure 3.8 Average cost vs. average delay for different values of weight ω. 50

Figure 3.9 Objective function of different algorithms when varying the number end-users communicating with the VNFs. 52

Figure 3.10 Average number of migrations per time slot with respect to the number of end-users. 53

Figure 3.11 Power consumption when varying the number of VNFs. 54

Figure 4.1 Illustrations of: (a) embedding solution for NS 1 obtained by the SG algorithm [8], (b) simple VNF migration for NS 1 [9], (c) cluster VNF migration for NS 1, (d) solution for NS 2 obtained by the SG algorithm, (e) cluster VNF migration for NS 2, (f) complex VNF migration. 59

Figure 4.2 Expanding the Cluster of VNFs for $threshold = 4$. 69

Figure 4.3 Total embedding cost vs. threshold T_0 for our proposed (a) sDCM and (b) mDCM algorithms for different values of $r_0 \in \{1, 2, 4, 10\}$ (17 nodes, 30 request, and [5-8] VNFs per request). 72

Figure 4.4 Average cluster size per request vs. threshold T_0 (with 17 nodes, 30 requests, $r_0 = 10$ and [5-8] VNFs per request. 74

Figure 4.5 Total embedding cost vs. number of requests for (a) $r_0 = 1$, (b) $r_0 = 2$, (c) $r_0 = 4$, and (d) $r_0 = 10$ (with 17 nodes, $r_0 = 10$, [5-8] VNFs per request). . . . 75

Figure 4.6 Total embedding cost vs. number of nodes ([5-12] VNFs per request, and $r_0 = 10$). 76

Figure 5.1 (a) Incoming NS 2 with possible decomposition options, (b) tree graph showing all possible VNF-FGs for NS 2, (c) one possible realization of VNF-FG for NS 2, (d) substrate network topology with NS 1 being embedded, (e) embedding of best decomposition option of NS 2 without considering migration, (f) migration of VNF J to Node 2, and (g) embedding of the best decomposition option of NS 2 after migration. 83

Figure 5.2 Illustrative examples for each step of Cost-Aware VNF Decomposition (CA-VNF-D) algorithm. 95

Figure 5.3 Illustrative examples for Node Aware VNF Migration (NA-VNF-M) Algorithm 99

Figure 5.4 An NS with 3 main VNFs and all decomposition options. 102

Figure 5.5 Ratio of embedding cost of Decomposition-only to embedding cost of the proposed method when (Fixed capacity, Variable capacity) = (10,1) for 500 requests for different nodes' capacity. 103

Figure 5.6 Embedding cost ratio vs. number of requests for (fixed capacity, variable capacity) = (10,1) for different values of VNF-FG size. 105

Figure 5.7 Examining the adaptiveness behavior in scenarios where decomposition options do not have equal CPU demand (14 requests). 106
Figure 5.8 Embedding Cost ratio vs. capacity demand for 10 requests (all options have an equal CPU demand). 107
Figure 5.9 Breakdown of embedding cost for 10 requests and 5 VNFs per request. . . . 108

List of Tables

Table 2.1 Evaluation of the existing solutions for application component migration in edge computing. 15
Table 2.2 Evaluation of the existing solutions for VNF migration, including simple and cluster migration approaches. 18
Table 2.3 Evaluation of the related works for joint VNF decomposition and migration. . 20
Table 3.1 Summary of main notations. 27
Table 3.2 Hyperparameter settings. 44
Table 3.3 Parameter settings and default values. 47
Table 3.4 Convergence episode and execution time of different algorithms. 47
Table 4.1 Summary of main notations. 60
Table 4.2 Parameter settings and default values 72
Table 4.3 Execution Time (seconds). 79
Table 5.1 Summary of main notations. 85
Table 5.2 Parameter settings and default values. 102

Chapter 1

Introduction

1.1 Overview

Network Function Virtualization (NFV) is an emerging technology initiated by the European Telecommunication Standards Institute (ETSI) [10]. NFV provides the possibility to decouple network functions from dedicated hardware and runs these functions as software instances on commodity servers through virtualization. In NFV, hardware-based network functions are replaced with software-based Virtual Network Functions (VNFs), which are executed in Virtual Machines (VMs) or container [11]. NFV increases the flexibility and agility of network service deployment. In addition, software-based VNFs significantly decrease the Capital Expenditure (CAPEX) and Operating Expense (OPEX) of the Service Provider (SP) [10]. VNF Forwarding Graph (VNF-FG) is a graph of interconnected VNFs that are linked in order to instantiate a Network Service (NS). Deploying an NS requires solving the NFV resource allocation (NFV-RA) problem, [11]. In general, the NFV-RA problem can be divided into three main stages: (i) VNF composition, (ii) VNF-FG embedding/placement, and (iii) VNF scheduling. The first stage refers to concatenating the different VNFs efficiently to compose an NS with respect to SP's objectives, whereas the second stage (VNF-FG embedding) refers to the mapping of virtual resources (i.e., VNFs) to physical resources (i.e., substrate nodes/links. The third stage focuses on determining the schedule for processing the VNFs on each substrate node.

With the development of cloud computing [12], Cloud Infrastructure Providers (CIPs) offer on-demand computing (e.g., in the form of VMs, and containers) with a pay-as-you-go pricing model. However, the fundamental limitation of cloud computing is the physical distance between a cloud service provider's data centers and end devices. This distance could cause end-to-end delays which may not be acceptable for latency-sensitive applications. Fog computing is a computing paradigm introduced to tackle the cloud latency-related challenge. Indeed, it extends the traditional cloud computing architecture to the edge of the network, enabling computing at the edge of the network, closer to the end-user devices. Extending cloud computing to the edge of the network results in a hybrid cloud/fog system. Fog computing enables the deployment of some VNFs at the edge of the network, on fog nodes, while keeping others in the cloud [13]. The SP can customize the location of VMs that host VNFs to reduce operational costs and latency. A VNF-FG can map on a cloud/fog provider infrastructure, also known as NFV Infrastructure (NFVI) [14].

While virtualization offers flexibility, it is subject to a number of challenges. Efficient algorithms are essential in NFV for determining the allocation of resources, deciding on which physical node to embed VNFs, and facilitating the migration of VNFs between nodes when necessary. A major challenge in NFV is how to maintain VNF embedding to adapt to changes in network configurations. For example, when network topology changes (e.g., network device failure, congestion on links or nodes, end-user mobility, etc.), it is necessary to dynamically adjust the network configuration to meet the predefined Quality-of-Service (QoS). When a network configuration changes, the administrator can relocate VNFs from one physical node to another to achieve better performance and adapt to the network changes via the so-called VNF migration.

This thesis focuses on algorithmic solutions in VNF migration. In the following subsections, we first discuss the challenges of VNF migration followed by our main thesis contributions. Next, we provide background information about important concepts related to our thesis. Finally, we present our thesis outline.

1.2 Challenges

The main challenges tackled in this thesis on VNF migration are summarized as follows:

- **Application Component Migration in NFV Cloud-Fog Environments:** In NFV settings, application components can be implemented as VNFs. The mobility of end-users (e.g., mobile devices) and fog nodes and limited fog node coverage may result in service discontinuity and increased application delay. More precisely, as a result of a fog node's mobility, the hosted component may become farther from end-users which results in high latency. In this case, migrating the component to a closer fog node helps in reducing the end-to-end latency. Furthermore, dynamic consolidation of NFVIs in as few nodes as possible is key to allowing several nodes to be switched off, thus helping minimize the overall cost [15]. Migration in NFV-based hybrid cloud/fog systems, where both end-users and fog nodes are mobile, and application components interact with each other is a complex problem, and it is particularly challenging to make an optimal migration decision. Therefore, there remains a need for efficient algorithms to determine when and where application components migrate and enable rapid decision-making.

- **Cluster Migration of VNFs:** The simplest form of migration aims to relocate VNFs individually. However, stringent latency requirements between VNFs of a given VNF-FG can make those VNFs coupled to each other. This may limit the simple VNF migration strategy since this strategy will only relocate the VNFs that are not coupled to their neighbor VNFs with stringent latency constraints. Moreover, given that the simple VNF migration aims to move a single VNF at a time, an additional transmission cost may be incurred, which can increase the resulting embedding cost. Thus, the stringent latency requirements between VNFs can make them tightly coupled, thus hindering each VNF from being migrated individually and resulting in poor performance. Besides simple VNF migration, another type of VNF migration can be realized by clustering a group of coupled VNFs and migrating them within a single physical node or across multiple physical nodes. We refer to this type of migration as cluster VNF migration, which can potentially reduce the embedding cost. Therefore, efficient algorithms are needed to identify a cluster of VNFs for migration and to manage the migration of this cluster to either a single physical node or multiple physical nodes.

- **Joint VNF Migration and Decomposition:** When a node's resources are all occupied by

running workloads, it becomes over-loaded, potentially causing bottlenecks that prevent the SP from admitting new requests. To address resource shortages and prevent the blockage of incoming NSs, VNF migration can be a viable solution. The emergence of VNF decomposition as a new functional architecture enables the VNFs to be decomposed into smaller sub-functions, thus offering flexibility, resource sharing, and scalability. VNF decomposition can lead to a notable reduction of VNF embedding cost, as different sub-functions can be flexibly reused by multiple network requests. However, the amount of available resources within substrate nodes can have a significant impact on determining the best decomposition option. On the other hand, selecting the proper decomposition options may require some VNFs to be moved from an over-loaded node to another. Thus, a major challenge lies in determining how and which of the already deployed VNFs can be migrated to other nodes, aiming to select the optimal decomposition option that minimizes embedding costs and promotes VNF reusability. Therefore, efficient algorithms are required to jointly select the best VNF decomposition option and migrate VNFs through the network.

Each of these challenges contributes to the costs and service quality of the final NS and thus should be addressed carefully. Therefore, it is an important and challenging problem to decide where, when, and how to migrate VNFs to achieve better performance in terms of embedding cost, power consumption, and delay, among others.

1.3 Thesis Contributions

The existing solutions do not fully address all these challenges. This Ph.D. thesis proposes algorithmic solutions to tackle the challenges related to VNF migration. It presents three primary contributions as follows, with each corresponding to a challenge addressed by this thesis.

1.3.1 Reinforcement Learning-based Optimization Framework for Application Component Migration in NFV Cloud-Fog Environments [1, 2]

The first contribution is focused on application components migration in NFV Cloud-Fog Environments. As we discussed earlier, application components can be implemented as VNFs. Some

application components can be hosted by the fog while others may reside in the cloud. The mobility of end-users and the fog nodes, and the limited fog nodes coverage result in service discontinuity and may increase application delay. In this contribution, we propose a component migration strategy in an NFV-based hybrid cloud/fog system considering the mobility of both end-users and fog nodes. We mathematically modeled the problem to minimize both the application delay and cost. As the problem of migrating application components is inherently complex, making optimal migration decisions is particularly challenging. Much of this complexity arises from the necessity to manage multiple uncertainties. For example, even when we have information about the current positions of fog nodes and end-users, their future locations remain unknown, complicating the evaluation of the cost/delay trade-off. Hence, we propose a Deep Reinforcement Learning (DRL) approach to decide where and when to migrate application components and to achieve rapid decision-making. The simulation results demonstrate that the proposed scheme offers favorable convergence and outperforms existing algorithms in terms of application delay and migration costs.

1.3.2 Cost-efficient Cluster Migration of VNFs for VNF Forwarding Graph Embedding [3]

In the second contribution, we aim to solve the problem of cluster VNF migration by considering the given inter-VNF latency requirements. As we discussed earlier, stringent latency requirements between VNFs can make them tightly coupled, thus hindering each VNF from being migrated individually, and resulting in poor performance. Furthermore, a physical node may not support all types of VNFs, and it has limited resources. Thus, migrating a cluster of VNFs to a single physical node would not always be a reasonable option. Therefore, in this contribution, we are aiming to identify clusters of VNFs for migration and migrate a cluster of VNFs to single and multiple physical nodes. We formulate the VNF migration problem as an Integer Linear Programming (ILP), aiming to minimize the total embedding cost while satisfying the given latency between the VNFs. In our developed formulation, we take into account computing resource cost, new VNF instantiations, VNF reusability, traffic routing, and latency requirements between VNFs. We proposed two scalable and efficient algorithms for migrating a cluster of VNFs. Through extensive experiments, we show that our proposed algorithms achieve lower total embedding cost compared to the existing

algorithm while being much more scalable than the brute-force approach.

1.3.3 Joint VNF Migration and Decomposition for Cost-efficient VNF Forwarding Graph Embedding [4, 5]

In the third contribution, we consider VNF migration by assuming each VNF can be decomposed into multiple options. In this contribution, we study the joint problem of VNF decomposition and migration. As previously discussed, overloaded nodes play a crucial role in determining the best decomposition option, and migrating VNFs from overloaded nodes to other nodes can significantly influence this decision. We specifically explore how VNF migration and decomposition can mutually benefit each other, leading to reduced embedding costs and increased VNF reusability. After formulating the joint problem of VNF decomposition and migration as an ILP to minimize the embedding cost, we propose two efficient heuristics to identify the best decomposition options while facilitating the migration of previously deployed VNFs across the substrate network. The simulation results indicate that our proposed algorithms outperform the decomposition-only approach in terms of embedding cost and number of new instances.

1.4 Background Information

In this subsection, the background information on important concepts used in this research work is presented. First, some background information background information on cloud, edge, and fog computing will be discussed. Then, the NFV Resource Allocation (NFV-RA) problem is provided. Finally, the general principles of reinforcement learning will be discussed.

1.4.1 Cloud, Edge and Fog Computing

Computing paradigms have evolved from distributed, parallel, and grid to cloud, edge, and fog computing [16]. This subsection discusses cloud, edge, and fog computing paradigms.

- Cloud computing has several inherent capabilities: scalability, on-demand resource allocation, reduced management efforts, pay-as-you-go, and easy applications and services provisioning. It comprises three key service models: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). IaaS provides virtualized resources, such as computing, storage, and networking [17]. The PaaS provides software environments for applications' development, deployment, and management. The SaaS provides software applications and composite services to end-users and other applications. Cloud computing is unsuitable for latency-sensitive applications since the connectivity between cloud and edge is associated with a long delay [16].

- Multi-access Edge Computing (MEC) was initiated in late 2014 by the ETSI Mobile Edge Computing Industry Specification Group (MEC ISG) [18]. MEC focuses on mobile networks and virtualization technology. The ETSI initiative aims to standardize the APIs between mobile users' platforms and their applications to foster innovations in an open environment. MEC only functions in standalone mode, and it is independent of the cloud. MEC aims to unite telecommunication and IT cloud services to provide cloud-computing capabilities within radio access networks close to mobile users.

- Fog computing is a concept introduced by Cisco in 2012 [16]. Fog computing is a highly virtualized platform that offers cloud and end-user computing, storage, and networking services. It is a novel architecture that extends the traditional cloud computing architecture to the edge of the network. Fog computing enables the processing of latency-sensitive application components at the network's edge, and delay-tolerant and computational-intensive components can be done in the cloud. In addition, fog computing supports low latency applications by allowing processing to occur at the network edge, near the end devices, by the so-called fog nodes, and the ability to enable processing at specific locations [16]. Unlike MEC, connectivity with the cloud is a key feature of fog computing. In addition, fog computing targets all applications, while MEC targets specific sub-sets of applications. For instance, MEC targets mobile offloading applications and those applications which are better provisioned at the edge [16].

1.4.2 NFV Resource Allocation

The provisioning of Network Services (NSs) in an NFV-based infrastructure faces a significant challenge in the form of NFV-RA, which has a substantial impact on both the SLA requirements of the NSs and the profits of Network Operators (NOs). In NFV environments, NSs are typically provisioned by directing their traffic through a predefined sequence of VNFs located between specific endpoints. NFV-RA can be generally categorized into three key stages [11, 10] : (i) VNF composition, (ii) VNF embedding/placement, and (iii) VNF scheduling. We will provide a brief description of each of these stages below.

1.4.2.1 VNF Composition

VNF composition involves the creation of VNF-FGs in response to NS requests. Typically, an NS request comprises a collection of VNFs with associated dependency constraints and SLA requirements, such as resource demands, traffic requirements, and latency constraints [19].The primary goal of VNF composition is to construct chains of VNFs to build VNF-FG. This process involves determining which VNFs should be instantiated, the quantity of VNF instances for each type, and the connections between these instances. This process is guided by Network Operator (NO) objectives and SLA requirements.

Moreover, each VNF may be associated with multiple realizations, commonly referred to as decomposition options [20, 21, 22]. For a better understanding, let us consider an NS request that an ordered set of VNFs comprises VNFs A, B, and C, as shown Fig. 1.1.a. We also assume that VNF B can be decomposed into multiple sub-functions and has two different decomposition options. Fig. 1.1.b and Fig. 1.1.c show possible VNF-FGs of two decomposition options of VNF B. If sub-functions B2 and BX are already deployed and have enough capacities to serve the NS, the VNF-FG in Fig. 1.1.b is preferable, as the sub-functions B2 and BX can be reused to maximize profits of SPs. Each of these decomposition options can highly influence SP's objectives (e.g., embedding costs, user experiences) as well as the NS performance. When considering function decomposition, a VNF can be decomposed into more fine-grained sub-functions. Furthermore, the sub-functions of the same type placed on the same VM can be re-used to save the node resource consumption.

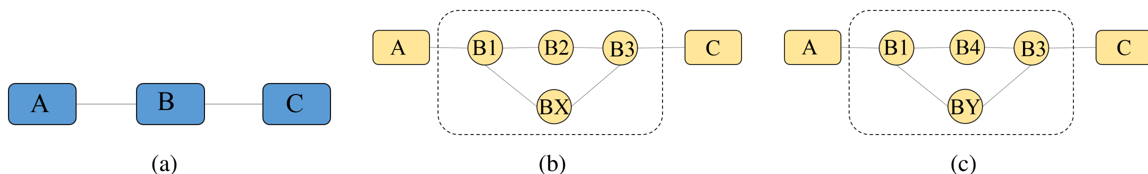


Figure 1.1: Illustrative example of a VNF-FG with two different realizations of VNF B: (a) chain of required VNFs, (b) VNF-FG with decomposition option 1 for VNF B, and (c) VNF-FG with decomposition option 2 for VNF B.

Therefore, how to efficiently select the decomposition option for each VNF of various NSs to meet SLA requirements as well as meet SP's objectives becomes challenging [20]. This raises a new problem, and we refer to it as VNF decomposition. It is worth mentioning that there is a subtle difference between the terms composition and decomposition. The VNF-FG composition generally aims at composing a suitable VNF-FG for an NS, whereas decomposition is used in micro-service architecture to decompose VNFs into various sub-functions (micro-services) and select the best decomposition options for VNF-FG composition.

1.4.2.2 VNF Embedding / Placement

The second phase of NFV-RA, known as VNF embedding or VNF placement, follows the VNF composition stage. Once a VNF-FG is derived from the VNF composition, VNF embedding's objective is to allocate network resources for VNF deployment and traffic routing. This allows for the integration of the VNF-FG into an NFV-based infrastructure. In VNF embedding, [23] each VNF within a VNF-FG is mapped to an NFV-enabled node. This implies that the resources of an NFV-enabled node, such as CPU, memory, and storage, need to be allocated to deploy the VNF. Additionally, a VNF can also be embedded into an existing VNF instance within the network, provided that the existing instance has the capability to support the VNF. In addition, each virtual link connecting the VNFs within the VNF-FG is embedded into physical links. This entails reserving resources, specifically link bandwidth, to facilitate the routing of traffic over these virtual links.

The VNF embedding stage may also be dynamic [24]; it brings an additional dimension of complexity in terms of keeping track of where a given VNF is running. In other words, the orchestrator may trigger the migration of a VNF from one node to another if necessary, in order to

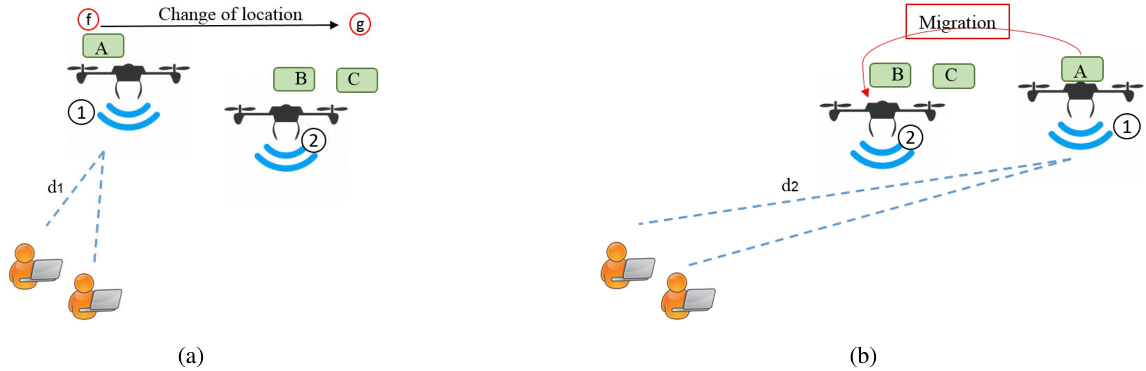


Figure 1.2: VNF Migration by considering the mobility of fog nodes.: (a) Before Migration, (b) After Migration.

optimize the use and allocation of physical resources [11]. The VNF Migration (VNF-M) problem generally refers to the process of migrating VNFs from one place to another due to specific requirements such as load balance, hardware maintenance, and mobility. During the migration process, the VNF-related state (e.g., CPU, memory, and storage) must be migrated to the destination [24]. One important issue in VNF-M is to decide which destination node should perform the computation for a particular VNF, taking into account user mobility and other dynamic changes in the network configuration. To provide a service to a user or a set of users, it is necessary to start VNF instances, which can run in either the cloud or fog nodes. The key question here is how to select the optimal location for running each VNF instance. Additionally, users may move across different geographical areas due to mobility [25]. Thus, another question is whether we should migrate the VNF instance from one node to another node when the user location or network condition changes over time. For every VNF, there is a cost associated with running the VNF instance in a specific node, and there is also a cost associated with migrating the VNF instance from one node to another node. Therefore, the placement and migration of VNF instances need to properly take into account these costs.

For better understanding, let us consider the NS shown in Fig. 1.1.a. Let us assume that VNFs are placed on UAV fog nodes. We assume that UAV 1, which hosts VNF A, moves from location f to location g, shown in Fig. 1.2. In Fig. 1.2a UAV 1 may move from location f to g to take care of a more urgent task at location g. As a result, UAV 1 becomes far away from the end-user devices from which it gets its desired information. Let's assume the distance between the end-user devices and UAV 1 is denoted by d_1. Also, assume that there is another UAV (i.e., UAV 2) at distance d_2

from the end-user which VNF B and C are hosted on it. Therefore, it is more efficient to migrate VNF A from UAV 1 to UAV 2 to reduce the communication latency, especially given that $d_1 > d_2$. Moreover, UAV 1 (which acts as a fog node) can be switched off, as it does not host any components. This action can reduce the power consumption cost (see Fig. 1.2b). However, migration of a VNF will consume a certain amount of time and resources, which cause different system costs and delays.

VNF migration can be categorized into the following three types [3]: (i) simple migration, (ii) cluster migration (iii) complex migration. Simple VNF migration refers to the migration of one VNF of a given VNF-FG request from one node to another. Simple migration may not always be enough to achieve a low-cost embedding outcome. More specifically, performing a simple migration may not always be feasible due to stringent inter-VNF latency constraints. In such cases, a group of VNFs can be gathered in the form of a cluster to be migrated to other nodes. Cluster migration can potentially improve the embedding cost by allowing the decision maker to handle the restricting impacts of the stringent inter-VNF latency constraints of VNF embedding. The third category of VNF migration is complex migration, which aims to migrate two VNFs/clusters from two different sets of nodes at the same time. A complex migration is realized by swapping the hosting nodes of two different VNFs/clusters, requiring both eviction and migration of VNFs at the same time. Unlike cluster migration, which considers a single NS at a time, complex migration can be realized by considering multiple VNFs of different NSs at a time. Therefore, a migration mechanism should be able to decide which, when, and where to migrate VNF. It is necessary to dynamically adjust the network configuration to meet the predefined QoS.

1.4.2.3 VNF Scheduling

In the final stage of NFV-RA, VNF scheduling takes center stage, with its primary objective being the determination of execution orders for the VNFs necessary for NSs. Building on the sets of VNF instances and physical links allocated to NSs during the VNF embedding phase, this stage operates under the assumption that VNF instances can be shared among multiple NSs.

1.4.3 Reinforcement learning

Reinforcement learning is one of the most important branches of machine learning, which has significant impacts on the development of Artificial Intelligence (AI) over the last 20 years. Reinforcement learning is a learning process in which an agent can learn its optimal policy through interaction with its environment. The agent observes its current state, then takes action, and receives its immediate reward with a new state. The immediate reward and the new state are used to adjust the agent's policy, and this process will be repeated until the agent's policy converges to the optimal policy.

Q-Learning is a model-free algorithm [26] that learn directly from interactions with the environment. It does not require the agent to know the system model parameters, e.g., the state-transition and reward models, in advance to estimate the pairs of state-action values. Specifically, the key idea is to approximate state-action pairs' values through samples obtained during the interactions with the environment. Unlike model-free, the model-based RL algorithms simulate transition using a learned model. It uses state-transition and reward models in order to estimate the optimal policy. Hence model-based approaches are generally not used in an online manner where the agent learns and recommends actions in real-time. Q-Learning is one of the most widely used RL strategies. Q-learning works by successively updating the evaluation of the long-term quality (the Q value) of actions at each state. In the Q-learning algorithm, all the states should be met, and all the actions should be experienced.

It is a simple way for an agent to learn how to act optimally. However, this learning process, even though proven to converge, takes a lot of time to reach the best policy as it has to explore and gain knowledge of an entire system. It makes it unsuitable and inapplicable to real-world problems as they deal with extremely complex and dynamic environments, and their states are large and vary rapidly over time. Recently, deep learning [26] has been introduced as a new breakthrough technique. It can address this shortcoming of reinforcement learning, namely Deep Reinforcement Learning (DRL). This approach eliminates the need to visit all the state/action pairs to compute the Q-values. DRL embraces the advantage of Deep Neural Networks (DNNs) to train the learning process, thereby improving the learning speed and the performance of reinforcement learning

algorithms. As a result, DRL has been adopted in numerous applications of Deep Reinforcement Learning in Communications and Networking such as wireless caching, data offloading, network security, connectivity preservation, and traffic routing [26].

1.5 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 presents a critical review of the state-of-the-art. Accordingly, Chapter 3 presents the reinforcement learning-based optimization framework for application component migration in NFV cloud-fog environments. In Chapter 4, we present algorithms for cluster migration of VNFs for VNF forwarding graph embedding. Chapter 5 presents the joint VNF decomposition and migration for VNF forwarding graph embedding. Finally, we conclude this thesis in Chapter 6 and provide future directions for this research work.

Chapter 2

Critical Review of the State of the Art

In this chapter, We review and evaluate the state of the art related to VNF migration. First, we review the research works on application component migration in edge computing. We then assess the works on VNF migration. Finally, we review the existing works on VNF decomposition and migration.

2.1 Application Component Migration in Edge Computing

In this section, We review the existing solutions for application migration in edge computing, where these components are not placed as VNF. This review encompasses a broad perspective of edge computing, including MEC and fog computing.

Migration of application components in hybrid cloud/fog systems was studied in [15], where the authors proposed a container migration algorithm to support mobile application tasks in fog computing. The problem was formulated as a Markov Decision Process (MDP) designed to reduce power consumption, communication delay, and migration costs, and a DRL-based approach was proposed to solve the MDP problem. Service migration for MEC has recently received much attention from academia [27, 28, 29, 30]. For example, the authors of [27] investigated the service migration problem for MEC while considering the mobility of end-users, formulating the problem as an MDP. Their proposed formulation represents a cost model that takes into account the migration and data transmission costs. VM migration in edge computing based on the multiple attribute

decision-making approach was proposed in [28], where the objective was to reduce the user response time and improve the user experience quality. The authors of [29] proposed a modified quantum particle swarm algorithm to solve the service migration problem for mobile end-users. The objective was to minimize the total delay (consisting of migration, processing, and transmission delays) as well as the energy consumption. A DRL-based approach was presented in [30] to solve the problem of service migration in an edge computing environment for a single user with the objective of minimizing migration and communication costs. In [31], the authors proposed a Multi-Agent Deep Reinforcement Learning (MADRL) algorithm for Vehicular Edge Computing (VEC) with the objective of meeting the service delay requirements with minimum migration cost and minimum travel time. We note that all the above-mentioned studies focus only on the migration of a single service or application. In real-world scenarios, however, an application is a collection of components that interact. In this thesis, application components are implemented as VNFs. Together these VNFs form a VNF-FG with sub-structures such as sequence, parallel, selection, and loop to establish the model of the execution sequence of the components. Moreover, none of the existing solutions reviewed above considers the mobility of both fog/edge nodes and end-users. The evaluation of the existing studies on application Component migration in NFV eddge computing is summarized in Table 2.1.

Table 2.1: Evaluation of the existing solutions for application component migration in edge computing.

References	Research Area	Mobility		Interaction between Application Component	Delay	Cost
		End-user	Edge Nodes			
[15]	Fog computing	✓	-	-	✓	✓
[27]	MEC	✓	-	-	-	✓
[28]	MEC	✓	-	-	✓	-
[29]	MEC	✓	-	-	✓	-
[30]	MEC	✓	-	-	-	✓
[31]	VEC	-	✓	-	✓	-

2.2 VNF Migration

2.2.1 Simple VNF Migration

In the following, we review the most pertinent works on simple VNF migration, where a single VNF is migrated at a time.

The authors of [32] proposed a VNF migration algorithm to switch off the servers during low-traffic periods in order to save energy. The authors of [33] formulated the migration problem as an integer programming problem and proposed two heuristic algorithms to migrate VMs hosting VNFs with the goal of reducing the migration delay. In [34], the migration problem was formulated as an integer programming problem with the objective of minimizing the migration cost. The authors proposed a heuristic algorithm for migrating VNFs whenever resources were limited in the nodes. The authors of [35] proposed a VNF deployment algorithm followed by a migration algorithm. They aimed to minimize both the power consumption cost and the use of bandwidth resources. The migration algorithm proposed in [36] is based on a neural network that predicts future resource requirements while considering migration delay and VNF reconstruction. A deep reinforcement learning framework for service function chain migration was proposed in [37] under dynamic traffic conditions with the objective of minimizing energy consumption and migration overhead. The authors of [38] investigated the problem of service function chain migration when mobile end-users move from one fog radio access network to another. This work aimed at reducing the migration time and reconfiguration cost by proposing a two-step migration algorithm.

While the works presented above [32, 33, 34, 35, 36, 37] study the VNF migration problem, they do not take into account the scenarios where VNFs must be migrated due to the mobility of end-users. Moreover, they do not consider application component migration in NFV-based hybrid cloud/fog environments. Contrary to cloud-only solutions, a hybrid cloud/fog approach holds great promise to offer faster response times, which can only be achieved if a number of challenges are overcome, most notably the resource heterogeneity and mobility of fog nodes. None of the above-mentioned works have addressed these challenges. Furthermore, it is important to identify the dependent VNFs within a network service, which can help make more sophisticated migration

decisions. We note that all previous studies, including [32, 33, 34, 35, 36, 37, 38] focused on the migration of a single VNF, thus realizing the simple migration scheme. They did not consider migrating a cluster of VNFs by taking into account the dependency between VNFs. Moreover, they did not investigate VNF migration with the main objective of reducing the embedding cost but rather reducing the migration time/overhead and power consumption.

2.2.2 Cluster VNF Migration

To the best of our knowledge, there is no work that has considered cluster VNF migration, though a few studies investigated cluster VM migration in multi-tier applications, e.g., [39, 40, 41, 42]. In [39], the authors aimed to reduce the volume of separated traffic and migration time via the migration of a cluster of VMs, which is realized by the combination of two proposed algorithms. The first algorithm identifies groups of intensively inter-communicating VMs that should not be separated during the migration process. Then, a greedy scheduling algorithm decides the order of migration of the different groups. Refs. [40, 41, 42] aimed to exploit dependencies among VMs, though their main objective was not to reduce the embedding cost. In [40], the authors studied the problem of Cloud-to-Cloud (C2C) migration based on the traffic dependency between multiple VMs to decrease service downtime. The authors of [40] proposed a service-aware strategy for C2C migration of services on multiple VMs, which analyzed the dependency of multiple VMs based on graph theory. Then, the network traffic intensity was used to determine the migration order of dependent VMs by finding the minimum spanning tree. Ref. [41] proposed a cloud migration scheme, which enabled a cluster of VMs to be migrated between different cloud infrastructures. They developed a VM grouping scheme using principal component analysis (PCA) based on traffic dependencies between VMs to reduce the migration downtime of applications. A VM migration method based on minimum-cut and k-means algorithms was proposed in [42] for inter-cloud environments. The proposed method partitioned VMs into subgroups based on the traffic between them. All VMs within a group were scheduled for migration at the same time. The goal was to reduce the amount of traffic between the clouds during the inter-cloud migration and minimize the network latency.

VNF instances run as software in VMs or containers, and a given network service consists of an ordered set of VNFs. Migrating one VNF can affect other VNFs in the chain, as multiple VNFs of

a service chain can be instantiated on a VM. By contrast, in VM migration, the VM is considered as a unique entity for migration. Furthermore, we may only need to migrate the stateful information of VNFs, such that the VNF migration process would require less time [43]. As a result, the main purpose of the papers on VM migration, e.g., [41], [40], [42] was to minimize the migration time and service downtime caused by the migration process. Therefore, the optimization objective of papers in VNF migration would be different from VM migration. Accordingly, the algorithms designed for VM migration would not always apply to the NFV environment. In addition, Refs. [39, 40, 41, 42] considered all VMs of a given network service as a cluster of VMs. Table 2.2 demonstrates the evaluation of the related works on the simple and cluster migration.

