Contribution to Multi-domain Network Slicing: Resource Orchestration Framework and Algorithms

PhD Thesis Dissertation

by

Godfrey Mirondo Kibalya

Submitted to the Universitat Politecnica de Catalunya (UPC) in partial fulllment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Supervisor: Prof. Joan Serrat Fernandez

Co-supervisor: Assoc. Prof. Juan-Luis Gorricho

Barcelona, February 2022

ABSTRACT

Network Function Virtualization (NFV) provides the possibility of migrating Network Functions (NFs) such as Firewalls and Proxies, among others, from dedicated hardware appliances to general purpose hardware. In this way, network services can be run as Network Slice Instances (NSIs), materialized in the form of customized Service Function Chains (SFCs) consisting of an ordered set of Virtual Network Functions (VNFs). However, in practice, the limited footprint of Infrastructure Providers (InPs) and the location dependencies of certain NFs may necessitate a given service chain to transcend a multi-InP network Infrastructure. In this way, by leasing resources from multiple InPs, the end-to-end service is realized as a concatenation of VNFs hosted by different InPs, thus eliminating the need for deploying new network infrastructure. However, given the limited network resources and the stringent requirements of 5G and future services, the success of such a business model will largely rely on how the diverse set of network and cloud resources under the control of the different InPs, will be coordinated and orchestrated. Moreover, this is complicated by the need to preserve the privacy of InPs and the differing internal polices (such as billing and QoS guarantee, among others) of the different InPs. Therefore, how to orchestrate network services across a multi-domain infrastructure in a seamless, reliable, cost-effective and resource efficient manner is non-trivial. This thesis contributes to the above challenge by breaking the multi-domain service orchestration problem into two interlinked sub-problems that are solved in a coordinated manner: (1) The request splitting/partitioning sub-problem which involves obtaining a subset of cooperating or competing InPs and the corresponding inter-domain links on which to provision the different VNFs and virtual links of the service request respectively; (2) Intra-domain VNF orchestration sub-problem which involves obtaining the intra-domain nodes and links to provision the VNFs and virtual links of the sub-SFC associated with each InP at the request partitioning sub-problem.

The request splitting sub-problem is NP-hard. Therefore, in order to simplify the problem, many existing works adopt heuristic approaches targeting to realize mapping solutions in acceptable run time. Nevertheless, the considerations (such as full information disclosure from the participating InPs, or assuming failure free networks, among others) adopted by many of these works may not be suited for practical 5G and future services. Aware of the stringent reliability requirements of mission critical applications, and the fact that networks may experience failures in practice, this thesis proposes a reliability-aware Reinforcement Learning (RL) based algorithm for splitting of service requests across multiple InPs. The algorithm targets to minimize the operational costs incurred by a Service Provider (SP) considering both the QoS-violation costs (due to service failure) and resource consumption cost. Moreover, this is achieved under limited information disclosure by participating InPs, thanks to the ability of the RL technique to infer the undisclosed information based on historical data.

In addition, previous solutions rely on a centralized entity for making request splitting decisions. However such an approach may not be scalable with an increase in the number of InPs. Aside from that, the level of information exposure required by the centralized entity to make optimal placement

decisions may violate the privacy requirements of the participating InPs. This thesis proposes a multi-stage graph aided distributed algorithm for request partitioning, in which the message exchange occurs only among a pre-computed subset of InPs, with each participating InP forwarding only a single message based on all the received messages, to a subset of InPs. In this way, the average number of InPs participating in the solution computation, the number of messages exchanged between InPs, and the total time for making a mapping decision are greatly reduced in comparison with conventional distributed algorithms, while preserving the privacy of participating InPs.

Similar to the request splitting sub-problem, the intra-domain VNF orchestration problem is Np-hard. Most of approaches that have previously been proposed to address this problem are not suited for practical scenarios due to the fact that they either: perform mapping of the VNFs and virtual links in uncoordinated manner which may result in a high resource consumption; or are not cognizant of the reliability requirements of mission critical applications or do so from the perspective of stateless VNFs. However, in practice, a number of VNFs such as Network Address Translators, among others, are stateful, which necessitates tight coordination in provisioning the active and stand-by instances, in order to minimize the state-update costs. In this regard, this thesis proposes a set of Metaheuristic algorithms based on Genetic and Harmony search algorithms for fault-tolerant orchestration of stateful VNFs. The algorithms target to jointly minimize the service deployment cost due to the primary and stand-by VNF instances, and state-update.

Moreover, given that the backup resources assigned to a service request may stay unused throughout the time the request doesn't experience failure, this thesis proposes a set of survivable intra-domain VNF orchestration algorithms that permit the non-critical applications to share the idle resources reserved for the critical applications. First, the thesis proposes a generic multi-stage graph based algorithm as an alternative intra-domain VNF orchestration algorithm targeting to achieve better resource utilisation by coordinating the mapping of the VNFs and virtual links of a service request. Then, based on the mentioned algorithm, the thesis proposes a new migration-aware algorithm for the mapping of non-critical services, enabling the non-critical services to borrow the unused backup resources from the critical services while minimizing the probability of preemption they could experience. Additionally, whenever low priority users are preempted from their borrowed resources, thesis proposes a new QoS-aware global-rerouting algorithm for remapping those users, reducing the impact of the service interruption thanks to avoiding the migration of surviving VNFs and virtual links when feasible.

Finally, whereas multiple applications may have common VNFs in their service chains, in the majority of previous works, each instance of a VNF only processes the input traffic coming from a single service request. However, dedicated VNF assignments may result in a low resource utilization, excessive resource fragmentation, and higher overall service deployment costs. This thesis proposes a RL based algorithm for cost-effective and resource efficient orchestration of online services from the perspective of sharing their VNF instances. The RL algorithm targets to make intelligent placement decisions while considering multiple conflicting costs including: transmission, VNF instantiation, resource fragmentation or energy consumption costs, among others.

Acknowledgment

First and foremost, I would like to express my sincere and heartfelt gratitude to Prof. Joan Serrat Fernandez for offering me the opportunity to undertake the PhD under the MAPS research group, and unconditionally accepting to be my thesis advisor. I am forever indebted to the excellent guidance, kindness, commitment, and support both academic and financial that I got from him and his lovely Wife. Without this support, there is no single doubt that I would not have moved this far. I would also like to extend my special gratitude to my Co-advisor Assoc. prof. Juan Luis Gorricho for the endless guidance, counsel, mentorship, and dedication from the first day I undertook this thesis work. Without my parents in Barcelona, my thesis advisors took up the parental role.

To my lovely parents and siblings, Mr and Mrs. Mirondo, Harriet, Christine, Fred, Richard, Aidha, Sarah, Peter, Agnes, Ronald, and Babirye who have walked besides me through all my academic Journey. You have always believed in me even in times when the clouds seemed dark. Specially, I owe earnest thankfulness to my lovely wife Patience and Sons Klaus and Kyle for their unfailing love and support throughout the pursuit of my PhD studies. This Thesis is a reflection of your endless motivation, encouragement and prayers.

I would also like to thank my colleagues and friends at UPC with whom we exchanged ideas on a daily basis: Khaled, Birkhan, Akshay, Mikel, Leticia, Juan-Pablo, Pablo, Carlos, Christian, Ahmed, Ahamed, John, Nasibeh, David, and Iliker, among others. Thank you for your support and contribution to the realization of this Thesis. Many thanks to the friends of life I met in Europe: Reagan, Njeri, Jemimah, Justine, Nicholas, Anne, Damalie, Antonio, Paula, Gloria, Peter, Enzo and Adrien, to mention but a few. Thank you for being my extended family a broad. I would like also to thank Prof. Heipeng Yao, Prof. Rafael Pasquini, Prof. Javier Rubi0-Loyola, Dr. Christian Aguilar, Dr. Zhang Peiying, Dr. Jonathan Serugunda, Dr. Dorothy Okello and Ms. Gift Doreen for accepting to collaborate with me on a number of research issues related to this Thesis.

Lastly but not least, I would like to thank the administrative staff, support staff, Mobility Office, and the Doctoral committee of the Technical University of Catalonia for their endless support and guidance throughout the time I spent in Barcelona.

But above all, my praise and thanks go to the merciful Lord who put those wonderful people in my life and who gave me the life, strength, good health and wisdom to undertake the PhD program, for many have such big dreams but few are blessed to see them come to realization.

Dedication

To my lovely parents Mr. and Mrs. Mirondo for their endless love and support. “There he was when I started this long academic race, but he is no more to witness my final lap”. I have no doubt that this is where you wished me to get. Rest in eternal peace Daddy.

Contents

Part I Introduction and Multi-domain service orchestration problem description xii

1 Introduction 1

1.1 Background and Motivation 1

1.1.1 Overview of the multi-domain service Orchestration problem 2

1.1.2 Problem Statement 3

1.2 Thesis objectives 4

1.3 State of the art 4

1.3.1 Distributed approaches for service orchestration 5

1.3.2 Fault tolerant Orchestration of Virtual Network Functions 6

1.3.3 Service survivability with failure recovery: 8

1.3.4 Application of machine learning techniques to the service orchestration problem 8

1.3.5 Application of Graph-based approaches to Service Orchestration 9

1.3.6 Summary 9

1.4 Thesis Contributions 10

1.5 Thesis Context 11

1.6 Scientific Product 12

1.7 Thesis Outline 14

2 Multi-domain service orchestration problem background 16

2.1 Introduction 16

2.2 Multi-domain Service Orchestration Problem 16

2.2.1 Service request model 16

2.2.2 Substrate network model 17

2.2.3 Problem description and Formulation 18

2.3 Solution techniques adopted by the thesis 22

2.3.1 Reinforcement Learning 22

2.3.2 Genetic Algorithm 26

2.3.3 Harmony Search 28

2.4 General performance evaluation environment 30

2.4.1 Performance metrics 31

2.4.2 Simulation environment and settings 35

Part II: Cross-domain Service Orchestration Algorithms 37

3 Reliability-Aware Deep Reinforcement Learning-Based Algorithm for Multi-domain Service Orchestration **38**
3.1 Introduction 38
3.2 Proposed architectural Framework 39
3.3 Proposed multi-domain orchestration sequence diagram 41
3.4 Description of the reliability-aware multi-domain orchestration problem 43
3.5 Proposed RL-Based Orchestration algorithm 46
3.5.1 Candidate Extraction Procedure 47
3.5.2 Markov Decision Process Model for the DRL problem 49
3.5.3 Feature extraction procedure 51
3.5.4 Policy neural network training procedure 53
3.5.5 Architecture of the Neural Network for Policy Evaluation 54
3.6 Performance Evaluation 56
3.6.1 Benchmark algorithms and Simulation Scenarios 56
3.6.2 Simulation environment and settings 58
3.6.3 Result Analysis 59
3.7 Conclusion 65

4 A multi-stage graph aided algorithm for distributed Service Orchestration **66**
4.1 Introduction 66
4.2 Description of the problem 66
4.3 Proposed algorithm for distributed service orchestration 68
4.3.1 Candidate InPs Identification step 68
4.3.2 Message Exchange Step 69
4.3.3 Consensus and Binding step 73
4.3.4 Time complexity analysis 74
4.4 Performance evaluation of the proposed algorithm 74
4.4.1 Simulation environment and settings 75
4.4.2 Results and discussion 75
4.5 Conclusion 84

Part III: Intra-domain Service Orchestration Algorithms **85**

5 Survivable Service Orchestration with backup resource Sharing **86**
5.1 Introduction 86
5.2 Problem description 89
5.2.1 High priority service requests 89
5.2.2 Low priority service requests objective 90
5.3 Proposed SFC deployment algorithms 92
5.3.1 Generic multi-stage SFC deployment algorithm 92
5.3.2 Migration-aware algorithm 98

5.3.3 High priority deployment algorithm 100
5.3.4 QoS-aware service restoration algorithm 100
5.4 Performance evaluation 104
5.4.1 Simulations settings 105
5.4.2 Simulation scenarios 105
5.4.3 Results and discussion 107
5.5 Conclusion 114

6 Fault tolerant placement of Stateful VNFs 118
6.1 Introduction 118
6.2 Description and formulation of the fault-tolerant stateful VNF orchestration problem 120
6.3 Proposed Bi-stage graph based algorithm 122
6.3.1 Candidates matching 122
6.3.2 Bi-stage graph computation 123
6.3.3 Computation of the provisioning solution 124
6.3.4 Time complexity analysis 126
6.4 Metaheuristic Orchestration algorithms 127
6.4.1 Request transformation 127
6.4.2 Encoding Scheme 127
6.4.3 Initialization Functions 128
6.4.4 Constraint Management Approach 131
6.4.5 Genetic Algorithm 132
6.4.6 Harmony Search Algorithm 132
6.5 Performance Evaluation 134
6.5.1 Simulation Settings 134
6.5.2 Considered simulation scenarios and benchmark algorithms 135
6.5.3 Results analysis 137
6.6 Conclusion 144

7 A Reinforcement Learning-based Service Orchestration with VNF Instances Sharing 146
7.1 Introduction 146
7.2 Virtual Network Function modelling and Problem description 147
7.2.1 Virtual Network Functions Model 147
7.2.2 Description and formulation of the service orchestration problem with VNF sharing 148
7.3 Proposed Reinforcement Learning Based algorithm 153
7.3.1 Markov Decision Process Model 153
7.3.2 Architecture of the Neural Network for policy evaluation 156
7.3.3 Neural Network training 157
7.4 Performance Evaluation 158
7.4.1 Simulation setup and parameters 159

7.4.2 Benchmark Algorithms 159
7.4.3 Result Analysis 162
7.5 Conclusion 168

Part IV: Summary of Thesis results, Conclusions and Future work 169

8 Summary of thesis results, Conclusion and Future work 170
8.1 Introduction 170
8.2 Summary of thesis results from the perspective of addressed sub-problems 172
8.2.1 Request splitting/partitioning sub-problem 172
8.2.2 Intra-domain service orchestration sub-problem 173
8.3 Summary of thesis observations from the perspective of adopted solution techniques 176
8.3.1 Reinforcement learning for resource management 176
8.3.2 Multi-stage graph for service provisioning 178
8.3.3 Genetic and Harmony Search Algorithms for fault-tolerant service provisioning 178
8.4 Future work Enhancements 179
8.4.1 Orchestration of stateful VNFs 179
8.4.2 Resource elasticity in multi-domain orchestration 179
8.4.3 Delay models 180
8.4.4 Results validation on test bed scenarios 180
8.5 Conclusion 180
Bibliography 181

List of Figures

1.1 ETSI proposed architecture option for Network Services provided using multiple administrative domains use case. 13

2.1 An illustration of two SFC service requests 17
2.2 An illustration of an SFC orchestration instance 19
2.3 Execution procedure of the Genetic Algorithm 28

3.1 Architectural framework for the excution of the proposed algorithms 40
3.2 Sequence diagram of the multi-domain SFC orchestration framework 41
3.3 Illustration of guided mapping 44
3.4 Flowchart of the multi-domain service embedding algorithm 48
3.5 Training Performance results 53
3.6 DRL Policy Neural Network Architecture 55
3.7 An illustration of DistNSE service provisioning 57
3.8 Scenario 1-Experiment 1: Impact of substrate network size for offline scenario 60
3.9 Scenario 1- Experiment 2: Impact of the number of offline requests 61
3.10 Experiment 1- Impact of substrate network size for online scenario 62
3.11 Experiment 2- Impact of the number of VNFs per request 63
3.12 Experiment 3 -Impact of arrival rate for online scenario 63

4.1 An illustration of a multi-stage graph 70
4.2 Experiment1: Impact of demand size for offline scenario. 76
4.3 Experiment 3: Impact of request size for offline scenario 77
4.4 Experiment 3: Impact of substrate network size for offline scenario. 78
4.5 Experiment 4: Impact of arrival rate for online scenario 80
4.6 Experiment 5: Impact of substrate network size considering online scenario 81
4.7 Experiment 6: Message exchange performance with increase in VNF size. 83

5.1 B5G main service categories and their key performance requirements 88
5.2 A flow chart showing the steps for a request admission and management 91
5.3 An illustration of a multi-stage graph for ML-SFCDA algorithm 94
5.4 An illustration of the localized routing strategy 104
5.5 An illustration of the ChinaNet physical network topology 106
5.6 Experiment 1 of Scenario 1 analyzing the impact of arrival rates 108
5.7 Experiment 2 of Scenario 1 analyzing the impact of substrate network size 109
5.8 Experiment 3 of Scenario 1 analyzing the impact of arrival rate of requests 110

5.9 Experiment 2 of Scenario 1 analyzing the impact of of the number of VNFs per request 111
5.10 Experiment 1 of Scenario 2 analyzing the impact of arrival rate of requests 112
5.11 Experiment 2 of Scenario 2 analyzing the impact of the fraction of high priority users 113
5.12 Experiment 1 of scenario 3 analysing the impact of arrival rate of requests 116
5.13 Experiment 2 of scenario 3 analysing the impact of SFC size 117

6.1 An illustration of a fault-tolerant stateful VNF placement problem 119
6.2 Illustration of the Bi-stage graph 126
6.3 Illustration of the request transformation procedure 128
6.4 Permutation-based SFC representation 129
6.5 Impact of demand size for offline scenario 138
6.6 Results of the online scenario with the arrival rates of requests varied from 10 to 60 requests per 100 time units for a total of 10000 time units 140
6.7 Results of Experiment 3 of scenario 01 in which the jk values are varied from 5 to 35 considering 500 requests and 40 DCs 141
6.8 Experiment 1 of Scenario 02: The impact of the arrival rate is evaluated by varying the arrival rate of SFCs from 2 to 12 arrivals per 100 time units for a total of 50000 time units 142
6.9 Impact of substrate network size 143
6.10 Impact of VNF size for online scenario 144

7.1 An illustration of service requests with shared VNFs 147
7.2 An illustration of SFC deployment with VNF instance sharing 150
7.3 Results of training step of the policy neural network 157
7.4 Results of the training performance of the Feed-forward and convolutional neural networks 158
7.5 Results of experiment 1 considering the impact of demand size for Abilene topology 164
7.6 Results of experiment 2 considering the impact of request size under BIC topology. 165
7.7 Results of experiment 3 considering the impact of substrate network size under synthetic topology. 166
7.8 Results of experiment 4 considering online arrival of requests under BIC topology. 168

List of Tables

1	List of Acronyms	xii
2	List of Sets and Variables	xiii
3	List of notations	xiv
4	Cont. list of notations	xv
3.1	Simulation Parameters	59
4.1	Simulation parameters for the multi-stage graph based distributed algorithm	75
5.1	Simulation parameters for the multi-stage graph based distributed algorithm	105
6.1	Simulation parameters for the Bi-stage graph and Metaheuristic based algorithms	135
7.1	Simulation Parameters for VNF sharing based algorithm	159

Table 1: List of Acronyms

Abb.	**Definition**	**Abb.**	**Definition**
VNE	Virtual Network Embedding	SFC	Service Function Chain
VNR	Virtual Network Request	AI	Artificial Intelligence
ML	Machine Learning	SDN	Software Defined Networking
NFVI	NFV Infrastructure	NFVO	NFV Orchestrator
MANO	NFV Management and Orchestration	NS	Network Service
NSP	Network Service Provider	InP	Infrastructure Provider
NGMN	Next Generation Mobile Network	RO	Resource Orchestrator
ETSI	European Telecommunication Standard Institute	SLA	Service level agreement
KPI	Key performance indicator	MNO	Mobile Network Operator
SDOs	Standards-Development Organizations	VNF	Virtual Network Function
uRLLC	ultra-reliable low-latency communication	NF	Network Function
3GPP	Third generation partnership project	MDP	Markov Decision Process
ITU	International Telecommunication Union	B5G	Beyond Fifth Generation
6G	Sixth Generation	5G	Fifth Generation
QoS	Quality of Service	eMBB	enhanced Mobile Broadband
DC	Data Center	CAPEX	Capital expenditure
OPEX	Operational expenditure	SP	Service Provider
NSI	Network Slice Instance	IoT	Internet of things
NFV	Network Function Virtualization	RL	Reinforcement Learning

PART I: Introduction and Multi-domain service orchestration problem description

Table 2: List of Sets and Variables

$\mathbb{R}$	A set of all received requests in the system
R_A	A set of all successfully admitted requests
N_s	A set of nodes of the substrate network
E_s	A set of links in the substrate network
$\mathbb{K}$	A set of all InPs comprising the substrate network
N_s^k	A set of nodes of the substrate network in InP $k \in \mathbb{K}$
E_s^k	A set of substrate links in InP $k \in \mathbb{K}$
E_{int}	A set of all inter-domain links in the substrate network
$\mathbb{F}_p^{n_s^k}$	A set of all function types that can be provisioned on $n_s^k \in N_s^K$
N_v	A set of VNFs of an SFC request
E_v	A set of virtual links of an SFC request
Q	A set of all resource types in a network
Q^k	A set of all resource types in InP $k \in \mathbb{K}$
C_{dem}^r	A set of resource requirements at the different VNFs of request $r \in \mathbb{R}$
S	A set of system states in an MDP
A	A set of all actions in an MDP
$\mathbb{R}_c$	A set of all high priority requests in the system.
$\mathbb{R}_c^q$	A set of all admitted high priority requests.
$CaS_S^{i,r}$	A set of substrate nodes that satify the constraints of VNF i of request r.
$Cand_s^r$	A set consisting of candidate nodes for the different VNFs of the request r.
$path_{disj}^{st}$	A set of all node disjoint paths between ingress and egress nodes of request r.
$feas_{path}$	A set of all feasible node disjoint paths between ingress and egress nodes a high priority request
DC_{opt}	A set of optimal DCs for provisioning request r.
$Cand_{act}^r$	A set of candidate substrate nodes for the active instance of request r.
$Cand_{std}^r$	A set of candidate substrate nodes for the standby instance of request r.
M	A set of all VNFs in the network.
P	A set of all VNFs types in the network.
Υ_{vnf}^p	A set of substrate nodes on which a VNF of type $p \in P$ can be provisioned.
$Path_K^{\tau_s^r, \tau_d^r}$	A set of K shortest path between ingress and egress nodes of request r .
$Cand_{n_v^p}^k$	A set of candidate InPs for VNF n_v^p .

Table 3: List of notations

$P_{s,r}^k$	The kth shortest path from s to r for the demand.
$P_{set}^{s,r}$	Set of all Kshortest paths from source s to destination r in the network.
V($P_{s,r}^k$)	Set of feasible physical substrate nodes to map VNFs for $P_{s,r}^k$.
t_{d_w}	The duration of demand w.
T	Duration of the simulation window in time units.
$\delta(l)$	Propagation delay
$\delta_{s,r}(l)$	Maximum acceptable E2E delay.
$Z(d_w)$	1 if demand w is successfully mapped.
Ψ^r	Tuple specifying attributes of request r.
ρ^r	Requested packet rate of request r.
τ_s^r	Ingress node of request r.
τ_d^r	Egress node of request r.
τ_d^r	Egress node of request r.
G_v^r	SFC connectivity graph of request r.
n_v^p	A VNF of type $p \in P$.
n_v^p	A VNF of type $p \in P$.
$dem_q^{n_v^p}$	Amount of type q resource required by n_v^p.
$loc^{n_v^p}$	Acceptable location region for n_v^p.
$\rho_{n_v^p}^r$	packet rate traversing n_v^p.
C_q^ρ	Amount of type q resources required to process each unit of packet rate by a given node.
l_{uv}	Request virtual link between VNFs u and v.
$dem_{bw}^{l_{uv},r}$	Bandwidth requirement of $l_{uv} \in E_v$.
τ_f^r	Life-time of request r.
del_τ^r	End-end delay requirement of request r.
G_s	Directed graph of the substrate network.
G_s^k	Directed graph of the substrate network for InP $k \in \mathbb{K}$.
n_s	A substrate node within N_s
$C_{q_{max}}^{n_s^k}$	Type q resource capacity at substrate node n_s^k
$\varrho_q^{n_s^k}$	Cost of each unit of type q resource.
$C_{q_{res}}^{n_s^k}$	. Amount of residual resources of type q on substrate node n_s^k
n_s	A substrate node within N_s
e_{int}	A single hop inter-domain substrate edge.
e^k	A single hop edge within InP k.

Table 4: Cont. list of notations

$B_{max}^{e^k}$, $B_{max}^{e_{int}}$	Bandwidth capacity for e^k, e_{int}.
$Bw_{res}^{e^k}$,$Bw_{res}^{e_{int}}$	Residual bandwidth resources at a given time on e^k, e_{int}.
δ^{e^k} , $\delta^{e_{int}}$	Propagation delay on e^k, e_{int} .
ϱ^{e^k} , $\varrho^{e_{int}}$	Cost for each unit of bandwidth resource consumed on e^k, e_{int} .
ζ^{e^k} , $\zeta^{e_{int}}$	Cost for transmitting each unit of packet rate on e^k, e_{int} .

CHAPTER 1

Introduction

This chapter presents a high level introduction to the thesis, including: the background and the motivation behind the work in the thesis, the problem statement and the thesis objectives. In addition, the chapter presents a list of scientific publications that resulted from the content of the thesis and the thesis positioning with respect to the State-of-the-art (SoA). The chapter ends with a summary of the SoA with respect to the thesis contribution and an outline of the thesis structure.

1.1 Background and Motivation

Future applications and services such as Tactile Internet, remote surgery and autonomous driving, among others, impose strict requirements in terms of throughput, latency, and reliability, among other requirements, that cannot be met with the traditional "one size fits all" monolithic architecture [1, 2]. As such, network operators are expected to leverage the flexibility introduced by the Network Function Virtualization (NFV) paradigm in order to cope with the stringent requirements of these services. The main target behind NFV comes from decoupling complex network functions (e.g., Firewalls, Proxies or Load Balancers) from dedicated hardware appliances, implementing those services in the form of chained Virtual Network Functions (VNFs), also called Service Function Chains (SFCs) [3]. In this way, network services and applications will be instantiated as Network Slice Instances (NSIs), that will be realized in the form of customized logical and self-contained networks, consisting of a mixture of both shared and dedicated resources, including VNFs [4, 5]. If well implemented, the above approach has high prospects of reducing its network deployment footprint, as well as its associated operational cost. This is premised on the ease with which such softwarized functions can be activated, scaled, migrated or shutdown, allowing a more efficient use of the network resources [6].

In practice, the limited footprint of InPs and the location dependencies of certain network functions (e.g, requiring packet filters to be placed close to traffic sources in order to conserve bandwidth resources at the event of DoS attacks) [7] may require the different service chains to transcend network Infrastructure belonging to multiple providers. Under the NFV paradigm, such applications are envisaged to be supported by leasing resources from different infrastructure providers without the need of deploying new network infrastructure. Therefore, the end-to-end NSI will be realized as a concatenation of slice parts or VNFs hosted by different InPs [8], resulting in a significant reduction in both cost and time for orchestrating a service request. In fact, the European

Telecommunications Standards Institute (ETSI) has already defined *NFV Infrastructure as a service (NFVIaaS)* use case in [9], and the *Network Services provided using multiple administrative domains* use case in [10], both of them applicable to scenarios where a given SP relies on resources of multiple InPs to meet its resource needs.

1.1.1 Overview of the multi-domain service Orchestration problem

In the general case, the grand multi-domain service orchestration problem can be viewed to consist of two interlinked sub-problems executed at two levels [7, 11, 12]: at the higher level, given a set of cooperating or competing InPs comprising the substrate network, the problem is to determine an optimal subset of InPs and the associated inter-domain links to provision the different VNF instances of a service request. This problem, which is denoted as **sub-problem 1** in this thesis document is commonly referred to as the service request partitioning/splitting problem in literature [11, 13], and has been shown to be NP-hard [13, 14]. This is usually modeled and solved as an ILP [7, 11, 12], with a goal of obtaining optimal solutions, albeit, at a cost of high run time; at the lower level, the problem is that of intra-domain service orchestration in which each individual InP makes a decision regarding the specific intra-domain nodes (e.g., servers, Data centers, etc.) and links to provision its assigned VNF instances, based on its intra-domain policies. Such a problem, which we denote as **sub-problem 2** in this thesis document, has also been shown to be NP-hard and computationally intractable since it can be reduced to the multi-way separator problem which is known to be NP-hard itself [15]. Moreover, even for a small number of nodes, the problem of optimally allocating a set of virtual links to single physical paths reduces to the un-splittable flow problem, which is also NP-hard [16, 17]. In this regard, it is not practical to achieve optimal solutions for the grand multi-domain service orchestration problem in polynomial time for a network of practical size. Moreover, the provisioning policies and decisions taken at the lower level directly impact the quality of the global provisioning solution, hence, necessitating the two sub-problems to be solved in a coordinated manner.

Given the diverse number of competing or cooperating InPs, obtaining an optimal set of InPs for orchestrating service requests in a cost-effective and resource-efficient manner, while meeting the associated constraints, is a challenging and complex problem [18]. This is due to the fact that the different InPs may be characterized by different internal policies regarding aspects such as billing and QoS guarantees, among others. Moreover, due to reasons related to security and business competition, there is a general reluctance among InPs to disclose their intra-domain information regarding topology and resource availability [19], requiring SPs to provision service requests without having control or even knowledge of any aspect of the physical infrastructure in the different domains. This may have serious implications regarding resource utilization, efficiency and QoS satisfaction [20]. Moreover, this inhibits the direct replication of single domain orchestration algorithms such as [21–25] to the multi-domain orchestration problem, since these rely on a full visibility of the intra-domain attributes. In light of the above, research regarding novel and innovative techniques for coordinating and orchestrating the diverse set of network and cloud resources under the control of the different InPs is not only relevant, but also urgent.

1.1.2 Problem Statement

Even though the benefits of *NFV Infrastructure as a service* and *Network Services provided using multiple administrative domains* use cases are evident and intuitive, in terms of reducing service deployment cost and network roll-out time, the practical realization of these benefits will largely rely on how the diverse set of virtualized and non-virtualized resources under the control of the different InPs will be coordinated and orchestrated and how the challenges brought about by network softwarization will be managed. First, in addition to the hardware failure experienced in the traditional Physical Network Functions (PNFs), the softwarized environment introduces an extra component of failure at software level due to software faults (e.g., OS errors), misconfigurations (e.g., configuration conflicts, wrong rule insertion) and malfunctions, among others [26], which poses critical concerns given the mission-critical applications envisaged to be supported by 5G and future network [27–29]. This is even more pertinent in a multi-domain service provisioning scenario, since a failure or malfunction of an instance provisioned within a single InP affects the entire end-to-end service, potentially resulting in penalties for SLA violations. Secondly, it is anticipated that these networks will support a myriad of applications with extreme resource requirements [30]. Therefore, how to effectively orchestrate these services in a resource efficient and cost effective manner remains a critical hurdle to be overcome by SPs. Moreover, aside from the resource consumption, the operation cost under the softwarisation environment is influenced by multiple attributes including energy consumption cost, VNF activation cost, and SLA violation costs (e.g., due to service failures and interruptions), some of which may be conflicting among themselves, further complicating the service deployment decisions.

Previously, the works in [7, 11–13] adopted an exact approach for splitting service requests across multiple domains. However, such approaches are not suited for delay sensitive applications due to their high run time. In this regard, most of the recent works adopt heuristic approaches [19, 31–37], targeting to realize provisioning solutions in feasible run time. However, despite the criticality of service reliability in future networks, to the best of our knowledge, there is no previous work that incorporates service reliability in the cross-domain service orchestration problem. Moreover, even the single domain approaches that consider service survivability either do so from the perspective of stateless VNFs in the service chaining [29, 38–45] or impose that the entire SFC instance should be provisioned on a single node e.g., server or Data center [26–28]. However, in practice, a number of VNFs are stateful, with typical examples of such VNFs being network address translators that store mappings between ports and hosts, and intrusion detection systems, that keeps track of pattern matching to detect attacks [46]. This aspect renders such algorithms not well suited for the fault-tolerant service orchestration which requires tight coordination in provisioning the active and stand-by instances, in order to minimize the state-update signaling overhead. Aside from this, most of the existing solutions make unrealistic considerations while provisioning service requests such as assuming full information exposure [47–50], or they donot fully coordinate the Intra-domain and cross-domain mapping steps of the algorithm.

1.2 Thesis objectives

The main objective of this thesis is to facilitate the realization of end-to-end network slicing across a multi-domain network infrastructure by proposing algorithms for service orchestration across multiple domains that jointly target to: (i) realize efficient resource utilization through provisioning of the service request VNFs and virtual links in a coordinated and resource-efficient manner; (ii) minimize operational costs and penalties resulting from SLA violations due to failure of the physical nodes, virtual network functions and virtual links; (iii) preserve the privacy requirements of the participating InPs; (iv) achieve all the above in practical run time for delay sensitive applications. In order to achieve the above targets, the thesis undertakes the following specific tasks:

- Develop a cross-domain service orchestration framework to guide the implementation and execution of the service orchestration algorithms while adhering to the privacy requirements of the InPs. The proposed framework is aligned with the ETSI NFV-MANO architectural framework described in [51] and the ETSI's architecture options introduced in [10] in which there is a single NFV Orchestrator (NFVO) per administrative domain, with the different orchestrators able to communicate through the Orchestrator-to-Orchestrator (Or-Or) interface proposed in [10].
- Define the models of the substrate network and service requests, and mathematically formulate the multi-domain service orchestration problem as an ILP including the underlying network and service request constraints.
- Develop and evaluate algorithms for the service request splitting/partitioning sub-problem of the grand cross-domain service orchestration problem that are aligned with the four aforementioned targets of the thesis main objective.
- Develop and evaluate intra-domain service orchestration algorithms that are aligned with the four aforementioned targets of the thesis main objective.
- Evaluate the performance of the proposed algorithms considering relevant metrics related to resource consumption, cost , execution time and acceptance ratio, among others, against related state-of-art algorithms.

1.3 State of the art

With the success of 5G and future networks tightly tied on how the diverse set of resources belonging to multiple InPs will be coordinated and orchestrated [52], research regarding the cross-domain service orchestration/federation problem has gained traction. In literature, this problem is largely tackled from two perspectives, thus [53]:

- The problem of distributing the different parts of the service request among multiple InPs.
- The problem of developing procedures and interfaces to effect this distribution.

Standards Development Organisations (SDOs) have been at the forefront of addressing the latter aspect of the problem, with proposals coming from: the Open Networking Foundation (ONF) [54] in which a peer-to-peer SDN controller based architecture is proposed; the Metro Ethernet Forum (MEF) [55] in which a Lifecycle service orchestration reference architecture is proposed; and the ETSI [10] where different architecture options for cross-domain service orchestration are proposed. In addition, there are a number of cross-domain orchestration architectures proposed by the research community in [52, 53, 56–61]. However, the mature proposal is the ETSI report at [10], in which a new “Or-Or" (orchestrator-to-orchestrator) interface for establishing connectivity between the orchestrators of the different domains is introduced for the various architecture options. In light of this, the thesis proposals are aligned with the architectural frameworks in [10].

In literature, the first aspect, which is the focus of this thesis is addressed either in a centralized or distributed manner: centralized approaches rely on a third party entity that uses the exposed global information to make decisions regarding the different InPs to which to distribute the different parts of a given service request. Such a decision is usually executed using exact approaches based on Integer Linear Program (ILP) models as adopted in [7, 11–13, 62]. However, ILP based approaches yield optimal solutions at a cost of high run time, making them not well suited for practical delay sensitive services and applications. As a result, recent proposals in [19, 36, 37, 47–50, 63] are based on heuristic approaches with a view of realising near-optimal solutions with practical run time. However, a key challenge with centralised schemes is that they may require a level of information exposure which violates the privacy requirements of the participating InPs, which renders them unsuited for practical scenarios in which the InP privacy has to be enforced. Moreover, it is not possible for multiple entities to compute the partitioning solution in parallel, which affects the scalability and run time of the centralised approaches; On the other hand, the request partitioning in distributed approaches involves the joint participation of multiple InPs [33, 64–66], making them well suited for dynamic network environments. However, conventional distributed approaches are penalized by an increasing processing delay and signaling overhead when making request provisioning decisions as the number of participating InPs increases, hence, compromising their scalability. This is attributed to the overhead in terms of message exchange between neighboring entities.

In the sub-sections below, the thesis introduces a high-level summary of the state-of-the-art with respects to the different aspects addressed by the thesis, while highlighting the major differences between the SoA and the thesis proposals in terms of the adopted solution techniques and other considerations.

1.3.1 Distributed approaches for service orchestration

The need for parallel processing, the desire to realise autonomous network components, and the limited visibility of the global context of the network infrastructure are some of the key driving forces behind the growing attraction of distributed approaches especially in the domain of network and service management. A distributed protocol and framework for cross-domain service embedding was proposed in [64], although no service provisioning algorithm is proposed in that work. Algorithms

for distributed service orchestration were proposed in [19, 35, 66]. For instance, in [35], upon the arrival of a SFC request, the centralized orchestrator forwards the request to the different participating InPs with each selecting a sub-SFC it can map following its internal policies. The different intra-domain mappings are then forwarded to the central orchestrator which selects the optimal InPs for provisioning the service request with the goal of minimizing the overall provisioning cost. In [66], a distributed embedding algorithm is proposed for the single VNE problem, in which each substrate node acts as an autonomous agent, with the solution computation being obtained through message exchange between the neighboring nodes. However, the messages exchange overhead is unavoidable as the number of substrate nodes increases, since, even the unfeasible nodes participate in the solution computation. In [19], a semi-distributed approach is proposed which relies on abstracting the intra-domain topologies of the InPs, and then computing possible paths between the ingress and egress nodes from which the path with the least mapping cost is selected for provisioning the request. However, such an approach has high time complexity as the substrate network size increases.

Different from the previously proposed works, the distributed multi-domain service orchestration algorithm introduced in Chapter 4 incorporates the following innovative aspects: first, in contrast to conventional distributed algorithms such as [35, 66] , where a messages exchange occurs between any node and all its neighbors, the messages exchange in the thesis proposal involves only a pre-computed set of candidate nodes, thanks to the use of a candidate extraction step. In this way, the average number of nodes participating in the solution computation and the average number of messages processed by each node is shown to be significantly reduced; secondly, the thesis proposal incorporates a message processing technique in which, upon receiving message blocks from other InPs, each candidate InP processes the received messages and forwards only a single message, based on all the received messages, to a given subset of InPs. These two techniques, result in a significant reduction in the message overhead on the network links and time for making a service provisioning decision. Moreover, with the adopted approach, it is possible to detect unfeasible requests in early stages of the algorithm execution, further enhancing the algorithm time performance, hence, rendering a well suited approach for delay sensitive applications. In addition, the thesis proposal incorporates multiple realistic intra-domain performance parameters, such as processing costs, intra-domain delays, VNF activation costs or energy costs, among others, an aspect which is not considered in the SoA multi-domain orchestration algorithms.

1.3.2 Fault tolerant Orchestration of Virtual Network Functions

Cognizant of the extreme reliability requirements of mission-critical applications such as auto-motives and online-surgery, among others, that are envisaged to be supported by 5G and future networks, service survivability has emerged as a relevant research topic. In the literature, service survivability is addressed through two main approaches:

- Pro-actively provisioning and reserving backup resources for each VNF and virtual link of the service request at the mapping stage [40, 43, 67].
- Adopting an intermediate approach such as: reactively provisioning restoration resources

upon failure [42, 68, 69]; provisioning back-up resources for only critical VNFs of the SFC [38, 39, 70–72]; selecting the most reliable nodes and links for hosting the SFC [73]; or transforming the request topology into a form that enhances its survivability [6].

However, the thesis proposals differ from these and other works in the following aspects: With respect to the fault-tolerant VNF orchestration approach proposed in Chapter 6, these works address this problem from the perspective of stateless VNFs. However, stateful VNF orchestration requires full coordination while provisioning both the active and stand-by instances of the service request. This is due to the continous state update between the active and stand-by instances which would result in a high consumption of bandwidth resources, potentially resulting in network congestion, rendering the above approaches unsuited for a practical scenario in which a number of VNFs are stateful. While considering stateful VNFs, the works in [74–77], deal with the problem of how to manage and transfer states from one VNF instance to a new VNF instance while considering elastic control events such as scale-in/-out and load balancing. Under such events, the states can be stored in the local state memory of the current VNF instance, then transferred to the new VNF instance only when an elastic control event is triggered. Therefore, these works deal with the problem of how efficiently such states can be transferred to a new VNF instance in order to facilitate fast and seamless traffic rerouting to the new VNF instance. However, such an approach is not feasible in a scenario characterised by server/VNF failure, since in practice, the stored state will be lost as well upon failure of the associated server/VNF. Moreover, these works only focus on the state transfer problem, without regard to the placement of the SFCs under fault-tolerance requirements. In [27] and [26] , a ranking approach is adopted in which the DC with the highest rank, in terms of computational and bandwidth resources, is favored to host the active instance, while the DC that results in the least resource cost to the above DC is chosen to host the stand-by instance, with the aim of minimizing the state update cost. In general, although such node ranking approaches are time efficient and easy to implement, they are not well suited for cases in which the placement decision is jointly influenced by multiple attributes(e.g., processing resource, VNF activation cost, and energy consumption, among others), since determining the influence/weight of each attribute towards the placement objective is nontrivial. The work in [28] adopts a reinforcement learning approach for the online fault-tolerant placement of VNF chains. However, the action space in that work grows as $\frac{N_{DC}!}{(N_{DC}-2)!}$ where N_{DC} is the number of DCs. This greatly impacts the algorithm convergence, hence, its performance for a manageable number of training episodes. Similar to our approach in [78], these works restrict an entire SFC instance to be mapped on a single Data Center. However, although such an approach simplifies the mapping problem, in practice, the different VNFs of an SFC instance may need to be deployed across different DCs due to coverage and resource constraints, further complicating the placement problem due to the additional degrees of freedom introduced into the problem. The thesis proposal presented in Chapter 6 differs from the above existing works from the perspective of:

- The adopted solution techniques, by use of genetic and harmony search algorithms as an enhancement to the RL proposal we presented in [78]. Moreover, different from [28], our RL proposal in [78] tames the action space size by sequentially selecting the host for the active and

stand-by instances, but with the two selection steps being coordinated. Moreover, the proposed approach uses a convolutional neural network architecture due to its fast training speed.

- Adopting a flexible deployment scheme that does not constrain the entire service chain instance to be placed on a single DC
- Adopting a fully coordinated approach for placement of both active and stand-by instances, thanks to the proposed request transformation technique.

1.3.3 Service survivability with failure recovery:

Even with the service reliability enhancements proposed in the above works, the primary resources of a service request will always have a non-zero probability of failure. In this regard, techniques and algorithms for recovering the service from the failed nodes and links become critical. In light of the above, the works in [42, 68, 69, 73] incorporate a failure recovery component in addressing the service continuity problem. In [42, 68, 69], a local rerouting strategy for service restoration is adopted in which the service restoration solution is obtained by detouring the failed nodes and links, then returning to the surviving path components. Similar to the thesis proposal in Chapter 5 , [42] considers a scenario comprised of critical and non-critical services, pro-actively provisioning backup resources for the critical services, and reactively provisioning backup resources, upon failure, for the non-critical services. The service restoration path is computed in two steps; with the first solution that is computed using the K-shortest path serving as the initial solution for the Tabu search algorithm used to compute the final solution. However, such a two-step approach may introduce an unacceptable service disruption in case the first step does not result in a feasible mapping solution. The thesis proposal regarding service survivability as proposed in Chapter 5 differs from existing works in the following aspects:

- Permitting the non-critical applications to borrow the unused resources reserved for the high priority requests when feasible, in a manner that minimizes the probability of such users being preempted from the borrowed resources, thereby minimizing the level of service disruption experienced by such requests.
- Proposal of a service restoration strategy that targets to minimize the level of service disruption by minimizing the number of surviving VNFs and virtual links of the failed service request that are migrated to new resources. Moreover, the service restoration procedure is executed in a single step, further minimizing the level of service disruption.

1.3.4 Application of machine learning techniques to the service orchestration problem

Machine Learning (ML) has emerged as a promising technique in the domain of network management due to its ability to make intelligent decisions in dynamic and fuzzy environments. The works in [28, 79–82], adopted ML techniques to the problem of SFC provisioning in single domain networks. For instance, [28] focusses on solving the fault-tolerant placement problem of stateful VNFs with

the goal of reducing the state update overhead. In [79] the target is to exploit the inter-relation between different SFCs by aggregating multiple requests and admitting them jointly as a bunch. In [80] the authors target the problem of large network state spaces, considering a real-time online SFC orchestration under dynamic network conditions. In [81] the work focuses on an accelerated approach for learning proper VNF sizing and placement considering various network conditions. Different from these and other previous works, the service orchestration decisions performed by the RL in this thesis is influenced by multiple conflicting cost components such as costs of transmission, VNF instantiation or energy consumption, and fragmentation costs as adopted in Chapter 6 and QoS violation and resource consumption costs as adopted in Chapter 3. This complicates the decisison problem requiring careful modeling of the state space and reward signal in order to realize learning convergence in feasible run time. Moreover, different from the above works, the neural network architecture of the policy in charge of making decisions in our work includes a convolutional layer, allowing a faster training stage and providing a higher convergence, since it uses a smaller number of trainable parameters compared to conventional feed forward neural networks. In addition, the thesis designs the policy neural network in such a way that it can be used even for substrate networks that are inferior to that used at the training stage, thanks to the innovative technique of adopting dummy features.

1.3.5 Application of Graph-based approaches to Service Orchestration

In [63, 83–85] different multi-stage based approaches are proposed for solving the service embedding problem. The embedding solution in these works is obtained by either applying the Viterbi-algorithm [83, 85], or a flow based algorithm [63, 84], directly on the graph. However, all these approaches require a centralized entity which has a global view of the weights of the nodes and links constituting the graph, something that is not feasible under a scenario of partial information disclosure regarding those node and link weights. Moreover, different from these approaches, the multi-stage graph tool adopted in Chapter 4 is only used to establish neighborhood relationships between the different candidates, and it is not used for directly computing the mapping solution.

1.3.6 Summary

The preceding subsections underpin the need for algorithms for orchestrating and accommodating a multitude of service requests on a shared underlying infrastructure belonging to multiple InPs in a flexible, agile and cost-effective manner while conforming to the required QoS. Algorithms that jointly target to minimize SLA violations, enforce efficient resource utilization, and preserve the privacy requirements of users while performing under stochastic environments will be key in realizing the vision of 5G and future networks. However, as noted, the previously proposed algorithms either make unrealistic assumptions regarding the nature of service requests such as stateless VNFs, or do not take into account the different components that affect the operation cost such as SLA violation penalties, energy, and VNF activation costs, among others.

1.4 Thesis Contributions

In light of the above , this thesis contributes to the problem of multi-domain network slicing by proposing a set of novel intra-domain and cross-domain service orchestration algorithms supporting multiple service constraints including end-to-end delay while preserving the privacy requirements of the different InPs. In summary, the key thesis contributions regarding the two sub-problems are as follows:
In addressing **sub-problem 1** that involves partitioning/splitting of a service request across multiple domains, the thesis proposes:

- A resource and cost efficient reliability-aware reinforcement learning based algorithm that incorporates both resource and QoS-violation costs while selecting the different InPs and their associated inter-domain links for provisioning the different service requests.
- A multistage-graph aided algorithm for distributed orchestration of service requests, that targets to minimize both the number of InPs involved in the computation of the service orchestration solution, and the number of messages processed by the participating InPs in search of the mapping solution.

In addressing **sub-problem 2** that involves orchestration of VNFs within a given domain, the thesis proposes:

- A set of survivable and Quality-of-service oriented multi-stage graph based algorithms targeting efficient resource utilization and a reduction in SLA violations due to service disruption. The specific underlying algorithms are:
  - A migration-aware VNF orchestration algorithm which allows low priority requests to borrow the unused backup resources of high priority requests, with a goal of increasing the resource utilization efficiency, while minimizing the level of service interruption due to preemption of these users from the borrowed resources.
  - A quality-of-service-aware (QoS-aware) service restoration algorithm for remapping the low priority users subject to preempted resources. The algorithm results in a reduction in the number of surviving VNFs that are migrated to new nodes, which not only reduces the migration delay and cost, but also the cost related to new VNF instantiations, as in practice, such migrations may involve activating new virtual machines and servers.
- A set of algorithms based on Genetic algorithm and Harmony search meta-heuristics for fault-tolerant orchestration of stateful VNFs with support for provisioning the different VNF instances of a service request across different nodes. To the best of our knowledge, this is the first work adopting these solution techniques to the problem of stateful VNF orchestration and also permitting flexibility in the service deployment.
- A Reinforcement learning algorithm for VNF orchestration with support for VNF instance sharing that jointly incorporates multiple cost components including; energy, VNF activation, processing, forwarding, and fragmentation costs.

Other contributions

In addition to the above contributions that are captured in this thesis document, the PhD work resulted in other contributions including the following:

- A generic algorithm for intra-domain service embedding proposed in [84]. The main idea behind the contributed algorithm consists of a request decomposition technique which involves transforming an otherwise complex request graph into a set of simple edge disjoint path segments whose embedding solutions are computationally tractable. Moreover, performing such a transformation makes the algorithm suitable for mapping requests of any topology; then, a multi-stage graph based technique is adopted for embedding the different path segments, yielding near-optimal solutions in practical run time.
- A genetic algorithm for cross-domain service embedding proposed in [86]. The proposed algorithm eliminates cross-over operation between identical parents which would otherwise result in off-springs that are identical to parents, thus reducing the population diversity. Moreover, instead of taking the best half of the parents into the next generation during the natural selection stage as it is the case in traditional GA algorithms, the selection of parents to enter the next generation in the proposal is based on a probability distribution.

1.5 Thesis Context

Considering that the mature proposals regarding an architectural framework and interfaces for cross-domain service orchestration have been proposed by ETSI, the multi-domain service orchestration framework and algorithms proposed by the thesis are aligned with the ETSI's *NFV Infrastructure as a service (NFVIaaS)* use case as described in [9], and the *Network Services provided using multiple administrative domains* use case described in [10], both of them applicable to scenarios where a single service provider is unable to meet the requirements of its consumers. In this context, a SP can meet the requirements of service requests that span beyond its foot print by leasing resources from multiple resource providers. As articulated in [10], under the NFVIaaS use case, the tasks of: VNF placement decision, management of software images for the deployed VNFs, SLA supervision or management of the intra-domain VNF infrastructure, among others, are delegated to the NFVIaaS provider, with whom the NFVIaaS consumer establishes an "a priori" commercial agreement. This underpins the necessity to consider a limited level of information exposure as adopted in this thesis, since in practice, a given InP will have limited control and visibility of the network operations happening in another InP domain.

The different architecture options, through which the logical interconnection and service orchestration in a multi-provider scenario can be supported, are already proposed and described in the ETSI report [10]. The ETSI NFV-MANO architectural framework described in [51], serves as the basis for the aforementioned multi-domain architecture options, with additional enhancements of the interfaces and reference points where necessary, depending on the specific architecture option. In particular, the ETSI NFV-MANO architectural framework is constituted of a set of functional

blocks, data repositories used by these blocks, and the respective interfaces and reference points through which the different blocks can exchange information in order to effectively manage the virtualized infrastructure and the corresponding services within a given administrative domain. The key building blocks of the architecture are: the NFV Orchestrator (NFVO), the Virtualized Infrastructure Managers (VIMs) and the VNF Manager (VNFM). The NFVO is responsible for the orchestration of NFVI resources across multiple VIMs and the life-cycle management of the deployed network services. The VNFM is responsible for the life-cycle management of VNF instances, including VNF instantiation, modification, healing and termination, among others. On the other hand, the VIM is in charge of controlling and managing the NFVI compute, storage and network resources within a given domain. In order to achieve a multi-domain connectivity, the architecture options permit the exchange of information among different domains, including IP addresses of the distinct functional blocks to be interconnected, such as the NFVO, the unique identifiers of the administrative domains to be interconnected, and the administrative organization they pertain to, among others. Moreover, the proposed architecture options may allow for auto-discovery mechanisms in which the different NFV-MANO functional blocks of the different domains can advertise their own information which can be exploited by the discovery mechanisms to establish a connectivity relation [10].

An architecture option for the *Network Services provided using multiple administrative domains* use case is shown in Fig. 1.1 [10], which considers a case where there is a single NFVO per administrative domain. In this case, a new reference point $Or - Or$ is proposed to be added to the NFV-MANO architecture to facilitate the communication between the different NFVOs in order to enable a life-cycle management of the deployed composite service. In the shown architecture example, domain A is the originating domain of the service request, with NFVO-1 (which is considered as the master orchestrator in this thesis) being in charge of the life-cycle management of the composite service, including initiation of scaling operations when necessary, while NFVO-2 and NFVO-3 are responsible for the life-cycle management of the nested services (NSs) running inside their respective administrative domains. However, NFVO-1 is unaware of the virtualized resources in the host domains of both NFVO-2 and NFVO-3, with the interaction between the VNFM of each domain being limited to the respective NFVO of that domain. In this regard, algorithms that are cognizant of the limited information exposure as those proposed by this thesis are well suited for service orchestration under this scenario. Moreover, abstracting the internal topology of the different domains from the master orchestrator has been found to result in a significant reduction in the solution computation time with a tenable cost increment [11, 12, 63]. The authors in [63] and [11, 12] analyzed the time reduction gain and provisioning cost performance, respectively, resulting from abstracting the internal topologies of the different domains in the multi-provider service deployment problem.

1.6 Scientific Product

The thesis work has yielded a number of scientific publications in international Journals and conferences. These contributions are listed below.

List of Journal papers published

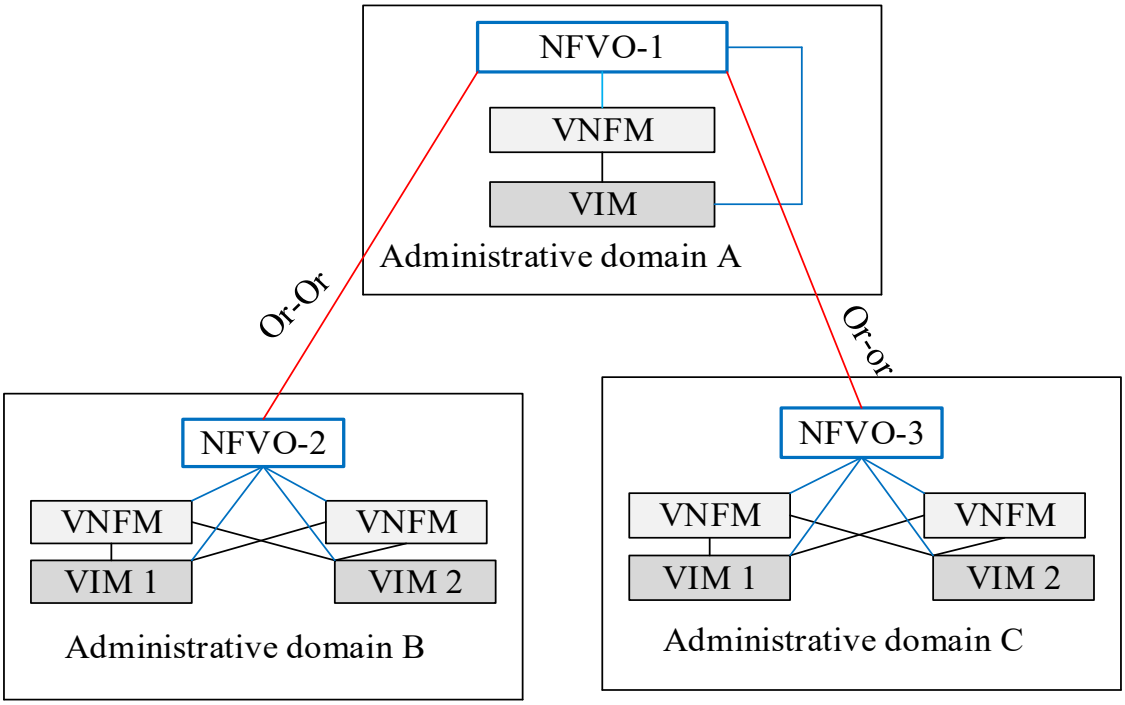


Figure 1.1: ETSI proposed architecture option for Network Services provided using multiple administrative domains use case.

1. Godfrey Kibalya, Joan Serrat, Juan-Luis Gorricho, Jonathan Serugunda, Peiying Zhang, “A multi-stage graph based algorithm for survivable Service Function Chain orchestration with backup resource sharing,” Computer Communications, Volume 174, 2021, Pages 42-60, ISSN 0140-3664.

2. Godfrey Kibalya, Joan Serrat, Juan-Luis Gorricho, Dorothy Okello, Peiying Zhang “A deep reinforcement learning-based algorithm for reliability-aware multi-domain service deployment in smart ecosystems.” Neural Computing and Applications (2020): 1-23.

3. Godfrey Kibalya, Joan Serrat, Juan-Luis Gorricho, Haipeng Yao, Peiying Zhang, “A novel dynamic programming inspired algorithm for embedding of virtual networks in future networks”, Computer Networks, Volume 179, 2020, 107349, ISSN 1389-1286.

4. P. Zhang, X. Pang, G. Kibalya, N. Kumar, S. He and B. Zhao, "GCMD: Genetic Correlation Multi-Domain Virtual Network Embedding Algorithm," in IEEE Access, vol. 9, pp. 67167-67175, 2021.

5. G. Kibalya, J. Serrat-Fernandez, J. -L. Gorricho, D. G. Bujjingo and J. Serugunda, "A Multi-Stage Graph Aided Algorithm for Distributed Service Function Chain Provisioning Across Multiple Domains," in IEEE Access, vol. 9, pp. 114884-114904, 2021, doi: 10.1109/AC-

CESS.2021.3104841.

6. G. Kibalya, J. serrat, J. -L. Gorricho and P. Zhang, "A Reinforcement Learning Approach for Virtual Network Function Chaining and Sharing in Softwarized Networks," in IEEE Transactions on Network and Service Management, doi: 10.1109/TNSM.2022.3175910.

7. P. Zhang, P. Gan, L. Chang, W. Wen, M. Selvi and G. Kibalya, "DPRL: Task Offloading Strategy Based on Differential Privacy and Reinforcement Learning in Edge Computing," in IEEE Access, vol. 10, pp. 54002-54011, 2022, doi: 10.1109/ACCESS.2022.3175194.

List of conference papers

1. Godfrey Kibalya, Joan Serrat, Juan-Luis Gorricho, Doreen Gift Bujjingo, Jonathan Serugunda, and Peiying Zhang. “A Reinforcement Learning Approach for Placement of Stateful Virtualized Network Functions”. IFIP/IEEE International Symposium on Integrated Network Management 17-21 May 2021 Bordeaux, France. .

2. Godfrey Kibalya, Joan Serrat, Juan-Luis Gorricho, Rafael Pasquini, Haipeng Yao, and Peiying Zhang. “A Reinforcement Learning Based Approach for 5G Network Slicing Across Multiple Domains.” 15th International Conference on Network and Service Management (CNSM 2019), Halifax, Canada, 21-25 October, 2019

3. Godfrey Kibalya, Joan Serrat, Juan-Luis Gorricho, “RAN Slicing Framework and Resource Allocation in Multi-domain Heterogeneous Networks.” Autonomous Infrastructure, Management and Security (AIMS 2018), Munich Germany, 4-5 June, 2018.

List of Book Chapters

1. Liu F., Kibalya G., Santhosh Kumar S.V.N., Zhang P. (2022) Challenges of Traditional Networks and Development of Programmable Networks. In: Aujla G.S., Garg S., Kaur K., Sikdar B. (eds) Software Defined Internet of Everything. Internet of Things (Technology, Communications and Computing). Springer, Cham.

1.7 Thesis Outline

This section outlines the organizational structure of the thesis document. Specifically, the document is structured in four parts with the specific chapters of the different thesis parts elaborated below:

Part 1 of the thesis gives a high-level introduction to the thesis and the multi-domain orchestration problem and is constituted of two chapters. **Chapter 1** has introduced the thesis background and motivation, the problem addressed in the thesis, the thesis Objectives, the thesis positioning with respect to SDOs, state-of-art with respect to the thesis contribution and a list of scientific contributions. **Chapter 2** introduces a description and mathematical formulation of the multi-domain service

orchestration problem, and an architectural framework for the multi-domain service orchestration problem. The chapter also introduces a review of the solution techniques adopted by thesis while highlighting the reasons behind the chosen approaches. In addition, the chapter introduces the metrics and simulation environment used.

Part II of the thesis is devoted to the algorithms for solving **sub-problem 1** of the thesis involving the partitioning of the different service request components across different InPs, and is constituted of two chapters. **Chapter 3** of the thesis introduces a reliability-aware reinforcement learning based algorithm for cross-domain service orchestration. **Chapter 4** introduces the multi-stage graph based distributed algorithm for service orchestration.

Part III of the thesis introduces the algorithms for solving **sub-problem 2** of the thesis which involves orchestrating the assigned sub-SFC or an entire SFC within a single InP infrastructure. This part is constituted of three chapters. **Chapter 5** introduces and exploits a multi-stage graph algorithm to propose a set of survivable service orchestration and failure recovery algorithms with backup resource sharing. Owing to the strict reliability requirements of mission-critical applications envisaged in future networks, **Chapter 6** introduces a set of Metaheuristic algorithms, namely Genetic and Harmony Search algorithms, for fault-tolerant placement of stateful VNFs. Finally, **Chapter 7** introduces a reinforcement learning based algorithm for service orchestration with support for sharing of deployed VNFs among different applications, with a target of minimizing the operational cost incurred by a SP.

Finally, **Part IV** of the thesis Concludes the thesis with **Chapter 8** introducing a summary of thesis results, future work and the thesis conclusion.

CHAPTER 2

Multi-domain service orchestration problem background

2.1 Introduction

This chapter introduces the multi-domain service orchestration problem with focus on its general formulation as an ILP including the underlying constraints that govern the relation between the substrate network attributes (e.g., residual resources and delay) and the corresponding request attributes. Given the NP-hard nature of the problem, the chapter introduces the solution techniques adopted by the thesis while highlighting the motivation behind the chosen approaches. In addition, the general evaluation environment for assessing the performance of the algorithms proposed in the thesis is introduced including the performance metrics and the simulation environments.

The rest of this chapter is organized as follows: Section 2.2 introduces the multi-domain service orchestration problem including the models adopted for service requests and substrate networks, and a description and mathematical formulation of the problem. The key solution approaches adopted by the thesis are introduced in Section 2.3. Finally, the general performance evaluation environment including the performance metrics and the simulation settings are introduced in Section 2.4.

2.2 Multi-domain Service Orchestration Problem

Under the NFV paradigm, the different segments of a service chain may require to be orchestrated across different InPs, a process commonly referred to as multi-domain/ cross-domain service orchestration. In this section, a detailed description of the multi-domain service orchestration problem is introduced. First, the section introduces the model adopted for the service requests and the substrate network on which these are to be provisioned. Then, a description and mathematical formulation of the multi-domain service orchestration problem including the underlying constraints is introduced in the subsequent subsections.

2.2.1 Service request model

The thesis denotes by $\mathbb{R}$ as a set of all service requests contending for the resources in the network. Each request $r \in \mathbb{R}$ is modeled as a chain of VNFs through which the traffic has to traverse, and specified as a tuple $\Psi^r = < \rho^r, G^r_v, C^r_{dem}, del^r_\tau, \tau^r_s, \tau^r_d, \tau^r_f >$. The parameter ρ^r denotes the specified user traffic in terms of packet rate (i.e., packets per second) from the ingress node τ^r_s to the egress

node τ_d^r. The parameter G_v^r denotes the SFC connectivity graph modeled as a directed graph denoted by $G_v^r = (N_v, E_v)$, where N_v and E_v denote the set of VNFs to be traversed by the traffic packets and the interconnecting virtual links of the SFC respectively. We refer to each of such required VNFs as a request virtual node or simply virtual node for convenience, denoted by $n_v^p \in N_v$, where N_v denotes the set of all such nodes and $p \in P$ denotes the function type of this node (e.g., firewall or NAT, among others). Each $n_v^p \in N_v$ is considered to be characterized by: i) required amount of type q resource (e.g., CPU, memory, etc.) denoted by $dem_q^{n_v^p}$ where Q denotes a set of all resource types (e.g., CPU, storage, and memory, among others). In general, we consider the amount of resources of a given type q required by a node n_v^p to be proportional to the packet rate to be processed by this node, i.e., $dem_q^{n_v^p,r} = \rho_{n_v^p}^r \times C_q^\rho$, where $dem_q^{n_v^p,r}$ is the amount of type q resource required by n_v^p, with $\rho_{n_v^p}^r$ and C_q^ρ denoting the packet rate traversing n_v^p and the amount of type q resources required to process each unit of packet rate by a given node, respectively; ii) function type denoted by $p \in P$; iii) acceptable location region denoted by $loc^{n_v^p}$. C_{dem}^r is a set indicating the amount of resource required at the different request virtual nodes of $r \in \mathbb{R}$.

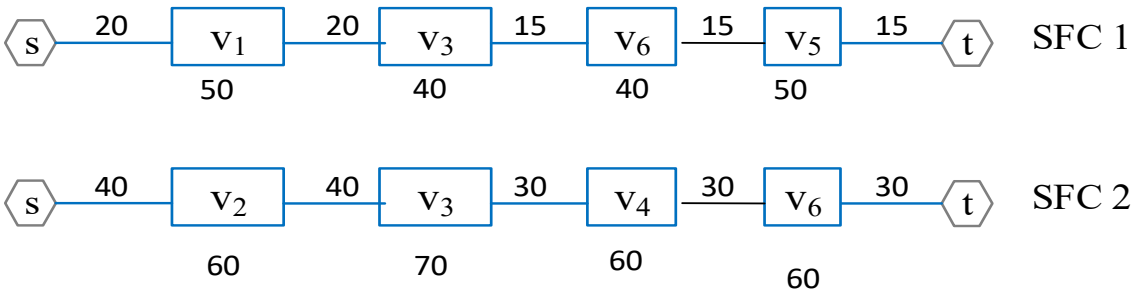


Figure 2.1: An illustration of two SFC service requests

Similarly, $l_{uv} \in E_v$ denotes the request virtual link between VNFs u and v, with the bandwidth requirement of such a link being denoted by $dem_{bw}^{l_{uv},r}$. We denote by del_τ^r and τ_f^r as the end-to-end delay requirement and life-time of the request respectively.

Figure 2.1 shows an example of two SFC requests each with a given source s and destination t. The resource requirement of each virtual node in terms of CPU is shown below the box, while the bandwidth requirement of each virtual link is shown on top of the link. Note that the bandwidth requirement may vary across different links since the packet rates may be altered by the traversed VNFs, for instance, as a result of filtering or splitting of packets due to applying some kind of networking functionality.

2.2.2 Substrate network model

The underlying substrate network on which the service requests are to be provisioned is considered to consist of a set $\mathbb{K} = \{1, 2, 3, ..., K\}$ of InPs and it is modeled as a weighted undirected graph $G_s = (N_s, E_s)$ where N_s, E_s denote the set of all physical nodes (e.g., servers) and physical links, respectively. The substrate network of a given InP/ domain $k \in \mathbb{K}$ is modeled as a weighted

undirected graph $G_s^k = (N_s^k, E_s^k)$, where N_s^k and E_s^k denote the set of substrate nodes and intra-domain substrate links within that domain, where $G_s^k \in G_s$, $N_s^k \in N_s$ and $E_s^k \in E_s$. Each physical node $n_s^k \in N_s^k$ within domain k is characterized by: i) a location specification $loc^{n_s^k}$, modeled as a point $p(x_{n_s^k}, y_{n_s^k})$, where $x_{n_s^k}$ and $y_{n_s^k}$ are the x and y Cartesian coordinates of n_s^k; ii) a set of function types that can be deployed onto this node, denoted as $\mathbb{F}_p^{n_s^k}$; iii) amount of residual resources of type q at a given time, denoted by $C_{q_{res}}^{n_s^k}$; iv) type q resource capacity denoted by $C_{q_{max}}^{n_s^k}$; v) a cost for each unit of consumed resource of type q denoted by $\varrho_q^{n_s^k}$ and finally, a power consumption κ^{n_s}. In a similar way, $e^k \in E_s^k$ denotes a single hop edge within domain $k \in \mathbb{K}$ and $e_{int} \in E_{int}$ denotes an inter-domain link, where $E_{int} \subset E_s$ denotes the set of all inter-domain links. Each link $e^k \in E_s^k$ or $e_{int} \in E_{int}$ is characterized by: i) a bandwidth capacity $B_{max}^{e^k}$ or $B_{max}^{e_{int}}$; ii) a residual bandwidth at a given time, denoted by $Bw_{res}^{e^k}$ or $Bw_{res}^{e_{int}}$; iii) a propagation delay δ^{e^k} or $\delta^{e_{int}}$, and iv) a cost per unit of consumed bandwidth resource, ϱ^{e^k} or $\varrho^{e_{int}}$, Or equivalently, a cost for transmitting each unit of packet rate ζ^e.