Table 2.2: Evaluation of the existing solutions for VNF migration, including simple and cluster migration approaches.

References	Research Area	Migration Approach	VNF Dependency	Embedding Cost	Sharing of VNFI
[32, 33, 34, 35, 37, 38]	VNF	Simple Migration	-	-	-
[34]	VNF	Simple Migration	-	-	✓
[39]	VM	Cluster Migration	-	-	-
[40]	VM	Cluster Migration	-	-	-
[41]	VM	Cluster Migration	-	-	-
[42]	VM	Cluster Migration	-	-	-

2.3 Joint VNF Decomposition and Migration

In this section, we first review the few research works that target the VNF decomposition method. Next, we classify and review the joint method by considering the joint problem of VNF-FG decomposition and embedding problem. While there is no work to consider the joint VNF decomposition and migration problem, we did not consider the joint VNF decomposition and migration.

2.3.1 VNF Decomposition

An architecture is presented in [44], where monolithic VNFs are disaggregated into lightweight “micro” VNFs, enabling a finergrained resource allocation and reducing redundancy in the deployed

stack. The problem of dynamic function composition optimization problem was studied in [45], and proposed a distributed algorithm, using Markov approximation method for the problem. The service chain composition problem with respect to both user and resource failures present in [46], the problem formulated the problem as a non-cooperative game to reduce request latency.

2.3.2 Joint Methods

VNF-FG composition generally aims at composing a suitable VNF-FG for an NS, while VNF-FG embedding seeks to embed the composed VNF-FG in the networks. There exist works that have studied the joint problem of VNF-FG composition and embedding, e.g., [19, 47, 48]. Also, a few works, e.g., [22, 49, 50], have considered the problem of joint VNF decomposition and embedding problems. To the best of our knowledge, there are no works considering joint VNF decomposition and migration problems. In the following, we review the works that consider joint methods for VNF composition and embedding, and VNF decomposition and embedding.

2.3.2.1 Joint VNF Composition and Embedding

The joint VNF-FG composition and the embedding problem was studied in [19][47][48] with the objective of minimizing the embedding cost. In [19], the problem was formulated as an ILP, which was then solved by a greedy heuristic. Similarly, a recursive heuristic algorithm was proposed in [47]. In [48], the authors proposed a priority-based algorithm that consists of a composition phase and a mapping phase. The work in [51] presents an ILP model for the problem of the chain composition and embedding problem with the objective of maximizing revenue of resource sharing and the acceptance rate. The authors in [52] aimed to minimize the service delay by considering the joint composition and embedding of SFCs. They proposed a 2-approximation algorithm by applying graph-theory based techniques, called Eulerian circuit-based hybrid SFP optimization (EC-HSFP).

2.3.2.2 Joint VNF Decomposition and Embeddding

In the VNF decomposition, a VNF can be decomposed into multiple sub-functions and the same type of sub-function. The decomposition method allows for composing function chains and reusing/sharing of sub-functions. Thus, the VNF decomposition brings the efficiency of the resource

Table 2.3: Evaluation of the related works for joint VNF decomposition and migration.

Methods	References	Approach Category	VNF Reusability	Composition	Decomposition	Migration	Resource Cost
Disjoint Method	[44]	Design an Architecture	✓	✓	✓	-	-
	[45],[46]	Markov Approximation Method	-	✓	-	-	-
Joint Method	[19],[47]	Heuristic	-	✓	-	-	✓
	[48]	Priority-based	-	✓	-	-	-
	[51]	Optimization	✓	✓	-	-	-
	[52]	Heuristic	✓	✓	-	-	-
	[22]	Heuristic	✓	✓	✓	-	✓
	[49]	Optimization	-	✓	✓	-	✓
	[50]	Heuristic	-	✓	✓	-	✓

allocation. Also, it makes the VNF embedding problem more flexible and challenging. In [22], the authors studied the VNF embedding problem by considering function decomposition. The authors of [49] presented an optimization framework for the decomposition and deployment of VNFs on a hybrid network. They formulated the problem as mixed-integer linear optimization, aiming to determine the best decomposition according to the traffic demands and network topology. In [50], the authors discussed the joint optimization of service graph decomposition and embedding problems. An ILP-based embedding algorithm was proposed to minimize embedding costs. In [53] proposed deep reinforcement learning for microservice-based NFV for efficient deployment and decomposition of VNFs onto substrate networks. Their objective is to maximize the service acceptance for microservices-based Service Function Chains (SFCs). The authors in [54] decomposed IoT network into multiple small VNF components and developed a deep Q-learning algorithm for embedding decomposed VNFs to improve QoS (e.g.transmission delay). The work in [55] presents a deployment model considering VNF decomposition. They formulated the problem as mixed-integer linear optimization with the objective of guaranteeing reliability and reducing the delay requirement of service function chains.

We note, however, that none of these works took into account VNF migration. VNF migration can largely affect selecting the proper decomposition option for an NS. Furthermore, [19, 47, 48] mainly focused on jointly determining the order of VNFs to compose a VNF-FG and then embed it and they are unsuitable when VNFs in the NSs can be decomposed into sub-functions. Moreover, they assumed the VNF provider had already acquired the details of the decomposed VNFs. This is not a viable option for online VNF mapping when the arrival of VNF or a request is unknown to the network. In addition, Table 2.3 summarizes the evaluation of the existing works on the Joint VNF decomposition and migration.

2.4 Conclusion

In this chapter, we first conducted a survey of the related work. Table 2.1, Table 2.2, and Table 2.3 provide summaries of the reviewed papers, respectively. For each paper, we show both the criteria that are met and those that are not met. As can be seen, none of the reviewed works satisfy all of the criteria.

Chapter 3

Reinforcement Learning-based Optimization Framework for Application Component Migration in NFV Cloud-Fog Environments[1]

3.1 Introduction

Fog computing extends traditional cloud computing architecture by providing computing and storage at the edge of the network, giving way to the so-called hybrid cloud/fog systems. In such systems, some application components can be hosted by the fog while others may reside in the cloud. We view applications as sets of interacting components. In NFV settings, application components can be implemented as VNFs and chained to form VNF-FG. In these systems, the NFVI in which the VNFs are run is provided by the cloud and by the fog. A challenging problem in NFV-based hybrid cloud/fog systems is component migration, given that, in addition to end-users,

[1]This chapter is based on published papers: [1] S. N. Afrasiabi et al, ”Reinforcement Learning-Based Optimization Framework for Application Component Migration in NFV Cloud-Fog Environments,” in IEEE Transactions on Network and Service Management, vol. 20, no. 2, pp. 1866-1883, June 2023.
[2] S. N. Afrasiabi, et al, ”Application Components Migration in NFV-based Hybrid Cloud/Fog Systems,” 2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), Paris, France, 2019, pp. 1-6.

fog nodes can also be mobile. For instance, a drone can act as a mobile fog node [56]. End-users' mobility combined with the mobility of fog nodes could significantly (and quickly) degrade the QoS offered to end-users. Latency may increase when end-users move further from the fog nodes that host components pertinent to the application they are using. The same concept applies when fog nodes move further from end-users. Ideally, when an end-user leaves the coverage area of a given fog node, the components hosted by that node and which are pertinent to the application the end-user is using should also be migrated/moved to another fog node close to the end-user. Such migration promises to ensure service availability and improved QoS [57]. Moreover, if the NFVIs are dynamically consolidated in as few nodes as possible and several nodes are switched off, it would help to minimize the cost (e.g., power consumption cost) [57].

This chapter proposed a solution for the problem of VNF migration in cloud–fog network environments where both fog nodes and end-users are mobile. We model the problem as an optimization framework to minimize the aggregated weighted functions of application delay and cost, and then model the optimization problem as an Markov Decision Process (MDP) problem. Moreover, we design a Deep Reinforcement Learning (DRL) algorithm that makes an efficient and rapid migration decision.

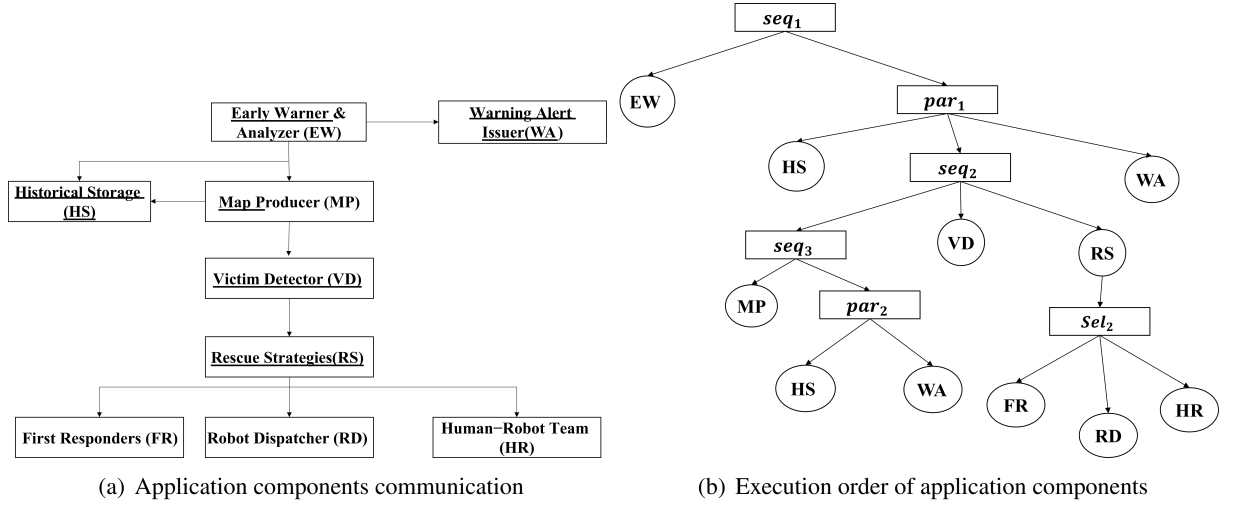
The rest of this chapter is organized as follows. First, we present 3.2 the motivating scenario. The system model and problem definition are explained in Section 3.3. Our DRL-based approach for solving the component migration problem is explained in Section 3.4, followed by the evaluation results in Section 3.5. Finally, We conclude this chapter in Section 3.6.

3.2 Motivating Scenario

As a basis for our motivating scenario, we consider the earthquake early warning and recovery application described in [6]. Figure 3.1 illustrates an example of earthquake early warning and recovery application. According to Fig. 3.1.a, the application comprises six components, and the communication between the application components is demonstrated. The six application components are: Early Warner Issuer & Analyzer (EW), Map Producer (MP), Warning Alert Issuer (WA),

Victim Detector (VD), Rescue Strategies (RS), and Historical Storage (HS). For example, EW processes the data received by various sensors such as cameras and seismic and accordingly detects prospective danger. The data are sent to HS for long-term storage and analysis and to WA for public warning. MP component processes these data to find the epicenter location and produce damage assessment maps. The VD uses these maps to locate possible humans. The RS is then informed about taking immediate life-saving decisions if a victim is detected. The RS instructs either First Responders (FR), Robot Dispatchers (RD), or HumanRobot Team (HR) to begin the rescue missions. The application can be described by a structured graph, which shows whether the components are executed in parallel or sequence. The execution order of elements can be defined by a structured graph (see Fig. 3.1.b). For example, "RS" and "HS" are executed in sequence, while "VD" and "WA" are executed in parallel. We consider a system with three layers: (i) an end-user layer that includes mobile end-user devices; (ii) a fog layer that includes both mobile fog nodes such as drones and static fog nodes e.g., sensors; and (iii) a cloud layer that consists of distant data centers as shown in Fig 3.2. Accordingly, some of the application components can run in the fog layer (e.g., Warning Alert Issuer), whereas the others can run in the cloud layer (e.g., Historical Storage). Given that the considered earthquake early warning and recovery use-case is a delay-sensitive application, relying only on static fog and/or cloud nodes may result in excessive response times, which can lead to catastrophic consequences. On the other hand, drones serving as mobile fog nodes not only reduce the response time significantly but also offer more reliable connectivity to end-users, especially given that end-users may not access terrestrial base stations due to high congestion and increased communication delay.

Given that a disaster may affect a large area, the drones may need to move around for several purposes including capturing images, damage estimation, and finding the presence of human beings, among others. As fog nodes (e.g., drones) and end-users (e.g., robots, cellphones) can be mobile, we assume that Drone 1, which hosts the Early Warner & Analyzer components, moves from location A to location B, and end-user devices move from location A to C. In Fig. 3.2.a, Drone 1 may move from location A to B to take care of a more urgent task at location B. For example, a group of mobile end-users may be suffering from a lack of reliable communication due to temporary congestion at location B, thus requiring Drone 1 to move from location A to B to provide the users



(a) Application components communication

(b) Execution order of application components

Figure 3.1: Example of earthquake early warning and recovery application [6].

with reliable connectivity. As a result, Drone 1 becomes far away from the mobile end-user devices from which it gets its desired information. Let the distance between the end-user devices and Drone 1 be denoted by $d1$. Also, assume that there is another drone (i.e., Drone 2) at a distance $d2$ from the end-user. Therefore, it is more efficient to migrate the Early Warner Issuer $\&$ Analyzer components from Drone 1 to Drone 2 to reduce the communication latency, especially given that $d1 > d2$. Moreover, Drone 1 (which acts as a fog node) can be switched off, as it does not host any components. This action can save power, reducing the power consumption cost (see Fig. 3.2.b). Such a migration decision can be made by the so-called migration decision module as a part of the platform-as-a-service (PaaS) option [58]. The migration decision module is in charge of deciding whether to migrate and where to migrate VNFs based on the data collected by other modules, such as the monitoring engine in the PaaS [58].

3.3 System Model and Problem Formulation

In this section, we first present the system model, and then formulate and analyze the problem of application component migration in an NFV-based hybrid cloud/fog system.

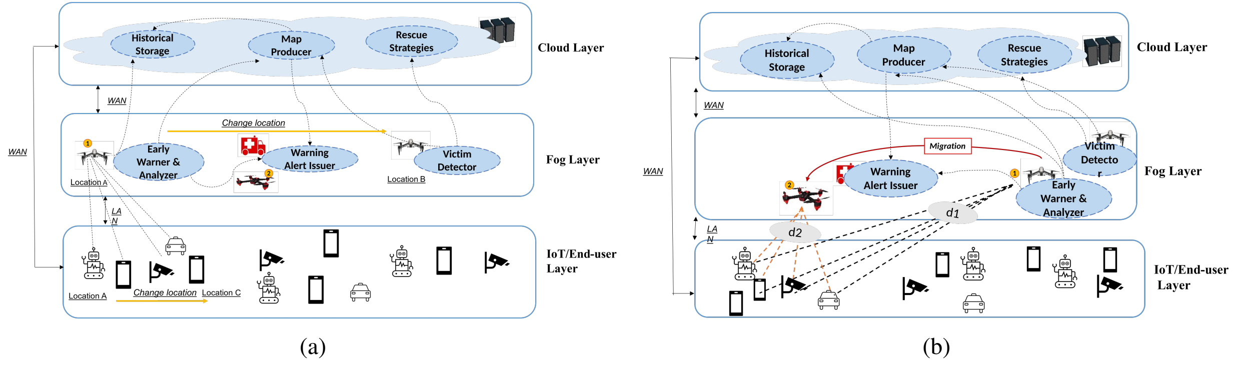


Figure 3.2: Migration of an application component for an earthquake early warning and recovery application :(a) Before Moving, (b) After Moving

3.3.1 System Model

In the following paragraphs, we explain the model of the application components implemented as VNFs, structured VNF-FGs, underlying network, and the end-users that interact with the application components. Table 3.1 lists the key notations and decision variables.

3.3.1.1 VNFs

We assume that an application is composed of several components and each component is implemented as a VNF. Let K denote the set of different VNF types (e.g., victim detector or warning alert issuer). The processing capacity and maximum allowable capacity utilization of VNF type k are denoted by c_k and μ_k, respectively. Each VNF k has a finite CPU resource requirement denoted by r_k and memory resource requirement denoted by m_k.

3.3.1.2 VNF-FG Request

The whole application is modeled as a chain of VNFs, also referred to as VNF-FGs. Let Req be the set of structured VNF-FG requests received by the system. We denote a single request with $R \in Req$ and the set of required VNF types for request R is presented by vnf_R. Each VNF-FG is converted into a tree structure [1]. While the leaf nodes represent the application components, the middle nodes represent one of the substructures in the form of a sequence, parallel, selection, or loop [1, 6]. Such tree structure allows for defining the relationship between the VNFs in the chain,

Table 3.1: Summary of main notations.

Parameter	Definition	Parameter	Definition
k	VNF type $k \in K$	$l_z(t)$	Location of $z \in Z$ at time slot t
c_k	Processing capacity of VNF k	$b_{e_{qm}}(t)$	Available bandwidth between nodes z_q and z_m at time-slot t
μ_k	Maximum allowable utilization of VNF k	$p_{e_{qm}}(X(t), Y(t))$	Propagation delay between node z_q located at X and node z_m located at Y at time slot t
m_k	Memory requirement of VNF k	μ_{qm}	Maximum allowed utilization between nodes z_q and z_m
r_k	Resource requirement of VNF k	U_R	Set of end-users
$ip(k)$	Immediate predecessors of VNF k	$l_{ur}(t)$	Location of end-user $u_r \in U_R$ at time slot t
$a^R_{ip(k),k}$	Traffic rate from $ip(k)$ to VNF k	$W^r_{u \times k}$	1 if there is a communication between end-user u and VNF k for request R and 0 otherwise
Req	Set of structured VNF-FG requests	$a^R_{u \times k}$	Traffic rate between end-user u and VNF k
vnf_R	Set of required VNF types for request R	$b_{e_{uz}}(t)$	Available bandwidth between end-user u and node z at time-slot t
Z	Set of cloud/fog nodes	$\|T\|$	Length of a time-slot
E	Set of links	ϕ	Cost of bandwidth for transmitting a single bit of traffic
$c^{CPU}_z(t)$	CPU capacity of node $z \in Z$ at t	β	Cost per consumed unit of power
$c^{Mem}_z(t)$	Memory capacity of node $z \in Z$ at t	$h_{i,k,z}(t)$	1 if an instance i of VNF k hosted at cloud/fog node z at time-slot t and 0 otherwise
$\mu_{c^{Mem}_z}(t)$	Maximum allowed utilization of $c^{Mem}_z(t)$	$h^R_{i,k,z}(t)$	1 if an instance i of VNF k hosted at cloud/fog node z at time-slot t for request R and 0 otherwise
$\mu_{c^{CPU}_z}(t)$	Maximum allowed utilization of $c^{CPU}_z(t)$	$\lambda_z(t)$	1 if the node z is switched on at time-slot t and 0 otherwise
d^k_z	Delay of processing one unit of traffic for VNF type k located at node $z \in Z$		

where the immediate predecessor of VNF k is denoted by $ip(k)$. For request R, the amount of traffic transmitted from the immediate predecessor $ip(k)$ to VNF k is indicated by $a^R_{ip(k),k}$.

3.3.1.3 Network

We consider the following three layers: (i) mobile end-user layer, (ii) fog layer (including both mobile and static fog nodes), and (iii) cloud layer. The cloud and the fog layers offer the underlying NFVI that can host the VNFs. Let Z be the set of nodes (which can be either cloud or fog nodes).

Each node can host a single or multiple VNFs. Node $z \in Z$ can be characterized by its CPU capacity $c^{CPU}_z(t)$ (in terms of the number of cores) at time t (which changes over time) and available

memory $c_z^{Mem}(t)$ (in terms of number of gigabytes available) at time t. The processing delay of a unit of traffic for VNF type k on node $z \in Z$ is denoted by d_z^k. In each time slot t, the nodes' locations may change. Let $l_z(t)$ represent the location of node $z \in Z$ at time t, $p_{e_{qm}}(X(t), Y(t))$ be the propagation delay between node z_q at location X and node z_m at location Y at time t. The set E of links between the nodes are also characterized by available bandwidth capacity $b_{e_{qm}}(t)$ between nodes z_q and z_m at time t. The maximum allowable link utilization between nodes z_q and z_m is denoted by μ_{qm}.

The application components may communicate with end-users' devices. We denote the set of end-users communicating with the application components in request $R \in Req$ by U_R. In each time slot t, the location of an end-user $u_r \in U_R$ is represented by $l_{u_r}(t)$. The available bandwidth and propagation delay between end-users at location X at time t and node z at location Y at time t are denoted by $b_{e_{uz}}(t)$ and $p_{e_{uz}}(X(t), Y(t))$, respectively. Also, let $\mu_{e^{uz}}$ be the maximum allowable link utilization between end-user devices and node z. We define a matrix $W_{u \times k}^R \in \{0, 1\}$ that is 1 if there is a communication between u and k for request R; otherwise, it is equal to 0. Finally, the traffic rate between end-user u and VNF k of request R is denoted by $a_{u \times k}^R$.

3.3.2 Problem Formulation

Given the current locations of mobile end-users and fog nodes along with their CPU capacity, memory capacity, and available bandwidth, we aim to solve the problem of VNF migration cost with the main objective of minimizing the overall delay and cost. In the following, we formulate the problem as an optimization problem. Similar to [56, 27, 59], we consider a slotted time model, where the time is divided into time slots of length T. The index of each time-slot is denoted by $t \in [1, 2, ..., t_{max}]$ and $t = 1$ is considered as the starting point of the system, whereas t_{max} corresponds to the final time-slot. Note that t_{max} is chosen large enough to cover the execution of all requests. We assume that the locations of both fog nodes and end-users remain fixed for the duration of a given time slot and change from one time slot to another [27]. The solution to the optimization problem is characterized by the following decision variables:

- $h_{i,k,z}(t)$: A binary variable equal to 1 when instance i of VNF k is hosted at cloud/fog node z at time t; Otherwise it is 0.

- $h^R_{i,k,z}(t)$: A binary variable equal to 1 when instance i of VNF k is hosted at cloud/fog node z at time t for request R; Otherwise, it is 0.
- $\lambda_z(t)$: A binary variable equal to 1 when node z is switched on at time t; Otherwise it is 0.

Next, we explain our delay and cost model. To calculate the carried traffic of VNFs, we need to consider all the inputs and output traffic. We note, however, that the traffic pattern of the first VNF is different from that of the other VNFs, as it cannot connect to the predecessor VNF. The total incoming traffic to a given VNF is the summation of the traffic from its predecessor VNFs ($a^R_{ip(k),k}$) and the traffic from end-user devices ($a^R_{u\times k}$), given by:

$$\begin{aligned} a^R_k &= \sum_{u\in U_R} W^R_{u\times k}\cdot a^R_{u\times k} && k=1 \\ a^R_k &= \sum_{ip(k)\in IP(K)} a^R_{ip(k),k} + \sum_{u\in U_R} W^R_{u\times k}\cdot a^R_{u\times k}, && k\geq 2. \end{aligned} \tag{3.1}$$

3.3.2.1 VNF-level calculation

The calculation of the application delay and cost is performed based on parsing the associated tree structure of the VNF-FG. The delay and cost of the leaf nodes representing VNFs are calculated first. These values are then aggregated to calculate the delay and the cost for the middle nodes which represent the sub-structure.

(a) *Delay:*

Application delay is the time it takes when the execution of the first component begins until the execution of the last component has been completed. The total delay consists of three parts: (1) the total processing delay d_{proc} of the traffic sent from end-user devices or predecessor VNFs, (2) communication delay d_{com} between mobile end-user devices and cloud/fog nodes; and (3) migration delay d_{migT} for migrating a VNF k.

The processing delay is the time spent by nodes (cloud/fog) in order to process a request sent by end-user devices or the immediate predecessors of VNF k. If node z is selected to process

traffic a_k^R, the processing delay of a time slot denoted by $d_{proc}(R, k, t)$ is calculated by:

$$d_{proc}(R, k, t) = \sum_{z \in Z} \sum_{i \in I_k} h_{i,k,z}(t) \cdot a_k^R d_z^k. \tag{3.2}$$

The communication delay is the time spent to transmit a bit of traffic between edges, and consists of both transmission and propagation delays [6]. The transmission delay in a time slot can involve the traffic transmission from $ip(k)$ to k (i.e., Eq. (3.4)) and traffic transmission from u_r to k (i.e., Eq. (3.5)). Then the total communication delay of VNF k belonging to request R with the predecessor VNFs and end-users in a given time slot is given by:

$$d_{com}(R, k, t) = \max(d_{com}(k, ip(k), t), d_{com}(k, u_r, t)), \tag{3.3}$$

where

$$d_{com}(k, ip(k), t) = \sum_{z \in Z} \sum_{i \in I_k} h_{i,k,z}(t)(\frac{a_{ip(k),k}^R}{b_{e_{qm}}(t)}) + p_{e_{qm}}(X(t), Y(t)), \tag{3.4}$$

$$d_{com}(k, u_r, t) = \sum_{z \in Z} \sum_{i \in I_k} h_{i,k,z}(t).(\frac{a_{u \times k}^R}{b_{e_{uz}}(t)}) + p_{e_{uz}}(X(t), Y(t)), \tag{3.5}$$

To manage the mobility of end-user devices and fog nodes, especially when they move far away from each other, migration can take place, and thus $h_{i,k,z}(t) \neq h_{i,k,z}(t-1)$. When a VNF is migrated, the propagation and transmission delays contribute to the migration time. The downtime of migrating a VNF is equivalent to the time taken to occupy the network bandwidth by migrating the memory data of VNF [59]. The larger the amount of memory to be migrated and the smaller the available bandwidth between nodes, the longer the time that the migration occupies the network bandwidth. The delay of migrating k from node z_q

at location X at time $t-1$ to node z_m at location Y at time t is estimated by:

$$d_{mig}(R,k,t) = \sum_{z \in Z} \sum_{i \in I_k} h_{i,k,z_q}(t-1) h_{i,k,z_m}(t) \cdot (p_{e_{qm}}(X(t), Y(t)) + \frac{m_k}{b_{e_{qm}}(t)}). \tag{3.6}$$

Clearly, without migration, there is no migration delay and so the processing delay is calculated based on Eq. (3.4), where all the whole traffic is processed at the host node. However, in the case of migration, the unfinished traffic will be processed on node z_m. While the processing delay of unfinished traffic at a previous time slot at node z_q was calculated in Eq. (3.2), the processing delay of unfinished traffic for VNF k at node z_m must be calculated in the current time slot where $h_{i,k,z_m}(t) = 1$ and the processing delay of the unfinished traffic of previous time slot $h_{i,k,z_q}(t-1)$ must be subtracted.

Therefore, the total migration delay consists of the migration downtime and the new processing delay of the unfinished traffic [59]. Let the size of the unfinished traffic be denoted as ${a'}_k^R$. If the transmission delay is longer than the length of the time slot, then ${a'}_k^R \geq a_k^R$, which means that all the traffic will be processed at the source node. If the traffic a_k^R is processed within a time slot, then the size of the unfinished traffic is ${a'}_k^R = a_k^R$, meaning that no migration will take place. As a result, $d_{proc}(R,k,t) + d_{com}(R,k,t) \leq |T|$. On the other hand, if only a fraction of the traffic can be finished within one time slot, then the amount of unfinished traffic at time-slot t is given by:

$${a'}_k^R = min\{a_k^R, max\{0, a_k^R - \frac{|T| - d_{com}(R,k)}{d_{proc}(R,k)} a_k^R\}\}, \tag{3.7}$$

which is processed at the destination node. It is also worth noting that there is no migration in the first time slot, i.e., $d_{migT}(R,k,t) = 0$ for $t = 1$. In the first time slot, the application components are placed randomly so there is no migration delay. The total migration delay of unfinished traffic ${a'}_k^R$ calculated by:

$$d_{migT}(R,k,t) = d_{mig}(R,k,t) + \sum_{z\in Z}\sum_{i\in I_k} h_{i,k,z_m}(t).{a'}_k^R D_k^R - \sum_{z\in Z}\sum_{i\in I_k} h_{i,k,z_q}(t-1).{a'}_k^R D_k^R \tag{3.8}$$

(b) *Cost:*

The overall cost consists of the monetary cost of migration and the cost of power consumption. The migration cost $C_{mig}(R,k,t)$ describes the cost of bandwidth occupation involved when migrating a VNF between nodes z_m and z_q at time t. Letting ϕ be the bandwidth occupation cost for transmitting a single bit of traffic, migration cost $C_{mig}(R,k,t)$ is obtained by:

$$C_{migT}(R,k,t) = \sum_{z\in Z}\sum_{i\in I_k} h_{i,k,z_m}(t-1)\cdot h_{i,k,z_q}(t)\cdot\phi\cdot m_k. \tag{3.9}$$

Several nodes can be switched off by migrating VNFs to nodes that have enough capacity, thereby realizing a considerable power consumption reduction. However, if too many VNFs are hosted in a single node, the resource over-utilization of that node will be increased dramatically, which can lead to performance degradation and longer processing delays, and the transmission delay could also increase as the distances between the mobile end user devices and nodes increase. A CPU consumes a larger proportion of energy than disk storage network interfaces. Hence, we calculate the power consumption cost as the CPU power consumption cost. If node z is switched off, then the power consumption would be negligible, i.e., $p_{total} \approx 0$. Let β denote the cost per consumed power watt. The cost of total energy consumption at a node for VNF k in a given time-slot can be estimated by:

$$C_{power}(R,k,t) = \sum_{z\in Z}\sum_{i\in I_k}\beta\cdot h_{i,k,z}(t)\cdot p_{idle} + (p_{max} - p_{idle})\cdot u_z(t). \tag{3.10}$$

where p_{max} is the maximum power consumed when the node is fully utilized, p_{idle} is the power consumption when the node is in the idle state (no computation task or no VNF), and

$u_z(t)$ is the CPU utilization (resource utilization) of z at time t given by

$$u_z(t) = \min\{\frac{\sum_{i \in I_k} h_{i,k,z} r_k}{c_z^{CPU}}, 1\},$$

which varies over time.

3.3.2.2 VNF-FG level calculation

The total delay and cost of application for a time slot can be calculated by traversing the tree structure from the leaves to the root and aggregating the calculated delay and cost values of the nodes from the bottom to the top. In the following, we estimate the processing delay at a given time slot for the sub-structures' sequence, parallel, selection, or loop. A sequence sub-structure accumulates the time and the cost of its children.

When a node in the tree is a substructure, namely S_i (sequence, parallel, selection, or loop), the processing delay for that node $d_{proc}(R, S_i, t)$ is calculated by Eq. (3.11). A sequence sub-structure accumulates the processing delay of all its children. A loop can be considered as a sequence sub-structure that is repeated for a certain number of iterations it, defined as the expected number of iterations of a loop structure to be calculated by: $it = \frac{q}{1-q}$, where q is the probability of the loop's occurrence.

For a parallel sub-structure, its children are all executed in parallel, hence the processing delay is determined based on the maximum delay value of its children. In a selection sub-structure, the probabilities of the selections of the children are involved in the calculation. Let $₱_k$ be the probability of selecting a child S_i a selection sub-structure.

$$d_{proc}(R, root, t) = \begin{cases} \sum_{k \in Si} d_{proc}(R, k, t) & S_i \text{ is } seq \\ \max_{\{k \in S_i\}} d_{proc}(R, k, t) & S_i \text{ is } par \\ \sum_{k \in Si} ₱_k \cdot d_{proc}(R, k, t) & S_i \text{ is } sel \\ it. \sum_{k \in Si} d_{proc}(R, k, t) & S_i \text{ is } loop \end{cases} \tag{3.11}$$

In this regard, $d_{mig}(R,k,t)$ and $d_{com}(R,k,t)$ are calculated by aggregating the migration and communication delay, respectively, of VNFs, similar to Eq. (3.11). while the power consumption and the migration cost for the sequence, selection, and loop substructures at a given time slot are calculated similar to Eq. (3.11), for the parallel substructure, the cost is the sum of the costs for all its children, for which the migration cost is given by:

$$C_{mig}(R, root, t) = \sum_{k \in S_i} C_{mig}(R, k, t), S_i \text{ is } par \tag{3.12}$$

In this regard, the $C_{power}(R,k,t)$ for the parallel substructure is calculated similar to Eq. 3.12.