In this way, the substrate network parameters can be expressed as:

$$G_s = G_s^1 \cup G_s^2 \cup ... \cup G_s^K \cup E_{int}, \tag{2.1}$$

$$N_s = N_s^1 \cup N_s^2 \cup N_s^3 ... \cup N_s^K, \tag{2.2}$$

$$E_s = E_s^1 \cup E_s^2 \cup ... \cup E_s^K \cup E_{int}. \tag{2.3}$$

2.2.3 Problem description and Formulation

Given a service request to be provisioned, and an underlying substrate network owned by multiple InPs whose internal network topology and pricing information is considered confidential, the problem of service orchestration across such a multi-domain infrastructure can be defined as a mapping M from the service request graph G_v^r to a subset of the substrate graph G_s, in such a way that all the constraints associated with G_v^r are satisfied. Given the NP-hard nature of this problem [15–17], it cannot be solved in polynomial time. In literature, this problem is usually modeled as an ILP with a goal of optimizing a given objective such as mapping cost [7, 19, 47, 87] or energy [37, 88], and solved using exact solution approaches [7, 11–13, 62] or heuristics [19, 35–37, 47–50, 63, 87, 88]. In a general sense, this problem can be decomposed into three main tasks. The first task involves exploiting the request requirement specifications and the public information exposed by the different InPs to identify a set of InPs that can potentially serve that request, either partially or in full. In executing this task, it is possible that a given virtual node of a request is associated with more than one possible candidate InP. Therefore, the second task involves splitting the request among a subset of feasible InPs, among all possible candidates, in order to optimize a given mapping objective, such as provisioning cost, energy consumption, and service reliability, among others. Obtaining an optimal splitting result has been proved to be NP-hard [14]. Once, the optimal InPs for embedding the request are identified as a solution to the second task, then, the third task involves reservation and allocation of intra-domain resources within the selected InPs and along the inter-domain paths linking these InPs, in order to

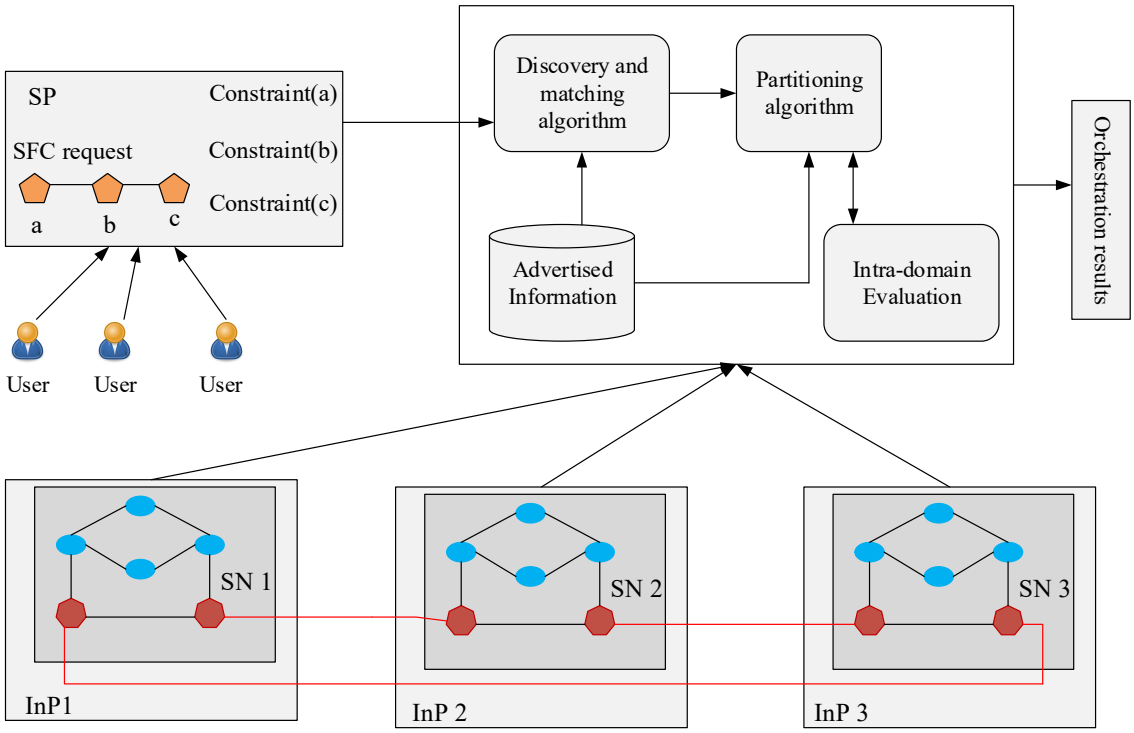


Figure 2.2: An illustration of an SFC orchestration instance

instantiate the end-to-end service, which task has also been shown to be NP-hard [16, 17].

An illustration of a multi-domain SFC orchestration instance is shown in Fig. 2.2 where a set of users place SFC requests to the SP with each request being constrained in terms of node, link and end-to-end delay requirements. Note that the SP could be an InP as well or a third party entity who leases resources from InPs. The different InPs advertise their global information which is used by the matching algorithm to identify a set of candidate InPs who meet the requirements of the service request as per the advertised information. This then serves as input into the request splitting algorithm which makes a decision regarding the final InPs for provisioning the service request based on the provisioning objective, the request constraints and the results from the intra-domain evaluation by the candidate InPs. In this simple scenario, the entire substrate network consists of three autonomous domains and their interconnections. The green ellipses inside each InP correspond to the intra-domain nodes such as servers or DCs while the maroon heptagons correspond to the boundary/peering nodes between the different InPs. The red lines connecting the heptagons correspond to the inter-domain links between the corresponding InPs.

In practice, the goal of the orchestration algorithm is to optimize a given objective such as revenue of the infrastructure provider [15, 36, 89], total energy consumption in the substrate network [37, 88–90] or QoS [91, 92], among others. Therefore, the general service orchestration problem can

be mathematically formulated as:

$$Optimise \ \ \textbf{Obj} \tag{2.4}$$

where the provisioning objective, **Obj** in Eqn. 2.4 could relate to aspects such as energy consumption, latency, reliability, and acceptance ratio, among others, or a weighted sum of these. In this thesis, the orchestration is performed by the Master Orchestrator (MO) with the objective of minimizing the average implementation cost of each request, since in practice, minimizing this results in minimizing the operation cost incurred by a SP. Therefore, the problem target is formulated as follows:

$$\textbf{Minimise} \quad \frac{1}{|R_A|} \sum_{r \in R_A} C_p^r(Gv) \tag{2.5}$$

where $C_p^r(Gv)$ is the provisioning cost for a request $r \in R_A$, and R_A denotes the set of all admitted requests, with $|R_A|$ being the cardinality of that set. $C(G_v)^r$ can incorporate multiple cost components including packet forwarding costs along the substrate edges, processing costs at the different nodes, energy cost, QoS violation cost, and VNF activation costs, among others. The specific details regarding the components influencing Eqn. 2.5 are discussed in the individual chapters of the thesis. Moreover, in order to increase competitiveness, the thesis considers that the mapping inside each domain is done with the objective of minimizing the provisioning cost for the sub-SFC that is bid for by the corresponding domain, although these are permitted to follow their own intra-domain policies.

Constraint formulation

Complementary, the optimization criterion expressed in Eqns. 2.4 and 2.5 should adhere to a number of constraints including resource constraints, location constraints, integrity constraints and domain constraints, among others. In this section, the constraints that hold for all the proposed contributions of the thesis are presented. Additional constraints that may be specific to a given proposal are presented in the respective chapters of the thesis. The underlying constraints to the above objective function are as follows:

• The resource consumption at a given substrate node should not exceed the node resource capacity for any resource type.

$$\sum_{r \in \mathbb{R}} \sum_{n_v^p \in N_v} y_{n_s^k}^{n_v^p} \times dem_q^{n_v^p, r} \leq C_{q_{max}}^{n_s^k} \ \ \forall n_s^k \in N_s^k, k \in \mathbb{K}, q \in Q \tag{2.6}$$

where $y_{n_s^k}^{n_v^p} \in \{0, 1\}$ =1 if n_v^p is provisioned on n_s^k, zero otherwise.

• The bandwidth consumption on a given edge $e^k \in E_s^k$ or $e_{int} \in E_{int}$ should not exceed the resource capacity of that edge.

$$\sum_{r \in \mathbb{R}} \sum_{l_{uv} \in L_v} \sigma_{l_{uv}}^{e^k} \times dem_{bw}^{l_{uv}, r} \leq B_{max}^{e^k} \ \forall e^k \in E_s^K \tag{2.7}$$

$$\sum_{r \in \mathbb{R}} \sum_{l_{uv} \in L_v} \sigma_{l_{uv}}^{e_{int}} \times dem_{bw}^{l_{uv}, r} \leq B_{max}^{e_{int}} \ \forall e_{int} \in E_{int} \tag{2.8}$$

where $\sigma_{l_{uv}}^{e^k} \in \{0,1\}$=1 if virtual link l_{uv} is provisioned on intra-domain substrate edge e^k, zero otherwise. Likewise, $\sigma_{l_{uv}}^{e_{int}} \in \{0,1\}$ =1 if l_{uv} is provisioned on inter-domain substrate edge e_{int}, zero otherwise.

• The end-to-end delay should not exceed the acceptable delay of the request.

$$\sum_{l_{uv} \in L_v} \sum_{k \in K} \sum_{e^k \in E_s^k} \sigma_{l_{uv}}^{e^k} \delta^{e^k} + \sum_{l_{uv} \in L_v} \sum_{e_{int} \in E_{int}} \sigma_{l_{uv}}^{e_{int}} \delta^{e_{int}} + \sum_{n_v^p \in N_v} y_{n_s^k}^{n_v^p} \times \delta_{vnf}^p \leq Del_{sd}^r \ \ \forall r \in \mathbb{R} \tag{2.9}$$

where δ_{vnf}^p denotes the processing delay experienced by a packet at a VNF of type p. The first and second terms of equation 2.9 correspond to the propagation delay of the intra-domain and inter-domain edges, respectively, and the third term corresponds to the processing delay at the different VNFs traversed by the user traffic.

• Each request virtual node must be mapped onto a single substrate node.

$$\sum_{k \in K} \sum_{n_s^k \in N_s^k} y_{n_s^k}^{n_v^p} = 1 \ \ \forall n_v^p \in N_v \tag{2.10}$$

•Each request virtual node should be provisioned on a substrate node that is within its acceptable geographical location.

$$loc^{n_s^k} \in loc^{n_v^p} \ \ \forall \ n_v^p \in N_v \tag{2.11}$$

where $loc^{n_s^k}$ denotes the location coordinates of node n_s^k and $loc^{n_v^p}$ denotes the acceptable location region of virtual node n_v^p.

• Each VNF of type p should be provisioned on a substrate node capable of supporting that type of VNF:

$$y_{n_s^k}^{n_v^p} = 1 \ iff \ \ p \in \mathbb{F}_p^{n_s^k} \ \ \forall n_v^p \in N_v, p \in P \tag{2.12}$$

where $\mathbb{F}_p^{n_s^k}$ denotes a set of function types that can be provisioned on n_s^k.

• Similarly, a request virtual node n_v^p is provisioned by substrate node n_s^k only if there is a VNF of type p already provisioned on that node.

$$y_{n_s^k}^{n_v^p} = min\{y_{n_s^k}^{n_v^p}, \gamma_p^{n_s^k}\} \tag{2.13}$$

where $\gamma_p^{n_s^k} \in \{0,1\}$ is equal to 1 if a VNF of type $p \in P$ is already provisioned on substrate node n_s^k, zero otherwise.

The problem as formulated above becomes an NP-hard problem. As such, solving it using conventional solvers like CPLEX or Gurobi is not feasible in terms of execution time, especially when dealing with large scale networks. In this regard, exact approaches proposed in [7,11–13,62] are not suited for scenarios involving practical delay sensitive online arrival of requests. This motivates the adoption of heuristics and meta-heuristic approaches introduced in this thesis and those we

proposed in [86, 93] that have a capability to realize near-optimal solutions in feasible run-times, while preserving the privacy of the participating InPs. The different solution techniques adopted by the thesis are introduced in Section 2.3.

2.3 Solution techniques adopted by the thesis

The problem being addressed by the thesis is NP-hard, making it not practical to realize optimal solutions in practical run time especially when considering large problem instances. This is further exacerbated by the dynamic attributes of the substrate network and service requests, and a large number of components that may affect the global orchestration objective. Therefore, given the complexity of the problem, the thesis targeted to propose solution approaches that are:

- robust and less problem specific, hence can be tailored to different optimization objectives, network topologies and service request scenarios and constraints.
- able to incorporate information learned in the previous solutions search space and historical orchestration decisions in making the current orchestration decisions, hence resulting in acceptable performance even in scenarios of limited information exposure and fuzzy environments.
- capable of dealing with an optimization problem that is jointly affected by multiple attributes.
- able to realize near-optimal solutions in practical run-times.
- able to demonstrate good generalization capability by intelligently performing a trade-off between exploration and exploitation in the search space.

In light of the above, the thesis adopted reinforcement learning based approaches in Chapters 3, 7 and the proposal in [78]. Metaheuristic approaches namely Genetic algorithm and Harmony search are adopted in the fault tolerant VNF placement problem addressed in Chapter 6 and in the cross-domain orchestration proposal in [86]. In Chapters 5 and 4, the thesis adopts a heuristic approach based on a multi-stage graph which can be flexibly tailored to different mapping objectives while guaranteeing to execute in polynomial time. In the sub-sections below, we give a brief overview of the key solution techniques adopted by the thesis with focus limited to the thesis scope, while justifying their choice for the specific sub-problems being addressed.

2.3.1 Reinforcement Learning

Reinforcement learning (RL) agents learn to make optimal decisions through experience obtained by interacting with the environment, hence, are able to learn even in the absence of a label set provided by a knowledgeable external supervisor. This is a key strength since in practice, it is not possible to obtain examples of all behaviors that are representative of all the situations in which the RL agent is expected to act. By exploiting the experience from previous actions and rewards, such an agent is able to learn good policies that improve the future reward, hence, by aligning the reward signal with the orchestration objective specified in Eqn. 2.5, the agent is able to make orchestration decisions that result in low operational costs being incurred by a given SP. Moreover, in the resource management

domain, attributes such as, traffic load characterizing the environment, are usually repetitive, with certain predictable temporal correlations, which enables the RL agent to be trained either offline or online as the environment executes.

Besides the agent and the environment with which it interacts, RL relies on four key elements [94]:

- A policy which defines the behavior of the RL agent at a given time, and can be viewed as a mapping from perceived states of the environment to actions to be taken when in those states. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine the agent's behavior.
- A reward signal which defines the goal in a reinforcement learning problem whose numerical value the RL agent targets to maximize over the long run while choosing the actions to execute under the different states it encounters. The reward is the primary basis for altering the policy, in that, if an action selected by the policy is followed by low reward, then the policy may be changed to select some other action if a similar or close state is encountered in the future.
- Value functions which can be thought of as a measure of the *goodness* given the current state of the environment and the policy π to be followed by the agent. Whereas the reward signal indicates what is good in an immediate sense, a value function specifies what is good in the long run with the value of a given state being related to the total amount of reward an agent can expect to accumulate over the future, starting from that state. Thus the value computation of a given state takes into account the states that are likely to follow, and the rewards available in those states.
- A model which allows inferences to be made about the behavior of the environment, for instance predicting the next state and reward given the current state and action to be taken in that state. However, given the dynamic nature of the environment, it may not be possible to obtain accurate models in most practical problems. Therefore, in this thesis, model-free methods are adopted wherein, knowledge of the environment model is not required.

Markov Decision Process

The RL problem considers an existence of a Markov property between state transitions. The Markov Decision Process (MDP) is modeled as a 4-tuple (S,A,P,R,γ) where S and A represent two finite sets of states and actions respectively while P is a transition model mapping $S \times A \times S$ into probabilities in [0,1] and given as [95]:

$$P^{a}_{ss'} = P[S_{t+1} = s' | S_t = s, A_t = a] \tag{2.14}$$

where S_t and A_t are the states and actions at time t respectively. The term R is a reward function mapping $S \times A \times S$ into real-valued rewards and given by:

$$R^{a}_{s} = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] \tag{2.15}$$

where R_s^a denotes the immediate reward when action a is performed when in state s. The parameter $\gamma \in [0, 1]$ is a discount factor for evaluating the total discounted reward G_t starting at time t. The total discounted reward G_t can be evaluated as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{k=\infty} \gamma^k R_{t+k+1} \tag{2.16}$$

The algorithms for solving MDPs target to return a policy π that maps from S to A a real valued function $V_\pi(s)$ on states, or a real valued function Q-function, $Q_{\pi(s,a)}$ on state-action pairs. The action policy π is a distribution of actions at state s, and indicates the probability of taking action $a \in A$ while in state $s \in S$ and is expressed as:

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{2.17}$$

In this regard, the goal of the RL agent is to find an optimal policy π^* or equivalently (V^* or Q^*) that maximizes the expected total discounted reward of the agent where V^* and Q^* are the optimal state-value and action-value functions respectively [95]. The state-value function $V_\pi(s)$ is related to the current state s, and can be evaluated as [94]:

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t|S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{k=\infty} \gamma^k R_{t+k+1} \right] \tag{2.18}$$

where $\mathbb{E}_\pi[.]$ is the expected value of a random variable given that the agent follows policy π. Similarly, the value of taking action $a \in A$ in state $s \in S$ under a policy π expressed by the action-value function $q_\pi(s, a)$ can be defined as the expected return starting from s, taking the action a, and thereafter following policy π. This is evaluated as [96].

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t|S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{k=\infty} \gamma^k R_{t+k+1} \right] \tag{2.19}$$

For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|St = s \right] \tag{2.20}$$

$$= \mathbb{E}_\pi \left[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + ...|St = s \right] \tag{2.21}$$

$$= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1}|S_t = s \right] \tag{2.22}$$

Accordingly, there is a recursive relation between the state S_t and S_{t+1} which can be expressed as:

$$= \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi S_t + 1|S_t = s \right] \tag{2.23}$$

The above successive relation expressed in Eqn.2.22 is commonly reffered to as the Bellman equation for V_π and it expresses a relationship between the value of a state and the values of its successor states. By a similar approach, the Bellman equation for the action-value function $q_\pi(s,a)$ can be expressed by [96]:

$$q_\pi(s,a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a \right] \tag{2.24}$$

Moreover, a translation relationship between $V_\pi(s)$ and $q_\pi(s,a)$ can be established as:

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s|a) \tag{2.25}$$

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'^a} V_\pi(s') \tag{2.26}$$

In this thesis, we consider that a decision making agent receives a service request with a set N_v of VNFs for provisioning across different substrate nodes within or across different domains. In this regard, the agent has to make $|N_v|$ independent decisions in order to provision all the VNFs of the request. In case we are to use value-based approaches such as Q-learning algorithm for provisioning the service request in such a problem, we need to calculate the reward obtained by executing different actions under each state and choose the action with the largest reward. Given that the the state space of our problem is made up of continuous values and we cannot get the transition probability distribution between different states, value-based RL is not well suited for the thesis problem. In this way, the thesis adopts a policy-based RL approach which targets to optimize the policy of actions directly as discussed in the subsection below.

Policy Gradient

Policy gradient algorithm is a reinforcement learning algorithm which optimizes the policy of actions directly. In this way, the algorithm instead learns a parameterized policy that can select actions without consulting a value function. Although a value function may still be used to learn the policy weights, in principle, it is not required while making the action selection decisions. At the beginning, the policy π can be described as a function containing the parameter θ, denoting the policy weight vector as:

$$\pi_\theta(s,a) = P(a|s,\theta) \approx \pi(a|s) \sum_{s' \in S} P_{ss'^a} V_\pi(s') \tag{2.27}$$

After the policy function is represented as a continuous function, we can use continuous function optimization methods such as gradient descent algorithm to optimize the strategy. The optimization function can be expressed by:

$$J(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) R_s^a \tag{2.28}$$

where θ denotes the parameters of policy gradient algorithm while $d^{\pi_\theta}(s)$ is the probability distribution of states. By invoking the likelihood ratio:

$$\Delta_\theta \pi_\theta(s,a) = \pi_\theta(s,a)\frac{\Delta_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} \tag{2.29}$$

$$= \pi_\theta(s,a)\Delta_\theta log\pi_\theta(s,a) \tag{2.30}$$

Then, the gradient of the objective function $J(\theta)$ can be evaluated as:

$$\Delta_\theta J(\theta) = \sum_{s\in S} d(s) \sum_{a\in A} \pi_\theta(s,s)\Delta_\theta log\pi_\theta(s,a)R_s^a \tag{2.31}$$

$$= \mathbb{E}_{\pi_\theta}\left[\Delta_\theta log\pi_\theta(s,a)r\right] \tag{2.32}$$

where r is the total reward over the entire process. Furthermore, according to the policy gradient theorem [94], under the multi-step MDP, we have:

$$\Delta_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\Delta_\theta log\pi_\theta(s,a)Q^{\pi_\theta}(s,a)\right] \tag{2.33}$$

where $Q^{\pi_\theta}(s,a)$ is the sum of the multi-step reward. During the actual optimization process, unbiased sampling is performed on $Q^{\pi_\theta}(s,a)$, with the weight parameters being updated as:

$$\theta = \theta + \alpha\Delta_\theta log\pi_\theta(s_t,a_t)v_t \tag{2.34}$$

2.3.2 Genetic Algorithm

The genetic algorithm (GA) is an optimization Metaheuristic inspired by the evolution theory. Although GA does not guarantee to find the optimal solution of the optimization problem, it is able to obtain acceptable solutions, in a competitive time with the rest of the combinatorial optimization algorithms such as simulated annealing and sequential search methods, among others [97]. The algorithm has previously been applied to solve a number of problems in the network and service management domain including VNE [98], multi-domain service orchestration [99], service function chaining [100–103], resource prediction [104] and load balancing [105], among others. The general execution procedures of a GA involves the following steps [97, 106, 107]:

1. Generate an initial population: This involves generating an initial set of random values called chromosomes where each chrome is composed of a string of values called genes. Each chrome is a probable solution to the optimisation problem under consideration.
2. Evaluate the fitness of each individual in the population: Each individual in the population is evaluated for its fitness value using a fitness function. Usually, the fitness value corresponds to degree to which the solution optimizes the objective function under consideration.
3. Select individuals from the population to be parents: This involves using the selection operator

to select chromosomes for later recombination. Solutions that performed best in terms of fitness value are usually preferred, since these are more likely to result in better off-springs. However, it is important not to completely discard weaker chromes since doing this may result in premature convergence. The most commonly used selection algorithms are tournament selection and roulette wheel. In the Tournament selection, k random chromosomes are selected from the population, then, the chromosome having the highest fitness value is used as a parent for crossover. When k (also known as the selection pressure) is chosen to be too small, small chromosomes have more probability of being selected, and the probability of selecting the best chromosomes increase with increase in the value of k, though with a low diversity in the population. In the roulette wheel selection, each chromosome i is associated with a probability p_i of being chosen which is directly related to its relative fitness value, and computed as:

$$p_i = \frac{fitness_i}{\sum_{k=1}^{n} fitness_k} \tag{2.35}$$

Where $fitness_i$ is the fitness value for chromosome i, and n is the total number of chromosomes in the population.

4. Produce children from the selected parents: Upon selection of parents for replication, then the crossover operator is applied to generate new children. The idea behind the crossover operator is that useful segments (i.e., partial solutions) of the selected parents should be combined in order to yield new individuals which will lead to better solutions over time. The commonly adopted crossover operators are one-point, multi-point, and uniform crossover. In one-point crossover, a crossover point is randomly generated and then segments of the two parents strings are swapped to produce one or two child strings. A multi-point crossover involves generating multiple crossover points, followed by exchanging every second segment between subsequent crossover points. In a uniform crossover, each gene in the child solution is created by copying the corresponding gene from one or other parent, chosen according to a binary random number generator U(0,1). If the random number is a 0, the gene is copied from the first parent; if it is a 1, the gene is copied from the second parent. However, in order to avoid premature convergence, a mutation operator is usually employed on the child population with a purpose of providing a small amount of random search. It also serves the purpose of reintroducing "lost" information into the population due to premature convergence. The mutation operator works by inverting each bit (in binary representation) in the child solution with some small probability.

5. Evaluate the fitness of the children and update the population: The fitness of the generated child solutions is evaluated as these are used to replace a number of members in the population so as to keep the population size constant. Two techniques are commonly adopted to ensure a constant population size, thus a generational approach and steady-state or incremental approach. In the generational approach, the number of child solutions generated is equal to the size of the parent population, therefore, the entire parent population is replaced in one generation.

However, this does not guarantee that the best solution obtained from the optimization so far will survive into the next generation. The steady-state approach generates and replaces only a few members of the parent population during each generations, usually the ones whose fitness values is below the average fitness of the population. Such an approach ensures that the best solutions found so far are able to survive through the next generation, albeit at an increased computation complexity since the selection probabilities have to be re-calculated every time a new child enters the population.

6. Repeat steps 3-5 until a solution with satisfied fitness is found or termination criterion is met.

The execution procedure of a general GA is shown in Fig. 2.3.

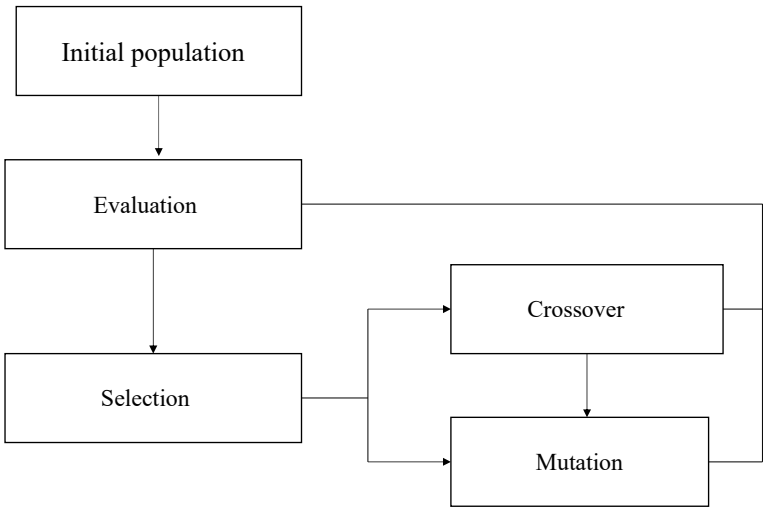


Figure 2.3: Execution procedure of the Genetic Algorithm

2.3.3 Harmony Search

The Harmony search [HS] algorithm is a recent random search Metaheuristic proposed in [108] which is inspired by the underlying principles of the musicians improvisation of the harmony by trying out different combinations of music pitches stored in their memory [109]. Although the algorithm is recent, it has already shown promising performances to the VNE problem in [110]. The strength of the HS algorithm stems from the fact that it does not require prior domain knowledge such as gradient information of objective functions. Moreover, it uses a single search memory to evolve, giving it a distinguishing feature of a higher level of computational simplicity and search efficiency compared to other population based approaches [109]. The algorithm execution involves 4 major steps [109, 111]:

- *Step 1.* Initialization of the HS Memory (HM): This involves populating the HM with initial solutions to the optimization problem being addressed. Considering an n dimension problem,

an HM with size HMS can be represented as follows:

$$HM = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 & ... & x_n^1 \\ x_1^2 & x_2^2 & x_3^2 & ... & x_n^2 \\ ... & ... & ... & ... & ... \\ x_1^{HMS} & x_1^{HMS} & x_1^{HMS} & ... & x_n^{HMS} \end{bmatrix}$$

where $x = [x_1^i, x_2^i, x_3^i, ...x_n^{HMS}](i = 1, 2, 3, ..., HMS)$ is a possible solution to the optimization problem. The j^{th} component of the i^{th} solution x_j^i could for-example represent a possible node (e.g., server, InP, Virtual machine, etc.) for hosting the j^{th} VNF of a given SFC considering the SFC embedding problem.

Although the initial solutions of HM are randomly generated during this step, in order to enable the HS to obtain better solutions quickly, it may be advantageous to start from a feasible or refined set of initial solutions that could be obtained from another heuristic technique or by applying some filtering technique based on for instance location or residual resource constraints. Note that choosing a big value of HMS increases the number, hence diversity of the initial solutions, albeit at a cost of slow convergence. On the other hand, a small value of HMS results in faster algorithm convergence but most likely with a sub-optimal solution, due to the reduced diversity of the initial solutions.

- *Step 2*. Improvisation of a new solution. This involves generation of a new solution based on the initial HM solutions generated in step 1. The j^{th} component of the new solution is chosen as the j^{th} component of a solution chosen from HM with a probability β given by the Harmony Memory Considering/accept Rate (HMCR), or generated randomly with a probability (1-β). In principle, HMCR defines the probability of selecting a solution component from the HM members. This parameter directly relates to the chances of the best solutions/harmonies in the HM being carried to the new memory. If HMCR is chosen to be too low, then few of the best solutions in the current HM are passed to the next HM, which may lead to a slow convergence of the algorithm. On the-other hand, if the value is chosen to be too high, then most of the components of the new solution are chosen from the current HM, which increases the chances of best solutions being reflected in the new HM in the next iteration, but reducing the exploration rate of new possible solutions potentially resulting in local optima or wrong solutions. If the solution component is selected from the HM members, then such a component is mutated to a new value with a probability α called the Pitching Adjust Rate (PAR), which is the probability with which a solution component selected from a member of HM is mutated. A low PAR value can slow down the convergence of HS due to the limitation of exploring only a small subspace of the the whole search space. On the other hand, a very high PAR may cause the solution to scatter around some potential optima as in a random search [111].
- *Step 3*. Updating of the HM: This involves evaluating the fitness of the new solution generated in step 2, and if it yields a better fitness value than the worst member in HM, then the worst solution is replaced by the new solution, otherwise, the new solution is rejected.

- *Step 4*. Repeat Steps 2-3 until a preset termination condition is satisfied such as maximum number of iterations or when there is no improvement in the solution for a given number of consecutive iterations.

The pseudo-code for the general HS execution procedure is given in Algorithm 1.

Algorithm 1 Pseudo-code of the Harmony Search Algorithm

Input: Substrate network graph G_s, Request tuple Ψ^r, fitness function, PAR, HMCR, HMS
Output: Best solutions, S_{best}
step 1: Initialize the HS memory
for *i=1 to HMS* **do**
  for *j=1 to N* **do**
    randomly initialize x_i^j in HM
  end
end
t←0
while *t< maximum iterations* **do**
  step 2: Improvise a new solution x^{t+1} and compute its fitness
  for *j=1 to N* **do**
    if *rand(0,1)<HMCR* **then**
      Select x_j be the j^{th} dimension of randomly selected member from HM
      if *rand(0,1)<PAR* **then**
        Mutate x_j
      end
    end
    else
      Randomly assign x_j from the feasible values
    end
  end
  step 3: Update HM:
  if *fitness of x is better than worst solution in HM* **then**
    Replace worst solution in HM by x
  end
  else
    Discard x
  end
  t←t+1
end
Return S_{best}

2.4 General performance evaluation environment

This section describes the general environment used for evaluating the performance of the algorithms proposed by the thesis. Specifically, the section introduces the performance metrics, and the general simulation settings regarding the physical substrate network and service request. However, the specific values of the simulation parameters are tabulated in the specific chapters, since these might have

slight variations across the different experiments in the different chapters depending on the intention of each experiment. Moreover, all experiments were conducted using a simulator implemented in python and executed on a desktop computer running a Windows Operating System with the following features: Intel(R) Core(TM) i7-8700K CPU @ 3.70GHZ and 64GB of RAM.

2.4.1 Performance metrics

The performance of the thesis proposals is evaluated against chosen benchmark algorithms considering a number of performance metrics, including: acceptance ratio , revenue to cost ratio, average cost per accepted SFC request, and the request processing time, among others. These are commonly used metrics in the literature for assessing the performance of service embedding algorithms [7, 36, 82, 89]. The various metrics are discussed below:

Average acceptance ratio, AR

This is computed as the ratio of the number of successfully accepted requests to the total number of arriving requests, i.e., the sum of both accepted and rejected. The AR metric is a direct indicator of the algorithm efficiency in using the resources of the substrate network. Therefore, an orchestration algorithm should target a high AR performance in order to maximize the revenue returned to the Service Provider, with a constraint that there is no violation or degradation of the QoS of the admitted users. This is computed as follows:

$$AR = \frac{No.\ of\ successfuly\ provisioned\ requests}{Total\ number\ of\ requests} \tag{2.36}$$

Average provisioning cost, APC

Within the scope of this work, the APC relates to the total cost incurred by a SP while provisioning a given service request. This cost may be related to consumption of both node and link resources, QoS violation penalties, resource fragmentation, VNF instantiation, and energy costs, among others. The average provisioning cost per admitted service request is computed as:

$$APC = \frac{1}{|R_A|} \sum_{r \in R_A} P_c^r \tag{2.37}$$

where P_c^r is the cost incurred by a SP while provisioning request $r \in R_A$, where R_A denotes the set of all accepted service requests and $|R_A|$ is the cardinality of that set. In order to realize a high value of net revenue, the provisioning algorithm should result in a low APC value.

Average revenue-to-cost ratio, $R2C$

In practice, the infrastructure provider is interested in maximizing the net revenue from mapping a given request, which can be expressed as the difference between the obtained revenue and the total cost incurred in provisioning a service request. The commonly used performance metric for this

purpose in literature is the revenue-to-cost ratio. The average revenue-to-cost ratio per admitted request is expressed as:

$$R2C = \frac{1}{|R_A|} \sum_{r \in R_A} r2c^r \tag{2.38}$$

where $r2c^r$ is the revenue-to-cost ratio of request $r \in R_A$ where R_A is the set of all admitted request.

Average revenue, Rev

This metric is used to express the average revenue over time obtained by the SP. This is computed as the monetary return from the use of the demanded CPU and bandwidth resources. If we denote by $rev^r(G^v)$ as the revenue received by a service provider from provisioning a request $r \in R_A$ with R_A denoting the set of all admitted request, then, the total revenue from all admitted requests is defined as:

$$Rev_{total} = \sum_{r \in R_A} rev^r(G^v) \tag{2.39}$$

Then, the average revenue obtained from each admitted request can be evaluated as:

$$Rev = \frac{1}{|R_A|} \sum_{r \in R_A} rev^r(G^v) \tag{2.40}$$

Average request provisioning time, Avg_T

This is the average time it takes for the service deployment algorithm to compute a provisioning solution for any admitted request. Aware that future services will have stringent latency start-up requirements, a useful service deployment algorithm must work with a low Avg_T. This is computed as:

$$Avg\_T = \frac{1}{|R_A|} | \sum_{r \in R_A} tim^r_{prov} \tag{2.41}$$

where $R_A \in \mathbb{R}$ denotes the set of all admitted requests, and tim^r_{prov} denotes the time taken by the algorithm to obtain a deployment solution for request $r \in \mathbb{R}$.

Average Virtual to Substrate Link Ratio, $VSLR$

This metric quantifies the average number of substrate links/edges that are allocated for each virtual link of the request, and it is a direct measure of the efficiency of the algorithm relative to the link resource utilization. If we denote by $y^{e,r}_{l_{uv}} \in \{0, 1\}$ a binary variable, equal to 1 if resources on substrate edge $e \in E_s$ are assigned to the virtual link l_{uv} of the SFC $r \in \mathbb{R}$, zero otherwise; the virtual to substrate link ratio, for a given SFC request $r \in \mathbb{R}$, can be computed as the total number of virtual links in the request divided by the total number of substrate links assigned to this request, and expressed as follows:

$$vslr^r = \frac{|E_v|}{\sum_{ij \in E_V} \sum_{e \in E_s} y^{e,r}_{l_{uv}}} \tag{2.42}$$

where $|E_v|$ is the total number of virtual links in the SFC request. The average virtual to substrate link ratio per request can then be computed as:

$$VSLR = \frac{1}{|R^p|} \sum_{r \in R^p} vslr^r \tag{2.43}$$

where R^p is the set of all admitted requests. A desirable orchestration algorithm should result in low values of VSLR in order to result in a low bandwidth consumption.

Load Balancing, LB

This parameter gives an indicator of the ability of an algorithm to use the substrate resources in a uniform manner. Load balancing has the inherent advantage of minimizing resource bottlenecks which guarantees an improved long term performance in terms of acceptance ratio. Moreover, it is also one way of ensuring resilience within the network since the traffic is uniformly distributed across the different resources. The thesis adopts the variance of the node and the link resource consumption as the measure of the load balancing in the network. The average node load balancing can be computed as follow:

$$LB(N_s) = \frac{\sum_{n^s \in N_s} (U(n^s) - \mu)^2}{|N_s|} \tag{2.44}$$

where N_s is the set of all substrate nodes in the network and $|N_s|$ is the cardinality of this set. $U(n_s)$ is the average cpu utilization at the substrate node $n_s \in N_s$ and μ is the mean value of the cpu resource utilization across the substrate network for each time unit. In this work, $U(n_s)$ is computed as follows:

$$U(n_s) = \frac{1}{T} \sum_{\forall t \in T} \frac{C_t^u(n_s)}{C(n_s)} \tag{2.45}$$

where $C_t^u(n_s)$ is the consumed cpu resource at node $n_s \in N_s$ at time $t \in T$ and $C(n_s)$ is the maximum available cpu at this node.

Similarly, the load balancing performance, $LB(L_s)$ across the links of the substrate network is computed as follows:

$$LB(L_s) = \frac{\sum_{l_s \in L_s} (U(l_s) - \mu)^2}{|L_s|} \tag{2.46}$$

where L_s is the set of all substrate edges in the network and $|L_s|$ is the cardinality of this set. $U(l_s)$ is the average bandwidth consumption on the substrate link $l_s \in L_s$ and μ is the mean value of this parameter across the substrate network. The bandwidth consumption $U(l_s)$ is computed as follows:

$$U(l_s) = \frac{1}{T} \sum_{\forall t \in T} \frac{Bw_t^u(ls)}{Bw(ls)} \tag{2.47}$$

where $Bw_t^u(ls)$ is the consumed bandwidth resource on the link $l_s \in L_s$ at time $t \in T$ and $Bw(ls)$ is the maximum available bandwidth on this link. From equations 2.44 and 2.46, the lower the value

of $LB(N_s)$ and $LB(L_s)$, the better the load balancing performance.

Number of failures

This parameter is used to evaluate the performance of the migration-aware algorithm. It gives a measure of the number of VNFs and virtual links of the low priority users that are preempted from the allocated resources throughout the request life-time. The average number of node failures/preemptions per admitted request is computed as :

$$A\_N\_F = \frac{\sum_{r \in R_{nc}} \sum_{t \in T} \sum_{n_v \in N_v} y_{n_v,r}^{n_s,t}}{R_{nc}} \quad \forall n_s \in N_s \tag{2.48}$$

where $y_{n_v,r}^{n_s,t} \in \{0,1\}$ equal to 1 if VNF $n_v \in N_v$ is preempted from substrate node $n_s \in N_s$ at time t. Similarly, the average number of link failures per admitted request is evaluated as:

$$A\_L\_F = \frac{\sum_{r \in R_{nc}} \sum_{t \in T} \sum_{i,j \in N_v} f_{ij,r}^{p_s^{qm}}}{R_{nc}} \quad \forall q, m \in N_s \tag{2.49}$$

where $f_{ij,r}^{p_s^{qm},t} \in \{0,1\}$ is equal to 1 if virtual link ij of SFC request $r \in R_{nc}$ is preempted from substrate path $p_s^{qm} \in P^{qm}$ at time t, zero otherwise, with VNF i and j respectively provisioned on substrate nodes q and m.

Average number of Service Restoration Failures, SRF

This is defined as the ratio of the number of failed/unsuccessful service restoration attempts to the total number of service restoration attempts, averaged across all time windows. This is computed as follows:

$$SRF = \frac{1}{T} \sum_{\forall t \in T} (\frac{1}{N_a} \times No.\ of\ unsucessful\ remapping\ attempts) \tag{2.50}$$

where N_a denotes the number of service restoration attempts within a given time window $t \in T$.

Average number of node and link migrations

This metric quantifies the fraction of surviving VNFs/virtual links that are migrated to new substrate nodes/paths during the service restoration phase. If we denote by $x_{n_v}^{mig,t} \in \{0,1\}$ a binary variable, equal to 1 if a surviving VNF n_v is migrated to another substrate node during the time window $t \in T$, zero otherwise, then, the average number of node migrations is calculated as:

$$Node_{mig} = \frac{1}{T} \sum_{t \in T} \frac{1}{|R_{nc}^p|} \sum_{r \in R_{nc}^p} \frac{1}{N_v^r} \sum_{n_v \in N_v} x_{n_v}^{mig,t} \tag{2.51}$$

where R_{nc}^p denotes the set of all low priority requests that are successfully remapped to new resources and N_v^r denotes the number of surviving VNFs of the request $r \in R_{nc}^p$. Similarly, if we denote by

$x_{ij}^{mig,t} \in \{0,1\}$ a binary variable, equal to 1 if a surviving virtual link e_v is migrated to another substrate path during time window $t \in T$, zero otherwise, then, the average number of virtual link migrations is computed as:

$$Link_{mig} = \frac{1}{T}\sum_{t\in T}\frac{1}{|R_{nc}^{p}|}\sum_{r\in R_{nc}^{p}}\frac{1}{E_v^r}\sum_{e_v\in E_v} x_{ij}^{mig,t} \tag{2.52}$$

Average service restoration time, Avg_T

This is the average time taken to obtain a service restoration solution for each preempted low priority request:

$$Avg\_T = \frac{1}{|R_{nc}^{p}|}\sum_{r\in R_{nc}^{p}} tim_{rst}^{r} \tag{2.53}$$

where tim_{rst}^{r} denotes the total time used to obtain a restoration solution for request $r \in R_{nc}^{p}$.

Average number of edges per admitted request,E_r

This relates to the average number of substrate edges used to provision each request on average. This is computed as follows:

$$E_r = \frac{1}{R_A}\sum_{r\in\mathbb{R}}\sum_{e\in E_s}\sigma_e^r \tag{2.54}$$

where $\sigma_e^r \in \{0,1\}$ is a binary variable equal to 1 if substrate edge $e \in E_s$ is used to provision request $r \in \mathbb{R}$, zero otherwise.

2.4.2 Simulation environment and settings

This section introduces the simulation environment regarding the physical substrate network topology and the service request models adopted by the thesis.

Network Topology

The thesis considers a substrate network composed of K InPs, where $K = 1$ for the single substrate intradomain proposals and $K > 1$ for the multi-domain proposals. For the internal topologies of the different InPs, the thesis considers both: real network topologies, and synthetic network topologies, generated using the Waxman algorithm with $\alpha = 0.5$ and $\beta = 0.3$. The real topologies include ChinaNet [22] with 55 nodes, Abilene [112] with 10 nodes and BIC [112] with 33 nodes. For the synthetic topology, the number of physical substrate nodes inside each InP is chosen according to the scenario under consideration with the connection probability between the different nodes fixed at 0.5. For both topologies, the resource capacity of the substrate links and nodes,and the propagation delay on each substrate edge, are chosen to follow a uniform distribution. The cost of processing and transmitting 1GB of data at each node and link follows a uniform distribution $U(\$0.15, \$0.22)$ and $U(\$0.05, \$0.12)$ respectively. This is aligned with common charging prices like those applied by Amazon EC2. The processing delay of a packet at each NF follows a uniform

distribution $U(0.045ms, 0.3ms)$, with the processing delay of a service chain being the sum of the processing delays of the constituent NFs.

SFC requests

The thesis considers two kinds of request behavior scenarios i.e., offline and online scenarios. In the offline case, all the requests to be served, including their attributes, are known in advance, and these, once admitted, do not leave the system for the entire simulation window time. Therefore, the resources allocated to these requests cannot be reused by other demands. Such a consideration gives a clearer insight into the algorithm's ability to deal with permanent loading stress. In the online case, the demands continuously arrive to the system with a given arrival distribution and with a finite life-time. In this case, the resources assigned to an admitted request are reclaimed upon expiry of this demand. We consider the arrival of such requests to follow a Poisson distribution with a mean value chosen according to the scenario and experiment under consideration. In addition, the life-time of each online request is exponentially distributed with a mean value chosen according to the experiment under consideration.

Each request $r \in \mathbb{R}$ is generated with a random source τ_s^r and a random destination τ_d^r from G_s, with $\tau_s^r \neq \tau_d^r$. The resource demand in terms CPU and bandwidth, the acceptable end-to-end delay, and the required packet rate of each request, are chosen to follow a uniform distribution with the specific values chosen according to the scenario under consideration. The thesis considers 5 categories of network functions: Firewalls, Proxies, NATs, DPIs and Load Balancers, with their computing resource demands adopted from [113]. The number of VNFs constituting each SFC instance is set different depending on the scenario under consideration.

PART II: Cross-domain Service Orchestration Algorithms

CHAPTER 3

Reliability-Aware Deep Reinforcement Learning-Based Algorithm for Multi-domain Service Orchestration

3.1 Introduction

As the Standardization Development Organizations (SDOs) perceive NFV as an integral component of 5G and future networks, innovative approaches for ensuring service reliability are pertinent especially in a multi-domain setting. Considering that the different InPs comprising the substrate network may be characterized by differing levels of service reliability guarantee, this chapter enhances our contribution in [86, 93] by incorporating service reliability in the cross-domain service orchestration problem while ensuring efficient utilization of network resources. While considering single domain service orchestration, the service survivability problem has been addressed in [29, 38–45] wherein the different service requests are pro-actively provisioned with stand-by instances as a way of enhancing their reliability. However, such approaches may result in resource under-utilization since the stand-by instances remain unused whenever there is no failure [6]. Aside from the fact that the approach adopted in this chapter is tailored to multi-domain service orchestration, it differs from the above and other existing works in that it directly incorporates the penalties resulting from service reliability violations into the service provisioning decision. This is achieved by intelligently making a trade-off between the cost resulting from the consumption of the network resources and the penalties resulting from the violation of the reliability requirements when selecting the InPs and the corresponding inter-domain links for provisioning the service, thanks to the intelligence inherent in the Deep Reinforcement Learning (DRL) approach that is adopted to make the placement decisions.

In light of the above, the key contributions of this chapter to the overall thesis is summarized as follows: first, a multi-domain service orchestration sequence diagram is proposed highlighting the service orchestration procedures and entities in a centralized service orchestration framework. The sequence diagram is aligned with the orchestration framework proposed in Chapter 2 and is compatible with the ETSI management and orchestration architecture proposed in [51], in which, each domain has an NFV orchestrator for intra-domain resource orchestration; then, aware of the stringent reliability requirements of mission-critical applications envisaged in future networks, and the fact that the scarce underlying resources are to be shared by a myriad of service requests, a resource efficient, DRL based service reliability-aware algorithm for multi-domain service orchestration is proposed. As

opposed to conventional heuristic approaches, DRL approaches are well suited for scenarios in which the service deployment decision jointly considers multiple network attributes including cost, delay and reliability, among others. This is due to the inherent ability of such approaches to effectively infer the influence of each attribute towards the service provisioning objective [24, 114, 115]. Moreover, with the need to preserve the privacy requirements of the different InPs, by adopting a DRL based approach, it is possible to exploit historical data based on previous service deployments to infer attributes that may not have been disclosed by the different InPs, thanks to the predictive intelligence of RL techniques.

The rest of this chapter is organized as follows: Section 3.3 introduces the proposed multi-domain orchestration sequence diagram. A description of the reliability-aware multi-domain service orchestration problem is introduced in Section 3.4. The proposed DRL based reliability-aware multi-domain service orchestration algorithm is presented in Section 3.5. The performance evaluation of the proposed multi-domain service deployment algorithm is presented in Section 3.6 and the chapter is concluded in Section 3.7.

3.2 Proposed architectural Framework

The multi-domain SFC orchestration framework considered in this thesis consists of three main players, as shown in Fig. 3.1: a tenant; a master orchestrator (MO); and a number of domain specific orchestrators (DSOs). The tenant is the initiator of a service request, with given specifications and constraints. This could be a service provider, a mobile virtual network operator or a vertical user, among others. The tenant specifies the request in terms of the topology, required resources (e.g.: computation, storage, memory, bandwidth) for the different VNFs and the corresponding interconnecting links, providing any other complementary constraints as well. The MO is the orchestrator corresponding to the domain from which the request is initiated. The proposed framework is based on the ETSI architecture option in [10] in which there is a single NFV Orchestrator (NFVO) per administrative domain, with the orchestrators of the different domains able to communication through the $Or - Or$ interface. In this way, the MO has access to the public information advertised by the DSOs. However, in order to adhere to the privacy requirements of the different domains, the information disclosed by each domain is considered to be limited to only the type of resources/functions that can be provisioned within that domain [37, 88]. The information regarding the amount, location, and topology of those resources is presumed to be private, hence, undisclosed. Consequently, with respect to a given InP k, the MO observes a tuple of global information denoted by $< Q^k, \Gamma_k, G^U >$ where Q^k is a set denoting the type of resources that can be provisioned within domain k, Γ_k denotes the span or the geographical bounds in which the Inp k operates, and G^U is the inter-domain connectivity graph comprised of the peering nodes and inter-domain links, including their respective attributes. The MO is responsible for, among other functions: $i)$ mapping the specifications of the request to the global information provided by the different administrative domains, with the goal of identifying the potential domains for hosting the SFC request. $ii)$ invoking the orchestration algorithms proposed by the thesis to split the request among the different feasible domains with the goal of optimizing the

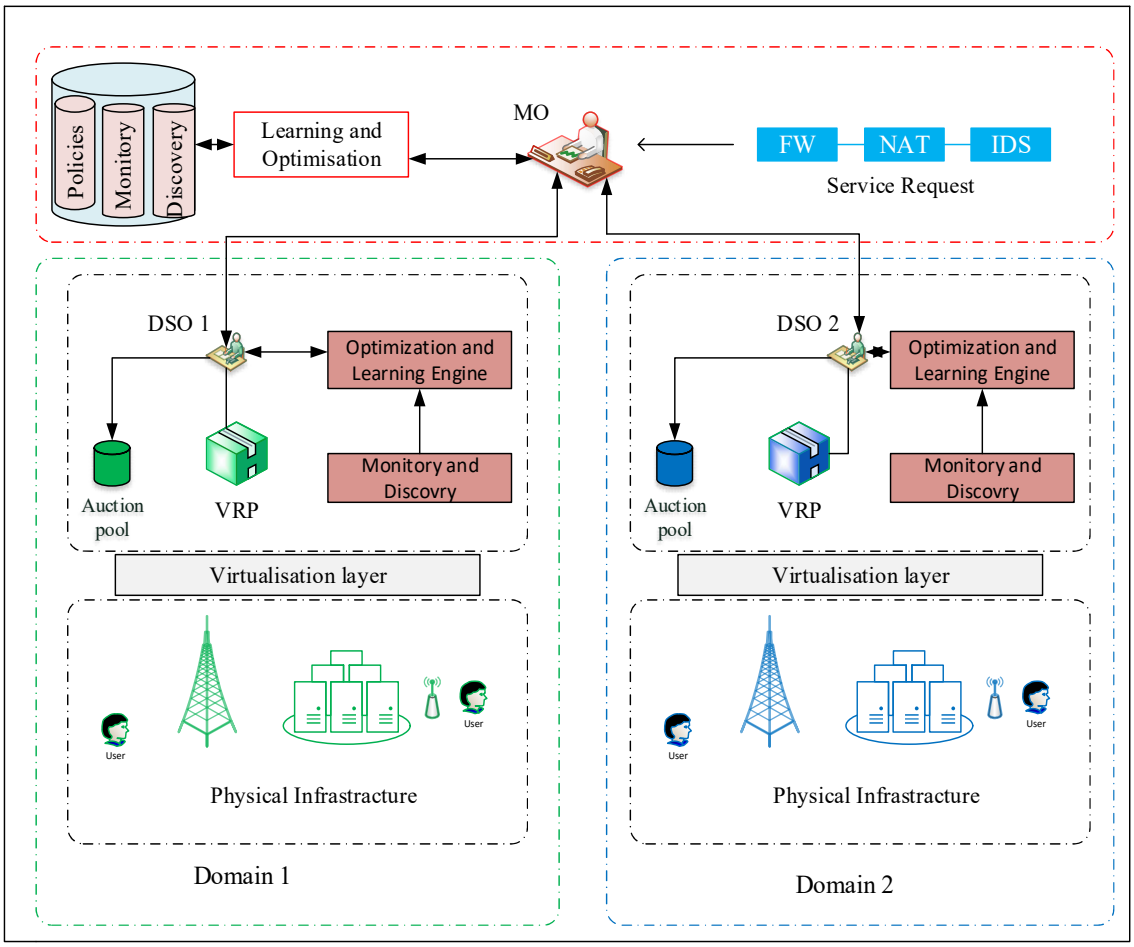


Figure 3.1: Architectural framework for the excution of the proposed algorithms

service orchestration objective specified in Eqn. 2.4; iii) guiding the DSOs regarding the specific peering nodes on which to terminate the resource reservations during their respective intra-domain mapping stages; iv) communicating the allocation results to the tenant, and the management of the life-cycle of the request in case of a successful provisioning. In case of a centralised approach, the MO can adopt the role of a Federation Manager as proposed in the frameworks of [59, 60].

In this thesis, the Domain Specific Orchestrators (DSOs) refers to all the orchestrators different from the one from which the request was initiated. Each DSO is envisaged to have a complete view of its internal domain, including; the internal network topology, amount and type of resources in the Virtualized Resource Pool (VRP), and QoS guarantees from the different resources through the different monitoring modules. The Optimization and learning Engine of the different domains is used for executing intra-domain orchestration of both intra-domain requests and the assigned sub-SFCs from external requests, and for updating policies regarding the interaction with other external entities. On the other hand, the Auction pool within each domain is composed of the resources that a given

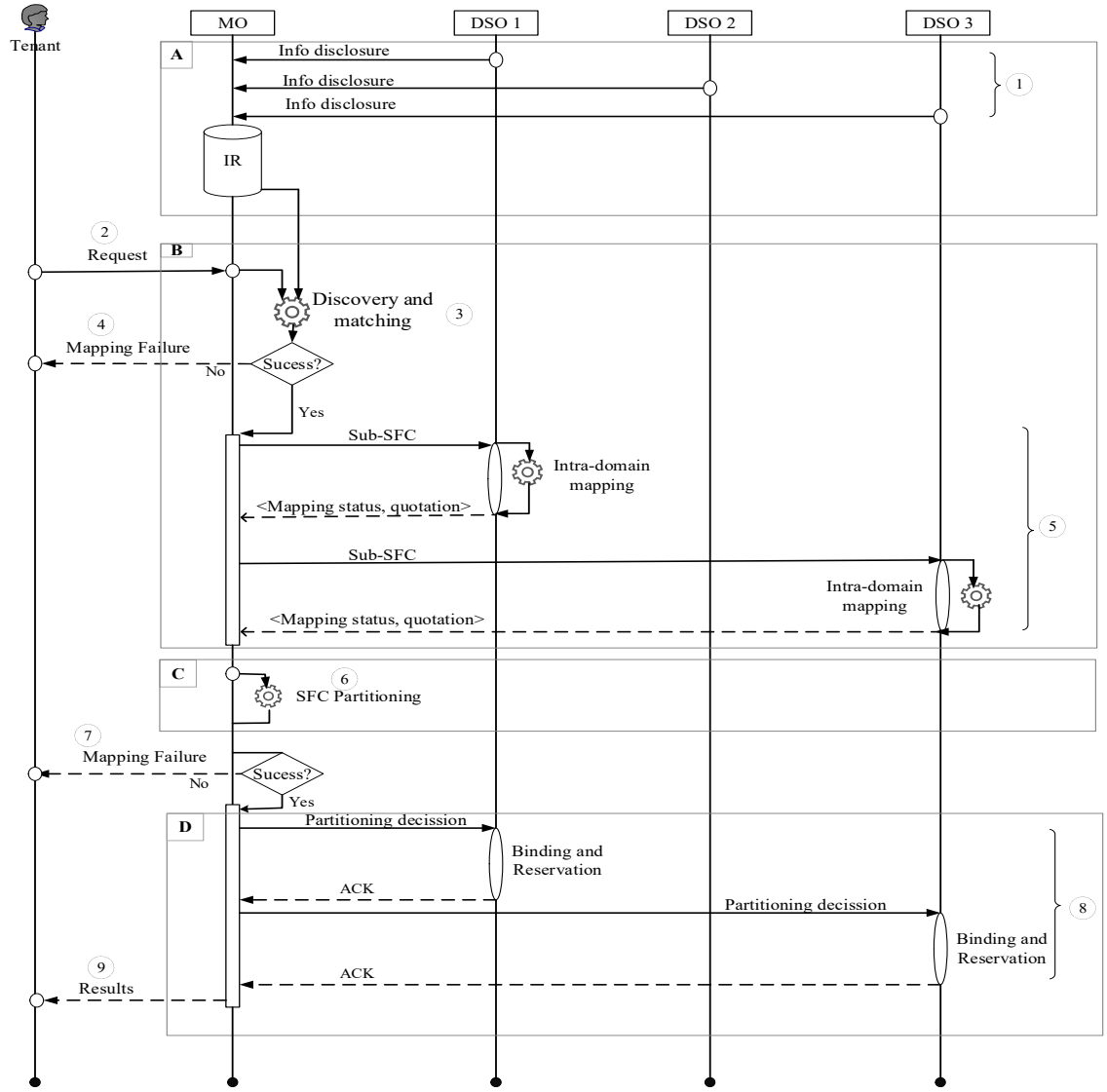


Figure 3.2: Sequence diagram of the multi-domain SFC orchestration framework

DSO is willing to allocate to service requests that are initiated from external domains.

3.3 Proposed multi-domain orchestration sequence diagram

In conformity with the orchestration framework proposed in Chapter 2 as shown in Fig. 3.1, the sequence diagram considers three main players engaged in the multi-domain service orchestration procedure, as shown in Fig. 3.2, thus; tenant; a master orchestrator (MO); and a number of domain specific orchestrators (DSOs). The tenant is the initiator of the service request towards the master Orchestrator. In the event that the MO is not capable of meeting the requirements of the request, then it engages the DSOs which are the orchestrators of the interconnected domains.

The proposed sequence diagram is shown in Fig. 3.2, with blocks A-D indicating the key phases of the framework. Block A corresponds to a continuous phase, in which the different DSOs periodically advertise their public information, that is stored in an information repository (IR) for each domain. In order to adhere to the privacy requirements of the different domains, the information disclosed by each domain is considered to be limited to only the type of resources/functions that can be provisioned within that domain [37,88]. However, information regarding the amount, location, and topology of those resources is presumed to be private, hence, undisclosed. Consequently, with respect to a given InP k, the MO observes a tuple of global information denoted by $< Q^k, \Gamma_k, G^U >$ where Q^k is a set denoting the type of resources that can be provisioned within domain k, Γ_k denotes the span or the geographical bounds in which the Inp k operates, and G^U is the inter-domain connectivity graph comprised of the peering nodes and inter-domain links, including their respective attributes.

Block B corresponds to the candidate extraction phase. The phase is executed upon the arrival of a request from the tenant (event 2). A request $r \in \mathbb{R}$ is specified as a tuple $< G_v^r, \tau_f^r, \phi^r >$ where $\mathbb{R}$ is a set of all requests. The parameters G_v^r is a graph specifying the topology of the request and capturing the attributes and constraints of the VNFs and inter-connecting links, including the amount and type of resources to be provisioned over these. The parameter τ^r captures the life-time of the request. The parameter ϕ^r captures any other tenant-specific constraints such as maximum available budget, domain preferences and QoS, among others. On receiving the request, the MO must identify potential domains that may serve the request, since in practice, such a request can only be served by a subset of all available domains due to the associated constraints. Moreover, even the potential domains may be able to serve only a portion of the request (i.e., sub-SFC). Therefore, the MO exploits the resource discovery and matching algorithm (event 3) to associate to each potential InP a sub-SFC of the request that it can serve. In the event that any segment/VNF of the request has no potential candidate, then the request is rejected with a mapping failure message sent to the tenant (event 4), otherwise, the MO sends to each DSO the associated sub-SFC (event 5). Then, each DSO performs the intra-domain evaluation of the assigned sub-SFC, and sends back the intra-domain mapping results to the MO, specifying computed values of each parameter specified by the MO, such as: total cost, average delay, bottleneck bandwidth and guaranteed reliability within this domain. To increase competitiveness and provisioning efficiency, we propose a flexible bidding scheme, in which each DSO can return a quotation of all the possible sub-strings of the assigned sub-SFC it can provision internally. For example, instead of a given DSO returning a single aggregated quotation for a sub-SFC as $< v2, v3, v4, X >$, where X denotes the cost and other attributes, such as guaranteed reliability or delay to the different peering nodes for mapping a sub-SFC with VNFs v2, v3, and v4; it instead returns the quotation for the sub-SFC components in the form: $\{< v2, X >, < v3, X >, < v4, X >, < v2, v3, X >, < v3, v4, X >, < v2, v3, v4, X >\}$. This is because, even if an InP can serve a sub-SFC (or even the entire SFC), the MO should have the flexibility to assign any number of the feasible VNFs to be served by that InP, as long as taking such an action is more beneficial to the MO. Moreover, since the DSO will not disclose details regarding the location where the VNFs are to be embedded, the information regarding the unit cost per resource is kept private to the DSO, since it returns just the total quotation, without revealing the amount of

resources to be incurred for mapping the request segment.

In the proposed framework, the MO is considered to have enough information in order to identify the potential candidate InPs for mapping each VNF, primarily due to location constraints. As a result, we benefit from that information to guide the different InPs about the most likely peering nodes to be used for the inter-domain connectivity (i.e.: the peering nodes connected to the candidate InPs of the preceding or subsequent VNFs). This way, the DSO always tries to map the assigned sub-SFC as close as possible to these peering nodes, not only to minimize the intra-domain resource consumption, but also to minimize the delay towards the exit or entry peering nodes. To understand the insight behind this guided mapping, let us consider a SFC consisting of 3 VNFs, as shown in Fig. 3.3. The request virtual link constraints are indicated as x/y where x and y denote the maximum delay and minimum bandwidth respectively. In the same way, the values on each substrate link indicate the delay and residual bandwidth associated with that link. As shown in the figure, for the mapping solution 1, InP A maps the sub-SFC composed of VNFs 1 and 2 close to peering node g, while for the mapping solution 2, these are mapped close to peering node c. Observe that mapping solution 1 results in a mapping failure due to the violation of the delay constraint on virtual link VNF 2 - VNF 3. As the internal topology of the domain is undisclosed, the DSO has no information regarding where the subsequent parts of the request will be embedded; consequently, the DSO may embed its allocated SFC segments in a location that is far from the peering nodes that connect to the candidate InP of the subsequent segment, resulting in rejection of a request that would otherwise be accepted. In this regard, we propose that the MO discloses the preferred peering nodes, as it sends the sub-SFC segments to the respective candidate InPs.

In the SFC partitioning phase shown in block C, the MO uses the results of the candidate extraction phase to identify the final domains to host each VNF of the request (event 6). If the partitioning phase is unsuccessful, then, the MO sends a mapping failure message to the tenant (event 7), otherwise, the MO instructs the selected InPs to reserve and bind the resources (event 8), including those connecting to the outgoing peering node selected by the partitioning algorithm. Observe that the partitioning algorithm jointly selects the intra-domain links to be used (by specifying the peering nodes to be used), the intra-domain nodes, and the inter-domain links. This coordinated approach guarantees a good mapping solution. Upon completing the binding step, the MO issues a request acceptance message to the tenant (event 9). This may be followed by the service deployment and the management of the life-cycle, including a periodic scaling of the allocated resources, depending on the real time utilization.

3.4 Description of the reliability-aware multi-domain orchestration problem

In this chapter, the reliability-aware service orchestration problem is formulated as an ILP with the objective of minimizing the service provisioning cost associated with both, the resource consumption and the penalties resulting from QoS violations due to service interruptions. The resource consump-

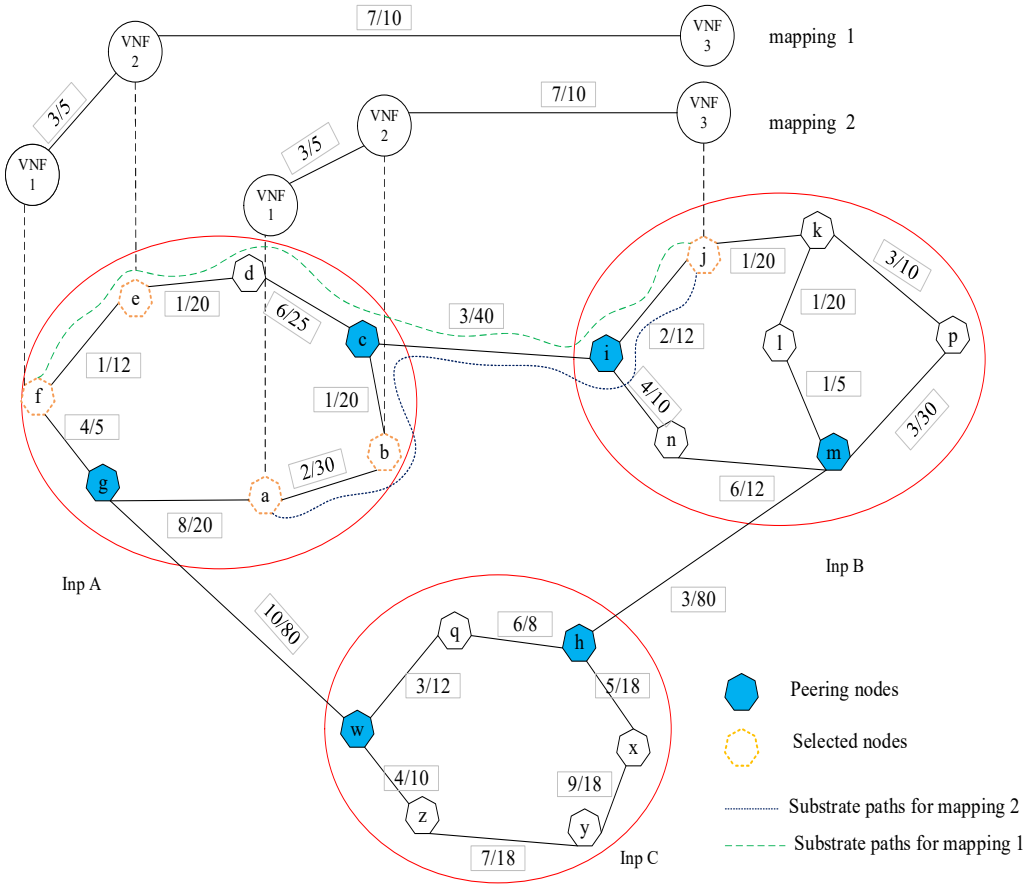


Figure 3.3: Illustration of guided mapping

tion cost is related to the cost of physical links and nodes resources used to provision the service request.

As highlighted in the service orchestration procedure described in Section 3.3, the orchestration process involves participation of both the master orchestrator and the domain specific orchestrators. Therefore, in the subsequent subsections, the provisioning objectives and constraints that need to be adhered to by both entities are introduced.