Finally, to calculate the total delay and cost of a VNF-FG for a time slot t the delay and the cost of the root of the tree are computed by aggregating the delay and the cost of the VNFs and of the basic sub-structures from the bottom to top. The total delay and the total cost of a VNF-FG request R at times slot t are given by:

$$Delay(R,t) = d_{proc}(R, root, t) + d_{com}(R, root, t) + d_{migT}(R, root, t), \tag{3.13}$$

$$Cost(R,t) = C_{mig}(R, root, t) + C_{power}(R, root, t). \tag{3.14}$$

Our objective is to enable the migration of VNFs in the cloud and fog NFVI such that the aggregated weighted function of the delay and cost is minimized over all the time slots. Specifically, we consider the following objective function:

$$F = \sum_{t=0}^{t=t_{max}} \sum_{\forall R \in Req} \omega \cdot Delay_0(R) + (1-\omega) \cdot Cost_0(R), \tag{3.15}$$

where ω (a real number in the range [0,1]) determines the weight of delay over cost. To make a proper trade-off between the two heterogeneous metrics of $Delay(R)$ and $Cost(R)$ in Eqs. (3.13) and (3.14), Note that $Delay_0(R)$ and $Cost_0(R)$ in Eq. (3.15) are normalized values of $Delay(R)$ and $Cost(R)$, respectively. More specifically, we aim to solve the following optimization problem:

$$\min F \tag{3.16}$$

subject to:

$$\sum_{R \in Req} \sum_{i \in I_k} h^R_{i,k,z}(t) \leq N \cdot \lambda_z(t), \quad \forall k \in K, z \in Z, \quad t \in [1, 2, .., t_{max}] \quad NlargeConstant, \tag{3.17}$$

$$\sum_{R \in Req} a^R_{ip(k),k} \cdot h^R_{i,k,z_q}(t) \cdot h^R_{i,k,z_m}(t) \leq \mu_{qm} \cdot b_{e_{qm}}(t) \quad \forall q, m \in Z, \quad t \in [1, 2, .., t_{max}], \tag{3.18}$$

$$\sum_{R \in Req} W^R_{u \times k} \cdot a^R_{u \times k} \cdot h^R_{i,k,z}(t) \leq \mu_{e_{uz}} \cdot b_{e_{uz}}(t) \quad \forall z \in Z, u \in U_R, \quad t \in [1, 2, .., t_{max}], \tag{3.19}$$

$$\sum_{R \in Req} \sum_{i \in I_k} r_k \cdot h_{i,k,z}(t) \leq \mu_{c^{CPU}_z} . c^{CPU}_z(t) \quad \forall z \in Z, \quad t \in [1, 2, .., t_{max}], \tag{3.20}$$

$$\sum_{R \in Req} \sum_{i \in I_k} m_k \cdot h_{i,k,z}(t) \leq \mu_{c^{Mem}_z} \cdot c^{Mem}_z(t) \quad \forall z \in Z \quad t \in [1, 2, .., t_{max}], \tag{3.21}$$

$$\sum_{\forall R \in Req} A^R_k h^R_{i,k,z}(t) \leq \mu_k \cdot c_k \quad \forall k \in K, z \in Z \quad t \in [1, 2, .., t_{max}], \tag{3.22}$$

The resultant solution must satisfy the following constraints of the optimization problem (F). Constraint (5.17) guarantees that the server node is switched on when it hosts at least one VNF k. Constraints (5.18-5.19) ensure that the communication links in the cloud, in the fog, between the cloud and the fog, between the end-user devices and the cloud, and between the end-user devices and the fog are not overloaded in terms of bandwidth capacity. Constraint (3.20) ensures that the computational resource capacity of cloud and fog nodes are not exceeded. Constraint (3.21) guarantees that the total amount of memory occupied by a VNF assigned to node z should not exceed the available memory. Finally, constraint (3.22) ensures that the capacity of an instance of VNF k is not overloaded.

3.3.3 Problem Analysis

The problem of application component migration in an NFV-based hybrid cloud/fog system is NP-hard, as it can be simplified to the NP-hard bin packing problem by considering nodes as bins and VNFs as items. Thus, the problem can be solved heuristically or meta-heuristically [60]. The migration problem that we define here in section 3.3 includes the dynamic parameters, such as the locations of mobile fog nodes and end-user devices. The availability of resources may also change from one-time slot to another. Moreover, when the parameters change, the heuristic algorithms need to execute in each time slot, and they cannot adjust their solutions in accordance with these changes. Most of the existing heuristic algorithms are unable to handle large-sized problems in a computationally efficient manner, usually resulting in a long decision-making delay. Since the dimensions of our problem are complex and vary over time, heuristics are not suitable for our problem.

To handle the level of uncertainty of mobility and dynamicity, we present the application component migration problem in the form of an MDP. We then adopt a DRL approach to find the solution. The RL agent can automatically learn the ever-changing environment and update its decision through interactions with the environment. In the following, we demonstrate that our defined problem has Markov property, which is a memory-less property [61]. Then, in Section 3.4, we define the main components of our developed MDP, including the state set, action set, and reward function. We then propose our RL-based application component migration algorithm.

Eq. (3.15) can be represented as the aggregation of cost and delay over each time slot with length $|T|_\tau$, $\forall \tau = 0, ..., \kappa$, which can be defined as:

$$\varphi(\tau) = \sum_{tau=0}^{k} \int_{t_0+\tau|T|}^{t_0+(\tau+1)|T|} \omega \cdot Delay(\tau) + (1-\omega) \cdot Cost(\tau) d\tau, \tag{3.23}$$

where $Delay(\tau)$ and $Cost(\tau)$ are the delay and cost over $|T|_\tau$.

Assume that $\varphi(\tau)$ is the aggregation of the cost and delay until $|T|_\tau$. We can conclude the following equality holds for $\varphi(\tau)$:

$$\varphi(\tau+1) = \varphi(\tau) + \omega \cdot Delay(\tau) + (1-\omega) \cdot Cost(\tau). \tag{3.24}$$

where the future value $\varphi(\tau + 1)$ depends only on the current values of the parameters. This shows that our component migration problem has a memory-less property. Moreover, the mobility of end-user devices and fog nodes' are a sequential decision-making processes also have a memory-less property [62]. Therefore, our problem can be modeled as an MDP and thus reinforcement learning can be exploited to solve the problem [63].

3.4 RL-Based VNF Migration

Using the formulation presented in Section 3.3, we define the main components of the MDP in our problem and show how we utilize this framework to develop our DRL-based application component migration solution. Since our problem is a sequential decision-making process, we exploit an advanced type of DQL, a double deep Q-network (DDQN) enhanced with LSTM memory cells. In the following, we first explain the motivation for choosing this specific Q-network architecture and then discuss the limitations of conventional recurrent neural networks and explain how LSTM memory cells can overcome those limitations. Finally, we propose our DDQN-based algorithm for solving the problem of application component migration.

3.4.1 MDP Framework

Our MDP model can be characterized by the following three main elements:

- System States: We consider state space $S = \{s_t | t = 0, 1, 2, .., t_{max}\}$ in a specific slot t where s_t described by a 2-tuple given by:

$$s_t = \{\mathrm{H}(t), \Gamma(t)\}, \tag{3.25}$$

  where $\mathrm{H}(t)$ and $\Gamma(t)$ represent the decision variables $h_{i,k,z}(t)$ and $\lambda_z(t)$, respectively. Therefore, the state of the system at time t represents whether the instance i of VNF k is hosted at cloud/fog node z or not, and indicates whether the cloud/fog node z is switched on or off.

- System Action: The system action consists of all the possible cloud/fog nodes that the current instance i of VNF k can migrate to. $A^{q,m}_{i,i+1}(t)$ specifies whether the VNF on node q at a state

i migrate to node m in state $i+1$ at time slot t. We use $a_t = \{A^{q,m}_{i,i+1}(t)\} \in A$ to identify the action of selecting a new node for s_t, where A is the set of all possible actions in time slot t. The action causes the system state to change to a new state s_{t+1}.

- Reward Function: The reward function determines the immediate consequence of choosing action a_t in state s_t order to assess the short-term quality of a performed action. Our goal is to minimize the value of the objective function given by Eq. (3.15). The action leading to a smaller value of the objective function is associated with a larger reward. The immediate reward of action a_t current state s_t based on Eq. (3.15) is defined as follows:

$$R^{s_t}_{a_t} = -(\sum_{R \in Req} \omega \cdot Delay_0(R) + (1-\omega) \cdot Cost_0(R)). \tag{3.26}$$

- P_a is the transition probability from state s_t at time t to state s_{t+1} at time $t+1$ after action a_t which is defined on Eq. (3.27):

$$P_a(s_t, s_{t+1}) = P(S_{t+1} = s'|s_t = s, a_t = a). \tag{3.27}$$

3.4.2 Design of the Deep RL Agent

In this subsection, we first introduce the traditional RL method to obtain optimal state-action value in an MDP and then explain our motivation for using DDQN. The concept that an agent learns by interacting with the environment is key to RL. The RL agent automatically learns the ever-changing environment and continually updates its decisions. The interaction of an agent and its environment utilizing the DDQN strategy is shown Fig. 3.3. We will use this figure and Algorithm 1 as we explain the theoretical aspect of our work.

Our proposed approach is based on Q-learning, a model-free algorithm and one of the most widely used RL strategies for placement problems [64], computation offloading strategies [65], and migration problems [57, 59]. Q-learning offers the most rapid computation of all the different types of reinforcement learning methods [63], and it is a simple way for an agent to learn how to act optimally. Our Q-learning policy is modeled as an action-value Q-function $Q(s_t, a_t)$ (also known

as the Q-value), given by:

$$Q_{\pi}(s_t, a_t) = \mathrm{E}_{\pi}[\sum_{t=0}^{t=t_{max}} \gamma^t R_{a_t}^{s_t}]. \tag{3.28}$$

where π is the policy (a series of actions, each executed at a given a state) and $\gamma = [0, 1]$ is the discount factor, which is utilized to balance between the immediate and long-term rewards. The Q-value is the maximum expected reward an agent can reach by taking a given action a_t from the state s_t. The optimal Q-function can then be defined as:

$$Q^*(s_t, a_t) = \max_{\pi} Q_{\pi}(s_t, a_t). \tag{3.29}$$

In our Q-learning algorithm, the action is selected based on the ϵ-greedy algorithm, which consists of exploration and exploitation [66]. The Q-learning algorithm recursively computes and updates the Q-value of the state-action pairs until all the states are met and all the actions have been experienced, as follows:

$$Q^{new}(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha[R_{a_t}^{s_t} + \gamma \max_{a} Q(s_{t+1}, a_t) - Q(s_t, a_t)], \tag{3.30}$$

where α is the learning rate. Although Q-learning works well when the state and action space is small, it is quite challenging to find the optimal strategy in a large Q-table [63].

In Q-learning, all the states should be met, and all the actions should be experienced. In our problem, the state and action spaces are not only large, but they also vary over different time slots. Moreover, our problem has no terminal state, which makes using a Q-learning algorithm unsuitable for our purpose [67]. In order to address this shortcoming, we approximate the Q-values for unmet states/actions using a deep neural network (DNN)-based approach, known as a nonlinear gradient-descent function approximation[63]. This approach eliminates the need to visit all the state/action pairs to compute the Q-values. In DQN, a DNN is used to learn $Q^*(s, a) \approx Q(s_t, a_t, \theta)$ where $Q(s_t, a_t, \theta)$ is the output of a DNN with weight θ. In deep Q-learning, (s_t, a_t, R_t, s_{t+1}) is abstractly stored in the DNN. As the state of the system continuously changes at each time-slot, it is important that the DNN is updated frequently. Therefore, the training of the DNN based on

historical information is stored in an experience replay memory D of size K. The experience replay ensures that the optimal policy cannot be driven to a local minima [67]. During the learning, instead of using only the currently experienced values of (s_t, a_t, R_t, s_{t+1}), the DNN can be trained by selecting a random mini-batch from D to form a learning batch. These learning batches are then used to feed the DQN and update the estimated Q-value. The DQN uses a *Target Network* with an *Online Network* to stabilize the overall network performance. The *Target network* is used to select an action, whereas the *Online Network* is used to evaluate these actions. In DQN, instead of updating the Q-table we update the parameters of the neural network to make better predictions. The DQN can be trained by minimizing the loss function $L(\theta)$ given by:

$$L(\theta) = \mathrm{E}_{s,a}[(y_t^{DQN}) - Q(s_t, a_t; \theta_t))^2], \tag{3.31}$$

where $y_t^{DQN} = R_{a_t}^{s_t} + \gamma \max_{a_t}(s_{t+1}, a_t; \overline{\theta_t})$ is the target Q-value with current parameter $\overline{\theta_t}$ (the weight of the *Target Network*), and $\overline{\theta_t}$ gets updated periodically to θ_t based on the gradient descent rule. The action a_t is selected from the *Online Network* using ϵ-greedy policy. The weights of the *Target Network* are fixed for a fixed number of iterations, while those of the *Online Network* are updated.

Moreover, since the same values of θ_t reused to select and evaluate an action in DQN, the Q-value function may be an overestimated and unstable estimation of the Q-value [7]. Therefore, the so-called double DQN [7] is used to mitigate this problem, so that two Q-value functions train simultaneously, one for selecting actions with weight θ_t (*Online Network*) and the other for evaluating an action's value or for estimating the Q-value (*Target Network*) with weight $\overline{\theta_t}$. In this regard, the target y_t^{DQN} is replaced by y_t^{DDQN} given by:

$$y_t^{DDQN} = R_{a_t}^{s_t} + \gamma Q(s_{t+1}, \text{argmax}_{a_t} Q(s_{t+1}, a_t; \theta_t); \overline{\theta_t}), \tag{3.32}$$

Finally, the gradient descent will be calculated by an *Online Network* and then the weight θ_t' is updated to θ_t the weight of the Q-network at time t. The gradient descent update rule for θ_t is defined as follows:

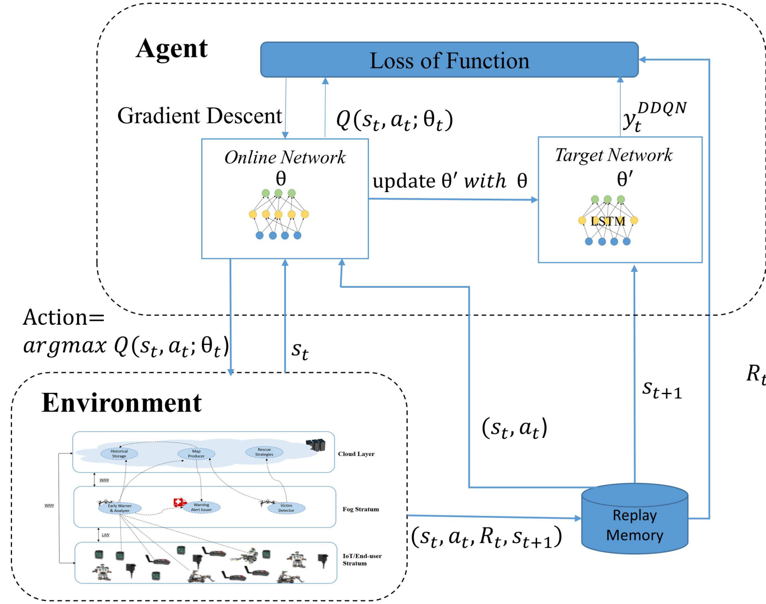


Figure 3.3: Agent-environment interaction with DDQN strategy.

$$\theta_{t+1} = \theta_t + \alpha(y_t^{DDQN} - Q(s_t, a_t; \theta_t)). \bigtriangledown_t .Q(s_t, a_t; \theta_t). \tag{3.33}$$

where $\bigtriangledown_t$ is the gradient descent at time t.

Figure 3.3 shows the RL procedure with DDQN and how the agent updates its decision by interaction with the environment. The DDQN algorithm is composed of three main components, the *Online Network* $(Q(s_t, a_t; \theta))$ that defines the behavior policy, the *Target Network* $(Q(s_t, a_t; \theta'))$ that is used to generate the target Q-values, and the replay memory that the agent uses to sample random transitions for training the Q-network.

3.4.3 DDQN with LSTM Cells

Here we explain the theoretical advantage of using LSTM memory cells in our designed DRL agent. Previous works have shown the effectiveness of LSTM in predicting user mobility [68], the task scheduling problem [69, 70], and in resource allocation [71]. LSTM is a good solution for predicting a time series problem with long-term dependencies [72], as it can hold long-term experiences. We integrate an LSTM layer in a DDQN to store the information acquired during

previous time slots, thereby allowing for additional information to give a more accurate prediction of the Q-value of the component migration problem. The prediction of the current Q-value is not only dependent upon more recent experience (s_t, a_t, R_t, s_{t+1}), it is also dependent upon some significant experiences in the distant past.

LSTM [37] is an approach to address the vanishing gradient problems of conventional RNNs [73]. The vanishing and exploding gradient problems result in inaccurate results and time-consuming learning processes [73]. In the following, we elaborate on the theoretical aspect of the vanishing gradient problem and explain how LSTM memory cells can be a suitable solution for our problem. Let us consider the gradient rule given by Eq. 3.33. After passing many gradient update steps, and when t becomes large enough, the error and the gradient-based descent become too small, which prevents the value of θ_t from being updated. This may completely stop the NN from further training. To address this problem, RNNs are enhanced with LSTM [74].

The memory cell is the key component of LSTM which is able to learn long-term dependencies information of the network states. Each cell contains a sigmoid layer to optionally pass information, where each cell is responsible for protecting and controlling its state.

In our work, we consider an LSTM cell consisting of three gates: (i) a forget gate, (ii) an input gate, and (iii) an output gate. The forget and the input gates are used to selectively forget the information of the previous cell or saved that cell state's information, respectively. The output gate controls which value in the cell is used to compute the output activation of the LSTM unit (i.e., which value should go to the hidden state). The LSTM gates are embedded in the hidden layer of the DNN of the *Target Network* and are parameterized with weights and biases that can be optimized simultaneously by applying the stochastic gradient descent update rule.

After choosing action a_t, the reward r_t and the next state s_{t+1} can be obtained by the predefined definitions (see line 15 in Algorithm 1). We store transition (s_t, a_t, r_t, s_{t+1}) in replay memory D to update the *Target Network* (see line 16 in Algorithm 1) and a batch of experience from D is randomly selected (see line 17 in Algorithm 1). The target value of y_t^{DDQN} is calculated next. The y_t^{DDQN} is used to calculate the loss of function. The weights of the *Online Network* are updated using a gradient descent-based approach. Every C step the weight of the *Target Network* will be copied based on the *Online Network* weight (line 21 of Algorithm 1).

Algorithm 1: DDQN for application component Migration

```
1  input: threshold ϵ, learning rate α ;
2  initialize: the Online Network Q(s_t, a_t; θ_t) with the weight of θ_t ;
3  initialize: the Target Network Q(s_t, a_t; θ_t) with the weight of θ_t;
4  Initialize: the experience replay memory D ;
5  for each episode do
6      observe the current placement of the VNF and the statuses of Fog/Cloud to determine whether a
         nod is switch on/off based on Eq.(25), and construct s_t;
7      for each time-slot do
8          Choose an action a_t at state s_t using the ϵ -greedy policy;
9          Generate a number φ ∈ [0, 1];
10         if η > ϵ then
11             the action is selected by exploitation select a_t = argmaxQ(s_t, a_t; θ_t);
12         else
13             the action is selected by exploration;
14         end
15         Calculate the immediate reward based on Eq. (26) and observe and next state s_t + 1
             Perform action a_t;
16         Store transition values (s_t, a_t, r_t, s_{t+1}) ;
17         Randomly sample a batch of experience from D ;
18         Calculate the target Q-value based on Eq.(31);
19         Calculate the loss function by L(θ) = E_{s,a}[(y_t^{DDQN} − Q(s_t, a_t; θ_t))^2] ;
20         Run the gradient descent algorithm using Eq. (33);
21         Set θ'_t to θ_t a fixed number of C step
22     end
23 end
```

3.4.4 Complexity Analysis

In this subsection, we present the complexity analysis of our proposed algorithm. The complexity of a deep Q-learning algorithm depends on complexity of action generation and complexity of training. In the proposed algorithm, the *Online Network* and *Target Network* are fully connected networks with G layers, where the input layer is proportional to the size of the states. As explained in Section 3.3.1, we have a total of Z nodes, K VNFs, and I VNF instances. Thus, the dimension of the state space is $Z \cdot K \cdot I$. Let d_g denote the number of neurons in layer g. The number N_ν of the multiplications can then be obtained as follows:

$$N_\nu = (Z \cdot K \cdot I) \cdot d_1 + \sum_{g=2}^{G} d_g \times d_{g-1},$$

Table 3.2: Hyperparameter settings.

Parameter	Value
LSTM layers,number of cells	4,[50,50,50,50]
Activation Function	softmax
Batch size	16
Optimizer	AdamOptimizer
Discount factor	0.9
Hidden layer	64 neurons
Replay memory D	500

where G is the total number of layers in the neural network. Therefore, the computational complexity of action generation at each time slot for one sample would be $\mathcal{O}(N_{\nu})$.

Let N denote the total number training episodes. At each episode, the DDQN loop terminates at the end of each time-slot. Given a total of M time-slots, the complexity of training for one minibatch of N episodes with M time-slots would be $\mathcal{O}(N \cdot M \cdot N_{\nu})$.

3.5 Results and Discussions

Here, we introduce the simulation settings and then show the simulation results. To train our DDQN, we used Python 3.6 to build a TensorFlow 1.8.0 simulation environment [75], which is Google's open-source machine learning library. In particular, we utilized *"tf.contrib.rnn.LSTMCell"* and *"keras.models"* classes to instantiate the two four-layer DNNs, namely *Online Network* and *Target Network*, with LSTM cells in hidden layers of the latter. We set $\gamma = 0.9$ and $\epsilon = 0.9$. To optimize the loss function for the training process (to learn the NN parameters), we used *AdamOptimizer*, an algorithm for the first-order gradient-based optimization of stochastic objective functions [76]. All the simulations were performed on a computer with a 2.67 GHz Intel Xeon CPU E5640 and 32 GB of memory. The results are shown with 95% confidence interval. The parameter settings of DDQN are given in Table 3.2.

3.5.1 Simulation Settings

We have carried out our simulations in Yet Another Fog Simulator (YAFS), which is a Python-based, lightweight, easy-to-configure, discrete event simulator for Fog computing scenarios [77].

The simulation setting of our component migration scenario is explained below.

3.5.1.1 Network Topology

Our network in the simulations consists of 14 mobile end-user devices, 3 cloud nodes, and 6 mobile fog nodes. An illustration of our considered network topology at the first time slot is shown in Fig. 3.4, where both fog nodes and end-users are mobile. All fog/cloud nodes and end-user devices are assumed to be deployed in a square-shaped environment with a dimension of 2×2 km^2. The duration of each time slot is set to 0.05 ms. In order to model the mobility of drone fog nodes and end-users, we have used Gauss-Markov [78, 79, 80, 81] and Random Walk [82, 83, 84] mobility models, respectively. Fog nodes are assumed to fly at a fixed altitude of H = 100 m. The maximum speed of fog nodes (i.e., drones) is set to $v_{max} = 50$ m/s. Other parameters are set based on [85]. Each end-user is assumed to move at a fixed altitude of H = 0 with a random speed distributed between 1 to 19 m/s [86]. The cloud nodes are located on the ground at a distance of 1000 km from the end-users and fog nodes. The capacities of the fog and cloud nodes are set to [2-4] and 8 (measured in number of cores), respectively, while the memory capacities are set to 2 GB for fog nodes and 32 GB for cloud nodes (which are compatible with currently deployed cloud computing platforms, e.g, Amazon AWs m4.2xlarge[2]). End-users communicate with UABS fog nodes with a data rate of 2 Mbps [87]. Fog-to-cloud and Fog-to-fog bandwidth is set to 1 Gbps and 100 Mbps, respectively [88]. Cloud nodes are connected through Gigabit Ethernet with a bandwidth of 10 Gbps [89]. Effective bandwidth between end-users and cloud is set to 1 Mbps. The processing delay per unit of traffic on the cloud and fog nodes is set 0.25 ms and 25 ms, respectively. The propagation delay can be estimated by round-trip time (RTT), which can be expressed as RTT (ms)=0.03$\times$ distance (km)+5 [90]. For cloud nodes, we set P_{max} and P_{idle} to 500 and 250 watts, respectively, and for fog nodes, we set P_{max} = 100 watts and P_{idle} = 50 watts. The average costs of power consumption in the cloud and fog nodes are set to 4 and 2 units of currency, respectively. The bandwidth cost of the links between cloud nodes is set to 0.155 units of currency per GB of transmission. For the links between the fog nodes, the bandwidth cost is set randomly between 0.25 and 2 units per GB of transmission, and for the links between cloud and fog

[2] https://aws.amazon.com/ec2/instance-types/

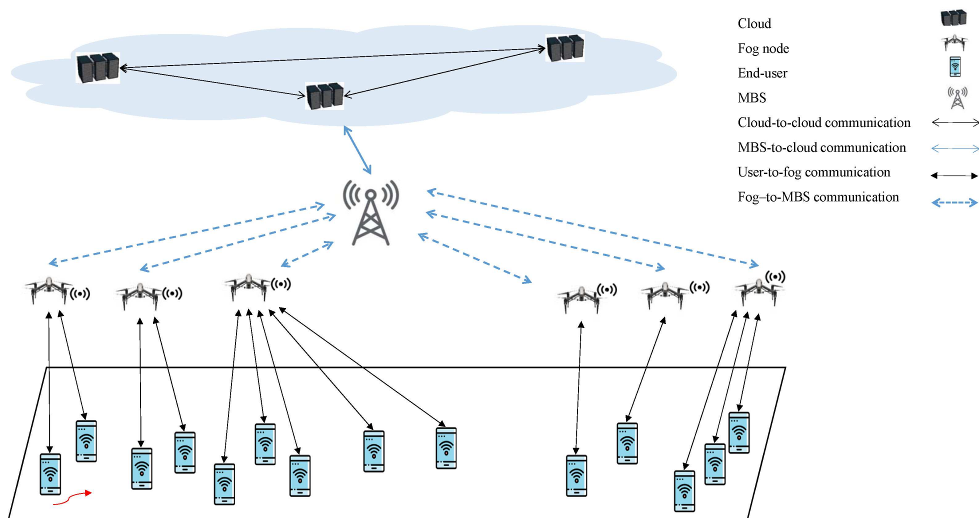


Figure 3.4: Network topology.

nodes, it is set randomly in the range of 10 to 20 units of currency per GB of transmission. We also assume that the traffic originated from end-users is randomly distributed between $[0.01, 2]$ MB. For convenience, we have summarized the simulation parameters and their assigned default values in Table 3.3.

3.5.1.2 VNFs

We consider there are 8 VNFs. For the sake of simplicity, we assume that there is a sequence substructure within each VNF. The resource requirements of VNFs are randomly chosen between [1-3] measured by the number of cores. The memory size of each VNF is set to 1 GB.

3.5.2 Convergence Performance

First, we examine the convergence performance of our proposed DDQN algorithm for the component migration problem (DDQN_CM) against that of the traditional DDQN algorithm with Double RNNs and no LSTM cells [7]. We define the convergence episode as the episode beyond which the objective function improvements lie within the interval $(G^* - \epsilon_0, G^* + \epsilon_0)$, where G^* is the best obtained solution and ϵ_0 is a (small) real number. In our evaluations, we have set $\epsilon_0 = 6 \times 10^-2$.

Table 3.3: Parameter settings and default values.

Parameter	Value
Maximum area (x_{max}, y_{max})	2×2 km^2
Cloud CPU core	8
Fog node CPU core	$[2-4]$
Cloud node memory	32 GB
Fog node memory	8 GB
Processing Delay cloud	0.25 ms/MB
Processing Delay fog nodes	25 ms/MB
Bandwidth between cloud nodes	10 Gbps
Bandwidth between fog nodes and cloud	1 Gbps
Bandwidth between end-user devices and fog nodes	2 Mbps
Bandwidth between end-user devices and cloud nodes	1 Mbps
Power Consumption fog nodes (P_{max}, p_{idle})	(100,50) Watts
Power Consumption cloud (P_{max}, p_{idle})	(500,250) Watts
VNF memory size	1 GB
VNF resource requirement	$[1-3]$
Traffic	$[0.01, 2]$ MB

Table 3.4: Convergence episode and execution time of different algorithms.

Algorithm	**Convergence episode**	**Execution Time**
Proposed DDQN_CM algorithm ($\alpha = 0.1$)	2000	54.85 min
Proposed DDQN_CM algorithm ($\alpha = 0.01$)	7000	185.8 min
Proposed DDQN_CM algorithm ($\alpha = 0.001$)	10000	273.33 min
Traditional DDQN algorithm ($\alpha = 0.001$) [7]	7500	183.75 min

Fig. 3.5 shows the values of the objective function vs. episode for our proposed DDQN_CM algorithm for three different learning rate values of $\alpha = 0.001$, 0.01, and 0.1. We observe from Fig. 3.5 that at the beginning of the learning process, the value of the objective function is high in all curves. This is mainly due to the fact that at the beginning, there is no knowledge about the environment and the agent chooses random actions. As the number of episodes increases, all curves tend to converge to a lower objective value. However, the convergence of DDQN_CM is faster for α =0.01 and 0.1 compared to that of $\alpha = 0.001$. While a small learning rate may lead to slow learning speed due to the need for more steps to obtain the global optimum, a larger learning rate require fewer training steps, though it may get stuck in a local optimum instead of finding the global optimum

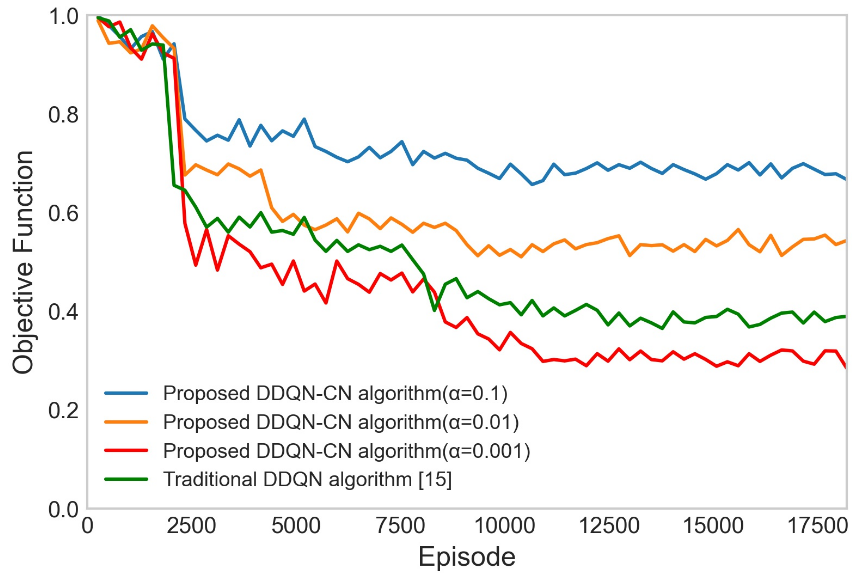


Figure 3.5: Convergence performance of our proposed DDQN-CM algorithm for different values of learning rate α against the traditional DDQN algorithm ($\alpha = 0.001$) [7].

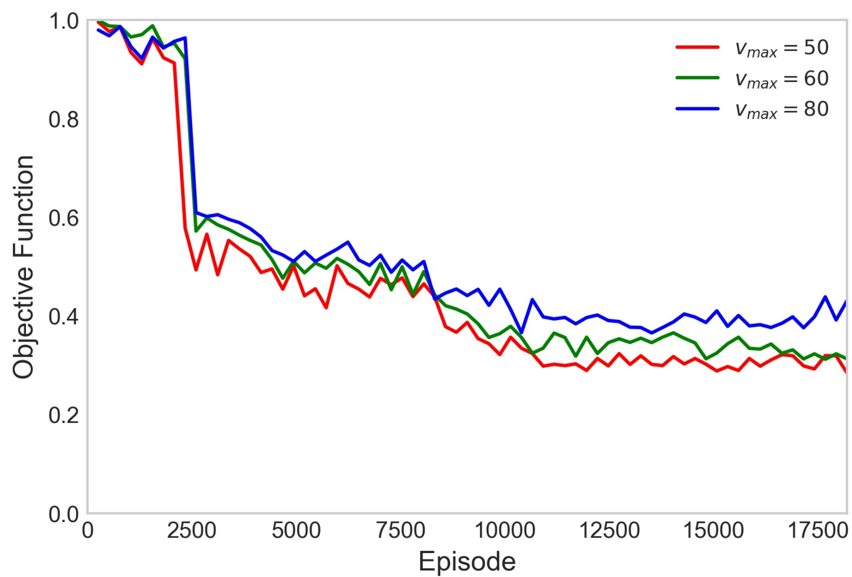
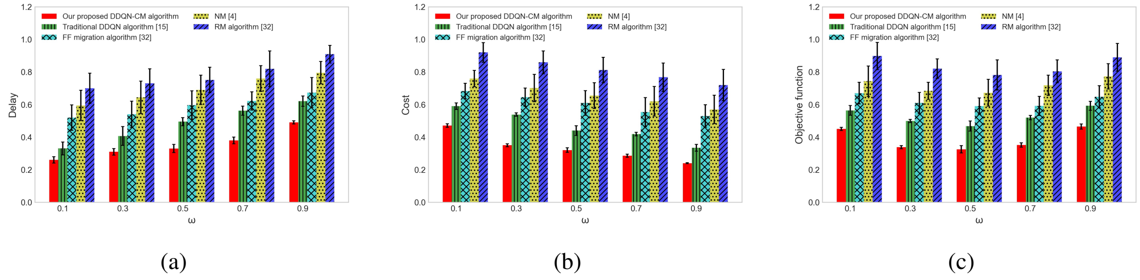


Figure 3.6: Objective function vs. episode for different values of fog node speed v_{max}.

(see converged objective values in Fig. 3.5). Figure 3.5 indicates that our proposed DDQN_CM algorithm and traditional DDQN algorithm (both run for $\alpha = 0.001$) perform closely for the first



(a) (b) (c)

Figure 3.7: (a) Delay, (b) cost, and (c) objective function vs. weight ω for different algorithms.

episodes, when there is not any knowledge about the environment. The proposed DDQN_CM algorithm converges to a smaller objective value compared to the traditional DDQN algorithm. This is mainly because our proposed DDQN_CM relies on using LSTM memory cells, which allow for remembering the most valuable experiences in the past observations, thus leading to better objective values. We note, however, that such superior performance is achieved at the expense of a slightly more convergence time, to be examined next.

Table 3.4 summarizes the convergence episode and execution time of different algorithms. According to Table 3.4, the proposed DDQN_CM algorithm converges after 2000, 7000, and 10,000 episodes for $\alpha = 0.1$, 0.01, and 0.001, respectively. Given that each episode corresponds to 1.69 s, the convergence time of our proposed DDQN_CM algorithm is 3291 s, 11148 s, and 16400 s for $\alpha = 0.1$, 0.01, and 0.001, respectively. On the other hand, the traditional DDQN converges after 7500 episodes and each episode takes 1.47 s, which translates into a total convergence time of 184 min. This happens because the proposed DDQN_CM deploys LSTM layers with 50 units and it requires to pass the input through the *Target Network*. Using Fig. 3.5 and Table 3.4, we make a suitable trade-off between performance and convergence speed by setting the learning rate α to 0.001, which offers the best performance with an acceptable execution time.

Next, we evaluate the convergence performance of our proposed method for different values of fog speed v_{max} in Fig. 3.6, where we observe that the proposed algorithm can converge under different fog node speeds. Further, we observe from the figure that the converged values of the objective function for different values of fog node speed are very close to each other, which indicates the adaptability of the proposed algorithm to different settings. To be more specific, we observe from

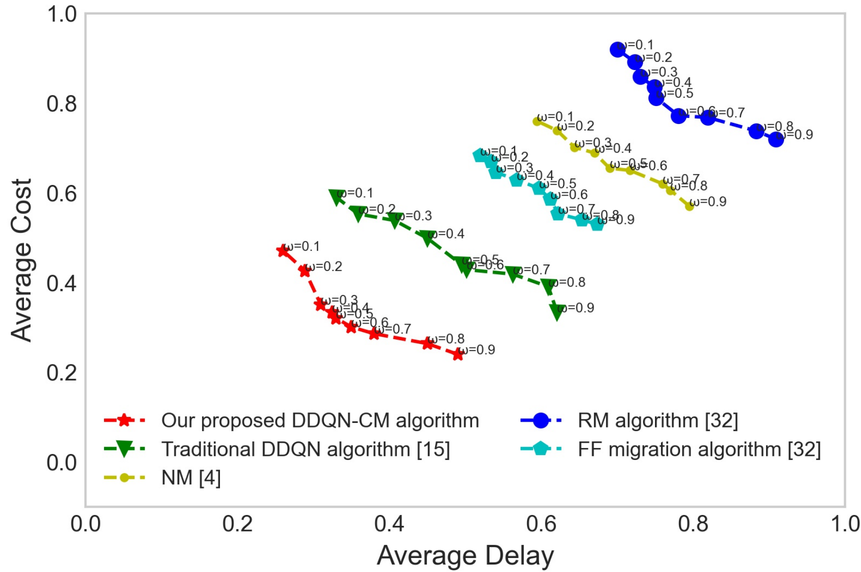


Figure 3.8: Average cost vs. average delay for different values of weight ω.

Fig. 3.6 that the performance of the proposed algorithm is slightly better for smaller values of the fog node speed. The reason is that the higher the average speed of the fog nodes is, the greater the fluctuations of network topology becomes. Thus, the fog nodes, on which application components are placed on, become far from end-user devices more often, thus requiring more migrations to be carried out. On the other hand, if the migration delay and the processing delay of fog nodes are sufficiently small, increasing the average fog node speed may decrease the delay, as the number of opportunities for migration increases. Moreover, we observe from Figs 3.5 and 3.6 that the objective function has some performance variations for increasing episodes. This is mainly due to the exploration, which is essential to continue progressively in order to migrate VNFs based on the most recent location of fog nodes and end-users, which keeps changing at each time slot.

3.5.3 Simulation Results

We compare the performance of our proposed DDQN_CM algorithm with four benchmarks, namely, traditional Double Q-learning (DDQN) [7], Random Migration (RM) [1], [59], Never Migration (NM) [1], and First Fit (FF) migration [59]. In the RM scheme, the components are always

migrated to another node randomly, while satisfying the given capacity constraint. In contrast, the FF migration scheme migrates components to the closest node while considering the given constraints (e.g., bandwidth, computational resource capacity).

We recall from Section 3.3 that our objective is to minimize the weighted sum of cost and delay, where the weights allow for making a desired trade-off between cost and delay. The cost-delay performance of our proposed DDQN_CM algorithm, along with those of our benchmarks, is shown in Figs. 3.7a-c for different values of ω. Figure 3.7.a indicates that as ω increases, the advantage of the delay in DDQN_CM increases, which is reasonable since the larger the ω, the greater the influence on the delay. More specifically, we observe from Fig. 3.7.a that our proposed DDQN_CM algorithm can achieve up to a 56% of improvement of average delay over other algorithms. The average cost obtained by different algorithms is illustrated in Fig. 3.7.b, which shows that our proposed DDQN_CM algorithm achieves the smallest average cost compared to all four other methods. According to Fig. 3.7.b, our proposed DDQN_CM algorithm achieves up to a 61% of improvement of average cost over other algorithms. Similarly, the average cost of the DDQN_CM algorithm is better compared to the other methods. Figure 3.7.c depicts the performance of the five different algorithms in terms of the obtained objective function for different values of ω. We observe that our proposed DDQN_CM algorithm outperforms all the other methods.

Figure 3.8 illustrates the average cost vs. the average delay for different values of weight ω. This information allows the decision makers to make suitable delay-cost trade-offs by tuning the value of weight ω. Importantly, we observe from Fig. 3.8 that as we decrease the value of ω, the average cost is further prioritized over the average delay (and vice versa). Moreover, Fig. 3.8 indicates that the cost and delay values of our proposed DDQN_CM are smaller than those of all four other methods. For our further simulation, we set $\omega = 0.5$. According to Fig. 3.8, by increasing ω from 0.1 to 0.9, our proposed DDQN_CM algorithm achieves a cost reduction of 50%, which is achieved at the expense of a 47% increase of the average delay.

Next, we examine the adaptability of our proposed method to an increasing traffic. Figure 3.9 depicts the obtained objective function vs. number of end-users ranging from 2 to 26. In general, we observe that the objective function increases as the number of users increases. This is because for smaller number of users, a smaller amount of traffic needs to be transmitted and processed by

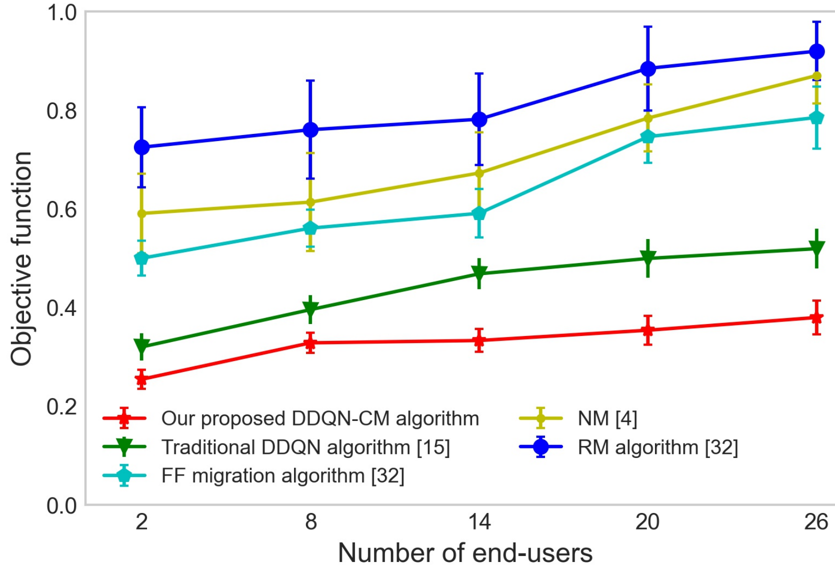


Figure 3.9: Objective function of different algorithms when varying the number end-users communicating with the VNFs.

the nodes, thus leading to small processing and transmission delays. Importantly, we observe from Fig. 3.9 that the difference between our proposed method and the other methods becomes more apparent for large number of end-user devices. This is because our proposed DDQN-CM algorithm avoids excessive migration of application components thus reducing the objective function. Also, our proposed DDQN-CM method achieves a considerably better result compared to the traditional DQQN approach. The reason is that our proposed DDQN-CM method is empowered by an LSTM memory cell, which allows the agent to remember the most valuable experience that it has had in its past observation. More importantly, we observe from Fig. 3.9 that the upward trend of the objective function for increasing number of end-users is considerably less compared to other approaches. To see this, it is interesting to note that the obtained value of the objective function of our proposed DDQN-CM algorithm for 26 end-user devices is still smaller than that of FF, NM, and RM methods for 2 end-user devices (see Fig. 3.9). This highlights the fact that our proposed DDQN-CM algorithm is well capable of adapting to more complex scenarios and maintaining its performance for a wide range of number of end-users.

Figure 3.10 shows the average number of migrations per time slot. vs. the number of end-users

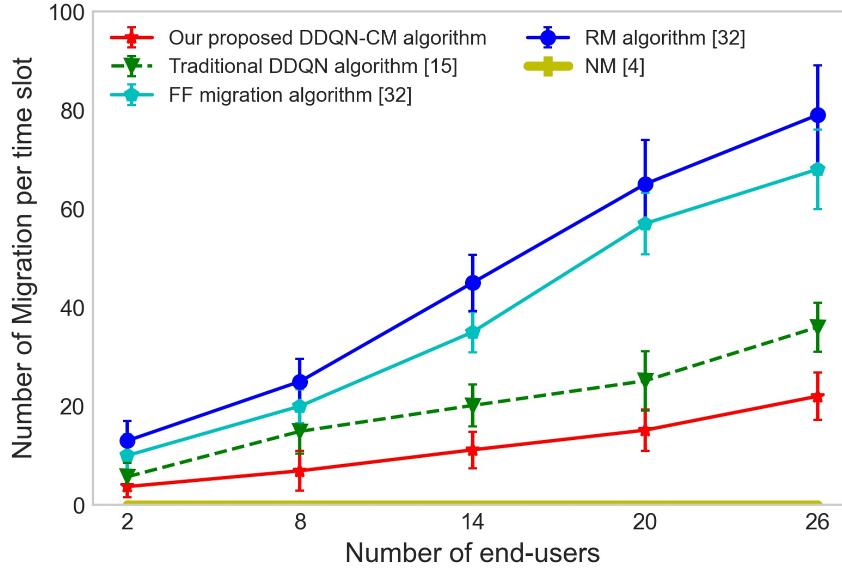


Figure 3.10: Average number of migrations per time slot with respect to the number of end-users.

ranging from 2 to 26. We observe that as the number of end-users increases, a larger number of VNF instances will likely become further from the end-users due to their movement. Therefore, a larger number of migrations will be needed to move the VNF instances closer to the hosted node in order to reduce the delay and the cost (see Fig. 3.10). Figure 3.10 shows that the improvement made by our proposed DDQN_CM in terms of the number of migrations is 72% in comparison with other algorithms. Finally, we evaluate the impact of the number of VNFs on the power consumption performance. Figure 3.11 depicts the power consumption vs. the number of VNFs in a VNF-FG, ranging from 4 to 14 VNFs. We observe in the figure that as we increase the number of VNFs in VNF-FG, the power consumption of all methods under consideration increases. This happens because not only a larger number of substrate nodes need to be switched on to host the VNFs, but also hosting a larger number of VNFs per substrate node increases its CPU utilization, which in turn increases the overall power consumption (see Eq. 3.10). According to Fig. 3.11, our proposed DDQN_CM algorithm outperforms other benchmarks in terms of power consumption, as they require switching on a large number of nodes to host newly migrated VNFs.

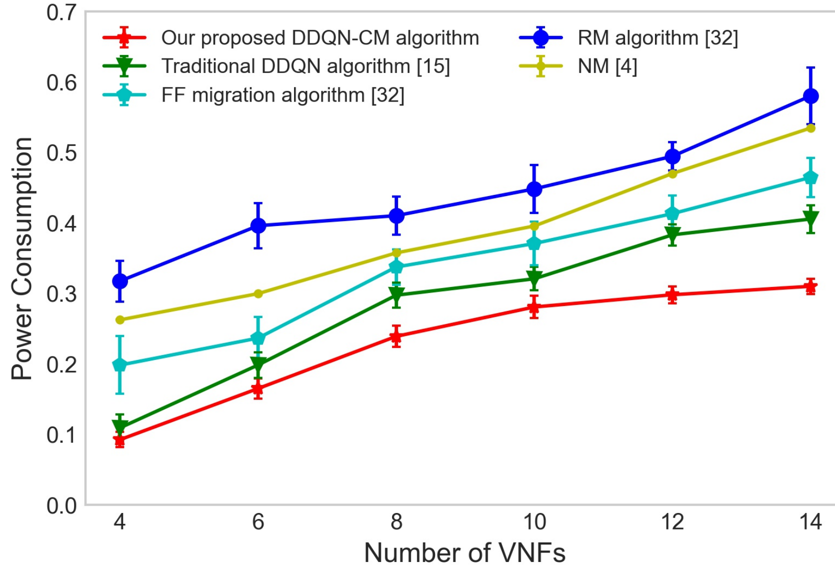


Figure 3.11: Power consumption when varying the number of VNFs.

3.6 Conclusions

In this chapter, we studied the problem of application component migration in an NFV-based hybrid cloud/fog environment. Application components were implemented as VNFs. We considered the mobility of fog nodes as well as the mobility of end-user devices. We aggregated the cost and delay by traversing a structured tree built over the input VNF-FG. The cost function includes power consumption and migration costs, while the delay includes processing, communication, and migration delay. We then adopted an MDP framework and proposed a deep Q-learning approach. We implemented our DRL migration agent with a deep double-Q learning method empowered by LSTM memory cells. The simulation results revealed that our method achieves a good performance in the aggregation of cost and delay compared to the existing methods. Furthermore, we analyzed the effect of communicating with end-user devices on the number of migrations. We showed that our proposed DDQN_CM algorithm achieves up to 61% and 56% of improvement of cost and delay, respectively.

Chapter 4

Cost-efficient Cluster Migration of VNFs for VNF Forwarding Graph Embedding[3]

4.1 Introduction

NFV enables the dynamic deployment and migration of VNF-FG over NFVI [91]. Where and how VNFs are migrated may have a significant impact on the total embedding cost. VNF migration can be a potential solution to ensure the survivability of VNF-FG against failures of physical nodes [92]. However, stringent latency requirements between VNFs of a given VNF-FG can make those VNFs coupled to each other. This may limit the simple VNF migration strategy since this strategy will only relocate the VNFs that are not coupled to their neighbor VNFs with stringent latency constraints. Moreover, in the simple VNF migration, an additional transmission cost may be incurred, which can increase the resulting embedding cost. Cluster VNF migration can potentially reduce the embedding cost by migrating a group of coupled VNFs within a single physical node or across multiple physical nodes. Ref. [93] studied live migration of virtual machines on multitier applications and showed that if coupled/dependent components become split across a high-latency

[3]This chapter is based on a published paper: [3]S. N. Afrasiabi et al., "Cost-efficient Cluster Migration of VNFs for Service Function Chain Embedding," in IEEE Transactions on Network and Service Management.

network path, the network performance can severely degrade. However, a physical node may not support all types of VNFs, and it has limited resources. Furthermore, the cost of VNF instantiation may vary from one node to another. Thus, migrating a cluster of VNFs to a single physical node would not always be a reasonable option.

To address the above-mentioned issues, in this chapter, we formulate the VNF migration problem as an ILP. In our cost modeling, we consider a wide variety of parameters, including computing resources, VNF instantiation, reuse, and transmission cost. With the objective of minimizing the total embedding cost, this chapter deals with the problem of migrating a cluster of VNFs to single and multiple destination nodes, and develops a novel VNF cluster migration algorithm called Single-/Multi-Destination Cluster VNF Migration. Our proposed Single-Destination Cluster VNF Migration (sDCM) algorithm migrates all the VNFs within a cluster to a single node, whereas the Multi-Destination Cluster VNF Migration (mDCM) algorithm migrates a cluster of VNFs to various nodes. We evaluate our proposed algorithm via extensive simulations. Specifically, we compare the performance of our proposed algorithm with those of the brute-force approach and existing sequential greedy embedding [8] and h-HSLG [9] algorithms. The simulation results indicate that our proposed algorithms outperform the existing benchmarks in terms of embedding cost, while having much shorter execution time compared to the brute-force approach.

The rest of this chapter is organized as follows. The motivating scenario is presented in Section 4.2. We present the system model and problem formulation in Section 4.3. The proposed VNF cluster migration strategies are introduced in Section 4.4. The numerical results are shown in Section 4.5. Finally, Section 4.6 concludes this chapter.

4.2 Motivating Scenario

In this section, we present an illustrative example shown in Fig. 4.1 to further explain the motivation behind using cluster migration. Let us consider two Network Services (NSs), namely NS 1 and NS 2 with an ordered set of VNFs $\{A, B, C\}$ and $\{A, B, C, D\}$, respectively. We assume that NS 2 arrives after NS 1. Inter-VNF latency requirements are shown in Fig. 4.1. For instance, the latency requirement between VNFs A_1 and B_1 of NS 1 is 2 ms, meaning that the transmission delay

of the physical link(s) that realize the connection between these VNFs should not exceed 2 ms. The VNF-FG associated with NS 1 requires a traffic rate of 1 unit to be transmitted from Node 1 as the source node and then processed by the VNF-FG (which is given in the form of an ordered set of VNFs) before reaching Node 4 as the destination node. In addition, each VNF type supported in the network is associated with a predefined resource requirement, measured by the number of CPUs, memory, and/or storage. The network topology consists of 4 physical nodes and 3 physical links, as shown in Fig. 4.1. Each physical node is associated with a node capacity (in terms of CPU, memory, and storage), cost per resource unit, and a set of supported VNF types. Each VNF also has a fixed instantiation cost along with a reuse cost on a given node. A new deployment of a VNF instance on a given physical node is subject to an instantiation cost, whereas the reuse of an already deployed instance is subject to a reuse cost. For simplicity and without loss of generality, we assume that the instantiation and reuse costs of all VNF types on a given physical node are \$4 and \$0, respectively, and all VNFs type require 1 unit of resource. Similarly, each link is associated with a bandwidth capacity, cost per bandwidth unit, and link latency. The attributes of each physical node and link are shown in Fig. 4.1. For instance, Node 1 has 10 units of node capacity, and a cost of \$7 per resource unit. The link between Nodes 1 and 2 has a cost of \$5 per bandwidth unit, and link latency of 1 ms.

Figure 4.1.a depicts the embedding outcome returned by the Sequential Greedy (SG) embedding algorithm [8]. The SG algorithm places each VNF one by one in a sequential manner by making the best local decisions. The total embedding cost of NS 1 in SG algorithm is \$44 (shown in Fig. 4.1.a), as the transmission cost from Node 1 to Node 4 is \$11 and the cost of embedding of each VNF of NS 1 in Node 1 is \$11. This is because the embedding cost of VNF A_1 is the summation of the resource cost in Node 1, which is \$7, and the instantiation cost \$4. The h-HSLG algorithm proposed in [9] aims to overcome the poor embedding outcome of [8], which suffers from the so-called causality issue. The causality issues refers to the fact that the optimal embedding decision of a VNF cannot be known until the embedding of its predecessor and successor VNFs are known. The h-HSLG algorithm proposed in [9] deals with the causality issue by allowing the VNFs to be revisited within a given exploration window in order to improve their embedding via simple VNF migration. In this way, the h-HSLG algorithm aims to reduce the embedding cost by revisiting each VNF and migrating it while taking into account the embedding decision of its neighbor VNFs. For

example, as shown in Fig. 4.1.b, VNF C_1 is migrated to Node 4 after the link cost between the destination node and VNF C_1 is known. The outcome of the simple migration of VNF C_1 from Node 1 to Node 4 for NS 1 is shown in Fig. 4.1.b. By migrating C_1 to Node 4, the total embedding cost for NS 1 would be \$41. This is because the transmission cost is \$11, the resource cost for A_1 and B_1 in Node 1 is \$14, the resource cost for C_1 in Node 4 is \$4, and the instantiation cost on Nodes 1 and 4 is $3 \times \$4$. We note, however, that even though Node 4 has a smaller resource cost, it is not feasible to migrate VNF B_1 to Node 4, mainly because the latency constraint between VNFs A_1 and B_1 would be violated. The latency requirement between VNF A_1 and VNF B_1 is 2 ms, and the total latency of the path from Node 1 to Node 4 is 6 ms. Unlike the simple VNF migration, which aims to migrate VNFs one by one, we can allow the creation of a cluster of VNFs to be migrated. More specifically, in the example under consideration, VNFs A_1, B_1, and C_1 may constitute a cluster of VNFs to be migrated to Node 4 without having any latency constraint violation issue; as seen Fig. 4.1.c. This can improve the quality of the embedding solution. The total embedding cost of NS 1 after migrating a cluster of VNFs would be \$35, which is the summation of the transmission cost \$11, the resource cost \$12, and instantiation cost \$12.

The example above shows how the migration of a cluster of VNFs can improve the cost of the simple migration-only approach. In the following, we show that the embedding cost can be further reduced via the complex migration scheme. In addition to NS 1, let us consider another NS 2 with an ordered set of VNFs A_2, B_2, C_2, and D_2 (see Fig. 4.1). The latency constraints between VNFs are also shown in Fig. 4.1. We assume that the instantiation cost of all VNFs is \$4, whereas the reuse cost is set to \$0. Figure 4.1.d shows the embedding solution obtained by the SG algorithm for NS 2 [8]. The total cost of embedding NS 1 and NS 2 is $\$35 + \$55 = \$90$, as the embedding cost of NS 2 is $\$11 + \$28 + \$4 \times 4 = \$55$. Clearly, the embedding cost of all VNFs at Node 4 is smaller than at other nodes, as the resource cost in this node is smaller. Moreover, VNF types A, B, and C have already been deployed on Node 4, which has the capacity of 6 units, meaning that it does not have enough capacity to host all the VNFs of NS 2. In addition, even though the embedding cost at Node 3 is smaller than at Node 1, it would not be possible to migrate VNF D_2 to Node 3 via a simple migration, mainly because the given latency constraint would be violated. Using a cluster migration, all the VNFs of NS 2 can be migrated to Node 3, as shown in Fig. 4.1.e,

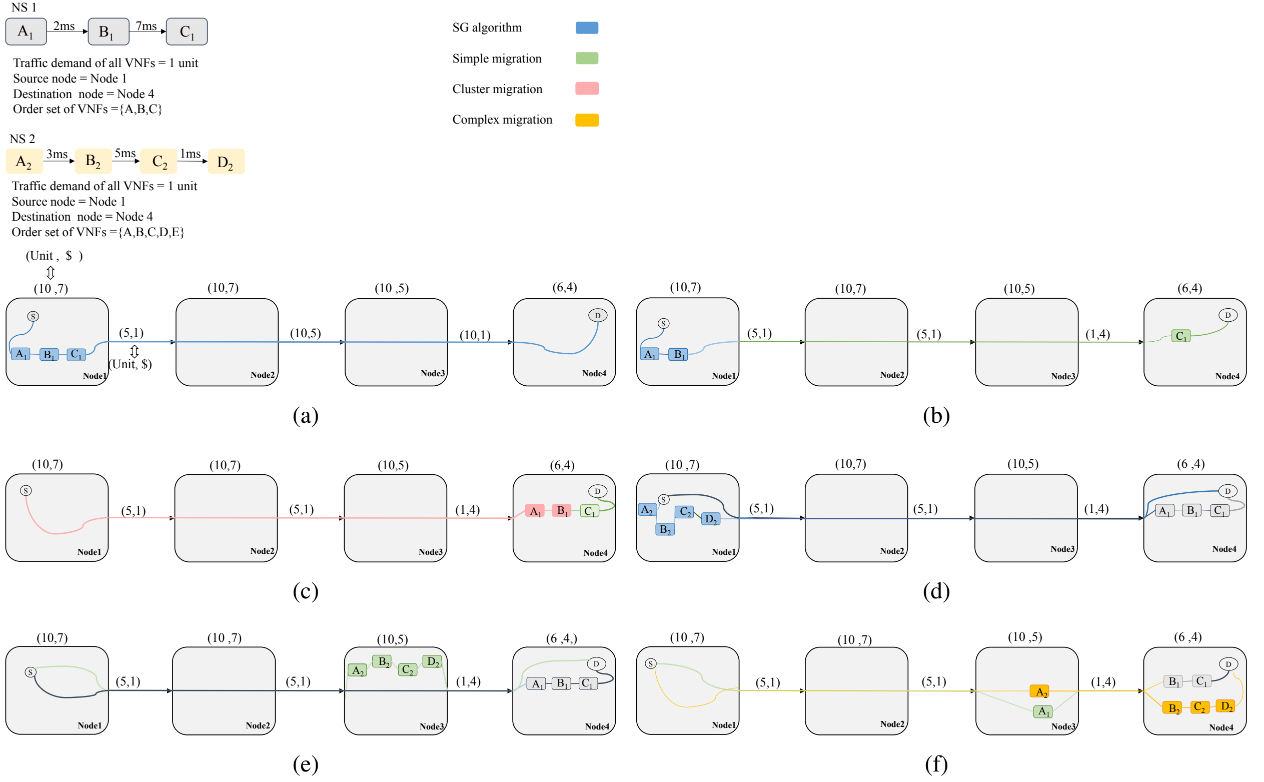


Figure 4.1: Illustrations of: (a) embedding solution for NS 1 obtained by the SG algorithm [8], (b) simple VNF migration for NS 1 [9], (c) cluster VNF migration for NS 1, (d) solution for NS 2 obtained by the SG algorithm, (e) cluster VNF migration for NS 2, (f) complex VNF migration.

where we observe that the total embedding cost for the NS 1 and NS 2 is \$35 + \$47 = \$82, as the embedding cost of NS2 is obtained as \$11 + \$16 + \$20 = \$47.

Using the complex migration scheme, VNF A_1 is evicted from Node 4 and moved to Node 3, and at the same time, VNFs B_2, C_2, and D_2 are evicted from Node 3 and moved to Node 4. The total embedding cost in this case would be \$68. By assuming NS 1 arrives as a first request the transmission cost would be \$11, the resource cost of VNF A_1 in Node 3 is \$5 and the instantiation cost is \$4. The total resource and instantiation costs for B_1, C_1 in Node 4 are \$8 and \$8 respectively (the total embedding cost of NS 1 is \$36). Furthermore, the embedding cost of NS 2 would be \$32. The transmission cost is \$11, A_2 is reused in Node 3 which is encounter to \$0 reused and \$5 resource costs. Consequently, the total resource cost for B_2, C_2, and D_2 in Node 4 \$12 and the total reused cost for B_2 and C_2 \$0 and the instantiation cost of D_2 is \$4. Therefore, the embedding cost can be reduced by \$35 and \$27 compared to the simple and cluster VNF migration strategies,

respectively.

4.3 System Model and Problem Formulation

In this section, we describe the system model and problem formulation.

4.3.1 System Model

We introduce the main terminology used to represent the physical network and VNF here. A summary of the notation used in this chapter is provided in Table 4.1.

Table 4.1: Summary of main notations.

Input Parameters	
E	Set of physical links
K	Set of VNF indices in a given NS
L	Set of virtual links in a given NS
R	Set of requests in network
Z	Set of NFV-enabled nodes
I_k	Set of instances of VNF type k in the network
K_z	Set of supported VNF types of physical node z
c_z	Available node capacity of physical node z
ϕ_z	Node cost per resource unit of physical node z Cost of each unit of node resource
b_e	Available link bandwidth of physical link e
d_e	Link delay of physical link e
τ_e	Link cost per bandwidth of physical link e cost of each unit of link resource
p_k	Processing capacity of VNF type k
u_k	Resource requirement for instantiation of VNF type k
a^k	Processing demand of VNF type k in the VNF-FG of request r
$m_{i,k}$	Processing load assigned to instance i of VNF type k
η_k^z	Instantiation cost of VNF type k on physical node z
$\Theta_{i,k}^z$	Reuse cost of instance i of VNF type k on physical node z
d_l	Latency requirement of virtual link l
o_l	Bandwidth requirement of virtual link l
w_k^z	Binary parameter, indicating whether or not physical node z can support VNF type k
$x_z^{i,k}(t)$	Binary variable, indicating whether or not instance i of VNF type k is a newly deployed instance on node z in snapshot t
$\tilde{x}_z^{i,k}(t)$	Binary variable, indicating whether or not instance i of VNF type k is an already deployed instance on node z in snapshot t
$y_e^l(t)$	Binary variable, indicating whether or not virtual link l is mapped onto physical link e in snapshot t

We denote the physical network as a directed graph $G = (Z, E)$, where $|Z|$ physical nodes and $|E|$ physical links are indexed by $z \in \{1, 2, \ldots, |Z|\}$ and $e \in \{1, 2, \ldots, |E|\}$, respectively. Each physical node $z \in Z$ has a maximum computing resource capacity c_z (which includes memory,

CPU, and storage) and cost ϕ_z per resource unit. Each physical node z may support multiple VNF types $K_z \subseteq K_T$, where K_z is the set of VNF types that are supported by physical node z and K_T is the set of all VNF types that are supported in the network. Let us define w_k^z as a binary parameter that indicates whether physical node z can support VNF type $k \in K_T$. Moreover, each physical link $e \in E$ is represented by $\{s(e), d(e), b_e, \tau_e, d_e\}$, where $s(e)$ and $d(e)$ are the source and destination nodes of link e, b_e is the available bandwidth capacity, τ_e is the link cost per bandwidth unit, and d_e is the link delay of physical link e.

Each request $r \in R$ is represented by set K indexed by $k \in \{1, 2, \ldots, |K|\}$, $K \subseteq K_T$, which describes the network service as a chain of $|K|$ VNFs. Each VNF type $k \in K$ has a processing demand a^k. We note that each VNF type k is associated with a predefined resource requirement u_k (measured by the number of CPU, memory, and/or storage units) and a processing capacity p_k (measured by traffic per time unit). Each VNF type k is subject to an instantiation cost η_k^z (which includes software and/or license cost) to create a new VNF instance of type k on physical node z. Let the set of instances of VNF type k is denoted by I_k, which is indexed by $i \in \{1, 2, \ldots, |I_k|\}$, where $|I_k|$ is the maximum number of instances of VNF type k that is allowed to be instantiated across the network. The VNFs in request r can either make use of existing VNF instances in the substrate nodes or create new instances therein [94]. Each instance $i \in I_k$ of VNF k is associated with a reuse cost $\Theta_{i,k}^z$ on physical node z. Typically, the reuse cost of an already deployed VNF is smaller than the instantiation cost of a new VNF instance. If VNF k is going to be instantiated on node z, it is subject to instantiation cost η_k^z. On the other hand, if instance i of VNF k is already deployed on node z, it is subject to reuse cost $\Theta_{i,k}^z$. Furthermore, we define $m_{i,k}$ as the amount of processing load that is already assigned to instance i of VNF type k. The set L of virtual links in the given VNF-FG request is indexed by $l \in \{1, 2, \ldots |L|\}$, where virtual link $l \in L$ connects a pair of VNFs. Each virtual link l is associated with latency, and bandwidth requirements denoted by d_l and o_l, respectively. We define a snapshot t as a representation of the system state over a fixed time interval. Note that our model operates over two network snapshots: the current snapshot $t-1$ (before migration of a cluster of VNFs) and the new one (after migration of a cluster of VNFs) at snapshot t. Migrations can be carried out between snapshots $t-1$ and t. Therefore, the number of newly deployed and reused VNFs on a specific node for a request may vary from one snapshot

to another [95]. In the following, we define the binary decision variables used in our problem formulation.

- $x_z^{i,k}(t) \in \{0, 1\}$: It specifies new VNF deployment. If $x_z^{i,k}(t)$ is equal to 1, the VNF instance $i \in I_k$ of VNF type $k \in K$ is newly deployed on physical node z on snapshot t; otherwise, it is 0.

- $\tilde{x}_z^{i,k}(t) \in \{0, 1\}$: It specifies whether or not a VNF is reused, meaning that if $\tilde{x}_z^{i,k}(t)$ is equal to 1, the instance $i \in I_k$ of VNF type $k \in K$ is an already deployed instance on physical node z in snapshot t; otherwise, it is 0, meaning that it is a newly deployed instance or not deployed at all.

- $y_e^l(t) \in \{0, 1\}$: It specifies new link assignment. If $y_e^l(t)$ is equal to 1, the virtual link $l \in L$ is mapped onto physical link $e \in E$ of a request r on snapshot t; otherwise, it is 0.

4.3.2 Problem Formulation

This contribution aims to minimize the total embedding cost, which comprises the cost of computing resource, VNF instantiation, VNF reuse, and transmission. We formulate the cluster migration problem as an ILP, to be explained in technical detail next.

4.3.2.1 Computing Resource Cost

Computing resource cost $C_{cr}(t)$ is the cost of resources assigned to VNFs in snapshot t [95]. Some resources may be released during the migration from snapshot $t-1$ to t, which leads to a cost reduction. Thus, we need to consider the difference in resource consumption of a physical node before and after migration. As an example, during the snapshot of $t-1$, if instance i of VNF K has 2 units of resource consumption and is hosted by node z_1, by having migration from node z_1 to node z_2, we need to consider the released resources of node z_1. Furthermore, if by switching from snapshot $t-1$ to snapshot t, no migration occurs, $C_{cr}(t)$ in snapshot t would become 0. As for a particular node z, instance i of VNF k, $x_z^{i,k}(t-1)$ and $x_z^{i,k}(t)$ are equal to 1. With all these

considerations, computing resource cost $C_{cr}(t)$ is given by:

$$C_{cr}(t) = \sum_{z \in Z} \sum_{k \in K_r} \sum_{i \in I_r} u_k \cdot \phi_z \cdot ([x_z^{i,k}(t) - x_z^{i,k}(t-1)]^+ + [\tilde{x}_z^{i,k}(t) - \tilde{x}_z^{i,k}(t-1)]^+), \tag{4.1}$$

where the term $[x_z^{i,k}(t) - x_z^{i,k}(t-1)]^+ = \max\{x_z^{i,k}(t) - x_z^{i,k}(t-1), 0\}$, which is used to prevent the cost from becoming negative. For example, let us assume that in snapshot $t-1$, instance i of VNF k is hosted in node z_1, meaning that $x_{z_1}^{i,k}(t-1) = 1$. If this instance is migrated to a new node in snapshot t, $x_{z_1}^{i,k}(t)$ would become 0 and therefore $[x_z^{i,k}(t) - x_z^{i,k}(t-1)]^+ = 1$.