Master Orchestrator objective

By exploiting the information from the resource matching step, the master orchestrator aims to select a subset of domains and their corresponding peering nodes on which to map the request, with a goal of incurring the least mapping cost, given that the physical network and service request constraints are not violated. Mathematically, for each admitted request, the problem for the MO can be formulated

as a cost minimization problem:

$$Minimise: \quad \frac{1}{|\mathbb{R}|}\sum_{r\in\mathbb{R}} \mathbf{C}(\mathbf{G_v})^{\mathbf{r}} \tag{3.1}$$

where $\mathbf{C}(\mathbf{G_v^r})$ denotes the implementation cost of request $r \in \mathbb{R}$, and it incorporates the resource consumption cost C^r_{cons} due to the use of links and nodes resources and QoS violation cost, C^r_{qoS}. In order to compute the resource consumption cost C^r_{cons}, we let the binary variables: $\sigma^e_{l_{uv}} \in \{0,1\}$=1 if the request virtual link l_{uv} is provisioned by the intra-domain edge $e^k \in E^k_s$ of domain $k \in \mathbb{K}$, zero otherwise; $\sigma^{e_{int}}_{l_{uv}} \in \{0,1\}$ =1 if the request virtual link l_{uv} is provisioned by the inter-domain edge $e_{int} \in E_{int}$, zero otherwise; $y^{n^p_v}_{n^k_s} \in \{0,1\}$=1 if the request virtual node n^p_v is provisioned onto substrate node n^k_s, zero otherwise. In this way, the cost for provisioning request $r \in \mathbb{R}$ can be evaluated as :

$$C^r_{cons} = \sum_{l_{uv}\in L_v}\sum_{k\in\mathbb{K}}\sum_{e^k\in E^k_s} \sigma^{e^k}_{l_{uv}} \times dem^{l_{uv},r}_{bw} \times \varrho^{e^k} + \sum_{l_{uv}\in L_v}\sum_{e_{int}\in E_{int}} \sigma^{e_{int}}_{l_{uv}} \times dem^{l_{uv},r}_{bw} \times \varrho^{e_i nt} + \sum_{n^p_v\in N_v}\sum_{k\in\mathbb{K}}\sum_{q\in Q} y^{n^p_v}_{n^k_s} \times dem^{n^p_v}_q \times \varrho^{n^k_s}_q \tag{3.2}$$

where the first, second and third terms of Eqn. 3.2 relate to the utilization of intra-domain links, inter-domain links, and node resources respectively.

On the other hand, the QoS violation cost C^r_{qoS} is related to the penalties incurred by the MO due to QoS violations as a result of service interruption. In general, such a cost is evaluated as:

$$C^r_{qoS} = \beta_d . f_d(Rel^r) \tag{3.3}$$

where $f_d(Rel^r)$ denotes the mapping from the end-to-end reliability Rel^r, of a given SFC request $r \in \mathbb{R}$ to the Quality-of-Service degradation. In other-words, how a given value of service reliability translates to a given level of QoS violation. The parameter β_d denotes the penalty factor for each unit level of QoS violation. In this work, for simplicity, C^r_{qoS} for a given request is evaluated as:

$$C^r_{qoS} = \sum_{t\in T} \frac{x^t_r}{\tau^f} * rev^r \tag{3.4}$$

where $x^t_r \in \{0,1\}$ is a binary variable equal to 1 if a request $r \in R_A$ experiences a QoS violation at time t, zero otherwise, and rev^r denotes the revenue associated with the request throughout its lifetime which is computed as below:

$$rev^r = (\sum_{n^p_v\in N_v}\sum_{q\in Q} \aleph_q \times dem^{n^p_v}_q + \sum_{l_{uv}\in L_v} dem^{l_{uv},r}_{bw} \times \aleph_{bw}) \times \tau^r_f. \tag{3.5}$$

where $\aleph_q$ denotes the price the MO charges the tenant for each unit of type $q \in Q$ resource allocated

to the different VNFs of the request, and $\aleph_{bw}$ denotes the price charged for each unit of allocated bandwidth resource.

From equation 3.4, the MO forfeits revenue from the tenant for all the time instants for which there is a violation of QoS of the admitted request. Therefore, in case the service experiences QoS violation throughout its life-time, the MO receives zero revenue from the tenant but pays the different InPs for the resources used to map the request, corresponding to the worst case net revenue. Therefore, in order to maximize the received net revenue/profit, the MO needs to minimize both the resource consumption cost and QoS violation cost.

Domain Specific Orchestrator objective

Once the DSO has received the sub-SFC (or the entire SFC) from the MO, it exploits the intra-domain topology information at its disposal to perform the intra-domain mapping evaluation with the goal of minimizing the intra-domain mapping cost. Mathematically, this can be formulated as:

$$Minimise \ \ \mathbf{c(gv)} \tag{3.6}$$

where $gv \in G_v$ is a sub-SFC of the request and **c(gv)** is the intra-domain cost for mapping the sub-SFC. If we denote $x^m_{i,n} \in \{0,1\}$ as a binary variable equal to 1 if VNF i is mapped on physical node n inside domain m, and denote by $p^{m,n}_c$ and $p^{m,e}_b$ as the cost per unit of node and link resources on physical node n and physical edge e respectively, then, $c(gv)$ in equation 3.6 can be expressed as:

$$c(gv) = \sum_{i \in N_v} dem^i_c p^{n,m}_c x^m_{i,n} + \sum_{ij \in E_v} \sum_{e \in E^m_s} dem^{ij}_{bw} p^{m,e}_b f^m_{ij,e} \tag{3.7}$$

where $f^m_{ij,e} \in \{0,1\}$ is a binary variable equal to 1 if virtual link ij is mapped onto substrate edge $e \in E^m_s$, where E^m_s is the set of all edges in domain m.

The optimization of Eqns. 3.1 and 3.6 is performed while adhering to the resource, location, delay, integrity and domain constraints specified introduced in Chapter 2 in Eqns. 2.6-2.13. The above formulation is a typical ILP problem which is NP-hard in nature. Therefore, solving such a problem following an exact solution approach as in [7, 11–13, 62] is not feasible for delay sensitive applications due to the high run time of such approaches. This motivates the adoption of heuristic/meta-heuristic approaches such as that proposed in this chapter, with the ability to realize near-optimal solutions in practical run time. The proposed RL based algorithm for solving the above formulation is described in Section 3.5 below.

3.5 Proposed RL-Based Orchestration algorithm

Given that each VNF of a service request may be potentially hosted by more than one domain, obtaining an optimal domain set for hosting the request is computationally intractable due to the large

number of possible mapping combinations even for a small-scale network. Therefore, this chapter proposes an algorithm that is able to obtain near-optimal deployment solutions in feasible time.

The proposed algorithm consists of three main tasks: Candidate extraction; Request splitting; and Binding. The candidate extraction task involves identifying the segment of the request that each InP can potentially host, including the associated cost and terms for hosting that segment. From this step, it is possible that more than one InP bids for the same request segment/VNF. Therefore, the request splitting/partitioning step is used to decide on the winner for the contested segment with the goal of optimizing the mapping objective introduced in $Eqn.$ 3.1. In this work, this step is executed by an intelligent agent implemented in form of a policy neural network. Based on the results of the splitting step, the binding task reserves both node and link resources for serving the SFC from the ingress to egress node. The execution steps of the proposed algorithm are shown in the flowchart in Fig. 3.4 wherein, on receiving a service request, the MO performs resource discovery and matching to identify the possible InPs for hosting the different VNFs of the request. In-case any VNF has no potential candidate, then the request is rejected, otherwise, each possible candidate is allocated a sub-SFC of the request it can potentially host, based on the disclosed public information. Next, each candidate domain tries to internally bid for the assigned sub-SFC considering its internal policies and available resources and sends the results of this operation to the MO. Since it is possible that multiple InPs bid for the same VNF, the MO invokes the RL-based algorithm to partition the request among the contending InPs. A detailed description of the algorithm tasks is given below:

3.5.1 Candidate Extraction Procedure

This task corresponds to block B of the sequence diagram introduced in Fig. 3.2, and it is aimed at extracting feasible InPs for provisioning the request. This consists of the following steps:

Resource discovery and matching

The different VNFs and the interconnecting links constituting the SFC request are constrained in terms of type and amount of required resources (processing, memory, storage, bandwidth, etc.), location and delay, among others, implying that these can be served by only a subset of available InPs. Based on the global information available at its information repository, the MO invokes the resource discovery and matching step to identify those InPs that can potentially host the request, partially or as a whole. This is done by matching the VNFs location and resource type constraints to the disclosed information of each InP, and matching the virtual links constraints to the inter-domain links' attributes. The idea behind this matching is that a VNF can only be served by an InP whose disclosed resource type set includes the required resource type of that VNF, and whose coverage satisfies the VNF location constraints. Similarly, for two successive VNFs u and v to be hosted by InP A and InP B, where $A \neg B$, there should exist an inter-domain path between InP A and InP B that satisfies the constraints on virtual link l_{uv}. By eliminating all unfeasible InPs, the candidates set for the subsequent intra-domain enumeration is significantly reduced, resulting in a substantial decrease of the execution time of the algorithm. Specifically, the discovery and resource matching

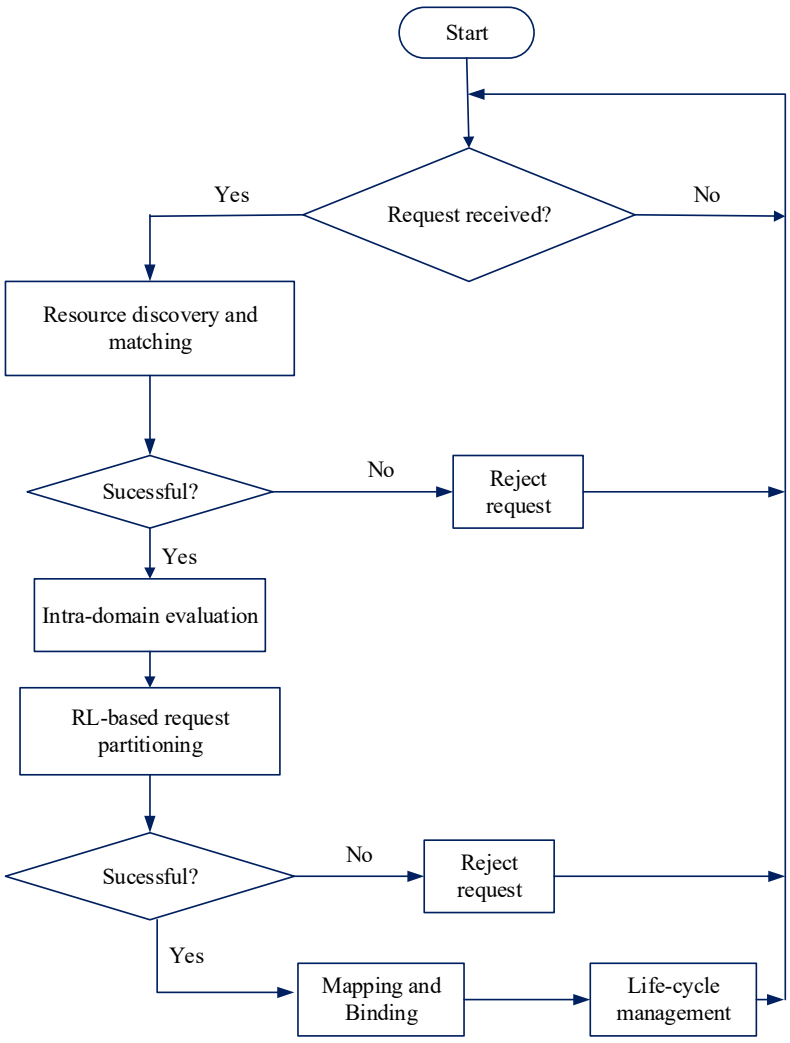


Figure 3.4: Flowchart of the multi-domain service embedding algorithm

algorithm associates each VNF u of SFC k with a candidate set $cand_u^k$ comprised of all InPs that can potentially serve this VNF. We incorporate location constraints to address cases, for example, in which the service provider may want to deploy cloud services to a set of end-users in which there could be many potential locations for a given cloud service.

Intra-domain evaluation

The resource discovery and matching step associates each InP with a sub-SFC (or SFC) that it can potentially host, based on the available global information. These sub-SFCs are then forwarded to the respective DSOs, to evaluate if and where these can be mapped, based on the intra-domain topology and policies. During this stage, each DSO will run its intra-domain mapping algorithm to evaluate the possible mapping that minimizes the mapping cost and guarantees the minimum bandwidth to the disclosed peering nodes. Once the intra-domain evaluation is completed, the DSO forwards

a message to the MO, indicating the portion of the assigned sub-SFC it is able to provision, the quotation (in terms of monetary cost), reliability guarantees and delay to each of the guiding peering nodes. Note that the location where the respective VNFs can be mapped, and the number of links to the peering nodes, is not disclosed. This attribute safeguards both, the topology and the pricing privacy of the InP, since the cost is returned as a single value for each sub-SFC sub-component.

Note that the intra-domain evaluation step is performed by the DSO of each InP independently and following its own policies. Therefore, any of the single domain orchestration algorithms proposed in Part III of the thesis can be used for this purpose. In this work, we adopted the single domain provisioning algorithm proposed in Chapter 5.

3.5.2 Markov Decision Process Model for the DRL problem

The DRL agent is used to decide on the final InPs for provisioning the service request among all the possible candidates obtained from the resource matching step, in such a way that minimizes the operational cost incurred by a SP. Obtaining an optimal solution for such a task involves enumerating multiple combinations of possible mapping assignments, making it computationally intractable to solve. Moreover, the SFC provisioning objective may be jointly influenced by multiple attributes, such as resource consumption and QoS as in our case, which makes the state of the art heuristics not well suited for this problem. The request partitioning task in this work is delegated to a reinforcement learning agent implemented in the form of a policy neural network. In the subsections that follow, a description of the proposed DRL approach for request splitting is given including its formulation as a MDP, the policy neural network architecture and the training of the policy neural network.

MDP model

The RL algorithm adopts a Markovian Decision Process (MDP) where A is the set of discrete actions, S is the set of discrete states, P is the transition probability distribution, R is the return function and $\gamma \in [0, 1]$ is the discount factor of future rewards. Considering a working scenario in which the state of a given system is fully observable by an RL agent, we model the system as a Markovian Decision Process (MDP) defined by the tuple (S, A, P, R), where: S denotes the set of possible states of the system; A denotes the set of possible discrete actions to be taken, actions for the selection of an InP to provision a given VNF of the request; $P = P(s_{t+1}|s_t, a_t)$ denotes the transition probabilities from state s_t to state s_{t+1} after taking action a_t. In this work, we adopt a model-free method, hence, we are not interested in learning the transition probability; and $R = R(s_t, s_{t+1}, a_t)$ denotes the reward obtained after taking action a_t from state s_t and transiting to state s_{t+1}. Considering an entire episode as a sequence of visited states due to the corresponding sequence of actions, the return of an episode is defined as the discounted sum of all the rewards received by the agent during that episode. It is expressed as:

$$Return = \sum_{i=t}^{T} \gamma^{i-t} R(s_i, s_{i+1}, a_i) \tag{3.8}$$

where $R(s_i, s_{i+1}, a_i)$ is the reward received by the agent after taking action a_i in state s_i at step i. In this problem domain, the goal of the RL agent is to learn a policy $\pi : S \rightarrow A$ which maximizes the expected return, $\mathbb{E}[Return]$, over all episodes.
The state space, action space and reward of the adopted DRL-based framework is defined as below:

State space: The state captures an abstraction of the environment, and it is the basis upon which the agent decides which action to take. Therefore, taking a similar action in the same state should result in a similar reward and should transition the environment to a close next state. In this work, the state captures the features of the substrate network in relation to the requirements of the request to be partitioned. The policy neural network architecture adopted in this chapter uses a convolutional neural network, which is well suited for processing information represented in form of an image [24]. Therefore, in order to conform to this requirement, the environment state is formulated as an image-like input using an $K \times F_k$ feature matrix, where K is the number of InPs and F_k is the number of relevant features associated with each InP $k \in \mathbb{K}$, whenever the policy neural network needs to choose an action. The features constituting such a state matrix are discussed in Section 3.5.3. A key challenge with neural-network based architectures is that they are trained with a fixed state dimension, which makes it impractical to use the same neural-network for a different number of InPs from those used at training stage. But in fact, such a policy network should be able to work with any number of InPs. To overcome this challenge, we introduce the concept of dummy InPs with dummy feature vectors, which permits the trained policy network to be reusable even for scenarios where the number of InPs is inferior to the one used for training; therefore, avoiding the need to retrain the neural-network. In order to achieve this, the policy neural-network was trained using the maximum possible number of InPs. Then, for the testing phase, when the number of InPs is less than the one used for training, we match the input matrix by appending dummy InPs with dummy feature vectors to reach the expected state size. The dummy features are obtained by providing the worst values of each feature, in order to make such dummy InPs less likely to be selected by the policy network. Moreover, in the worst event that a dummy InP is assigned a high probability of being selected to host a given VNF, the filtering layer that we adopt at the output end of the architecture will be able to sieve out such an InP.

Action space: For each request received by the MO, the number of decision epochs is equal to the number of VNFs in the request. Therefore, for each VNF of the request, the policy neural-network has to decide on which InP to assign the VNF to. Therefore, the action space for each decision epoch is equal to the number of InPs in the system, including dummy InPs in the event that the number of available InPs is less than the training size. Since we have a discrete number of InPs, the action space is discrete as well, hence, well suited for the proposed RL agent.

Reward: The reward signal is aligned with the SFC deployment objective, which in this work is to maximize the long term revenue by minimizing both: the SFC mapping cost C^r_{cons}; and the QoS degradation cost C^r_{QoS}. The SFC mapping cost captures the cost of resources used for embedding the SFC across the different InPs, as shown in Eqn. 3.2. The QoS degradation cost captures the penalties resulting from the negative effects of QoS degradation due to service interruption as expressed in

Eqn. 3.4. In order to encourage the agent to take actions that result in low mapping cost and high service availability, we formulate the reward signal obtained after embedding a given request as the revenue-to-cost ratio:

$$Reward = \frac{revenue}{Cost} \tag{3.9}$$

where $Cost$ is evaluated as the sum of the resource cost C^r_{cons} and the QoS degradation penalty cost C^r_{QoS}. The revenue considers the sum of the revnue resulting from the assigned node and link resources as expressed in Eqn. 3.5 [116].

3.5.3 Feature extraction procedure

For the policy network to maximize the reward signal as introduced in $Eqn.$ 3.9, there is need to jointly minimize both the cost of resource consumption and the penalties resulting from QoS violations, while accepting requests associated with high revenue. However, it is possible that provisioning paths that result in the least resource consumption and cost are associated with low service reliability values, which decreases the total reward received by the agent. Moreover, greedily minimizing the total cost in the short term may impact negatively on the long term received reward. This therefore requires the RL agent to intelligent trade-off the two cost components in a way that maximizes the long term reward, a requirement that cannot be met using classical heuristic approaches. The reward signal as indicated in Eqn. 3.9 is directly affected by the resource consumption cost, service reliability, and the received revenue which is directly reflected in the ability to admit requests with high resource requirements. Therefore, the features that form the environment state for the policy neural network are selected in a manner that reflects those attributes. Moreover, these features take into consideration the current state of both the substrate network and the pending service request.

For each request $r \in \mathbb{R}$ to be mapped, the algorithm uses the policy neural network to make a decision for each VNF of the request, one at a time, regarding the InP onto which this will be placed, starting with the VNF closest to the ingress node. Therefore, the number of decision epochs (hence the actions taken) for each request is equal to the number of VNFs of the request. For each VNF $n^p_v \in N_v$ from request $r \in \mathbb{R}$ to be scheduled for placement, the following are the features associated with each InP node $k \in \mathbb{K}$ of the substrate network to form the state matrix:

- Average cost per VNF, $Cost^k_{vnf}$: For a given InP k, when mapping the i^{th} VNF, $Cost^k_{vnf}$ denotes the bidding price for InP k for mapping each VNF on average. This is computed as the cost that the InP quotes for mapping the largest sub-SFC divided by the number of VNFs within that sub-SFC. The idea behind choosing the largest Sub-SFC is to avoid greedily selecting an InP based on only the cost of mapping the current VNF. If InP k is not a candidate for the current VNF under consideration, such an InP is assigned an infinite value of $Cost^k_{vnf}$, hence, discouraging the policy neural network from selecting that InP as a host for the current VNF. Note that the cost for the sub-SFC is part of the quotation returned to the MO after the intra-domain evaluation stage as discussed in the orchestration framework introduced in Section 3.3. The feature $Cost^k_{vnf}$ directly affects the cost component of the reward value as expressed in $Eqn.$ 3.9. Therefore, this parameter guides the agent in choosing InPs whose

$Cost^k_{vnf}$ values result in maximization of the long term reward.

- Average cost of the path to the ingress and egress node, $Cost^k_{path}$: For a given InP $k \in \mathbb{K}$, when mapping the i^{th} VNF, this feature is computed as the cost of the inter-domain path from this InP to the ingress and egress node and computed as:

$$Cost^k_{path} = \frac{Dist(k, Ing) + Dist(k, egr)}{2} \tag{3.10}$$

  where $Dist(k, Ing)$ and $Dist(k, egr)$ denote the cost of the inter-domain path between InP k and the ingress and egress node respectively. This feature serves two purposes: first, InPs that are not reachable from the egress or ingress nodes are unfeasible for mapping a given VNF, hence, associated with infinite $Cost^k_{path}$ values. Therefore, the probability of the policy neural network to select such InPs drastically decreases; secondly, this feature biases the policy network towards selecting InPs with low $Cost^k_{path}$ values, hence, translating in low mapping cost in terms of inter-domain paths from ingress to egress node, whenever such a decision results in a higher long term reward.

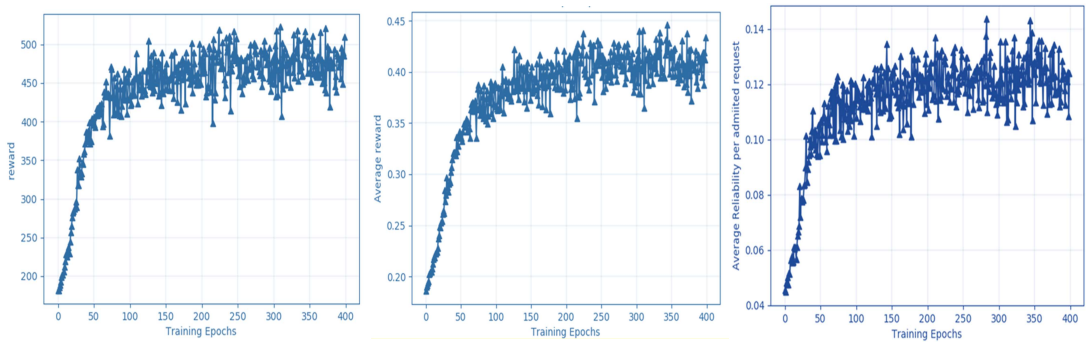
- Average reliability of InP k, $Reliab^k_{vnf}$. For a given InP k, when mapping the i^{th} VNF of the request, $Reliab^k_{vnf}$ relates to the reliability guarantee disclosed by the InP in mapping the sub-SFC containing this VNF. This value is disclosed along with the price quotation for the sub-SFC. In the event that it is not disclosed by the InP, this can be easily inferred from the previous failures and preemptions experienced from the previous mappings, to give the likelihood of the current VNF experiencing failure if hosted by InP k, thanks to the predictive capabilities of ML approaches. The goal of the policy neural network is to select InPs with high $Reliab^k_{vnf}$ values for each VNF whenever possible with the target that eventually, all the VNFs of the SFC are placed on domains that can guarantee service survivability. This reduces the QoS violation cost, resulting in a high reward value, according to Eqn. 3.9.

- Success probability Suc^k_{vnf}. For a given InP, when mapping the i^{th} VNF of a request, Suc^k_{vnf} captures the fraction of the $i-1$ previously mapped VNFs of the same request that have been mapped onto this InP. The purpose of this parameter is to encourage mapping as many VNFs as possible to the same InP as long as such an InP has low mapping cost and high reliability values. This results in a reduced number of inter-domain connections, hence the less the embedding costs, since in practice, higher costs are linked to inter-domain links. On the contrary, however, this feature may also be used to encourage the mapping of VNFs along different InPs for objectives related to load balancing or fault resilience, among others.

Once the above values have been computed for each InP, then each InP $k \in \mathbb{K}$ is associated with a feature vector f^k_i as:

$$f^k_i = (Cost^k_{vnf},\ Cost^k_{path},\ Reliab^k_{vnf},\ Suc^k_{vnf})^T \tag{3.11}$$

Then, the feature vectors of the different InPs are concatenated into a feature matrix, M^f_i which



(a) Total reward per epoch **(b)** Average reward per SFC request **(c)** Average end-to-end reliability per SFC

Figure 3.5: Training Performance results

serves as the state input for the policy neural network when making a placement decision for the i^{th} VNF of the request: Each row of M_i^f corresponds to the feature vector of a given InP, including the dummy InPs where applicable, and is expressed as

$$M_i^f = \begin{bmatrix} Cost_{vnf}^1 & Cost_{path}^1 & Reliab_{vnf}^1 & Suc_{vnf}^1 \\ Cost_{vnf}^2 & Cost_{path}^2 & Reliab_{vnf}^2 & Suc_{vnf}^2 \\ \dots & \dots & \dots & \dots \\ Cost_{vnf}^K & Cost_{path}^K & Reliab_{vnf}^K & Suc_{vnf}^K \end{bmatrix}$$

where K denotes the total number of possible InPs for provisioning VNF i.

3.5.4 Policy neural network training procedure

In most service and resource management problems, attributes such as, traffic load characterizing the environment, are usually repetitive, with certain predictable temporal correlations. This enable the agent to learn online, as the system executes, or offline by exploiting historical information. In this thesis, the latter option was adopted. The policy network was trained using a set of 1000 demands generated offline for each training epoch, for a total of 400 epochs. The training phase considered 12 InPs as the maximum possible number of InPs in the system. For each training epoch and for each request in the demand set, the candidate evaluation step is used to generate the candidate InPs for mapping the request. Then, for one VNF of the request at a time, the corresponding feature matrix is generated which is then fed into the policy neural network.

The policy network then associates a probability to each InP for embedding this VNF, inline with the mapping objective. Note, however, that since the neural network parameters are initially randomly assigned, the InP with the highest probability may not necessarily be the best InP for embedding this VNF, since the neural network accuracy is low at this stage especially at the initial epochs. This necessitates to perform a trade off between exploration and exploitation during the training phase.

For each selection made for a given VNF by the policy network, we use the cross-entropy as the loss function for running the back-propagation algorithm to obtain the gradients of the neural network parameters; and stack the resulting gradients. If the entire request is embedded successfully, we compute the resulting reward signal, otherwise the stacked gradients are deleted. The final gradient of the entire request is then computed as:

$$g := \alpha.r.g_s \tag{3.12}$$

where α is the learning rate parameter, which directly influences the learning speed and training convergence, r is the reward and g_s are the stacked gradients. The resulting gradients from the different requests are stacked into a buffer. When the number of virtual requests reaches the batch size, all the gradients previously stacked are jointly applied to the model and the stack buffer is emptied. Note that whereas it is possible to perform a gradient update for each successful request individually, adopting batch processing guarantees a faster and more stable training process.
The performance of the neural network during the training phase is shown in Fig. 3.5 for training duration of 400 epochs.

3.5.5 Architecture of the Neural Network for Policy Evaluation

The SFC request partitioning is performed by the MO which runs a policy neural network with the ability to learn new policies based on its environment and past decisions. The policy neural network takes as input a feature matrix which is extracted from: i) the results of the resource discovery step (e.g.: the average cost for mapping each VNF within each InP); ii) the global information available in the information repository (e.g.: connectivity to the ingress and egress node); iii) the results of the previous actions taken by this agent (e.g., number of successful VNF mappings inside this InP). In order to make the input compatible with the convolutional network adopted in our architecture, the extracted features are converted into an image-like input, in the form of an $K \times F_k$ feature matrix, where K is the number of InPs and F_k is the number of features extracted from each InP, as shown in Fig. 3.6. The final output of the policy is a probability distribution indicating the feasibility of each InP to host the corresponding VNF, according to the desired reward function.

As shown in Figure 3.6 the policy neural network adopted consists of 4 layers, which are: input layer, convolutional layer, softmax layer and filtering layer. The core part of the architecture is the convolutional layer, which takes as input the feature matrix from the input layer and performs a convolution between this matrix and the learnable weight values of the filter. This operation can be seen as a dot product of each InP feature vector and the filter weights. The output of the convolutional layer is a K-dimensional vector where K is the number of considered InPs. Thus, the convolutional layer associates a single score value to each InP depending on the values of the extracted features. The score v_k corresponding to the k^{th} InP is directly related to the suitability of that InP to host the VNF under consideration. The motivation for using the convolutional layer is its ability to easily learn the influence of each input feature towards the desired objective. Moreover, this is achieved with minimal memory overhead compared to conventional neural network architectures.

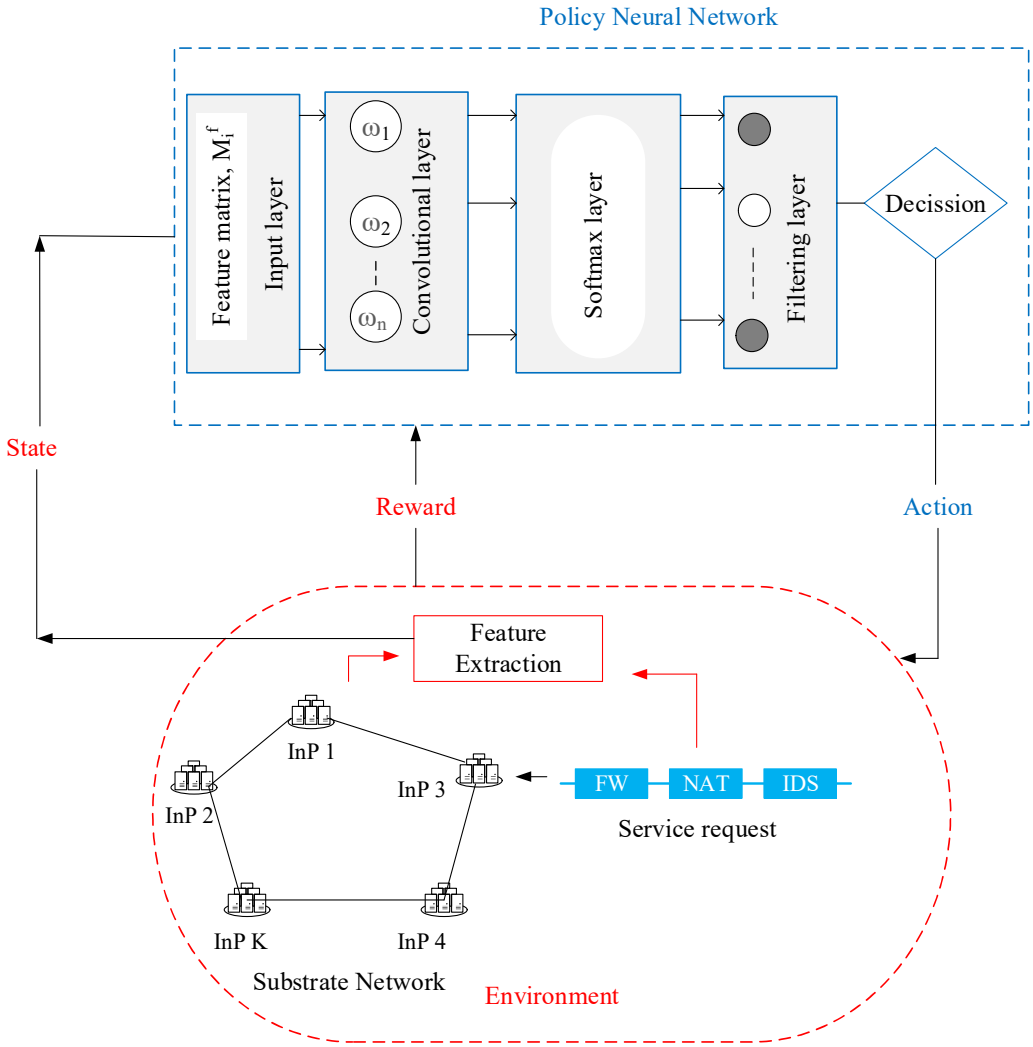


Figure 3.6: DRL Policy Neural Network Architecture

The output from the convolutional layer is fed as input to the softmax layer, which transforms the K-dimensional vector of InP score values into a K-dimensional vector of probability distributions. The probability attached to each value v_k relates to the probability of the corresponding InP being selected for provisioning the VNF under consideration, and this is evaluated as:

$$P(k) = \frac{e^{v_k}}{\sum_{j \in \mathbb{K}} e^{v_j}} \tag{3.13}$$

The k^{th} value in the vector of probabilities indicates the degree to which the k^{th} InP fits to host the current virtual node under consideration.

The filtering layer is adopted at the output end of the architecture to prune those InPs that are not able to meet the request constraints. Once such InPs are filtered out, then the final candidate InP

for the virtual node under consideration is chosen as the one with the highest probability value. In case of a tie between InPs for the highest probability, the candidate InP is chosen randomly from the contending InPs. Once such unfeasible nodes are pruned, then, the substrate node with the highest probability is selected for hosting the corresponding VNF of the request. The processing of a request can be summarized as follows:

Upon arrival of a request to the admission control block asking for service, each virtual node of the request is processed at a time with the policy neural network deciding the InP on which such a node is provisioned. This way, for each arriving request, all its virtual nodes are processed sequentially, one after another, starting with the virtual node next to the ingress node. This ordering ensures that the VNF order in the service chain is preserved, and also enables an early detection of infeasible virtual link mappings on executing the algorithm. The feature extraction block takes as inputs the current state of the substrate network and the requirements of the request. With all that information, the system state is composed in the form of a $K \times F_k$ feature matrix. Then, the feature matrix is used as input to the neural network, which produces an action in the form of an InP identifier for hosting the virtual node under consideration. After mapping all virtual nodes, and based on the obtained mapping solution, the resultant reward is calculated using Eqn. 3.9.

3.6 Performance Evaluation

This section describes the performance evaluation of the proposed algorithm, including the benchmark algorithms, simulation scenario, simulation environment, and discussion of obtained results.

3.6.1 Benchmark algorithms and Simulation Scenarios

The performance of the proposed multi-domain service orchestration algorithm is evaluated considering both offline and online scenarios. For both simulation scenarios, the proposed algorithm is compared with the following algorithms:

Distributed Network Service Embedding (DistNSE) algorithm

This is an algorithm for distributed multi-domain service orchestration proposed in [19]. The choice of this work for comparison is justified for a number of reasons: First, just like our work, DistNSE considers limited information disclosure and was found to be optimized in terms of mapping cost performance. Secondly, DistNSE can be easily customized to perform service deployment with the objective of enhancing service reliability. Moreover, for the fairness sake, we considered the best performance scenario of the algorithm in which all feasible paths from the ingress to the egress node are evaluated.

Reliab-DistNSE

This algorithm is implemented as the above DistNSE algorithm , but with a difference that for $Reliab - DistNSE$, the final path for embedding the service request is selected as the one which

results in the highest service reliability from the ingress to egress nodes.

The enhanced DistNSE (E-DistNSE)

This algorithm enhances the DistNSE algorithm by incorporating our proposed enhancements. To

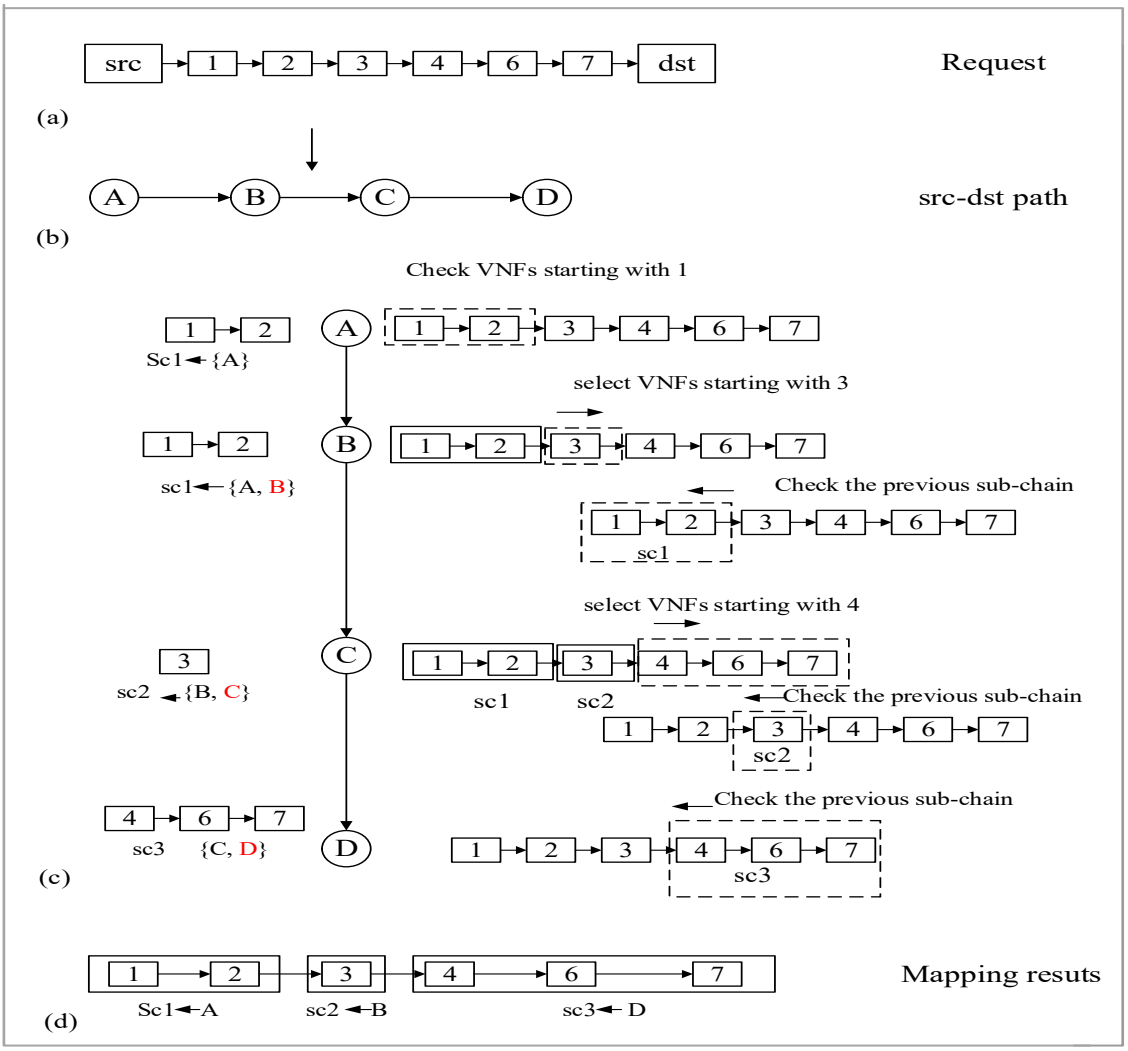


Figure 3.7: An illustration of DistNSE service provisioning

give an insight into the enhancements made to the DistNSE algorithm, first, a brief description of the operation of the DistNSE algorithm is given. Consider Fig. 3.7 in which figures 3.7(a), and 3.7(c) respectively show a SFC request with 7 VNFs and an illustration of the provisioning steps of the DistNSE algorithm to provision such a request. By exploiting the exposed global information such as residual bandwidth on the inter-domain links and the exposed boarder nodes of the different InPs, the algorithm starts by obtaining all feasible paths from the ingress node to the egress node. An example of such a path is shown in Fig. 3.7(b) consisting of 4 InPs . Then, for each of these paths, the algorithm execution starts by each first InP along the path (InP A in this case) receiving the SFC request to be provisioned. Then, based on its internal policies, this InP selects a sub-SFC

of the request to bid for (i.e., sc1 with VNFs 1,2). Next, InP A forwards the selected sub-SFC and the original SFC request to the next InP along the path (InP B), which in turn, selects a sub-SFC (sc2) from the the VNFs that have not been selected. This InP also tries to compete for the sub-SFC selected by the preceding InP i.e., sc1. Next, InP B forwards the selected sub-SFC and the original chain to the proceeding InP on the path where the same steps are performed. This procedure is repeated till the last InP has been reached. An example of the mapping solution from the path in Fig.3.7(b) is shown in Fig.3.7(d) in which the different VNFs of the request are mapped by InPs A, B and D. Next, the mapping results for the different paths are collected, and the path with the least mapping cost is selected as the host for the SFC request.

From the execution of this algorithm, the authors allow the InPs to compete only for the last selected sub-SFC in order to avoid violations in the service chain order. As an example, if InP C bids for sc1 and sc2 and wins sc1, whereas InP B bids for and wins sc2, the service chain order will be violated. The E-DistNSE incorporates two enhancements to the DistNSE algorithm as follows:

Firstly, E-DistNSE permits each InP irrespective of its position along a given path to compete for and be assigned any number of previously selected sub-SFCs as long as those sub-SFCs are consecutive and include the most recently selected sub-SFC along the path. As an example, considering Fig. 3.7(c), InP D can compete for sub-SFC set {sc1, sc2, sc3} or {sc2, sc3} or {sc3}. If InP D wins the first set, then the entire SFC request is mapped inside this InP. However, InP D cannot compete for {sc1, sc3} or {sc1, sc2} since the sub-SFCs in the first set are not consecutive and the second set does not include sc3, the most recent selected sub-SFC, hence, allocating such sub-SFCs to InP D would lead to violation of the order of the SFC. Note that this enhancement respects both the order of the SFC and enhances the performance of the algorithm by not restricting the number of VNFs that an InP can compete for. This makes it possible for even the last InP along the path to host the first VNFs in the request. Secondly, as an enhancement towards the execution time of the algorithm, starting with the first path from source to destination, the minimum value of the cost observed so far is stored. Then, for the subsequent paths, whenever the mapping cost seen so far along this path is equal to or exceeds the stored minimum cost value, the computation along this path is aborted, and the algorithm execution goes to the next path. This enhancement realizes time saving from two aspects: First, the time that would be spent on enumerating the remaining InPs along the path; Secondly, the time that would be spent on analyzing the results from all the returned paths by the central orchestrator in order to select the best path. Such an analysis would involve for-example sorting the returned paths according to cost, which tends to be time consuming as the number of returned paths grows.

3.6.2 Simulation environment and settings

The performance evaluation of the proposed algorithms is based on substrate networks in which the number of InPs is varied from 4 to 12. The values of the diferent parameters of the substrate network and the service request are shown in Table. 3.1.

Table 3.1: Simulation Parameters

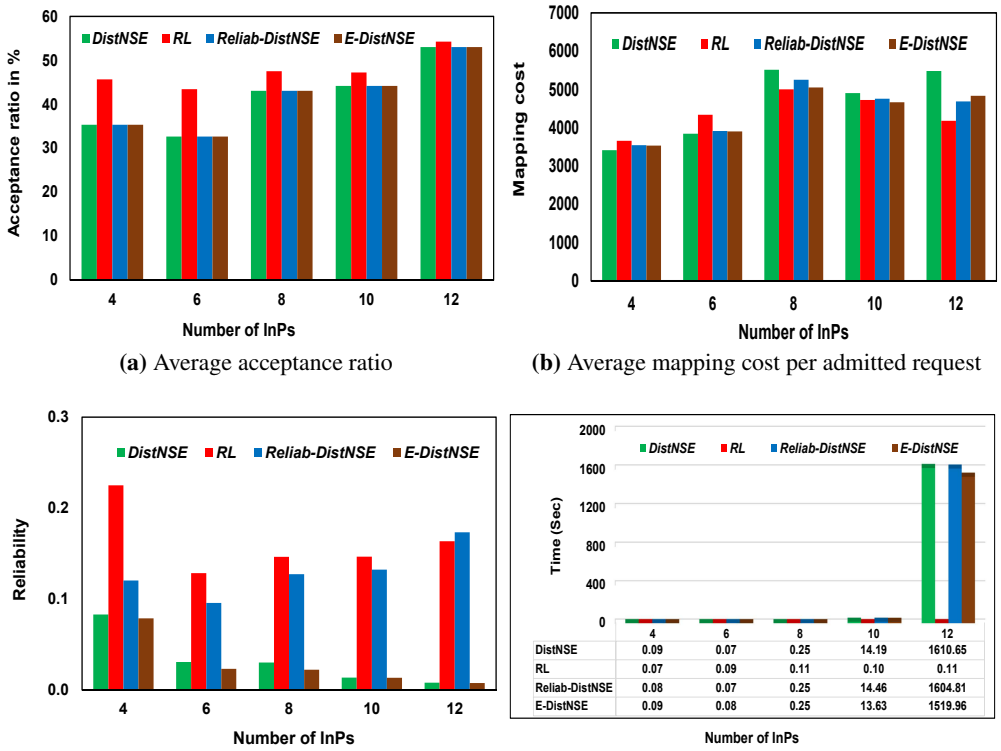
Substrate Network:	
parameter	Value
Number of InPs	4-12
Number of nodes per InP	40
Inter-provider link bandwidth capacity	unif distrib.[400,600]
Inter-provider link delay	unif distrib.[1,100]
Inter-provider link reliab	unif distrib.[0.88,1]
Inter-provider link cost	unif distrib.[10,50]
Intra-provider link bandwidth capacity	unif distrib.[200,300]
Intra-provider link cost	unif distrib.[1,50]
Intra-provider link delay	unif distrib.[1,6]
Inp connectivity probability	0.5
Node CPU capacity	unif distrib.[200,300]
Substrate node connection probability	0.5
Slice Request:	
parameter	Value
No.of VNFs per request	uni.distrib.[3,15]
Arrival distrubution	poison
Node cpu	uni.distrib.[1,20]
Bandwidth demand	uni.distrib.[1,50]
Bandwidth delay	uni.distrib.[50,200]
Mean arrival rate	2-14 requests per 100 time units
Life-time	1000 (mean)

3.6.3 Result Analysis

In this section, we analyze the performance of the proposed RL-based algorithm against the benchmark algorithms discussed in Section 3.6.1 considering both offline and online scenarios, in terms of acceptance ratio, revenue to cost ratio, average cost per accepted SFC request, and the request processing time. These performance metrics were explained in Section 2.4.1. The obtained results for the different scenarios are discussed in the following sub-sections

Offline scenario

In this section, we discuss the results obtained from the scenario considering offline demands. The results in Fig. 3.8 correspond to experiment 1 of this scenario in which the impact of substrate network size is analysed by varying the number of InPs from 4 to 12 considering an offline demand size of 3000 requests. From the results in Fig. 3.8a, as the number of InPs increases, the AR performance of all approaches increases. This is expected due to the increased amount of resources in the network. The proposed RL algorithm results in the highest value of AR with an average value of 47.63% while the rest of the approaches resulted in an average value of 41.66%, averaged across the different number of InPs. This is due to the fact that the RL approach jointly considers both cost minimization and reliability enhancement when mapping the request which leads to load balancing in the network, hence, avoiding link bottlenecks especially for a small number of InPs as reflected from the results. From Fig. 3.8b, DistNSE results in the best performance in terms of mapping cost with low InP numbers, but this performance degrades as the number of InPs increases. This is due to the fact

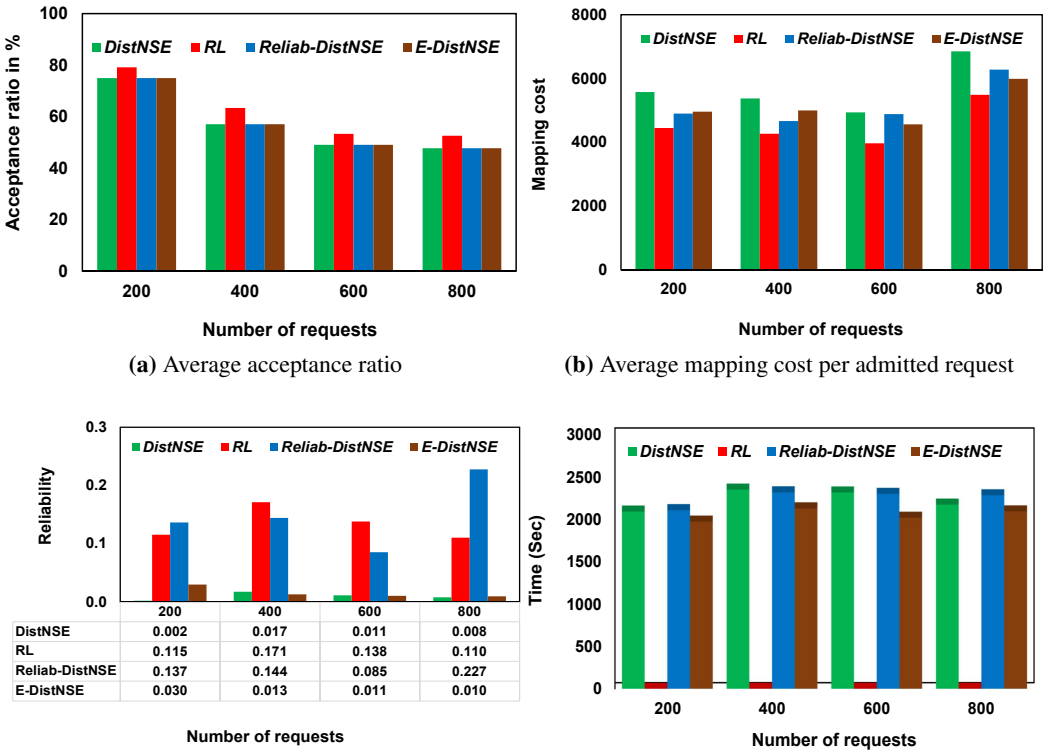


(a) Average acceptance ratio **(b)** Average mapping cost per admitted request

(c) Average end-to-end reliability per admitted SFC **(d)** Average processing time per admitted SFC

Figure 3.8: Scenario 1-Experiment 1: Impact of substrate network size for offline scenario

that DistNSE allows each InP along a given path to compete for only the sub-SFC selected by the immediate preceding InP along the path. This degrades the performance of this algorithm especially as the number of InPs along the path from source to destination increases, since the InPs at the far end of the path cannot compete for the VNFs selected by earlier InPs in the path. Overall, the RL approach results in the best performance in terms of mapping cost with an average cost of 4382.91 followed by E-DistNSE with an average cost of 4399.72 for each admitted request averaged across all substrate size. From these results, the E-DistNSE results in 5% improvement in terms of cost performance with no additional execution time compared to DistNSE whose average cost value per admitted request is 4632.69, further justifying the proposed enhancements to the DistNSE algorithm. The results in Fig. 3.8c show that the proposed RL based algorithm results in the best performance in terms of service reliability with an average value of 0.163, implying that on average, each request has a 16.3% chance of not experiencing a QoS violation during its life-time. This is followed by Reliab-DistNSE, DistNSE, and E-DistNSE with average values of 0.129, 0.03 and 0.03 respectively. The good performance of the RL and Reliab-DistNSE in terms of service reliability is expected since these incorporate this parameter while mapping the request as opposed to DistNSE and E-DistNSE. The processing time of the RL algorithm is relatively constant for the different network size with an average value of 5 milliseconds for each admitted request. This is expected since the input size of the



	200	400	600	800
DistNSE	0.002	0.017	0.011	0.008
RL	0.115	0.171	0.138	0.110
Reliab-DistNSE	0.137	0.144	0.085	0.227
E-DistNSE	0.030	0.013	0.011	0.010

(a) Average acceptance ratio

(b) Average mapping cost per admitted request

(c) Average end-to-end reliability per admitted SFC

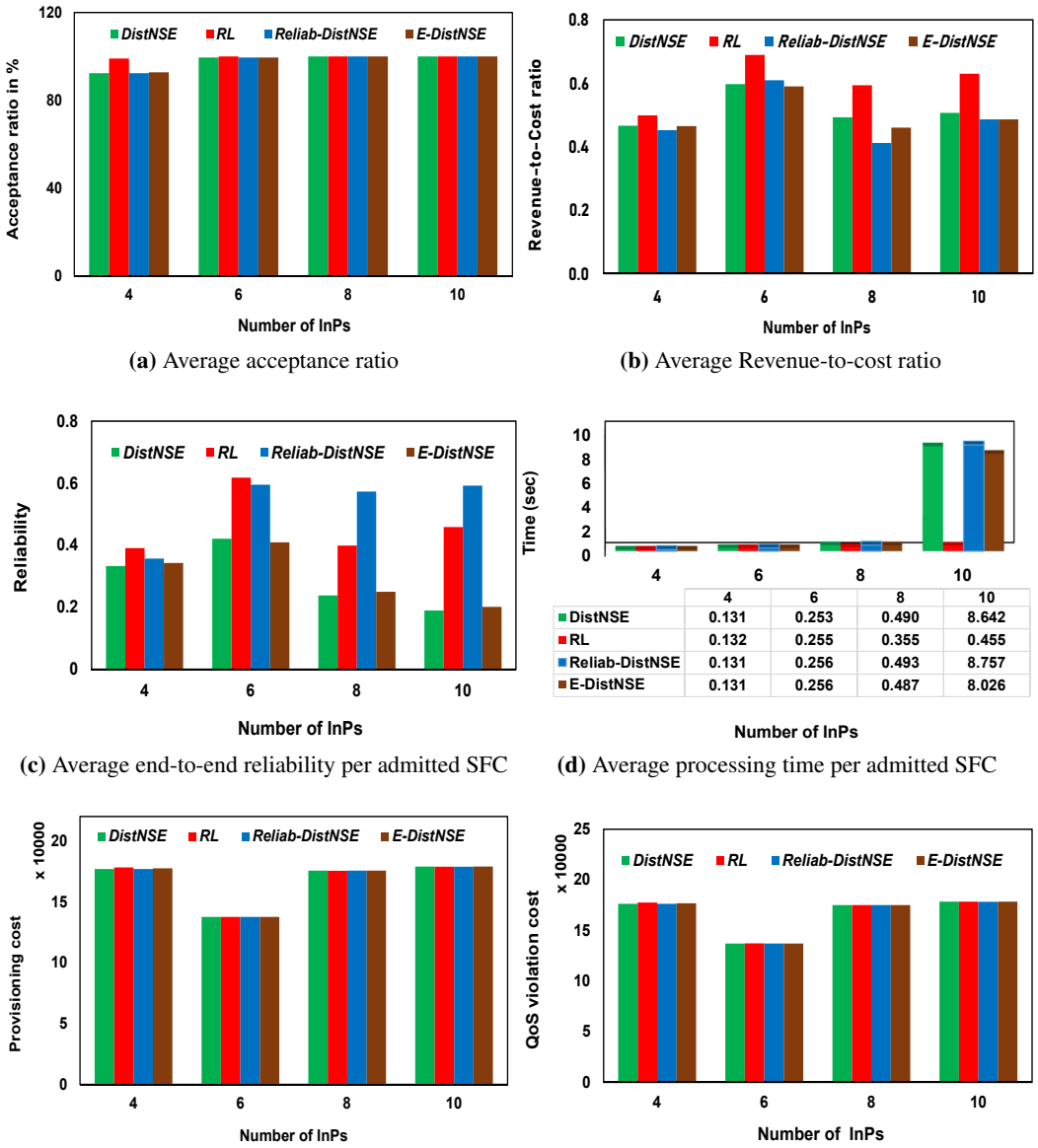
(d) Average processing time per admitted SFC

Figure 3.9: Scenario 1- Experiment 2: Impact of the number of offline requests

policy neutral network is constant for all the different network size. However, the time complexity of the rest of the algorithms exhibit a non-linear growth with the number of InPs. This is due to the fact that these algorithms involve computing all inter-domain paths from source to destination which is complex with a big network size.

The results in Fig. 3.9 correspond to experiment 2 of the offline scenario in which the impact of demand size is analysed by varying the number of offline demands from 200 to 800 demands considering 12 InPs. From Fig. 3.9a, the RL algorithm results in the highest performance in terms of acceptance ratio with an average value of 62.09% across the different demand size which a 5% improvement compared to the rest of the algorithms whose average AR value is 57.0%. The results in Fig. 3.9b reveal that RL results in up to 20% improvement in terms of mapping cost compared to DistNSE with an average value of 4542.81 per admitted request across the demand different sizes. This is followed by E-DistNSE, Reliab-DistNSE and DistNSE with average values of 5127.47, 5179.74 and 5683.47 respectively. These results again demonstrate that the proposed enhancements to the DistNSE algorithm results in an improvement of up to 10% in terms of mapping cost compared to the conventional DistNSE algorithm. The results in Fig. 3.9c show that Reliab-DistNSE results in the highest service reliability with an average value of 15% followed by RL with an average value of 13% across all demand size, further demonstrating the capability of RL to intelligently balance the

resource consumption cost and the service reliability requirements. E-DistNSE and DistNSE results in 2% and 1% service reliability respectively.



(a) Average acceptance ratio

(b) Average Revenue-to-cost ratio

(c) Average end-to-end reliability per admitted SFC

(d) Average processing time per admitted SFC

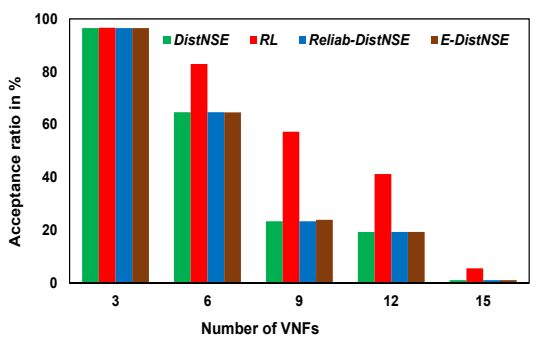
(e) Average provisioning Cost

(f) Average QoS violation cost

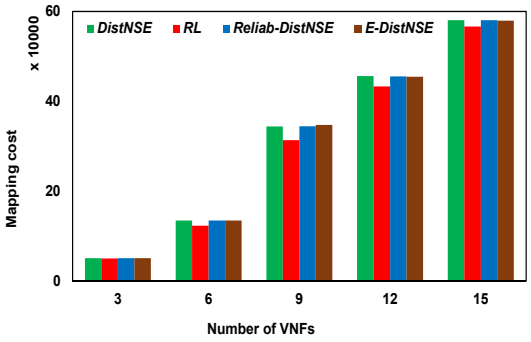
Figure 3.10: Experiment 1- Impact of substrate network size for online scenario

Online Scenario

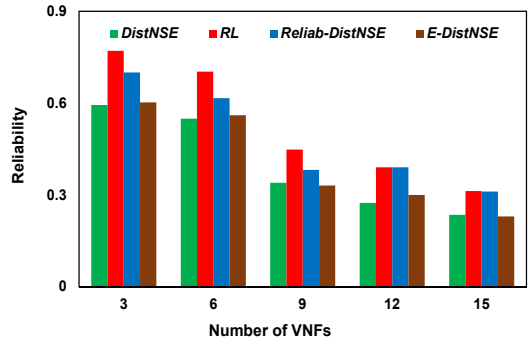
This section discusses the results obtained from the scenario considering online demands. The results of Fig. 3.10 correspond to experiment 1 of the online scenario in which the impact of the substrate network is analyzed by varying the number of InPs from 4 to 10 considering arrival rate of 4 users per 100 time units for a total of 40000 time units. From Fig. 3.10a, the AR performance of all algorithms improves with increase in the number of InPs as expected due to an increased amount of network



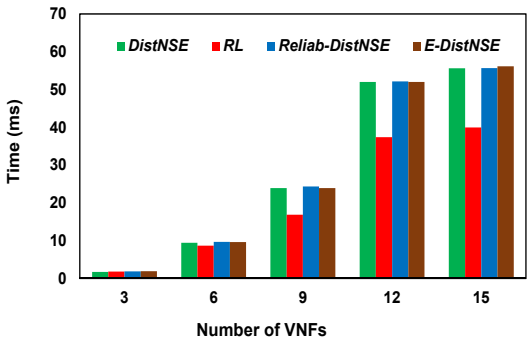
(a) Average acceptance ratio



(b) Average provisioning cost per admitted request

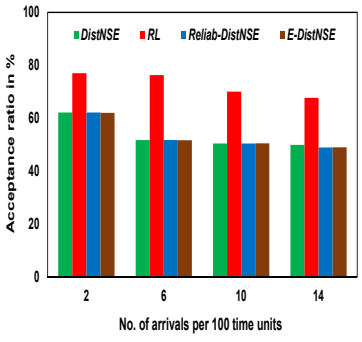


(c) Average end-to-end reliability per admitted SFC

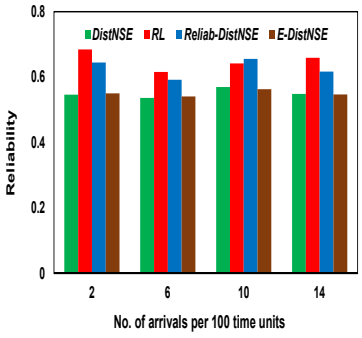


(d) Average processing time per admitted SFC

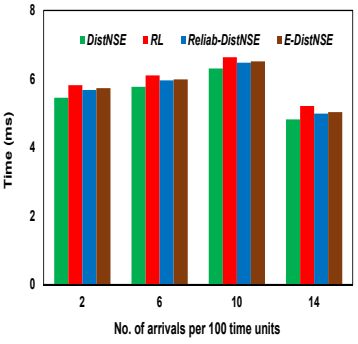
Figure 3.11: Experiment 2- Impact of the number of VNFs per request



(a) Average acceptance ratio



(b) Average end-to-end reliability per admitted SFC



(c) Average processing time per admitted SFC

Figure 3.12: Experiment 3 -Impact of arrival rate for online scenario

resources, with the RL algorithm resulting in the highest AR performance with a value of 99.74% averaged across all substrate network size. The average value of the other algorithms is 98.0%. From Fig. 3.10b, the proposed RL algorithm results in the highest value of revenue-to-cost ratio with average value of 0.60 and this is followed by DistNSE, E-DistNSE and Reliab-DistNSE with average values of 0.52, 0.50 and 0.49 respectively averaged across the different InP size. The reason for the superior performance of the proposed RL algorithm in terms of revenue-to-cost ratio is due to its ability to admit requests of big size, resulting in higher revenue compared to other algorithms. This is evident from Fig. 3.10f in which the QoS violation cost for the RL algorithm is the highest with an average value of 167147.1774 with DistNSE as an example having an average value of 166784.0736 averaged across the different InP size; yet, the average service reliability per request for the RL algorithm is 0.467 while that of DistNSE is 0.295 as shown in Fig. 3.10c. This means that even with more counts of service QoS violation, the revenue forfeited by the DistNSE based algorithms is lower since the requests involved are of small size, hence, less revenue and mostly likely fewer in number (due to lower AR performance), as compared to those admitted by the RL algorithm due to its inherent load balancing attribute. From Fig. 3.11c, Reliab-DistNSE and RL result in the best performance in terms of availability with average values of 0.529 and 0.467 respectively. This is expected since these incorporate service reliability attribute during the solution computation. These are followed by E-DistNSE and DistNSE with an average values of 0.30 and 0.295 respectively. As reflected in Fig. 3.10d, the average execution time for all the algorithms is fraction of seconds for small numbers of InPs. However, as the number of InPs increases, all the algorithms aside from RL exhibit a drastic increase in execution time due to the paths computation strategy of these algorithms. The average execution time of the RL algorithm is 0.3 seconds which is approx. 80% improvement over the other algorithms. This is followed by E-DistNSE with 2.2 seconds, and DistNSE and Reliab-DistNSE each with average value of 2.4 seconds, making RL a well suited approach for delay sensitive applications.

In the results shown in Fig. 3.11, the impact of the size of service requests is analyzed by varying the number of VNFs of each request from 3 to 15 considering arrival rate of 4 users per 100 time units for a total of 40000 time units and 6 InPs. From the results of Fig. 3.11a, the AR performance of all the algorithms degrades as the number of VNFs for each request increases. This is due to the fact that as the number of VNFs increases, the probability of the different VNFs finding candidate InPs to host them decreases due to the reduced amount of resources in the network, especially along the inter-domain links, since in this case, the different VNFs are more likely to be mapped across multiple InPs. The proposed RL algorithm results in the highest AR performance with an average value of 56.76% (ie 15% improvement) averaged across the different VNF numbers. The rest of the algorithms result in almost the same AR performance with an average value of 41.0%. The RL algorithm results in a superior performance, especially with high VNF numbers, due to the load balancing attribute that is inherent in this algorithm which leads to a reduction in the number of link bottlenecks in high traffic load. The results of the average provisioning cost (sum of mapping cost and QoS violation cost) are shown in Fig. 3.11b, the RL algorithm results in the lowest average provisioning cost per admitted request with an average value of 297,210.28 (5.1% improvement compared to DistNSE) followed by Reliab-DistNSE, DistNSE, and E-DistNSE with average values

of 313,157.52, 313,307.63 and 313,473.73 respectively. The superior performance of RL is attributed to the fact that it considers both reliability and cost in mapping the request, hence, incurring low costs in terms of mapping and QoS violation cost. The average processing time per admitted request of all the algorithms tends to grow with the number of VNFs as reflected in Fig. 3.11d. This is expected due to the intra-domain mapping overhead. However, for this case, all algorithms process each request in fractions of a second with the average processing time of the RL algorithm being 40 milliseconds while that of other algorithms is approximately 56 milliseconds. As expected, the RL and Reliab-DistNSE result in the highest service reliability with an average value of 0.52 and 0.48 respectively as reflected in Fig. 3.11c. The results in Fig. 3.12 correspond to experiment 3 of the online scenario in which the arrival rates of the requests is varied from 2 to 14 considering 6 InPs. The results in Fig. 3.12a show that the RL algorithm results in the highest performance in terms of AR with an average value of 72.722 (26.3% improvement) with the rest of the algorithms having an average AR performance of 53.3%. The running time of all algorithms tend to increase with the number of arrivals, with all algorithms executing in a fraction of seconds for this case.

3.7 Conclusion

This chapter has incorporated service reliability consideration to the problem of multi-domain service orchestration. A request partitioning algorithm relying on a policy neural network has been proposed. The algorithm is shown to result in up-to 26% improvement in terms of acceptance ratio and up-to 10% improvement in terms of provisioning cost in some scenarios compared to a state-of-art benchmark algorithm. Moreover, the proposed algorithm is found to be scalable with change in both network and request size, executing in polynomial time in both cases. In addition, the Chapter has proposed an enhancement to a state-of-the-art algorithm resulting in up-to 10% and 5% performance improvement in terms of acceptance ratio and mapping cost respectively.

CHAPTER 4

A multi-stage graph aided algorithm for distributed Service Orchestration

4.1 Introduction

The thesis proposal in chapter 3 adopts a centralized scheme in which a centralized entity relies on the disclosed global information to make decisions about the different InPs for hosting the requests. However, such an approach which is also adopted in [7, 11–13, 19, 35–37, 47–50, 62, 63, 93, 117] may not be scalable when considering large scale networks, due to the excessive overhead imposed on the central decision entity. Moreover, any malfunction within that entity affects the performance of the entire system. This motivates the adoption of distributed algorithms in which the orchestration decision involves participation of different InPs, rendering them more suited for dynamic network environments and limited information disclosure scenarios. However, conventional distributed approaches are penalized by an increasing processing delay and signaling overhead as the number of participating InPs increases, hence, compromising their scalability. In light of the above, this chapter seeks to address the above challenge by proposing an algorithm for the distributed provisioning of service requests across multiple infrastructure providers which targets to minimize both, the number of InPs participating in the service orchestration process, and the average number of messages processed by each of the participating InPs.

The rest of this chapter is organized as follows: Section 4.2 introduces a description of the problem addressed in this Chapter. Then, the distributed algorithm proposed for solving the above problem is discussed in Section 4.3 including its time-complexity analysis. The performance of the proposed algorithm is evaluated in Section 4.4 and the chapter is concluded in Section 4.5.

4.2 Description of the problem

Similar to the proposal in Chapter 3, the target of the distributed orchestration algorithm introduced in this chapter is to minimize the average provisioning cost for each admitted service request. In principle, minimizing the provisioning cost of any request minimizes the operational cost of the service provider and maximizes the resultant net revenue, which is aligned with the target of NSPs in practice. Moreover, the thesis considers a practical scenario where the cost for each unit of any node

or link resource may vary across different InPs. In this chapter, the provisioning cost of any request is considered to be influenced by the following cost components:

- Energy consumption cost associated with running the different VNFs onto the substrate nodes of the different domains.
- Transmission cost of transferring the user traffic from the ingress node to the egress node along all the intermediate links.
- Processing cost incurred for processing the user traffic at the different VNFs traversed.