4.3.2.2 VNF Instantiation Cost

Instantiation cost $C_{in}(t)$ includes the cost associated with the network function software licenses for installing a new VNF instantiation [96], which is given by:

$$C_{in}(t) = \sum_{z \in Z} \sum_{k \in K} \sum_{i \in I_k} \eta_k^z [x_z^{i,k}(t) - x_z^{i,k}(t-1)]^+, \tag{4.2}$$

It should be noted that by having migration from a snapshot $t-1$ to t, some VNFs need to be newly instantiated in a node. As such, $[x_z^{i,k}(t) - x_z^{i,k}(t-1)]^+$ calculates the number of new VNF instantiations of VNF k on node z in snapshot t, which is obtained similarly to Eq. (4.1).

4.3.2.3 VNF Reuse Cost

The already-deployed VNFs may be reused to reduce the embedding cost. We compute the reuse cost $C_{ru}(t)$ of such VNFs from snapshot $t-1$ to snapshot t, as follows:

$$C_{ru}(t) = \sum_{z \in Z} \sum_{k \in K} \sum_{i \in I_k} \Theta_k^z [\tilde{x}_z^{i,k}(t) - \tilde{x}_z^{i,k}(t-1)]^+. \tag{4.3}$$

where the terms $[\tilde{x}_z^{i,k}(t) - \tilde{x}_z^{i,k}(t-1)]^+ = \max\{\tilde{x}_z^{i,k}(t) - \tilde{x}_z^{i,k}(t-1), 0\}$.

4.3.2.4 Transmission Cost

The transmission cost $C_{tr}(t)$ includes the cost of network bandwidth consumed for the communication between VNF instances, as follows:

$$C_{tr}(t) = \sum_{e \in E} \sum_{l \in L} \tau_e o_l [y_e^l(t) - y_e^l(t-1)]^+. \tag{4.4}$$

which considers the differential cost of the assigned link that may happen between two consecutive snapshots as a result of the migration of VNFs.

The objective of our optimization problem is to minimize the sum of the aforementioned costs, as follows:

$$C_{total} = \min(C_{cr}(t) + C_{in}(t) + C_{ru}(t) + C_{tr}(t)). \tag{4.5}$$

In the following, we present the constraints of the problem to be considered. Constraint (4.6) ensures that instance i of VNF type k can be deployed in physical node z, only if node z can support VNF type k:

$$x_z^{i,k}(t) \leq w_k^z, \quad \forall k \in K, \quad \forall i \in I_k, \quad \forall z \in Z. \tag{4.6}$$

Constraint (4.7) ensures that each VNF instance i of VNF type k is not deployed more than once in the network:

$$\sum_{z \in Z} x_z^{i,k}(t) \leq 1, \quad \forall k \in K, i \in I_k. \tag{4.7}$$

Constraint (4.8) ensures that instance i of VNF type k can either be reused or instantiated in physical node z:

$$x_z^{i,k}(t) \geq \tilde{x}_z^{i,k}(t), \quad \forall k \in K, \quad \forall i \in I_k, \quad \forall z \in Z. \tag{4.8}$$

Constraint (4.9) ensures that the capacity of each physical node z is not exceeded:

$$\sum_{k \in K} \sum_{i \in I} u_k(x_z^{i,k}(t) + \tilde{x}_z^{i,k}(t)) \leq c_z, \quad \forall z \in Z. \tag{4.9}$$

Constraint (4.10) ensures that the processing load assigned to instance i of VNF type k does not exceed its available processing capacity:

$$\sum_{z \in Z} a^k x_z^{i,k}(t) \leq p_k - m_{i,k}, \quad \forall k \in K \quad \forall i \in I_k. \tag{4.10}$$

Constraint (4.11) ensures that the bandwidth capacity of each physical link e is not exceeded:

$$\sum_{l \in L} o_l y_e^l(t) \leq b_e, \quad \forall e \in E. \tag{4.11}$$

Constraint (4.12) ensures that the delay requirement of each virtual link is met:

$$\sum_{e \in E} d_e y_e^l(t) \leq d_l, \quad \forall l \in L. \tag{4.12}$$

Constraint (4.13) ensures that if two consecutive VNFs are placed on different physical nodes, at least a physical link is assigned between them:

$$x_z^{i,k}(t) \cdot x_{z'}^{j,k+1}(t) \leq y_e^l(t), \quad \forall z, z' \in Z, \quad \forall k \in K, \quad \forall l \in L, z \neq z'. \tag{4.13}$$

Clearly, Constraint 4.13 is a non-linear equation, as it involves the multiplication of two decision

variables. Thus, we linearize it as follows:

$$Q_e^l(t) = x_z^{i,k}(t) \cdot x_{z'}^{j,k+1} \tag{4.14a}$$

$$Q_e^l(t) \leq y_e^l(t) \tag{4.14b}$$

$$Q_e^l(t) \leq x_z^{i,k}(t) \tag{4.14c}$$

$$Q_e^l(t) \leq x_{z'}^{j,k+1} \tag{4.14d}$$

$$Q_e^l(t) \leq x_z^{i,k}(t) + x_{z'}^{j,k+1} - 1 \tag{4.14e}$$

4.4 Proposed Single-/Multi-Destination Cluster VNF Migration Algorithm

In this section, we introduce our single- and multi-Destination cluster VNF migration algorithm. Our proposed migration methods aim to enhance the h-HSLG algorithm [9], which relies on go-back and move forward mechanisms. A subset of VNFs in the VNF-FG of a NS is identified based on a sliding window w, the embedded VNFs are revisited and migrated, if needed. Please refer to [9] for details on this algorithm and the windowing strategy.

4.4.1 Single-Destination Cluster Migration

In single-Destination cluster VNF migration (sDCM), the VNFs are placed on a single or different nodes. The placed VNFs are grouped into a cluster of VNFs, which is then migrated to a single node in order to have an improved embedding cost. The algorithm aims to select the best cluster of VNFs $B_{cluster}$, to be migrated to a new node. The algorithm takes the threshold T_0 as an input parameter, which controls how much the algorithm can expand the cluster. T_0 helps us to avoid trapping into the local optimum; if at some stage there is no benefit in terms of cost, the algorithm tries to continue exploring by further expanding the cluster of VNFs to achieve cost improvement. Furthermore, T_0 is used to control the trade-off between complexity and solution quality.

The pseudo-code of our proposed sDCM algorithm is presented in Algorithm 1, where the inputs are threshold parameter T_0 , set $\{n_1, n_2, \ldots, n_{K'}\}$ of K' VNFs that reside in the current

Algorithm 2: Single-Destination Cluster Migration (sDCM)

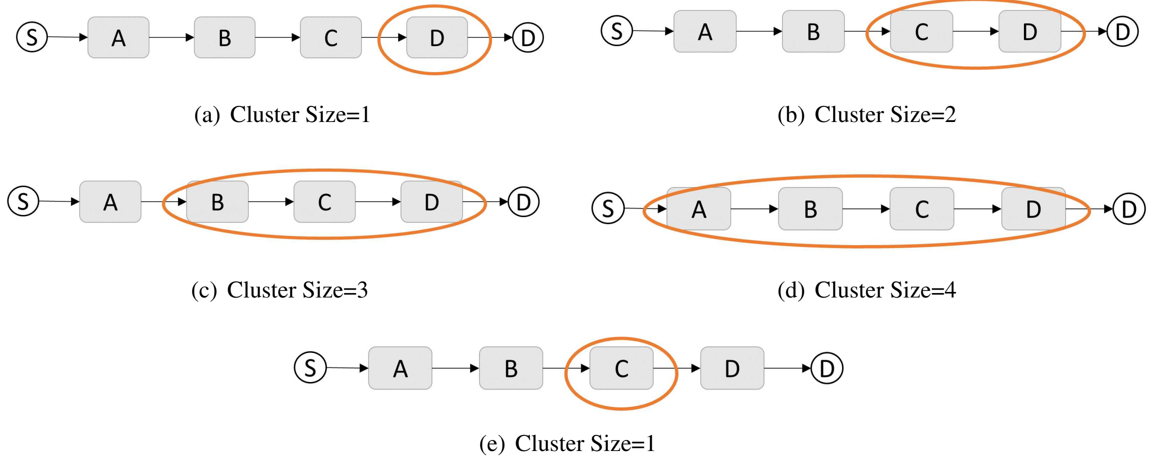
```
1  input: Threshold T_0, VNFs residing in the current sliding window N_{K'} ∈ {n_1, n_2, ..., n_{K'}},
     network information;
2  Initialize: ClusterSize=1, B_cluster = ∅, δ = 0;
3  for j = 1 : K' do
4  |   Embed VNF n_j using the h-HSLG algorithm ;
5  end
6  Calculate the total embedding cost C_t;
7  C_t^old ← C_t;
8  U ← ∅;
9  s ← K';
10 while δ < T_0 OR u_1 == n_1 do
11 |   for i = ClusterSize; i ≤ 1; i = i − 1 do
12 |   |   Add VNF n_{s−i+1} to U;
13 |   end
14 |   Find potential candidate nodes for hosting U ;
15 |   v_b ←Select the best node in terms of embedding cost ;
16 |   C_t^new ← Calculate the embedding cost by considering v_b as a destination node for U ;
17 |   if C_t^old ≤ C_t^new then
18 |   |   δ ← δ + 1 ;
19 |   else
20 |   |   B_cluster ← U ;
21 |   |   C_t^old ← C_t^new ;
22 |   end
23 |   ClusterSize ← ClusterSize + 1;
24 |   if u_1 == n_1 & B_cluster == ∅ then
25 |   |   CluterSize = 1;
26 |   |   s ← s − 1 ;
27 |   end
28 end
29 if B_cluster ≠ ∅ then
30 |   Migrate the B_cluster to v_b
31 end
```

sliding window w, and substrate network information. Given the input parameters, the algorithm returns the decision about the migration of the selected VNF cluster to a single node as an output. The algorithm starts by initializing $B_{cluster}$ as an empty set and expands the cluster of VNFs by increasing the parameter $ClusterSize$ at each iteration. We initialize parameter $ClusterSize$ to 1. The algorithm expands the cluster of VNF by keeping track of the value of δ, which counts the number of times that there is no cost improvement. In the beginning, δ is initialized to 0.

First, we use the h-HSLG algorithm [9] and embed/re-embed the VNFs of the current sliding window (see lines 3-5 in Algorithm 1). The h-HSLG algorithm aims to migrate only one VNF at a time. The total embedding cost C_t of VNFs that are located in window w is calculated using

Eq. 4.5 in line 6 of Algorithm 1. We consider a cluster U of VNFs, where $U \subseteq K$ and $U \in \{u_1, u_2, ..., u_{K'}\}$. Let u_1 and $u_K\prime$ denote the first and last VNFs of U, respectively. Before starting to form the cluster, U is set to $\emptyset$ (see line 8 in Algorithm 1). After setting $ClusterSize = 1$, the clustering process starts from the youngest VNF in the sliding window w (i.e., the last VNF $n_{K'}$). U is created from u_1 to $u_{K'}$ (see lines 11-13 in Algorithm 1). For illustration, in Fig.4.2.a, we assume a network service with four VNFs types A, B, C and D and threshold T_0=4. We consider a maximum window size for the sliding window, meaning that all the VNFs are placed in the network. The algorithm starts from the youngest VNF, which is D. In the first iteration, where $CluterSize = 1$, only one VNF exists in the cluster, which means u_1 is equal to $u_K\prime$. After creating a cluster of VNF U, the algorithm finds the potential destination nodes for U. The potential destination nodes are selected among the nodes that are already part of the embedding of the given network service. More specifically, we select the nodes that (i) have enough capacity to host U, (ii) support all types of VNFs belonging to U, and (iii) satisfy the latency constraints. Next, we find the best node v_b from the obtained potential nodes (see line 15 in Algorithm 1). v_b is a node with the smallest embedding cost among all other potential nodes, and can be a destination node for all the VNFs of U. After that, the new total embedding cost C_t^{new} is calculated by considering v_b as a destination node for all the VNFs of U (see lines 14-16 in Algorithm 1).

Next, we examine whether there is any cost improvement by comparing the C_t^{old} and C_t^{new}. If C_t^{old} is equal or smaller than C_t^{new} for a specific U, δ is incremented by one; otherwise, the current cluster of VNFs is considered as $B_{cluster}$ (see lines 17-22). The $ClusterSize$ is then incremented by one (see line 23). If $B_{cluster}$ is empty and u_1 is equal to n_1, there is no cost improvement by considering v_b as the destination node for U and the first VNF of the cluster equals the first VNF of the K. In this case, the clustering would start again with $ClusterSize = 1$, and the last VNF inside the cluster is shifted to the right (see lines 24-27). To be more specific, as shown in Fig. 4.2 the cluster size is incremented by one at each iteration by assuming $T_0 = 4$ until $ClusterSize = 4$. In Fig. 4.2.d, we assume that $B_{cluster}$ is equal to $\emptyset$ and $ClusterSize = 4$. The first VNF of U and $N_{K'}$ is A. In the next iteration, clustering starts from VNF C and $ClusterSize = 1$ (see Fig. 4.2.e), as it might be a case by starting from VNF D we cannot see any improvement, while by starting cluster from VNF C, we see a huge cost improvement. The algorithm repeats lines 10-32 until it



(a) Cluster Size=1

(b) Cluster Size=2

(c) Cluster Size=3

(d) Cluster Size=4

(e) Cluster Size=1

Figure 4.2: Expanding the Cluster of VNFs for $threshold = 4$.

reaches the termination condition, where threshold T_0 is greater than δ or the first VNF inside the w is equal to the first VNF within a cluster. Finally, if $B_{cluster}$ is not an empty set, the VNFs of the $B_{cluster}$ are migrated to v_b.

4.4.2 Multi-Destination Cluster Migration

A physical node would not be able to support all the VNF types. Furthermore, physical nodes have limited capacity, and the instantiation and reuse costs of VNFs may vary on different physical nodes. Thus, migrating a cluster of VNFs to a single physical node is not always feasible and/or cost-efficient. To address this limitation of the sDCM algorithm, we present our Multi-Destination Cluster Migration (mDCM) algorithm, where VNFs from multiple hosting nodes may form a cluster to be migrated to multiple destination nodes in order to improve the embedding cost. The pseudo-code of our proposed mDCM algorithm is presented in Algorithm 2. Similar to the sDCM algorithm, we use threshold T_0 as an input parameter. For each VNF inside the cluster U, we find a potential list P of nodes, to which VNFs can be migrated (see line 16 Algorithm 2). For finding P, we only consider the latency of youngest neighbor of VNF. For example, in Fig. 4.2.c, the youngest neighbor for VNF D is the destination, or VNF C is the youngest neighbor for VNF B. After that, we check whether P is an empty list or not. If P is not empty, the algorithm finds a feasible node v_f for a VNF from P; the feasible node is a node that satisfies all the constraints and latency requirements

Algorithm 3: Multi-Destination Cluster Migration (mDCM)

1 **input**: Threshold T_0, VNFs residing in the current sliding window $N_{K'} \in \{n_1, n_2, ..., n_{K'}\}$, network information;
2 **Initialize**: $ClusterSize$=1, $B_{cluster} = \emptyset$,, $\delta = 0$;
3 **for** $j = 1 : K'$ **do**
4 | Embed VNF n_j using the h-HSLG algorithm ;
5 **end**
6 Calculate the total embedding cost C_t;
7 $C_t^{old} \leftarrow C_t$;
8 $U \leftarrow \emptyset$;
9 **while** $\delta < T_0$ *OR* $u_1 == n_1$ **do**
10 | **for** $i = ClusterSize;\ i \leq 1;\ i = i - 1$ **do**
11 | | Add VNF n_{s-i+1} to U;
12 | **end**
13 | $s = ClusterSize$;
14 | **while** $s \leq ClusterSize$ **do**
15 | | $P \leftarrow$ Find potential candidate nodes for hosting u_s by considering examined neighbor ;
16 | | **if** $P \neq \emptyset$ **then**
17 | | | $v_f \leftarrow$ choose a feasible node from P ;
18 | | | $s = s - 1$;
19 | | **else**
20 | | | $s = s + 1$;
21 | | **end**
22 | **end**
23 | $C_t^{new} \leftarrow$ Calculate embedding cost by considering v_f as a destination node for each VNF inside U ;
24 | **if** $C_t^{new} \leq C_t^{old}$ **then**
25 | | $B_{cluster} \leftarrow U$;
26 | **else**
27 | | $\delta \leftarrow \delta + 1$;
28 | **end**
29 | $ClusterSize \leftarrow ClusterSize + 1$;
30 **end**
31 **if** $B_{cluster} \neq \emptyset$ **then**
32 | Migrate each VNF of $B_{cluster}$ to their own v_f;
33 **end**

of the youngest neighbors of the VNF. As an example for finding P for VNF D in Fig. 4.2, the latency requirement between D and destination is checked. After selecting the feasible node for a given VNF, the next VNF inside the cluster is checked. It might be the case that for a VNF, P is empty, and therefore the algorithm is not able to find a feasible node for a VNF. In this case, the algorithm returns to the previous VNF inside the cluster and selects another v_f instead of the previous one. For example, in Fig. 4.2.d, the algorithm finds v_f for VNF D and VNF C, but when it reaches VNF B, it cannot find any feasible node for B because of latency violation between VNFs B and C. The algorithm then returns to VNF C and selects another node (v_f) from P for VNF B. This back-tracking mechanism repeats until a feasible node is found for VNF B. After finding v_f for each VNF inside the cluster, the algorithm calculates the embedding cost by considering v_f for each VNF inside U (see line 24 Algorithm 2). As with to sDCM, if C_t^{new} is less than C_t^{old}, the U is considered as $B_{cluster}$; otherwise, the value of δ increases by 1. We increase the cluster until $\delta < T_0$ or u_1 is equal to n_1. Finally, all the VNFs of cluster $B_{cluster}$ are migrated to their own destination nodes (see line 34 Algorithm 2).

4.4.3 Complexity Analysis

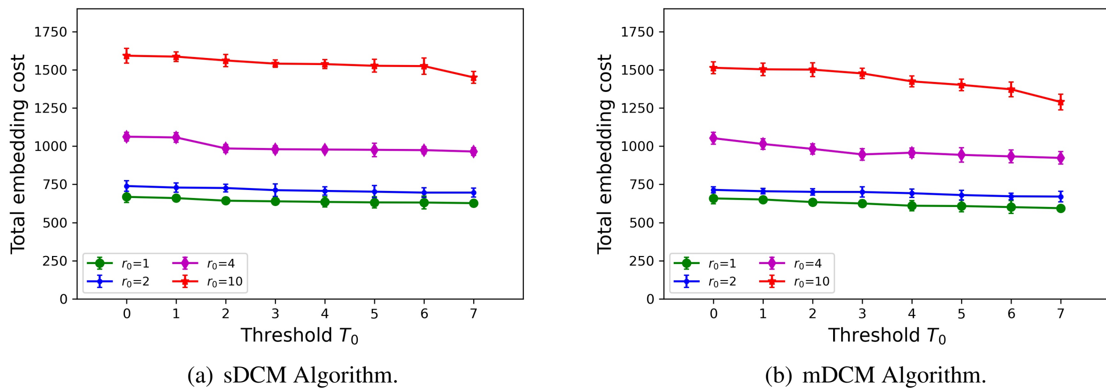
In the following, we present the complexity analysis of our proposed algorithms. In the proposed sDCM algorithm, the h-HSLG algorithm is called in lines 3-5 of Algorithm 1, which runs with a worst case complexity of $\mathcal{O}(|K|{\cdot}(|Z|{+}|E|))$. The WHILE loop (in line 10 of Algorithm 1) runs for a maximum of $|K|$ iterations. At each iteration, a maximum of $|K|{+}(|Z|{+}|E|)$ operations are carried out. This suggests that the time complexity of the proposed sDCM is $\mathcal{O}(|K|^2 + |K| \cdot (|Z| + |E|))$. Similarly, the time complexity of the proposed mDCM is $\mathcal{O}(|K|^2{+}|K|{\cdot}(|Z|{+}|E|){+}|K||Z|\text{log}|Z|)$.

4.5 Results and Discussions

In this section, we conduct simulations to evaluate the performance of our proposed algorithms against existing algorithms. We first describe the simulation environment and then present our obtained results.

Table 4.2: Parameter settings and default values

Parameter	Value	Ref.
Number of nodes $\|Z\|$	17, 37, 65, 113	[97, 98]
Number of links E	26, 57, 108, 184	[97, 98]
Available node capacity c_z	50 vCPU	[99]
Node cost ϕ_z	[1-5] vCPU	[97]
Available link bandwidth b_e	50 Gbps	[99]
Link cost τ_e	$[1-5]	[97]
Link delay d_e	[1-3] ms	[97]
Number of required VNFs in a request K	[5-12]	[100]
VNF resource requirement u_k	[1-3] vCPU	[100]
Instantiation cost η_k^z	1,2,4,10 $	[97]
Reuse cost $\Theta_{i,k}^z$	1 $	[97]
Processing demand a^k	[1-3] Gbps	[97]
Bandwidth requirement between VNF o_l	[1-3] Gbps	[100]
Latency requirement between VNFs d_l	[1-5] ms	[97]



(a) sDCM Algorithm.

(b) mDCM Algorithm.

Figure 4.3: Total embedding cost vs. threshold T_0 for our proposed (a) sDCM and (b) mDCM algorithms for different values of $r_0 \in \{1, 2, 4, 10\}$ (17 nodes, 30 request, and [5-8] VNFs per request).

4.5.1 Simulation Setup

We have implemented our algorithms in JAVA. All simulations are conducted on a PC with 1.60 GHz Intel(R) Core(TM) i5-8250U CPU and 8 GB of RAM. All simulation results are presented with 95% confidence intervals.

4.5.1.1 Settings

We considered four substrate network topologies with different sizes, namely, NOBEL-GERMANY [97] with 17 nodes 26 links, COST266 [97] with 37 nodes and 57 links, TA2 [97] with 65 nodes and 108 links, and ITC Deltacom [98] with 113 nodes and 184 links. The capacity of each physical node is set to 50 vCPU [99]. Furthermore, the resource cost of the node is selected from the range [$1-$5] per vCPU. Each link has an available bandwidth of 50 Gbps. The delay of each physical link varies from 1 ms to 3 ms. The physical link cost is between $1 to $5. We also assume that all nodes can not support all VNF types, and each supports 7 to 12 VNF types. Each request is represented by chain of 5 to 12 VNFs. Each VNF requires [1-3] vCPU [100] for instantiation. The processing capacity of all VNF types is set to 4 Gbps. We define a parameter r_0, which is the ratio of instantiation cost to reuse cost. The instantiation cost is selected from set {$1, $2, $4, $10}, whereas the reuse cost is set to $1. The processing demand of each VNF is [1-3] Gbps, and the bandwidth requirement between two VNFs is [1-3] Gbps. The latency requirement between two VNFs is [1-3] ms. All the network parameters and default values are summarized in Table 5.2.

4.5.1.2 Benchmarks

We compare the performance of our proposed algorithms with the following benchmarks:

- Brute-Force Search (BFS): Using an exhaustive search through all possible migration outcomes, the best decision is made to migrate a cluster of VNFs with the main objective of achieving the smallest cost. BFS is used to assess the optimality gap of our proposed algorithms.
- SG algorithm [8]: The SG algorithm is a greedy approach, which embeds the VNFs greedily in a sequential manner without performing any VNF migrations.
- h-HSLG algorithm [9]: It is a window-based approach along with go-back and move-forward mechanisms to offer efficient embeddings via simple migrations of visited VNFs.

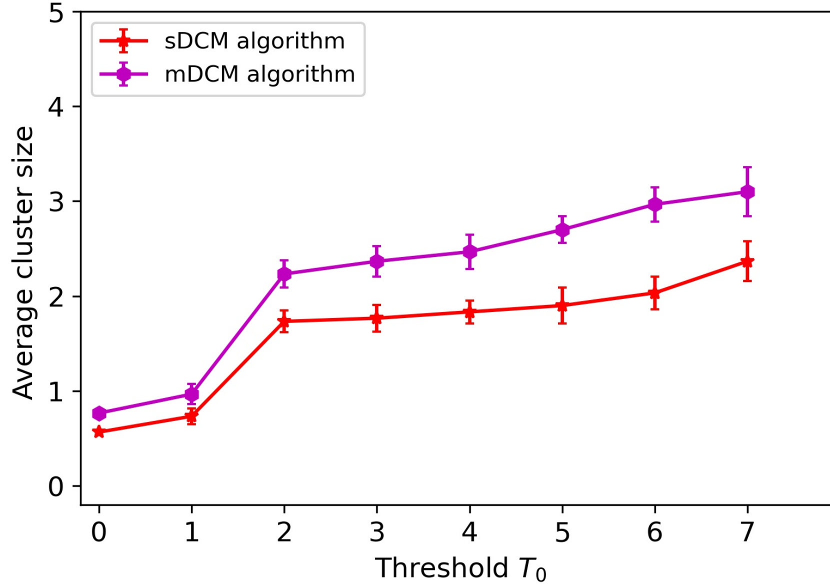
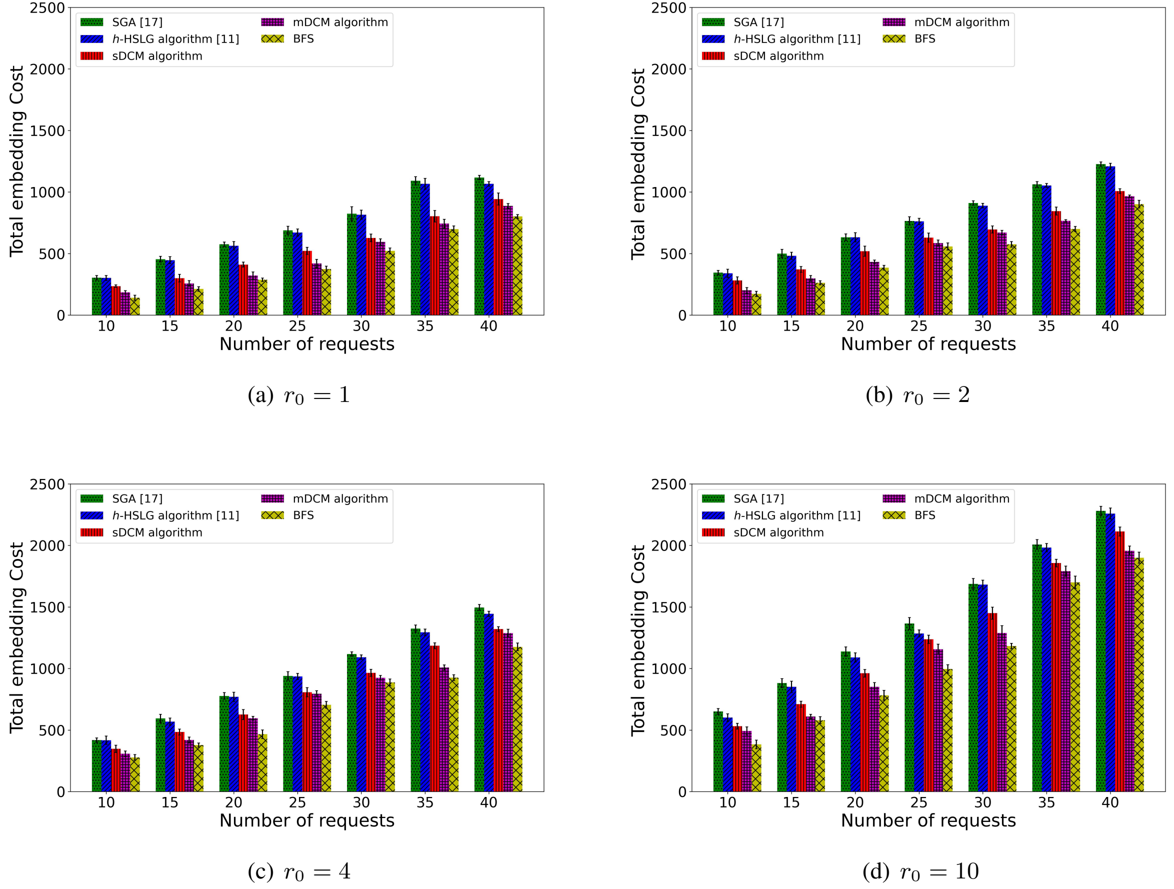


Figure 4.4: Average cluster size per request vs. threshold T_0 (with 17 nodes, 30 requests, $r_0 = 10$ and [5-8] VNFs per request.

4.5.2 Results

Figure 4.3 illustrates the total embedding cost vs. threshold T_0 for different values of $r_0 \in \{1, 2, 4, 10\}$ for our proposed sDCM and mDCM algorithms. In this scenario, the maximum number of required VNFs in a request is between 5 and 8. Thus, the value of threshold T_0 can vary between 0 and 7. We observe from Fig. 4.3.a that for a given r_0, the total embedding cost reduces as threshold T_0 increases. This trend can also be seen in Fig. 4.3.b, where the total cost reduces by increasing threshold T_0. We also observe from Fig. 4.3.a and Fig. 4.3.b that as r_0 increases, the proposed sDCM and mDCM algorithm achieves a larger performance gain. More specifically, for $r_0 = 10$, by increasing the threshold T_0 from 0 to 7, the proposed sDCM and mDCM algorithms can reduce the total embedding cost by 8% and 14%, respectively. This happens for two reasons. First, when the ratio r_0 of instantiation cost to reuse cost is large, there is a greater chance of leverage on the newly exposed context and further reduction of the embedding cost via migration of cluster of VNFs. This means that for large values of r_0, the reusability of VNFs becomes more beneficial, such that it would be worth migrating more clusters of VNFs to further decrease the embedding



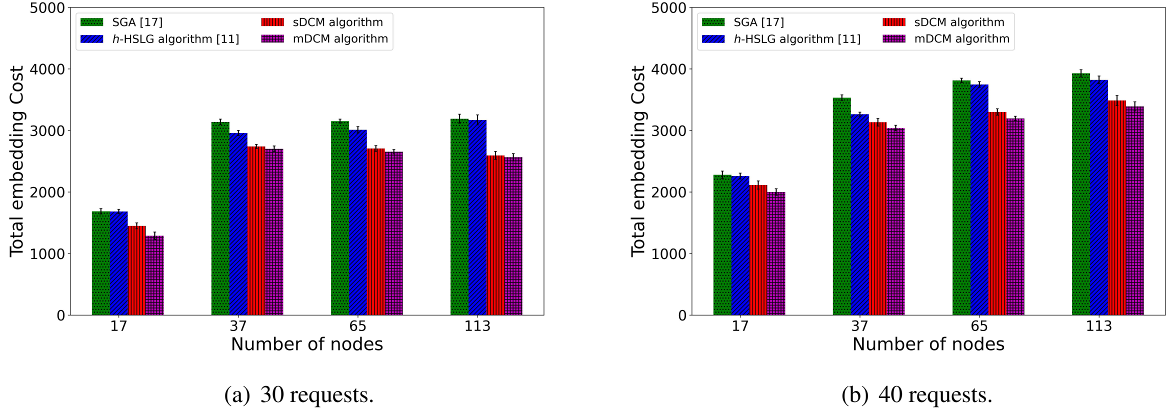
(a) $r_0 = 1$ (b) $r_0 = 2$

(c) $r_0 = 4$ (d) $r_0 = 10$

Figure 4.5: Total embedding cost vs. number of requests for (a) $r_0 = 1$, (b) $r_0 = 2$, (c) $r_0 = 4$, and (d) $r_0 = 10$ (with 17 nodes, $r_0 = 10$, [5-8] VNFs per request).

cost. Second, by increasing the threshold T_0, we allow the cluster of VNFs to expand as much as possible, thus increasing the chance of migrating a larger number of VNFs.

Next, we investigate the impact of threshold T_0 on the resultant cluster size. Figure 4.4 illustrates the average cluster size per request vs. threshold T_0 for sDCM and mDCM algorithms when $r_0 = 10$. We set r_0 to 10, as the instantiation cost is generally higher than the reuse cost. We observe from the figure that for larger values of threshold T_0, VNF clusters of larger sizes are formed in both sDCM and mDCM algorithms. Also, the average cluster size for mDCM algorithm is larger than the sDCM for any given threshold T_0. While the sDCM algorithm only migrates the VNF cluster to a single node, the already deployed instances of various VNF types from previous requests may reside on different nodes, and the nodes cannot support all VNF types in the given request. Thus,



(a) 30 requests.

(b) 40 requests.

Figure 4.6: Total embedding cost vs. number of nodes ([5-12] VNFs per request, and $r_0 = 10$).

the expansion of cluster size in the sDCM algorithm may stop at a smaller-size cluster of VNFs, as the VNFs are not allowed to migrate to more than one destination node.

Next, we examine the performance of our proposed sDCM and mDCM algorithms compared to those of the SG algorithm, h-HSLG algorithm, and BFS approach. We set the threshold T_0 to its maximum value, which leads to the smallest embedding cost, as shown in Fig. 4.3. Figure. 4.5 illustrates the total embedding cost vs. the number of requests ranging from 10 to 40 for different values of $r_0 \in \{1, 2, 4, 10\}$. We observe from Fig. 4.5 that as the number of requests increases for different values of r_0, the total embedding cost increases as a result, which was expected. We observe from Fig. 4.5.a that as the number of requests increases from 10 to 40 for r_0=1, the total embedding cost increase 79% and 75% in the proposed sDCM and mDCM algorithms, respectively. We can see similar trends for other values of r_0. For instance, by increasing the number of requests from 10 to 40 for r_0=10, the total embedding cost increases 75% and 73% for the proposed sDCM and mDCM algorithms, respectively as shown in Fig. 4.5.d. Clearly, the proposed sDCM and mDCM algorithms outperform both SG and h-HSLG algorithms for a given r_0 and number of requests. For instance, as shown in Fig. 4.5.b, for $r_0 = 2$ and 10 requests, the mDCM algorithm can reduce the total embedding cost by 34% and 33% compared to SG and h-HSLG algorithms, respectively. Also, by increasing the number of request to 40, the mDCM algorithm can reduce the embedding cost by 27% and 26% compared to the SG and the h-HSLG algorithms, respectively. We

notice from Fig. 4.5 that the SG algorithm performs worse than other approaches, mainly because it relies on local decisions, and is thus prone to deviate from the global optimum easily. The total embedding cost of the h-HSLG algorithm is larger than that of our proposed sDCM and mDCM algorithms. This happens mainly because the h-HSLG algorithm relies on simple migration only, which may not perform well when inter-VNF latency requirements are stringent. Therefore, the proposed algorithms can outperform the existing benchmarks not only for the small number of requests, e.g., 10 requests, but also for the large number of requests (e.g. 40 requests). In addition, the total embedding cost of our proposed sDCM and mDCM algorithms are, on average, only 15% and 6% higher than that of BFS, respectively. Moreover, we observe from Fig. 4.5 that our proposed mDCM algorithm performs better than the sDCM algorithm, mainly because a cluster of VNFs has more chance to be migrated to multiple destination nodes (compared to a single destination node). VNFs in a cluster can spread to different nodes that can support them and have an already deployed instance of that VNF type from a previous request. Therefore, the total embedding cost can be further decreased compared to the single-destination case.

Figure 4.6 depicts the total embedding cost vs. the number of nodes for 30 and 40 requests, [5-13] VNFs per request and $r_0 = 10$. We note that the results of BFS approach are not shown here due to its exponential time complexity, which limits its applicability to very small size problem instances. We can notice from the figure 4.6.a that when the number of nodes increases, the total embedding cost increases in all algorithms under consideration because as the number of nodes increases, the transmission cost increases as a result. Consequently, the instantiation cost becomes more preferable. According to Fig. 4.6.b, while the proposed sDCM and mDCM algorithms outperform the existing benchmarks in all scenarios, the mDCM achieves the smallest total embedding cost. Furthermore, the total embedding cost of our proposed sDCM algorithm is on average only 4.16% higher than that of the mDCM algorithm. We observe from Fig. 4.6.b that when the number of nodes increases, the total embedding cost increases in all algorithms under consideration. As shown in Figs. 4.6.a and 4.6.b, when the number of requests increases from 30 to 40 for 17 nodes, the total embedding cost increases by almost 12% in sDCM and mDCM algorithms, respectively. These changes become 22% and 19% when the number of nodes becomes 113. According to Fig. 4.6.b, while the proposed sDCM and mDCM algorithms outperform the existing benchmarks

in all scenarios, the mDCM achieves the smallest total embedding cost. More specifically, the total embedding cost of our proposed mDCM for 40 requests is on average 4.75% smaller than the sDCM algorithm. According to the results explained above, our proposed sDCM and mDCM algorithms outperform the existing benchmarks in terms of total embedding costs. We note, however, that this comes at the expense of an additional computational burden, which is evaluated next.

Table 4.3 illustrates the execution time of all algorithms under consideration for different scenarios and problem sizes. We observe from Table 4.3 that the execution time, of our proposed sDCM and mDCM algorithms are not only much shorter, but they also grow at a slower rate than that of BFS. According to Table 4.3, for 17 nodes, when the number of requests increases from 10 to 30, the execution time of the BFS algorithm increases from 768 to 3480 seconds, whereas the execution times of the sDCM and mDCm algorithms increase from 0.54 s to 1.3 s, and from 0.73 s to 1.6 s, respectively. For instance, when the number of nodes and requests are 17 and 30, respectively, and $r_0 = 10$, the execution times of our proposed sDCM and mDCM algorithms are about 60% and 68% higher than those of the SG and h-HSLG algorithms, enabling us to obtain 14% and 20% improvement of total embedding cost, respectively. On the other hand, the BFS takes 99 times more execution time than the sDCM and mDCM algorithms to gain only 16% and 8% improvement in the total embedding cost (see Fig. 4.6.a and Table 4.3). We note that the BFS algorithm cannot find the solution within 2 hours for a larger size problem due to its exponential time complexity. Furthermore, by increasing the number of requests from 10 to 30 for 17 nodes, the execution time of SGA, h-HSLG algorithm, and the proposed sDCM and mDCM algorithms increases by 44%, 47%, 76%, and 87%, respectively. While increasing the number of nodes from 17 to 113 for 30 requests, the execution times of the proposed sDCM and mDCM algorithms increase by 94% and the execution times of SGA and h-HSLG algorithms increase by 93%, respectively. For instance, when the number of nodes and requests are 113 and 30, and r_0=10, the execution time of our proposed sDCM and mDCM algorithms is about 60% higher than the SG and h-HSLG algorithms, producing 13% and 14% improvements of the total embedding cost (see Fig. 4.6.a and Table 4.3). Moreover, for 17 nodes and 40 requests, the BFS did not return any solution in 2 hours, while the proposed sDCM and mDCM algorithms returned the solution in less than 2 s. This indicates that the proposed algorithms can achieve near-optimal results within a small execution time. This also highlights the

Table 4.3: Execution Time (seconds).

Nodes	# of Req.	BFS	SG [8]	h-HSLG [9]	sDCM	mDCM
17	10	768	0.26	0.32	0.54	0.73
17	20	1632	0.3	0.56	0.82	0.92
17	30	3480	0.47	0.61	1.34	1.61
17	40	>2hrs	0.52	0.65	1.38	1.72
37	10	>2hrs	0.7	0.8	2.12	2.73
37	20	>2hrs	1.02	1.3	3.35	4.32
37	30	>2hrs	1.4	1.6	4.67	4.8
37	40	>2hrs	1.7	1.9	4.92	5.24
65	10	>2hrs	0.9	1.1	2.76	3.73
65	20	>2hrs	1.53	1.92	4.6	5.1
65	30	>2hrs	2.2	2.77	4.91	5.87
65	40	>2hrs	2.83	3.62	5.5	6.7
113	10	>2hrs	2.1	2.79	4.42	6.03
113	20	>2hrs	3.3	3.08	7.3	9.01
113	30	>2hrs	4.09	4.73	10.3	13.21
113	40	>2hrs	4.71	5.53	13.79	18.03

Note: ">2 hrs" means that the algorithm cannot find solutions within 2 hours.

fact that using an exhaustive search algorithm (e.g., BFS) to solve the cluster migration problem becomes impractical in large-scale scenarios. Therefore, the proposed algorithms can deal with the scalability issue of the BFS approach. To see this, when the number of requests increases from 10 to 40 for 113 nodes, the execution time for the proposed sDCM and mDCM algorithms increases by only 66%, and 76%, respectively. Although the execution time of the proposed algorithms are slightly larger than SGA and h-HSLG algorithms, they can improve the total embedding cost by 17% and 7%, respectively (see Fig. 4.6.b and Table 4.3).

4.6 Conclusions

In this chapter, we studied the problem of migrating a cluster of VNFs, taking into account the latency requirement between VNFs and reusing the already-deployed VNFs. The objective was to migrate the cluster of VNF so that the total embedding cost, including resource, instantiation, reuse, and transmission cost, is minimized. We proposed two variants of VNF cluster migration algorithms, namely, single- and multiple-Destination cluster migration algorithms, which allow for migrating a cluster of VNFs rather than a single VNF. Extensive simulations were conducted to

evaluate the performance of our proposed algorithm. The simulation results show that our proposed algorithms can improve the existing benchmarks. Our results indicate that the proposed algorithm can achieve up to a 14% improvement in total embedding cost compared to the existing benchmarks, which comes at the expense of a 60% increase in execution time.

Chapter 5

Joint VNF Decomposition and Migration for Cost-efficient VNF Forwarding Graph Embedding[4]

5.1 Introduction

To cope with resource shortage, which may block the incoming NSs from being admitted successfully, VNF migration can be a viable solution to avoid over-and/or under-loaded situations in VNF-enabled networks [102, 103]. When a node's resources are all occupied by running workloads, it becomes over-loaded, potentially causing bottlenecks that prevents the service provider from admitting new requests. Conversely, under-loaded nodes that host only a few traffic flows lead to inefficient resource usage and increased power consumption. The amount of available resources within substrate nodes can have a significant impact on determining the best decomposition option. On the other hand, selecting the proper decomposition options may require some VNFs to be moved from an over-loaded node to another. Thus, a major challenge is to determine how the already deployed VNFs can be shuffled around the substrate network so as to select the best decomposition

[4]This chapter is based on a published paper [4] and a submitted paper [101]: S. N. Afrasiabi et al, "Joint VNF decomposition and migration for cost-efficient VNF forwarding graph embedding," 2023 IEEE Global Communications Conference (GLOBECOM).
[101] S. N. Afrasiabi, et al, "Joint VNF decomposition and migration for cost-efficient VNF forwarding graph embedding," Submitted To IEEE Trans. Netw. Serv. Manag (To be submitted).

option to minimize the embedding cost. To fully reap the benefits of VNF decomposition, it can be used in conjunction with VNF migration to help promote VNF reusability across the network to reduce the embedding cost.

In this chapter, we propose an efficient heuristic algorithm to solve the joint problem of VNF decomposition and migration. After modeling the problem as an optimization problem to minimize the embedding cost of a given NS, we propose our heuristic, which comprise two components, namely, Cost-Aware VNF Decomposition (CA-VNF-D) and Node-Aware VNF Migration (NA-VNF-M) algorithms. While the CA-VNF-D algorithm gives us a near-optimal decomposition option, the NA-VNF-M algorithm aims to migrate the already deployed VNFs to other nodes to make room for embedding the selected decomposition option successfully. The simulation results indicate that our proposed algorithm outperforms the decomposition approach in terms of embedding cost .

The rest of this chapter is organized as follows. The motivation scenario is presented in Section 5.2. We present the system model and problem formulation in Section 5.3. The proposed topological decomposition and VNF migration strategies are introduced in Section 5.4. The numerical results are shown in Section 5.5. Finally, Section 5.6 concludes this chapter.

5.2 Motivating Scenario

In this section, we demonstrate how VNF migration can affect the selection of decomposition options and subsequently the VNF-FG embedding cost. First, let us consider a network, comprising 3 physical nodes and 2 physical links, as shown in Fig. 5.1.a. Each node is associated with a node resource capacity and a cost per resource unit. For simplicity, we assume all nodes support all VNF types. Similarly, each link has a bandwidth capacity, a cost per bandwidth unit, and a link latency. Furthermore, each VNF type is associated with a predefined capacity demand that consists of fixed and variable capacities. The allocated capacity during runtime is associated with fixed capacity, and whenever a workload is added to a VNF instance, that VNF encounters a variable capacity. Therefore, once a VNF instance is newly instantiated, it is attributed with fixed and variable capacities and a variable capacity when it is reused by another NS. For simplicity, we assume that the fixed capacity of all VNF types is 4 units, and the variable capacity is 2 units, except we assume the fixed

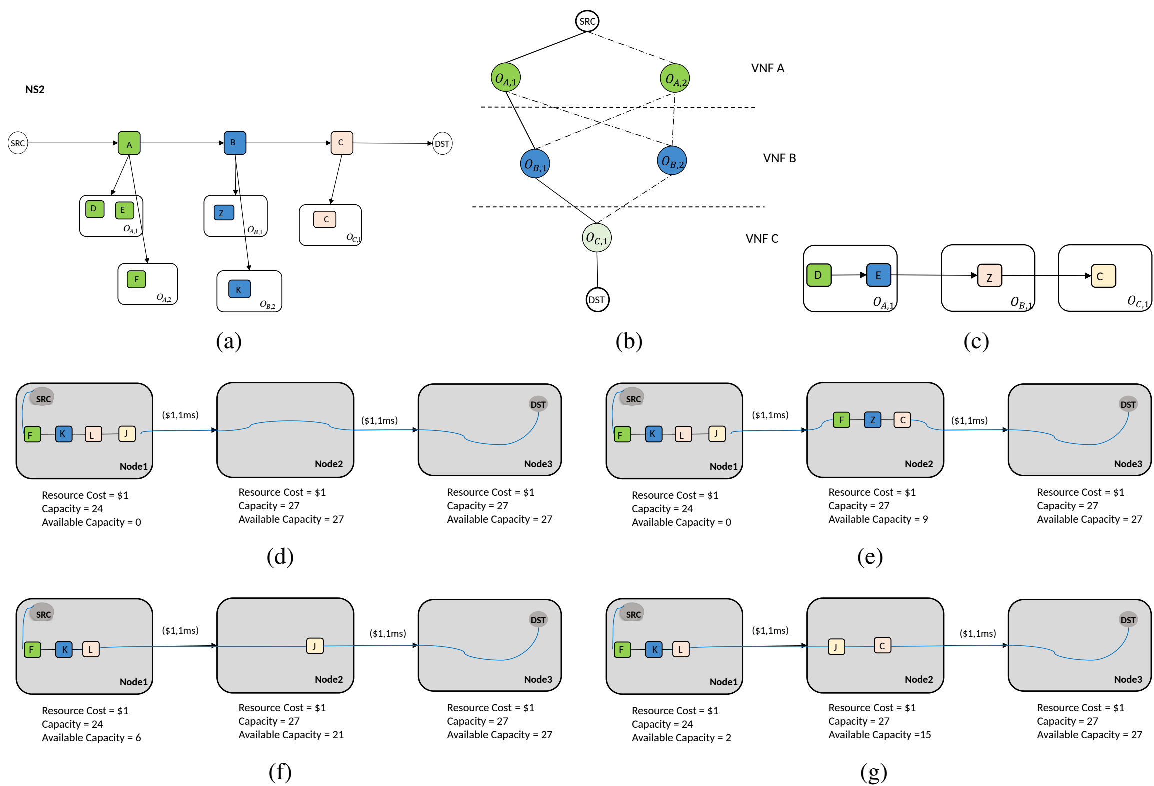


Figure 5.1: (a) Incoming NS 2 with possible decomposition options, (b) tree graph showing all possible VNF-FGs for NS 2, (c) one possible realization of VNF-FG for NS 2, (d) substrate network topology with NS 1 being embedded, (e) embedding of best decomposition option of NS 2 without considering migration, (f) migration of VNF J to Node 2, and (g) embedding of the best decomposition option of NS 2 after migration.

and variable capacity demand of VNF type Z is 3 and 2 units, respectively. Next, let us assume two NSs arrive in the network in which NS 1 is already deployed to a network while we are trying to embed NS 2 into a network. NS 1 which consists of VNFs F, K, L and J, is placed in Node 1. NS 1 requires its traffic to be transmitted from source node Node 1 to destination Node 3. The capacity demand by NS 1 is 24 units. As all the VNFs on Node 1 are newly instantiated, they require fixed and variable capacities (e.g., VNF F needs 4 units of fixed capacity and 2 units of variable capacity). Therefore, the available capacity of Node 1 is 0.

Let us also consider NS 2 requires the ordered set of VNFs consisting of VNFs A, B, and C from Node 1 to Node 3, as shown in Fig. 1a. The SP must embed VNFs A, B, and C onto the network and make sure that the traffic of NS 2 is transmitted from the source to the destination while satisfying

the given requirements. We note that each VNF can have one or multiple decomposition options, and each option can comprise one or multiple subfunctions as illustrated in Fig. 1a. For instance, VNF A has two decomposition options. Let $o_{i,j}$ be the decomposition option j of VNF i. For example, Option 1 of VNF A ($o_{A,1}$) consists of two subfunctions (i.e., D and E). Figure 1b shows a tree graph illustrating all connections of decomposition options between the VNFs for NS 2, which has a total of 4 decomposition options. Figure 1d depicts one possible VNF-FG of NS 2, which can be achieved via topological decomposition. Figure 1c depicts one possible VNF-FG of NS 2, which can be achieved via topological decomposition.

In this example, it would be more cost-efficient to select a decomposition option that can reuse other VNFs in the network and has a smaller number of sub-functions. Sub-functions F and K from NS 2 can be reused as the same type of VNF instances of NS 1 are already deployed in Node 1 (see Fig 5.1.a). However, Node 1 does not have enough capacity to serve any NS. Without considering any migration, the best decomposition option would be F, Z, and C. VNFs F and C require 4 fixed capacity and 2 variable capacity, and VNF Z requires 3 fixed capacity and 2 variable capacity. Assuming that the resource cost of Node 2 is \$1, the total resource cost of this VNF-FG would be \$17 and the link cost would be \$2. Therefore, the embedding cost would be \$19 and the total embedding cost is \$43, as shown in Fig. 5.1.e. However, by migrating VNF J to Node 2, the available capacity of Node 1 becomes 6 units (see Fig1. 5.1.f. Therefore, Node 1 will have enough capacity to serve other NSs. We can reuse VNFs F and K on Node 1, and therefore the best decomposition option of NS 2 will be F, K, and C. The total embedding cost would be \$38 for this VNF-FG (see Fig. 5.1.g). Migration can help reduce the total embedding cost by \$5.

The discussion above demonstrates that the selection and embedding of VNF-FG without doing any migration strategy can potentially select an inefficient decomposition option which leads to a poor embedding solution. To address this issue, efficient mechanisms are still needed to jointly select the best decomposition option and migrate VNFs. In this chapter, we aim to select the best decomposition option by migrating embedded VNFs to other nodes to make room for selecting the optimal decomposition option such that embedding cost is reduced by reusing the already instantiated VNF instants.

Table 5.1: Summary of main notations.

Input Parameters	
N	Set of substrate nodes
L	Set of substrate links
c_n	Available capacity of node n
ω_n	Cost per capacity unit of node n
b_l	Available bandwidth of physical link l
d_l	Delay of substrate link l
θ_l	Cost per bandwidth unit for link l
K	Set of all VNF types supported in the network
f_k	Fixed capacity demand for VNF type k
v_k	Variable capacity demand for VNF type k
p_k	Processing capacity of VNF type k
E	Set of virtual links
b_e	Bandwidth requirement of virtual link e
d_e	Delay requirement of virtual link e
I_k	Set of VNF instances of VNF k
$m_{i,k}$	Processing load assigned to instance i of VNF type k
DC	Decomposition options for the given NS
K_{dc}	Set of all VNFs in decomposition option dc
E_{dc}	Set of all virtual links in decomposition option dc
$z_n^{i,k,dc}$	Binary parameter, indicating whether instance i of VNF type k of decomposition dc has been deployed in substrate node n
w_n^k	Binary parameter, indicating whether physical node n can support VNF type k
$z_n^{i,k,dc}(t)$	Binary variable, indicating whether or not instance i of VNF type k of a decomposition dc is mapped on physical node n in snapshot t
$Z_n^{i,k,dc}(t)$	Binary variable, indicating whether or not instance i of VNF type k of a decomposition option dc is mapped on substrate node n as a newly deployed instance in snapshot t
$y_l^{e,dc}(t)$	Binary variable, indicating whether or not virtual link e of decomposition dc is mapped to substrate node $l \in L$ (1) for decomposition option $dc \in DC$ in snapshot t
x^{dc}	Binary variable, indicating whether or not decomposition $dc \in DC$ is selected for the mapping

5.3 System Model and Problem Formulation

5.3.1 System Model

We describe the main notations in the following. Table 5.1 lists the key notations and decision variables. Let the network be represented as an undirected graph $G = (N, L)$, where N is a set of substrate nodes, and L is the set of substrate links. Each substrate node n is associated with an available node capacity c_n (e.g., CPU, memories, storage), cost per capacity unit ω_n, and a set of supported VNF types $K_z \subseteq K$, where K is a set of all VNF types that are supported in the

network. Each substrate link $l = (n, n') \in L, \forall n, n' \in N$, has an available bandwidth b_l, cost per bandwidth unit θ_l, and link delay d_l. Each VNF $k \in K$ is associated with a predefined fixed and variable capacity demand in terms of computation, memory, and storage which are defined as f_k and v_k, respectively. While f_k is attributed to VNF type k when they are newly instantiated, v_k is considered when a workload is added to VNF type k. Also, the processing capacity of VNF type k is denoted as p_k. The set of virtual links is denoted by E, where $e(i, j) \in E$ is the link between VNF indices i and j. Also, the maximum allowed delay and bandwidth requirements of virtual link $e \in E$ are defined as b_e and d_e, respectively. Furthermore, we define $I_k = \{1, 2, ..., |I_k|\}$ to specify a set of VNF instances corresponding to VNF type $k \in K$, where $|I_k|$ is the maximum number of instances of VNF type k that is allowed to be instantiated across the network. In addition, we define $m_{i,k}$ to specify the amount of processing load that has already been assigned to instance i of VNF type k.

Now, let us focus on an online scenario, where NSs arrive in the network at different time instants. The incoming NS requires its traffic to be traversed through a set of VNFs connected in the form of a VNF-FG. As explained in section 5.1, an NS can be realized through multiple decomposition options. Therefore, for each NS, a decomposition set $DC = \{dc_1, dc_2, ..., dc_n\}$ contains all the possible decomposition options. Each decomposition option $dc \in DC$ is represented as a directed graph $G_{dc} = (K_{dc}, E_{dc})$ to support the dependency between VNFs. Therefore, a set of all VNFs $K_{dc} \subseteq K$ in a decomposition dc are represented as nodes connected via directed virtual links E_{dc} in the graph, where $E_{dc} \subseteq E$ is the set of all virtual links for a decomposition dc. Finally, we define two binary parameters that are used in our problem formulation. The first binary parameter is $\tilde{z}_n^{i,k,dc} \in \{0, 1\}$ that indicates whether instance $i \in I_k$ of VNF type $k \in K$ of the decomposition option $dc \in DC$ has already been deployed in substrate node $n \in N$ (1) or not (0). The second one is w_n^k which indicates whether substrate node $n \in N$ can support VNF type $k \in K$ (1) or not (0). Our model operates over a set of snapshots. We define a snapshot t as a representation of the system state over a fixed time interval. The model operates over two network snapshots: the current snapshot $t-1$ (before migration of a cluster of VNFs) and the new one (after migration of a cluster of VNFs) at snapshot t. Migrations can be carried out between snapshots $t-1$ and t. The number of newly deployed and reused VNF instances on a specific node for a request may vary from one

snapshot to another [95].

5.3.2 Problem Formulation

In the following we develop an ILP model to solve the joint VNF decomposition and migration problem. To this end, we consider the following four decision variables:

- $z_n^{i,k,dc}(t) \in \{0,1\}$ is a binary variable, specifying whether instance $i \in I_k$ of VNF type $k \in K$ of a decomposition option $dc \in DC$ is mapped on physical node $n \in N$ in snapshot t (1) or not (0).
- $Z_n^{i,k,dc}(t) \in \{0,1\}$ is a binary variable, specifying whether instance $i \in I_k$ of VNF type $k \in K$ of a decomposition option $dc \in DC$ is mapped on physical node $n \in N$ as a newly deployed instance in snapshot t (1) or an already deployed instance/not deployed (0) in the network.
- $y_l^{e,dc}(t) \in \{0,1\}$ is a binary variable, indicating whether virtual link $e \in E_{dc}$ of decomposition dc is mapped to physical node $l \in L$ (1) for decomposition option $dc \in DC$ in snapshot t, or not (0).
- $x^{dc} \in \{0,1\}$ is a binary variable, to indicate whether decomposition $dc \in DC$ is selected for the mapping (1), or not (0).

Next, we describe the constraints of the problem. Constraints (5.1) and (5.2) guarantee that only one of the decompositions of an NS is selected and all the VNFs of the selected decomposition are mapped only once.

$$\sum_{n \in N} z_n^{i,k,dc}(t) = x_{dc}, \quad \forall dc \in DC \quad \forall k \in K, \quad \forall i \in I_k \tag{5.1}$$

$$\sum_{dc \in DC} x_{dc} = 1, \tag{5.2}$$

Constraint (5.3) ensures that instance i of VNF type k can be deployed in physical node z, only if node z can support VNF type k:

$$z_n^{i,k,dc}(t) \leq w_k^n, \quad \forall dc \in DC, \quad \forall k \in K, \quad \forall i \in I_k, \quad \forall n \in N. \tag{5.3}$$

Constraint (5.4) ensures that each VNF instance i of VNF type k is not deployed more than once in the network:

$$\sum_{n \in N} z_n^{i,k,dc}(t) \leq 1, \quad \forall dc \in DC, \quad \forall k \in K, \quad \forall i \in I_k. \tag{5.4}$$

Constraint (5.5) ensures that instance i of VNF type k can either be reused or instantiated in physical node n:

$$z_n^{i,k,dc}(t) \geq \tilde{z}_n^{i,k,dc}(t), \quad \forall k \in K, \quad \forall i \in I_k, \quad \forall n \in N, \quad \forall dc \in DC. \tag{5.5}$$

Constraint (5.6) helps to indicate that if instance i of VNF type k is newly deployed in physical node n, $Z_n^{i,k,dc}$ is equal to 1; otherwise, it is equal to 0:

$$Z_n^{i,k,dc} = z_n^{i,k,dc}(t) - \tilde{z}_n^{i,k,dc}(t), \quad \forall k \in K, \quad \forall i \in I_k, \quad \forall n \in N, \quad \forall dc \in DC. \tag{5.6}$$

Constraint (5.7) ensures that the capacity of each physical node n does not exceeded the capacity of that node:

$$\sum_{k \in K} \sum_{i \in I} f_k Z_n^{i,k,dc}(t) + v_k z_n^{i,k,dc}(t) \leq c_n, \quad \forall n \in N, \quad \forall dc \in DC. \tag{5.7}$$

Constraint (5.8) ensures that the processing load assigned to instance i of VNF type k does not exceed its available processing capacity:

$$\sum_{n \in N} z_n^{i,k,dc}(t) \leq p_k - m_{i,k}, \quad \forall k \in K \quad \forall n \in N, \quad \forall i \in I_k, \quad \forall dc \in DC. \tag{5.8}$$

Constraint (5.9) ensures that the bandwidth capacity of each physical link l is not exceeded:

$$\sum_{e \in E} b_e y_l^{e,dc}(t) \leq b_l, \quad \forall dc \in DC \quad \forall l \in L. \tag{5.9}$$

Constraint (5.10) ensures that the delay requirement of each virtual link is met:

$$\sum_{l \in L} d_e y_e^l(t) \leq d_l, \quad \forall e \in E, \quad \forall dc \in DC. \tag{5.10}$$

Constraint (5.11) ensures that two consecutive VNFs k and k' with a virtual link e are deployed on a physical link $l(n, n') \in L$:

$$\begin{aligned} \sum_{l_{n,n'} \in L} y_{l_{nn'}}^{e,dc}(t) - \sum_{l_{n',n} \in L} y_{l_{n'n}}^{e,dc}(t) = x_n^{i,k,dc}(t) - x_n^{i,k',dc}(t), \\ \forall n \in N, \quad \forall k, k' \in K, \quad \forall e \in E, \quad \forall dc \in DC. \end{aligned} \tag{5.11}$$

With all these considerations, our objective is to minimize the total embedding cost, which comprises the cost of fixed capacity, variable capacity, and transmission. We formulate the problem as an ILP, to be explained in technical detail next.

5.3.2.1 Fixed Capacity Cost

Fixed capacity cost $C_{fix}(t)$ associated with the cost of capacity demand for instantiating a new VNF instance in snapshot t on a node. The node capacity may be released during the migration of a VNF k from snapshot $t-1$ to t, which leads to a cost reduction. Therefore, it requires considering the difference in occupied capacity of a physical node before and after migration, which is given by:

$$C_{fix}(t) = \sum_{dc \in DC} \sum_{n \in N} \sum_{k \in K_{dc}} \sum_{i \in I_k} \omega_n . f_k [Z_n^{i,k,dc}(t) - Z_n^{i,k,dc}(t-1)]^+, \tag{5.12}$$

where the terms $[Z_n^{i,k,dc}(t) - Z_n^{i,k,dc}(t-1)]^+$ is given by $\max\{Z_n^{i,k,dc}(t) - Z_n^{i,k,dc}(t-1), 0\}$. It should be noted that by migration from a snapshot $t-1$ to t, some VNFs need to be newly

instantiated in a node and accordingly encounter the capacity cost of the node based on their capacity demand. As such, $[Z_n^{i,k,dc}(t) - Z_n^{i,k,dc}(t-1)]^+$ calculates the number of new VNF instantiations of VNF k on node n in snapshot t.

5.3.2.2 Variable Capacity Cost

The variable capacity cost $C_{var}(t)$ is introduced when an additional workload is added to a VNF which is associated with the cost per variable capacity demand of a VNF on a physical node. We compute the variable capacity cost $C_{var}(t)$ of such VNFs from snapshot $t-1$ to snapshot t, as follows:

$$C_{var}(t) = \sum_{dc \in DC} \sum_{n \in N} \sum_{k \in K_{dc}} \sum_{i \in I_k} \omega_n . v_k [z_n^{i,k,dc}(t) - z_n^{i,k,dc}(t-1)]^+. \tag{5.13}$$

where the terms $[z_n^{i,k,dc}(t) - z_n^{i,k,dc}(t-1)]^+$ is given by $\max\{z_n^{i,k,dc}(t) - z_n^{i,k,dc}(t-1), 0\}$.

5.3.2.3 Transmission Cost

The transmission cost $C_{tr}(t)$ includes the cost of network bandwidth consumed for the communication between VNF instances, as follows:

$$C_{tr}(t) = \sum_{dc \in DC} \sum_{e \in E_{dc}} \sum_{l \in L} \theta_l b_e [y_l^{e,dc}(t) - y_l^{e,dc}(t-1)]^+. \tag{5.14}$$

which considers the differential cost of the assigned link that may happen between two consecutive snapshots as a result of the migration of VNFs.

The objective of our optimization problem is to minimize the sum of the aforementioned costs, as follows:

$$C_{total} = \min(C_{fix}(t) + C_{var}(t) + C_{tr}(t)). \tag{5.15}$$

5.4 Proposed Solution

In this section, we propose our solution to solve the problem of joint VNF decomposition and migration efficiently. Our proposed solution comprises two phases, namely, (i) decomposition selection, and (ii) migration. In the first phase, our so-called Cost-Aware VNF Decomposition (CA-VNF-D) algorithm aims to select the best decomposition option of a given NS among its all possible options. More specifically, we measure the cost associated with each decomposition based on a scoring strategy, which then helps select the one with the minimum score. Additionally, we determine the potential node for each VNF of the selected decomposition option for embedding. In the migration phase, we run our Node-Aware VNF Migration (NA-VNF-M) algorithm, where the CA-VNF-D algorithm is executed for both loaded and unloaded networks. More specifically, "loaded network" refers to the network in its current state, where each node bears the load. On the other hand, the "unloaded network" refers to the network without any load, where nodes have enough capacity to host VNFs. Our migration strategy examines whether the score of the selected decomposition in the unloaded network is smaller than that in the loaded network. If this criterion is satisfied, the migration process is triggered to move the already embedded VNFs from their hosting nodes to alternative nodes, thereby creating capacity for the embedding of the selected decomposition options. In the following, we explain the two phases of our proposed solution in technically greater detail.

5.4.1 Cost Aware VNF Decomposition (CA-VNF-D)

Our proposed CA-VNF-D algorithm aims to identify the most cost-efficient VNF-FG without requiring the embedding of all decomposition options onto physical nodes. Given the substrate network, NS, and all of its decompositions represented by DC, the so-called decomposition score S_d of option dc is calculated as follows:

$$S_d = S_f + S_v + S_{tr}, \tag{5.16}$$

where S_f, S_{var}, and S_{tr} are given as follows:

$$S_f = \omega_n.f_k, \tag{5.17}$$

$$S_{var} = \omega_n.v_k. \tag{5.18}$$

$$S_{tr} = \theta_l b_e. \tag{5.19}$$

respectively. In Eq. (5.16), S_f is the score of the fixed capacity cost, S_v is the score of the variable capacity cost, and S_{tr} is the score of the transmission cost.

The pseudo-code of our proposed CA-VNF-D algorithm is shown in Algorithm 4. The CA-VNF-D algorithm works as a pre-evaluation step for each VNF prior to its embedding into the network. To explore the decomposition options of VNFs within an NS, the algorithm first initializes a tree graph with the source node (src) and destination node (dst). For a given NS, a source-destination pair is assigned to the tree graph. Let Ł denote the number of levels in the tree graph, where each level is associated with a VNF type in DC. As an example, consider Fig 5.1(b), where Node 1 and Node 3 are source node (src) and destination node (dst), respectively. Additionally, there are a total of Ł $= 3$ levels in the tree graph. Each edge within the tree graph is associated with a $start_{value}$ and an end_{value}. To begin with, $start_{value}$ is set to src, while end_{value} is set to the first level of the tree graph ($Ł_1$) (see lines 2-3 in Algorithm 4). For the first edge of the tree between src and $O_{A,1}$, the source node of the tree is $start_{value}$ (e.g., the source node for subfunction D within $O_{A,1}$ is Node 1 in Fig 5.1.c), and for subsequent VNFs, $start_{value}$ is determined by the potential candidate node n_b of the preceding VNF. Next, all the decomposition options O between $start_{value}$ and end_{value} are explored (see line 7 in Algorithm 4). For each option $o \in O$, the algorithm examines all the VNFs (subfunctions) of option o (see lines 8-9 in Algorithm 4).