Note, however, that other cost components, such as VNF instantiation, could be easily integrated into the adopted cost model of the algorithm. Therefore, the multi-domain service orchestration problem targets to minimize the average provisioning cost for each admitted request, and mathematically formulated as:

$$Minimize \quad \frac{1}{|R_A|} \sum_{r \in R_A} C_p^r(Gv) \tag{4.1}$$

where $C_p^r(Gv)$ is the provisioning cost for a request $r \in R_A$, and R_A denotes the set of all admitted requests, with $|R_A|$ being the cardinality of that set. In order to evaluate $C_p^r(Gv)$, the following binary variables are introduced: $\sigma_{l_{uv}}^{e} \in \{0,1\}$ =1 if the request virtual link l_{uv} is provisioned by the intra-domain edge $e \in E_s^k$ of domain $k \in \mathbb{K}$, zero otherwise; $\sigma_{l_{uv}}^{e_{int}} \in \{0,1\}$ =1 if the request virtual link l_{uv} is provisioned by the inter-domain edge $e_{int} \in E_{int}$, zero otherwise; $y_{n_s^k}^{n_v^p} \in \{0,1\}$ =1 if the request virtual node n_v^p is provisioned onto substrate node n_s^k, zero otherwise. Then, the request provisioning cost can be evaluated as:

$$\begin{aligned} C_p^r(Gv) = & \sum_{l_{uv} \in L_v} \sum_{k \in K} \sum_{e^k \in E_s^k} \sigma_{l_{uv}}^{e^k} \times \zeta^{e^k} \times \rho^r \\ & + \sum_{l_{uv} \in L_v} \sum_{e_{int} \in E_{int}} \sigma_{l_{uv}}^{e_{int}} \times \zeta^{e_{int}} \times \rho^r + \\ & \sum_{n_v^p \in N_v} \sum_{k \in K} y_{n_s^k}^{n_v^p} \times \zeta_{n_s}^k \times \rho^r + \chi_w \sum_{k \in K} \sum_{n_s^k \in N_s^k} E_{n_s^k} \end{aligned} \tag{4.2}$$

where the first and second terms of Eqn. 4.2 correspond to the transmission costs due to the use of the intra-domain and inter-domain substrate edges, respectively, and the third and fourth terms correspond to the processing costs, due to the use of the selected substrate nodes, and the energy costs, respectively. The parameter $E_{n_s^k}$ from the energy cost term denotes the energy consumption at node n_s^k, and χ_w denotes the cost per unit of energy consumption. $E_{n_s^k}$ is computed using the model adopted in [89] as follows:

$$E_{n_s^k} = e_{n_s^k}^{idle} + [e_{n_s^k}^{busy} - e_{n_s^k}^{idle}] \times U_{til}^{n_s^k} \tag{4.3}$$

where $e_{n_s^k}^{idle}$, $e_{n_s^k}^{busy}$ denote the idle and peak power consumption of the node n_s^k. The term $U_{til}^{n_s^k}$ refers to the utilization level of substrate node n_s^k.

The optimization of Eqn. 4.1 is performed while adhering to constraints specified in Eqns. 2.6-

2.13 as introduced in Chapter 2. Given the NP-hard nature of the above problem formulation, this chapter proposes a multi-stage graph aided heuristic that targets to achieve near-optimal solutions within feasible run-times, while preserving the privacy of the participating InPs. The proposed algorithm is introduced and described in the following section.

4.3 Proposed algorithm for distributed service orchestration

This section introduces the proposed multi-stage graph aided algorithm , denoted as MuL in the remainder of the chapter, for distributed orchestration of service requests across multi-domain networks. Specifically, the steps involved in the algorithm execution, including their corresponding pseudo-codes, are described. For the multi-domain service deployment problem, the service embedding algorithm targets to obtain a set of InPs that minimizes the service deployment cost while satisfying the request requirements. Given the large number of possible combinations for mapping the different VNFs of the request, this problem is computationally intractable. Hence, looking for exact solutions becomes unfeasible, especially for large network scenarios. This is further exacerbated by the reluctance of InPs to disclose information related to their internal topology or policies. This way, it makes conventional heuristics, based on full information exposure, unfeasible for this problem. With this motivation, this chapter proposes an approach that obtains the provisioning solution in three phases that will subsequently be described, with the aim to reduce the problem dimension successively. The proposal is able to obtain near-optimal solutions in practical run-times while preserving the privacy of the different InPs.

The algorithm consists of 3 main steps: a Candidate InP identification, a Message exchange and a Consensus step. The proposed algorithm uses a candidate search technique to identify potential InPs that can host a fraction or the whole request. Then, these candidate InPs are used to build a multi-stage graph, where, at each stage, all the candidate InPs of a given VNF are represented as a different node, and the interconnecting edges between the nodes of consecutive stages are the available inter-domain substrate paths. Using this multi-stage graph, a message block is constructed at the leftmost stage (source end) and propagated upstream towards the destination node. Each node, through which the message block passes, updates the received message block by increasing the cumulative mapping cost, the total cumulative delay and the list of traversed nodes, before forwarding this message block to all the nodes of the next stage. At the output end, the message block associated with the least cost value is chosen as the optimal message block, and the nodes through out which this message block was transiting are chosen as the optimal nodes/InPs for embedding the request. A detailed description of these three steps follows below:

4.3.1 Candidate InPs Identification step

This step exploits the fact that each request virtual node of a given SFC is constrained by a function/resource and location requirement. Similarly, the corresponding virtual links between any two virtual nodes are considered to be constrained by a delay and a bandwidth requirement. Therefore,

each request virtual node of a given request may only be served by a subset of the available InPs that satisfy the associated constraints. The aim of this step is to associate each request virtual node with a set of InPs that can satisfy its associated constraints. Different to approaches such as in [35], in which all domains participate in the distributed solution computation, selecting a subset of InPs to participate in the solution computation minimizes the execution time of the algorithm and the amount of signaling information exchanged among the involved participants.

Whenever a request arrives, the master Orchestrator (MO) compares the specifications of the request with the global information disclosed by the orchestrators of the different administrative domains, with the goal of identifying the potential domains that could host the SFC request. The set of candidate InPs is obtained by matching the virtual nodes location and resource type constraints with the disclosed information of each InP, and also by matching the virtual links' constraints with the inter-domain links' attributes. Thus, for an InP to be among the candidate set of InPs for a given request virtual node i, the set of offered resource types disclosed by this InP must include the function type of node i, and the geographical span of that InP must include the acceptable location of this virtual node. Similarly, for two consecutive request virtual nodes i and j to be hosted by InP A and InP B, where $A \neg B$, there should exist an inter-domain path between InP A and InP B that satisfies the constraints of virtual link $i-j$. If we denote by Γ_k, $loc^{n_v^p}$ and η_{vnf}^k as the geographical span of InP k, the desired location of virtual node n_v^p, and a set of VNF types that can be provisioned inside InP k respectively, then an InP is considered a potential candidate for virtual node n_v^p iff:

$$loc^{n_v^p} \in \Gamma_k \tag{4.4}$$

$$p \in \eta_{vnf}^k \tag{4.5}$$

Equation 4.4 requires that the acceptable location of virtual node n_v^p lies within the coverage span of InP k. From equation 4.5, such an InP should support the resource type required by n_v^p. The pseudo-code of the candidate InPs Identification step is shown in Algorithm 2. The algorithm starts by initializing the set of candidate InPs for the request, $Cand_s^r$, to an empty set. Then, for each virtual node $n_v^p \in N_v$ of the request, the algorithm extracts all InPs that satisfy the resource type constraint, location constraint and also have a feasible connection (in terms of bandwidth and delay) between the source node S_n and destination node τ_n, as potential candidates for this virtual node n_v^p, and stores these in the set $Cand^{n_v^p}$. In the case that any VNF has no potential candidate, the request is rejected at this point. Otherwise, the algorithm returns the candidate set $Cand_s^r$ made up of candidates for all the virtual nodes of the request.

4.3.2 Message Exchange Step

The Message exchange step can also be viewed as a distributed computation step involving each candidate InP of the request forwarding Message Blocks (MBs) to a given subset of candidate InPs in order to deduce a mapping solution. The Message exchange is guided by a multi-stage graph constructed by the MO and based on the obtained candidate InPs. The leftmost stage in the

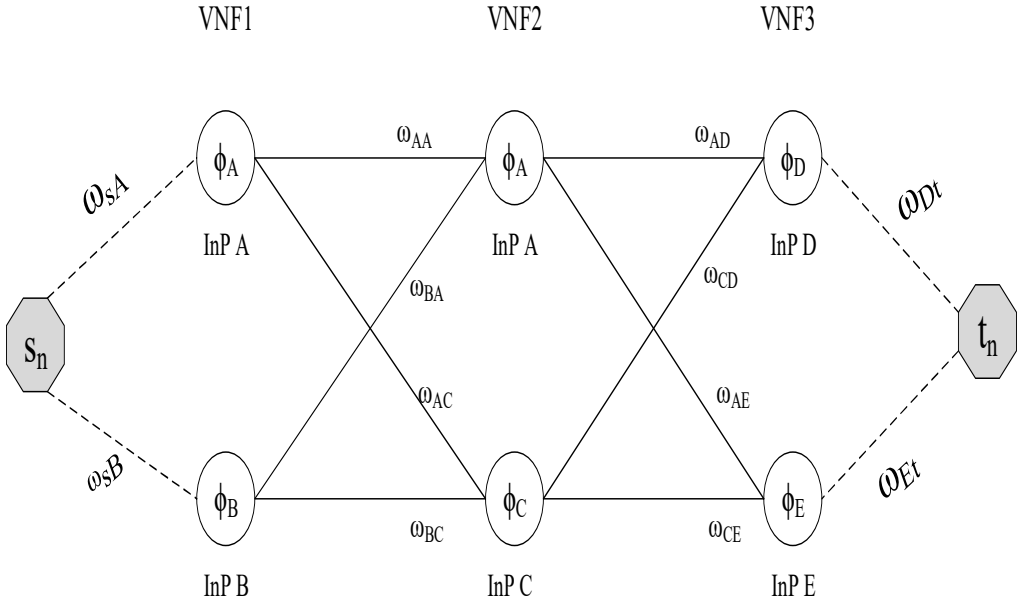


Figure 4.1: An illustration of a multi-stage graph for a request in which the traffic has to traverse 3 VNFs. In this case, InP A is a candidate for both VNF1 and VNF2.

graph corresponds to the source node (originating InP) and the rightmost stage corresponds to the destination node (terminating InP). Each intermediate stage of the graph corresponds to a specific required VNF of the request. The nodes considered at each stage of the multi-stage graph are the candidate InPs for the provisioning of the VNF at that stage.

An example of such a multi-stage graph is shown in Fig. 4.1 for a request in which the traffic has to traverse three VNFs, a single source and a single destination, and candidate sets for the VNFs being: VNF1=$\{A, B\}$, VNF2=$\{A, C\}$, VNF3=$\{D, E\}$. The connection between any two InP nodes X and Y from adjacent stages of the graph, where $X \neg Y$, corresponds to the physical path connection between the peering nodes connecting InPs X and Y. As such, the weight parameter ω_{XY} stands for the weight of that path in terms of features such as delay, residual bandwidth, and monetary cost, among others. For the particular case of the same InP being a candidate for two consecutive VNFs, (i.e., X=Y), the connection path and weight metric ω_{XY} corresponds to the intra-domain path between the candidate hosting nodes of these VNFs. Each node k in the graph is characterized by a parameter ϕ_k which represents the undisclosed information matrix of the corresponding InP. This includes the cost per unit of resource and the internal topology, among others, attributes that are only known by the specific domain orchestrator.

To understand the executed procedure of this step, and using Fig. 4.1 as an illustrative example of a possible multi-stage graph for an SFC with 3 VNFs, we define the following terms:
• Message Block (MB): This denotes a single message unit built as a tuple <ID_track, $Edge\_track$, $Cost_{cum}$, Del_{cum}>. The ID_Track component, which is initialized as an empty list, stores all the

Algorithm 2 Candidate InPs Identification Algorithm

Input:G_s,G_v
Output: Candidate set,$Cand_s^r$
Initialise: $Cand_s^r$=$\emptyset$
for *Each virtual node* $n_v^p \in N_v$ **do**
  $Cand^{n_v^p} = \phi$
  for *Each Inp* $k \in K$ **do**
    if $r^{n_v^p} \in \Gamma_k$ *AND* $p \in \eta_{vnf}^k$ **then**
      if *dijkstra(k,*S_n *) AND dijkstra(k,*t_n *)*$\neq Inf$ **then**
        Add k to $Cand^{n_v^p}$
      end
    end
  end
  if $Cand^{n_v^p} = \phi$ **then**
    Reject Request
  end
  else
    Add $Cand^{n_v^p}$ to $Cand_s^r$
  end
end

IDs of the nodes/InPs that have modified the message block at the different stages (i.e., feasible candidates for the different VNFs through which the MB has traversed) from source to destination. As an example, if a message block from the source InP traverses InPs B,C,D before reaching the terminal node, then, the ID_Track for this MB at the terminal node will be , $ID\_Track = [s_n, B, C, D, \tau_n]$. The $Edge\_track$, initialized as an empty list, stores all the inter-domain edges that have been traversed by the message block from source to destination. Considering the above example in which $ID\_Track = [s_n, B, C, D, \tau_n]$, then $Edge\_track = [s_n\text{-}B, B\text{-}C, C\text{-}D, D\text{-}\tau_n]$ at the terminal node. The $Cost_{cum}$ and Del_{cum} components, initialized to zero both of them, capture the cumulative cost and delay respectively along the different paths traversed by the message block (computing for both nodes and links) from source to destination. Note that each message block corresponds to a possible mapping solution from the source node to the VNF corresponding to the last index in ID_Track. We denote by MB_n^m the message block sent from node n to node m. Note that, in this case, the stage of node n has to be to the left of that of node m.

• Message Block Set (MBS): This denotes a set of one or more message blocks.

• Optimum Message Block (MB_{opt}): This denotes the message block from the message block set that has the least cost value among all valid message blocks in that set.

• Pushing node set, k_{push}^n: The pushing node set of a given node $k \in K$ at stage n of the multi-stage graph denotes the set of all nodes in the preceding stage $(n-1)$ to which the node k has a feasible connection. Any node in such a set is called a pushing node with respect to k. As an example, the pushing node sets for the nodes in the third and fourth stages of Fig 4.1 are: $A_{push}^3 = \{A, B\}$, $C_{push}^3 = \{A, B\}$, $D_{push}^4 = \{A, C\}$, $E_{push}^4 = \{A, C\}$

• Receiving node set, k_{rec}^n: A receiving node set with respect to node $k \in K$ at stage n refers to the

set of all nodes in stage (n+1) to which the node k has a feasible connection. As an example, the receiving node set for node C is $C^3_{rec} = \{D, E\}$.

Then, the execution of the Message exchange step is as follows:
Starting from the leftmost stage (source node), the MO initializes N message blocks, where N is the number of candidate InPs at the next stage (i.e. the size of the receiving node set for the MO. This is equal to 2 for the source node in Fig. 4.1), with each message block MB^n_{MO} intended to be forwarded to a specific receiving node n. Then, for each receiving node n, it computes the shortest available path from the source node to node n, and obtains the delay, cost and the inter-domain edges constituting this path. Then, for each message block MB^n_{MO} to be forwarded to each receiving node n, the MO appends: its index into the ID_Track component, the obtained inter-domain edges into the $Edge\_track$ component, and the cost and delay values to $Cost_{cum}$ and $Delay_{cum}$ components, respectively. Then, the MO forwards to each receiving node n the corresponding MB, i.e. MB^n_{MO}, for further processing. On receiving the MB, each node n at stage l ($l = 1$ if received from the source stage) identifies the receiving node set (i.e., the candidate nodes at stage $(l + 1)$) from the multi-stage graph. Note that these are the candidates of the VNF to be enumerated in the next round. Then, for each node m, among the receiving candidates, node n performs the following steps:
• Obtains the optimal message block $MB^{n,m}_{opt}$ from all the message blocks it has received. The $MB^{n,m}_{opt}$ refers to the message block at node n with the least cost among the feasible message blocks to be propagated to node m. A message block is feasible to be forwarded to node m of the next stage if: $i)$ the node m is not already part of the ID_track, unless it is the same as the current sending node (i.e., it is a candidate for both the current VNF and the next VNF, implying m=n); $ii)$ the sum of $Delay_{cum}$ and the additional intra-domain delay to the substrate node where the VNF is to be mapped inside node n does not exceed the acceptable delay.
• Obtains the available shortest path from node n to node m. Note that this path should not include already used edges that appear in the $Edge\_track$ of the $MB^{n,m}_{opt}$. This is done in order to guarantee that the user traffic from source to destination does not traverse the same edge twice. The intra-domain delay and the intra-domain cost (for nodes and links) is evaluated and added to the delay and cost of the obtained shortest path from n to m. These are then used to increase the $Delay_{cum}$ and $Cost_{cum}$, respectively, of the $MB^{n,m}_{opt}$. Also, the index of node n is added to the ID_Track component.
• Forwards $MB^{n,m}_{opt}$ to node m. Node m and the following ones will also execute the same steps until the message blocks will reach the last stage. In the case that a node is unable to push a message block to at least one of the nodes of the next stage, that node mutes all the received MBs and sends back a mute message to the MO. In the event that all the candidate nodes at a given stage have responded with a mute message, the request is rejected, and the algorithm execution stops, since this means that there is no feasible connectivity between the current VNF and the VNF corresponding to the next stage.

4.3.3 Consensus and Binding step

Once the node at the last stage of the graph has computed its associated MB_{opt}, it forwards its MB_{opt} back to the MO, the MO then selects the ID_track component of the message block with the lowest cost as the definitive mapping solution for the SFC request. This is constituted by IDs of InPs that result in the least mapping solution from the source to the destination. Finally, the resources across the inter-domain links and those inside the selected domains are reserved for deploying the request.

In possible situations where the last stage could be associated with multiple nodes (e.g., in case of multiple alternative servers in which the user may access content), then each of the nodes in the last stage computes its corresponding MB_{opt} and forwards it to the MO. This then selects the MB_{opt} with the least cost as the definitive mapping solution. If this algorithm is to be executed in a fully distributed fashion, the execution of this step implies that the candidates of the last stage know each other (through the multi-stage graph which can be shared by the MO with all the candidate nodes). Then, once each node in the last stage of the graph has done its internal computation and evaluation, it forwards a copy of its resulting MB_{opt} to each of the other candidates in this stage. Then, each node inspects all the MB_{opt} messages at its disposal including its own. If the MB_{opt} of such a node has the lowest cost value, the node sends a "$back-off$" message to all the rest of the nodes, and it forwards its own MB_{opt} to the MO from which the definitive mapping solution is chosen as the ID_track component of the MB_{opt}, with the $Edge\_track$ indicating the inter-domain edges of the solution.

Algorithm 3 Distributed Computation Step

```
Input: Multi-level Graph, G_v
Output: Mapping Solution
j=0
while j < J do
    for Each level j ∈ J do
        for Each recepient node m at level (j + 1) do
            Rec_MB^m=[]                                   ▷ Collect received MBs
            for Each forwarding node n at level j do
                Evaluate the optimal MB, MB_opt ∈ Rec_MB^n
                Update MB_opt
                if n==Terminal node then
                    Return MB_opt                         ▷ Chosen mapping solution
                end
                else
                    Forward MB_opt to Rec_MB^m
                end
            end
        end
        if Rec_MB^m ∀ m at level (j+1) then
            Reject request
        end
        j=j+1
    end
end
```

4.3.4 Time complexity analysis

The main steps of the proposed algorithm are: the computation of the candidate sets of InPs for each VNF of the request; the processing and forwarding of message blocks from each node at each stage towards the receiving nodes of the next stage; and the selection of the InP set for the provisioning of the request. The time complexity of extracting a candidate set of InPs for each request virtual node is linear in terms of the number of InPs K, $\Theta(K)$. The messages exchange step involves the use of the Dijkstra algorithm to compute the shortest path between each node i at stage n and each node j at stage (n+1) of the multi-stage graph, where $i \neq j$, as well as the evaluation of the intra-domain cost for the mapping of the VNF corresponding to stage n. The time complexity associated with the shortest path computations can be approximated as $\Theta((2C_N + (M-3)C_N^2) \times |K|log(|K|)) \approx \Theta((V-3)C_N^2) \times |K|log(|K|))$, where C_N is the number of candidate nodes for each VNF (in practice, this may be different for the different VNFs, and the same InP may be a candidate of more than one VNF). V is the number of stages in the graph, including those corresponding to the ingress and egress nodes. The time complexity of the intra-domain cost evaluation depends on the specific single domain algorithm used for the intra-domain mapping. In this work we used the algorithm we proposed in [84] with some modifications to suit the mapping of chained Sub-SFCs. In general, the time-complexity of the entire proposed algorithm is guaranteed to be less than $\Theta([(V-3)C_N^2) \times |K|log(|K|)] + [K \times (|N_v|-3)N_s) \times |N_s|log(|N_s|)])$, where N_v is the number of VNFs and N_s is the number of substrate nodes for an InP. In practice, the different InPs can only support a finite number of VNFs, hence, limiting the number of candidates for each VNF. Moreover, due to the finite number of VNFs that can be supported by each InP (due to resource type and capacity constraints), the number of possible candidates for each VNF decreases as the SFC size increases, binding the time complexity of the algorithm as the SFC size increases.

4.4 Performance evaluation of the proposed algorithm

In this section, the performance evaluation of the proposed algorithm is described including a description of the simulated scenarios and a discussion of the obtained results. The performance of the proposed algorithm is evaluated against the following bench-mark algorithms:

• Distributed Network Service Embedding (DistNSE) algorithm proposed in [19]. DistNSE exploits the disclosed public information to compute feasible paths between source and destination nodes, and from that, the path with the least cost is chosen for mapping the service request. By considering all possible paths from source to destination to obtain all feasible solutions, the benchmark DistNSE algorithm is optimal in terms of acceptance ratio, hence, it becomes a suitable algorithm for performance bench-marking. In this comparison, the thesis considered the best performance scenario of the DistNSE algorithm, in which all feasible paths from the ingress to the egress nodes are evaluated, and from them the best path was selected.

• Multi-level Aggregation Algorithm (MuL-Ag). The thesis designs this as a benchmark algorithm with its execution being similar to the proposed MuL algorithm. However, at each stage, instead of each node evaluating the optimal message block to be forwarded to the nodes of the next stage, that

node aggregates/combines all the received messages into a message set and forwards all these to the next stage nodes as it is the case adopted by distributed algorithms in literature [35]. The target of this design is to demonstrate the gain resulting from the technique proposed by the thesis of only sending a single message from a node to another given node.

4.4.1 Simulation environment and settings

The performance evaluation is done considering a substrate network with InPs varied from 4 to 12 depending on the experiment under consideration, with each InP being modeled by a real network topology, namely a BIC topology [112]. The specific values of the different simulation parameters and settings for the substrate network and service requests are given in Table. 4.1.

Table 4.1: Simulation parameters for the multi-stage graph based distributed algorithm

Substrate Network:	
parameter	Value
Number of InPs	4-12
Inter-Inp connection probability	0.4
Number of nodes per InP	33
Node CPU capacity	unif distrib.[200, 300]
Link bandwidth capacity	unif distrib.[200, 300]
Link delay	unif distrib.[1, 6]
Processing cost per 1 GB of data, ζ^{n_s}	unif distrib.[$0.15, $0.22]
Transmission cost per 1 GB of data, ζ^{e}	unif distrib.[$0.05, $0.12]
Processing delay of a packet at each VNF	unif distrib.[0.0045, 0.3] milliseconds
Service Request:	
Parameter	Value
Number of VNFs per request	unif.distrib.[2, 10]
Packet rate, ρ^{r}	unif.distrib.[400, 4000] packets/s
Packet size	64KB
Mean arrival rate	2-10 per 100 time units
Arrival distribution	Poisson
Life-time	Exponentially distributed with mean 1000

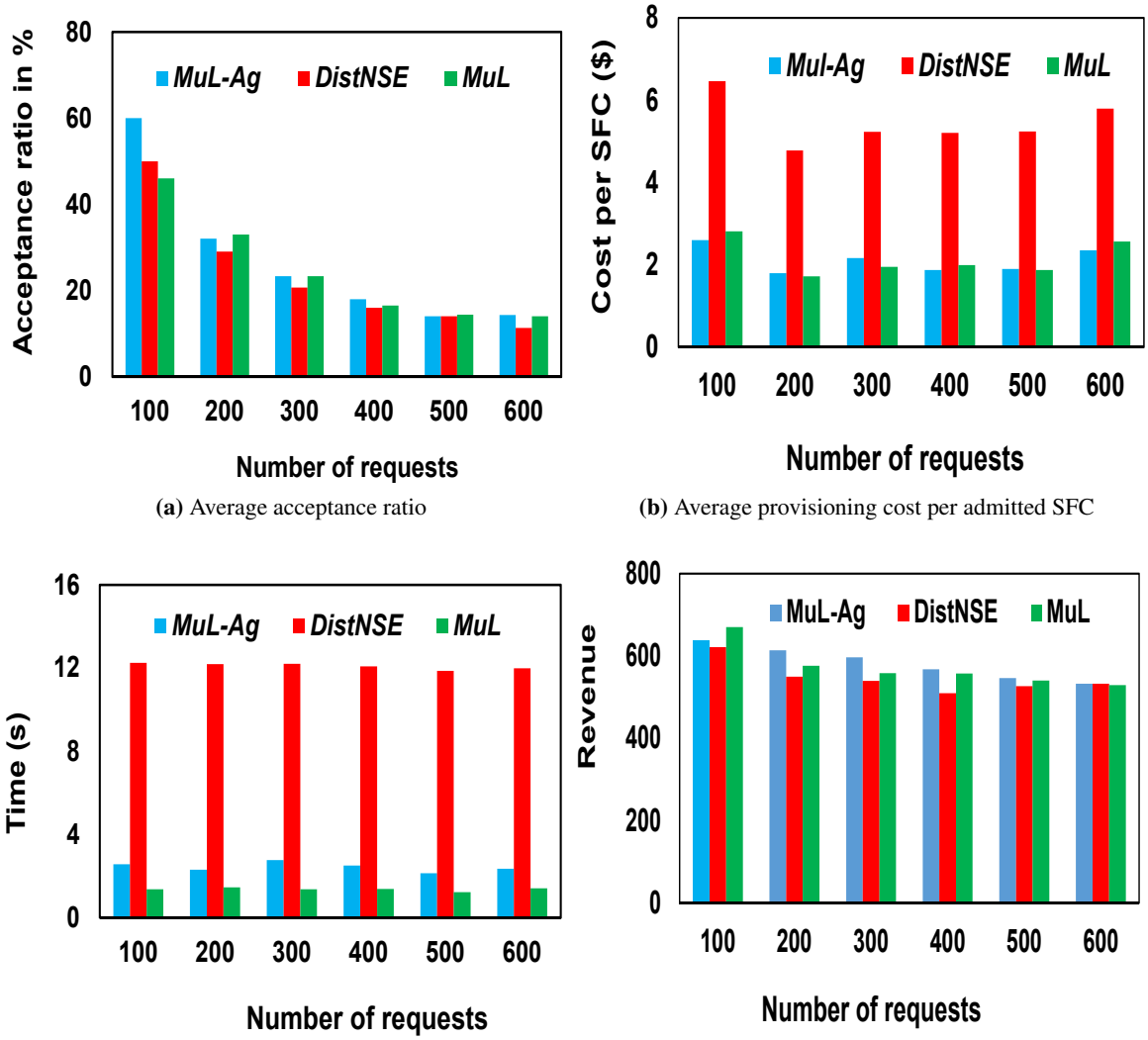
4.4.2 Results and discussion

This section presents and discusses the results obtained from the different experiments conducted considering both online and offline behavior of service requests. The performance is evaluated considering metrics of average acceptance ratio, average provisioning cost per admitted request, average revenue and average request provisioning time as described in Section 2.4.1:

Offline scenario

This section presents and discusses the results obtained from different experiments considering the offline scenario:

In Experiment 1, whose results are shown in Fig. 4.2, the impact of the demand size on the performance of the algorithms is analyzed by varying the number of requests from 100 to 600 requests considering 10 InPs. From Fig. 4.2a, the 3 algorithms have the same competitiveness



(a) Average acceptance ratio

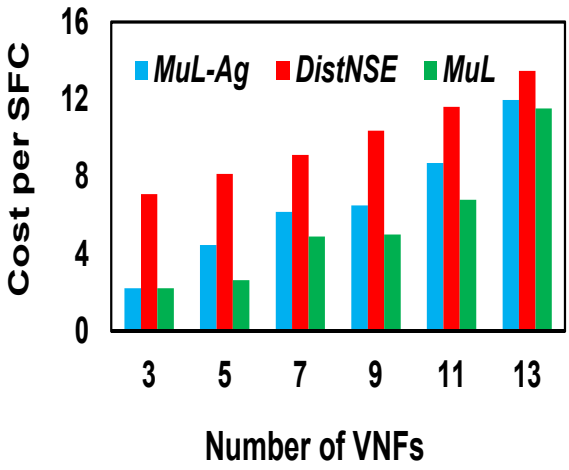
(b) Average provisioning cost per admitted SFC

(c) Average processing time per admitted SFC

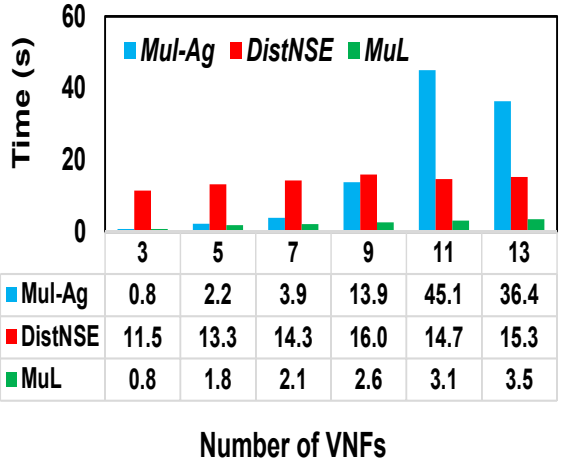
(d) Average revenue per admitted request

Figure 4.2: Experiment1: Impact of demand size for offline scenario.

(within a 4% margin) in terms of acceptance ratio, with an average value of: 27.0%, 23.5% and 24.5% for MuL-Ag, $DistNSE$ and MuL, respectively, averaged across all the tested demand sizes. However, $DistNSE$ results in the worst performance in terms of average mapping cost per admitted SFC, with an average value of 5.45$, which is approximately 60% higher compared to MuL and MuL-Ag, whose average cost values are: 2.11$ and 2.15$, respectively, averaged across the different demand sizes. The poor performance of $DistNSE$ in terms of mapping cost is attributed to the fact that in $DistNSE$, InPs can compete only for the previously mapped sub-SFC, as opposed to the multi-stage algorithms, in which an InP can compete for any sub-SFC of the request as long as it is a valid candidate. The results in Fig. 4.2c demonstrate the superior performance of MuL algorithm in terms of average processing time per admitted request, with an average value of 1.35



(a) Provisioning cost per admitted SFC



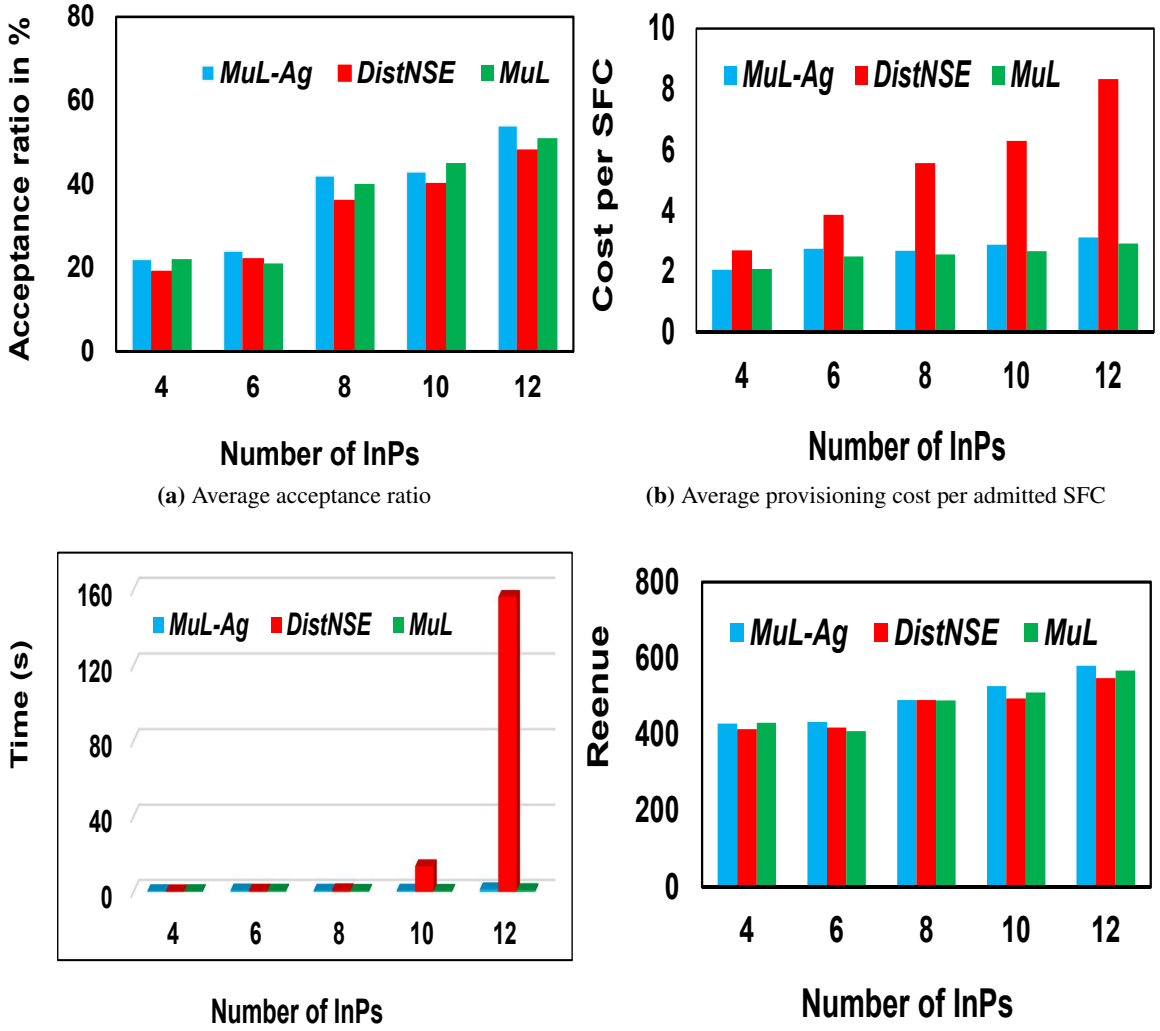
	3	5	7	9	11	13
Mul-Ag	0.8	2.2	3.9	13.9	45.1	36.4
DistNSE	11.5	13.3	14.3	16.0	14.7	15.3
MuL	0.8	1.8	2.1	2.6	3.1	3.5

(b) Average processing time per admitted SFC

Figure 4.3: Experiment 3: Impact of request size for offline scenario

seconds across all demands. This translates into a performance improvement of 44.1% and 88.79% compared to MuL-Ag and $DistNSE$, respectively, whose average values are: 2.45 seconds and 12.1 seconds. For the MuL algorithm, each node at a given stage forwards a single message block to each receiving node at the next stage, this results in a lower processing load at the receiving nodes, hence, reducing the execution time compared to MuL-Ag, in which each node forwards all aggregated message blocks to each receiving node at the next stage. In terms of average revenue per admitted request, MuL is as competitive as MuL-Ag (within a 2% margin), and results in a 4.4% improvement compared to $DistNSE$, as shown in Fig. 4.2d. Moreover, the average revenue for each admitted request tends to decrease when increasing the demand size, due to the decreased resources in the network, making it increasingly difficult to admit requests with high revenue. In summary, experiment 1 has demonstrated that MuL results in a better performance in terms of mapping cost and execution time compared to $DistNSE$. In terms of AR, cost and average revenue per admitted request, it is found to be as competitive as MuL-Ag, yet, achieving up to a 44.1% improvement in terms of execution time.

Experiment 2, whose results are shown in Fig. 4.3 analyses the impact of the request size, by varying the number of VNFs from 3 to 13 considering 10 InPs and a demand size of 100 requests. From Fig. 4.3a, the average mapping cost per admitted request for all the 3 algorithms tends to increase as the number of VNFs per SFC increases. This is something expected since SFCs with more VNFs are associated with a higher consumption of both node and link resources, resulting in a higher provisioning cost. However, like in experiment 1, $DistNSE$ results in the worst performance in terms of cost, with an average value of 9.97$, which is 33% higher than MuL-Ag, whose average value is 6.67$, and 44.7% higher than MuL, whose average cost value is 5.52$. The poor performance of $DistNSE$ in terms of mapping cost is largely attributed to the inability of the different InPs



(a) Average acceptance ratio

(b) Average provisioning cost per admitted SFC

(c) Average processing time per admitted SFC

(d) Total revenue

Figure 4.4: Experiment 3: Impact of substrate network size for offline scenario.

along the different paths to compete for all the sub-SFCs that they could potentially map, as they only compete for the previously mapped sub-SFC. The results in Fig. 4.3b demonstrate the superior performance of the MuL compared to MuL-Alg in terms of average execution time per admitted request. In general, the average execution time per admitted SFC grows with the increase in the number of VNFs per SFC for the 3 algorithms. This is expected since each additional VNF (and hence, virtual link) comes with an extra processing time of any intra-domain mapping. However, as observed, the time complexity of MuL-Ag tends to grow exponentially when increasing the SFC size, resulting in an average value of 17.03 seconds, which is 86.3% worse than the MuL, whose value is 2.33 seconds, averaged across all SFC sizes. This is attributed to the fact that, as the number of VNFs increases, the number of stages of the multi-stage graph increases. As a result, the number

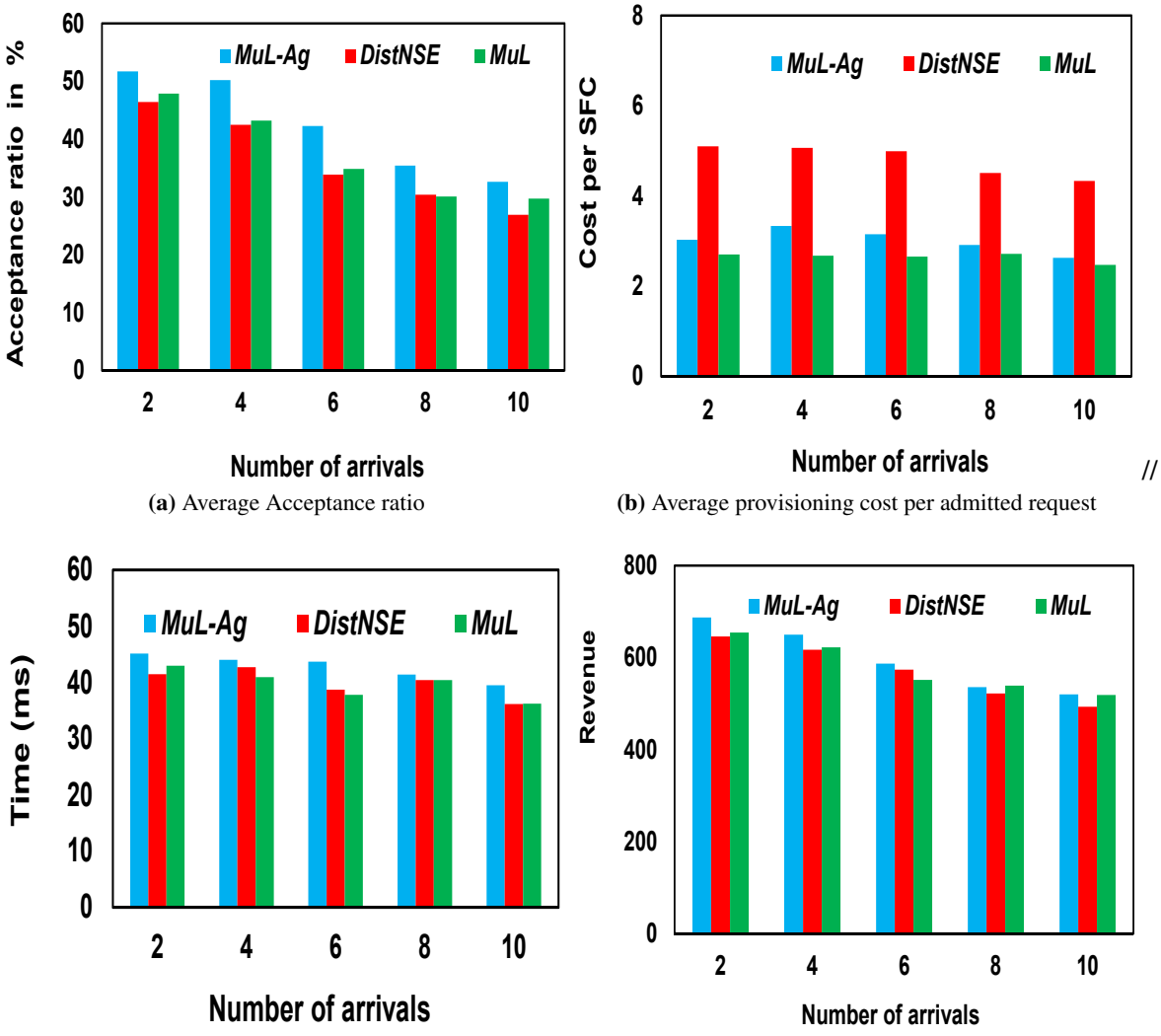
of messages received by each node increases drastically for the MuL-Ag algorithm, especially for the nodes at the rightmost stages, this increases the computational load at these nodes, resulting into extremely high execution times. On the other hand, for the MuL algorithm, each node forwards only a single message block to each node at the next stage. Therefore, the number of messages received by a given node at a given stage is only dependent on the number of pushing nodes in the preceding stage, and not on the stage depth of the node. The DistNSE algorithm results in a 83.4% increase in terms of execution time compared to MuL, with an average value of 14.17 seconds.

In Experiment 3 whose results are shown in Fig. 4.4, the impact of the substrate network size is analyzed by varying the number of InPs from 4 to 12. The 3 algorithms have a close performance in terms of AR (an approx. 4% difference) with average values of: 37.0%, 33.0% and 35.8% for the MuL-Ag, $DistNSE$ and MuL algorithms respectively. Moreover, the AR performance of the algorithms slightly improves as the number of InPs increases due to an increase in the amount of available resources. The results in Fig. 4.4b show that the average mapping cost per admitted SFC for the 3 algorithms tends to increase as the number of InPs increases. This is attributed to the fact that, increasing the number of InPs, increases the prospects of admitting requests with more VNFs and resource requirements, which are associated with higher costs. Moreover, the probability of traversing multiple inter-domain paths between ingress and egress nodes increases as the number of InPs increases. However, this figure also reveals that MuL results in a 52.5% improvement in terms of average mapping cost per admitted request compared to $DistNSE$, with an average value of 2.54$ compared to 5.35$ from $DistNSE$. The MuL-Ag results in an average value of 2.70$, representing a 5.6% difference with respect to MuL. Moreover, in terms of execution time, MuL results in a significant gain, especially as the number of InPs increases, with an average value of 0.69 seconds, averaged across the different number of InPs, as shown in figure 4.4c. This translates into a performance improvement of up to 15.9% and 98.0% compared to MuL-Ag and $DistNSE$, respectively, whose average processing times per admitted request are: 0.83 seconds and 34.4 seconds, respectively. The exponential growth in execution time of the $DistNSE$ algorithm results from the path computation step of the $DistNSE$, which requires computing all paths from source to destination, which tends to grow fast as the number of InPs increases. In a similar way, as the number of InPs increases, the number of candidate InPs (hence, nodes at each stage of the multi-stage graph) increases. This increases the number of aggregated messages that are forwarded between the different nodes of the multi-stage graph, hence, affecting the computational complexity of the MuL-Ag algorithm, since, each node forwards all feasible message blocks to its receiving nodes under this approach. The total revenue from the three algorithms is almost the same across the different substrate network sizes, as shown in Fig. 4.4d.

The results from the above offline experiments have demonstrated that the MuL algorithm is only 3% inferior compared to $DistNSE$ and MuL-Ag considering the worst case scenario across all applied metrics, yet, resulting in up to 86.3% and 98.0% improvements in terms of execution time with respect to MuL-Ag and $DistNSE$, respectively, in some cases. Moreover, all the experiments revealed that the MuL algorithm executes in linear time. Finally, the DistNSE algorithm results in

more than a 44.7% increase in terms of provisioning cost per admitted request compared to MuL for all considered experiments.

Online scenario



(a) Average Acceptance ratio

(b) Average provisioning cost per admitted request

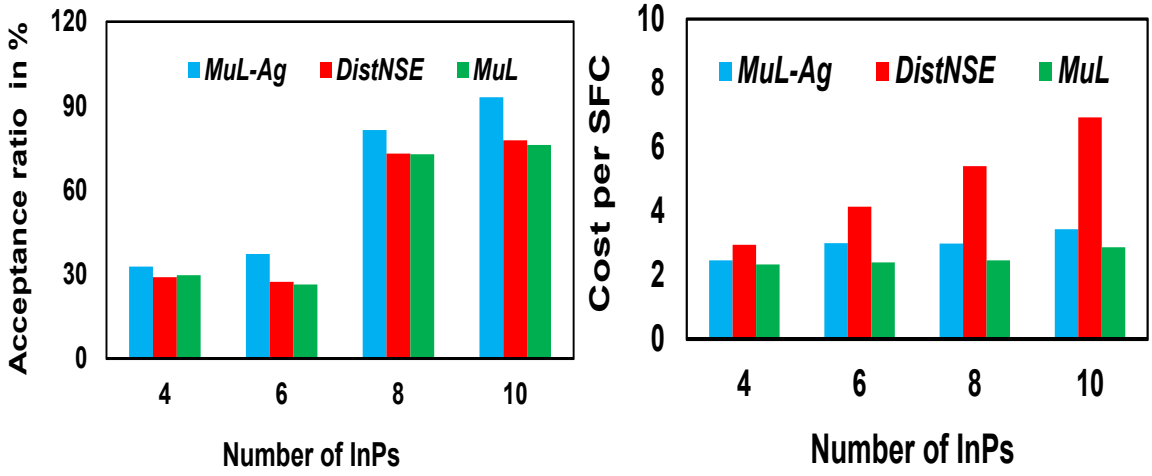
(c) Average processing time per admitted SFC

(d) Average revenue per admitted request

Figure 4.5: Experiment 4: Impact of arrival rate for online scenario

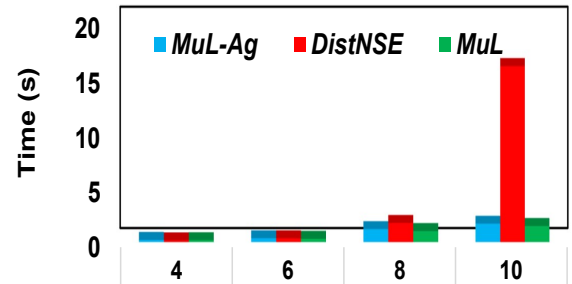
In this section we analyze the results obtained from the experiments conducted while considering online requests. The results of the different experiments for this scenario are discussed below:

Experiment 4, whose results are shown in Fig. 4.5 analyses the impact of the arrival rate of the requests considering 7 InPs for a total of 10000 time units. The AR performance results shown in Fig. 4.5a reveal that the AR for all algorithms decreases when increasing the arrival rate. This is expected since increasing the arrival rate results in an earlier exhaustion of the available resources, leading to an increase in the request rejection rate. Moreover, $DistNSE$ and MuL have shown to



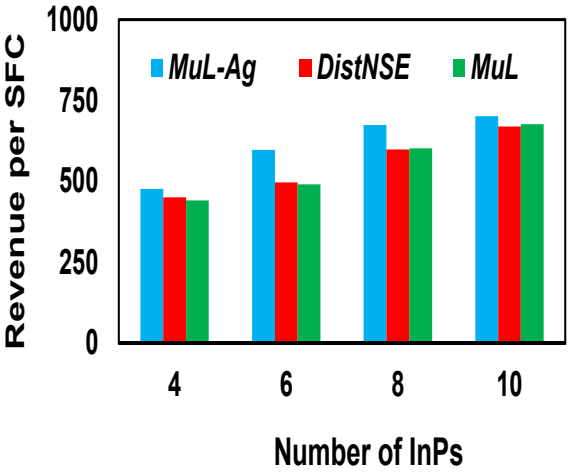
(a) Average Acceptance ratio

(b) Average provisioning cost per admitted request



	4	6	8	10
MuL-Ag	0.19	0.34	1.18	1.67
DistNSE	0.15	0.34	1.74	15.92
MuL	0.17	0.30	1.00	1.46

(c) Average processing time per admitted SFC



(d) Average revenue per admitted request

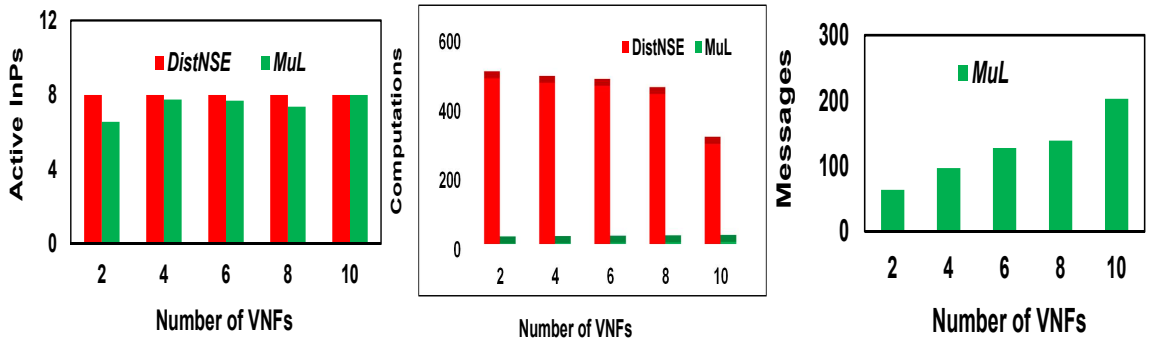
Figure 4.6: Experiment 5: Impact of substrate network size considering online scenario

have the same competitiveness (i.e., within less than a 1% difference) in terms of AR, with average values of: 36.03% and 37.2% for $DistNSE$ and MuL respectively, averaged across all arrival rates. MuL-Ag results in a higher performance with an average value of 42.45% (a 5.3% improvement over MuL) , due to the fact that it forwards all possible messages, increasing chances of finding better solutions, albeit at the cost of higher run times. In terms of average cost per accepted SFC, as shown in Fig. 4.5b, the average values of $DistNSE$, MuL-Ag and MuL are: 4.80$, 3.01$ and 2.64$, respectively. Therefore, MuL results in a 44.9 % improvement in terms of mapping cost compared to $DistNSE$, and a 12.2% improvement compared to MuL-Ag. Moreover, all the algorithms execute in polynomial time for this scenario, with each algorithm executing even in a fraction of a second, with average values of: 42.76 milliseconds, 39.89 milliseconds and 39.67 milliseconds, for the MuL-Ag, $DistNSE$ and MuL, respectively, as reflected in Fig. 4.5c. The

$DistNSE$ algorithm is able to achieve this performance because this experiment uses 7 InPs, which is a relatively small number of InPs. For all the algorithms, the average processing time per admitted request tends to decrease with an increase in the arrival rate. This is due to the fact that, as the arrival rate increases, the number of feasible links and nodes with enough resources decreases, resulting in a decreasing number of paths to be considered for the solution computation. From the results of Fig. 4.5d, the average revenue per admitted request decreases as the arrival rate increases. This is expected since, as the rate increases, the available resources decrease, hence, the prospects of admitting requests returning a high revenue (i.e., usually those with a high number of VNFs and a high resource demand specification) decreases, hence, affecting the average revenue per admitted request. The average revenues per admitted request, averaged across the different arrival rates, for the different algorithms, are: 595.8650141$, 570.1003804$ and 577.0008115$, for MuL-Ag, $DistNSE$ and MuL, respectively. Therefore, MuL behaviour is inferior to MuL-Ag for less than 4%, and within a 1% margin with respect to $DistNSE$, in terms of revenue per admitted SFC.

In Experiment 5, whose results are shown in Fig. 4.6, the impact of the substrate network size on the algorithms' performance is analyzed considering an arrival rate of 5 requests for each 100 time units for a total of 10000 time units. MuL and $DistNSE$ result in similar performance in terms of average AR, with average values of: 51.25% and 51.81% respectively. MuL-Ag results in a 7% improvement in terms of AR with an average value of 58.64%. Moreover, the AR performance of all the algorithms is shown to increase when increasing the number of InPs. This is expected since increasing the number of InPs results in an increase in both node and link resources, hence, improving the AR performance. For the considered number of InPs, the average execution times in seconds per admitted request, for the three algorithms, are: 0.84, 4.54, and 0.73, for the MuL-Ag, $DistNSE$ and MuL algorithms, respectively, averaged over all InP values as reflected in Fig. 4.6c. This result reveals that MuL results in a 13.6% and a 83.9% improvement compared to MuL-Ag and MuL, respectively. Moreover, the execution time for all the algorithms increases when increasing the number of InPs. This is expected since this leads to an increased number of paths from source to destination for the $DistNSE$ algorithm, and an increase in the number of nodes at each stage of the multi-stage graph of the MuL and MuL-Ag algorithms. From Fig. 4.6b, the average mapping cost per admitted request for all the algorithms tends to increase with the number of InPs. This is attributed to the fact that, increasing the number of InPs, increases the prospects of admitting requests with more VNFs and resource requirements, which are associated with higher costs. This is evident in Fig. 4.6d where the average revenue per admitted request increases with increase in substrate size. In this scenario, MuL results in a 48.0% and 15,2% improvement in terms of average mapping cost compared to $DistNSE$ and MuL-Ag, respectively: 2.98$, 4.86$ and 2.52$, for MuL-Ag, $DistNSE$ and MuL, respectively. The results of the average revenue per accepted VNR are: 612.4$, 554.0$ and 552.7$ for MuL-Ag, $DistNSE$ and MuL, respectively, revealing a close performance (within less than a 10% difference) among the three algorithms in terms of this metric. The average revenue per admitted request among all the algorithms increases when increasing the number of InPs. This is expected, since, with an increased availability of node and link resources, the different algorithms are able to map SFCs with a higher number of VNFs, hence, producing a greater revenue.

The results from both online and offline experiments reveal that the proposed algorithm performance is optimized in terms of acceptance ratio, execution time and embedding cost. Moreover, the simulation results further reveal that the strategy of, for each InP node in the graph, processing the received message blocks to only forward the least cost message block, significantly reduces the execution time of the algorithm without degrading its performance.



(a) No. of participating InPs **(b)** No. of computations per active InP **(c)** N0. of messages

Figure 4.7: Experiment 6: Message exchange performance with increase in VNF size.

Analyzing the computation Overhead

In general, distributed algorithms have an inherent drawback of high signalling overhead in terms of messages exchanged between participating nodes, especially with increasing network and request sizes. In Fig. 4.7, we evaluate the performance of the proposed MuL algorithm against DistNSE in terms of the number of nodes/InPs that participate in the computation of the provisioning solution for each request and the number of messages received by each node for to make a computation. The experiment considers 8 InPs with the inter-InP connection probability set to 0.3. In order to evaluate the message exchange overhead involved in the proposed MuL algorithm, we denote by C_n^v as the number of candidate InPs for the VNF corresponding to stage v of the multistage graph, and denote by C_n^{v+1} as the number of candidate nodes for the stage $v+1$. Since each node in a given stage of the multi-stage graph forwards a single message to each node of the following stage of the graph, the number of messages forwarded from stage v to stage $v+1$ of the graph is evaluated as follows:

$$Msg_v^{v+1} = C_n^v \times C_n^{v+1} \tag{4.6}$$

In this way, the total number of messages exchanged throughout the graph is evaluated as follows:

$$Msg_{tot} = \sum_{v=1}^{v=V-1} C_n^v \times C_n^{v+1} \tag{4.7}$$

where $|V|$ is the total number of stages in the multi-stage graph, including those corresponding to the ingress and egress nodes. If we denote by β^v as the probability that a given InP $k \in \mathbb{K}$ is a candidate node for the VNF corresponding to stage v, then, C_n^v can be approximated as $\beta^v \times K$, where K is the

total number of InPs. Therefore, from Eqn. 4.7, the number of messages involved in the distributed computation of the provisioning solution is increased as the number of VNFs of the request increases, since this results in an increase in the number of stages of the multi-stage graph, as shown in Fig. 4.7c. Additionally, as the number of substrate nodes in the network increases, the number of possible candidates for each VNF increases, further increasing the number of messages. Therefore, by limiting the number of nodes at any stage of the graph, the total number of messages can be reduced, which is the motivation behind the candidate extraction step, which targets to consider only feasible candidates to participate in the solution computation.

From Fig. 4.7a, on average, the number of InPs participating in the solution computation are 7.3 and 8 (all InPs) for MuL and $DistNSE$, corresponding to an 8.2% improvement of MuL over $DistNSE$. Moreover, from Fig. 4.7b, each participating InP receives 5 and 106 messages for processing for MuL and $DistNSE$ respectively which is approximately 95% overhead for $DistNSE$. This performance is attributed to the fact that $DistNSE$ relies on computing paths between ingress and egress nodes using an abstracted topology of peering nodes. In this way, it is possible for a given InP to be part of multiple paths, hence participating in the computations of those paths. Moreover, even nodes that are not feasible candidates for the solution receive the sub-SFC for intra-domain provisioning evaluation during the solution computation, as long as they are part of a potential solution path.

4.5 Conclusion

This chapter has proposed a multi-stage graph aided algorithm for provisioning SFCs across multiple domains in a distributed fashion, while considering a limited disclosure of information from the involved InPs. In this way, the privacy requirements of the different InPs is respected. The multi-stage graph is constructed from a pre-computed set of InPs obtained by a candidate search technique which enhances the run-time complexity of the algorithm thanks to reducing the set of InPs involved in the solution computation. Moreover, the simulation results have revealed that the proposed algorithm can result in up to a 7.9 % improvement in terms of acceptance ratio, while spending a shorter execution time, in comparison with a state-of-the-art benchmark algorithm. Moreover, the algorithm results in up to 8.2% and 95% reduction in terms of the number of InPs participating in the solution computation and the average number of messages processed by each participating node respectively, compared to the benchmark state-of-art distributed algorithm. Moreover, considering different offline and online experiments, the proposed distributed computation algorithm has been found to be scalable when increasing both the substrate network size and the request demand, rendering it well suited for large scale networks.

PART III: Intra-domain Service Orchestration Algorithms

CHAPTER 5

Survivable Service Orchestration with backup resource Sharing

5.1 Introduction

The growing interest in the virtualization paradigm is based on two main premises: First, NFV is expected to result in a reduction in Capital and Operational expenditures incurred by service providers by enabling a multitude of applications to be provisioned on a shared Infrastructure [8,21,59,118,119]. This will be facilitated by supporting the migration of complex network functions (e.g., Firewalls, Proxies or Load Balancers) from dedicated hardware appliances to general purpose commodity hardware, with those functions being implemented as software modules, and executed inside virtual machines/containers hosted on general commodity servers [6, 69, 120]. Second, NFV, is envisaged to result in an improved Quality of Service (QoS) by granting the users the flexibility to customize their requests according to the specific requirements of the services to be supported [91]. However, realizing the two-fold benefits is non-trivial from two main perspectives:

First, provisioning a myriad of service requests on a shared resource-constrained infrastructure requires novel resource -efficient orchestration algorithms [92, 121–126], that can easily adapt to changes in the request topology, constraint specification and embedding objectives, with minimum modification in the algorithm's execution strategy. Previously, the works in [127–131] adopted an exact solution approach to the service provisioning problem with the goal of realizing optimal embedding solutions. However, such approaches are not well suited for delay sensitive applications due to their high run times. This motivated the adoption of heuristic approaches such as [21, 67, 89, 132–140]. However, most of the previously proposed solutions are not flexible enough to adapt to the heterogeneity of service requirements. In the line of jointly accommodating multiple embedding objectives such as reliability, cost, etc, algorithms based on node ranking emerged as promising candidates in which the substrate node for hosting a virtual node is selected as the one with the highest rank or potential with respect to a given objective [23, 25, 124, 126, 137, 141, 142]. Aside from the fact that such approaches do not coordinate the node and link mapping stages of the problem, the potential of a given substrate node is also influenced by the potential of the neighboring links and nodes. As a result, neighboring nodes and links that can never be part of the final solution due to other constraints such as location or residual resources may also directly influence the rank of a given

node. This will translate into a poor mapping solution if such nodes have high influence.

Secondly, NFV introduces additional concerns regarding service survivability, due to the inherent potential causes of failure that exist in a software approach, including software faults or misconfigurations, among other reasons [29, 39, 143]. Two main approaches are adopted in literature for achieving service survivability: $i)$ pro-actively provisioning and reserving backup resources for each VNF and virtual link of a given request at the mapping stage [43, 67]; or $ii)$ adopting an intermediate approach such as: reactively provisioning restoration resources upon failure [42, 68, 69]; provisioning back-up resources for only critical functions of the SFC [38, 70]; selecting the most reliable nodes and links for hosting the SFC [73]; or transforming the request topology into a form that enhances its survivability [6]. Although the second main approach results in better resource utilization, it cannot guarantee the availability of resources for service restoration upon failure. Moreover, the additional time for provisioning new resources may result in unacceptable levels of service disruption for mission critical services, potentially resulting in fatal damages. Aware of the criticality of some future services [1], this chapter argues for the first approach, in which each request is pro-actively provisioned with backup resources at the deployment stage. Moreover, the heterogeneity of future services in terms of criticality levels and priorities [144] will be determinant for their proper deployment and maintenance. For instance, the Ultra reliable and low latency communications (uRLLC) group of services will be characterized by a low latency and a ultra high reliability requirement, whereas the massive machine-type communication group of services will be constrained by its energy efficiency and the support for a high-dense connectivity, as depicted in figure 5.1.

With this motivation, this chapter envisions a practical scenario in which a service provider allocates resources for requests belonging to two service groups/priorities: the critical/high priority service group, such as the uRLLC, in which the survivability of a service must be guaranteed through backup resources, and the non-critical service group, such as the enhanced mobile broadband (eMBB) group, in which the services can tolerate a service disruption to a higher level. However, note that the approach proposed in the chapter can be applied to any number of service categories with different priorities regarding their access to resources. Moreover, even within the same service group, it will be possible to associate different criticality levels and resource access priorities to different users, depending on the business model and the SLA specification. In this regard, we pro-actively provision backup resources for high priority requests at the SFC deployment stage, with the possibility of sharing these resources with low priority users when they are unused. However, since such resources may be reclaimed by high priority users, the thesis proposes a migration-aware algorithm for the deployment of low priority users that will minimize the average number of preemptions, with the goal of minimizing the level of service disruption experienced by low priority users. Complementary, we propose a QoS-aware algorithm for the remapping of preempted low priority users. This remapping algorithm aims to reuse as much as possible the surviving nodes and paths, since in practice, this minimizes: the service restoration time associated with VNF loading and state transfer onto new virtual machines; and the cost associated with new VNF instantiations, traffic migrations and state transfers to the new host nodes. In light of the above, the main contribution of this chapter can be

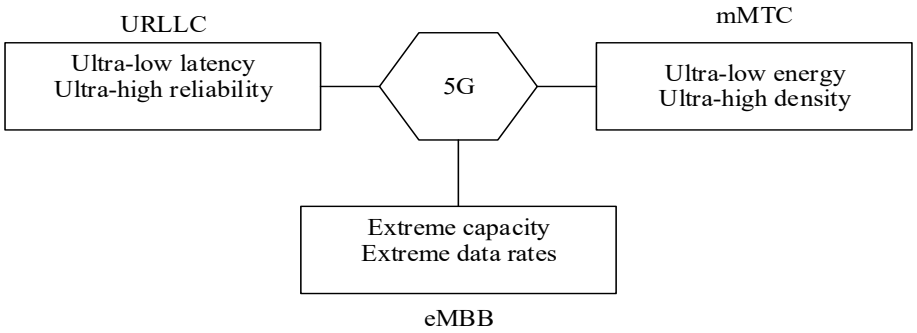


Figure 5.1: An illustration of the main service categories envisioned in 5G and beyond and their key performance requirements

summarized as follows:

1. A multi-stage graph based SFC deployment algorithm (ML-SFCDA), which is adaptable to different mapping objectives. The simulation results show that the proposed algorithm yields a near-optimal solution with tolerable execution time.
2. A migration-aware algorithm based on ML-SFCDA, which allows low priority requests to borrow the unused backup resources of high priority SFCs, with the goal of increasing the resource utilization efficiency, while minimizing the level of service interruption due to the preemption of the borrowed resources from the low priority users.
3. A quality-of-service-aware (QoS-aware) service restoration algorithm for remapping the low priority users subject to preempted resources. The algorithm results in a reduction in the number of surviving VNFs that are migrated to new nodes, which not only reduces the migration delay and cost, but also the cost related to new VNF instantiations, as in practice, such migrations may involve activating new virtual machines and servers, resulting in an increase in power consumption.
4. Numerous simulations to evaluate the performance of the proposed algorithms against benchmark algorithms under different working scenarios.

The rest of this chapter is organized as follows: Section 5.2 introduces a description of the intra-domain survivable service orchestration problem with backup resource sharing. Section 5.3 presents the generic intra-domain multi-stage graph based SFC orchestration algorithm and the mapping algorithms for the two service groups, namely, the high and low priority SFCs. In Section 5.3.4, the proposed quality-of-service-aware service restoration algorithm for remapping the preempted low priority users is presented. The performance evaluation of the proposed algorithms is presented in Section 5.4; finally, this chapter is concluded in Section 5.5.

5.2 Problem description

This chapter considers service requests of two different priorities, hence, with different SLA guarantees and QoS specifications. We exploit this heterogeneity with the proposal of two algorithms for orchestrating the requests, one for each priority group and with different provisioning objectives. The orchestration problem then involves associating each VNF and virtual link of a service request belonging to a given priority group with a feasible substrate node and path in such a manner that optimizes a given objective of the service provider. The orchestration objectives of the different service groups are presented below:

5.2.1 High priority service requests

The high priority/critical requests are pro-actively provisioned with backup resources, to which traffic will be rerouted upon failure of the primary resources. Therefore, the provisioning problem for these involves obtaining a pair of feasible node disjoint paths between the requested source and destination nodes. The least cost path is chosen as the primary path and the second as the backup path. Therefore, the SFC orchestration problem for such requests is modeled with the objective of minimizing the amount of resources used to provision both the primary and the backup paths. Mathematically, this is formulated as a provisioning cost minimization problem as:

$$Minimize \quad \textbf{C(G\_v)} \tag{5.1}$$

where **C(G_v)** in equation 5.1 denotes the average provisioning cost for each accepted request belonging to the high priority service group. This cost is related to the amount of node and link resources (primary and backup) allocated to any request. Let us denote by $\sigma^{e}_{l_{uv}} \in \{0,1\}$ a binary variable equal to 1 if virtual link l_{uv} of SFC request $r \in \mathbb{R}_C$, is provisioned on substrate edge $e \in E_s$, zero otherwise,where $\mathbb{R}_C$ denotes a set of all high priority requests. Likewise, let $y^{n^p_v}_{n_s} \in \{0,1\}$ denote a binary variable equal to 1 if VNF n^p_v of SFC request $r \in \mathbb{R}_C$ is provisioned on substrate node n_s, zero otherwise. Then, the cost incurred for provisioning a request $r \in R_c$ can be expressed as:

$$C(G_v)^r = \sum_{n_v \in N_v} x^{n_s}_{n_v,r} \varrho^{n^k_s}_{q} dem^{n^p_v}_{q} + \sum_{ij \in E_v} \sum_{e \in E_s} x^{e}_{ij,r} \gamma^{e}_{bw} dem^{ij}_{bw} \tag{5.2}$$

The first and second terms in equation 5.2 relate to the VNF and virtual link provisioning costs, respectively. The terms $dem^{n_v}_{cpu}$ and dem^{ij}_{bw} denote the required cpu resource by the VNF n_v and the required bandwidth resource by the virtual link ij. The terms $\gamma^{n_s}_{c}$ and γ^{e}_{bw} denote the cost per unit of cpu and bandwidth resource consumed on substrate node n_s and edge e, respectively. Note that in practice, such costs may be different across different nodes and links depending on the available resources. However, in this chapter, we consider such costs to be uniform and fixed across the different nodes and links. Hence, the provisioning cost is affected by the number of substrate links

assigned to a given request. The average mapping cost per accepted request can be evaluated as:

$$\textbf{C(G\_v)} = \frac{\sum_{r \in R_c} C(G_v)^r}{|R_c^q|} \tag{5.3}$$

where R_c^q is the set of all accepted high priority requests and $|R_c^q|$ is the cardinality of this set. The thesis opted for a linear cost model as shown in equation 5.2, since such a model has been justified through numerical analysis [145].