In the following, we explain how we assign a potential hosting node based on score calculation. Initially, the algorithm identifies a list N_b of potential hosting nodes, which satisfy the given constraints (line 10 in Algorithm 4). Next, we calculate the each potential hosting node belonging to set N_b, and then select the node with the smallest score (lines 11-13 in Algorithm 4). Next, node n_b with the smallest score is then assigned for exploring VNFs (line 14 in Algorithm 4). We note

Algorithm 4: Cost Aware VNF Decomposition (CA-VNF-D)

Input: $G(N, L)$, src, dst, DC
Output: $VNF\text{-}FG$
Initialize: Tree graph (src,dst)

1 **Function** `DecompositionSelection`(G, src, dst, DC):
2 $\quad start_{value} = src$
3 $\quad end_{value} = Ł_1$
4 $\quad VNF\text{-}FG = \emptyset$
5 $\quad S_d = 0$
6 $\quad$ **while** $end_{value} \leq Ł_t$ **do**
7 $\quad\quad O \leftarrow$ Find all options from $start_{value}$ to end_{value}
8 $\quad\quad$ **for** *each* $o \in O$ **do**
9 $\quad\quad\quad$ **for** *each* $k \in o$ **do**
10 $\quad\quad\quad\quad N_b \leftarrow$ List of potential hosting nodes
11 $\quad\quad\quad\quad$ Calculate S_d for N_b
12 $\quad\quad\quad\quad S_{d_{cur}} \leftarrow$ Select the smallest S_d
13 $\quad\quad\quad\quad n_b \leftarrow$ The node with the smallest S_d
14 $\quad\quad\quad\quad$ assign n_b to k
15 $\quad\quad\quad\quad start_{value} = n_b$
16 $\quad\quad\quad\quad S_d \leftarrow S_d + S_{d_{cur}}$
17 $\quad\quad\quad$ **end**
18 $\quad\quad\quad$ Assign S_d to o
19 $\quad\quad$ **end**
20 $\quad$ **end**
21 $\quad$ Sort O in an ascending order of S_d
22 $\quad o_b \leftarrow$ Select the best o with the smallest S_d
23 $\quad$ Add o_b to Tree graph
24 $\quad end_{value} = Ł + 1$
25 $VNF\text{-}FG \leftarrow$ Tree graph
26 **return** $VNF\text{-}FG$

that although n_b is identified as a potential hosting node, it will not necessarily be selected as the actual hosting node for VNF embedding. Then the $start_{value}$ and total score S_d are updated (see lines 15-16 in Algorithm 4).

For each option o, the total score S_d is assigned to o (see line 18 in Algorithm 4). After computing the scores for all options of a specific VNF type (corresponding to a level of the tree), the algorithm proceeds to select the option with the smallest score and eliminates all tree paths except for the one that comprises the selected node. More specifically, the option with the lowest score, denoted as o_b, is chosen. Then, option o_b is added to the tree graph (lines 21-23 in Algorithm 4). Next, we move to the next level of the tree by updating end_{value} (line 24 in Algorithm 4). These

steps are executed for each VNF type of NS in the tree. Finally, we identify the path from source to destination that has the smallest score. This process continues until all paths between the last option and the destination have been assessed, where end_{value} becomes smaller than the number of levels in the tree, indicating that all potential paths have been examined. The resultant tree graph will have exactly one child node per parent node, thus forming a chain of VNFs. The algorithm then returns the resulting tree as the VNF-FG (lines 25-26 of Algorithm 4)).

To better understand different steps of our proposed CA-VNF-D algorithm, let us consider the network topology in Fig. 5.1.d and NS 2 in Fig. 5.1.a. To find potential VNF-FGs, we first initialize a tree graph showing all possible VNF-FGs for NS 2 (see Fig. 5.1.b). In this example, src is Node_1 and dst is Node_3, as shown in Fig. 5.2.a. Next, we explore all the options between src and the first level of the tree. In the first iteration, two options exist: src to $O_{A,1}$ and src to $O_{A,2}$. For each option, we assign a potential hosting node (n_b) and calculate the corresponding S_d. Let us now consider $O_{A,1}$, which consists of VNFs D and E (see Fig. 5.1.c). For each VNF within $O_{A,1}$, we calculate its respective S_d and find a potential hosting node (n_b). Firstly, we must determine N_b for VNF D. N_b for VNF D include Node 2 and Node 3. Given that Node 1 does not have enough capacity, it cannot be part of N_b. Subsequently, we calculate S_d for each node in N_b. For VNF D, $S_{d_D}^{Node2} = 7$ with $S_l = 1$, $S_f = 4$, and $S_v = 2$. Similarly, $S_d^{Node3} = 8$ with $S_l = 2$, $S_f = 4$, and $S_v = 2$. The algorithm selects the node with the smallest value of S_d. For VNF D, Node 2 has a smaller S_d compared to Node 3. Therefore, Node 2 is selected as n_b for VNF D. This process continues for all the VNFs within $O_{A,1}$. For VNF E, N_b =[Node 2, Node 3], and $S_{d_E}^{Node2} = 14$, while $S_{d_E}^{Node3} = 15$. The smallest S_d is $S_d^{Node2} = 14$ for VNF E. To obtain S_d of a given VNF, we sum the scores of all the preceding VNFs. For VNF E, it is the sum of S_{d_E} for E and S_{d_D} for VNF D. Therefore, $S_d^{O_{A,1}} = 13$, and Node 2 is assigned as n_b for $O_{A,1}$.

The hosting node (n_b) for the last VNF of $O_{A,1}$ is considered as n_b for the entire $O_{A,1}$. In this case, Node 2 is selected as n_b for VNF E. Thus, Node 2 is considered as n_b for $O_{A,1}$. This process continues for the remaining options at the current level. For $O_{A,2}$, the algorithm identifies N_b and selects the node with the smallest S_d. $O_{A,2}$ consists of VNF F, with N_b =[Node 2, Node 3]. For VNF F, $S_{d_F}^{Node2} = 7$, and $S_{d_F}^{Node3} = 8$. The hosting node (n_b) is Node 2, and $S_d^{O_{A,2}} = 7$. We eliminite all the paths except the one with the smaller decomposition score. In this case, the selected

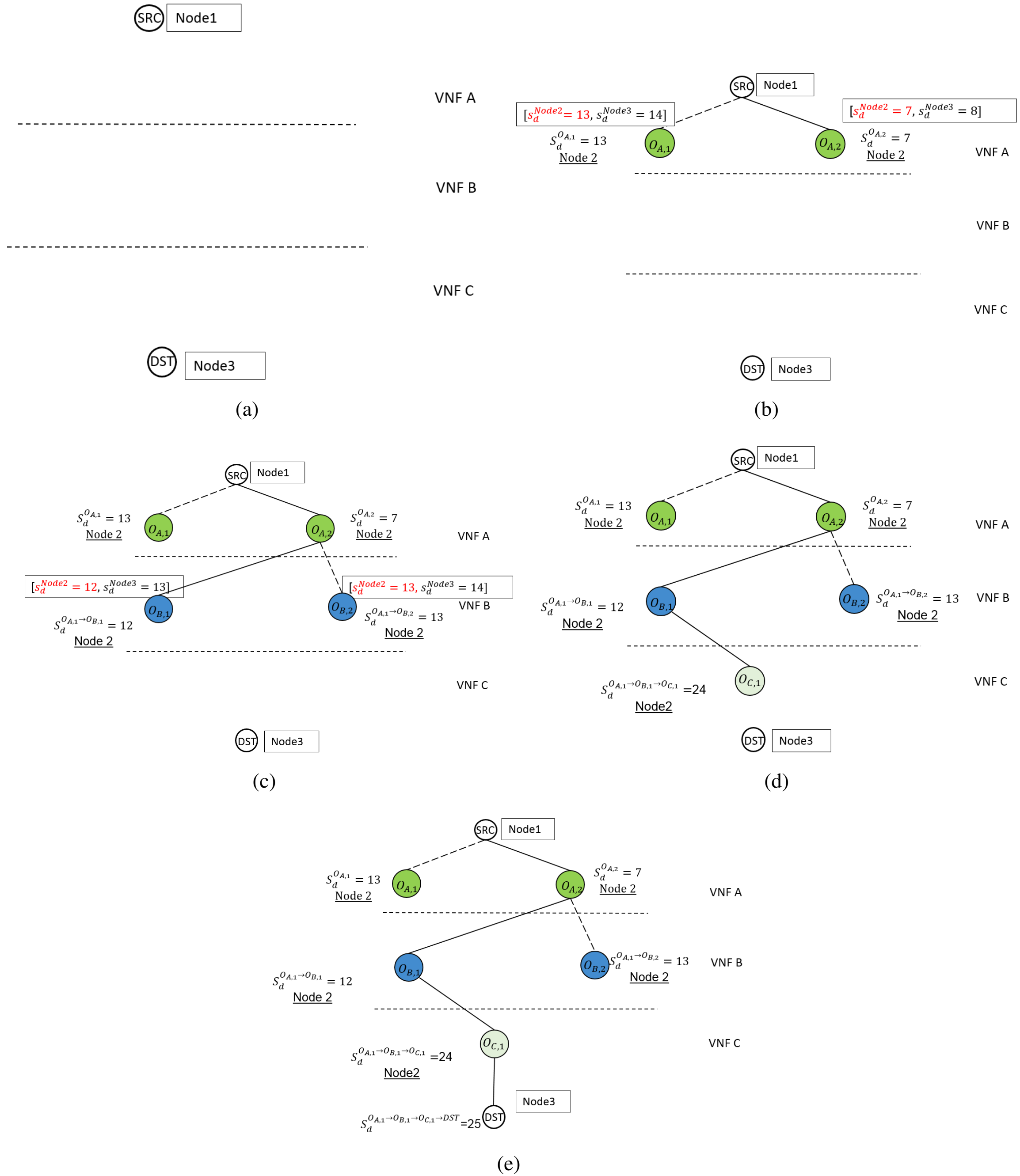


Figure 5.2: Illustrative examples for each step of Cost-Aware VNF Decomposition (CA-VNF-D) algorithm.

option is $O_{A,2}$, as shown in Fig. 5.2.b. Next, $start_{value}$ and end_{value} of the tree will be updated. The start value is Node 2 from $O_{A,2}$, and end_{value} is the next level of the tree, which is $l = 2$ of the tree graph. As shown in Figs.5.2.c-d, this process continues for the remaining VNFs of NS 2 until we reach dst. In this example, $S_d^{O_{A,2} \rightarrow O_{B,1} \rightarrow O_{C,1}} = 19$ has the smallest score among the paths in the graph tree, which corresponds to the VNF-FG comprising $O_{A,2} \rightarrow O_{B,1} \rightarrow O_{C,1}$.

5.4.2 Node Aware VNF Migration (NA-VNF-M)

Algorithm 2 presents the pseudo-code of the proposed Node-Aware VNF Migration (NA-VNF-M) algorithm. In the migration strategy, the VNF-FG obtained from CA-VNF-D is embedded into the network. If the embedding cost can further be improved via migration, we migrate VNFs from overloaded nodes to other nodes. To accomplish this, we start by initializing a network, considering both loaded network G_l and unloaded network G_u (see line 3 in Algorithm 2). We first consider the loaded network G_l followed by the unloaded network G_u (see lines 4-5 in Algorithm 5). By doing so, two VNF-FGs along with their respective ranking scores and the assigned hosting nodes for each VNF are obtained. Let the VNF-FG and score derived from G_l be denoted as VNF-FG$_l$ and S_{d_l}, respectively. Similarly, the VNF-FG and score derived from G_u are denoted as VNF-FG$_u$ and Sd_u, respectively. If the VNF-FGs obtained from G_l and G_u are identical and S_{d_l} is smaller than or equal to S_{d_u}, the VNFs within VNF-FG$_l$ are embedded into the network using the Sequential Greedy (SG) algorithm [8]. We note that SG is a greedy approach, which embeds the VNFs in a sequential manner (see line 6 in Algorimthm 5). Conversely, if the obtained VNF-FGs from G_l and G_u are not identical and if S_{d_l} is greater than S_{d_u}, the migration process is initiated. This may necessitate the eviction of certain VNFs to create room for embedding VNF-FG$_u$ into a network.

For doing the migration, for each VNF, in VNF-FG$_u$, the algorithm finds a list of candidate nodes denoted as N_c, on which the VNF can be embedded (based on the given constraints, e.g., latency, capacity), and if n_b (the potential hosting node designated to each VNF after running the decomposition algorithm in Algorithm 4) is included in N_c, the algorithm embeds VNF k using SG (see lines 8-12 in Algorithm 5); otherwise, if n_b is not included in N_c and the same VNF type k is hosted in note n_b, it means the node n_b does not have enough capacity and we create a list MIG of migrating VNFs, which is a list of potential VNFs that can be migrated to other nodes (see line

Algorithm 5: Node-Aware VNF Migration (NA-VNF-M)

```
1  input: G(N, L)network information, NS;
2  output: Embedding an NS;
3  Initialize: G_l, unloaded network G_u, tree graph;
4  VNF-FG_l ← DecompositionSelection(G_l, src, dst, DC)  VNF-FG_u ←
     DecompositionSelection(G_u, src, dst, DC);
5  if VNF-FG_l == VNF-FG_u  &  S_{d_l} ≤ S_{d_u} then
6  |   Embed VNF-FG_l to the network by using SG algorithm [8];
7  else
8  |   for each k ∈ VNF-FG_u do
9  |   |   N_c ← Find a list of candidate nodes;
10 |   |   if n_b ∈ N_c potential hosting node is in the list of candidate node then
11 |   |   |   Embed k by using SG algorithm [8]
12 |   |   end
13 |   |   else
14 |   |   |   MIG ← Find a list of migrating VNFs;
15 |   |   |   MIG_s ← Sort the MIG based on Order of Migration;
16 |   |   |   MIG_r ← Refine MIG_r based on the migration order;
17 |   |   |   relocate MIG_r;
18 |   |   |   Embed k to n_b;
19 |   |   end
20 |   end
21 end
```

14 in Algorithm 5). The list MIG comprises all embedded VNFs located on node n_b, excluding the preceding VNFs of the VNF k. After that, the algorithm needs to sort the list MIG based on the order of migration denoted as MIG_s (see line 15 in Algorithm 5). The order of migration starts from the last NS and VNF, which is embedded in the assigned node and continues the reverse order of embedding of VNFs on the assigned node. If the VNF is reused by more than one NS in list MIG, all the subsequent VNFs of NSs that reused that type of VNF need to be examined and the algorithm should start from the last VNF of that NS, which is in list MIG. This process continues until all the VNFs of list MIG are sorted. Next, the algorithm refines list MIG_s (see line 16 in Algorithm 5) based on the capacity demand of the current VNF. The refined ignore list MIG_r can include a VNF or multiple VNFs based on the capacity demand for a VNF that needs to be embedded in node n_b. The VNFs of list MIG_r need to be evicted from node n_b to a new node. using the SG algorithm. To do so, the VNFs in MIG_r are embedded to the node, which is associated with the smallest cost. Once this is done, VNF k is embedded in the network using SG (see lines 17-18 Algorithm 5). The process ends once all the VNFs of the VNF-FG are embedded.

To better understand the proposed ranking strategy, let us consider NS 1, NS 2, and NS 3 shown in Fig. 5.3.a-c. Let us assume that NS 1 and NS 2 are already deployed in the network as depiceted in Fig. 5.3.d, where Node 1 comprises all the VNFs of NS 1 and NS 2. Let us also assume that after considering the overloaded network and running Algorithm 4, NS-3 is the returned VNF-FG. Node n_b for each VNF of NS 3 is also shown in Fig. 5.3.c. As an example, node n_b for VNF K of NS 3 is Node 1. The capacity demand of VNFs of NS 3 is also shown in Fig. 5.3.c. As an example, F requires a fixed capacity f_k of 4 and a variable capacity v_k of 30. In this example, it is assumed the score of VNF-FG$_u$ is smaller than VNF-FG$_l$, which indicates that a migration is needed.

As shown in Figs. 5.3.a-d, let us assume that all the VNF types of NS 1 and NS 2 require 4 fixed and 2 variable capacities. The algorithm starts with the first VNF of NS 3, which is F. First, the algorithm finds a list of candidate nodes for VNF F. The list of candidate nodes for F is Node 2 and Node 3. As Node 1 does not have enough capacity (as shown in Fig. 5.3.d), it cannot be part of the list of candidate nodes. Next, we examine whether or not node n_b (which is Node 1) is part of the list of candidate nodes (N_c). Then, we examine whether or not Node 1 hosts VNF type F. As VNF type F is already deployed in Node 1, then the algorithm needs to create the migrating list MIG for VNF F. List MIG includes all the VNF residing in Node 1. Therefore, the migrating list for VNF F includes a list of [X, Y, L, J, K, F]. List MIG is then sorted in a descending order based on the embedding in the network.

In our example, the last NS (which is embedded in Node 1) is NS 2 and the last VNF of NS 2 is VNF X. VNF X is added to the sorted ignore list MIG_s. Next, we consider the second last VNF, which is VNF L. Given that VNF L is already reused by NS 1, before adding L to the sorted ignore list, all the VNFs after VNF L in NS 1 need to be added to MIG_s, starting from the last VNF in NS 1. The last VNF in NS 1 is VNF J. Therefore, VNF J is added to MIG_s. Then, VNF L is added to MIG_s followed by VNF Y from NS 2, and VNFs K and F from NS 1. Thus, MIG_s for F of NS 3 is an ordered list comprising VNFs X, J, L, Y, K, and F. Next, we refine MIG_s based on the capacity demand of VNF F. The variable capacity demand of F is 30, as shown in Fig. 5.3.c. Thus, MIG_r includes [X, J, L, Y]. Next, all the VNFs of MIG_r are migrated to Node 2, and F is also embedded in Node 1 (see Fig. 5.3.e). We then consider VNF K of NS 3. Node n_b for VNF K is Node 1, but VNF K in the previous step was migrated to Node 2. Thus, Node 2 is considered

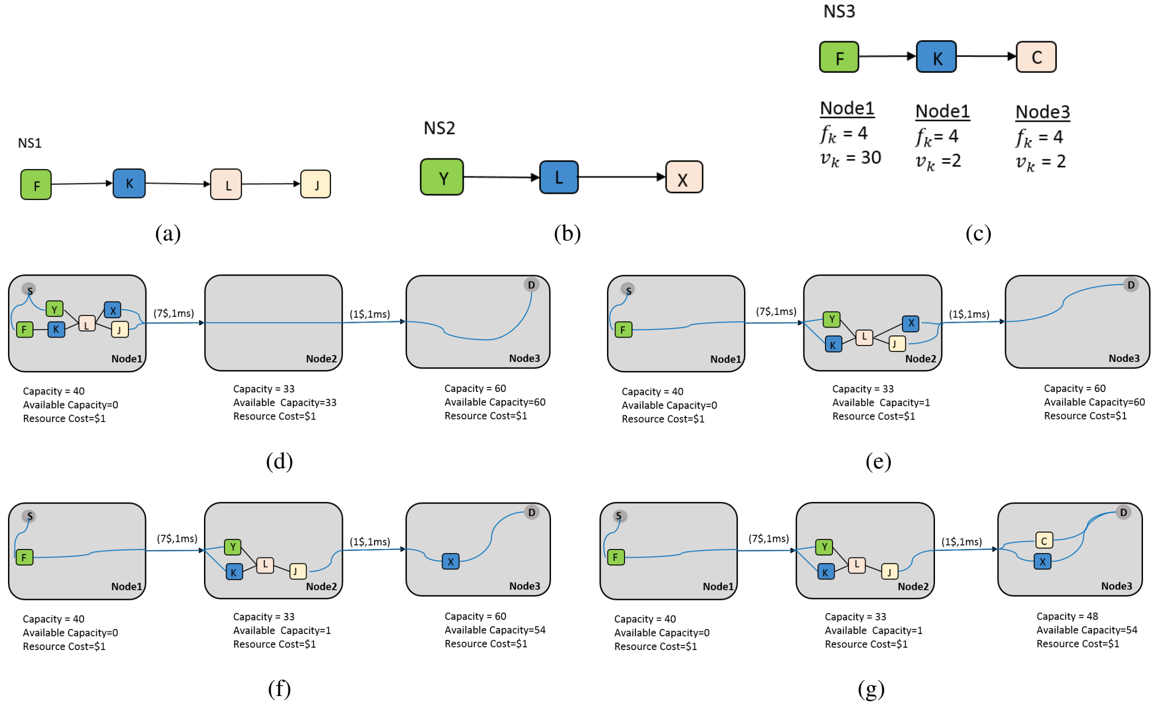


Figure 5.3: Illustrative examples for Node Aware VNF Migration (NA-VNF-M) Algorithm

as node n_b for VNF K, and a list of candidate nodes for VNF K is obtained. The list of candidate nodes for VNF K is Node 3 because there is a VNF type K in Node 2. However, Node 2 does not have enough capacity to serve VNF K. Thus, MIG for VNF K needs to be created to include all the VNFs that are embedded on Node 2, i.e., VNFs X, L, Y, and J. List MIG_s for VNF K is therefore VNFs X, J, L, and Y. Based on the capacity demand of VNF type K, we refine list MIG_s. The variable capacity of VNF K in NS 3 is 2. Thus, list MIG_r includes only VNF X. VNF X is migrated to Node 3 and the VNF K is embedded in Node 2 (see Fig. 5.3.f). This process continues for the other VNFs of NS 3 until all the VNFs are embedded in the network (see Fig. 5.3.g).

5.4.3 Asymptotic Analysis

To provide insights into the behavior of our proposed algorithm as the input size grows, we conduct an asymptotic analysis. We show that the proposed algorithm demonstrates asymptotic optimality, signifying that the embedding cost achieved through migrations closely approaches the optimal solution as the size of the input grows large. Let R denote the number of incoming requests.

Also, the length of the VNF chain associated with request $r \in R$ is denoted by L. For analytical tractability, we assume that all the incoming requests have an equal number of VNFs with the same length L. The number s of requests that fit into a single node can be estimated as follows:

$$s = \lfloor \frac{(c_n - f_k)}{v_k} \rfloor, \tag{5.20}$$

The number of nodes that are fully occupied (meaning a node that is fully occupied cannot accept any other VNFs to host on it)is denoted as g which is calculated by:

$$g = \lfloor \frac{R}{s} \rfloor, \tag{5.21}$$

The number of requests in the latest node (to be filled up) is denoted by q which is calculated by:

$$q = R \bmod s, \tag{5.22}$$

Therefore, the total CPU usage ($Total_{cpu}$) from the request 1 to R is calculated by:

$$Total_{cpu}(R) = L \cdot (g \cdot (c_n + s \cdot v_k) + (\left\lceil \frac{R}{s} \right\rceil - g) \cdot f_k + q \cdot v_k). \tag{5.23}$$

where $(\lceil \frac{R}{s} \rceil - g)$ is either 1 or 0. More specifically, it is 0 if the last node has enough capacity to be filled up by the embedding of VNFs for request R, otherwise, it is 1.

5.5 Results and Discussions

In this section, we conduct simulations to evaluate the performance of our proposed algorithms against a decomposition-only approach. We first describe the simulation environment and then present our obtained results.

5.5.1 Simulation setup

Our results are obtained from an experiment conducted on the Ericsson testbed. Our testbed environment consists of two digital twins; a workflow engine for VNF embedding algorithms (a greedy sequential VNF embedding method), and Netconf/Yang-based configuration management application programming interfaces (APIs) [8]. The hardware specifications of our testing platform include a 1.4 GHz Quand Core Intel Core i5 and 8GB of RAM.

5.5.1.1 Settings

We consider a substrate network typology consisting of 11 nodes and 13 bidirectional links [104, 9]. Other parameter settings are those that were used in our previous works [9, 104]. We assume that the number of available vCPUs of each node varies between 20 to 65 [4], and the node, as a configurable parameter, is set to $2 per vCPU. In addition, each physical link has an available link bandwidth of 50 Gbps. The link cost and the link delay are configurable and set as [0-10]$ per Gbps and 1 ms, respectively [104, 3]. We also assume that the network can support 7 types of VNFs and sub-functions. Also, the value of the fixed capacity is set to [6-11] vCPUs and we set the variable capacity to [1-6] vCPUs. Also, we assume that some VNFs are already embedded in the network and nodes have limited available capacity. In the evaluation scenario, we consider a request with a chain of 3 VNFs, which need to be embedded in the substrate network. We consider 3 to 8 possible combinations of VNF decomposition options for the VNFs required by a given NS [104], as shown in Fig. 5.4. Each incoming request has a different source-destination pair, which is selected randomly. All the network parameters and default values are summarized in Table 5.2.

We compare our algorithm's results with the decomposition-only approach. In the decomposition-only approach, we select the best decomposition option by considering loaded networks. Subsequently, the obtained VNF-FG is sequentially embedded in a node with the lowest hosting cost, following SG algorithm [8], without considering any migration strategy.

Table 5.2: Parameter settings and default values.

Parameter	Value	Ref.
Number of nodes N	11	[9, 104]
Number of links L	14	[9, 104]
Available node capacity c_n	$[20 - 65]$ vCPU	[3, 4]
Node cost ω_n	2	[5, 4]
Available link bandwidth b_l	50 Gbps	[5, 4]
Link cost θ_l	\$2	[3, 4]
Link delay d_l	1 ms	[4, 9]
Number of required VNFs in a request K	[3-5]	[104]
Fixed capacity f_k	$[6 - 11]$ vCPU	[104, 4]
Variable capacity v_k	$[1 - 6]$ vCPU	[104, 4]

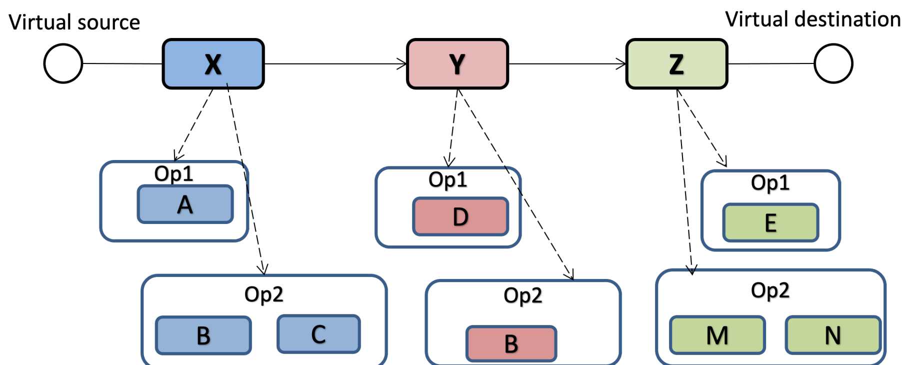


Figure 5.4: An NS with 3 main VNFs and all decomposition options.

5.5.2 Results

Asymptotic analysis: In Figs. 5.5 and 5.6, we assess the asymptotic behavior of the proposed algorithm. We set the transmission cost to zero and consider five VNFs per request. Figure 5.5 illustrates the embedding cost ratio (which is defined as the the embedding cost of the decomposition-only approach to the proposed algorithm) vs. the number of requests. This analysis considers nodes with varying CPU capacities of 20, 22, 33, and 55 (e.g., each node has an available capacity of 55 CPU units and all nodes have the same capacity). For all VNFs of a given request, we consider up to 500 requests and $f_k = 10$ and $v_k = 1$. Our results are based on the analysis presented in Section 5.4.3. A node capacity of 55 CPUs is the best-case scenario, where the proposed algorithm can achieve the maximum improvement of the embedding cost compared to the decomposition-only

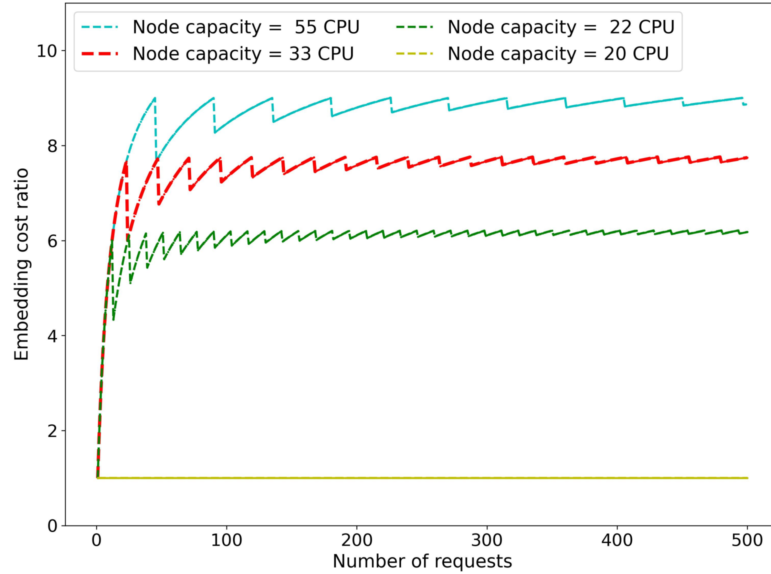


Figure 5.5: Ratio of embedding cost of Decomposition-only to embedding cost of the proposed method when (Fixed capacity, Variable capacity) = (10,1) for 500 requests for different nodes' capacity.

approach. In this best-case scenario, no VNFs in the decomposition-only option can be reused due to the fixed capacity demand of VNFs and limited node capacity. Additionally, a node capacity of 20 CPUs is considered as the worst-case scenario, where the embedding cost of the proposed method is the same as the decomposition-only approach. In the proposed method, no migration is needed due to the high fixed capacity demand for VNFs and the limited capacity of nodes. Each type of VNF can be embedded in a node and reused by other VNFs of the incoming requests. In the worst-case scenario, each node has an available capacity of 20 CPUs, where the proposed method cannot execute migrations due to node capacity limit. For each VNF type, we cannot add more than one type per node due to VNF fixed capacity and limited node capacity. Hence, both proposed method and decomposition-only approach exhibit a similar trend, and we can observe that the embedding cost ratio is 1. In the best-case scenario, achieving a maximum gain requires nodes with a capacity of 55 CPUs. In the decomposition-only approach, VNFs cannot be reused, as each request necessitates 55 CPU units. For each VNF in a new request, a new VNF instance must be instantiated, as the VNFs of a request fully occupy the node capacity. Conversely, in the proposed algorithm,

VNFs can be reused by migrating previously embedded VNFs to other nodes, thereby reducing the embedding cost. In the best-case scenario, the embedding cost ratio is 8. Interestingly, we observe that for the curves representing various CPU capacities (22, 33, and 55), the observed saw-tooth behavior in these three curves (22, 33, and 55) is primarily attributed to the migration strategy of the proposed method. Through migration and increased reuse of VNFs, the embedding cost decreases. Consequently, the embedding cost ratio increases. However, as the number of requests increases, the nodes become fully occupied, thus decreasing the chance of migration. This in turn leads to new instantiation of VNFs, which is associated with a higher embedding cost.

Additionally, we observe in Fig. 5.5 that the oscillations of the saw-tooth curve decreases as the number of requests increases, moving asymptotically towards a single line. This behavior primarily depends on the capacity of nodes to accommodate reused VNFs, ultimately affecting the embedding cost ratio. Further, as observed in Fig 5.5, the transition from 22 to 33 CPUs demonstrates a similar asymptotic behavior, while displaying different saw-tooth oscillations. Similarly, from 33 to 44 CPUs, there is asymptotic uniformity but with varying saw-tooth oscillations. Within these different ranges, a consistent lower bound is apparent. For instance, when considering 22 CPUs, the ratio gradually approaches the highest gain of 6.1 asymptotically. Within the range of 22-33 CPUs, the lower bound remains similar to that of 22 CPUs, resulting in a higher embedding cost ratio and a larger saw-tooth oscillation than 22 CPUs.

Figure 5.6 depicts the embedding cost ratio vs. number of requests for different VNF-FG size. We set the capacity of nodes to 55 CPUs, fixed capacity of VNFs to 10, and variable capacity of VNFs to 1. According to Fig. 5.6, the proposed method achieves a maximum improvement of the embedding cost compared to the decomposition-only approach for 5 VNFs. This is primarily due to the limitation of the decomposition-only approach, which mostly embeds new VNF instances in the network rather than reusing them. In contrast, in the proposed algorithm, VNFs are migrated from over-utilized nodes, thus allowing for VNF reusability. More interestingly, we observe from Fig. 5.6 that as the number of VNFs per request increases, the performance gain obtained by our joint migration and decomposition solution becomes even more highlighted compared to the decomposition-only approach.