5.2.2 Low priority service requests objective

Whenever a high priority request is migrated to its backup path, any low priority request currently utilizing part of those resources will be preempted from those resources. This may need additional time to find alternative link and node resources and possibly to instantiate new VNFs to support the preempted traffic, which may result in some service degradation. Therefore, the low priority requests will be deployed with the goal of minimizing the average number of preemptions experienced by those requests. Considering a given SFC request $r \in R_{nc}$, let the binary variable $f_{uv,r}^{p_s^{qm},t} \in \{0,1\}$ be equal to 1 if virtual link l_{uv} is preempted from the substrate path $p_s^{qm} \in P^{qm}$ at time t, zero otherwise, where VNFs u and v are respectively provisioned on substrate nodes q and m. Also let $y_{n_v,r}^{n_s,t} \in \{0,1\}$ denote a binary variable equal to 1 if virtual node n_v is preempted from substrate node $n_s \in N_v$ at time t, zero otherwise. The amount of service interruptions experienced by that request $r \in R_{nc}$ for the duration of its life-time $\tau^d \leq T$ can be computed as the number of VNF and virtual link preemptions that have occurred, and this can be quantified as:

$$P_{rempt}^r = \sum_{t \in T} \sum_{n_v \in N_v} y_{n_v,r}^{n_s,t} + \sum_{t \in T} \sum_{u,v \in N_v} f_{uv,r}^{p_s^{qm},t} \;\; \forall q, m, n_s \in N_s \tag{5.4}$$

The goal of a migration-aware algorithm, therefore, can be defined as:

$$Minimise \; P_{rempt}^{avg} \tag{5.5}$$

where P_{rempt}^{avg} denotes the time-average node and link failures due to service preemption among all low priority requests, expressed as:

$$P_{rempt}^{avg} = \frac{\sum_{r \in R_{nc}} P_{rempt}^r}{T} \tag{5.6}$$

While achieving the goals specified in Eqns. 5.1 and 5.5, the orchestration algorithm should adhere to the constraints specified in Eqns. 2.6- 2.13 related to resource, location, flow and domain requirements as introduced in chapter 2.

The problem discussed above is a common ILP problem, whose resolution by means of conventional solvers such as CPLEX and Gurobi is not affordable in terms of execution time because of the NP-hardness nature of such problems. Therefore, the motivation of this chapter is to propose

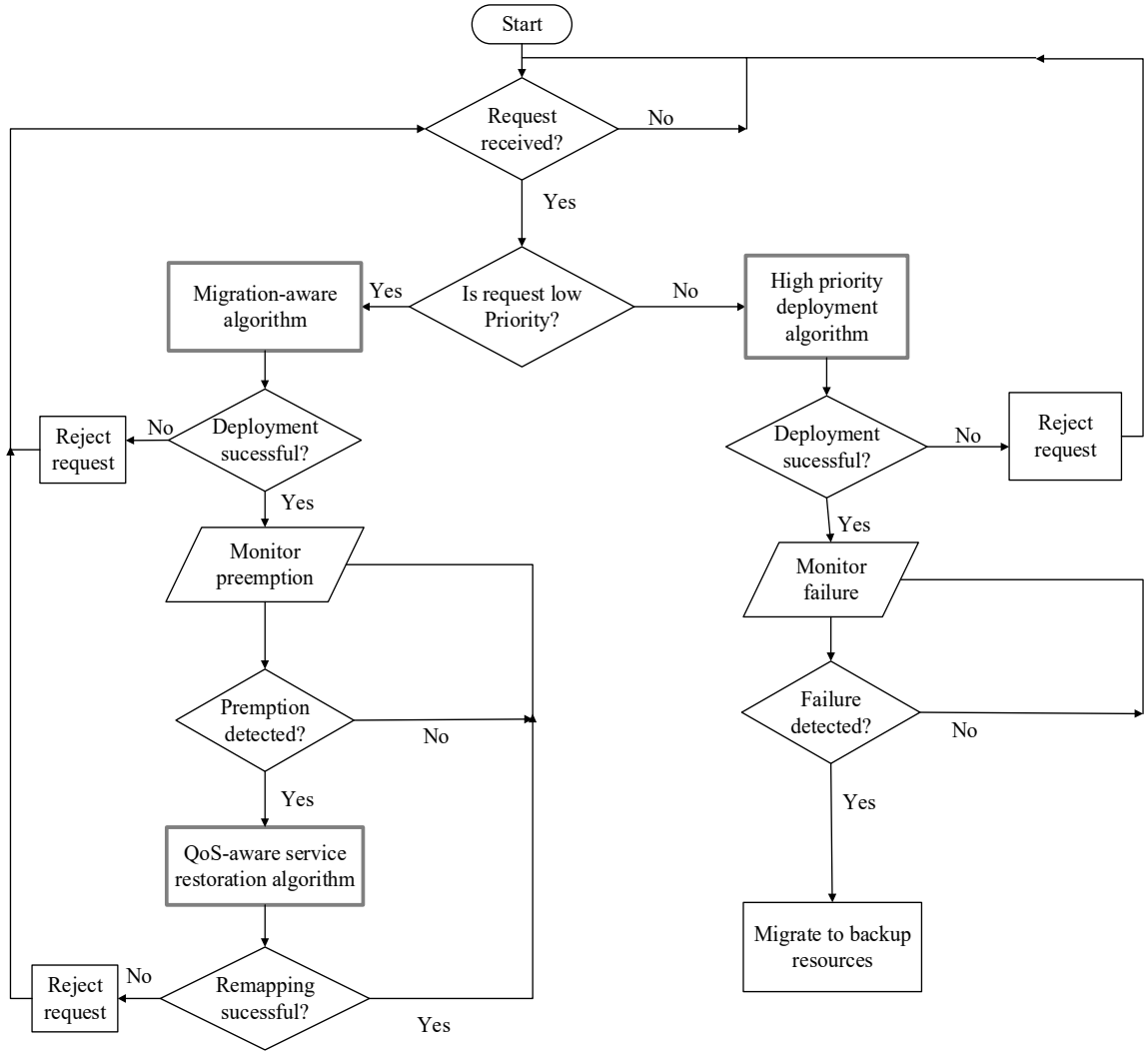


Figure 5.2: A flow chart showing the steps of admitting a given SFC request and the management procedure while in the system. The thick grey boxes correspond to the algorithms for the provisioning of high priority and low priority requests and for remapping preempted requests.

alternative algorithms that are able to obtain close to optimal solutions in a feasible execution time. The flow-chart in Fig. 5.2 illustrates the sequence of steps for accepting the received requests and their management once in the system. The algorithms proposed to handle the different events of the request (i.e., admission and remapping) are indicated with boxes with thick grey lines. Note that the migration-aware and the QoS-aware algorithms use the proposed generic multi-stage SFC deployment algorithm as their base algorithm.

5.3 Proposed SFC deployment algorithms

This section introduces and describes the working mechanism of the multi-stage graph based SFC deployment algorithm, and it will describe how this algorithm is transformed into the migration-aware algorithm for the mapping of low priority SFCs, as well as how it is transformed to the QoS-aware service restoration algorithm for remapping the preempted low priority requests. Additionally, the algorithm for the mapping of the high priority requests is described.

5.3.1 Generic multi-stage SFC deployment algorithm

This section describes the steps involved in the execution of the **M**ulti-**L**ayer graph based **SFC** **D**eployment **A**lgorithm (ML-SFCDA), a generic SFC deployment algorithm that can be tailored to different SFC deployment objectives and topologies. Due to the multiple constraints associated with the VNFs and the corresponding virtual links, each SFC request can only be served by a subset of the underlying substrate nodes and links. Consequently, the first step of the ML-SFCDA algorithm involves extracting a subset of feasible substrate nodes that can potentially satisfy the VNF constraints. The second step then involves using the above subset of nodes to construct a weighted multi-stage graph in which the nodes of each stage are the candidate substrate nodes for each VNF. The weights of the graph are then updated by propagating the initial nodes weights from the first stage to the last stage. Finally, the last step involves traversing the graph backwards by selecting nodes and links that made the least contribution to the last stage, as the nodes and links for provisioning the SFC request. A detailed description of the four main steps involved in the algorithm execution is given below:

Step 1: Candidate evaluation

For each one of the VNFs of the SFC, this step identifies the subset of underlying nodes that are capable of serving that VNF. These are the nodes that satisfy the resource constraints (amount of computational resource and function type), location constraint and QoS constraint (e.g., reliability) of the corresponding VNF. This step therefore associates each VNF i of the SFC request $r \in \mathbb{R}$ with a set $CaS_s^{i,r}$ consisting of the underlying nodes that satisfy the constraints of this VNF, where $\mathbb{R}$ is the set of all SFC requests to be provisioned. Aside from significantly reducing the time for traversing the multi-stage graph due to the reduced number of involved nodes in the graph, the candidate evaluation step guarantees that unfeasible nodes will not be selected for hosting a given VNF in the final mapping solution. A node $n_s \in N_s$ is a candidate for VNF i if it satisfies the following conditions:

$$dist(n_s, i) \leq dev(i) \tag{5.7}$$

$$dem_{cpu}^{n_v} \leq \omega_{cpu}^{n_s} \tag{5.8}$$

$$fn^{n_v} \in fn^{n_s} \tag{5.9}$$

where $dist(n_s, i)$ denotes the distance of the substrate node $n_s \in N_S$ from the preferred location of VNF $i \in N_v$. This distance can refer to a geographical metric, such as a country/region preference,

or to the number of hops between the different VNFs' location. In this work, we have focused on the geographical approach. Specifically, equation 5.7 requires the chosen substrate node to be within the acceptable location radius of the requested VNF. Equation 5.8 is the amount of resource constraint and equation 5.9 is the resource type constraint, which requires that the underlying node should host the resource type required by the VNF.

Algorithm 4 presents the pseudo-code for the candidate evaluation step. For a given SFC request $r \in \mathbb{R}$, the algorithm takes as input the substrate graph G_s and the SFC request graph G_v^r, and outputs a set $Cand_s^r$ consisting of candidate nodes for the different VNFs of the request. The algorithm starts by associating each VNF of the SFC with a set of nodes that meet its cpu, location and function type specifications. In the case that a VNF has no candidate nodes, the request is rejected, and the entire algorithm is terminated at this point. Otherwise, the obtained candidate set for any VNF is stored in the set $Cand_s^r$.

Algorithm 4 Candidate Evaluation Algorithm

Input:G_s,G_v^r
Output: Set of candidate sets for request r, $Cand_s^r$
Initialise: $Cand_s^r = \emptyset$ ▷ Initialise set of candidate sets
for *Each VNF* $i \in N_v^r$ **do**
  $CaS_s^{i,r} = \emptyset$ ▷ Initialise candidate set
  for *For each substrate node* $n_s \in N_s$ **do**
    if $dist(n_s, i) \leq dev(i)$ & $\omega_{cpu}^{n_s} \geq dem_{cpu}^{i}$& $fn^i = fn^{n_s}$ **then**
      Add n_s to $CaS_s^{i,r}$ ▷ n_s is a possible candidate
    end
    if $CaS_s^{i,r} = \emptyset$ **then**
      Reject request break
    end
    else
      Add $CaS_s^{i,r}$ to $Cand_s^r$
    end
  end
end
Return $Cand_s^r$

Step 2: Construction of the multi-stage graph

The multi-stage graph is constructed using the candidate sets of substrate nodes obtained from the candidate evaluation step for each VNF. The ingress and egress nodes are also included in the graph as the first and last stages. Figure 5.3(b) illustrates a multi-stage graph for a SFC consisting of 3 VNFs, in which s_n and t_n denote the ingress and egress nodes as shown in figure 5.3(a). The multi-stage graph for this SFC is composed of 3 intermediate stages, with the nodes at each stage being the candidate substrate nodes for the corresponding VNF. Each candidate substrate node $n_s \in Cand_s^r$ within the multi-stage graph is associated with a local state ϕ_{n_s}, which quantifies its suitability to the

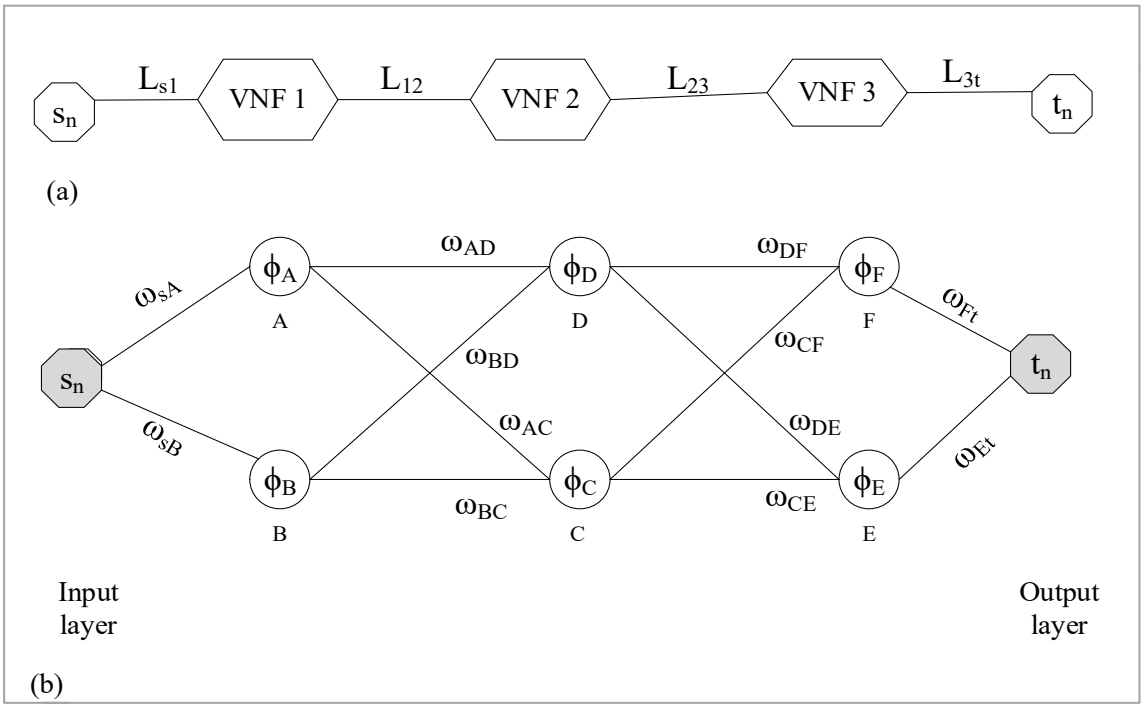


Figure 5.3: A multi-stage graph illustration: (a) An SFC comprised of a single source and destination and 3 VNFs; (b) The corresponding multi-stage graph with the 3 VNFs constituting the hidden stages each with two possible candidate substrate nodes.

objective being optimized. As an example, if the SFC provisioning objective is to balance the load among all the nodes of the network, ϕ_{n_s} is computed as:

$$\phi_{n_s} = 1 - \frac{\omega_{cpu}^{n_s}}{\Omega_{cpu}^{n_s} + \epsilon} \tag{5.10}$$

where $\omega_{cpu}^{n_s}$ are the residual cpu resources at node $n_s \in N_S$ and $\Omega_{cpu}^{n_s}$ is its node capacity. The term ϵ is a small bias added to avoid the cancellation of the local state for the case $\Omega_{cpu}^{n_s}$ =$\omega_{cpu}^{n_s}$. From equation 5.10, substrate nodes with more residual resources are associated with low state values, hence, are preferred in the computation of the SFC orchestration solution, since the solution involves using less weighted paths.

The interconnections between nodes of adjacent stages correspond to the weighted substrate paths between those substrate nodes. The weight parameter ω_{XY} on the path between substrate nodes X and Y could capture attributes such as number of hops along the path $X - Y$, reliability of the path, or cost of the path, among others. Considering the migration-aware algorithm, which is proposed in this chapter, the weight of such a path is related to the probability with which the corresponding virtual link will be preempted when provisioned on that path. For example, the weight ω_{AD} in figure 5.3(b) relates to the probability with which SFC virtual link L_{12} will be preempted if it is provisioned on the substrate path A-D. The computation of the above inter-stage connecting paths

and their corresponding weights is explained in Section 5.3.1 below.

Step 3: Forward propagation of node states

This step aims to update the weights of the nodes constituting the multi-stage graph by associating each node and substrate path with a degree of suitability to host the corresponding VNF and virtual link. Each stage of the forward propagation step establishes a connectivity relationship between a given node and its preceding stage. This enables the pruning of candidate nodes that have no connectivity to any of the nodes of the preceding stage. To understand the execution of this step, we define the following parameters on the multi-stage graph:

• Incoming node set, n^k_{In}: The incoming node set of a given node $n \in N_s$ at stage k of the multi-stage graph denotes the set of all nodes in the preceding stage $(k-1)$ to which the node n has a feasible connection. Any node in such a set is called an incoming node with respect to n. As an example, the incoming node sets for the nodes in the second and third stages of Fig. 5.3(b) are: $A^2_{In} = \{s_n\}$, $B^2_{In} = \{s_n\}$, $D^3_{In} = \{A, B\}$, $C^3_{In} = \{A, B\}$

• Outgoing node set, n^k_{Out}: An outgoing node set with respect to node $n \in N_s$ at stage k refers to all nodes in the stage (k+1) to which the node n has a feasible connection. As an example, nodes A and B are outgoing nodes with respect to the source node s_n

• Global node state, Φ^k_n: The global state Φ^k_n of node n at stage k is a measure of the suitability of that node to host the corresponding VNF. This parameter is evaluated as the product between its local state ϕ_n and the minimum value of the product between the global state Φ^{k-1}_m of each node m in the incoming node set n^k_{In} of n and the weight of the path between m and n. As an example, the global state of node D in the 3rd stage (k=3) is evaluated as : $\Phi^3_D = \phi_D \times min\{\Phi^2_A \times \omega_{AD}, \Phi^2_B \times \omega_{BD}\}$, where Φ^2_A and Φ^2_B denote the global state of substrate nodes A and B respectively. Observe that since the first stage, for the ingress node, has no preceding stage, its local node state is equal to its global state. Therefore $\Phi^1_{s_n}$=ϕ_{s_n} and Φ^2_A=$\Phi^1_{s_n} \times \omega_{sA} \times \phi_A$. In general, if we denote by Φ^{k-1}_m the global state of a node $m \in cand^r_s$ at the $(k-1)^{th}$ stage, then for any node $n \in cand^r_s$ at stage k, Φ^k_n is evaluated as:

$$\Phi_{n_s,k} = \phi_{n_s} \times min\{\Phi_{1,k-1}\ w_{1,n_s}, ...\Phi_{M,k-1}\ w_{M,n_s}\} \tag{5.11}$$

where M is the cardinality of n^k_{In}, the incoming node set for n. If none of the nodes of the preceding stage can reach a node n, then $\Phi^k_n = \infty$ for that node, therefore, such a node cannot host the corresponding VNF. Consequently, if the global state of all the nodes at a given stage is infinite, the VNF corresponding to that stage cannot be provisioned, hence, the entire request is rejected at this point.

• Node Tail, Tl^k_n: The tail of a node n at stage k refers to the node $m \in n^k_{In}$ from the preceding stage that fixed the global state of node n. Specifically, this is the node from the preceding stage whose product of its global state and the path weight towards the receiving node n is the smallest among all the pushing nodes of n.

• Node Trail, $Trail^k_n$: The trail of a node refers to the path constructed recursively from n towards its tail node $m \in n^k_{In}$, then, towards the tail of node m, and so on, until we reach the source node. In

other words, the trail of a node n is the path from the ingress node to reach the node n.

• Outgoing edge-list, E_n^k. The outgoing edge-list $E_n^k \in E_s$ for node n at stage k refers to the list of all feasible edges in the substrate network excluding all edges already used in the trail of node n. This is the set of edges that is then used to compute the possible paths from this node n to its outgoing node set at the next stage.

The forward propagation step therefore involves associating each candidate node in stage k of the multi-stage graph with a node attribute tuple $< E_n^k, \Phi_n^k, Tl_n^k, Trail_n^k >$ where E_n^k, Φ_n^k, Tl_n^k and $Trail_n^k$ respectively denote: the node edge-list, the node global state, the node tail and the node trail. Starting with the ingress node which constitutes the input stage of the graph, the algorithm progresses, stage by stage, and node by node inside a given stage. At the ingress node, the algorithm starts by constructing a list of all valid edges (e.g., those with enough bandwidth resources) from the substrate graph. This becomes the outgoing node edge-list $E_{s_n}^1$ for the ingress node s_n. Then, each of these edges is associated with a weight computed according to the objective being optimized. As an example, if the objective is to achieve the load balancing, the edge weight is computed as:

$$\phi_{e_s} = 1 - \frac{\omega_{bw}^e}{\Omega_{bw}^e + \epsilon} \tag{5.12}$$

where Ω_{bw}^e and ω_{bw}^e denote the bandwidth capacity of the edge $e \in E_s$ and its residual bandwidth resources. Therefore, if the residual bandwidth of an edge is close to its capacity, the weight will be close to zero, hence, more likely to be selected, since the algorithm prefers selecting nodes and edges with the least weight. Using this edge-list, the algorithm obtains the path between the ingress node and each of the outgoing nodes m in the next stage (stage 2) using the Dijkstra algorithm. Note that the Dijkstra algorithm yields both the least weight path from s_n to m and its associated weight. Then, the algorithm computes the global state of each of the outgoing nodes according to the expression 5.11. Since the input stage has a single node s_n, this node becomes the tail node of all reachable nodes in stage 2. Therefore, the trail of each node in the outgoing stage consists of the substrate path between the source node s_n and the outgoing node m. For each outgoing node m, its resulting node edge-list E_m^2 is obtained by updating $E_{s_n}^1$ after removing all edges that are already used in the trail of node m. Note that this is the edge-list that will be used to compute the path from node m to its receiving nodes in the subsequent propagation step. Therefore, removing already used edges guarantees that such edges do not appear in subsequent paths, hence, ensuring that any traffic does not traverse the same link two times. Such an approach eliminates the possibility of multiple virtual links sharing a single edge, which would complicate achieving traffic isolation. Moreover, such a scenario would make local rerouting service restoration approaches impractical, in case of failure of such a shared edge, since in practice such a failure would correspond to multiple virtual link failures. However, in cases where reusing a given edge is desired, then, $E_{s_n}^1$ is updated by updating the available bandwidth resources on the edges used in the trail of node m, which guarantees that only feasible edges are used in the solution computation at subsequent stages. These parameters (i.e., global state, trail, tail and edge-list) will constitute the node attribute tuple for node m. In the next

round of forward propagation, each node m at stage 2 will propagate its global state to each reachable node in the 3rd stage. This consists of first calculating the shortest paths from m to each one of its outgoing nodes. Each shortest path is calculated by means of the Dijkstra algorithm, making use exclusively of edges in the set E_m^2, and their corresponding weights as distances. Then, for each node in the 3rd stage, the corresponding node attribute tuple is evaluated. This forward propagation step continues until the last stage is reached.

Step 4: Solution construction

The final mapping solution is obtained by inspecting the node attribute tuple of the egress node after extracting its trail $Trail_{t_n}^K$, where K is the total number of stages. This trail serves as the request deployment solution, and it is composed of the nodes and links that result in the best mapping solution according to the deployment objective. In particular, preferred links and nodes for embedding the request will be those that resulted in the least global state value at the last stage. Consequently, deploying a service on such a path will be compliant with the mapping objective.

Our proposed approach manifests a number of interesting features: first, during the forward propagation phase, whenever the global state of all candidate nodes at a given stage is infinite, such a request is considered unfeasible, hence, rejected. This ability to detect unfeasible requests early in any mapping stage saves time that would be spent on processing such requests in subsequent stages. In other graph based approaches, such as those adopted in [50] and [22], the SFC deployment solution is computed as follows: starting from the egress node, a path is computed to all nodes that can potentially host the VNF preceding the egress node. Then, the node that results in the least cost path is chosen as the host for this VNF, with the path between this and the egress node being selected for hosting the corresponding virtual link. In the next step, the selected node is taken as a fictitious egress node, and paths are computed to all nodes that can host the VNF preceding that node, in a similar way as it was done before. This process continues until the source node is reached. However, adopting such a greedy approach may result in dead-ends during the solution computation, making necessary a back-tracking mechanism. Different from such greedy schemes, our approach results in the globally best deployment solution.

Time complexity analysis of ML-SFCDA

The key steps of the proposed multi-stage service deployment algorithm to obtain the mapping solution are: the computation of the candidate sets for each VNF and the forward propagation of global states. The time complexity to obtain the candidate nodes for each VNF is linear in terms of the number of substrate nodes N_s, $\Theta(N_s)$. The computation of the solution using the multi-stage graph involves obtaining the shortest path of each incoming node set with its associated node, for all the nodes of each stage, and for all the stages. If we consider the multi-stage graph to consist of K stages, including the source and destination nodes, then, the number of inter-stage path computations/connections is $K-1$. If we consider the number of candidate nodes for each VNF to be N_c (in practice, this may be different for the different VNFs), the total number of

Algorithm 5 Forward node state propagation

Input: Multi-stage graph, G_ρ
Output: Multi-stage graph G_{ρ_s} with global states
for *Each stage* $k \in \{1, 2, 3..K\}$ *in* G_ρ **do**
  if $k < K$ **then**
    for *Each node* $m \in CaS^m_{k+1}$ *at stage* $k+1$ **do**
      for *Each node* $n \in CaS^n_k$ *at k with global state* $\Phi_{n,k}$ **do**
        Propagate $\Phi_{n,k}$ to $m \in CaS^m_{k+1}$ $\forall m \in CaS^m_{k+1}$ Compute resulting global state $\Phi_{m,k+1}$ for node m
      end
    end
    if $\Phi_{m,k+1}$=*0* $\forall\ m \in CaS^m_{k+1}$ **then**
      Reject request
    end
    k=k+1
  end
end

shortest paths computed between the source node and the first VNF, and between the last VNF and the destination node, is $2 \times N_c$. The total number of shortest paths between the inter-VNF stages is $(K-3) \times N_c^2$. Considering the time-complexity of the Dijkstra algorithm as $\Theta(|E_s| + |N_s|log(|N_s|)) \approx \Theta(|N_s|log(|N_s|))$, and neglecting the time complexity of the candidate evaluation step, the time complexity for accepting a given request can be expressed as $\Theta((2N_c + (K-3)N_c^2) \times |N_s|log(|N_s|)) \approx \Theta((K-3)N_c^2) \times |N_s|log(|N_s|))$. In practice, the different substrate nodes can only support a finite number of VNFs, hence, limiting the number of candidates for each VNF. Moreover, due to the finite number of VNFs that can be supported by each substrate node, the number of possible candidates for each VNF decreases as the SFC size increases, binding the time complexity of the algorithm as the SFC size increases.

5.3.2 Migration-aware algorithm

In this section we present the migration-aware algorithm for mapping low priority SFCs onto the underlying infrastructure. The operation and implementation of this algorithm is similar to the general ML-SFCDA algorithm discussed in Section 5.3.1. However, in this case, the node and edge weights are computed with the objective of minimizing future preemptions of resources allocated to low priority users. Therefore, in this section, we focus on how these weights are computed.

After constructing the multi-stage graph, as discussed in Section 5.3.1, each candidate substrate node $n_s \in Cand^r_s$ of the multi-stage graph is associated with a local state ϕ_{n_s} that is related to the degree of likelihood that a VNF provisioned by this node will be preempted by a high priority request.

The local state value of the node is computed as:

$$\phi_{n_s} = 1 - \frac{\Omega^{n_s}_{cpu} - C^{bac}_{n_s}}{\Omega^{n_s}_{cpu} + \epsilon} \tag{5.13}$$

where $\Omega^{n_s}_{cpu}$ and $C^{bac}_{n_s}$ denote the resource capacity of node $n_s \in N_s$ and the amount of resources already allocated to high priority requests as backup resources on this node. The term ϵ is a small bias added to avoid the cancellation of the local state when $C^{bac}_{n_s} = 0$. From equation 5.13, substrate nodes with less resources marked for backup are associated with lower state values, hence, with lower preemption probability. Therefore, such nodes will be preferred in the computation of the SFC orchestration solution. The backup resources at a given substrate node are computed as:

$$C^{bac}_{n_s} = \sum_{r \in R_c} \sum_{n_v \in N_v} \gamma^{n_s,r}_{n_v,t} dem^{n_v}_{cpu} \; \forall n_s \in N_S, \forall t < \tau^d \tag{5.14}$$

where $\gamma^{n_s,r}_{n_v,t} \in \{0,1\}$ is a binary variable equal to 1 if substrate node $n_s \in N_s$ is a backup node for VNF $n_v \in N_v$ belonging to request $r \in R_c$ at time $t \in T$, zero otherwise. R_c denotes the set of all high priority requests.

Starting from the ingress node, which is the first stage of the graph, the algorithm extracts the list of all valid edges (i.e., those with enough bandwidth resources) as $E^1_{s_n}$. Then, each of those edges is associated with a preemption weight computed as follows:

$$\phi_{e_s} = 1 - \frac{\Omega^{e_s}_{bw} - Bw^{bac}_{e_s}}{\Omega^{e_s}_{bw} + \epsilon} \tag{5.15}$$

where $\Omega^{e_s}_{bw}$ and $Bw^{bac}_{e_s}$ denote the bandwidth capacity of the edge $e_s \in E_s$ and the amount of resources marked for backup on this edge. Mathematically, $Bw^{bac}_{e_s}$ is computed as:

$$Bw^{bac}_{e_s} = \sum_{r \in \mathbb{R}} \sum_{b, ij \in E_v} \gamma^{e_s,r}_{ij,t} dem^{ij}_{bw} \;\; \forall e_s \in E_s, t \in T \tag{5.16}$$

where $\gamma^{e_s,r}_{ij,t} \in \{0,1\}$ is a binary variable equal to 1 if substrate edge $e_s \in E_s$ is part of the backup path for virtual link ij.

This is followed by the forward propagation step in which each candidate node in the graph is associated with a node attribute tuple consisting of: the node edge-list, the global state, the node tail and the node trail, as discussed in Section 5.3.1. Upon a successful completion of the forward propagation step, the trail of the egress node is selected as the solution for mapping the SFC. This will be composed of the nodes and paths that are characterized by their low preemption probability. Consequently, a service deployed on such a path will be subject to less disruptions due to any preemption forced by the priority services.
Note that the migration-aware algorithm prefers to place low priority requests on nodes and edges that are associated with less resources reserved as backup for high priority request. In this regard, such an approach could have a negative impact regarding the acceptance of new high priority requests.

Therefore, in cases where high priority requests need to be admitted at the expense of low priority requests, then alternative approaches could be adopted, such as reserving a portion of the total network resources for those high priority users. In this case, such reserved resources could be treated as fictitious backup resources that could be borrowed by the low priority users using our proposed migration-aware algorithm.

5.3.3 High priority deployment algorithm

The critical services are deployed with the objective of minimizing the amount of resources allocated to both: the primary and secondary paths. Although the backup resources can be used by non-critical users, these resources can be reclaimed by the critical users whenever there is a failure on their primary resources. Therefore, both, the primary and backup paths should be provisioned with the minimum resources to maximize the resource utilization. In this work, we assume that, for a critical service deployment to be successful, the algorithm should be able to provision both: the primary and secondary paths of the request. This is a reasonable consideration since this could be part of the SLA specification to guarantee the minimum service reliability. The pseudo code for this algorithm is shown in Algorithm 6.

For a given critical request $r \in R_c$, the algorithm takes as input the underlying substrate graph G_s and the request graph G_v^r, and returns the primary and secondary mapping solutions. The algorithm starts by computing all possible node disjoint paths from the ingress node to the egress node. For small networks, such paths can be computed using a brute force algorithm. For large networks, existing flow-based algorithms can be adopted [146,147]. Then, from all these paths, only the feasible paths are considered and stored in the set $feas_{path}$. If the resultant feasible set is empty, the request will be rejected, otherwise, the feasible set is sorted according to the consumed resources of each path, selecting the least resource consuming path as the primary path and the next least resource consuming path as the secondary path. A path is considered feasible if: $i)$ it satisfies the bandwidth requirements of the SFC; $ii)$ it has no cycles and no repeated edges; $iii)$ has at least one candidate substrate node for each VNF, and these candidate nodes should appear in the same sequence as their respective VNFs.

5.3.4 QoS-aware service restoration algorithm

This paper argues for the possibility of low priority requests to use the unused backup resources allocated to high priority requests, with the aim of maximizing the resource utilization. However, these requests may suffer the preemption from those backup resources whenever they are reclaimed by the high priority users. We consider the preemption of any node or link resource used for the provisioning of a VNF or virtual link as a node or link failure respectively. In order to ensure the service continuity of such a request, alternative nodes and links resources must be identified, and the affected traffic will be efficiently rerouted.

This traffic rerouting can be achieved by two strategies [68]: $i)$ a global rerouting, in which the service restoration from a node or link failure is achieved by remapping the entire request with the aim

Algorithm 6 Critical SFC deployment Algorithm

Input:G_s, G_v^r
Output: primary and secondary mapping solution
$feas_{path}$=∅ ▷ Initialise set of feasible node disjoint paths
Generate set $Paths_{disj}^{st}$ of all node disjoint paths from s to t **for** *Each path in* $Paths_{disj}^{st}$ **do**
    if *path is feasible* **then**
        Add path to $feas_{path}$ **else**
            continue
        end
    end
end
if *len(*$feas_{path}$*)* < 2 **then**
    Reject request **else**
        Sort $feas_{path}$ according to resource consumption Select primary and secondary mapping solution
    end
end
Return mapping solution

of obtaining a failure free path from the ingress to the egress nodes; or $ii)$ by a local rerouting, which aims to locally detour the failed node or link, and then return to the surviving path components. On reusing the surviving nodes and links with service restoration, the local rerouting strategy minimizes the level of service interruption due to traffic migrations and signaling delays. Moreover, in case of a single node or link failure, finding a service restoration solution is faster, compared to the global rerouting approach. However, the local rerouting scheme, being a greedy approach, may not be able to find a feasible restoration solution, even when it is possible to obtain a solution using the global counterpart. Moreover, in the presence of multiple non-consecutive node and link failures, the local rerouting approach is not guaranteed to execute faster than the global approach, since such a solution has to be computed independently for each failed element, resulting in a high execution time. For the same reason, such strategy may lead to a higher resource consumption compared to the global rerouting approach, which is globally optimal.

With the above considerations and motivation, the thesis proposes a quality-of-service-aware (QoS-aware) algorithm based on the global traffic rerouting approach. We call this algorithm QoS-aware because the remapping of the preempted resources is performed with the objective of minimising QoS violations by reusing surviving substrate nodes and links whenever possible. In doing so, the number of virtual functions that are migrated to new resources will be minimised, hence, minimising the delays associated with migration, synchronisation, and traffic rerouting to new resources. Moreover, energy costs that may result from the activation of new virtual machines to support migrated VNFs will be minimized as well. The proposed algorithm adopts a global rerouting strategy to overcome the problems associated with local rerouting (i.e., higher resource consumption, possible service restoration failure and a higher execution time for multiple non-consecutive failures),

while minimizing the level of service interruption due to the migration of the involved VNFs and virtual links. The algorithm is computed using the ML-SFCDA algorithm approach discussed in Section 5.3.1, customized to the way in which the node states and link weights will be computed.

The pseudo-code for this algorithm is shown in Algorithm 7. For each request $r \in R_{nc}$ affected by a preemption failure, the algorithm constructs a multi-stage graph following the approach discussed in Section 5.3.1, with the candidate nodes at each stage being constituted by only failure-free nodes. On each stage of the graph, each candidate node for the VNF corresponding to this stage is assigned a service interruption weight ω_s. The weight ω_s is computed as follows: if this substrate node is the current host of a surviving VNF, then its service interruption weight is computed as $\omega_s = \gamma$, otherwise $\omega_s = 1 - \gamma$, where γ takes a significantly small value, e.g., 10^{-4} or less. This weight reflects the likelihood of experiencing a service interruption (i.e. migration of surviving VNF and/or virtual link). The substrate nodes currently hosting the surviving VNFs are assigned such small weights to increase their probability of being reselected during the computation of the restoration solution.

Considering the example in figure 5.3, assuming that 5.3(a) is the SFC request to be remapped, and 5.3(b) is the resulting multi-stage graph of surviving substrate nodes, the QoS-aware service restoration algorithm is executed as follows: all the node and link resources previously allocated to this request are released. Then, the source node s_n is assigned a node attribute tuple $< E^k_{s_n}, \Phi^k_{s_n}, Tl^k_{s_n}, Trail^k_{s_n} >$ where $E^k_{s_n}$, $\Phi^k_{s_n}$, $Tl^k_{s_n}$ and $Trail^k_{s_n}$ denote: the outgoing node edge-list consisting of failure-free edges with enough bandwidth resources, the node global state, the node tail and the node trail, as defined earlier in Section 5.3.1. The service interruption weight ω_s of this node becomes its global state, since this is the ingress node. Then, each edge in the edge-list $E^k_{s_n}$ is allocated a service interruption weight ω_s as follows: if the edge is part of the substrate path hosting the virtual link L_{s1} (i.e., in general, the virtual link between the VNF corresponding to the current stage and the VNF corresponding to the next stage), the service interruption weight of this edge is set to $\omega_s = \gamma$, otherwise $\omega_s = 1 - \gamma$, where γ is chosen as before. Then, the algorithm executes the forward propagation step until the last stage, as discussed in Section 5.3.1. Finally, the service restoration path is obtained by inspecting the tuple received at the last stage after extracting the trail of this node. The substrate nodes and links constituting this trail are chosen as the hosts for the VNFs and virtual links of the SFC request to be remapped. Observe that, since the previously used edges and nodes are assigned smaller ω_s values, such nodes and edges are more likely to be re-selected in the computation of the remapping solution, hence, increasing the probability of reusing all surviving nodes and links. This minimizes the need to migrate surviving VNFs and virtual links to new resources, which could result otherwise in an increased delay and energy cost. Note that, at each receiving node at stage k, the edge list is updated by: 1) removing all edges contained in the trail between this node and its tail node in the preceding stage (or updating the bandwidth resources in case traffic can traverse the same link multiple times); 2) updating the weight of each edge left in the edge-list as follows: if the edge is part of the substrate path currently supporting virtual link $L_{k,k+1}$, then the service interruption weight of this edge is set to $\omega_s = \gamma$, $\omega_s = 1 - \gamma$ otherwise.

Moreover, the proposed ML-SFCDA algorithm is also well suited for computing a restoration solution for multiple consecutive VNF failures, such as those shown in figure 5.4(c). In this case, the algorithm starts by extracting the p_{node} and s_{node} which denote the surviving nodes preceding and succeeding the failed VNFs respectively, as shown in figure 5.4(c). In this case, p_{node} and s_{node} act as the fictitious ingress and egress nodes. Therefore, the failed VNFs can be viewed as a sub-SFC with p_{node} and s_{node} as source and destination nodes. The intermediate stages of the multi-stage graph of this sub-SFC correspond to the failed VNFs, with their surviving candidate nodes constituting the nodes at those respective stages. Each edge of the substrate network is assigned a unitary weight for the case of a restoration goal focused on obtaining the least cost restoration solution. Whenever a remapping solution cannot be obtained, the entire request is rejected, otherwise, the previously assigned resources are released and the resource matrix is updated according to the new assignment.

Algorithm 7 QoS-aware Global Rerouting Algorithm

Input: Multi-stage graph, G_ρ
Output:Restoration solution
for *Each stage* $k \in \{1, 2, 3..K\}$ *in* G_ρ **do**
  if $k < K$ **then**
    for *Each node* $m \in CaS^m_{k+1}$ *at stage* $k+1$ **do**
      for *Each node* $n \in CaS^n_k$ *at* k *with global state* $\Phi_{n,k}$ **do**
        Prune unfeasible nodes and edges Propagate $\Phi_{n,k}$ to $m \in CaS^m_{k+1}$ $\forall m \in CaS^m_{k+1}$
      end
      Compute resulting global state $\Phi_{m,k+1}$ for node m Obtain the attribute tuple for node m
    end
    if $\Phi_{m,k+1}$=*0* $\forall\ m \in CaS^m_{k+1}$ **then**
      Reject request
    end
    k=k+1
  end
end
Obtain node $n^{opt} \in CaS^n_K$ with minimum $\Phi_{n,K}$ $Trail^{n^{opt},K}$ as the restoration solution

Time complexity analysis

Since the proposed QoS-aware algorithm uses the multi-stage algorithm described in Section 5.3.1 as its base algorithm, the time complexity of this algorithm is approximated as $\Theta((K-3)N_c^2) \times |N_s|log(|N_s|))$. The parameters K, N_c and N_s denote: the total number of stages in the multi-stage graph to be used for computing the restoration solution, the possible number of candidate substrate nodes for each VNF, and the number of substrate nodes in the graph, respectively. As evidenced from the simulation results shown in Section 5.4.3, the overhead in terms of time execution resulting from the execution of this algorithm is negligible compared to a local rerouting algorithm which locally detours the failed nodes/paths. This is due to the fact that, at the remapping step, all unfeasible nodes and edges, including those from which the request has been preempted, are excluded from the multi-stage graph; hence, reducing the number of possible candidate nodes and edges for each VNF. Moreover, each node can host only a finite number of VNFs due to resource constraints, binding the

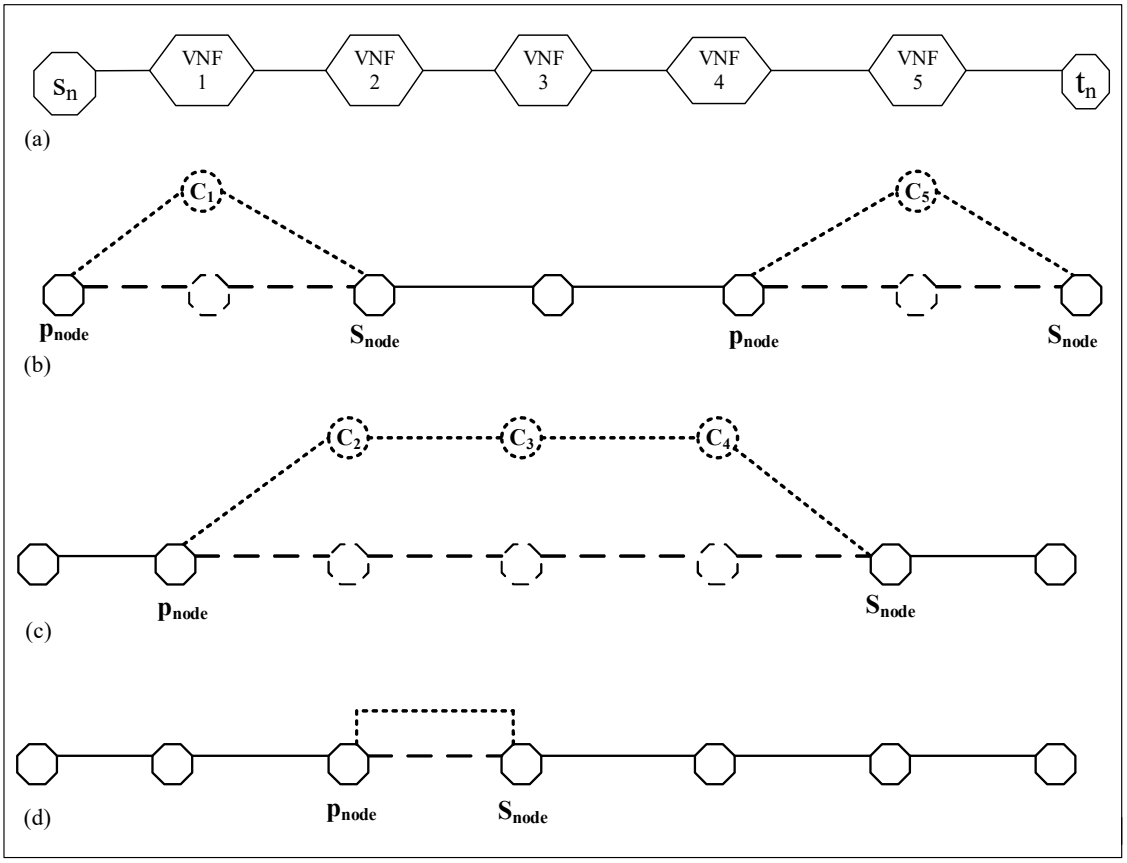


Figure 5.4: An illustration of the localized routing strategy. The failed nodes and links are denoted by dashed lines while the restoration nodes and links are denoted by dotted lines: (a) An SFC comprised of a single source and destination and 5 VNFs; (b) Illustration of non-adjacent multiple node failures in which the nodes hosting VNF 1 and VNF 5 fail. Service restoration is achieved by locally detouring failed nodes by migrating the affected VNFs to candidate nodes C1 and C5 respectively; (c) An illustration of the adjacent multiple node failure and the possible local restoration solution in which each failed VNF is migrated to respective surviving candidate nodes; (d) illustration of single link failure with the failed link migrated to a surviving path

algorithm's time complexity as the number of stages (i.e., SFC size) increases.

5.4 Performance evaluation

This section presents the performance evaluation of the proposed algorithms, including the adopted simulations settings, considered simulation scenarios, and a discussion of the results obtained from the different scenarios.

5.4.1 Simulations settings

The performance evaluation has been made considering both: a real network topology, named ChinaNet, as in [22] and [148], and synthetic network topologies, generated using the Waxman algorithm with the Waxman parameters $\alpha = 0.5$ and $\beta = 0.3$. The ChinaNet topology is made up of 55 nodes and 103 edges, as shown in Fig. 5.5. For both topologies, the delay on each link, the bandwidth resources of each link, and the computing resources of each node, follow a uniform distribution with values specified in Table. 5.1. For the service requests, the node and link resource requirements of each request follow a uniform distribution as adopted in the benchmark work proposed in [22]. The number of VNFs per request, and the end-to-end delay requirements are chosen according to a uniform distribution with the specific values tabulated in Table. 5.1. For the reliability-based scenarios, we consider the reliability of the primary resources to follow a uniform distribution U(0.99, 0.999) as in [149].

Table 5.1: Simulation parameters for the multi-stage graph based distributed algorithm

Substrate Network:	
parameter	Value
Number of nodes for synthetic topologies	12-80
Link delay	unif distrib.[30,130]
Link bandwidth capacity	unif distrib.[50, 100]
Node CPU capacity	unif distrib.[50, 100]
Reliability of primary resources	unif distrib.[0.92, 0.99]
Service Request:	
Parameter	Value
Number of VNFs per request	unif.distrib.[3, 15]
VNF CPU requirements	unif.distrib.[1, 20]
Virtual link Bandwidth requirements	unif.distrib.[1, 20]
Mean arrival rate	2-18 per 100 time units
Arrival distribution	Poisson
Life-time	Exponentially distributed with mean 1000

5.4.2 Simulation scenarios

This section describes the simulation scenarios considered in evaluating the performance of the proposed algorithms while highlighting the rationale behind each one.

Scenario 01: ML-SFCDA performance analysis

This scenario assesses the performance of the ML-SFCDA algorithm as an alternative proposal for the SFC orchestration against an optimal solution based on a brute-force approach, and also against a state-of-the-art Service Function Chain Deployment Optimization (SFCDO) algorithm proposed in [22]. The comparisons are made using a real network topology namely ChinaNet and synthetic network topologies of different sizes. The choice of the SFCDO algorithm for comparison is justified for different reasons: the work is a recent one and, from the results reported by the authors, the algorithm was argued to be near-optimal in terms of execution time, resource consumption and end-to-end delay satisfaction. The SFCDO algorithm arranges the underlying substrate nodes in

Figure 5.5: An illustration of the ChinaNet physical network topology

different levels according to their distance from the source node. Then, starting from the egress node, the algorithm evaluates the paths to the nodes of the previous level. Then, it greedily selects the node that results in the best value of the considered metric (e.g., cost, load balance or delay) from the egress node as the hosting node for the VNF preceding the egress node. This process continues with the selected host node at each stage acting as the fictitious egress node in the next round, until the source node is reached. Different experiments are evaluated in this scenario and the corresponding results are presented in Section 5.4.3. Since ML-SFCDA is the base algorithm for the migration-aware and the QoS-aware service restoration algorithms that are evaluated in other scenarios, in this scenario, we show the performance comparison of ML-SFCDA algorithm including the confidence interval in experiments 1 and 2 in which the arrival rates and substrate network size respectively are varied. In order to obtain stable performance values, we consider 10 trials for each arrival rate/ substrate network size. For each trial, a new set of requests and network topology are generated. The resulting results are shown with a confidence interval of 95%.

Scenario 02: Migration-aware algorithm performance analysis

The chapter has proposed a migration-aware algorithm for the deployment of low priority SFCs, with the aim of minimizing the number of preemptions/failures experienced by such users from their assigned resources. We use this scenario to demonstrate the performance gain of our proposal, which we will denote as Mig-aware. The performance of the Mig-aware algorithm is compared against two benchmark strategies: i) a dedicated backup strategy denoted as Dedicated-BS, in which the high priority requests are provisioned with backup resources that cannot be assigned to low priority users. Hence, such resources remain idle until failure of the associated primary resources; and ii)

a cost-based ML-SFCDA algorithm denoted as ML-SFCDA-Cost, in which the low priority users can use the idle backup resources of high priority users. But, different from the Mig-aware, the low priority users are mapped using the ML-SFCDA algorithm explained in Section 5.3.1 with the objective of minimizing the provisioning cost. Observe that in all the above three strategies the high priority users are provisioned with the same algorithm discussed in Section 5.3.3. The above algorithms are compared considering metrics such as: resource consumption, execution time and number of preemptions experienced by low priority SFCs, among others.

Scenario 03: QoS-aware service restoration algorithm performance analysis

In Section 5.3.4, a QoS-aware service restoration algorithm was proposed for the remapping of preempted low priority requests. The algorithm favors surviving VNFs and virtual links to be reused whenever possible, hence, minimizing delays and costs associated with migrations of such VNFs and virtual links. In this scenario we evaluate the performance of this algorithm, denoted as Global-QoS, against two conventional strategies:
1) a cost-based ML-SFCDA algorithm, in which whenever a low priority request is preempted from any part of the allocated resources, all the resources allocated to this request are released, and the entire request is remapped as a new request using the ML-SFCDA algorithm discussed in Section 5.3.1 with the objective of minimizing the provisioning cost. We denote this algorithm as Global-cost, since it is based on a global rerouting strategy; and 2) a local rerouting based service restoration algorithm, denoted as Local-SR, as proposed in [68]. This Local-SR algorithm locally detours the failed nodes and links to obtain a failure free restoration solution while maintaining any surviving VNF and virtual link onto the currently allocated resources.

The three strategies are compared in terms of several metrics like: the number of successful restoration attempts, the number of service disruptions and the resource consumption, as discussed in Section 5.4.3. Note that in order to have fairness in the analysis of the algorithms, we will use the same algorithm based on cost minimization (ML-SFCDA-Cost) for the initial mapping of the low priority requests. Then, upon failure, we adopt the respective failure recovery algorithm for remapping the failed requests.

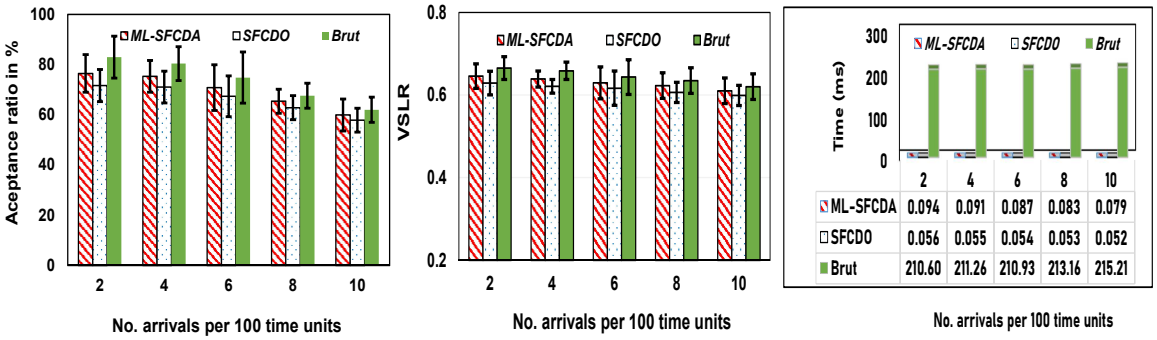
5.4.3 Results and discussion

This section presents the results obtained from the different simulation described above and their corresponding analyses.

Scenario 01: ML-SFCDA performance analysis

This scenario assesses the performance of ML-SFCDA against SFCDO and a brute-force algorithm, denoted by Brut, running different experiments as explained below:

In **experiment 1**, whose results are shown in figure 5.6, the impact of the arrival rate is analyzed considering a network topology of 16 nodes. A small network size has been used for this comparison



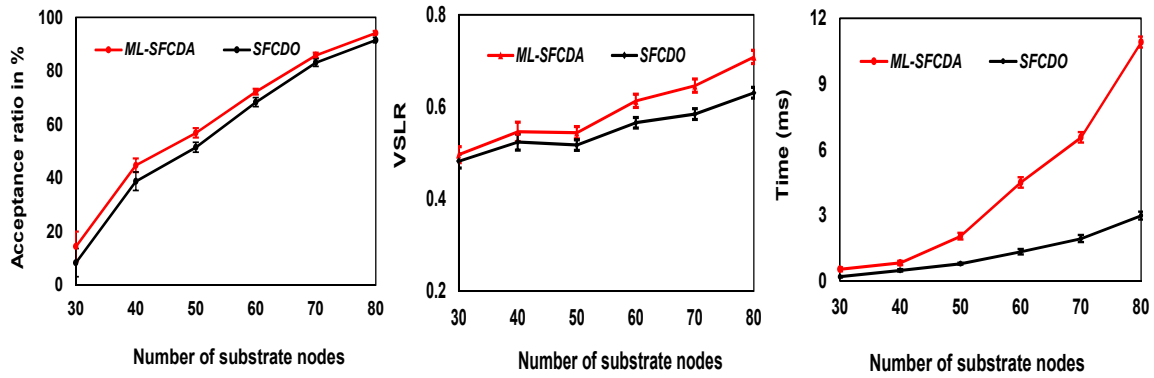
(a) Average acceptance ratio **(b)** Average virtual to substrate link ratio per SFC **(c)** Averege execution time per admitted request

Figure 5.6: Experiment 1 of Scenario 1: The impact of arrival rates analyzed by varying the arrival rates from 2 to 10 for each 100 time units for a total of 5000 time units considering substrate network with 16 nodes.

.

due to the execution time required by Brut for its completion. From figure 5.6a, ML-SFCDA results in an average AR performance of 69.58%, averaged across all arrival rates, which is 3.97% lower than the brute-force algorithm, whose average value is 73.55%, and 3.44% above SFCDO, whose average value is 66.14%. Since the brute-force algorithm enumerates all feasible mappings, it results in a higher AR performance, compared to ML-SFCDA and SFCDO. The better AR performance of ML-SFCDA with respect to SFCDO is due to the fact that SFCDO greedily selects the nodes for hosting the VNFs using only the outcome of the previous step. This may result in a dead-end in any subsequent step, hence, affecting the AR and cost performance. A similar trend is observed in terms of VSLR as reflected in figure 5.6b, with Brut resulting in the highest average VSLR value of 0.64, which is 2.4% and 4.7% superior compared to ML-SFCDA and SFCD0, respectively, whose average values are 0.63 and 0.61. However, figure 5.6c shows that Brut is not feasible in terms of execution time, resulting in up to more than 99% overhead in terms of average execution time for each admitted request, compared to both ML-SFCDA and SFCDO. This is because the solution computation of Brut relies on enumerating all possible path combinations from the ingress to the egress nodes, which is computationally intractable, even for a small network size. For the considered topology, figure 5.6c shows that for such a small network, ML-SFCDA and SFCDO provision each request on average in a fraction of a millisecond (0.09 ms and 0.05 ms respectively), but Brut, on average, requires 3.5 seconds to process each request.

In **experiment 2**, whose results are shown Fig. 5.7, the performance of ML-SFCDA and SFCDO is tested considering medium size networks, with the number of substrate nodes varied from 30 to 80 nodes. From figures 5.7a and 5.7b ML-SFCDA results in up to 4.5% and 7% improvements in terms of AR and VSLR, respectively, with average values of 61.4% and 0.6% compared to SFCDO, whose average values for the same metrics are 56.9% and 0.55%, averaged across all substrate nodes. Moroever, from figure 5.7c, both algorithms demonstrate an increment in terms of average time for



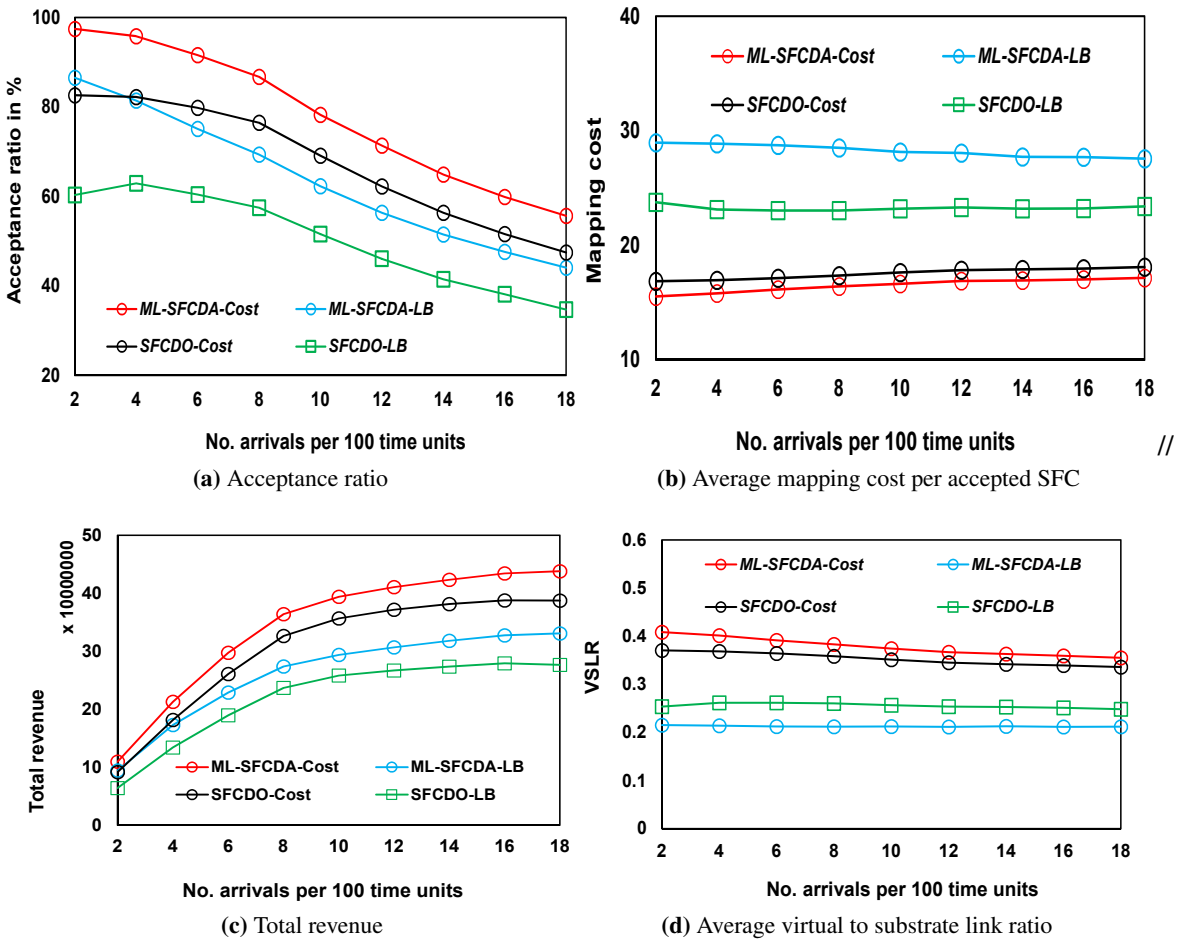
(a) Average acceptance ratio **(b)** Average virtual to substrate link ratio per SFC **(c)** Averege execution time per admitted request

Figure 5.7: Experiment 2 of Scenario 1: The impact of substrate network size is analyzed by varying the number of substrate nodes from 30 to 80 considering arrival rate of 8 requests per 100 time units for a total of 50000 time units.

.

processing each request as the number of substrate nodes increases. This was expected, since the number of possible candidates for each VNF increases with the increase of the substrate network size, hence, increasing the number of paths computations for both algorithms. ML-SFCDA on average provisions each request within several milliseconds (4.2 milliseconds), although the execution time of this algorithm is higher than for SFCDO, whose average processing time per admitted request is 1.3 milliseconds. These results further demonstrate the ability of ML-SFCDA to yield a good performance in terms of both: acceptance ratio and request provisioning cost, while executing in a feasible time, even for large scale networks.

In **experiment 3**, whose results are shown in Fig. 5.8, the performance of ML-SFCDA and SFCDO is tested on a real network topology considering different request arrival rates. For the two algorithms, two objectives have been considered: the cost (mapping aimed at reducing consumed resources) and the load-balancing (mapping aimed at balancing load traffic among links). From the obtained results shown in figure 5.8a, ML-SFCDA-Cost achieves a performance of 77.9%, followed by SFCDO-Cost with 67.6% in terms of AR averaged across all arrival rates. These are followed by ML-SFCDA-LB and SFCDO-LB whose avearge AR values are 63.8% and 50.4% respectively. Therefore, the proposed ML-SFCDA algorithm outperforms SFCDO by more than 10% on average, in terms of AR, considering each mapping objective. Moreover, with the objective of minimizing the mapping cost, ML-SFCDA outperforms SFCDO in terms of mapping cost by 5%, of VSLR by 6.7%, and of total revenue by 12%, as shown Figs. 5.8b, 5.8d, 5.8c. This is because, at any time, SFCDO greedily selects the node for hosting the VNF considering only the performance at the previous step, which affects the overall AR and cost performances. The better delay performance of the ML-SFCDA is attributed to the ability to use fewer substrate links for each virtual link. The lower performance of the load balance based ML-SFCDA-LB is due to its ability to select the least utilized links for



(a) Acceptance ratio

(b) Average mapping cost per accepted SFC

(c) Total revenue

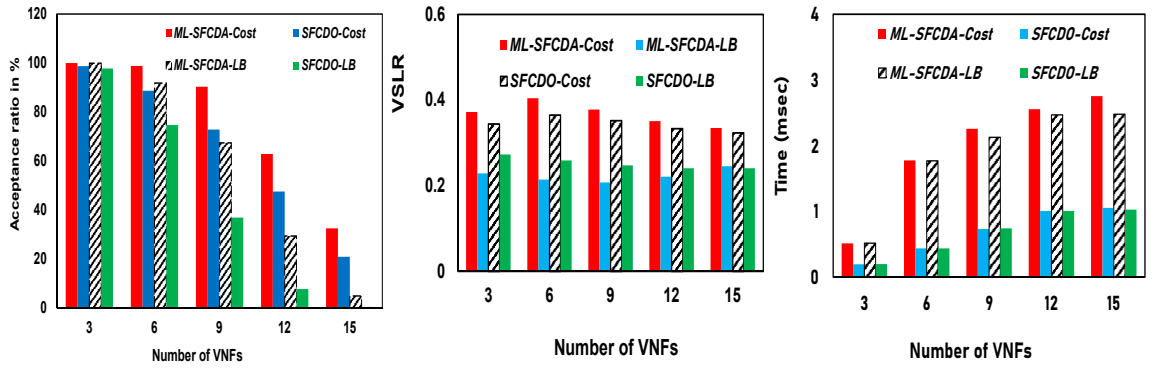
(d) Average virtual to substrate link ratio

Figure 5.8: Experiment 3 of Scenario 1: The impact of the arrival rate is evaluated by varying the arrival rate of SFCs from 2 to 18 arrivals per 100 time units for a total of 50000 time units considering ChinaNet topology. The results of the the proposed ML-SFCDA considering objectives of mapping cost (ML-SFCDA-Cost), load balancing (ML-SFCDA-LB) and the state of the art SFCDO algorithm considering mapping cost minimization (SFCDO-Cost) and link load balance (SFCDO-LB) are indicated in figures 5.8a - 5.8d

mapping the request, compared to the SFCDO-LB algorithm. Consequently, ML-SFCDA-LB may map requests on longer paths, hence, affecting its performance in terms of mapping cost, VSLR, total revenue and end-to-end delay, compared to SFCDO-LB.

In **experiment 4,** whose results are shown in figure 5.9, the impact of the SFC size is evaluated under an increasing number of VNFs per request, from 3 to 15, considering the ChinaNet topology. In terms of AR, ML-SFCDA outperforms SFCDO algorithm by 11.1% and 15.2%, considering cost and load balancing objectives, respectively, as reflected in Fig. 5.9a. As shown in figure Fig. 5.9c, the average provisioning time of each admitted request for both algorithms increases as the number of VNFs per request increases, due to an increase in the number of stages of the graph to obtain

the solution. Moreover, the execution time of ML-SFCDA is higher than that of SFCDO by 1.3 milliseconds on average for the two mapping objectives. This is due to the extra number of shortest paths computed by the ML-SFCDA at each stage. However, both algorithms are able to provision each request within a fraction of a second.



(a) Average acceptance ratio **(b)** Average virtual to substrate link ratio per SFC **(c)** Average execution time per admitted request

Figure 5.9: Experiment 2 of Scenario 1: The impact of the number of VNFs per SFC is evaluated by varying the number of VNFs per SFC from 3 to 15 with arrival rate fixed at 3 requests per 100 time units for a total of 50000 time units. The results of the the proposed ML-SFCDA considering objectives of mapping cost (ML-SFCDA-Cost), load balancing (ML-SFCDA-LB) and the state of the art SFCDO algorithm considering mapping cost minimization (SFCDO-Cost) and link load balance (SFCDO-LB) are indicated in figures 5.9a-5.9c

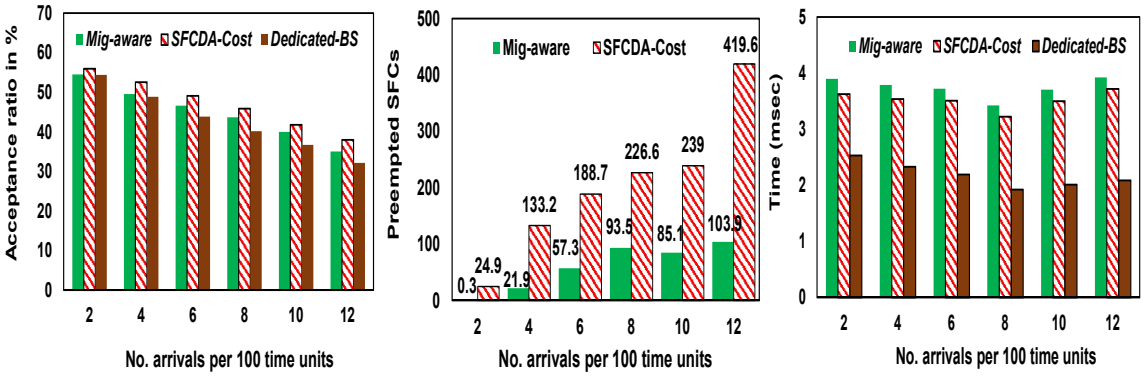
.

Scenario 02: Migration-aware algorithm performance analysis

This scenario aims to analyze the performance of the proposed migration-aware algorithm (Mig-aware). In this case, the algorithms are compared in terms of AR, average number of preempted users per unit of time, execution time and mapping cost. All experiments have been run using the ChinaNet topology.

In **experiment 1,** whose results are shown in figure 5.10, the impact of the arrival rate on the performance of the algorithms is analysed. From figure 5.10b, Mig-aware results in an average of 60 preempted demands per unit of time, averaged across all arrival rates, while the cost based SFCDA-Cost approach results in an average preemption of 205 requests (approximately 70% extra preemptions) per unit of time. This result demonstrates that the Mig-aware obtains a significant gain, of up to 70%, in terms of service availability for low priority users by minimising the number of preemptions. Moreover, such preemptions are penalized with greater delays when remapping the preempted requests, including the possibility that the remapping algorithm may fail trying to find a feasible solution for restoring the preempted service. The average processing time of each admitted request for all the three algorithms is in a fraction of a second, with the Dedicated-BS having the best performance of 2.18 milliseconds, followed by SFCDA-Cost with 3.52 milliseconds and Mig-aware with 3.74 milliseconds, averaged across the different arrival rates, as shown in figure

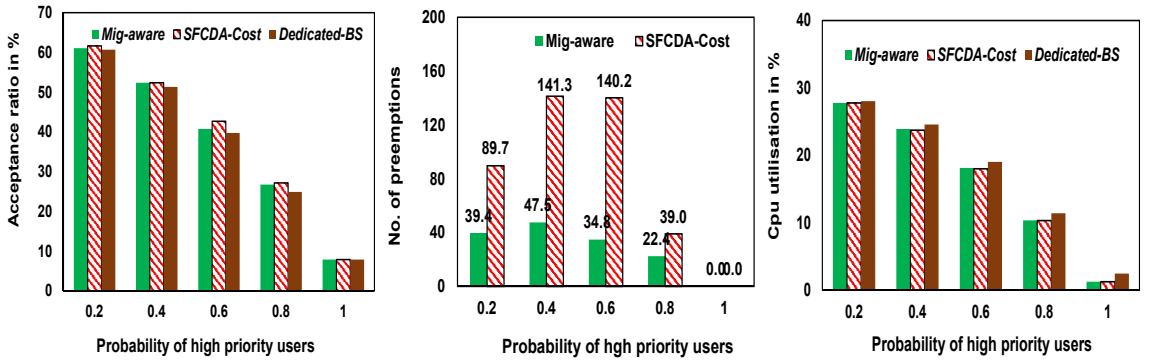
5.10c. The additional overhead in terms of execution time for Mig-aware and SFCDA-Cost is due to the remapping of the preempted low priority users, which is not done in the Dedicated-BS, since it does not contemplate the resource sharing. Moreover, the Mig-aware approach experiences a small overhead in terms of time, due to the extra work of edge and node weight computation. The Cost based SFCDA-Cost results in the best performance, in terms of AR, with an average value of 47.17%, averaged over the different arrival rates, followed by Mig-aware with 44.87% and Dedicated-BS with 42.66%, as shown in figure 5.10a. The superior performance of the SFCDA-Cost is attributed to the ability to map both the high priority and low priority requests when minimizing the mapping cost, as for Dedicated-BS, but at the same time, reusing the idle backup resources, like the Mig-aware algorithm.



(a) Average acceptance ratio **(b)** Average number of preempted requests per unit time **(c)** Average processing time per admitted request

Figure 5.10: Experiment 1 of Scenario 2: The impact of the arrival rate is evaluated by varying the arrival rate of SFCs from 2 to 18 arrivals per 100 time units for a total of 50000 time units. The results of the the proposed migration-aware algorithm (Mig-aware), the cost based bench mark algorithm (SFCDA-Cost) and dedicated backup scheme (Dedicated-BS) are shown in figures 5.10a-5.10c

In **experiment 2,** whose results are shown in figure 5.11, the impact of the fraction of high priority users among the total arriving requests is analysed by varying the generation probability of high priority users from 0.2 to 1. Figure 5.11a reveals a decline in the AR performance, across all the provisioning strategies, as the fraction of high priority users increases. This is because such users have a lower probability of being mapped, compared to the lower priority users, due to the difficulty in obtaining a pair of disjoint paths of nodes for provisioning the primary and backup solution for those users. Consequently, the overall AR performance decreases. This also explains the decrease in the total consumed processing resources, as shown in figure 5.11c, even when the average mapping cost per admitted request increases, as the number of high priority requests increases. Moreover, from these results, the AR performance of the Mig-aware algorithm is not worsening in this experiment, in comparison to the other algorithms. Again, this experiment shows that Mig-aware is efficient to decrease the number of preemptions, and, therefore, the low priority services result in a better service condition with an average value of 28.8 low priority requests being preempted per unit of time, compared to 82 requests of the cost based approach, as shown in figure 5.11b.



(a) Average acceptance ratio **(b)** Average number of preempted requests per unit time **(c)** Average processing time per admitted request

Figure 5.11: Experiment 2 of Scenario 2: The impact of the fraction of the high priority users among the arriving demand is evaluated by varying the probability of generation of high priority demand from 0.1 to 1 considering 5 arrivals per 100 time units for a total of 40000 time units. The results of the the proposed migration-aware algorithm (Mig-aware), dedicated backup scheme (Dedicated-BS) and the cost based bench mark algorithm (SFCDA-Cost) are shown in figures 5.11a-5.11c

Scenario 3: QoS-aware service restoration algorithm performance analysis

In Section 5.3.4, a QOS-aware algorithm for remapping preempted low priority requests was descried, whose target was to minimize the QoS violations by reusing surviving substrate nodes and paths whenever possible. This scenario evaluates the performance of this algorithm (denoted in this section as Global-QoS) against two benchmark algorithms: one where the preempted users are remapped using the ML-SFCDA algorithm discussed in Section 5.3.1, with the objective of minimising the provisioning cost and denoted as Global-cost. And the other using a local rerouting based service restoration algorithm, as proposed in [68] and denoted as Local-SR.

In **experiment 1**, whose results are shown in figure 5.12, we analyze the impact of a varying arrival rate on the performance of the algorithms. Figures 5.12b and 5.12a reveal that Global-QoS results in up to 21% and 30% improvements, in terms of the number of surviving VNFs and virtual links, respectively, that are migrated to new substrate nodes and paths on average, for each remapped SFC. Global-QoS results in an average of 9.6% and 48.4% of the number of surviving VNFs and virtual links, respectively, migrated to new substrate nodes and paths for each remapped request, while the resulting performance for Global-Cost algorithm is 31.4% and 78.5%, averaged over all arrival rates. These results are the consequence of the weighting approach adopted by Global-QoS, which favors surviving VNFs and virtual links to reuse the same nodes and links during service restoration. Such an approach achieves a reduction on the delays associated with the migration and instantiation of VNFs on new substrate nodes, hence, minimizing the level of service disruption, resulting in a better QoS performance. Moreover, Global-QoS remains competitive, in terms of average number of successful request remappings (within a 2% margin) and acceptance ratio (within a 6.5% margin) compared to Global-Cost, as reflected in figures 5.12d and 5.12c. From figure 5.12d

QoS-aware results in an improvement of 14% in terms of successful request remappings, averaged over all arrival rates, compared to the local-rerouting approach, with minimal execution time overhead. Considering average values across all arrival rates, Global-Cost results in the best performance in terms of average AR and total revenue, with average values of 47.13% and 9088061.0, as depicted in figures 5.12c and 5.12f. This is due to the fact that Global-Cost remaps the failed nodes and links on shorter paths, hence, resulting in a lower resource consumption. This is followed by Global-QoS and Local-SR with average values of 40.5%, 7089438.52, and 28.9%, 4557531.723, respectively, in terms of average AR and total revenue. The poor performance of the Local-SR approach is due to the fact that computing the solution locally may result in a higher resource consumption and even failure to obtain a remapping solution.

In **experiment 2,** whose results are shown in figure 5.13, we analyse the impact of a varying number of VNFs per SFC on the performance of the algorithms. Figure 5.13a shows a decline in terms of AR performance for all the algorithms, as the SFC size increases. This is because the probability of finding a feasible mapping solution for a given SFC request decreases with the increase of the SFC size, due to the difficulty on satisfying the request constraints in terms of delay and resource requirements. Global-QoS achieves an average value of 22.8%, which is approximately 5% higher than Local-SR, and within a 3% margin compared to Global-Cost, whose AR values are 17.7% and 25.7%, averaged across all SFC sizes. In 5.13b, Local-SR results in the highest value of remapping failure, with a remapping failure rate of 30.79%, which is 9% and 21% higher than Global-Qos and Global-Cost, respectively. Figure 5.13c, shows that Global-QoS and Global-Cost result on average result in 8.6% and 29.4% respectively of the number of surviving VNFs being migrated to new nodes upon service restoration. This translates in up to a 21% performance improvement of Global-Qos compared to Global-Cost. From figure 5.13d, as the number of VNFs increases, the performance gain in terms of service restoration time, compared to using a local rerouting approach, decreases. This is because, as the VNFs increase, the probability of multiple VNFs/virtual links failures increases, and this requires running the local rerouting algorithm more than once for the same request, hence, increasing its execution time. Moreover, all the algorithms show a similar performance (within a 0.01 millisecond margin) in terms of average processing time per remapped request, with an average value of 0.6 milliseconds, averaged over all SFC sizes.

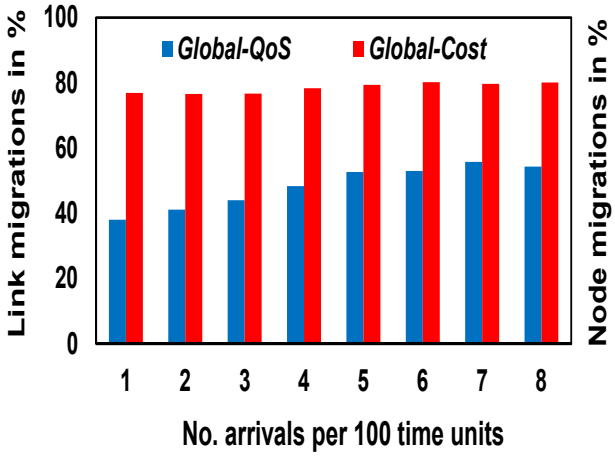
This scenario has demonstrated that the proposed Global-QoS results in a significant reduction in the number of surviving VNFs that are migrated to new VNF instances, while yielding competitive results in terms of acceptance ratio and mapping cost, in comparison with the Global-Cost algorithm.

5.5 Conclusion

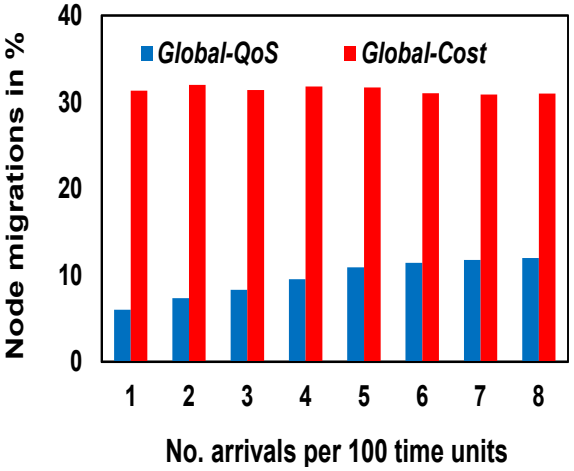
In this chapter, a multi-stage graph based algorithm has been proposed as an alternative algorithm for SFC provisioning within single substrate networks. Simulation results reveal that the proposed algorithm can result in more than a 10% performance improvement, in terms of acceptance ratio, compared to a state-of-the-art algorithm and within a 4% margin of the optimal solution. Therefore,

the algorithm has been found to be scalable when considering an increasing network size and demand. Moreover, the algorithm can be tailored, easily and flexibly, to different mapping objectives.

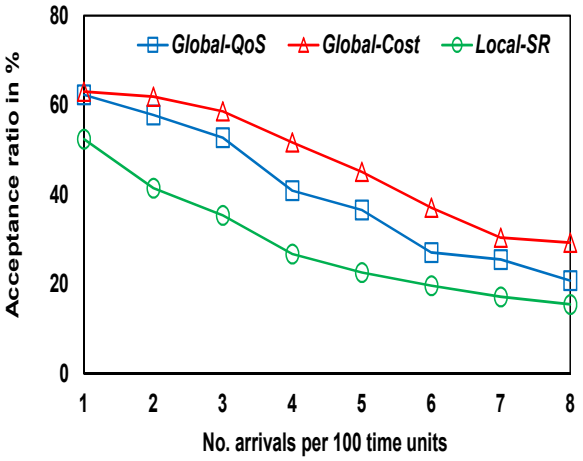
On considering a scenario with requests belonging to two different priorities, with the possibility of the low priority requests to borrow the unused backup resources of the high priority users, the above mentioned algorithm has been modified to implement a migration-aware algorithm for the provisioning of low priority requests. The new algorithm results in a gain of up to 70% in terms of service availability of low priority users, thanks to minimising the number of preemptions. Moreover, for the low priority users preempted from their assigned resources, we have proposed a third algorithm, the QoS-aware service restoration algorithm, that prioritises the reuse of surviving VNFs and virtual links, instead of considering the whole request remapping, hence, minimising the overall delay and cost associated with the migration.



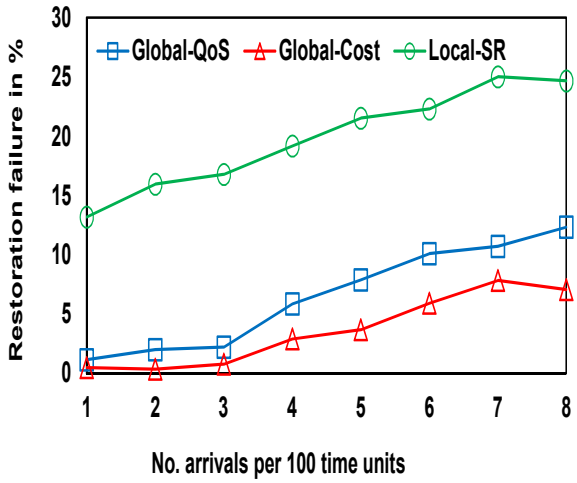
(a) Average link migrations



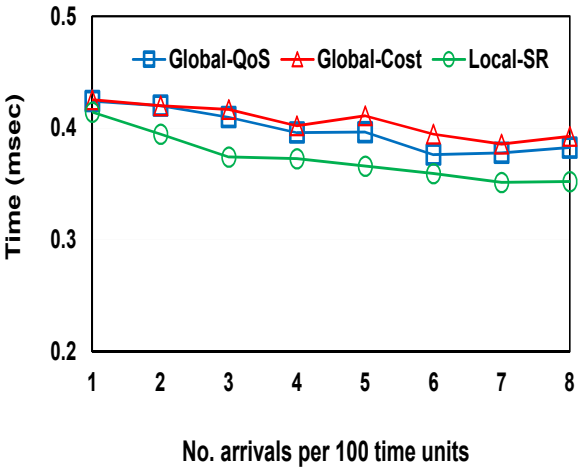
(b) Average node migrations



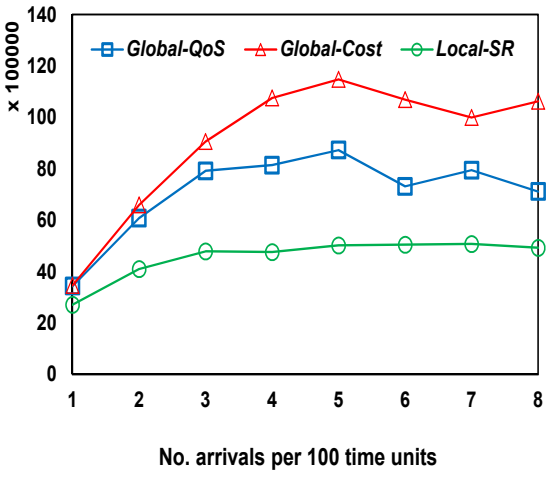
(c) Average acceptance ratio



(d) Average serevice restoration failure

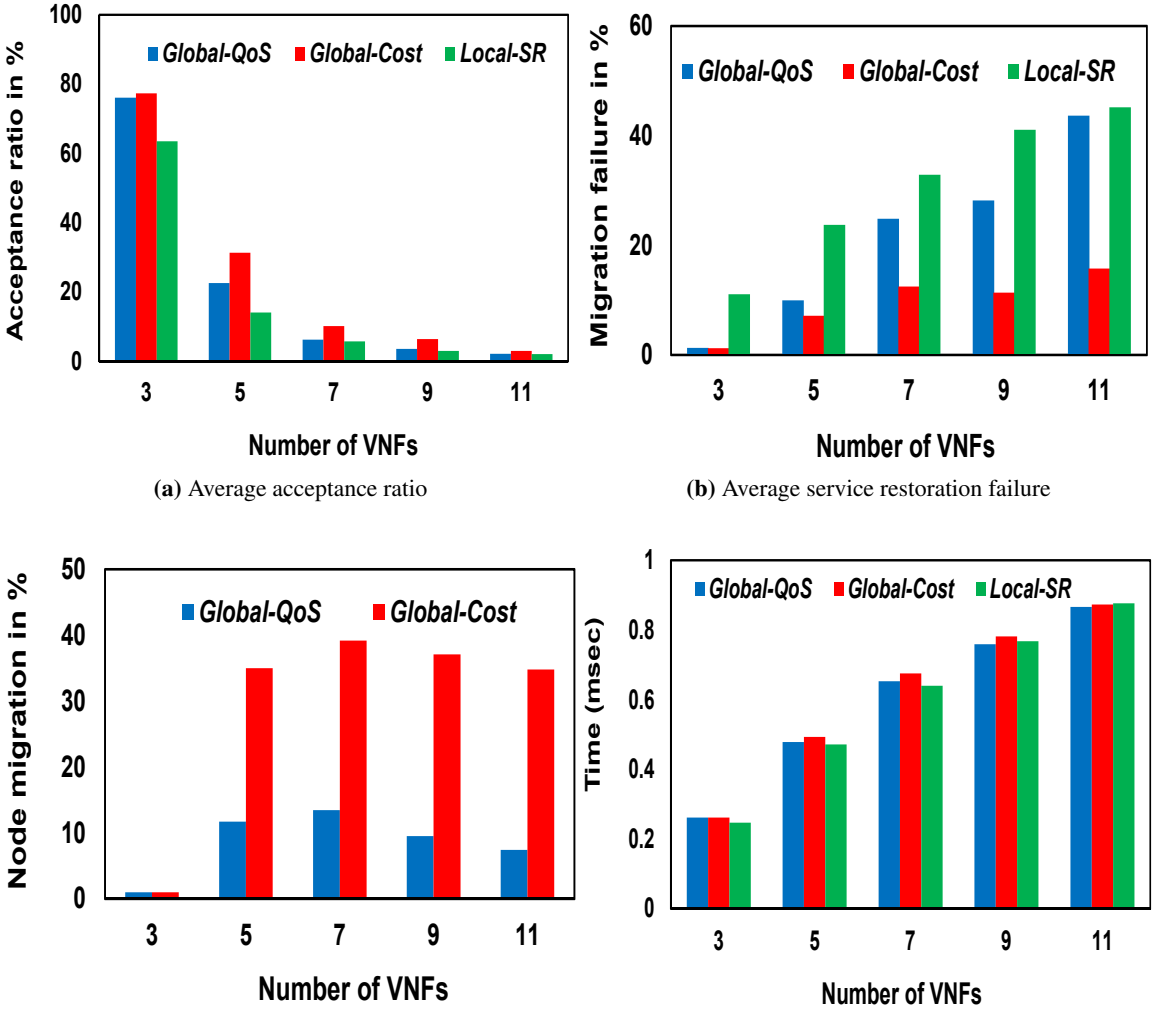


(e) Average processing time per SFC



(f) Total received revenue

Figure 5.12: Experiment 1 of simulation scenario 3: performance analysis of the QoS-aware algorithm considering an online scenario. The impact of arrival rate is evaluated in this experiment by varying the arrival rates from 1 to 8 arrivals per 100 time units for a total of 50000 time units.



(a) Average acceptance ratio

(b) Average service restoration failure

(c) Average number of migrated VNFs

(d) Average processing time per SFC

Figure 5.13: Experiment 2 of simulation scenario 3: Performance analysis of the QoS-aware algorithm considering online case. The experiment evaluates the impact of the SFC size by varying the number of VNFs per SFC from 3 to 11 with the arrival rate fixed at 5 requests per 100 time units for a total of 20000 time units.

CHAPTER 6

Fault tolerant placement of Stateful VNFs

6.1 Introduction

The proposals in Chapters 3 and 5 addressed the problem of survivable VNF orchestration from the perspective of stateless VNFs. However, stateful VNFs generate states that require to be continuously transferred to their stand-by instances in order to guarantee seamless traffic redirection upon failure. If not well managed, such updates may result in a high consumption of bandwidth resources, potentially resulting in network congestion and rejection of incoming requests. In Fig. 6.1, an illustration of the fault-tolerant placement of an SFC instance considering a scenario in which the entire instance is placed on a single Data Center (DC) is given. With the active and stand-by instances placed on DC B and D respectively, there is need for a state-update path between these DCs as shown in the red dotted line. In this way, the placement problem needs to jointly optimize the resource consumption due to state update in addition to that due to the active and stand-by instances. This requires full coordination during the mapping of the active and stand-by instances, since these are inter-linked through the state update event, unlike for the stateless case as addressed in [29, 38–45].

As an enhancement to the previous chapters of the thesis, this chapter proposes a Metaheuristic approach to tackle the problem of resource efficient and fault-tolerant orchestration of SFCs while considering stateful VNFs. Metaheuristic approaches have previously been adopted for the problem of SFC orchestration including GA [99, 104, 150], Particle Swarm [137, 151], Anti-colony [152, 153], and Harmony-search [110]. However, these donot focus on fault-tolerant orchestration and consider stateless VNFs in their execution. To the best of the authors' knowledge, with respect to the stateful VNF placement, this is the first work that: i) adopts a flexible deployment scheme that does not constrain an entire SFC to be placed on a single DC; ii) adopts a fully coordinated approach for placement of both active and stand-by instances, thanks to a request transformation/augmentation technique proposed by the chapter; and iii) adopts a genetic and harmony search meta-heuristics approach to solve the fault-tolerant orchestration problem. In light of the above, the key contributions of this chapter to the overall thesis can be summarized as follows:

- Formulation of the fault-tolerant stateful VNF orchestration problem considering a scenario in which the different VNFs of an SFC instance can be deployed across different substrate nodes (e.g., Data Centers, servers etc.)

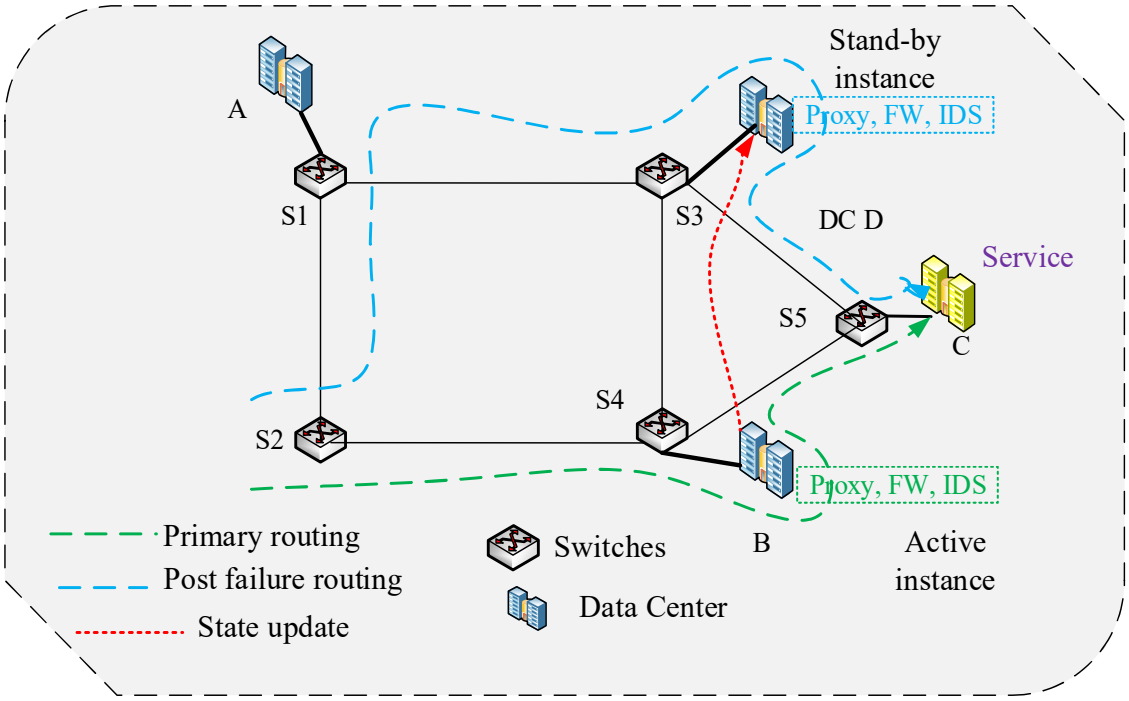


Figure 6.1: An illustration of the Fault-tolerant SFC placement with the active instance provisioned on Data Center B and the stand-by instance provisioned on Data center D. The red dotted line corresponds to the state update transfer from DC B to DC D

- Proposal of a request augmentation technique in which the primary and stand-by instances of a request are combined, and jointly mapped as a single request graph, with a target of achieving full coordination between the mapping of the active and stand-by instances of a service request.
- Proposal of Genetic and Harmony Search meta-heuristic algorithms for solving the formulated NP-hard problem in feasible time.
- Additionally, for a special case in which an entire SFC instance is required to be mapped in a single node as adopted in literature, the chapter proposes a heuristic algorithm based on a bi-stage graph with near optimal performance while executing in practical run time.

The rest of this chapter is organized as follows: Section 6.2 introduces a description of the fault-tolerant stateful VNF orchestration problem. Then, the proposed service orchestration algorithms are introduced in Sections 6.3 and 6.4. Specifically, Section 6.3 introduces the Bi-stage graph based algorithm for mapping an entire SFC instance on a single DC, while Section 6.4 introduces the Metaheuristic i.e., Genetic and Harmony search algorithms in which the different VNFs of an SFC instance may be mapped across different nodes. The performance evaluation of the proposed algorithms is presented in Section 6.5 and the chapter is concluded in Section 6.6.

6.2 Description and formulation of the fault-tolerant stateful VNF orchestration problem

This chapter envisions a scenario in which the survivability of a given service request is achieved by pro-actively provisioning backup instances for each VNF of the request [40, 154]. This is a realistic consideration given the stringent service reliability requirements of mission-critical applications expected in 5G and beyond networks. In addition, we constrain the active and stand-by instance solutions to be disjoint in terms of the host nodes, with a target of minimizing the probability of joint failure of both the active and stand-by instances. Thus,

$$x_j^{i,r} \times \gamma_j^{i,r} = 0 \;\; \forall i \in N_v, \forall j \in N_s \tag{6.1}$$

Where $x_j^{i,r} \in \{0,1\} = 1$ if the active instance of VNF i is mapped onto substrate node $j \in N_s$ and $\gamma_j^{i,r} = 1$ if the standby instance of VNF i is mapped on substrate node $j \in N_s$. In light of the above considerations, the fault-tolerant stateful VNF orchestration problem involves making the following two key decisions:

- Selection of the node(s) and substrate edges/paths on which to map the different VNFs and virtual links respectively of the active SFC instance.
- Selection of the node(s) and substrate edges/paths on which to map the different VNFs and virtual links of the stand-by SFC instance respectively.

With the view that the different VNFs of an instance can be placed on different substrate nodes, the total provisioning cost for a given request consists of: $i)$ the cost of the computational resources allocated/reserved at the node(s) supporting the active/ stand-by instances; $ii)$ transmission cost for the paths between the ingress and egress nodes for the active and stand-by instances; and $iii)$ transmission cost for propagating the state information from the node(s) supporting the active instance to each of the node(s) supporting the stand-by instances.

Therefore, for each request, the placement problem involves obtaining a set DC_{opt} composed of nodes that jointly minimize the above cost components, where $|DC_{opt}| > 1$. In order to formulate the total provisioning cost, we let $x_j^{i,r} \in \{0,1\}$, $\gamma_j^{i,r} \in \{0,1\}$, denote binary variables each equal to 1 if the active instance and stand-by instance respectively of VNF i of SFC request r is provisioned on node j, zero otherwise. Similarly, we let $\sigma_e^{r,act} \in \{0,1\}$, $\sigma_e^{r,bk} \in \{0,1\}$ and $\sigma_e^{r,su} \in \{0,1\}$ denote binary variables equal to 1 if the substrate edge $e \in E$ is used in the active path, backup path and state update path respectively, zero otherwise. Then, the placement cost C^r for request $r \in \mathbb{R}$ is computed as the sum of processing cost and transmission cost as below:

$$C^r = C_{prcs} + C_{trans} \tag{6.2}$$

where C_{prcs} denotes the processing cost at the nodes and computed as below:

$$C_{prcs} = \sum_{i \in N_v} x_j^{i,r} \rho_j^p C_{dem}^{i,r} + \sum_{i \in N_v} \sum_{j \in N_s} \gamma_j^{i,r} \rho_j^p C_{dem}^{i,r} \tag{6.3}$$

where the first and second terms in equation 6.3 correspond to the cost from the active and stand-by instances respectively. $C_{dem}^{i,r}$ denotes the computational resources associated with VNF instance i. Note that the formulation for the second term caters for the possible case in which each VNF instance may be associated with more than one stand-by instance. If we denote by Bws_{dem}^r as the state update rate from the active to stand-by instance, then, the transmission cost, C_{trans} is computed as shown in equation 6.4 with the first, second and third terms respectively corresponding to the active instance, backup instance and state update cost.

$$\begin{aligned} C_{trans} = \sum_{e \in E} \sigma_e^{r,act} \rho_e^t Bwt_{dem}^r + \sum_{e \in E} \sigma_e^{r,bk} \rho_e^t Bwt_{dem}^r + \\ \sum_{e \in E} \sigma_e^{r,su} \rho_e^t Bws_{dem}^r \end{aligned} \tag{6.4}$$

Note that this work considers the cost for each unit of processing rate for a given VNF to be uniform across the different nodes of the network. Consequently, the variations in placement cost as expressed in equation 6.2, is majorly influenced by the number of edges used for hosting the active , state update and stand-by instances of the request. Consequently, the optimisation problem is formulated as that of minimising the number of substrate edges used for provisioning the request, and mathematically expressed as:

$$Minimise: \sum_{e \in E} \sigma_e^{r,act} + \sum_{e \in E} \sigma_e^{r,bk} + \sum_{e \in E} \sigma_e^{r,su} \tag{6.5}$$

where the first, second and third terms of equation 6.5 correspond to the edges used in the active, stand-by and state-update paths respectively. Equation. 6.5 is solved under the constraints specified in Eqns. 2.6 - 2.13 and 6.1.

Owing to the NP-hard nature of the above problem, this chapter proposes a meta-heuristic approach with the ability to realize near-optimal solutions to the problem in feasible running time. In formulating a solution to the above problem, the chapter considers two possible service deployment scenarios:

1. A scenario in which an entire SFC instance is constrained to be placed in a single node(e.g., Data center) as adopted in the existing works [26–28]. For this scenario, the thesis proposes a heuristic approach that relies on a Bi-stage graph for computing the mapping solution.
2. A scenario in which the different VNFs of an SFC instance can be deployed across multiple DCs. Due to the complexity involved in the second scenario, the paper proposes a meta-heuristic approach, namely Genetic and Harmony search algorithms for obtaining the orchestration solutions.

A description of these algorithms follows below starting with the former scenario.

6.3 Proposed Bi-stage graph based algorithm

This section introduces the Bi-stage graph based algorithm for fault-tolerant orchestration of stateful VNFs considering a scenario in which an entire SFC instance has to be deployed within a single DC node. In this regard, the placement problem involves obtaining a single DC for hosting the active SFC instance and a set of DC(s) for hosting the stand-by instance(s). The proposed algorithm is executed in three sequential steps: candidates matching, Bi-stage graph computation and solution evaluation. The candidate matching step targets to associate each SFC instance with a set of candidate DCs that can potentially map either the active or stand-by instances of the SFC. Given the above candidate sets, Bi-stage graph computation step constructs a weighted Bi-stage graph, which is then used by the solution evaluation step to obtain a request mapping solution. A detailed description of the above steps is given below:

6.3.1 Candidates matching

Aware that the residual resources and the QoS guarantees across the different DCs continuously change with time, the different resource constraints (e.g., processing, storage, memory, and bandwidth, among others) associated with a request received at a given time instance may be met by only a subset of the available DCs. Therefore, the candidates matching step targets to associate each SFC request with a set of such DCs that can support the deployment of either the active or stand-by instances of the request. In so doing, the step ensures that only feasible DCs are used in the construction of the bi-stage graph, which not only results in a reduction in the execution time of the algorithm, but also ensures that unfeasible DCs are not returned as part of the mapping solutions.

To be a candidate for an SFC instance $r \in \mathbb{R}$, $DC_j \in Dc$ must satisfy the following:

- The residual resources (e.g., processing, memory and storage, among others) on DC_j should satisfy the corresponding resource requirements of the service request. This chapter considers only processing resources. However, other resources can be easily integrated into the proposed model.
- There should exist a path between the ingress and egress nodes going through DC_j that satisfies the requirements of the service request in terms of bandwidth and end-to-end delay requirement.

The pseudo-code for the candidates matching step is shown in Algorithm 8. The algorithm takes as input the substrate network graph and the request tuple containing the request specifications. Then, for each substrate node n_s, the algorithm checks if the residual resources $C^{n_s}_{res}$ of the node can satisfy the computational requirements of the SFC and if there is a feasible path (in terms of bandwidth and end-to-end delay) from source to destination that goes through n_s. In case there are no feasible candidates for the SFC, the algorithm rejects the request, otherwise, it returns $Cand^r_{act}$, $Cand^r_{std}$ and

$dict^r_w$ corresponding to a set containing candidate nodes for the active SFC instance, a set containing candidate nodes for the stand-by instance, and a dictionary containing the weights of the different nodes. The weight of a node n_s in this case corresponds to the number of hops of the shortest available path between the ingress and egress nodes that goes through node n_s. Note that in this work, we reserve the resources for the stand-by instance, hence, we consider the stand-by instances of the SFC to have the same requirements as the active instance in terms of end-to-end delay and computational requirements. Therefore, a node which is a candidate for the active instance is also a feasible candidate for the stand-by instance, making $Cand^r_{act}$=$Cand^r_{std}$. However, in scenarios where resources need not to be reserved for the stand-by instance, such sets may be different since there is no resource constraint imposed on the candidates of the stand-by instance.

Algorithm 8 Candidates matching step

Input: $G_s, < G^r_v, C^r_{dem}, Bwt^r_{dem}, Del^r_{sd}, \tau^r_s, \tau^r_d, \tau^r_f >$
Output: $Cand^r_{act}$, $Cand^r_{std}$, $dict^r_w$
Initialise: $Cand^r_{n_s}$=$\emptyset$, $Cand^r_{std}$= $\emptyset$, $dict^r_w$=$\emptyset$
for $n_s \in N_s$ **do**
  if $C^{n_s}_{res} \geq C^r_{dem}$ **then**
    $path^{s,d}$=$Dijkstra(\tau^r_s, \tau^r_d, n_s)$
    if $path^{s,d}$ *is feasible* **then**
      Add n_s to $Cand^r_{n_s}$
      Add n_s to $Cand^r_{std}$
      Add $path^{s,d}$ to $dict^r_w$
    end
  end
end
if $Cand^r_{act}$==$\emptyset$ *or* $Cand^r_{std}$==$\emptyset$ **then**
  Reject request
end
else
  Return $Cand^r_{act}$, $Cand^r_{std}$, $dict^r_w$
end

6.3.2 Bi-stage graph computation

This step exploits the sets $Cand^r_{act}$ and $Cand^r_{std}$ containing the candidates for the active and stand-by instances respectively, and the dictionary $dict^r_w$ containing the weights of the different candidate nodes as obtained from the candidate matching step to construct a Bi-stage graph as shown in Fig. 6.2. First, the sets $Cand^r_{act}$ and $Cand^r_{std}$ are sorted in increasing order of their respective node weights. Then, the nodes in $Cand^r_{act}$, and $Cand^r_{std}$ are placed at the first and second stages of the graph respectively, with the least weight nodes being placed at the topmost levels at each stage. Each node $n_s \in N_s$ at each stage is associated with a weight ω_{n_s} which corresponds to the number of hops of the path from the ingress node to the egress node that goes through that node. For the nodes at the first stage, such a weight corresponds to the number of hops to be traversed by traffic of the active instance from source to destination. Similarly, for each node $n \in Cand^r_{std}$ at the second stage,

the weignt ω_{n_s} corresponds to the number of hops to be traversed by traffic from the source to the destination nodes if node $n \in N_s$ is chosen as a host for a stand-by SFC instance. Then, the algorithm selects the topmost K nodes at the first stage from which to select the DC_{active} and the topmost J nodes at the second stage from which to select the stand-by DC(s) Dc_{std}. Observe that $J \geq N_{std}$ where N_{std} is the required number of stand-by DCs. Then, each node $m \in Cand^r_{act}$ among the top most K nodes at the first stage is connected to each node $n \in Cand^r_{std}$ among the top most J nodes at the second stage, where $m \neq n$, through a weighted path computed using Dijkstra algorithm. The requirement $m \neq n$ ensures that a single node is not selected for hosting both the active and stand-by instances, since in practice, its possible that a given node is a candidate for both the active and stand-by instances.

The weight ω_{mn} of the path connecting nodes m and n, relates to the state update cost between the two nodes considering that m and n are chosen as the host for the active and stand-by instances respectively. This weight is computed as follows:

$$\omega_{mn} = \frac{Bws^r_{dem}}{Bwt^r_{dem}} \times H_{mn} \tag{6.6}$$

where H_{mn} is the number of hops of the substrate path $m - n$. We consider weighting H_{mn} since the state update rate Bws^r_{dem}, i.e., the transmission rate between nodes m and n, may be different from the transmission rate Bwt^r_{dem}, i.e., the transmission rate between the source s and the target node t, hence, imposing different impacts in terms of cost even for the same number of hops. Note that in case of no connection between m and n, then, this denotes an infeasible pair, hence, such a weight is infinite.

6.3.3 Computation of the provisioning solution

The mapping solution is obtained by associating each node m among the K candidate nodes at the first stage with a global cost $C^{N_{std}}_m$. The cost $C^{N_{std}}_m$ relates to the global cost that would be incurred by selecting the node m as the host for the active instance and selecting the $N_{std} \leq J$ node(s) from stage 2 as the hosts for the stand-by instance (s), where N_{std} denotes the required number of stand-by instances. In this work, we set $N_{std} = 1$. The global cost $C^{N_{std}}_m$ is evaluated as follows:

Each node $n \in Cand^r_{std}$ connected to m is associated with a cost value with respect to m, denoted as c^n_m which is evaluated as the sum of the interconnecting path weight ω_{mn} and its node weight w_n. Thus:

$$c^n_m = \omega_{mn} + \omega_n \tag{6.7}$$

Then, among all the J nodes, N_{std} nodes with the least node cost values are selected. Then $C^{N_{std}}_m$ is evaluated as the sum of w_m and the node values of the selected N_{std} nodes. Thus:

$$C^{N_{std}}_m = \omega_m + \sum_{n=1}^{N_{std}} c^n_m \tag{6.8}$$

Algorithm 9 Solution Construction

Input: $Cand^r_{act}$, $Cand^r_{std}$, $dict^r_w$
Output: $Cand_{active}$, $Cand^{N_{std}}_{std}$
Initialise: $Cand_{active}$=$None$, $Cand^{N_{std}}_{std}$= $None$
$k_{counter}$=0
$Global\_Cost\_dict = \emptyset$
for $m \in Cand^K_{act}$ **do**
  if $m_{counter} \geq K$ **then**
    break
  end
  else
    $k_{counter} = k_{counter} + 1$
    $j_{counter}$=0
    std_cost_array=[]
    for $n \in Cand^J_{std}$ **do**
      if $j_{counter} == J$ **then**
        break
      end
      else
        $j_{counter} = j_{counter} + 1$
        $c^m_n = \omega_{mn} + \omega_n$
        append c^m_n to std_cost_array
      end
    end
    if *len(*std_cost_array*)*<N_{std} **then**
      continue
    end
    else
      -Sort std_cost_array in increasing value
      -Extract the first N_{std} nodes
      -Compute the global cost $C^{N_{std}}_m$
      -Store $C^{N_{std}}_m$,m and N_{std} in $Global\_Cost\_dict$
    end
  end
end
if $Global\_Cost\_dict == \emptyset$ **then**
  Reject Request
end
else
  sort $Global\_Cost\_dict$ in increasing cost
  -Extract $Cand_{active}$ and the corresponding $Cand^{N_{std}}_{std}$
end

Then, the mapping solution is chosen as the node m with the least global value, in case m is chosen as the host for the active instance and the corresponding N_{std} nodes being chosen as the host for the stand-by instances.

The parameters K and J are optimization parameters that set a trade-off between solution optimality and execution time of the solution evaluation step. The bigger the values of K and J, the higher the chances of getting a better solution, albeit at a higher execution time. The case K=1 and J=1, corresponds to selecting nodes with the lowest cost in terms of placement of both the active and stand-by instances, although these may be associated with a high state update cost.

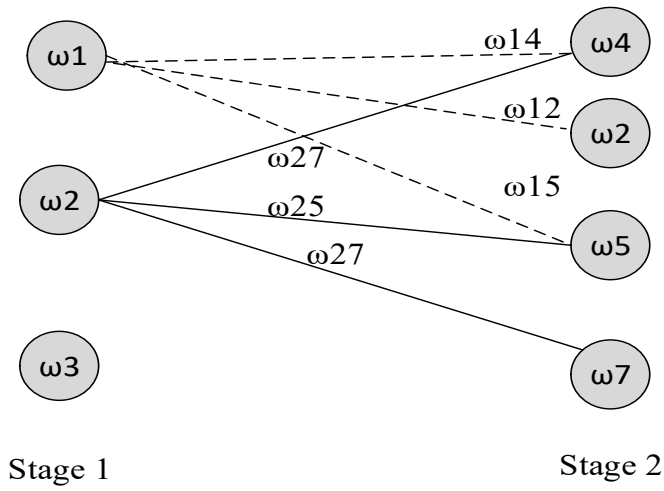


Figure 6.2: Illustration of the bi-stage graph in which there are 3 candidate nodes for active instance and 4 candidates nodes for stand-by instance. In this case, J=3 and K=2 and node 2 is a candidate for both the active and stand-by instances

Forexample, if k=J=1, this means that the algorithm tries to select the topmost node in the first layer as the candidate for the active instance, and the topmost node (if not equal to DC_{active}) in the second stage as the host for the stand-by instance. Incase any of these nodes is not feasible, then the request is rejected. Its clear that by increasing the value of K and J, the algorithm explores more alternatives from which to select DC_{active} and DC_{std}, hence increases the chances of obtaining better solutions, albeit at the expense of increased running time. Figure 6.2 shows an example of a Bi-stage graph with 3 candidate nodes for the active instance and 4 candidate nodes for the stand-by instance, with DC 2 being a candidate for both instances. In this case, $K = 2$ and $J = 3$. The weight costs c_x for each node x at the first and second stage correspond to the weight of the node in terms of the total number of hops between source and destination for the shortest path going through that node.

6.3.4 Time complexity analysis

The proposed algorithm consists of a candidates matching step, a Bi-stage graph computation step and the solution evaluation step. In the candidates matching step, the feasibility of each DC is evaluated by running Dijkstra algorithm from the source node to the DC , and from the DC to the terminal node. Considering the time-complexity of the Dijkstra algorithm as $\Theta(|E_s| + |N_s|log(|N_s|)) \approx \Theta(|N_s|log(|N_s|))$, the time complexity of the candidates matching step considering a substrate network composed of N_{Dc} Data centers can be approximated as $2N_{Dc} \times \Theta(|N_s|log(|N_s|)$. The Bi-stage graph computation step involves computing shortest path between each of the K candidate nodes in stage 1 to each of the J nodes in stage 2. Therefore the total number of shortest paths computed in this step are $K \times J$. Additionally, the step involves sorting the DCs according to their weights. The time complexity of this sorting step is $\Theta(N_{Dc} \times log(N_{Dc}))$. Therefore the time complexity of the

proposed Bi-stage algorithm can be approximated as $(k \times J + 2D_C)\Theta(|N_s|log(|N_s|)$.

6.4 Metaheuristic Orchestration algorithms

This section describes the orchestration algorithms based on the population metaheuristics Genetic Algorithm (GA) and Harmony Search (HS). In order to achieve coordinated mapping of the SFC instance, the two algorithms rely on a request transformation technique which involves combining the active SFC instance and the stand-by instance including the respective state update requirements into a single request graph which is then jointly embedded into the substrate network. First, the request transformation technique is introduced below.

6.4.1 Request transformation

The request transformation technique augments the SFC request primary topology, and the stand-by topology including the corresponding state-update paths into a single request graph. This enables the computation of the active and stand-by instance solutions in a single step, which ensures full coordination during the computation of the service orchestration solution. Such coordination is specifically vital for the stateful VNF placement problem since it involves information exchange between the active and stand-by instances, whose transfer costs are dependent of the placement of both the active and stand-by instances, necessitating full coordination between these two steps. Moreover, the transformation technique is well suited for a provisioning scenario in which the state update rates between the different VNF instances may be different and where the different VNFs of the SFC request can be provisioned on different DCs. An illustration of the request transformation step is shown in Fig. 6.3 in which the SFC request shown in Fig. 6.3(a) is shown with two possible transformations. The transformation in Fig. 6.3(b) is for a case in which the different VNFs of the SFC instance can be mapped on different DCs. In this case, the state update information is exchanged directly between the DCs hosting the corresponding VNFs. This permits flexible mapping in which the state update rates and intervals of the different VNFs can be accommodated. Fig. 6.3(b) corresponds to a scenario in which an entire SFC instance is deployed on a single DC.

6.4.2 Encoding Scheme

In order to solve the considered problem using the proposed metaheuristic algorithm, the chapter adopts a problem-specific representation [155], where each potential assigment solution is depicted using a permutation vector of $|N_s|$ (number of DCs in a substrate network) elements in which we have to choose $2 \times |N_v|$ (number of active instances in an SFC, and number of stand-by instances per VNF) elements with repetition. Every permutation vector (chromosome and harmony for GA and HS, respectively) consists of several positions (genes and notes for GA and HS, respectively). Each position value is represented by an integer in [0, $|N_s| - 1$] interval and corresponds to a physical node where a VNF is provisioned. Permutation vectors could be created randomly or by a specific function and must satisfy the constraints of the optimization model. Figure 6.4 shows an example of

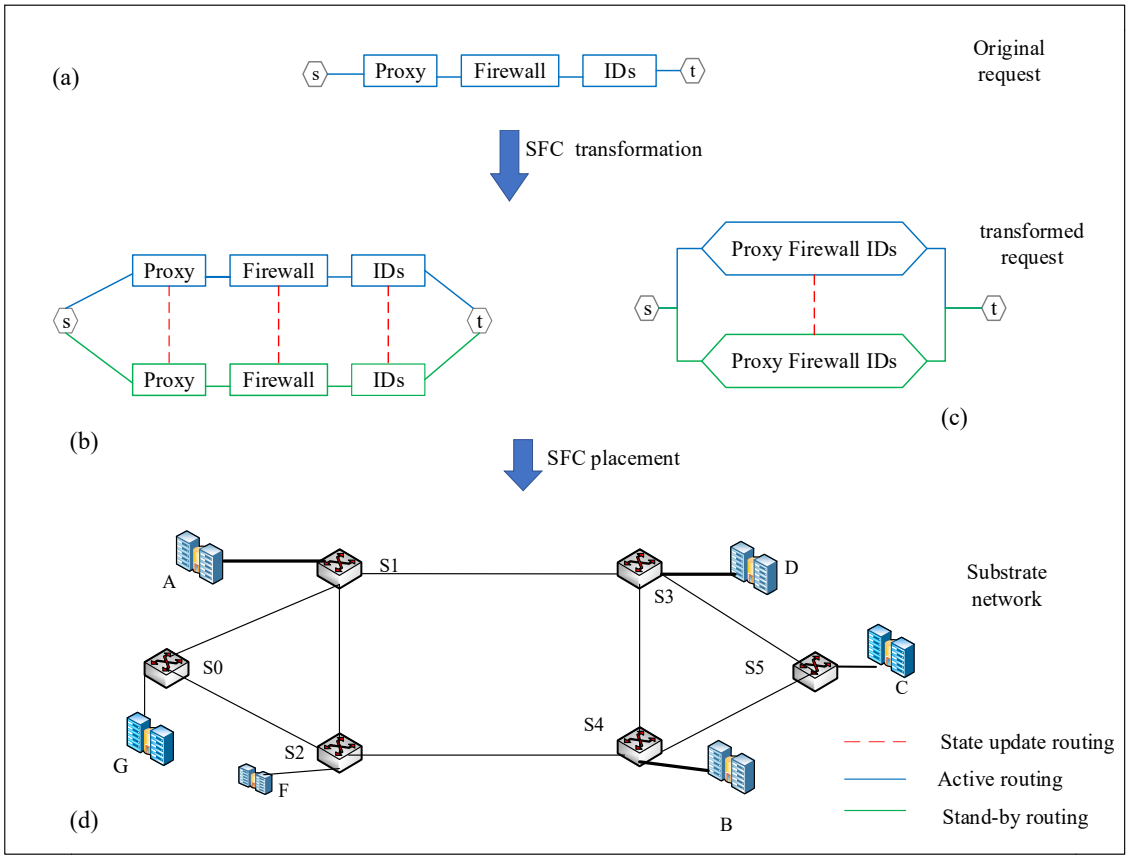


Figure 6.3: An illustration of the request transformation procedure for achieving coordinated node and link mapping

the encoding scheme for a sample service chain composed of 3 active instances of VNFs. Therefore, each permutation vector is composed of 6 positions, 3 for the active instances and 3 for the stand-by instances. In this example, the VNFs 0 to 2 (active instances) are mapped to the physical nodes 1, 2, and 5, whereas the VNFs 3 to 5 (stand-by instances) are assigned to the physical nodes 0, 3, and 4. A fixed number of chromosomes constitute the population as an initial set of solutions.

6.4.3 Initialization Functions

All metaheuristic optimization algorithms require some initialization, and the initialization for such algorithms is usually carried out randomly. However, initialization can have some significant influence on the performance of such algorithms [156]. In this thesis,two initialization functions for the metaheuristic algorithm are proposed, one based on a random selection and another based on node ranking considering the transformed request. For the basic version of the metaheuristic algorithms, the initial solutions are randomly generated, i.e., for each VNF in N_v, a physical node is randomly selected from N_s to host its active instance; for this, all the physical nodes in N_s are considered to have the same probabilities to be selected. In order to ensure that the nodes selected

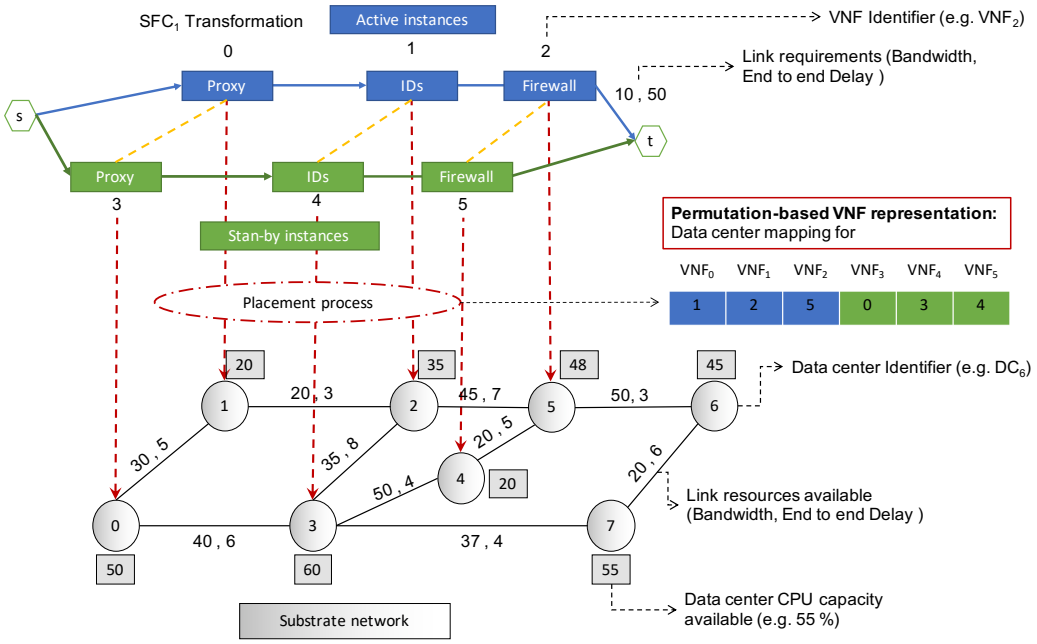


Figure 6.4: Permutation-based SFC representation. In this case, active instances=3 and stand-by instances per VNF=1

for the active instances are not reassigned to the stand-by instances, once all the nodes for the active instance are selected, the selected nodes are removed from N_s, before the process is repeated for the stand-by instances, as shown in the algorithm 12. Although easy to implement and fast to execute, such an approach lacks coordination between mapping of the active and stand-by instances which may result in high state-update costs and request rejection rate due to high probabilities of selecting unfeasible nodes during the initialization process.

Algorithm 10 Pseudocode of random initialization.

Input: $n, N_s, |N_v|$ // n No. of candidate solutions
// N_s Set of physical nodes,
// $|N_v|$ Number of VNFs in the service function chain.
Output: A set of n permutation vectors (initial solutions)
Step 1 Initialize the $InitialSolutions$ list. $InitialSolutions \leftarrow []$
for $i = 1$ *to* n **do**
  Step 2 initialize a $candidateSolution[2|N_v|]$ vector with zeros.
  Step 3 Randomly select the physical nodes from N_s for the active instances.
  for $j = 0$ *to* $|N_v| - 1$ **do**
    $position =$ select a random number from 0 to $|N_s|$.
    $candidateSolution[j] = N_s[position]$
  end
  Step 4 Remove the physical nodes that has just been selected for the active instances from N_s.
  Step 5 Randomly select the physical nodes from N_s for the stand-by instances.
  for $j = |N_v|$ *to* $(2|N_v|) - 1$ **do**
    $position =$ select a random number from 0 to $|N_s|$.
    $candidateSolution[j] = N_s[position]$
  end
  Step 6 add the $candidateSolution$ vector to the $InitialSolutions$ list.
end
Return $InitialSolutions$ list with n permutation vector.

Algorithm 11 Pseudocode of node ranking initialization.

Input: $n, N_s, E_s, |N_v|$ // n No. of candidate solutions
// N_s Set of physical nodes,
// E_s Set of physical links,
// $|N_v|$ Number of VNFs in the service function chain.
Output: A set of n permutation vectors (initial solutions)
Step 1 Initialize the $InitialSolutions$ list. $InitialSolutions \leftarrow []$
for $i = 1$ *to* n **do**
  Step 2 initialize a $candidateSolution[2|N_v|]$ vector with zeros.
  for $j = 0$ *to* $|N_v| - 1$ **do**
    Step 3 Construct a node candidate list (CNL_j), in which every physical node must meet the resource constraint of the VNF_j.
    Step 4 Select the physical node from the CNL_j according to the NR value of the physical nodes. The probability of select a physical node k is given by $\frac{NR_k}{\sum_{h=1}^{m} NR_h}$, where m is the total number of candidate nodes in CNL_j. $candidateSolution[j] = CNL_j[k]$
  end
  Step 5 Remove the physical nodes that has just been selected for the active instances from N_s.
  Step 6 Repeat the step 3 and 4 for each stand-by instance.
  Step 7 add the $candidateSolution$ vector to the $InitialSolutions$ list.
end
Return $InitialSolutions$ list with n permutation vector.

On the other hand, taking the context of the problem into account, the continuous reservation and release of resources resulting from the arrival and departure of service function chains can produce congestion at certain nodes and links; this congestion may result in the physical network resources to be unbalanced and fragmented and hinder the physical network from accepting larger SFCs. Besides, because the permutation vector encodes the candidate mapping solutions without considering the link mapping stage, it may dissatisfy the connectivity constraints in the link mapping

stage. Therefore, we propose using an initialization function based on node ranking that reflects the computational resource and the bandwidth resource consumption of a node n_s simultaneously, given by the following equation:

$$R_{n_s} = \frac{C_{avl}^{n_s}}{\sum_{n_j \in E_n} H_{n_s}^{n_j}} \tag{6.9}$$

where R_{n_s} denotes the rank of a physical node n_s and $H_{n_s}^{n_j}$ denotes the number of hops between node n_s and an essential node $n_j \in E_n$ where E_n denotes a set of all essential nodes. Essential nodes are chosen as those physical nodes that directly influence the cost incurred if node n_s were to be chosen as the host for the VNF under consideration. In this way, a node is considered to be essential with respect to n_s when obtaining candidate nodes for VNF i if it meets any of the following conditions: $i)$ it is the source or ingress node of the service request. In this way, its a preferred that n_s results in low resource consumption when traffic flows from the ingress to egress node through n_s; $ii)$ a host node for the previously mapped VNF $i-1$. In this way, its preferred that n_s results in less bandwidth resource consumption to the node hosting the preceding VNF ; or $iii)$ a host for the active instance of the current VNF in case of obtaining the stand-by instance solution. This directly affects the state update cost between n_s and the host of the corresponding active instance. The term $C_{avl}^{n_s}$ denotes the remaining computational resources of n_s. The idea behind this initialization function is that the VNFs have a higher probability of being mapped to the physical nodes that have more available resources and result in low consumption of bandwidth resources as well as increasing the possibility of satisfying the virtual link's delay constraints. The initialization function proposed to feed the initial solutions of the metaheuristic algorithms is presented in Algorithm 13.

6.4.4 Constraint Management Approach

When using metaheuristics for the SFC placement problem, there can be DCs/nodes and paths on the substrate network that do not meet the QoS requirements of the VNF and virtual links respectively during the evaluation of the candidate solutions: Solutions that contain these kinds of DCs or paths are considered infeasible solutions. Search space regions with many infeasible solutions can influence the algorithm's performance in solving such problems; to address this issue, we adopt a penalty function that assigns penalty values to each infeasible solution according to the degree of fulfillment of the constraints so that, the more infeasible a candidate solution is, the higher the penalization it gets. Penalization values are used to place infeasible solutions a distance to the feasible region proportional to such values, enhancing the metaheuristic algorithms' performance in a search space with infeasible regions. Based on all the above, in order to compare two solutions the following criterion are followed:

1. A feasible solution is preferred over an infeasible one.
2. If both solutions are feasible, the one with the best objective value is preferred
3. If both solutions are infeasible, the one with the lowest penalization value is preferred.

6.4.5 Genetic Algorithm

GA is a popular metaheuristic that has been successfully used to solve a wide range of applications, including the combinatorial SFC placement problem [155]. During the search process, GA algorithms imitate the natural evolution, i.e., the solution of the current service chain proceeds towards a more optimal solution after each generation. In the proposed GA algorithms, each potential placement solution is defined as a chromosome (see Fig. 6.4). The GA algorithms start with an initial set of chromosomes (initial population), usually randomly generated. In this work, two versions of the GA algorithm are proposed based on the function used to initialize the population (random or node ranking initialization as introduced in Section 6.4.3). Given the initial population, the more suitable chromosomes that have possibilities of producing lower fitness function values (Expression 6.5) are selected and allowed to crossover for preserving their genes in succeeding generations. Some chromosomes are discarded to be unsuitable for producing lower fitness values. To do this, the tournament selection approach [157] is used to select the solutions that will be the next generation's parents, in which the chromosomes are compared taking into account the criterion's described in Section 6.4.4.

After selecting the chromosomes using a selection operator, they must be employed to create a new generation. In nature, the genes in the chromosomes of a male and a female are combined to produce a new chromosome. This is emulated by combining two chromosomes (parent solutions) to produce two new chromosomes (children solutions) in the GA algorithm. There are different techniques for the crossover operator in the literature; in this work however, the the single-point crossover is adopted [107]. In the single-point crossover, the chromosomes of two-parent solutions are swapped before and after a single point. Finally, the last GA operator is the mutation operator, in which one or multiple genes are altered after creating children's solutions according to a user-defined mutation probability. This probability should be set low, otherwise, the search will turn into a primitive random search. The pseudo code of GA customized for the SFC placement problem is shown in Algorithm 12. The GA algorithm starts with an initial population of chromosomes. Until the end of the stop criterion, this algorithm improves the population using the above-mentioned operators. Finally, the best chromosomes of all generations is returned.

6.4.6 Harmony Search Algorithm

The Harmony Search (HS) algorithm is a relatively new addition to the metaheuristics search techniques for solving discrete combinatorial optimization problems [**?**]. This algorithm is inspired by the jazz music, where the different musicians that are playing together are optimizing harmonies over time to achieve a fantastic harmony by aesthetic standards [**?**]. The analogy between improvisation and the SFC placement problem is as follows:

- Each musician corresponds to each decision variable (VNF assignment);
- Musical instruments pitch range corresponds to the decision variables value range (DCs in the substrate network);

- Musical harmony at a certain time corresponds to the solution vector at a certain iteration (the assignments of the VNFs, see Figure 6.4);
- The audience's aesthetics corresponds to the objective function (Expression 6.5).

Algorithm 12 Pseudocode of Genetic Algorithm (GA).

Input: n, p_c,p_m // n No. of candidate solutions
// p_c Crossover probability, p_m Mutation probability
// t_{max} Max number of iterations.
Output: the best chromosome of all populations
$t \leftarrow 0$
Step 1 Initialize population $P[n]$ randomly with n chromosomes.
while $t \leq t_{max}$ **do**
  $t \leftarrow t + 1$
  Step 2 Calculate aptitude $f(x)$ for each chromosomes $x \in P[n]$.
  Step 3 Select parents of next generation based on their aptitude.
  Step 4 Crossover selected partner based on p_c to generate new population.
  Step 5 Mutate the chromosomes of new generation based on p_m.
end
Return the best chromosome of all populations.

The harmony search algorithm has three parameters, the harmony memory size HMS, the harmony memory consideration rate HMCR, and the pitch adjustment rate PAR. In the first step, the harmony memory HM is initialized with HMS harmonies, where each harmony memory vector represents a candidate solution. Then iteratively, a new harmony is generated (improvised) using the following steps: memory consideration, pitch adjustment, or random consideration. In the memory consideration step, the value for a decision variable of the new harmony vector is taken uniformly at random from the corresponding values stored in the HM with an HMCR probability. A memory consideration step is followed by a pitch adjustment step with a PAR probability. With a probability of 1/2, the pitch adjustment increments the decision variable by 1; otherwise, it decrements the decision variable by 1. In case no memory consideration step is performed, i.e., with a 1-HMCR probability, a random consideration step is performed instead. This step assigns to the decision variable a value uniformly at random according to its possible range.

After improvising the new harmony, it is compared with the worst harmony in HM (using the criterion's from Section 6.4.4, if it is better than the worst harmony in HM, the worst solution is replaced by the new one. Otherwise, this new vector is ignored, i.e., there are no changes in HM. The iteration process is terminated when the maximum number of improvisations is reached. Finally, the best harmony memory is selected and is considered to be the best solution to the problem. Algorithm 13 shows the HS algorithm pseudocode. In this work, we propose two versions of the HS algorithm according to the function used to initialize the population (random or node ranking initialization see Section 6.4.3).

Algorithm 13 Pseudocode of harmony search (HS) VNE

Input: $hms, HM[hms], hmcr$, par,n
// hms No. of candidate solutions,
// $HM[hms]$ list of candidate solutions,
// $hmcr$ probability of selecting a note in HM,
// par probability of selecting a neighbor tone within,
n number of VNFs,
// t_{max} Max number of iterations.
$t \leftarrow 0$
Step 1 Initialize the $HM[hms]$ randomly with hms harmonies.
Step 2 Calculate the value of the evaluation function $f(x)$ for each harmony.
while $t \leq t_{max}$ **do**
  $t \leftarrow t + 1$
  Step 3 Sort HM according to the value $f(x)$.
  Step 4 improvise a new harmony p^{t+1} and calculate its value $f(x)$.
  $p^{t+1} = \emptyset$
  for $i = 1$ *to* n **do**
    $r = [0, 1]$ uniformly distributed.
    if $r \leq hmcr$ **then**
      p_n^{t+1} = select a value within HM
      if $r \leq par$ **then**
        take the next value, above or below the selected value.
      end
    end
    else
      p_n^{t+1} = allowed range of variables values out of HM
    end
  end
  Step 5 Update HM.
  if $f(p_n^{t+1}) > f(HM^{worst})$ **then**
    include p^{t+1} in HM and exclude HM^{worst} of HM
  end
end
Return the best harmony of all harmonies.

6.5 Performance Evaluation

This section presents the performance evaluation of the proposed algorithms including: the simulation settings, the benchmark algorithms, the simulation scenarios adopted, and a discussion of the obtained results.

6.5.1 Simulation Settings

For the evaluation of the proposed algorithms, we consider both real network topologies, namely Abilene and BIC [112] as adopted in [28], and synthetic topologies as adopted in [26, 27]. For the synthetic topologies, the number of DCs are varied from 20 to 120 depending on the scenario under consideration with an inter-DC connectivity probability of 0.2. The computing resources, link

Table 6.1: Simulation parameters for the Bi-stage graph and Metaheuristic based algorithms

Substrate Network:	
Parameter	Value
Number of nodes for synthetic topologies	20-120
Node CPU capacity	unif distrib.[60000,80000]
Link bandwidth capacity	unif distrib.[400, 800] Mbps
Link propagation delay	unif distrib.[2,5] milliseconds
Processing cost per 1GB	unif distrib.[$0.15, $0.22]
Transmission cost per 1 GB	unif distrib.[$0.05,$0.12]
processing delay at each VNF	unif distrib.[0.0045,0.3] milliseconds
β_w	$ 0.01
e_{max}	2735
e_{idle}	80.5
$\gamma_c^{n_v^p}$	$0.22 and
$\gamma_{bw}^{e_v}$	$0.12
Fragmentation penalty, α^{n_s}	$0.01
Service Request:	
Parameter	Value
Number of VNFs per request	unif.distrib.[2, 10]
Packet rate	unif.distrib.[400,4000]
Mean arrival rate	20
Delay requirement of request	unif.distrib.[10ms, 30ms]
Mean arrival rate	2-18 per 100 time units
Arrival distribution	Poisson
Life-time	Exponentially distributed with mean 500

bandwidth, link delay, the processing delay of a packet at each VNF, cost for data processing and transmission, the number of VNFs per request and the packet rate of each request are chosen to follow a uniform distribution. The specific values of these and other parameters used in the simulation are given in Table. 6.1. We consider 5 categories of network functions: Firewall, Proxy, NAT, DPI, and Load Balancers with their computing resource demands adopted from [113].

6.5.2 Considered simulation scenarios and benchmark algorithms

The performance evaluation of the proposed algorithms is made considering two major scenarios as discussed below:

Scenario 01: Single DC mapping

This scenario targets to analyse the performance of the proposed Bi-stage graph based algorithm (denoted as $Bi\text{-}stage$) against the benchmark algorithms while considering a case in which an entire SFC instance is mapped onto a single DC node. The proposed algorithm is compared against the following benchmark algorithms:

- A node-rank based algorithm ($Node\text{-}rank$) proposed in [26, 27]. This algorithm selects the DC for mapping the active instance based on its rank in terms of computational and bandwidth resources. Then, the feasible DC with the least cost to the above DC is chosen as the host for

the stand-by instance.

- A Greedy algorithm ($Greedy$). This algorithm greedily targets to minimise the bandwith resources used to provision both the active and stand instances by placing the active and stand-by instances on a path pair that results in the least mapping cost for these instances. Then, based on these selected paths, it computes the least cost path for transferring the state information between the active and stand-by instances.
- Brute-force algorithm ($Brute$). This algorithm on the-other hand checks for all possible mapping solutions, then, among all the feasible solutions, the solution with the least aggregate cost in terms of hosting the active instance, the state-update, and the stand-by instance is selected for mapping the request.