Adaptiveness: Next, we assess the adaptiveness of the proposed algorithm. Specifically, we

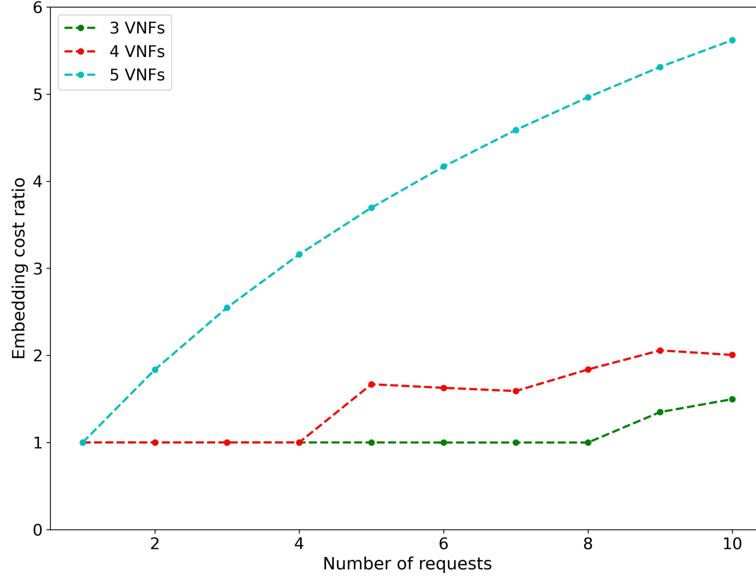
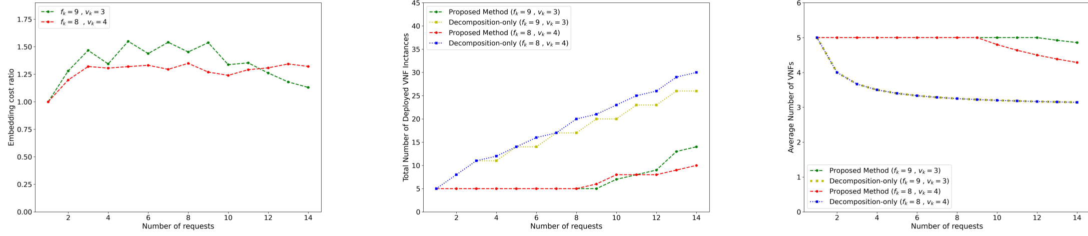


Figure 5.6: Embedding cost ratio vs. number of requests for (fixed capacity, variable capacity) = (10,1) for different values of VNF-FG size.

examine whether the proposed algorithm has the capability to adapt itself and selects the chain with the fewest VNFs when decomposition options do not have equal CPU demands. First, we set set $f_k = 9$, $v_k = 3$, and $f_k = 8$, $v_k = 4$, and consider a node capacity of 66 CPU. Figure 5.7.a depicts the embedding cost ratio vs. the number of requests. We can observe that as the number of requests grows from 1 to 2, the embedding cost ratio becomes greater than one, which indicates the embedding cost of the proposed method becomes smaller than that of the decomposition-only approach. This is because the proposed method instantiates a smaller number of deployed VNF instances in the network. Total number of deployed VNF instances vs. number of requests is shown in Fig. 5.7.b, where we observe that our proposed algorithm can reduce the number of deployed VNFs by 66% and 40% for $f_k = 8$, $v_k = 4$ and $f_k = 9$, $v_k = 3$, compared to the decomposition-only approach, respectively. Figure 5.7.c depicts the average number of VNFs vs. number of requests, in which the average number of VNFs is calculated based on the total number of newly instantiated VNFs through the networks, divided by the number of requests. We observe in Fig. 5.7.c exhibits adaptiveness when it reaches the limit of reusing VNFs from the longest chain within the



(a) Total embedding cost vs. Number of requests

(b) Total number of deployed VNF instances vs. number of requests

(c) Average number of VNFs vs. number of requests

Figure 5.7: Examining the adaptiveness behavior in scenarios where decomposition options do not have equal CPU demand (14 requests).

network. For $f_k = 8$ and $v_k = 4$ and 10 requests, the proposed method switches to the shorter chain. A similar transition occurs for $f_k = 9$ and $v_k = 3$, and 13 requests. Interestingly, for $v_k = 3$, the transition to the smallest chain happens at a larger number of requests compared to $v_k = 4$. This is because for $v_k = 3$, VNFs require less capacity for reusing and the nodes reach their capacity limit at higher loads compared to $v_k = 4$.

In Fig. 5.8 we evaluate the impact of an increased capacity demand on the embedding cost ratio for various chain capacities of 30, 40, 50, and 60 for 10 requests. In this result, all options have an equal CPU demand. We can observe from the figure that the embedding cost ratio for any given capacity demand is larger than one, indicating that the proposed method outperforms the decomposition-only approach in terms of embedding cost. Moreover, we observe that as the capacity demand of VNFs increases, the performance gain also increases. This difference is most noticeable in the gap between the embedding cost of the decomposition-only approach and the proposed method. In the decomposition-only approach, with an increase in the amount of capacity demand, more VNFs are newly instantiated throughout the network. This necessitates higher capacity demand, leading to an increase in the total embedding cost. In contrast, the proposed method involves migrating VNFs, allowing for more VNF reuse and a reduced amount of required capacity demand.

Impact of transmission cost on the average embedding cost: Figure 5.9 shows the average embedding cost and average number of deployed VNF instances for 10 requests and 5 VNFs per request, and a node capacity of 55 CPUs. We note that the average embedding cost can be broken

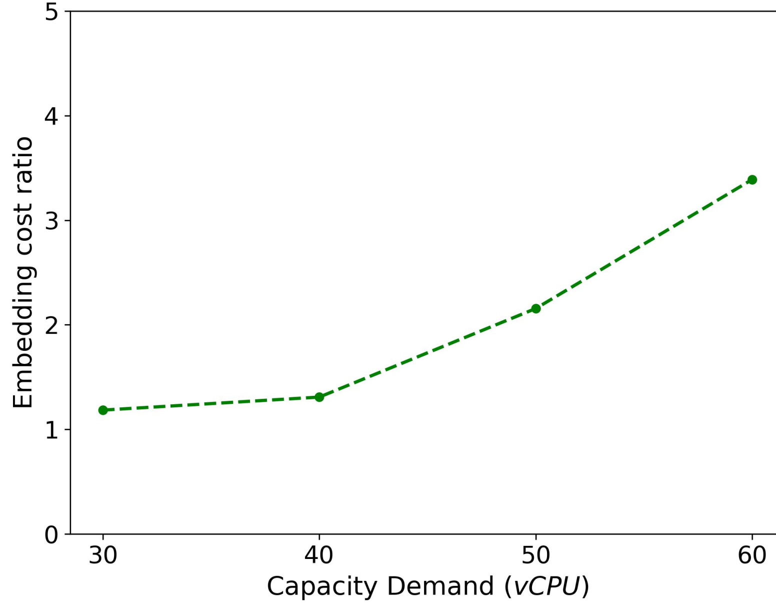
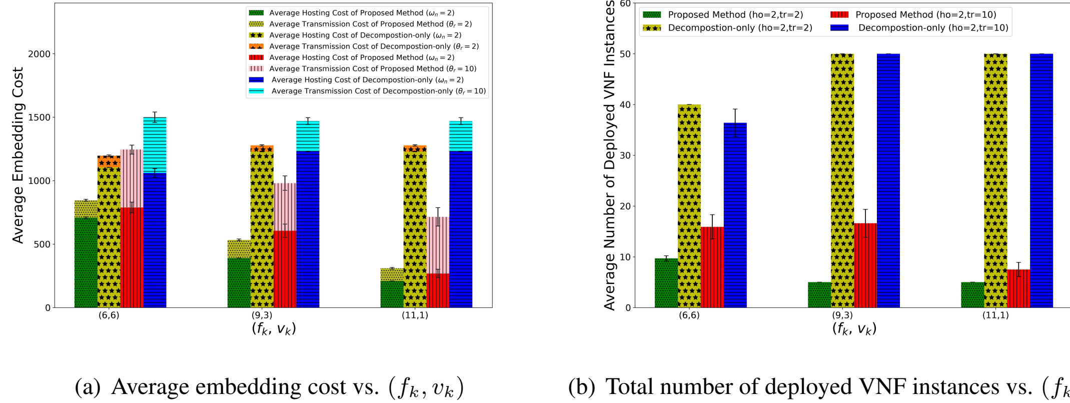


Figure 5.8: Embedding Cost ratio vs. capacity demand for 10 requests (all options have an equal CPU demand).

down to hosting cost (fixed and variable capacity costs) and transmission cost. Figure 5.9.a illustrates the average embedding cost for different values of f_k and v_k as stacked bar chart. The bottom of each bar represents the average hosting cost, while the top one represents the average transmission cost. In these results, we consider two scenarios with different values of transmission costs: $\theta_r = 2$ and $\theta_r = 10$. Throughout both scenarios and for various values of f_k and v_k, we set the hosting cost to $\omega_n = 2$ for all nodes. The results are accompanied with 99% confidence intervals. We observed that in the decomposition-only approach, by increasing the value of f_k from 6 to 11, the average hosting cost increases due to an escalated capacity demand for newly instantiated VNFs. Given that the decomposition-only approach does not have any migration, it requires more capacity for instantiating VNF instances in the network. Conversely, within the proposed method, increasing f_k from 6 to 11 results in a decrease of the average hosting costs. This reduction is primarily attributed to fewer deployed VNF instances being instantiated across the network, while a greater number of VNFs are reused, benefiting from increased fixed capacity as it shows in Fig 5.9.b. Furthermore, in the proposed method, the average transmission cost for both $\theta_r = 2$ and $\theta_r = 10$ is slightly higher



(a) Average embedding cost vs. (f_k, v_k) (b) Total number of deployed VNF instances vs. (f_k, v_k).

Figure 5.9: Breakdown of embedding cost for 10 requests and 5 VNFs per request.

compared to the decomposition-only approach. This happens because reused VNFs may be located far from the source and destination, consequently leading to an increased transmission cost. Despite this increase in transmission costs, the potential decrease in hosting costs through VNF reuse makes it more cost-efficient to embed VNFs even at a greater distance from the source and destination.

5.6 Conclusions

In this chapter, we studied the problem of joint VNF decomposition and migration with the main objective of minimizing the embedding cost by promoting VNF reusability. We proposed a cost model, which includes fixed capacity, variable capacity, and transmission cost. The objective is to select the optimal VNF decomposition options and migrate the previously deployed VNFs from overloaded nodes. After formulating the joint problem of VNF decomposition and migration as an ILP, we proposed two algorithms to solve the problem. Extensive simulations were conducted to evaluate the performance of our proposed algorithms, considering the asymptotic behavior and adaptiveness of the proposed algorithm. Simulation results demonstrate that our proposed algorithms can reduce the embedding cost by 62% and the number of deployed VNF instances by 60% compared to a decomposition-only approach. An interesting direction for future work would be to explore the problem of joint VNF decomposition and migration by considering more complex VNF-FG typologies such split and split-and-merge. Additionally, another interesting research avenue is

to explore the impact of dynamic traffic on migration strategies for a network with mobile users.

Chapter 6

Conclusions and Future Works

6.1 Conclusions

The VNF embedding may also be dynamic due to changes in the network configuration. This adds an additional layer of complexity in terms of keeping track of where a given VNF is running. In other words, the VNF migration problem generally refers to the process of migrating VNFs from one node to another to achieve objectives such as load balancing, cost reduction, energy saving, and recovery from failures. Making decisions about where, when, and how to transfer the VNFs from one node to another in response to variations in service requests is challenging. Especially when facing the following challenges: Firstly, when there is mobility among end-users and fog nodes, coupled with limited fog node coverage, this results in service discontinuity and increased application delay. Secondly, stringent latency requirements between VNFs can tightly couple the VNFs, hindering the individual migration of each VNF. Thirdly, an overloaded node can significantly impact the determination of the best VNF decomposition option among all possible choices, consequently degrading QoS. This thesis proposes algorithmic solutions to address these issues.

For the problem of application component migration in an NFV-based hybrid cloud/fog environment by considering mobile end-user and fog nodes, in chapter 3, we proposed a deep Q-learning approach to minimize the overall cost and delay. The cost consists of power consumption and migration costs, and the delay consists of processing, communication, and migration delay. The simulation results revealed that our method achieves up to 61% and 56% of improvement in cost

and delay compared to the existing methods.

In Chapter 4, we addressed the problem of migrating a cluster of VNFs while considering straight latency requirements between VNFs. Our study focused on reusing the already deployed VNFs. The objective was to determine a cluster of VNFs and migrate this cluster of VNFs so that the total embedding cost is minimized. We proposed two variants of VNF cluster migration algorithms, which allow migrating a cluster of VNFs to single and multiple destinations. Our results indicate that the proposed algorithm can achieve up to a 14% improvement in total embedding cost compared to the existing benchmarks, which comes at the expense of a 60% increase in execution time.

The focus of our study in Chapter 5 involved considering migration strategies while assuming each VNF could have multiple decomposition options. In this chapter, our goal is to select the best decomposition option in situations where nodes are overloaded. Our research delves into the problem of joint VNF decomposition and migration by considering the reuse of already deployed VNFs. We mathematically model the problem as an ILP aimed at minimizing the embedding cost. This cost includes fixed capacity, variable capacity, and transmission costs. To address this, we introduced two algorithms to select the best VNF decomposition option by migrating previously deployed VNFs from overloaded nodes. This strategy aimed to minimize the embedding cost while promoting VNF reusability. The simulation results demonstrate that our proposed algorithms can improve the embedding cost of the decomposition-only approach by 62% and reduce the number of new instances by 60%.

6.2 Future Works

This thesis presented a significant contributions on VNF migration. However, there still exist some future research directions in this area.

6.2.1 Application Component Migration

As future research, the work presented in Chapter 3 could be extended by exploring application component migration in conjunction with load balancing within hybrid cloud/fog environments.

Other aspects to consider is interoperability and federation of fog nodes to minimize the latency further. From a failure management perspective, it may be interesting to deal with the failure of often distributed fog nodes by means of designing efficient fault-tolerant migration techniques. Another interesting research avenue is to consider the resource allocation and scheduling in conjunction with the migration of VNFs in hybrid cloud/fog environments

6.2.2 VNF Cluster Migration

An interesting direction for future work would be to explore the problem of complex migration. This involves considering more sophisticated patterns of VNF migration, allowing for the eviction of VNFs of an NS from a specific node and the migration of a cluster of VNFs from another NS to that specific node. Another viable research avenue would be to extend our proposed algorithm in Chapter 4 by considering the situation in which multiple requests arrive in the system at one time and decide the order of cluster migration by considering the highest affinity requests. Furthermore, complex topologies such as linear chains, splits, and split-and-merged configurations [105] may need to be taken into account when determining clusters of VNFs for a given VNF forwarding graph. This presents an intriguing research direction to build upon this work.

6.2.3 Joint VNF Decomposition and Migration

Future research could explore the challenge of joint VNF decomposition and migration, particularly in the context of complex network topologies like linear chains, splits, and split-and-merged configurations. Additionally, investigating the impact of dynamic traffic on migration strategies presents another promising avenue for research in this field. This can include examining migration strategies for overloaded links and their influence on the selection of decomposition options.

Bibliography

[1] S. N. Afrasiabi, S. Kianpisheh, C. Mouradian, R. H. Glitho, and A. Moghe, "Application components migration in NFV-based hybrid cloud/fog systems," in *Proc. IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pp. 1–6, 2019.

[2] S. N. Afrasiabi, A. Ebrahimzadeh, C. Mouradian, S. Malektaji, and R. H. Glitho, "Reinforcement learning-based optimization framework for application component migration in NFV cloud-fog environments," *IEEE Transactions on Network and Service Management*, 2022.

[3] S. N. Afrasiabi, A. Ebrahimzadeh, C. Mouradian, W. Li, A. Recse, R. Szabo, and R. H. Glitho, "Cost-efficient cluster migration of vnfs for virtualized network function forwarding graph embedding," *IEEE Transactions on Network and Service Management*, 2022.

[4] S. N. Afrasiabi, A. Ebrahimzadeh, A. Azhdari, R. Szabo, C. Mouradian, W. Li, and R. Glitho, "Joint VNF decomposition and migration for cost-efficient VNF forwarding graph embedding," in *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, 2023.

[5] S. N. Afrasiabi, A. Ebrahimzadeh, C. Mouradian, R. Szabo, and R. H. Glitho, "Cost-efficient cluster migration of vnfs for virtualized network function forwarding graph embedding," *IEEE Transactions on Network Science and Engineering*, 2022.

[6] C. Mouradian, S. Kianpisheh, M. Abu-Lebdeh, F. Ebrahimnezhad, N. T. Jahromi, and R. H. Glitho, "Application component placement in NFV-based hybrid cloud/fog systems with mobile fog nodes," *IEEE Journal on Selected Areas in Communications*, vol. 37, pp. 1130–1143, May 2019.

[7] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[8] A. Recse, R. Szabo, and B. Nemeth, "Elastic resource management and network slicing for IoT over edge clouds," in *Proc. ACM International Conference on the Internet of Things*, pp. 1–8, 2020.

[9] A. Ebrahimzadeh, N. Promwongsa, S. N. Afrasiabi, C. Mouradian, W. Li, Recse, R. Szabó, and R. H. Glitho, "h-horizon sequential look-ahead greedy algorithm for VNF-FG embedding," in *Proc. IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 41–46, 2021.

[10] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De 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, 2016.

[11] 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, 2016.

[12] L. Gupta, M. Samaka, R. Jain, A. Erbad, D. Bhamare, and C. Metz, "COLAP: A predictive framework for service function chain placement in a multi-cloud environment," in *Proc. IEEE Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 1–9, 2017.

[13] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 416–464, 2018.

[14] N. ETSI, "Network functions virtualisation (NFV); terminology for main concepts in NFV," *Group Specification*, vol. 3, pp. 1–10, 2014.

[15] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 712–725, 2018.

[16] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, pp. 416–464, Firstquarter 2018.

[17] Q. Duan, Y. Yan, and A. V. Vasilakos, "A Survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing," *IEEE Transactions on Network and Service Management*, vol. 9, no. 4, pp. 373–392, 2012.

[18] Q.-V. Pham, F. Fang, V. N. Ha, M. J. Piran, M. Le, L. B. Le, W.-J. Hwang, and Z. Ding, "A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art," *IEEE Access*, vol. 8, pp. 116974–117017, 2020.

[19] J. Li, W. Shi, Q. Ye, W. Zhuang, X. Shen, and X. Li, "Online joint VNF chain composition and embedding for 5G networks," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2018.

[20] S. Sahhaf, W. Tavernier, M. Rost, S. Schmid, D. Colle, M. Pickavet, and P. Demeester, "Network service chaining with optimized network function embedding supporting service decompositions," *Computer Networks*, vol. 93, pp. 492–505, 2015.

[21] R. Szabo, M. Kind, F.-J. Westphal, H. Woesner, D. Jocha, and A. Csaszar, "Elastic network functions: opportunities and challenges," *IEEE Network*, vol. 29, no. 3, pp. 15–21, 2015.

[22] D. Li, P. Hong, W. Wang, and J. Pei, "Virtual network function placement with function decomposition for virtual network slice," in *Proc. IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 1–4, 2018.

[23] B. Yi, X. Wang, K. Li, M. Huang, *et al.*, "A comprehensive survey of network function virtualization," *Computer Networks*, vol. 133, pp. 212–262, 2018.

[24] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23511–23528, April 2018.

[25] Z. Rejiba, X. Masip-Bruin, and E. Marín-Tordera, "A survey on mobility-induced service

migration in the fog, edge, and related computing paradigms," *ACM Comput. Surv.*, vol. 52, sep 2019.

[26] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[27] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on Markov decision process," *IEEE/ACM Transactions on Networking*, vol. 27, pp. 1272–1288, May 2019.

[28] D. Zhao, T. Yang, Y. Jin, and Y. Xu, "A service migration strategy based on multiple attribute decision in mobile edge computing," in *Proc. IEEE International Conference on Communication Technology (ICCT)*, pp. 986–990, May 2017.

[29] L. Liang, J. Xiao, Z. Ren, Z. Chen, and Y. Jia, "Particle swarm based service migration scheme in the edge computing environment," *IEEE Access*, vol. 8, pp. 45596–45606, March 2020.

[30] Z. Gao, Q. Jiao, K. Xiao, Q. Wang, Z. Mo, and Y. Yang, "Deep reinforcement learning based service migration strategy for edge computing," in *Proc. IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 116–1165, May 2019.

[31] Q. Yuan, J. Li, H. Zhou, T. Lin, G. Luo, and X. Shen, "A joint service migration and mobility optimization approach for vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, pp. 9041–9052, June 2020.

[32] V. Eramo, M. Ammar, and F. G. Lavacca, "Migration energy aware reconfigurations of virtual network function instances in NFV architectures," *IEEE Access*, vol. 5, pp. 4927–4938, March 2017.

[33] J. Xia, D. Pang, Z. Cai, M. Xu, and G. Hu, "Reasonably migrating virtual machine in NFV-featured networks," in *Proc. IEEE International Conference on Computer and Information Technology (CIT)*, pp. 361–366, March 2016.

[34] J. Xia, Z. Cai, and M. Xu, "Optimized virtual network functions migration for NFV," in *Proc. IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 340–346, Jan 2016.

[35] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," *IEEE/ACM Transactions on Networking*, vol. 25, pp. 2008–2025, March 2017.

[36] L. Tang, X. He, P. Zhao, G. Zhao, Y. Zhou, and Q. Chen, "Virtual network function migration based on dynamic resource requirements prediction," *IEEE Access*, vol. 7, pp. 112348–112362, Aug. 2019.

[37] R. Chen, H. Lu, Y. Lu, and J. Liu, "MSDF: A deep reinforcement learning framework for service function chain migration," in *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, June 2020.

[38] D. Zhao, G. Sun, D. Liao, S. Xu, and V. Chang, "Mobile-aware service function chain migration in cloud-fog computing," *Future Generation Computer Systems*, vol. 96, pp. 591–604, July 2019.

[39] A. Yazidi, F. Ung, H. Haugerud, and K. Begnum, "Affinity aware-scheduling of live migration of virtual machines under maintenance scenarios," in *Proc. IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–4, 2019.

[40] J. Narantuya, H. Zang, and H. Lim, "Service-aware cloud-to-cloud migration of multiple virtual machines," *IEEE Access*, vol. 6, pp. 76663–76672, 2018.

[41] Narantuya, Jargalsaikhan and Zang, Hannie and Lim, Hyuk, "Automated cloud migration based on network traffic dependencies," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–4, 2017.

[42] T. Lu, M. Stuart, K. Tang, and X. He, "Clique migration: Affinity grouping of virtual machines for inter-cloud live migration," in *Proc. IEEE International Conference on Networking, Architecture, and Storage*, pp. 216–225, 2014.

[43] B. Yi, X. Wang, M. Huang, and A. Dong, “A multi-criteria decision approach for minimizing the influence of vnf migration,” *Computer Networks*, vol. 159, pp. 51–62, 2019.

[44] S. R. Chowdhury, Anthony, H. Bian, T. Bai, and R. Boutaba, “A disaggregated packet processing architecture for network function virtualization,” *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1075–1088, 2020.

[45] P. Wang, J. Lan, X. Zhang, Y. Hu, and S. Chen, “Dynamic function composition for network service chain: Model and optimization,” *Computer Networks*, vol. 92, pp. 408–418, 2015.

[46] S. Bian, X. Huang, Z. Shao, X. Gao, and Y. Yang, “Service chain composition with failures in nfv systems: A game-theoretic perspective,” in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2019.

[47] B. Spinnewyn, P. H. Isolani, C. Donato, J. F. Botero, and S. Latré, “Coordinated service composition and embedding of 5G location-constrained network functions,” *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1488–1502, 2018.

[48] M. Wang, B. Cheng, S. Zhao, B. Li, W. Feng, and J. Chen, “Availability-aware service chain composition and mapping in NFV-enabled networks,” in *Proc. IEEE International Conference on Web Services (ICWS)*, pp. 107–115, 2019.

[49] D. Moro, G. Verticale, and A. Capone, “A framework for network function decomposition and deployment,” in *Proc. IEEE International Conference on the Design of Reliable Communication Networks (DRCN)*, pp. 1–6, 2020.

[50] S. Sahhaf, W. Tavernier, M. Rost, S. Schmid, D. Colle, M. Pickavet, and P. Demeester, “Network service chaining with optimized network function embedding supporting service decompositions,” *Computer Networks*, vol. 93, pp. 492–505, 2015. Cloud Networking and Communications II.

[51] S. M. A. Araújo, F. S. H. de Souza, and G. R. Mateus, “A composition selection mechanism for chaining and placement of virtual network functions,” in *2019 15th International Conference on Network and Service Management (CNSM)*, pp. 1–5, 2019.

[52] D. Zheng, C. Peng, X. Liao, L. Tian, G. Luo, and X. Cao, "Towards latency optimization in hybrid service function chain composition and embedding," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 1539–1548, 2020.

[53] S. B. Chetty, H. Ahmadi, M. Tornatore, and A. Nag, "Dynamic decomposition of service function chain using a deep reinforcement learning approach," *IEEE Access*, vol. 10, pp. 111254–111271, 2022.

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

[55] L. Qu, C. Assi, M. J. Khabbaz, and Y. Ye, "Reliability-aware service function chaining with function decomposition and multipath routing," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 835–848, 2020.

[56] N. Mohamed, J. Al-Jaroodi, I. Jawhar, H. Noura, and S. Mahmoud, "UAVFog: A UAV-based fog computing for Internet of Things," in *Proc. IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pp. 1–8, June 2017.

[57] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Transactions on Services Computing*, vol. 12, pp. 712–725, Sep. 2018.

[58] C. Mouradian, F. Ebrahimnezhad, Y. Jebbar, J. K. Ahluwalia, S. N. Afrasiabi, R. H. Glitho, and A. Moghe, "An IoT platform-as-a-service for NFV-based hybrid cloud/fog systems," *IEEE Internet of Things Journal*, vol. 7, pp. 6102–6115, Jan. 2020.

[59] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. S. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Transactions on Mobile Computing*, vol. 20, pp. 939–951, Dec. 2021.

[60] Y. Cui, V. K. Lau, R. Wang, H. Huang, and S. Zhang, "A survey on delay-aware resource control for wireless systems—large deviation theory, stochastic lyapunov drift, and distributed stochastic learning," *IEEE Transactions on Information Theory*, vol. 58, pp. 1677–1701, March 2012.

[61] G. E. Monahan, "State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms," *Management Science*, vol. 28, pp. 1–16, Jan. 1982.

[62] Y. Zhai, Y. Wang, I. You, J. Yuan, Y. Ren, and X. Shan, "A DHT and MDP-based mobility management scheme for large-scale mobile internet," in *Proc. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 379–384, June 2011.

[63] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, pp. 3133–3174, May 2019.

[64] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, and F. Liberal, "Virtual network function placement optimization with deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, pp. 292–303, Dec. 2019.

[65] Y. Li, B. Shen, J. Zhang, X. Gan, J. Wang, and X. Wang, "Offloading in hcns: Congestion-aware network selection and user incentive design," *IEEE Transactions on Wireless Communications*, vol. 16, pp. 6479–6492, July 2017.

[66] M. Tokic and G. Palm, "Value-difference based exploration: adaptive control between epsilon-greedy and softmax," in *Proc. Springer Annual conference on Artificial Intelligence*, pp. 335–346, 2011.

[67] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *Proc. Workshop on Deep Learning, NIPS*, 2013.

[68] C. Wang, L. Ma, R. Li, T. S. Durrani, and H. Zhang, "Exploring trajectory prediction through machine learning methods," *IEEE Access*, vol. 7, pp. 101441–101452, July 2019.

[69] G. Rjoub, J. Bentahar, O. A. Wahab, and A. Bataineh, "Deep smart scheduling: A deep learning approach for automated big data scheduling over the cloud," in *Proc. IEEE International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 189–196, Jan 2019.

[70] M. Hassan, H. Chen, and Y. Liu, "Dears: A deep learning based elastic and automatic resource scheduling framework for cloud applications," in *Proc. IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pp. 541–548, March 2018.

[71] D. Yi, X. Zhou, Y. Wen, and R. Tan, "Toward efficient compute-intensive job allocation for green data centers: A deep reinforcement learning approach," in *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 634–644, 2019.

[72] Y. Hua, Z. Zhao, R. Li, X. Chen, Z. Liu, and H. Zhang, "Deep learning with long short-term memory for time series prediction," *IEEE Communications Magazine*, vol. 57, pp. 114–119, March 2019.

[73] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, pp. 157–166, March 1994.

[74] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2625–2634, 2015.

[75] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "TensorFlow: a system for large-scale machine learning," in *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 265–283, 2016.

[76] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. International Conference on Learning Representations*, pp. 1–13, Dec. 2015.

[77] I. Lera, C. Guerrero, and C. Juiz, "YAFS: A simulator for IoT scenarios in fog computing," *IEEE Access*, vol. 7, pp. 91745–91758, 2019.

[78] C. Qu, P. Calyam, J. Yu, A. Vandanapu, O. Opeoluwa, K. Gao, S. Wang, R. Chastain, and K. Palaniappan, "DroneCOCoNet: Learning-based edge computation offloading and control networking for drone video analytics," *Elsevier Future Generation Computer Systems*, vol. 125, pp. 247–262, 2021.

[79] D. A. Korneev, A. V. Leonov, and G. A. Litvinov, "Estimation of mini-UAVs network parameters for search and rescue operation scenario with Gauss-Markov mobility model," in *Proc. IEEE Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*, pp. 1–7, 2018.

[80] S. K. Maakar, Y. Singh, and R. Singh, "Implementation of three dimensional model for flying Ad Hoc network," in *Proc. IEEE International Conference on Intelligent Communication and Computational Techniques (ICCT)*, pp. 303–307, 2019.

[81] S. Qazi, A. S. Siddiqui, and A. I. Wagan, "UAV based real time video surveillance over 4G LTE," in *Proc. IEEE International Conference on Open Source Systems Technologies (ICOSST)*, pp. 141–145, 2015.

[82] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Elsevier Performance Evaluation*, vol. 91, pp. 205–228, 2015.

[83] C. Song, T. Koren, P. Wang, and A.-L. Barabási, "Modelling the scaling properties of human mobility," *Nature physics*, vol. 6, no. 10, pp. 818–823, 2010.

[84] S. Roy, N. Ghosh, P. Ghosh, and S. K. Das, "BioMCS: A bio-inspired collaborative data transfer framework over fog computing platforms in mobile crowdsensing," in *Proc. ACM International Conference on Distributed Computing and Networking (ICDCN)*, pp. 1–10, 2020.

[85] D. Broyles, A. Jabbar, J. P. Sterbenz, *et al.*, "Design and analysis of a 3–D Gauss-Markov mobility model for highly-dynamic airborne networks," in *Proc. International Telemetering Conference (ITC)*, pp. 25–28, Oct. 2010.

[86] M. McGuire, "Stationary distributions of random walk mobility models for wireless ad hoc networks," in *Proc. ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pp. 90–98, May 2005.

[87] C. T. Cicek, H. Gultekin, B. Tavli, and H. Yanikomeroglu, "Backhaul-aware optimization of UAV base station location and bandwidth allocation for profit maximization," *IEEE Access*, vol. 8, pp. 154573–154588, 2020.

[88] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On reducing IoT service delay via fog offloading," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 998–1010, 2018.

[89] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, and J. P. Jue, "FOGPLAN: A lightweight QoS-aware dynamic fog service provisioning framework," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5080–5096, 2019.

[90] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On reducing IoT service delay via fog offloading," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 998–1010, 2018.

[91] B. Zhang, Q. Fan, X. Zhang, Z. Fu, S. Wang, J. Li, and Q. Xiong, "A survey of VNF forwarding graph embedding in B5G/6G networks," *Springer Wireless Networks*, pp. 1–24, 2021.

[92] S. Aidi, M. F. Zhani, and Y. Elkhatib, "On optimizing backup sharing through efficient VNF migration," in *Proc. IEEE Conference on Network Softwarization (NetSoft)*, pp. 60–65, 2019.

[93] J. Zheng, T. S. E. Ng, K. Sripanidkulchai, and Z. Liu, "Comma: Coordinating the migration of multi-tier applications," *SIGPLAN Not.*, vol. 49, p. 153–164, mar 2014.

[94] Z. Xu, Z. Zhang, W. Liang, Q. Xia, O. Rana, and G. Wu, "QoS-aware VNF placement and service chaining for IoT applications in multi-tier mobile edge networks," *ACM Trans. Sen. Netw.*, vol. 16, no. 3, pp. 1–27, 2020.

[95] M. Abu-Lebdeh, D. Naboulsi, R. Glitho, and C. W. Tchouati, "On the placement of VNF managers in large-scale and distributed NFV systems," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 875–889, 2017.

[96] C. Morin, G. Texier, C. Caillouet, G. Desmangles, and C.-T. Phan, "Optimization of network services embedding costs over public and private clouds," in *2020 International Conference on Information Networking (ICOIN)*, pp. 360–365, 2020.

[97] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "Sndlib 1.0—survivable network design library," *Networks*, vol. 55, no. 3, pp. 276–286, 2010.

[98] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[99] "Configure per VM Tier_1 networking performance." (online). https://cloud.google.com/compute/docs/networking/configure-vm-with-high-bandwidth-configuration#:~:text=To%20create%20a%20a%20VM,Specify%20your%20firewall%20rules.

[100] D. Cho, J. Taheri, A. Y. Zomaya, and P. Bouvry, "Real-time virtual network function (VNF) migration toward low network latency in cloud environments," in *Proc. IEEE International Conference on Cloud Computing (CLOUD)*, pp. 798–801, 2017.

[101] S. N. Afrasiabi, A. Ebrahimzadeh, C. Mouradian, R. Szabo, and R. H. Glitho, "Cjoint vnf decomposition and migration for cost-efficient vnf forwarding graph embedding," *Submited ToIEEE Trans. Netw. Serv. Manag*, 2023.

[102] J. Fu and G. Li, "An efficient vnf deployment scheme for cloud networks," in *Proc. IEEE International Conference on Communication Software and Networks (ICCSN)*, pp. 497–502, 2019.

[103] F. Zhang, H. Lu, F. Guo, and Z. Gu, "Traffic prediction based vnf migration with temporal

convolutional network," in *2021 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2021.

[104] A. Azhdari, A. Ebrahimzadeh, N. Afrasiabi, S. Robert, C. Mouradian, and R. Li, Wubin Glitho, "Cost-aware topological decomposition of virtual network function forwarding graphs," in *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, 2023.

[105] Q. Zhang, F. Liu, and C. Zeng, "Adaptive interference-aware VNF placement for service-customized 5G network slices," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, pp. 2449–2457, 2019.