The results of this scenario are discussed in Section 6.5.3

Scenario 02: Meta-heuristic algorithms performance analysis

This scenario evaluates the performance of the proposed meta-heuristic based algorithms considering different experiments. The GA and HS algorithms incorporating a specific initialization (here after refereed to as $GA\text{-}SpecInit$ and $HS\text{-}SpecInit$ respectively) are compared against GA and HS algorithms based on random initialization (denoted as $GA\text{-}RandInit$ and $HS\text{-}RandInit$ respectively), in addition to a bandwidth greedy (denoted as $Bw\text{-}Greedy$) algorithms as benchmark algorithms. The $Bw\text{-}Greedy$ algorithm seeks to minimize the bandwidth consumption by minimizing the number of substrate edges used for mapping both the active and stand-by instances of the request. The pseudo-code of $BW\text{-}Greedy$ is shown in algorithm 14. First the algorithm computes a set $Path_K^{\tau_s^r,\tau_d^r}$ containing the K shortest paths between the ingress and egress nodes with the paths being sorted in order of increasing length. Starting with the first path in the sorted list, the algorithm checks for its validity in terms of end-to-end delay, available bandwidth and computational resources. For a given valid path, the DCs along the path are sorted according to their residual computational resources from which the minimum number of DCs required to support the entire SFC instance is evaluated. Then, the VNFs of the request are placed on the obtained DCs along this path. Once the active instance has been placed, the same procedure is done for the stand-by instance. Then, the state update paths are computed based on the selected paths and nodes for the active and stand-by instances. Using the least number of DCs along a selected path is targeted to minimise bandwidth resource fragmentation due to state update paths.

Algorithm 14 Bw-Greedy Algorithm

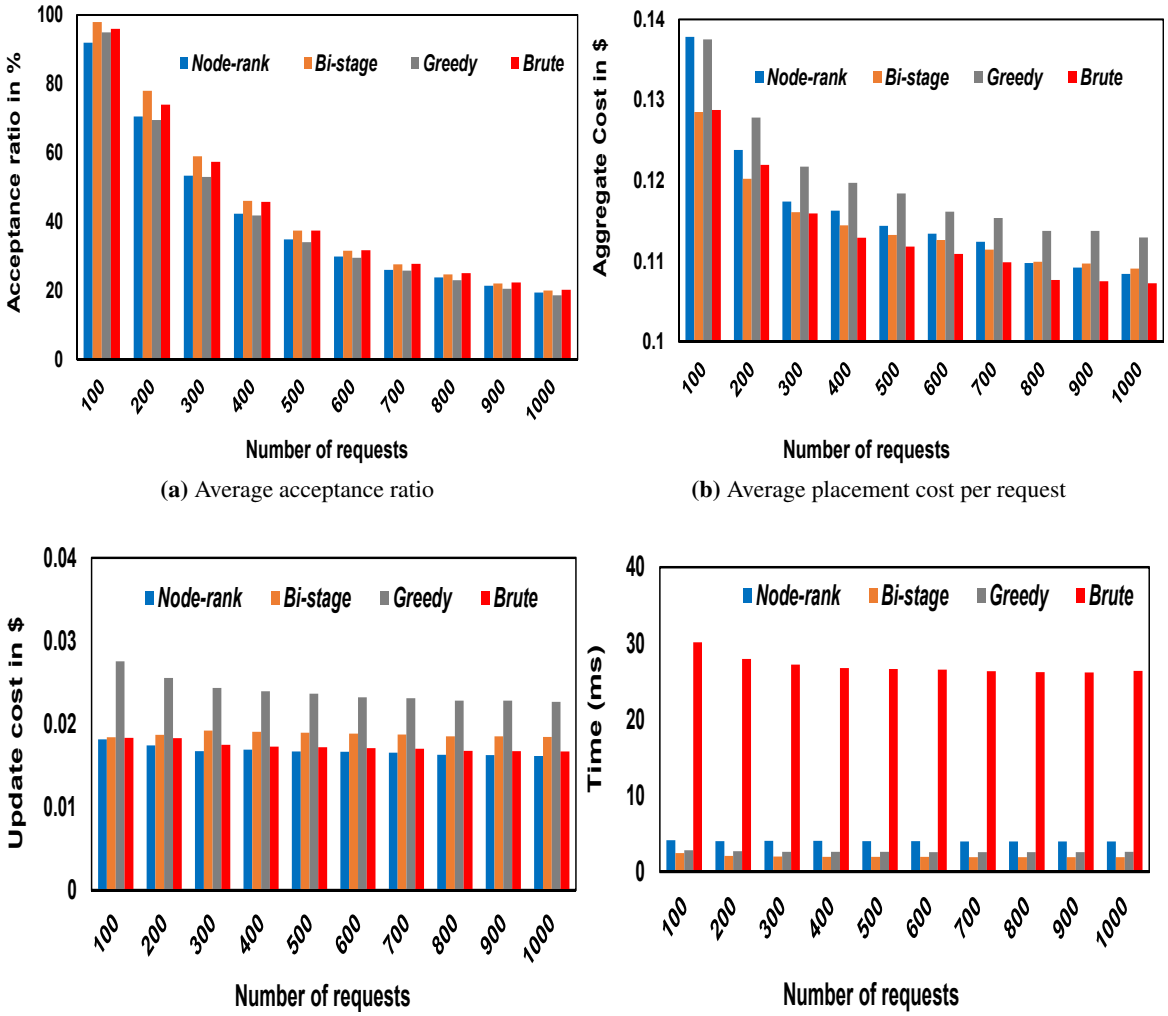
Input: $G_s, < G_v^r, C_{dem}^r, Bwt_{dem}^r, Del_{sd}^r, \tau_s^r, \tau_d^r, \tau_f^r >$
Initialise: $Path_{act}^r$=$Path_{std}^r$=$path_{sup}^r$=$None$
Compute $Path_K^{\tau_s^r, \tau_d^r}$.
if $|Path_K^{\tau_s^r, \tau_d^r}|$<2 **then**
  reject request
  return
end
for $path_k \in Path_K^{\tau_s^r, \tau_d^r}$ **do**
  check path validity
  if $path_k$ *is valid* **then**
    Extract the minimum host nodes
    Map the request
    if *mapping is feasible* **then**
      if $Path_{activ}^r$==$None$ **then**
        $Path_{activ}^r$=$path_k$
      end
      else
        $Path_{std}^r$=$path_k$
        Terminate for loop
      end
    end
  end
end
if $Path_{activ}^r$==$None$ *or* $Path_{std}^r$==$None$ **then**
  Reject request
  return
end
else
  Compute $path_{sup}^r$
  if $path_{sup}^r$!=$None$ **then**
    Return $Path_{activ}^r$, $Path_{std}^r$, $path_{sup}^r$
  end
  else
    Reject request
    return
  end
end

6.5.3 Results analysis

This section introduces the results obtained from the two considered scenarios. The different algorithms are compared in terms of acceptance ratio, average processing time per admitted request, request placement cost and average revenue per admitted request which metrics were introduced in Chapter 2 under Section 2.4.1.

Scenario 01: Single DC mapping.

Experiment 1 of this scenario analyses the impact of demand size by varying the number of offline requests from 100 to 1000 with the results shown in Fig. 6.5. From Fig. 6.5a, $Bi\text{-}stage$ results in the highest AR performance with an average value of 44.41% followed by $Brute$, $Node\text{-}rank$



(a) Average acceptance ratio

(b) Average placement cost per request

(c) Average state-update cost per request

(d) Average execution time per request

Figure 6.5: Results of the offline scenario with the number of users varied from 100 to 1000 considering 35 DC nodes

and $Greedy$ with average values of 43.74%, 41.32%, and 41.05% averaged across all demand size, demonstrating that the proposed $Bi\text{-}stage$ algorithm performs as good as the optimal (within 0.67% margin) in terms of AR. This performance is due to the fact that $Bi\text{-}stage$ algorithm coordinates the mapping of both the active and stand-by instances, resulting in less consumption of bandwidth resources. Although $Greedy$ targets to minimize the resource consumption by both the active and standby instances, these are deployed in uncoordinated manner, resulting in a high consumption of resources due to state update as manifested in Fig. 6.5c. Moreover, there is a possibility of failure to obtain a state-update path after mapping the active and stand-by instances, leading to rejection of requests. On the-other hand, $Node\text{-}rank$ targets to minimise the state-update cost but results in high resource consumption for placement of the active and standby instances. In terms of execution

time per accepted request as reflected in Fig. 6.5d, $Bi\text{-}stage$ results in up to 92.7% in terms of time saving with an average value of 1.93 milliseconds compared to 26.41 milliseconds for $Brute$ under the considered scenario. $Node\text{-}rank$ and $Greedy$ on average process each request in 3.99 milliseconds and 2.60 milliseconds respectively. These results demonstrate that $Bi\text{-}stage$ is able to achieve near-optimal results in practical execution time.

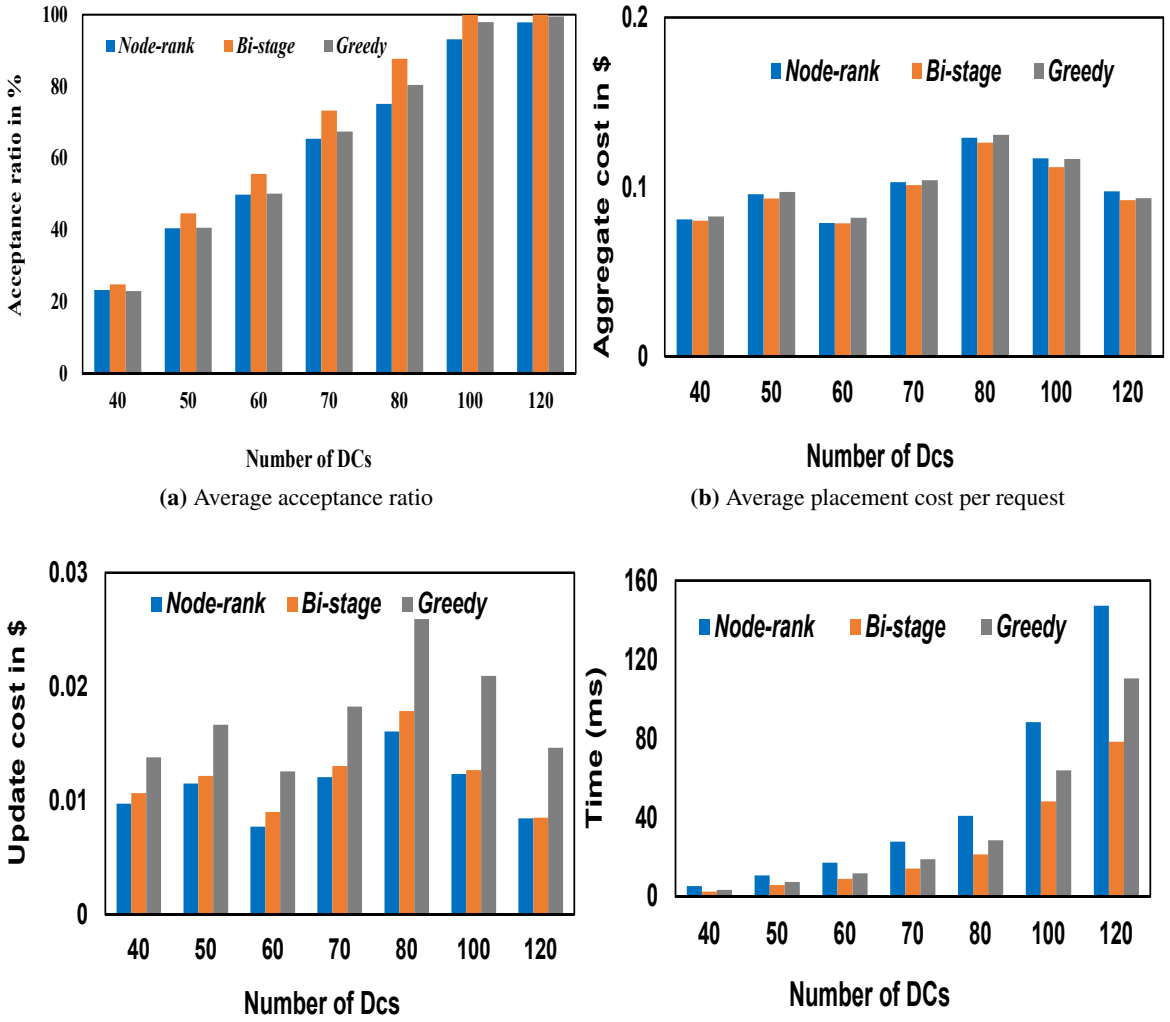
Experiment 2 whose results are shown in Fig. 6.6 analyses the impact of the number of DCs on the algorithms' performances. From Fig. 6.6a, the AR performance of all the algorithms increases as the number of DCs increases due to the increased resources in the network. $Bi\text{-}stage$ results in the highest AR value of 69.4% which is approximately a 6% and 4% improvement over $Node\text{-}rank$ and $Greedy$ respectively whose AR values are 63.5% and 65.5% respectively averaged across all arrival rates. Moreover, the AR performance of $Greedy$ improves over $Node\text{-}rank$ as the number of DCs increases, since the probability of failure to obtain a state-update path for $Greedy$ algorithm decreases due to increased link resources. From Fig. 6.6b, with an average mapping cost value of 0.09$, $Bi\text{-}stage$ results in a performance improvement of up to 10% over $Node\text{-}rank$ and $Greedy$ whose average mapping cost value for each admitted request is 0.1$ averaged across all DC substrate sizes. Fig. 6.6c, demonstrates that by greedily minimizing the active and stand-by resources, with an average value of approx. 0.02$, $Greedy$ results in high costs for state up-date with up to approx. 50% overhead over $Node\text{-}rank$ and $Bi\text{-}stage$ whose average state-update cost values are approximately 0.01$. This justifies the need to intelligently trade-off the different cost components. Moreover, from Fig. 6.6d, the proposed $Bi\text{-}stage$ algorithm demonstrates a faster execution time with an average execution time of 25.6 milliseconds which translates into up to a 46.9% and 26.6% improvement over $Node\text{-}rank$ and $Greedy$ respectively, whose average execution time per service request is 48.2 and 34.88 milliseconds respectively.

Experiment 3 whose results are shown in Fig. 6.7 analyses the impact of the optimisation parameters J and K on the performance of the $Bi\text{-}stage$ algorithm. From the results in Fig. 6.7a and 6.7b, increasing the J and K values from 3 to 35 results in an improvement of only 0.4% and 1.82% in terms of AR and average number of substrate edges used to map each request, while resulting in an increment of up-to 90.44% in terms of average execution time per admitted request as shown in Fig.6.7c. This performance behaviour is attributed to the weight sorting strategy adopted by $Bi\text{-}stage$ at the two stages of the graph that ensures that the least weight nodes are selected first during the solution computation. Therefore, the probability of obtaining better mapping solutions decreases as the values of J and K increases, hence enabling the algorithm to obtain near-optimal solutions even for small values of J and K parameters.

Scenario 02: Meta-heuristic performance Analysis

The results of scenario 02 considering a case in which the different VNFs of a service request can be mapped across different substrate nodes are discussed below:

In **Experiment 1** whose results are shown in Fig. 6.8 , the impact of demand size is analyzed by varying the arrival rates of requests from 2 to 12 requests per 100 time units for a total of 50000 time



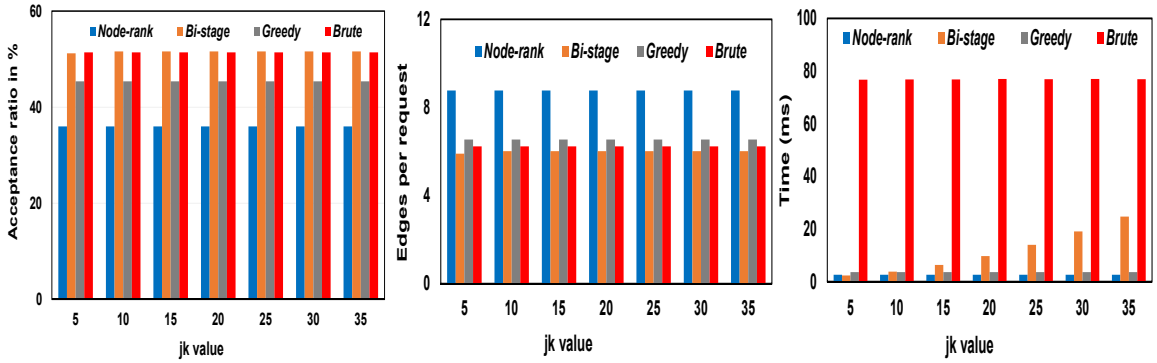
(a) Average acceptance ratio

(b) Average placement cost per request

(c) Average state update cost per request

(d) Average execution time per request

Figure 6.6: Results of the online scenario with the arrival rates of requests varied from 10 to 60 requests per 100 time units for a total of 10000 time units

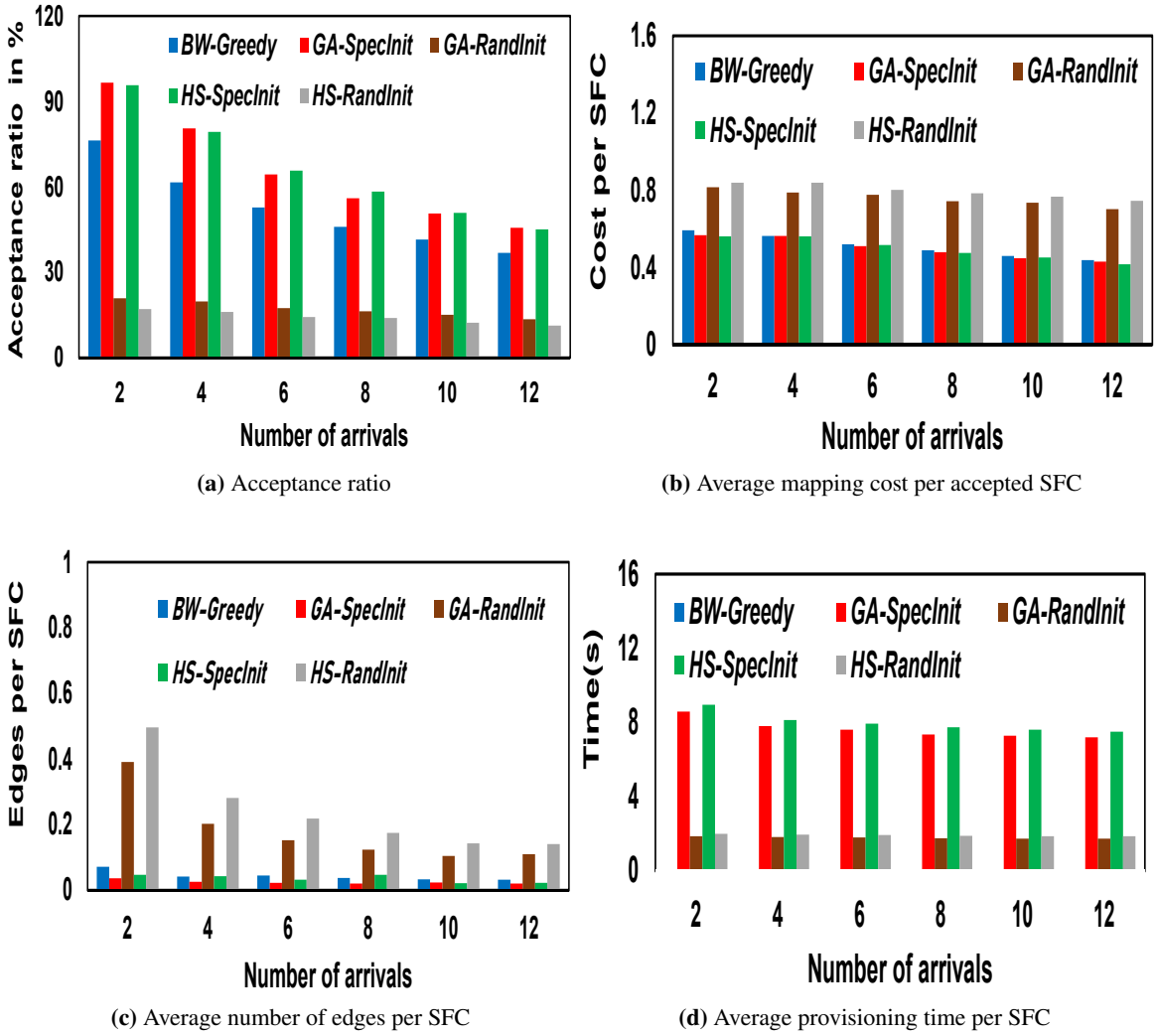


(a) Average acceptance ratio **(b)** Average number of edges per request **(c)** Average execution time per request

Figure 6.7: Results of Experiment 3 of scenario 01 in which the jk values are varied from 5 to 35 considering 500 requests and 40 DCs

units. This corresponds to a maximum number of 6000 requests. From Fig. 6.8a, both GA and HS with specific Initialization (i.e., $GA\text{-}SpecInit$ and $HS\text{-}SpecInit$ respectively) result in a similar performance in terms of AR with an average value of 65% averaged across all arrival rates. This corresponds to up to a 48% difference (approx. a 73% improvement) in terms of AR compared to the similar approaches with random Initialization whose average AR values are 17.1%, and 14.19% for $GA\text{-}RandInit$ and $HS\text{-}RandInit$ respectively and an improvement of up to 20% over $Bw\text{-}Greedy$ whose average AR value is 52.5%. From Fig. 6.8b, with an average value of 0.49$, $GA\text{-}SpecInit$ and $HS\text{-}SpecInit$ result in an improvement of up to 34.3% and 37.2% in terms of average mapping cost per admitted request over $GA\text{-}RandInit$ and $HS\text{-}RandInit$k,. whose average cost values are 0.76$ and 0.80$ respectively. $Bw\text{-}Greedy$ results in a provisioning cost of 0.51(approx. 4% overehead over $HS\text{-}SpecInit$ and $GA\text{-}SpecInit$). This is attributed to the fact that the Metaheuristic approaches with specific Initialization are able to deploy requests in a resource efficient manner while considering remaining CPU and bandwidth resources associated with the different nodes.

Experiment 2 whose results are shown in Fig. 6.9 analyses the impact of the size of the substrate network by varying the number of substrate nodes from 20 to 70 nodes. From the results in Fig. 6.9a, $GA\text{-}SpecInit$ and $HS\text{-}SpecInit$ result in a competitive performance in terms of AR with average values of 72.9% and 72.5% respectively. This corresponds to a performance improvement of approx. 78% over $GA\text{-}RandInit$ and $HS\text{-}RandInit$ whose average AR values are 15.9% and 13.1% respectively. The $Bw\text{-}Greedy$ algorithm results in AR value of 60.18% ie 18% less than the Metaheuristic approaches with specific initialization. In Fig. 6.9b, $GA\text{-}SpecInit$ and $HS\text{-}SpecInit$ result in the least provisioning cost per admitted request with average values of 0.43$ which is 31% compared to $GA\text{-}RandInit$ and $HS\text{-}RandInit$ whose average values are 0.64$ and 0.67$ respectively averaged across all substrate sizes. $Bw\text{-}Greedy$ results in average cost of 0.45$ which is approximately 2 % overhead compared to the Metaheuristic approaches with specific initialization. Moreover, from Fig.6.9c, the metaheuristc approaches result in approx. 2% better performance in



(a) Acceptance ratio

(b) Average mapping cost per accepted SFC

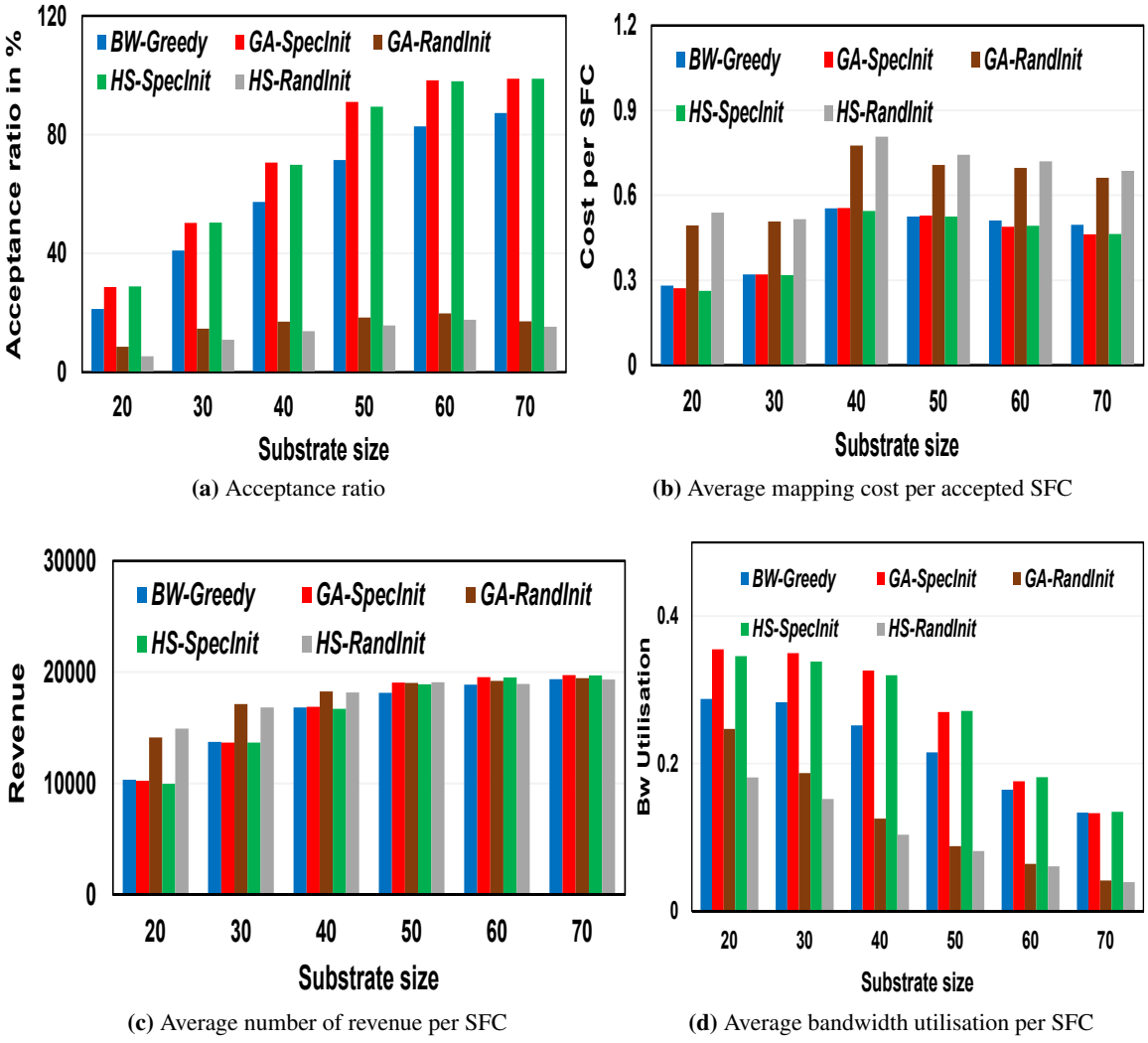
(c) Average number of edges per SFC

(d) Average provisioning time per SFC

Figure 6.8: Experiment 1 of Scenario 02: The impact of the arrival rate is evaluated by varying the arrival rate of SFCs from 2 to 12 arrivals per 100 time units for a total of 50000 time units

terms of average revenue per admitted request with an average value of 16515$ compared to 16202$ from $Bw\text{-}Greedy$.

In **Eperiment 3** whose results are shown in Fig. 6.10, the impact of the request size is analyzed by varying the number of VNFs per request from 2 to 12. From Fig. 6.10a, the acceptance ratio of all algorithms decreases with increase in request size as expected due to an increase in the amount of resources consumed by each request as the number of VNFs increases. Moreover, with increase in request size, the probability of obtaining a feasible solution for a given request decreases due to failure to meet the associated constraints such as delay and non-sharing of nodes between the active and stand-by instances. The average AR values for $GA\text{-}SpecInit$, $HS\text{-}SpecInit$, $GA\text{-}RandInit$, $HS\text{-}RandInit$ and $Bw\text{-}Greedy$ are 98.3%, 98.2%, 52.4%, 51.1% and 83.3%



(a) Acceptance ratio

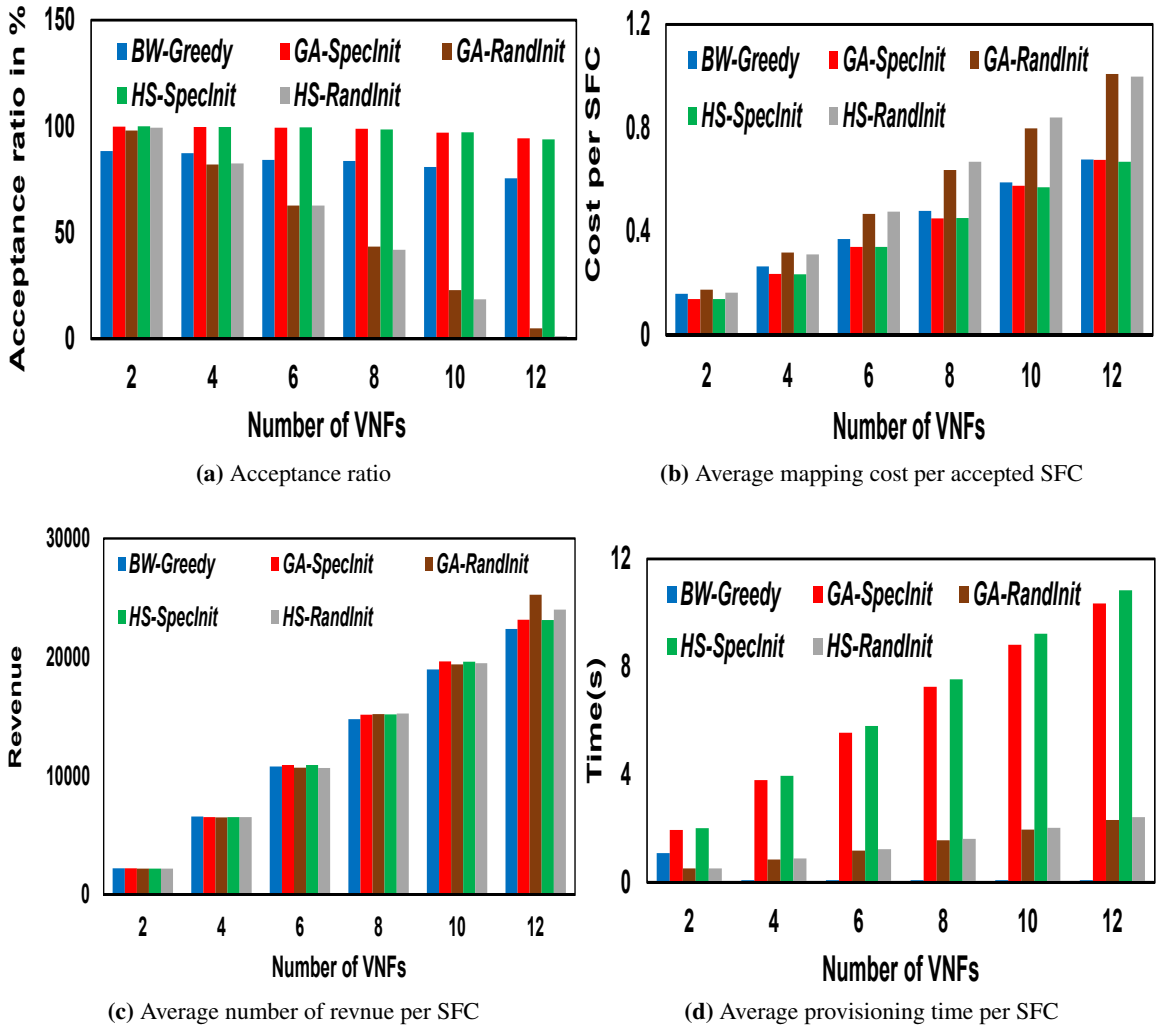
(b) Average mapping cost per accepted SFC

(c) Average number of revenue per SFC

(d) Average bandwidth utilisation per SFC

Figure 6.9: Experiment 2 of Scenario 2: The impact of substrate network size for online scenario

respectively, demonstrating that the Metaheuristic algorithms with specific initialization result in up to 36% improvement in terms of AR compared to their counterparts with random initialization and up to 15% improvement compared to the Bandwidth greedy algorithm $Bw\text{-}Greedy$. From Fig. 6.10b, the average cost for mappping each request increases with an increase in the number of VNFs across all algorithms due to an increase in the amount of resources consumed by each request. $GA\text{-}SpecInit$, $HS\text{-}SpecInit$ result in an average value of 0.4$ in terms of average mapping cost. This translates into approximately a 5%, 29.8%, 31.0% improvement over $Bw\text{-}Greedy$, $GA\text{-}RandInit$, $HS\text{-}RandInit$ whose cost value is 0.43$, 0.57$ and 0.58$ respectively averaged across all request sizes. Moreover, Fig. 6.10c reveals that the performance of the algorithms in terms of received revenue per admitted request is within a 2% margin for the different algorithms, demonstrating that the Metaheuristic algorithms are able to admit many requests including those associated with high resource requirements. On the low side, Fig. **??**



(a) Acceptance ratio

(b) Average mapping cost per accepted SFC

(c) Average number of revnue per SFC

(d) Average provisioning time per SFC

Figure 6.10: Experiment 2 of Scenario 2: The impact of request size for online scenario

6.6 Conclusion

This chapter has formulated the problem of cost effective and resource-efficient fault-tolerant orchestration of stateful VNFs considering a scenario in which different VNFs of a given SFC instance can be placed on different nodes (e.g., servers) of the substrate network. First, in order to achieve full coordination between the mapping of the active and stand-by instances of a given service request, the chapter has proposed a request augmentation technique in which the primary and stand-by instances of a request are combined, and jointly mapped as a single request graph. Then, a Genetic and Harmony Search Metaheuristic algorithms are proposed for solving the formulated problem. Simulation results demonstrate that the proposed Metaheuristic solutions with a specific solution initialization based on a node ranking approach results in up to a 78% and 34% difference in terms of AR and average mapping cost per admitted request respectively compared to similar approaches

with randomly initialized solutions. These also demonstrate a 20% and 5% improvement in terms of AR and request mapping cost compared to a greedy approach that targets to minimize bandwidth resource consumption.

Additionally, for a special case in which an entire SFC instance is required to be deployed in a single node such as server or DC, the chapter proposed a heuristic algorithm based on a Bi-stage graph whose AR performance is within a 0.67% margin compared to an optimal solution, with up to 92.7% time saving. In addition, the proposed Bi-stage algorithm results in more than 10% improvement in terms of average mapping cost per admitted request compared to a greedy and node-ranking based solutions.

CHAPTER 7

A Reinforcement Learning-based Service Orchestration with VNF Instances Sharing

7.1 Introduction

With a myriad of network service requests envisaged to share the scarce substrate resources, innovative approaches for the deployment of services, in a cost-effective and resource-efficient manner, without degrading the Quality-of-Service, are of utmost necessity, if the vision of 5G and the anticipated benefits of network virtualization are to be realized. Although not well explored, sharing VNF instances among multiple service requests provides a good alternative direction for reducing service deployment costs [158]. This is premised on the fact that future services will be implemented in the form of chained VNFs, with a number of these VNFs being common among different services and applications [2, 3]. For instance, Fig. 7.1 shows two SFC requests in which VNFs V3 and V6 are common between them. In the majority of existing works addressing the problem of service orchestration, such as [22, 28, 36], each instance of a VNF only processes the input traffic coming from a single service request. However, such an approach, although easy to implement, can result in a low resource utilization, especially since the input traffic may experience severe fluctuations [3, 158]. Aside from that, assigning VNFs in a dedicated manner may result in an excessive resource fragmentation, hence, leading to lower acceptance ratios of service requests, and higher overall service deployment costs. However, optimizing the service deployment cost using VNF sharing is a complex task due to the need to intelligently trade-off the involved cost components. Cost components like the transmission cost along the traversed links, the processing cost at the servers, the energy cost from both active and idle servers, or the deployment cost of instantiating new VNFs. This trade-off appears due to the fact that those costs may conflict in such a way that lowering one cost component will lead to raising another one. For instance, minimizing the transmission cost by naively deploying the service on the shortest path between the ingress and egress nodes may require the activation of new servers and VNF instances, resulting in an increase in VNF instantiation and energy costs, aside from promoting link bottlenecks. Conversely, minimizing the energy and VNF instantiation costs by reusing already deployed VNF instances and servers may result in an increased transmission cost and a poor node-load balancing, hence, affecting the long term performance of the acceptance ratio and the fault tolerance. As demonstrated by the results of the simulations, conventional approaches that greedily target to minimize only one cost component are less effective in general, since any request

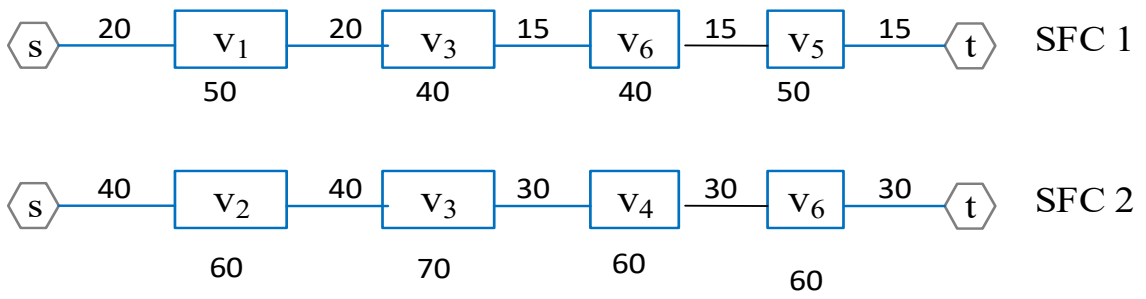


Figure 7.1: An illustration of two SFC requests each with 4 VNFs where VNFs V_3 and V_6 are common between the two requests

deployment is always conditioned by multiple costs. With this motivation, this chapter enhances the contributions of the previous chapters by tackling the problem of online orchestration of services from the perspective of sharing their VNF instances. First, a formal formulation of the service orchestration problem under VNFs instances sharing is introduced including the underlying resource and service constraints. Then, given the NP-hard nature of the above problem, the chapter proposes a reinforcement learning based algorithm capable of making cost effective and resource efficient service placement decisions while considering multiple conflicting costs. Costs of transmission, VNF instantiation or energy consumption, among others. As opposed to conventional heuristic approaches, a RL based approach is able to intelligently infer the effect of each placement decision on the long-term performance of the network, resulting in near-optimal solutions with less execution time.

The rest of the chapter is organised as follows: Section 7.2 presents a description and formulation of the service orchestration problem including the model of the virtual network functions. The proposed RL based algorithm for solving the optimization problem formulated in Section 7.2 is described in Section 7.3. Then, the performance evaluation of the proposed algorithm, including a description of the simulation scenarios, benchmark algorithms, and a discussion of the results obtained from the different simulations is presented in Section 7.4. Finally, the chapter is concluded in Section 7.5.

7.2 Virtual Network Function modelling and Problem description

This section introduces the intra-domain orchestration problem with VNF instance sharing. But first, the model of the VNFs is introduced in the subsection below.

7.2.1 Virtual Network Functions Model

This chapter envisages a NFV environment in which the different network functions, such as IDs and Firewalls, among others, have been virtualised and provisioned by the underlying physical

infrastructure, from which they are assigned the required node resources upon activation. Moreover, we assume multiple instances of a given VNF type and denote by M the set of all VNFs in the system. We consider each VNF $m \in M$ of type p to be characterised by assigned resources in terms of type $q \in Q$, denoted by C^m_{vnf}, a deployment cost which captures the cost of the image transfer and booting of that VNF, denoted by δ^p_{vnf}, the processing capacity which denotes the maximum packet rate the VNF can process, denoted by ϑ^m_{vnf}, the cost for processing each unit of packet rate at this VNF if the VNF is provisioned on node $n_s \in N_s$, denoted by ζ^{m,n_s}_{vnf}, the average processing delay experienced by a packet when processed by the VNF, denoted by del^m_{vnf}, and a set of substrate nodes on which such VNF can be provisioned, denoted by Υ^p_{vnf}.

7.2.2 Description and formulation of the service orchestration problem with VNF sharing

The problem addressed in this Chapter involves obtaining a mapping from the SFC graph G^r_v to a subset of the substrate network graph G_s that minimize the operational cost incurred by a NSP thanks to minimizing the implementation cost of the requests, with a requirement that all the constraints associated with the service request and the substrate network are respected. The implementation cost of a request is considered to be influenced by the following cost components:

- Energy consumption cost associated with running a VNF on a given node.
- Communication/forwarding/transmission cost of transferring the user traffic from the ingress node to the egress node along the intermediate links.
- Processing cost incurred for processing the user traffic at the different VNFs traversed by the traffic.
- Cost of deploying new VNF instances.
- Cost due to the fragmentation of substrate network resources.

The thesis argues for the selection of substrate nodes and links for provisioning the request in a manner that jointly considers all the above cost components, since in practice, the benefit obtained by greedily optimising a single component may be offset by an increased cost of another component(s). To illustrate this claim, consider Fig. 7.2 which shows an example of services deployment with VNF sharing that considers online service request arrivals. At time $t1$, the placement agent receives request $SFC\ 1$ and places the whole SFC instance on Data Center (DC) D. Then, at time $t2$, the agent receives another request $SFC\ 2$ with ingress and egress nodes $s2$ and $s5$ respectively, and requesting the same sequence of VNFs as in $SFC\ 1$. Considering VNF sharing, the $SFC\ 2$ would have to be placed on DC D to reuse the VNF instances from $SFC\ 1$, as shown by the blue dotted line in the figure. This would result in the allocation of three inter-DC links for $SFC\ 2$, but with the advantage of not having to activate any new VNF instance. On the other hand, considering the shortest path routing, $SFC\ 2$ would have to be placed on DC B, activating the corresponding new VNF instances, and using only two inter-DC edges for this request as shown by the red dotted line. Depending on the cost associated with VNF activation, bandwidth usage, energy cost and resource fragmentation,

the two approaches will most likely result in different implementation and operational costs. This example illustrates the need for an approach that intelligently trades off the different cost components if we want to optimize the overall placement objective. The placement objective in this chapter is formulated to minimise the average implementation cost of each request as follows:

$$Minimise: \quad \frac{1}{|\mathbb{R}|}\sum_{r\in\mathbb{R}} C(G_v)^r \tag{7.1}$$

where $C(G_v)^r$ denotes the implementation cost of request $r \in \mathbb{R}$, and it is evaluated as:

$$C(G_v)^r = C^r_{proc} + C^r_{fwd} + C^r_{enrg} + C^r_{depl} + C^r_{frag} \tag{7.2}$$

where C^r_{proc}, C^r_{fwd}, C^r_{enrg}, C^r_{depl} and C^r_{frag} denote the processing cost, the traffic forwarding cost, the energy cost, the VNF deployment cost and the resource fragmentation cost, respectively. Those costs are detailed below.

1) Processing cost: The processing cost C^r_{proc} incurred by a request $r \in \mathbb{R}$ is evaluated as:

$$C^r_{proc} = \sum_{n^p_v \in N_v} \gamma^{n^p_v,r}_{n_s} \times \zeta^{m,n_s}_{vnf} \times \rho^r_{n^p_v} \tag{7.3}$$

where the binary variable $\gamma^{n^p_v,r}_{n_s} \in \{0,1\}$ is equal to 1 if virtual node $n^p_v \in N_v$ of request $r \in \mathbb{R}$ is provisioned on substrate node n_s, zero otherwise, with $\rho^r_{n^p_v}$ and ζ^{m,n_s}_{vnf} denoting the traffic rate to be processed by n^p_v and the processing cost for each traffic rate unit at node n_s respectively. Note that in this work, we allow the processing cost per unit of traffic rate to be different across different physical nodes, even for the same VNF type. The reason for this design choice is twofold; first, this is permissible in practice since different physical nodes could belong to different providers who may have different billing policies. Moreover, different nodes may be characterised by different QoS guarantees in terms of service reliability and holding priority, justifying the different billing rates. But most importantly, such a design choice introduces an extra degree of freedom for selecting the physical nodes on which to provision the request virtual nodes. This fact increases the complexity of the placement problem, and somehow justifies the need to use RL algorithms in front of greedy approaches in order to solve the problem in a satisfactory way.

2) VNF deployment Cost: The cost C^r_{depl} is incurred whenever new VNFs are activated in order to provision the request $r \in \mathbb{R}$. This cost is evaluated as:

$$C^r_{depl} = \sum_{m\in M|\chi^m=1} z_{mp} \times \delta^p_{vnf} \times \chi^\tau_m \tag{7.4}$$

where $\chi^m \in \{0,1\}$ is a binary variable, equal to 1 if VNF $m \in M$ is active, zero otherwise; and $\chi^\tau_m \in \{0,1\}$ is a binary variable, equal to 1 if the state (i.e., active or inactive) of VNF m in the current provisioning event is different from the previous provisioning event. $z_{mp} \in \{0,1\}$ = 1 if VNF m is of type $p \in P$ and δ^p_{vnf} is the deployment cost for a VNF of type p.

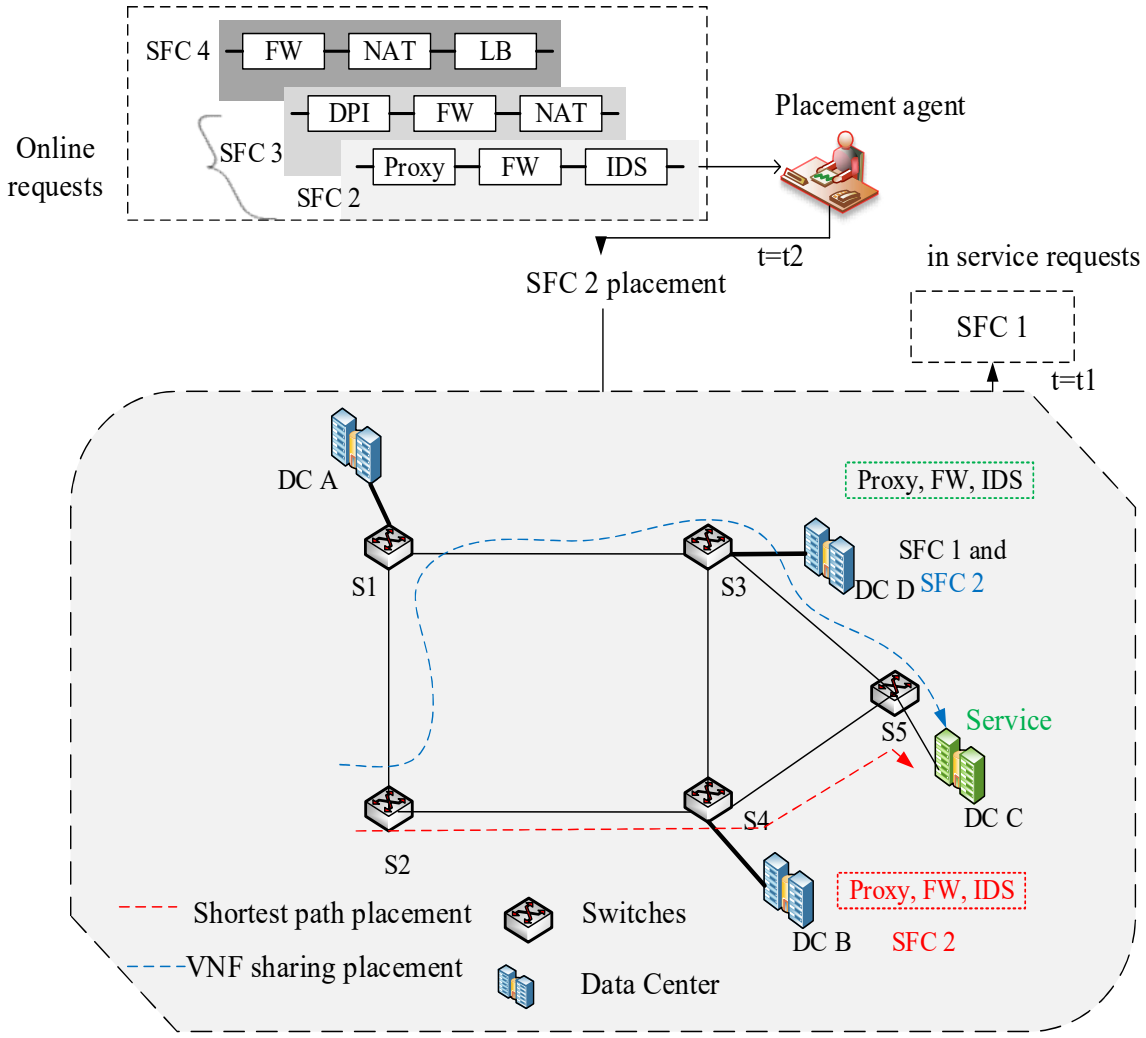


Figure 7.2: An illustration of SFC deployment with VNF instance sharing

3) Energy cost: In order to evaluate the energy cost C^r_{enrg}, we consider the energy consumption of a node to consist of two components [83]: the active state component, which is proportional to the amount of node resources being used; and the idle state component, which is the energy consumption in the idle state. In this regard, the energy consumption of a node n_s is computed as:

$$E_{n_s} = \sum_{m \in M} y_m^{n_s} \times \chi^m \times (e_{max} - e_{id}) \times \frac{Cpu_{vnf}^m}{C_{max}^{n_s}} + e_{id} \tag{7.5}$$

where $y_m^{n_s} \in \{0,1\} = 1$ if VNF m is provisioned on physical node $n_s \in N_s$, zero otherwise. $C_{max}^{n_s}$ and Cpu_{vnf}^m denote the CPU capacity of node n_s and the CPU allocated to VNF m from node n_s respectively. The parameters e_{max} and e_{id} denote the energy consumption in the peak consumption state and the idle state, respectively. If we denote by E'_{n_s} the energy consumption at n_s prior to provisioning the request $r \in \mathbb{R}$, and denote by E''_{n_s} the energy consumption at n_s after provisioning

request r, then, the energy consumption at n_s due to the current request is evaluated as: $E^r_{n_s}$=$E^{''}_{n_s}$-$E^{'}_{n_s}$. Therefore, the energy cost C^r_{enrg} across all nodes is evaluated as:

$$C^r_{enrg} = \beta_w \sum_{n_s \in N_s} E^r_{n_s} \tag{7.6}$$

where β_w is the cost per unit of energy consumption.

4) Forwarding cost: This cost is incurred along the substrate edges traversed by the traffic from the ingress to the egress nodes of the request. Therefore, this cost component increases as the number of used substrate edges increases. This is evaluated as:

$$\sum_{l_{uv} \in L_v} \sum_{e \in E} \sigma^r_{uv,e} \times \rho^r_u \times \zeta^e \tag{7.7}$$

where the binary variable $\sigma^r_{uv,e} \in \{0, 1\}$ is equal to 1 if substrate edge $e \in E$ is part of the substrate path provisioning the virtual link uv, zero otherwise. ρ^r_u and ζ^e denote the packet rate traversing link $e \in E$ and the cost per packet rate unit on $e \in E$.

5) Resource fragmentation cost: An envisaged key factor of the use of NFV comes from the flexibility we will have for sharing different network resources among multiple traffic flows and services. In this work, we evaluate the resource sharing capacity of the different algorithms in terms of resource fragmentation. Resource fragmentation can occur at the node level as a result of activating new nodes when there are others underutilised, or can occur at VNF level by deploying new VNF instances when there are other active instances underutilised. Therefore, fragmentation cost at node n_s can be evaluated as:

$$C_{frag} = F_{n_s} + F_{vnf} \tag{7.8}$$

where F_{n_s} is the fragmentation cost of the physical servers and F_{vnf} is the fragmentation cost of the VNFs (including their deployment platforms e.g., virtual machines and containers). The fragmentation cost across all nodes is evaluated as:

$$F_{n_s} = \sum_{n_s \in N_s | \lambda^{n_s} = 1} \alpha^{n_s} (C^{n_s}_{max} - \sum_{m \in M} \chi^m Cpu^m_{vnf} y^{n_s}_m) \tag{7.9}$$

where α^{n_s} is the fragmentation penalty for each unit of unused CPU resource on that node. The binary variable $\lambda^{n_s} \in \{0, 1\}$ = 1 if the node n_s is active, zero otherwise. A node is considered active if there is at least one active VNF provisioned by that node, thus :

$$\sum_{m \in M} y^{n_s}_m \chi^m \geq 1 \tag{7.10}$$

The VNF fragmentation cost relates to the amount of CPU resources allocated to the active

VNFs that is not used by the request virtual nodes. This is evaluated as below:

$$F_{vnf} = \alpha^{vnf} \sum_{m \in M} (\chi^m Cpu^m_{vnf} - \sum_{r \in \mathbb{R}} \sum_{n^p_v \in N_v} q^r_{n^p_v,m} C_{dem} n^{pr}_v) \tag{7.11}$$

where $q^r_{n^p_v,m} \in \{0,1\}$ = 1 if virtual node $n^p_v \in N_v$ is provisioned by VNF $m \in M$. The parameters α^{vnf}, Cpu^m_{vnf} and $C_{dem} n^{pr}_v$ denote the fragmentation penalty for each unit of unused VNF CPU resource, the amount of CPU allocated to VNF $m \in M$ and the amount of CPU resources consumed from VNF m by virtual node n^p_v of request $r \in \mathbb{R}$.

In addition to the constraints in Eqns. 2.6- 2.13 related to resource, location, flow and domain requirements as given in chapter 2, the optimization expressed in Eqn. 7.1 is achieved under the following additional constraints:

• The CPU consumption by the VNFs provisioned on a physical node $n_s \in N_s$ should not exceed the resource capacity of that node.

$$\sum_{m \in M | \chi^m = 1} Cpu^m_{vnf} y^{n_s}_m \leq C^{n_s}_{max} \ \ \forall n_s \in N_s \tag{7.12}$$

where $y^{n_s}_m \in \{0,1\}$ = 1 if VNF m is provisioned on node $n_s \in N_s$

• The amount of traffic going through a given VNF $m \in M$ should not exceed the VNF processing capacity:

$$\sum_{r \in \mathbb{R}} \sum_{n^p_v \in N_v} q^r_{n^p_v,m} \rho^r_{n^p_v} \leq \vartheta^m_{vnf} \ \ \forall m \in M \tag{7.13}$$

where $q^r_{n^p_v,m} \in \{0,1\}$ = 1 if the virtual node $n^p_v \in N_v$ of request r is provisioned by VNF $m \in M$.

• Each VNF of type p should be provisioned on a substrate node capable of supporting that VNF type:

$$y^{n_s}_m \times z_{mp} = 1 \ iff \ \ n_s \in \Upsilon^p_{vnf} \ \ \forall m \in M, p \in P \tag{7.14}$$

where Υ^p_{vnf} is a set containing all nodes that can provision a VNF of type p.

• Similarly, each virtual node must be mapped onto a VNF of the same type:

$$f^p_{n^p_v} \times q^r_{n^p_v,m} = z_{mp} \ \ \forall n^p_v \in N_v, r \in \mathbb{R}, m \in M, p \in P \tag{7.15}$$

where $f^p_{n_v} \in \{0,1\}$ = 1 if virtual node $n^p_v \in N_v$ is of type p, zero otherwise.

• Every virtual node $n^p_v \in N_v$ must be provisioned by exactly one VNF:

$$\sum_{m \in M} q^r_{n^p_v,m} = 1 \ \forall n^p_v \in N_v, r \in \mathbb{R}, m \in M \tag{7.16}$$

• The end-to-end mapping delay should not exceed the acceptable delay of the request:

$$\sum_{uv \in L_v} \sum_{e \in E} \sigma^r_{uv,e} del^e + \sum_{n^p_v \in N_v} f^p_{n^p_v} \times del^p_{vnf} \leq Del^r_{sd} \ \ \forall r \in \mathbb{R} \tag{7.17}$$

where the first and second terms of equation 7.17 correspond to the propagation and processing delay, respectively.

The above problem can be solved by means of conventional solvers, such as Gurobi and CPLEX. However, given the NP-hard nature of the problem, such approaches are not well suited for delay sensitive applications envisaged in future networks due to their high run time even for medium sized networks. This motivates the use of heuristic approaches such as those based on node ranking, load balancing, and shortest paths computation, among others. However, although these approaches can execute in polynomial time, they are not well suited for scenarios such as the one considered in this chapter where the placement objective is jointly influenced by multiple attributes. This motivates the use of the proposed reinforcement learning approach that is able to intelligently infer the long term influence of each cost component to the placement objective, yielding near-optimal solutions in acceptable run times. The proposed algorithm is introduced in the section below.

7.3 Proposed Reinforcement Learning Based algorithm

This section describes the proposed RL-based orchestration algorithm including: its MDP modeling, the architecture of the policy neural network in charge of making the VNF assignments, and the training procedure of that neural network. These are discussed below.

7.3.1 Markov Decision Process Model

Similar to the approach in Chapter 3, we model the system as a Markovian Decision Process (MDP) defined by the tuple (S, A, P, R), where: S denotes the set of possible states of the system; A denotes the set of possible discrete actions to be taken, actions for the selection of a physical node to host a given VNF of any request; $P = P(s_{t+1}|s_t, a_t)$ denotes the transition probabilities from state s_t to state s_{t+1} after taking action a_t; and $R = R(s_t, s_{t+1}, a_t)$ denotes the reward obtained after taking action a_t from state s_t and transiting to state s_{t+1}. In this way, the goal of the RL agent is to learn a policy $\pi : S \rightarrow A$ which maximizes the expected return, $\mathbb{E}[Return]$, over all episodes, where the Return of an episode is computed as:

$$Return = \sum_{i=t}^{T} \gamma^{i-t} R(s_i, s_{i+1}, a_i) \tag{7.18}$$

where $R(s_i, s_{i+1}, a_i)$ is the reward received by the agent after taking action a_i in state s_i at step i. In this thesis chapter, we relate the reward signal with the placement objective expressed in equation 7.1. Consequently, by maximizing the reward signal, the RL agent will be able to minimise the operational expenditures of a SP. The parameters of the MDP tuple for the above working scenario are discussed below:

State space

The state, at any time, is defined by relevant features of the environment, features that will be used by the agent to infer the actions that will produce high rewards. Those features will come from the substrate network state and the present incoming request, as those features will impact, directly or indirectly, the upcoming rewards. In order to conform to the neural network architecture of the policy selecting the actions, which incorporates a convolutional layer , the system state is modelled as an image-like $|N_s| \times F_{n_s}$ feature matrix, where N_s and F_{n_s} denote the set of substrate nodes and the number of relevant features associated with each substrate node respectively. In order to avoid the need to retrain the proposed neural network for different test scenarios with less nodes than those used for the original training, we will use dummy nodes to assure fixed dimensions of the state matrix, since in practice, once learned, the internal structure of a neural network cannot be modified. In order to achieve this, the policy neural-network is trained using the maximum possible number of nodes. Then, for the testing phase, when the number of nodes is less than the ones used for training, we match the input matrix by appending dummy nodes with dummy feature vectors to reach the expected state size. For instance, if the policy network is trained with N nodes and the test scenario has M nodes, where $M < N$, the number of dummy nodes created in this case is N-M, with each node being assigned a vector of dummy features. The dummy features are obtained by providing the worst value of each feature, in order to make such dummy nodes less likely to be selected by the policy network. Moreover, in the worst event that a dummy node is assigned a high probability of being selected to host a given VNF, the filtering layer that we use at the output end of the architecture will be able to sieve out such a node.

For each request $r \in \mathbb{R}$ to be provisioned, the algorithm uses the neural network to make a placement decision for each VNF of the request, one at a time, starting with the VNF closest to the ingress node. Therefore, the number of decision epochs (hence the actions taken) for each request is equal to the number of VNFs of the request. For each virtual node n_v^p from request $r \in \mathbb{R}$ to be scheduled for placement, the following are the features associated with each node n_s of the substrate network to form the state matrix:

- the residual computing resources, evaluated as:

$$CPU_{n_v^p}^{n_s} = max(0, \frac{C_{cpu_{res}}^{n_s} - dem_{cpu}^{n_v^p,r}}{C_{cpu_{max}}^{n_s}}) \tag{7.19}$$

  where $C_{cpu_{res}}^{n_s}$ denotes the residual computing resources at n_s and $dem_{cpu}^{n_v^p,r}$ denotes the required CPU resources of the current virtual node. This feature guides the RL agent in selecting nodes with sufficient CPU resources and selection of more loaded nodes when beneficial in order to reduce node resource fragmentation.

- the cost for processing each unit of packet rate through a VNF provisioned on n_s, denoted by ζ_{vnf}^{m,n_s}. This feature relates to the processing cost component of the service deployment cost.
- Deployment cost of a type p VNF denoted as δ_{vnf}^{p}. This feature guides the agent regarding

the trade-off between activation of a new VNF instance and reuse of active ones in relation to other cost components.

- The cost for transmitting a packet rate unit ζ^e. This feature directly relates to the forwarding cost component.
- the number of edges of the shortest path between $n_s^{n_v-1}$ and n_s, denoted by $E_{n_s}^{n_v}$. Where $n_s^{n_v-1}$ denotes the substrate node used to map the virtual node preceding n_v. Note that $n_s^{n_v-1}$ = τ_s^r if it is the first virtual node to be mapped. This feature relates to the suitability of node n_s in terms of traffic forwarding cost contribution to the overall cost.
- the node delay computed as the delay of the shortest path between $n_s^{n_v-1}$ and n_s, denoted by $Delay^{n_s}$. For fairness and to conform to the approach adopted in the benchmark algorithm proposed in [83], we distribute the end-to-end delay requirement of the request among the different virtual links which we denote as inter-VNF delay. In this regard, $Delay^{n_s}$ is evaluated as:

$$Delay^{n_s} = max(0, \frac{Del^r - Del_{sd}^{n_s}}{\delta_{max}}) \tag{7.20}$$

where $Del_{sd}^{n_s}$ is the delay along the path from $n_s^{n_v-1}$ to n_s, while Del_{uv}^r is the inter-VNF delay of the request. δ_{max} is a normalisation term evaluated as the maximum delay between $n_s^{n_v-1}$ and all the alternative nodes for hosting the current request virtual node.

- the available (bottleneck) bandwidth on the shortest path from the source node to the terminal node going through n_s, denoted by $Bandwidth^{n_s}$ and computed as:

$$Bandwidth^{n_s} = max(0, \frac{Bw_{av} - Bwt_{dem}^r}{Bw_{max}}) \tag{7.21}$$

where Bw_{av} denotes the bottleneck bandwidth along the shortest path between τ_s^r and τ_d^r. Bwt_{dem}^r and Bw_{max} denote the request desired bandwidth and the maximum edge bandwidth respectively. This term guides the agent in selecting nodes that result in feasible paths to the egress node and also to trade-off between link load balance and mapping cost.

- A binary variable $k_p^{n_s} \in \{0, 1\} = 1$ if there is an active VNF of type p at n_s, zero otherwise. This guides the agent about the VNF activation status at a given node.

Action space

The proposed algorithm uses the neural network to select the nodes on which to place each VNF of the service request. In order to achieve this, each substrate node is assigned a unique identifier in the range $[1, |N_s|]$, where N_s denotes the set of all substrate nodes and $|N_s|$ is the cardinality of this set. Therefore, during each decision epoch, the action of the policy implemented by the neural network is $a \in A$, where the action space $A = \{0, 1, 2, 3....|N_s|\}$. The zero value of a corresponds to the case where no substrate node is selected for hosting a given virtual node, consequently, the request will be rejected. This will occur, for example, when all the nodes have been deemed infeasible by the filtering layer of the policy neural network.

Reward

We formulate the reward signal in such a manner that by maximising the reward, the agent optimises the deployment objective given in Eqn. 7.1. In this regard, the reward signal is formulated as:

$$reward = \frac{Rev(G_v)^r}{C(G_v)^r} \tag{7.22}$$

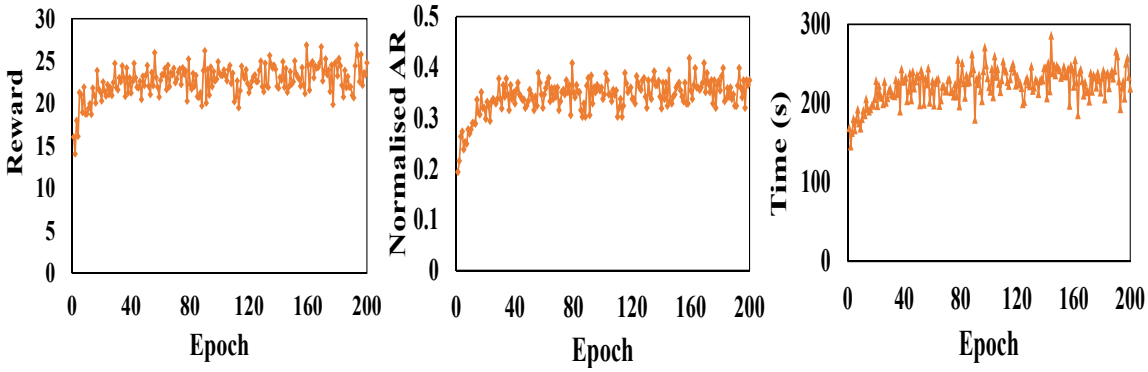
where $Rev(G_v)^r$ denotes the request revenue as given in Eqn. 7.23, and $C(G_v)^r$ denotes the implementation cost of request $r \in \mathbb{R}$ as given by Eqn. 7.2. Intuitively, Eqn. 7.22 is maximised by minimising the implementation cost of each request as desired by the deployment objective, but accepting those requests that are associated with high revenues, hence, increasing the net profit obtained by the service provider. This is different from the objective of only minimizing implementation costs. This way, the above reward formulation avoids biasing the RL agent towards requests with low resource requirements, in contrast with requests with high resource requirements, since the former are more likely to be associated with low provisioning costs, but they will also contribute less to the overall net profit. If we denote by $\gamma_c^{n_v^p}$ and $\gamma_{bw}^{e_v}$ the price charged by the service provider for processing and transmitting a unit of packet rate through virtual node $n_v^p \in N_v$ and virtual link l_{uv}, where l_{uv} is the inter-VNF path between u and v; then, the revenue $Rev(G_v)^r$ obtained from provisioning a request $r \in \mathbb{R}$ can be defined as:

$$Rev(G_v)^r = \begin{cases} \sum_{n_v^p \in N^v} \gamma_c^{n_v^p} \rho^r + \sum_{\forall e_v \in E_v} \gamma_{bw}^{e_v} \rho^r & \text{if } z_\tau^r\text{=1} \\ 0 \quad \text{otherwise} \end{cases} \tag{7.23}$$

where $z_\tau^r \in \{0, 1\}$ is a binary variable equal to 1 if request $r \in \mathbb{R}$ is assigned resources, zero otherwise.

7.3.2 Architecture of the Neural Network for policy evaluation

The proposed algorithm adopts a policy neural network whose architecture is similar to that introduced in Chapter 3 in Fig. 3.6 which incorporates an input layer, a convolutional layer, a softmax layer and a filtering layer. The convolutional layer performs a convolution between the input feature matrix and the internal weights of the layer to output a numerical vector of size $|N_s|$ where N_s denotes a set of possible substrate nodes for hosting a given VNF. The softmax layer then transforms the convolutional layer output into a vector of probabilities, where each element of the vector indicates the probability of the corresponding substrate node to be selected for hosting the VNF at hand. The filtering layer is added, at the end, to avoid unfeasible nodes from being selected, for instance, the dummy nodes. Once such unfeasible nodes are pruned, then, the substrate node with the highest probability is selected for hosting the corresponding VNF of the request.



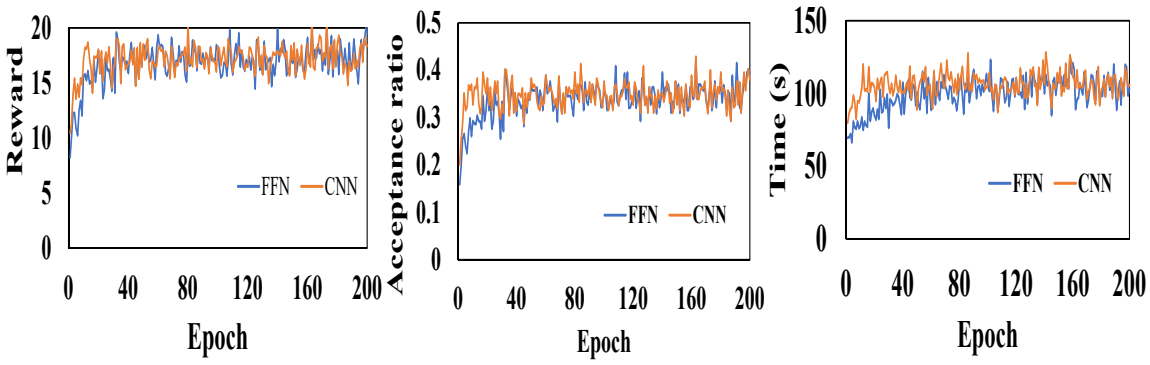
(a) Average reward per epoch **(b)** Average acceptance ratio **(c)** Average time per training epoch

Figure 7.3: Results of training step of the policy neural network

7.3.3 Neural Network training

The training of the neural network has been done using the maximum number of foreseeable substrate nodes. As mentioned in the previous section, the placement of a request is performed virtual node by virtual node, until all the nodes of the request have been provisioned, or until a given node cannot be provisioned and the request is rejected. For each virtual node, the policy neural network is giving the substrate node with the highest probability for serving that virtual node. However, since the neural network parameters are initially assigned randomly, during the training phase we perform a trade off between exploration and exploitation to determine the substrate node on which to provision the virtual node [159].

If a virtual node cannot be provisioned, the entire request is rejected, and a new request is scheduled for placement. Otherwise, the resultant reward is obtained according to the mapping decisions made for the different virtual nodes. This reward is used to calculate all the gradients of all the internal weights of the neural network applying back propagation. The gradients from different requests are stored in a buffer until a given batch size is reached. Then, all the gradients previously stacked are jointly applied to update the internal weights of the neural network, that done, the buffer is emptied. Note that, whereas it is possible to perform a gradient update for each successfully deployed request, adopting a batch processing strategy guarantees a faster and more stable training process. In the resource management domain, attributes, such as traffic load, residual resources, among others, are usually characterized by a certain predictable temporal correlation. Those repetitive patterns enable the agent to learn online as the system executes or offline by exploiting historical information. The thesis adopted the offline option in which the neural network was trained using offline demand sets of size 600 requests per epoch for a total of 200 epochs, considering a substrate network of 60 nodes. The results of the policy neural network training are shown in Fig. 7.3. From Figs. 7.3b and Fig. 7.3c, the acceptance ratio and execution time per epoch is seen to increase with the training time. This is because the policy network accuracy improves on increasing the training time, thus resulting



(a) Average reward per epoch **(b)** Average acceptance ratio **(c)** Average time per training epoch

Figure 7.4: Results of the training performance of the Feed-forward and convolutional neural networks

in a reduction in both wrong placement decisions and bandwidth resource consumption. Therefore, the number of admitted requests increases, consequently resulting in an increase in the execution time per training epoch, since admitted requests are characterized by additional steps such as the updating of the substrate resources and the computation of the mapping cost.

The performance of the CNN and FFN has recently been investigated in [160]. With a CNN model size which is 68 times smaller than the FFN, the CNN was found to result in a similar or better performance than the FFN when applied to the problem of speech enhancement. Such a good performance with fewer trainable parameters makes CNN based architectures more memory efficient, making them implementable even in memory constrained embedded systems. In Fig. 7.4, we compare the training performance of a convolutional neural network (CNN) with a feed forward network (FFN) architecture considering 50 substrate nodes and 400 offline demands per epoch. From the obtained results, both architectures converge to a similar performance in terms of training time, acceptance rate and reward value. Fig. 7.4a reveals that CNN converges earlier than FFN by approx. 5 epochs. This explains the slightly higher running time for the initial epochs in Fig. 7.4c, since the number of requests admitted by CNN is higher than those of FFN during this stage.

7.4 Performance Evaluation

This section first introduces the simulation parameters considered for the evaluation of the performance of the proposed algorithm. This is followed by a description of four algorithms used as a benchmark against the proposed RL algorithm. Finally, a discussion of the obtained results is given in the last sub-section.

7.4.1 Simulation setup and parameters

For the evaluation of the proposed algorithms, both real network topologies, in particular, the Abilene and BIC networks and synthetic topologies are considered. For the service request, five categories of network functions: Firewalls, Proxies, NATs, DPIs and Load Balancers are considered, with their computing resource demands adopted from [113]. The values of the different substrate network and service request parameters adopted in the simulation are indicated in Table. 7.1.

Table 7.1: Simulation Parameters for VNF sharing based algorithm

Substrate Network:

Parameter	Value
No. substrate nodes for synthetic topology	30-60
Node connection probability	0.2
Node CPU capacity, $C_{CPU^{n_s}_{max}}$	unif distrib.[60000, 80000]
Link bandwidth capacity, B^e_{max}	unif distrib.[400,800] Mbps
Link propagation delay, δ^e	unif distrib.[2,5] ms
Processing cost per 1 GB of data, ζ^{n_s}	unif distrib.[$0.15,$0.22]
Transmission cost per 1 GB of data, ζ^e	unif distrib.[$0.05,$0.12]
Processing delay of a packet at each VNF	unif distrib.[0.0045,0.3] ms
Cost per unit of energy consumption, β_w	$ 0.01
Energy consumption in peak state, e_{max}	2735
Energy consumption in idle statee_{idle}	80.5
Charge by SP per unit packet rate processing $\gamma^{n^p_v}_c$	$ 0.22
Charge by SP per unit packet rate transmission $\gamma^{e_v}_{bw}$	$ 0.12
Fragmentation penalty, α^{n_s}	$0.01
Packet rate, ρ^r	unif distrib.[400,4000] packets/s
Acceptable end-to-end delay	unif distrib.[10, 30] ms
No. of VNFs per request	unif distrib.[1,10]
Mean arrival rate	20
Life-time	exponential with 500 (mean)

7.4.2 Benchmark Algorithms

The proposed RL-based service deployment algorithm is compared with a state-of-the-art multi-stage graph based algorithm (denoted as $graph\text{-}based$ in this section) proposed in [83]. Similar to the thesis proposal, the work in [83] considers a detailed cost modeling incorporating the VNF deployment cost, the energy cost and the traffic forwarding cost to the problem of SFC deployment with VNF sharing. First, the problem is formulated as an Integer Liner Programming (ILP) problem. Then, a heuristic approach based on a multi-stage graph, including the Viterbi algorithm, is proposed to overcome the time complexity of the ILP problem.The heuristic maps one request at a time, hence, it is well suited for both offline and online scenarios. Moreover, the algorithm is shown to provide near optimal solutions, hence rendering a good candidate for performance comparison. In addition to the above work, the proposed algorithm is evaluated against a brute-force algorithm and three greedy algorithms which are well known in literature. These are discussed below:

Brute-force algorithm ($Brut$)

For a given request to be provisioned, the brute-force algorithm obtains all possible mapping combinations for that request. This is then followed by extracting all feasible solutions (i.e., solutions that do not violate the request and substrate network constraints). Then, from all feasible solutions, the solution that results in the least deployment cost is selected for provisioning that service request. Since such an approach explores all mapping possibilities, it results in an optimal solution, albeit with the penalty of the excessive run-time employed to obtain that one.

Bandwidth Greedy Algorithm (BwGA)

This algorithm targets to jointly minimise: the amount of bandwidth resources used to provision each request, the node resource fragmentation and the VNF deployment cost. This will be done by provisioning each request on the shortest feasible path between the ingress and the egress nodes and using the minimum number of nodes. The pseudo-code of BwGA is shown in Algorithm 15. First, the algorithm computes the set $Path_K^{\tau_s^r,\tau_d^r}$ that contains the K shortest paths between the ingress and egress nodes, with the paths being sorted in increasing length order. Starting from the first path in the sorted list, the algorithm checks for the validity of that path in terms of end-to-end delay, available bandwidth and availability of CPU resources along the path to map the different request virtual nodes. For a given valid path, the physical nodes along that path are sorted in increasing order of their residual resources. Then, the minimum number of nodes with the least amount of residual resources that are required to support the entire SFC instance are selected. Then, the VNFs of the request are placed on the deduced nodes along that path. The choice of the nodes with the least amount of residual resources for provisioning the virtual nodes targets to minimise the energy and resource fragmentation costs, since such nodes are more likely to have VNFs already active on them, preventing the activation of new nodes and VNFs.

Greedy activation algorithm (GAA)

This algorithm targets to minimize the deployment cost by greedily minimizing the VNF deployment cost thanks to reusing as much as possible already active VNF instances. If an instance of a given VNF is not active or the node resources are not sufficient to enable its sharing, then, the new virtual node is placed on a substrate node that is closest to the host of the preceding virtual node (or the ingress node in case of mapping the first virtual node). The pseudo-code of this algorithm is shown in Algorithm 16. For each virtual node to be mapped, the function $compute\_candidates()$ computes a set $Cands_{n_v^p}$ consisting of the candidate substrate nodes for virtual node $n_v^p \in N_v$. Then, if there are substrate nodes with already active VNFs to support n_v^p, among these, the node closest to the host of the preceding virtual node is chosen to provision the current virtual node. Otherwise, all candidates are sorted according to the distance to the host of the preceding virtual node, with the closest candidate being chosen to provision the current virtual node. Once all the virtual nodes are provisioned, then the host for the virtual links are computed using the shortest available paths between adjacent nodes of the selected substrate nodes.

Algorithm 15 BwGA Algorithm

Input: G_s, Ψ^r
Initialise: Deployment_solution =$None$
Compute $Path_K^{\tau_s^r,\tau_d^r}$.
if $|Path_K^{\tau_s^r,\tau_d^r}|$=*0* **then**
  reject request
  return
end
for $path_k \in Path_K^{\tau_s^r,\tau_d^r}$ **do**
  $check\_path\_validity(path_k)$
  if $path_k$ *is valid* **then**
    Extract_minimum_host_nodes($path_k$, Ψ^r)
    Map the request
    if *mapping is feasible* **then**
      Deployment solution=$path_k$
      Terminate for loop
    end
  end
end
if $Deployment\_solution$=*None* **then**
  Reject request
end
else
  Return Deployment_solution
end

Load balance based algorithm (LBA)

For each virtual node of the request to be provisioned, this algorithm ranks all candidate nodes according to their resource score, which is calculated as the product of the node's computational resources by the accumulative residual bandwidth of the node's inbound links. Then, the candidate node with the highest score is selected for hosting the current virtual node. In this way, the algorithm targets to minimise node and link resource bottlenecks, guaranteeing a good long term acceptance ratio performance. The score of a given node $n_s \in N_s$ is computed as:

$$Score^{n_s} = C_{res}^{n_s} \times \sum_{e \in E_{adj}^{n_s}} B_{res}^{e} \tag{7.24}$$

where $C_{res}^{n_s}$ denotes the available computational resources at the node and B_{res}^{e} denotes the residual bandwidth resources at an inbound edge $e \in E$. The pseudo-code of this algorithm is shown in Algorithm 17. Once all virtual nodes have been provisioned, then, the end-to-end traffic link is obtained by running the Dijkstra algorithm in between each pair of the hosting nodes, while updating the available edge resources.

Algorithm 16 GAA Algorithm

Input: G_s, Ψ^r
Output: $Deployment\_solution$
Initialise: $prev\_host$=τ_s^r; $node\_solution = []$
for $n_v^p \in N_v$ **do**
  $Cands_{n_v^p}$=$Compute\_candidates(G_s, \Psi^r)$
  if $Cands_{n_v^p}$=$\emptyset$ **then**
    Reject request
    break
  end
  else
    Extract $Cands_{n_v^p}^{act}$ from $Cands_{n_v^p}$
    if $Cands_{n_v^p}^{act} \neq \emptyset$ **then**
      $Cands$=$Sort\_dist(Cands_{n_v^p}^{act}, prev\_host)$ n_v^p←Cands[0]
      $node\_solution.append(Cands[0])$ $prev\_host$=Cands[0]
    end
    else
      Cands=$Sort\_dist(Cands_{n_v^p}, prev\_host)$
      n_v←Cands[0]
      $node\_solution.append(Cands[0])$ $prev\_host$=Cands[0]
    end
  end
end
Run $link\_mapping(node\_solution, G_s, \Psi^r)$
if *Sucessful* **then**
  Deploy request
end
else
  Reject request
end

7.4.3 Result Analysis

In this section, the performance of the proposed algorithm against the benchmark algorithms discussed in Section 7.4.2 is analyzed in terms of: average acceptance ratio of requests , average deployment cost per accepted request, average processing time of a request, and average bandwidth utilization as introduced in Section 2.4.1. The results obtained for the different simulation scenarios are discussed below:

Performance with respect to the optimal solution

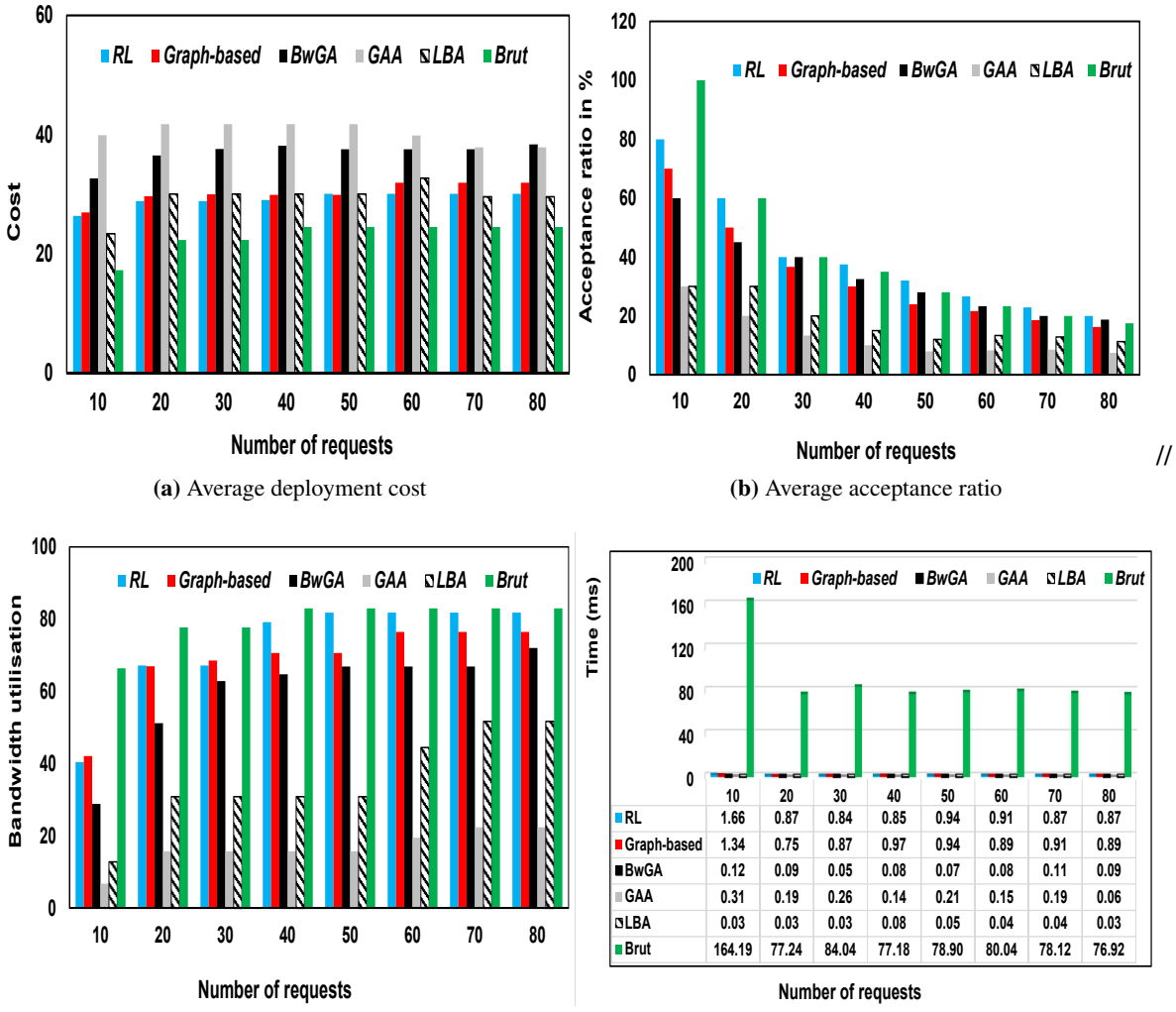
In experiment 1, whose results are shown in Fig. 7.5, we analyse the performance of the proposed algorithm in comparison with the optimal (brute-force) algorithm considering an offline case while varying the number of requests. Due to the high time consumption of the brute-force algorithm, the comparison has been done using the Abilene topology with 11 nodes. The results in Fig. 7.5a show

Algorithm 17 LBA Algorithm

Input: G_s, Ψ^r
Output: $Deployment\_solution$
Initialise: $prev\_host$=τ_s^r; $node\_solution = []$
for $n_v^p \in N_v$ **do**
  $Cands_{n_v^p}$=$Compute\_candidates(G_s, \Psi^r)$
  if $Cands_{n_v^p}$=$\emptyset$ **then**
    Reject request
    break
  end
  else
    Cands=$Sort\_Score(Cands_{n_v^p}, G_s)$
    n_v^p←Cands[0]
    $node\_solution.append(Cands[0])$
  end
end
Run $link\_mapping(node\_solution, G_s, \Psi^r)$
if *Sucessful* **then**
  Deploy request
end
else
  Reject request
end

that $Brut$ results in the best performance in terms of deployment cost per request with an average value of \$23, averaged over all request numbers. This is followed by RL, LBA, $Graph\text{-}based$, $BwGA$ and GAA with average values of \$27, \$29.38, \$30.25, \$37.0 and \$40.31, respectively. Therefore, RL is within a 14% margin of the optimal solution and results in more than approx. an 8% improvement compared to $Graph\text{-}based$ and LBA, 27% compared to $BwGA$ and 33% compared to GAA.

Moreover, the RL performance in terms of AR, as shown in Fig. 7.5b, is within a 1% margin of $Brut$, with an average value of 39.88% compared to 40.48% from $Brut$. This becomes a performance improvement of approximately 6% compared to $BwGA$ and $Graph\text{-}based$, 21.8% compared to LBA, and 26.66% compared to GAA, whose average AR values are 33.45%, 33.40%, 18.06% and 13.22%. $BwGA$ emerges competitive with respect to $Graph\text{-}based$, since it maps requests on the shortest possible paths, resulting in a low bandwidth utilization, which was in fact the bottleneck resource, as reflected from the high bandwidth consumption shown in Fig. 7.5c. However, even with this desirable behaviour, it remains inferior to RL since it may greedily reuse the shortest paths between any source and destination, which may result in resource bottlenecks in the long term, yet, RL is able to intelligently trade-off the resource consumption and cost in order to guarantee a good long term performance. The greedy nature of GAA and LBA may result in virtual nodes being mapped far from each other, which may result in a high consumption of link resources, leading to link resource bottlenecks. Moreover, this also increases the likelihood of failure to obtain a feasible substrate path



(a) Average deployment cost

(b) Average acceptance ratio

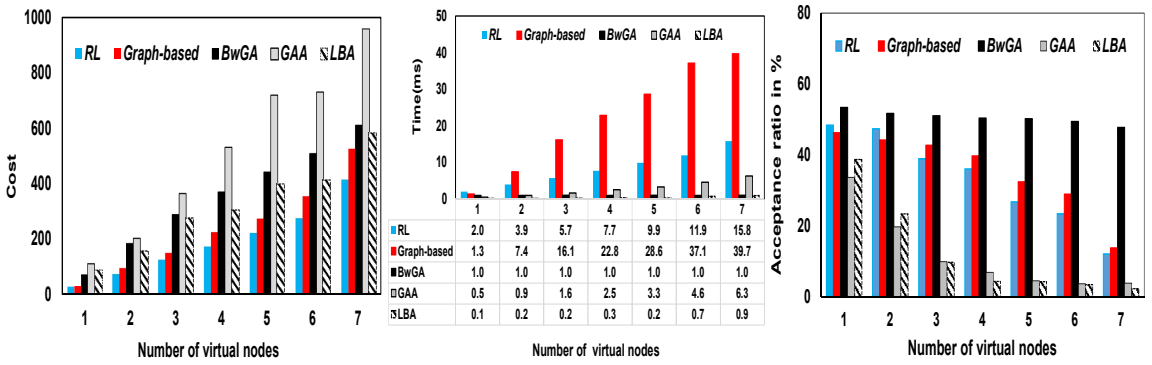
(c) Average Bandwidth utilisation

(d) Average request provisioning time

Figure 7.5: Results of experiment 1 considering the impact of demand size for Abilene topology

for hosting the corresponding virtual links due to delay constraints. This is evident from Fig. 7.5c where GAA and LBA result in the lowest values of bandwidth utilisation with average values of 16.8% and 35.54% respectively, demonstrating that most of the link resources remain unused due to link bottlenecks and the failure to meet delay constraints.

From Fig. 7.5d, $Brut$ results in more than a 98.9% overhead in terms of the average time for provisioning each request compared to the other algorithms, with an average value of 89.58 milliseconds for the considered topology. The rest of the algorithms are provisioning each request in a fraction of a millisecond with average values of 0.98, 0.94, 0.10, 0.20 and 0.04 milliseconds for RL, $Graph\text{-}based$, $BwGA$, LBA and GAA respectively, further demonstrating that the proposed RL-based algorithm is well suited for provisioning delay sensitive applications in resource constrained networks.



	1	2	3	4	5	6	7
RL	2.0	3.9	5.7	7.7	9.9	11.9	15.8
Graph-based	1.3	7.4	16.1	22.8	28.6	37.1	39.7
BwGA	1.0	1.0	1.0	1.0	1.0	1.0	1.0
GAA	0.5	0.9	1.6	2.5	3.3	4.6	6.3
LBA	0.1	0.2	0.2	0.3	0.2	0.7	0.9

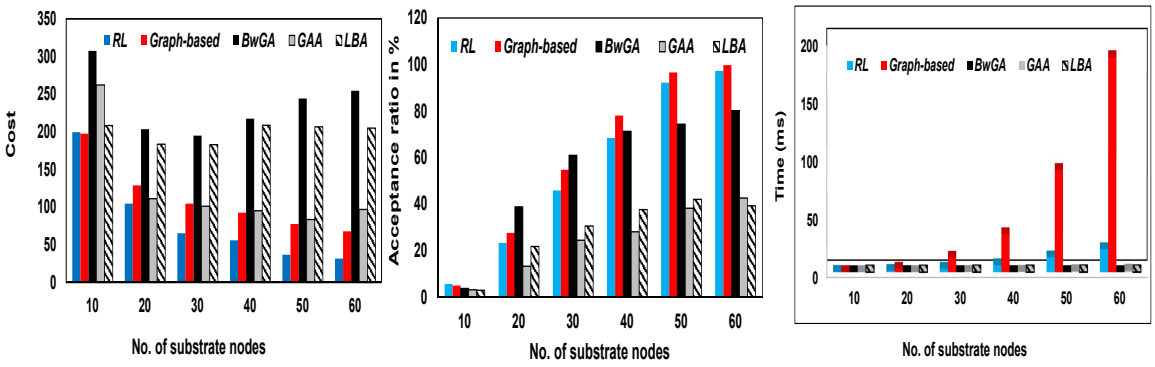
(a) Average deployment cost **(b)** Average request provisioning time **(c)** Average acceptance ratio

Figure 7.6: Results of experiment 2 considering the impact of request size under BIC topology.

Impact of request size

In experiment 2, whose results are shown in Fig. 7.6, the impact of the request size is analyzed by varying the number of requested virtual nodes from 1 to 7, and considering BIC network topologies for each request size and 200 offline demands. The experiment targets to demonstrate the scalability of the proposed algorithm as the size of the requests increases. From the results in Fig. 7.6a, RL results in an average value of \$186.8 in terms of service deployment cost for each request, resulting in a 20.72%, 46.93%, 63.71% and 40.65% improvement compared to $Graph\text{-}based$, $BwGA$, GAA and LBA respectively, whose average cost values are \$235.62, \$352.05, \$514.74 and \$314.77 respectively. These results reveal that: i) the average deployment cost of each algorithm under the BIC topology is higher than under the Abilene topology. This is due to the fact that BIC has more nodes and links, hence, the probability of activating new VNFs and traversing longer substrate paths increases, resulting in increased deployment costs; ii) The greedy algorithms, namely LBA, $BwGA$ and GAA, result in up to a 40% overhead in terms of cost, demonstrating the inefficiency of greedy approaches for solving problems in which the objective is influenced by multiple attributes; iii) even for a medium size topology such as BIC, RL remains superior over the state-of-the-art multi-layer graph based approach in terms of cost, service provisioning time and AR.

From the results in Fig. 7.6c, the AR performance of all algorithms tends to decrease with the increase of the request size. This is partly due to the increased resource consumption, and partly due to the increased probability of failure to satisfy the end-to-end constraints of the request. $BwGA$ results in the highest value of AR with an average value of 49.5% averaged across all request sizes. RL and $Graph\text{-}based$ result in a similar performance (within a 2% margin) with average values of 33.52% and 35.3% respectively, followed by LBA and GAA with 12.22% and 11.68% respectively. The results demonstrate that, even while achieving above a 20.72% improvement in terms of request provisioning cost over $Graph\text{-}based$, RL remains competitive in terms of AR. The $BwGA$ algorithm results in a high AR performance since it greedily targets to map requests using the least possible



(a) Average deployment cost **(b)** Average acceptance ratio **(c)** Average request provisioning time

Figure 7.7: Results of experiment 3 considering the impact of substrate network size under synthetic topology.

bandwidth resources at the expense of the request provisioning cost, the objective addressed in this paper. On the other hand, RL and $Graph\text{-}based$ may need to map virtual nodes further away from each other in order to minimise the VNF activation and the resource fragmentation costs, whenever such a decision results in a lower service deployment cost. However, this is achieved at the expense of an increased bandwidth resource consumption, especially as the network size increases.

From the results in Fig. 7.6b, RL provisions each request on average in 8.12 milliseconds, which is 62.81% faster than $Graph\text{-}based$, whose processing time per request is 21.86 milliseconds. LBA, GAA and $BwGA$ result in the lowest provisioning times, with average values in milliseconds of 0.36, 0.98 and 2.81 respectively, albeit at the expense of a high request deployment cost. As the number of required VNFs per request increases, the run time of $Graph\text{-}based$experiences a drastic increase since this increases the number of layers of its multi-layer graph, increasing the computation steps of the algorithm. On the contrary, the computational complexity in RL relies on a feature matrix whose size is dependant on the substrate network size, hence, a change in the request size only affects on the number of such computations.

Impact of substrate network size

In experiment 3, the impact of the substrate network size on the algorithms' performance is analyzed by varying the substrate nodes from 10 to 60, with the results shown in Fig. 7.7. The aim of this experiment is twofold: $i)$ to assess the impact of the substrate network size on the performance of the proposed algorithm regarding the aforementioned performance metrics; and $ii)$ to demonstrate the generalization capability of the proposed policy neural network regarding the use of substrate networks with sizes different from the one used for training, without the need of retraining the neural network. From the results in Fig. 7.7a, the average cost for the deployment of each request tends to decrease as the number of nodes increases, that applies for most of the algorithms. This is due to an increase in the availability of resources, which increases the number of available options for deploying

a service request at a lower cost. Moreover, RL results in an average request deployment cost of \$82.19, which becomes a percentage improvement of 26.2%, 65.3%, 34.2% and 58.7% compared to $Graph\text{-}based$, $BwGA$, GAA and LBA, whose average values are \$111.40, \$236.87, \$124.9 and \$199.04 respectively. The results reveal that the performance gain of using RL in terms of cost tends to increase as the substrate network increases. This is because, as the network size increases, the number of alternative nodes and links for provisioning the request increases, which complicates the decision making for the other algorithms. Thanks to its intelligence, RL is able to select the optimal nodes and links for the provisioning of the request among the multiple alternatives.

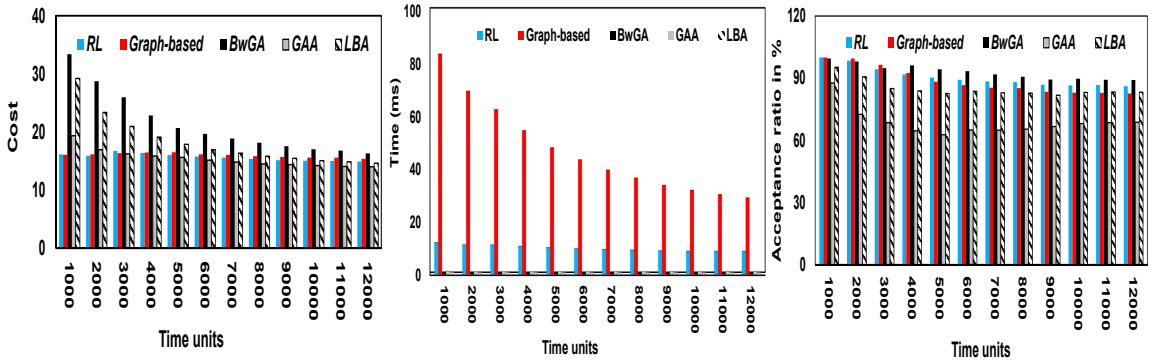
Moreover, Fig.7.7b reveals that RL is only 5% inferior in terms of AR compared to $Graph\text{-}based$, with an average AR value of 55.4% compared to 60.3% of $Graph\text{-}based$. $BwGA$, GAA and LBA result in AR average values of 54.87%, 25.0% and 29.05%, demonstrating the superiority of RL due to intelligently trading-off the service deployment objective and the resource utilisation efficiency in comparison with the alternative approaches. Moreover, the policy neural network is able to make efficient placement decisions, even for substrate networks whose size is inferior to the one used at the training stage, demonstrating the generalization capability of our adopted approach using dummy nodes when needed.

In terms of execution time, $Graph\text{-}based$ results in up to a 86% overhead, with an average processing time of 53.68 milliseconds per admitted request, compared to an average value of 7.13 milliseconds for RL. As the number of substrate nodes increases, the run time of $Graph\text{-}based$ greatly increases, since this fact increases the number of possible candidates for each virtual node of the request (hence, increasing the number of nodes at each layer of the multi-layer graph), and consequently, increasing the time-complexity of the multi-layer graph.

Online behaviour of the algorithms

From the results of Fig. 7.8, corresponding to experiment 4 of the performance analysis, we analyse the behaviour of the algorithms while considering a poison arrival of the requests with a mean value of 20 requests on each interval of 100 time units, for a total of 12,000 time units. This corresponds to a total of 2,400 requests. From Fig. 7.8a, the deployment cost for the algorithms tends to decrease along the time, with the decrease being dominant for $BwGA$, LBA and GAA. This is because, as new requests arrive, most of the VNFs remain activated, which decreases the VNF activation cost. Moreover, the number of feasible nodes and links decreases with the arrival of new requests, which simplifies the mapping decision of the greedy algorithms. On average, RL results in the lowest mapping cost per request, with an average value of \$13.91, followed by GAA, LBA, $Graph\text{-}based$ and $BwGA$, with average values of \$14.01, \$14.61, \$15.41 and \$16.31 respectively.

From Fig. 7.8b, RL provisions each request in less than 8.52 milliseconds, which is 70.3% faster than the alternative $Graph\text{-}based$ algorithm. The running time of $Graph\text{-}based$ decreases along the time due to a reduction in the number of feasible nodes for provisioning each virtual node of the request. This results in a reduction in the number of candidate nodes at each layer of the multi-layer graph, decreasing the time-complexity for obtaining a mapping solution.



(a) Average deployment cost **(b)** Average request provisioning time **(c)** Average acceptance ratio

Figure 7.8: Results of experiment 4 considering online arrival of requests under BIC topology.

7.5 Conclusion

This thesis chapter has addressed the problem of cost-effective and resource-efficient deployment of service requests with a possibility of sharing VNFs among multiple service requests, and under multiple conflicting cost components, including: resource fragmentation, VNF activation, energy consumption, packet processing and traffic forwarding. Given the complexity and the NP-hard nature of the above problem, the chapter has proposed a RL-based algorithm to solve the problem in practical time while yielding near optimal solutions. Moreover, the policy neural network has been designed in such a way that it can be adopted for making deployment decisions for substrate networks of different size, as long as that size is smaller than the one used for training the neural network. From the simulation results, the proposed algorithm has been found to be optimal in terms of acceptance ratio, placement cost and request provisioning time. The algorithm results in a performance similar to the brute-force algorithm in terms of AR while executing in less than 98% of the time required by the brute-force algorithm. In terms of service deployment cost, the proposed algorithm obtains solutions which are within a 14% margin of the optimal one; and results in up to a 20% and a 40% improvement in comparison with a state-of-the-art algorithm and a set of algorithms that greedily target to minimise only one cost component, respectively. Moreover, in some scenarios, the proposed algorithm is found to provision each request within up to 70% less time compared to a state-of-the-art multi-layer graph based algorithm. In addition, the proposed algorithm has been found to be scalable under changes in both network and request size, exhibiting good generalized properties. Thanks to the intelligence of the proposed algorithm, the above results have demonstrated that the proposed algorithm is well suited for the deployment of delay sensitive service requests under resource constrained networks, and where the placement objective is jointly influenced by multiple conflicting costs, which is a common occurrence in network and service management problems.

Part IV: Summary of Results, Conclusions and Future work

CHAPTER 8

Summary of thesis results, Conclusion and Future work

8.1 Introduction

5G/6G service providers expect to leverage the flexibility introduced by NFV in order to deploy services and applications, in the context of the eMBB, mMTC and uRLLC network slicing framework, whose network infrastructure requirements may span beyond the coverage area of a single InP, by leasing resources from multiple InPs in order to provide the end-to-end service. In this way, a challenging aspect for a service provider is how to obtain an optimal set of InPs on which to provision the service requests and the particular substrate nodes and links within each InP on which to map the different VNFs and virtual links of the service requests, respectively. Given the NP-hard nature of such a problem, its impractical to yield optimal solutions in polynomial time, and yet, the quality of the obtained mapping solution and the adopted solution technique will have a direct impact on both, the operational cost incurred by a Service Provider and the QoS of the deployed service in terms of reliability and deployment time, among others. In this way, this thesis sets out to contribute to this important aspect of the network and service management domain by proposing algorithms and an architectural framework to aid the deployment of service requests that may span multiple InPs. First, the thesis formulated the multi-domain service orchestration problem as an Integer Linear Program which is a typical NP-hard problem. Then, aware that this is a two-level decision making problem (i.e., choice of InPs and choice of substrate nodes and links within each InP to map a service request), the thesis breaks the grand multi-domain service orchestration problem into two interlinked sub-problems that are solved in a coordinated manner, namely: (1) the request splitting/partitioning sub-problem which involves obtaining a subset of cooperating or competing InPs and the corresponding inter-domain links on which to provision the different VNFs and virtual links of the service request, respectively. This is denoted as **sub-problem 1** throughout the thesis; (2) the intra-domain VNF orchestration sub-problem which involves obtaining the intra-domain nodes and links to provision the VNFs and virtual links of the sub-SFC associated with each InP at the request partitioning sub-problem. This is denoted as **sub-problem 2** throughout the thesis. Aware of the stringent reliability requirements of future services, and the fact that the underlying resource constrained network is to be shared by multiple services, the thesis sets out four key targets that are necessary to align the thesis proposals with the envisaged scenario in terms of service deployment cost and quality-of-service as follows:

1. Coordinated mapping of the VNFs and virtual links of the service requests with the aim of realizing better utilization of the underlying substrate resources, hence achieving higher acceptance ratios, revenue and reduced operational costs.
2. Survivable and fault-tolerant orchestration of service requests with the aim of minimizing QoS degradation and penalties resulting from SLA violations due to failure of the physical nodes, VNFs, or virtual links.
3. Compliance with the privacy requirements of the participating InPs which may be imposed due to reasons related to competition or security, among others.
4. Achieving all the above in practical run time in order to support deployment of delay sensitive applications.

However, achieving the above thesis targets is non-trivial given the NP-hard nature of both, the grand orchestration problem and its constituent sub-problems. In this way, the thesis adopted solution techniques that manifest the following key attributes:

- able to incorporate information learned in the previous solutions search space and historical mapping decisions in making the current mapping decision, hence, resulting in acceptable performance even in scenarios of limited information exposure and fuzzy environments.
- robust and less problem specific, hence, can be tailored to different optimization objectives, network topologies and service request constraints, thus enabling to deal with requests with either chained topologies or with bifurcated paths.
- capable of dealing with an optimization problem that is jointly affected by multiple attributes. This is critical since in practice, the service deployment cost is jointly affected by multiple conflicting costs including SLA violation costs, resource consumption costs, VNF activation cost, and resource fragmentation costs, among others. In this way, the thesis sought to propose solution techniques that intelligently infer the influence of each cost component on the service request deployment objective.
- able to realize near-optimal solutions in practical run-times, thus rendering well suited approaches for delay sensitive and resource constrained scenarios.

In this chapter, first, we summarize the contributions and results of the thesis towards solving the aforementioned sub-problems. Then, the key observations of the thesis regarding the solution techniques and algorithms proposed in the thesis are highlighted. Furthermore, this chapter outlines the technical limitations of the thesis proposals, which forms the basis for the possible future research work that can leverage and enhance the proposals developed in the thesis. Finally, the chapter presents the overall conclusion of the thesis.

The rest of this chapter is organized as follows: Section 8.2 gives a summary of the thesis results from the proposed algorithms for the two sub-problems. Section 8.3 introduces the thesis observations regarding the different adopted solution techniques while the possible future work enhancements to the thesis proposals are introduced in Section 8.4. Finally, the thesis is concluded in Section 8.5.

8.2 Summary of thesis results from the perspective of addressed sub-problems

This thesis is consisted of four parts with the proposals of the thesis and the corresponding results presented in **Parts II** and **III**. Specifically, the proposals to **sub-problem 1** of the thesis are presented in **Part II** with the results from these proposals presented at the end of Chapters 3 and 4 of the thesis. Additional contributions to this sub-problem are introduced in [86] and [93]. The contributions to **sub-problem 2** of the thesis are discussed in **Part III** of the thesis with the different proposals and their corresponding results introduced in Chapters 5- 7 of the thesis. In the sub-sections below, we summarize the contributions of the thesis for the two sub-problems including a summary of the corresponding results.

8.2.1 Request splitting/partitioning sub-problem

In order to deal with the problem of how to choose a set of InPs on which to deploy the different VNFs and virtual links of a service request in a way that maximizes the service deployment objective while adhering to the service request and network constraints, the thesis makes the following contributions:

A Reinforcement Learning based reliability-aware algorithm for orchestrating service requests across multiple domains. The algorithm incorporates both resource and QoS-violation costs while selecting the different InPs and their associated inter-domain links for provisioning a given service request. This is targeted to jointly minimize the cost associated with resource consumption and that resulting from QoS violation due to service failure. The simulation results from both online and offline behavior of service requests confirmed that the proposed algorithm:

- Results in up to a 10% performance improvement in terms of acceptance ratio and up to a 10% reduction in terms of mapping cost per admitted request compared to the state-of-the-art benchmark algorithms, demonstrating that the proposed algorithm offers a better utilization of both substrate node and link resources.
- Results in up to more than 90% saving in execution time for large networks compared to a SoA benchmark algorithm. Thus making it an ideal candidate for the practical case of resource constrained networks and delay sensitive applications.
- Is scalable with increase in both request and substrate network size.

To the best of our knowledge, this is the first work incorporating service reliability in the cross-domain service orchestration problem, which is critical for 5G and future scenarios involving mission critical applications. Moreover, different from the existing RL implementations, the neural network architecture of the policy in charge of making placement decisions in our work includes a convolutional layer, allowing a faster training stage and providing a higher convergence, since it uses a smaller number of trainable parameters compared to conventional feed forward neural networks. Moreover, we incorporate innovative approaches which enable the same policy neural network to be

used for substrate network sizes inferior to that used at the training stage and also for different cost components without the need for its retraining.

However, just like the existing approaches in literature, the above RL algorithm relies on a centralized entity for making the deployment decision. However, such approaches if not well managed, may require a level of information exposure which violates the privacy requirements of the participating InPs, which renders them unsuited for practical scenarios in which the InP privacy has to be enforced. Moreover, it is not possible for multiple entities to compute the partitioning solution in parallel, which affects the scalability and run time of the centralized approaches. In this way, with a view of overcoming the limitations associated with centralized approaches, the thesis proposes an additional algorithm for multi-domain service orchestration that relies on a fully distributed approach. The proposed algorithm relies on a multi-stage graph and the results from the conducted simulations confirmed that the proposed distributed algorithm:

- Is optimized in terms of acceptance ratio and embedding cost with up to a 60.0% reduction in terms of embedding cost per admitted request for some scenarios in comparison with a SoA benchmark algorithm.
- Can be executed in polynomial time with up to 88.7% performance improvement in terms of execution time for each admitted request compared to the benchmark SoA algorithm.
- Results in up to 8.2% and 95% reduction in terms of the number of InPs participating in the solution computation and the average number of messages processed by each participating node, respectively, compared to the benchmark SoA algorithm.
- Achieves a better utilization of the substrate node and link resources while preserving the privacy of the participating InPs and executing with acceptable run-time.
- Is scalable when increasing both the substrate network size and the request demand, rendering it well suited for large scale networks.

To the best of our knowledge, this is the first work incorporating a fully distributed algorithm to the cross-domain service orchestration problem involving coordination of both the request splitting and intra-domain mapping steps. Moreover, in contrast with conventional distributed algorithms, the solution computation in the thesis proposal involves only a pre-computed subset of InPs, thus drastically reducing both the number of nodes and the average number of messages processed by each node participating in the solution computation.

8.2.2 Intra-domain service orchestration sub-problem

In order to identify an optimal sub-set of substrate nodes and links for mapping the VNFs and virtual links respectively of a sub-SFC assigned to a given InP, that InP relies on an intra-domain algorithm that is executed in accordance with the internal policies of that InP. However, the quality of such an embedding algorithm not only affects the resource consumption inside that InP, but also the overall QoS and deployment cost for the entire service request. However, most of the existing intra-domain

mapping algorithms are inefficient in that they either perform the mapping of VNFs and virtual links in uncoordinated manner or do not incorporate service reliability during the mapping process, or they do so from the perspective of stateless VNFs. In this regard, this thesis contributes towards narrowing this existing gap by making the following contributions:

First, in order to achieve coordinated mapping of VNFs and virtual links of a service chain within an InP network, the thesis proposes a generic multi-stage graph based algorithm that can be tailored to different mapping objective functions such as reliability, energy consumption, deployment cost or a combination of these with minimal modification in the algorithm execution. Whereas most existing works consider chained SFC topologies that are typical of service applications in which traffic flows through a chain of VNFs with a single source and destination nodes, in practice, SFC topologies may need to support flows passing through a bifurcated path with single source and multiple destination nodes. An example of which could be an SFC chain composed of a load balancer which may be required to split the traffic flow between two or more servers according to a given policy. Such a heterogeneity inhibits the applicability of a number of existing algorithms for intra-domain service embedding such as those that rely on computing the shortest path between the ingress and egress nodes as the service mapping path. In this way, the proposed algorithm incorporates a request decomposition technique which splits a service request graph into a set of edge disjoint paths that are then mapped in a coordinated manner, thus facilitating the mapping of SFC chains with any topology such as star or mesh, among others. The results from the simulations reveal that the proposed algorithm:

- Is optimized in terms of resource utilization resulting in a 10% improvement in terms of acceptance ratio compared to a benchmark SoA algorithm.
- Is within a 4% margin of the optimal solution in terms of acceptance ratio, yet resulting in up to a 99% performance improvement in terms of execution time compared to the optimal.
- Is scalable when considering an increasing network size and demand, and can be tailored, easily and flexibly, to different mapping objectives.

Then, based on the above generic intra-domain embedding algorithm, the thesis proposes a set of survivable service deployment algorithms applicable to a practical case in which a service provider allocates resources for requests belonging to two service groups/priorities: the critical/high priority service group, such as the uRLLC, in which the survivability of a service must be guaranteed through backup resources, and the non-critical service group, such as the enhanced mobile broadband (eMBB) group, in which the services can tolerate a service disruption, hence can be provisioned on the unused backup resources of the high priority requests. In this respect, the thesis makes the following specific proposals:

1. A migration-aware algorithm which targets to maximise resource utilisation by enabling non-critical service applications to share the unused backup resources of critical services, in a way that minimizes the average number of preemptions of these users from the shared resources.

The results from the simulations confirmed that:

- The proposed algorithm results in more than 8% resource saving in most scenarios compared to a provisioning scheme in which the backup resources allocated to the critical requests cannot be shared by the non-critical requests.
- The proposed algorithm results in more than 70% performance improvement in terms of the number of service preemptions compared to a scheme in which the non-critical applications are mapped with a target of minimizing deployment costs.
- The proposed algorithm results in a negligible overhead in terms of execution time compared to an algorithm in which backup resources are not shared.

2. A QoS-aware global-rerouting algorithm for remapping low priority users that may be pre-empted from the borrowed resources, while minimizing the level of service interruption resulting from migration of surviving VNFs and virtual links to other substrate nodes and paths, hence, minimizing the overall delay and cost associated with the migration. The proposed algorithm is shown to outperform a service restoration scheme based on local rerouting in terms of successful service restoration and resource consumption.

The above approach enhances service survivability from the perspective of stateless VNFs. However, in practice, a number of VNFs are stateful, hence necessitating full coordination when mapping the active and stand-by instances of a service request, with a target of minimizing the overall resource consumption, including that due to state updates. This may render algorithms developed for stateless VNFs scenarios not well suited for the stateful VNFs case. In this way, the thesis proposes a set of Metaheuristic algorithms, namely: Genetic and Harmony Search algorithms, for fault-tolerant orchestration of service requests within a single domain Infrastructure considering stateful VNFs. These algorithms have been proven to exhibit good performance in problems such as that addressed in this thesis in which the objective function is influenced by multiple components including those that may be conflicting. The results from the simulations have revealed that:

- The proposed metaheuristic solutions with a specific solution initialization result in up to a 78% performance improvement in terms of average AR and up to a 34% reduction in terms of mapping cost per admitted request, compared to similar approaches with randomly initialized solutions.
- The coordinated mapping of both the active and stand-by SFC using the metaheuristic approach with a specific solution initialization results in up to a 20% performance improvement in terms of average AR and up to a 5% reduction in terms of mapping cost per admitted request compared to an approach that greedily targets to minimize bandwidth resource consumption.

To the best of our knowledge, this is the first work incorporating Genetic algorithm or Harmony Search algorithm to the problem of fault-tolerant orchestration of stateful VNFs. Moreover, different from existing works, we consider a practical scenario in which the different VNFs of the service request may need to be deployed across different nodes such as servers or data centers.

Finally, owing to the fact that multiple services can have common VNFs in their chaining, the thesis proposes a solution for cost-effective and resource efficient intra-domain orchestration of online services from the perspective of sharing their VNF instances. However, service deployment problem under VNFs sharing is non-trivial given the multiple cost components that may influence the decision on when and when not to share deployed VNFs among different services. In this way, the thesis proposes a reinforcement learning based algorithm capable of making intelligent placement decisions while considering multiple conflicting costs. Costs of transmission, VNF instantiation or energy consumption, among others. Thanks to the intelligence of the RL algorithm, simulation results confirm that the proposed algorithm:

- Is within a 14% margin and similar to an optimal solution in terms of request provisioning cost and acceptance ratio, respectively.
- Results in more than a 20% and 40% reduction in terms of request deployment cost compared to a state-of-the-art algorithm and an algorithm that greedily minimizes the transmission or VNF activation costs.
- Executes in practical time and is scalable with increase in both request and substrate network size resulting in up to a 70% improvement in terms of execution time compared to the SoA benchmark algorithm.

To the best of our knowledge, this is the first work adopting a machine learning approach to the problem of VNF instance sharing. Moreover, we incorporate multiple cost components that might influence the VNF sharing decision.

8.3 Summary of thesis observations from the perspective of adopted solution techniques

The thesis sought to contribute to the problem of how network services and resources can be orchestrated across multiple InPs under a virtualized environment in a resource efficient and cost-effective manner while adhering to the various service and system requirements including service end-to-end delay, service reliability, and InP privacy, among others. Given that the grand multi-domain service orchestration problem involves both Intra-domain and Cross-domain orchestration, the thesis proposed algorithms for both of the above cases based on a number of solution techniques including Deep Reinforcement Learning (DRL), Genetic Algorithm (GA), Harmony Search (HS) and a multi-stage graph based approach, among others. In the subsequent subsections of this section, the key observations regarding the proposed algorithms and adopted solution techniques are introduced.

8.3.1 Reinforcement learning for resource management

The reinforcement learning technique was applied to the request splitting sub-problem in Chapter 3 and to the intra-domain orchestration sub-problem in Chapter 7 and [78] while considering both online and offline scenarios. From the different experiments conducted for the different simulation

scenarios, the following observations were made.

RL for multiple attribute problems

The results from Chapter 7 and [78] reveal that in comparison to conventional heuristic approaches, RL is more effective in dealing with problems whose objective function is jointly influenced by multiple attributes. For instance, in Chapter 7 in which the request mapping objective of service deployment cost was jointly affected by multiple conflicting costs, including: transmission, VNF instantiation or energy consumption, among others, the RL approach was found to be within a 14% margin and similar to an optimal solution in terms of request provisioning cost and acceptance ratio, respectively, while resulting in more than a 20% and a 70% reduction in terms of request deployment cost and execution time respectively compared to a state-of-the-art heuristic, and up to more than a 40% reduction in terms of cost compared to an algorithm that greedily minimizes the transmission or VNF activation costs. These results demonstrated the ability of the RL approach to intelligently infer and make a trade-off regarding the influence of each cost component towards the service orchestration objective.

RL for varying state size

A key limitation of machine learning techniques incorporating policy neural networks in general is that once learned, the internal structure of a neural network cannot be modified. In this way, it is not possible to use policy neural networks for problem instances whose state size is different from that used during the training stage. However, the results from Chapters 3 and 7 demonstrated that in situations where the test state size is inferior to that used at the training stage, it is possible to overcome this restriction by incorporating dummy features to assure fixed dimensions of the state matrix in a manner that does not result in a performance loss. In this way, for a substrate network of a smaller size than that used at the training stage, the input matrix is matched by appending dummy nodes with dummy feature vectors to reach the expected state size used at the training stage. The results in Chapter 7 reveal that doing this does not result in any loss in performance or generalization capability of the policy neural network during the testing stage. For instance, when considering 50 substrate nodes, the proposed RL outperforms the SoA benchmark algorithms by 52.7% in terms mapping cost and by 53.45% when considering 60 substrate nodes that are used for training the policy network, which translates to less than a 1% margin. This is attributed to the fact that values of the features assigned to the dummy nodes are chosen in a way that makes them less probable for selection by the policy neural network.

Adoption of convolutional neural network architecture

The policy neural network training results obtained in Chapter 7 demonstrate that a policy neural network incorporating a convolutional neural network (CNN) architecture converges to a similar reward value with that based on a conventional feed-forward neural network (FFN) architecture, but with a faster convergence. Moreover, considering a similar number of accepted requests, the two

architectures demonstrate a similar training time per epoch. Such a good performance with fewer trainable parameters makes CNN based architectures more memory efficient, making them better candidates for edge-computing and multi-agent learning scenarios in which the different service life-cycle management decisions may need to be executed by distributed resource constrained nodes.

RL for NP-hard problems

The problem addressed by the thesis is typically NP-hard, hence not feasible to realize optimal solutions in practical time. In this way, RL was proposed targeting realization of near-optimal solutions in acceptable run time. For instance, the results from Chapter 7 demonstrate that RL results in up to a 86% reduction in terms of average time for mapping each request in some scenarios compared to a state-of-art benchmark algorithm, while remaining competitive with respect to a brute-force algorithm. Moreover, the algorithm is found to remain scalable with increase in both substrate network and request size.

8.3.2 Multi-stage graph for service provisioning

The proposals in Chapters 4, 5 and [84] relied on a multi-stage graph for solution computation. The results have demonstrated that a multi-stage graph approach is effective and able to obtain near-optimal solutions in practical run time. However, the scalability of such an approach is shown to be affected by the number of stages in the graph and the number of nodes at each layer of the graph. The number of stages is directly related to the number of VNFs in the SFC request while the number of nodes at each stage of the graph is directly related to the number of candidate nodes for the VNF corresponding to that stage. In this way, this approach found to be well suited for problems and scenarios in which a VNF can only be served by a subset of available nodes. This is typical for the VNE problem since a given node cannot host two virtual nodes from the same request, and for SFC orchestration problems in which the different VNFs are constrained by parameters such as location and resource requirements, among others, hence rendering a number of substrate nodes unfeasible for hosting any given VNF of the request. However, in a possible scenario in which a large number of nodes are suitable candidates for a given VNF, the complexity of the graph can be tamed by selecting a subset of these nodes based on factors such as cost or reliability, among others.

8.3.3 Genetic and Harmony Search Algorithms for fault-tolerant service provisioning

The thesis adopted a Metaheuristic approach based on Genetic Algorithm (GA) and Harmony Search Algorithm (HS) in order to realize fault-tolerant orchestration of stateful VNFs in Chapter 6. The obtained results reveal that by initializing the solution space based on a specific initialization technique, the algorithms' AR performance improves by up to 78% while the average mapping cost per admitted request decreases by 34% over similar schemes in which the solution space is randomly initialized. This is due to the fact that the fault-tolerant orchestration problem is associated with additional constraints such as non-sharing of nodes between the active and stand-by instance

solutions, which may result in a high number of invalid solutions if such considerations are not reflected in the initialization process. Moreover, due to the stateful nature of VNFs, minimizing the state-update cost requires coordinated mapping of both the stand-by and active instances, which attribute results in better deployment solutions if reflected in the initialization process.

8.4 Future work Enhancements

Although the thesis has attempted to address critical aspects of the multi-domain service orchestration problem, the multifaceted nature of such a problem leaves room for making enhancements to the thesis proposals. In the subsequent subsections, we provide suggestions regarding aspects that can be investigated to enhance and also build on the proposed algorithms regarding the different aspects addressed by the thesis.

8.4.1 Orchestration of stateful VNFs

In this thesis, we have articulated that the state-update information needs to be continuously transferred to the standby instances of all the stateful VNFs during traffic processing in order to have seamless transition in case of failure of the active instance. However, this results in a high resources consumption and wastage in the event that no failure is experienced on the active instances. In this regard, future work will explore and investigate the applicability of failure prediction techniques as input to the fault-tolerant service orchestration algorithm in order to make intelligent decisions regarding the optimal intervals for state update that will guarantee seamless transition in case of failure. In this way, state updates would only need to be made when failures are bound to be experienced, hence saving resources and time for unnecessary processing of state-update information. Moreover, such updates could only be made for a subset of stateful VNFs.

8.4.2 Resource elasticity in multi-domain orchestration

The algorithms proposed in the thesis considered the service requests to be characterized by immutable requirements in terms of link and node resources throughout their life-time. However, in practice, such requirements may have temporal variations in terms of the required amount of resources, requiring the embedding algorithm to intelligently adapt to such dynamism. As opposed to the existing approaches that deal with elasticity in single domain networks, the elasticity problem when considering a limited information disclosure in a multi-domain setting is non-trivial. This scenario requires the elasticity algorithm to intelligently rely on the partially disclosed information and its past experience to infer short-term future resource availability or any request requirements alteration. Therefore, it would be interesting to investigate and propose innovative and intelligent algorithms or techniques for ensuring an elastic service provisioning across multiple domains while considering limited information disclosure and intra-domain topology abstraction. One possible approach for investigation is the application of multi-agent reinforcement learning in which the different InPs could be treated as autonomous cooperative or competing agents that interact in a manner that ensures

that the requirements of the supported services are always met with the least possible amount of resources.

8.4.3 Delay models

The thesis has adopted linear models for resource costs, and assumed the propagation delay along substrate links to be uniformly distributed between given values. However, it would be interesting to investigate the impact of different delay models such as queuing models for both the packet propagation and processing delays on the performance of the algorithms proposed by the thesis, especially under VNF sharing considerations. This is because an increase in delay experienced by service requests increases the request rejection rate, which may degrade the performance gain from the adoption of a given solution approach.

8.4.4 Results validation on test bed scenarios

Whereas the algorithms proposed by the thesis have demonstrated excellent performance under the adopted simulation environment, future works and enhancements to the current thesis proposal will target the experimental implementation of an end-to-end slice instance spanning multiple domain infrastructure based on the algorithms proposed in the thesis proposal. This will entail the usage of realistic considerations regarding traffic models, resource requirements behavior, and life-cycle management. Furthermore, such an implementation needs to analyze the performance of the RL algorithm in possible situations in which the values of the parameters and features could change from those used at the training stage when using the policy neural network at test stage.

8.5 Conclusion

In this thesis, a detailed review of the state-of-the-art regarding the multi-domain service orchestration problem has been provided, highlighting the limitations of existing approaches. Based on this, comprehensive proposals, targeting to narrow the existing research gap, were introduced in **part II** and **part III** of the thesis. Specifically, the thesis decomposed the grand multi-domain orchestration problem into two sub-problems: **sub-problem 1** deals with the problem of partitioning / splitting the different components of a given service request across a subset of InPs. The corresponding algorithms proposed by the thesis are introduced in **part II** of the thesis comprised of Chapters 3 and 4 with additional contributions introduced in [86] and [93]; **sub-problem 2** involves orchestrating the assigned sub-SFC or an entire SFC within a single InP infrastructure. The solution approaches for this sub-problem are introduced in **part III** of the thesis which is comprised of Chapters 5, 6 and 7 with additional contributions introduced [84].

While addressing the problem, the thesis set out four key targets that were necessary to align the proposals with the multi-domain NFV scenario in terms of deployment cost and quality-of-service of service requests: (1) coordinated mapping of service requests not only in terms of VNFs and virtual links but also in terms of request splitting and intra-domain mapping, with a view of realizing better

utilization of the underlying substrate resources, (2) survivability and fault-tolerant orchestration of service requests with a view of not only taming QoS violations but also minimizing penalties that may result from such violations, (3) limited disclosure of InP internal information with a view of adhering to the privacy requirements of the participating InPs, and (4) achieving all the above targets in polynomial time with a view of supporting delay sensitive applications. Specifically, the proposals in Chapters 5 and 6, and [78] focus on survivability and fault-tolerant orchestrations of network services. The proposals in [84], [86], [93] and part of Chapter 5 focus on coordinated and cost-effective mapping of service requests. On the other hand, the fully distributed algorithm proposed in Chapter 4 targets scenarios with limited or no information disclosure from the participating InPs. Moreover, in order to realise near-optimal solutions in feasible time, the thesis proposed techniques based on metaheuristic and heuristic solutions designed in a manner that allows faster execution without compromising the solution quality. For instance, in the multi-stage graph based heuristic, only a precomputed set of candidate nodes where used in constructing the multi-stage graph which, not only results in faster execution, but also better solution quality, since unfeasible nodes which would possibly leak into the solution are not part of the graph.

Moreover, in order to guide the implementation, execution and adherence of the thesis proposals to the four main targets of the thesis, an architectural framework is proposed in Chapter 3. The proposed framework is aligned with the ETSI NFV-MANO architectural framework described in [51] and the ETSI's architecture options introduced in [10], in which there is a single NFV Orchestrator (NFVO) per administrative domain, with the different orchestrators able to communicate through the Orchestrator-to-Orchestrator (Or-Or) interface proposed in [10].

Based on the above thesis proposals, this thesis-report has highlighted the key observations regarding the adopted solution techniques and algorithms, and it has provided recommendations regarding possible aspects that need to be investigated in order to enhance the contributions of the thesis concerning both the proposed algorithms and the adopted solution approaches. Overall, the results from the different experiments conducted have proved that the thesis proposals are optimized in terms of request acceptance ratios, mapping cost and execution time. This renders such proposals well suited for 5G and future scenarios characterized by stringent delay, priority and resources requirements for which the thesis was targeted.

References

[1] Ahmed El-mekkawi, Xavier Hesselbach, and Jose Ramon Piney. Squatting and kicking model evaluation for prioritized sliced resource management. *Computer Networks*, 167, 2020.

[2] Yuncan Zhang, Fujun He, Takehiro Sato, and Eiji Oki. Network Service Scheduling with Resource Sharing and Preemption. *IEEE Transactions on Network and Service Management*, 17(2):764–778, 2020.

[3] Bo Yi, Xingwei Wang, and Min Huang. A Generalized VNF Sharing Approach for Service Scheduling. *IEEE Communications Letters*, 22(1):73–76, 2018.

[4] Mathieu Leconte, Georgios S Paschos, Panayotis Mertikopoulos, and Ulas C Kozat. A Resource Allocation Framework for Network Slicing. *Proceedings - IEEE INFOCOM*, 2018-April:2177–2185, 2018.

[5] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges. *Computer Networks*, 167(November), 2020.

[6] Prabhu Kaliyammal Thiruvasagam, Vijeth J Kotagi, and C Siva Ram Murthy. The More the Merrier: Enhancing Reliability of 5G Communication Services With Guaranteed Delay. *IEEE Networking Letters*, 1(2):52–55, 2019.

[7] David Dietrich, Ahmed Abujoda, Amr Rizk, and Panagiotis Papadimitriou. Multi-Provider Service Chain Embedding With Nestor. *IEEE Transactions on Network and Service Management*, 14(1):91–105, 2017.

[8] Jose Ordonez-Lucena, Pablo Ameigeiras, DIego Lopez, Juan J Ramos-Munoz, Javier Lorca, and Jesus Folgueira. Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges. *IEEE Communications Magazine*, 55(5):80–87, 2017.

[9] Etsi Gs Nfv 001 V1.1.1. NFV-Use Cases. *IEEE Network*, 1(5):1–50, 2013.

[10] Group Report. Network Functions Virtualisation (NFV) Release 3 ;. 1:1–59, 2018.

[11] David Dietrich, Amr Rizk, and Panagiotis Papadimitriou. Multi-domain virtual network embedding with limited information disclosure. *2013 IFIP Networking Conference, IFIP Networking 2013*, 12(2):188–201, 2013.

[12] David Dietrich, Amr Rizk, and Panagiotis Papadimitriou. Multi-Provider Virtual Network Embedding With Limited Information Disclosure. *IEEE Transactions on Network and Service Management*, 12(2):188–201, 2015.

[13] Ines Houidi, Wajdi Louati, Walid Ben Ameur, and Djamal Zeghlache. Virtual network provisioning across multiple substrate networks. *Computer Networks*, 55(4):1011–1023, 2011.

[14] Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Salete Buriol, Marinho Pilla Barcellos, and Luciano Paschoal Gaspary. Piecing Together the NFV Provisioning Puzzle : Efficient Placement and Chaining of Virtual Network Functions. (March), 2015.

[15] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on Networking*, 20(1):206–219, 2012.

[16] Ilhem Fajjari and Orange Labs. Resource Allocation Algorithms for Virtual Networks within Cloud Backbone Network Doctor of Science Thesis University Pierre et Marie Curie – Resource Allocation Algorithms for Virtual Networks within Cloud Backbone Network Commitee in charge :. (June), 2015.

[17] Clifford Stein. Improved approximation algorithms for unsplittable flow problems. pages 426–435, 1997.

[18] Tuan-minh Pham and Hoai-nam Chu. Multi-Provider and Multi-Domain Resource Orchestration in Network Functions Virtualization. *IEEE Access*, 7:86920–86931, 2019.

[19] Ahmed Abujoda and Panagiotis Papadimitriou. DistNSE: Distributed network service embedding across multiple providers. *2016 8th International Conference on Communication Systems and Networks, COMSNETS 2016*, (i):1–8, 2016.

[20] Christoph Werle, Panagiotis Papadimitriou, Ines Houidi, Wajdi Louati, Djamal Zeghlache, Roland Bless, and Laurent Mathy. Building virtual networks across multiple domains. *Computer Communication Review*, 41(4):412–413, 2011.

[21] Peiying Zhang, Haipeng Yao, and Yunjie Liu. Virtual Network Embedding Based on Computing, Network, and Storage Resource Constraints. *IEEE Internet of Things Journal*, 5(5):3298–3304, 2018.

[22] Gang Sun, Zhu Xu, Hongfang Yu, Xi Chen, Victor Chang, and Athanasios V Vasilakos. Low-latency and Resource-efficient Service Function Chaining Orchestration in Network Function Virtualization. *IEEE Internet of Things Journal*, PP(c):1, 2019.

[23] Peiying Zhang, Haipeng Yao, Chao Qiu, and Yunjie Liu. Virtual network embedding using node multiple metrics based on simplified electre method. *IEEE Access*, 6:37314–37327, 2018.

[24] Peiying Zhang, Chao Wang, Chunxiao Jiang, and Abderrahim Benslimane. Security-Aware Virtual Network Embedding Algorithm based on Reinforcement Learning. *IEEE Transactions on Network Science and Engineering*, 4697(c):1–11, 2020.

[25] Peiying Zhang, Haipeng Yao, and Yunjie Liu. Virtual network embedding based on the degree and clustering coefficient information. *IEEE Access*, 4:8572–8580, 2016.

[26] Guochang Yuan, Zichuan Xu, Binxu Yang, Weifa Liang, Wei Koong, Daphné Tuncer, Alex Galis, George Pavlou, and Guowei Wu. Fault tolerant placement of stateful VNFs and dynamic fault recovery in cloud networks. *Computer Networks*, 166:106953, 2020.

[27] Binxu Yang, Zichuan Xu, Wei Koong Chai, Weifa Liang, Daphne Tuncer, Alex Galis, and George Pavlou. Algorithms for fault-tolerant placement of stateful virtualized network functions. *IEEE International Conference on Communications*, 2018-May, 2018.

[28] Weixi Mao, Lei Wang, Jin Zhao, and Yuedong Xu. Online Fault-tolerant VNF Chain Placement: A Deep Reinforcement Learning Approach. *IFIP Networking 2020 Conference and Workshops, Networking 2020*, pages 163–171, 2020.

[29] Michael Till Beck, Juan Felipe Botero, and Kai Samelin. Resilient allocation of service Function chains. *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2016*, pages 128–133, 2017.

[30] Jose Ordonez-lucena, Oscar Adamuz-hinojosa, Pablo Ameigeiras, Pablo Mu, and Juan J Ramos-mu. The Creation Phase in Network Slicing : From a Service Order to an Operative Network Slice. pages 31–36, 2018.

[31] Fida E Zaheer, Jin Xiao, and Raouf Boutaba. Multi-provider service negotiation and contracting in network virtualization. *Proceedings of the 2010 IEEE/IFIP Network Operations and Management Symposium, NOMS 2010*, pages 471–478, 2010.

[32] Yufeng Xin, Ilia Baldine, Anirban Mandal, Chris Heermann, Jeff Chase, and Aydan Yumerefendi. Embedding virtual topologies in networked clouds. *Proceedings of the 6th International Conference on Future Internet Technologies, CFI11*, (January):26–29, 2011.

[33] Arpit Shukla, Rajesh Gupta, Sudeep Tanwar, Neeraj Kumar, and Joel J.P.C. Rodrigues. BlockRAS: A P2P Resource Allocation Scheme in 6G Environment with Public Blockchains. *2020 IEEE Global Communications Conference, GLOBECOM 2020 - Proceedings*, 2020-Janua:1–6, 2020.

[34] Kailing Guo, Ying Wang, Xuesong Qiu, Wenjing Li, and Ailing Xiao. Particle swarm optimization based multi-domain virtual network embedding. *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*, pages 798–801, 2015.

[35] Qiong Zhang, Xi Wang, Inwoong Kim, Paparao Palacharla, and Tadashi Ikeuchi. Vertex-centric computation of service function chains in multi-domain networks. *IEEE NETSOFT 2016 - 2016 IEEE NetSoft Conference and Workshops: Software-Defined Infrastructure for Networks, Clouds, IoT and Services*, pages 211–218, 2016.

[36] Gang Sun, Yayu Li, Dan Liao, and Victor Chang. Service function chain orchestration across multiple domains: A full mesh aggregation approach. *IEEE Transactions on Network and Service Management*, 15(3):1175–1191, 2018.

[37] Gang Sun, Yayu Li, Guangyang Zhu, Dan Liao, and Victor Chang. Energy-efficient service function chain provisioning in multi-domain networks. *IoTBDS 2018 - Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security*, 2018-March(January):144–163, 2018.

[38] Leyi Zhang, Ying Wang, Xuesong Qiu, and Hantao Guo. Redundancy mechanism of service function chain with node-ranking algorithm. *2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM 2019*, pages 586–589, 2019.

[39] Weiran Ding, Hongfang Yu, and Shouxi Luo. Enhancing the reliability of services in NFV with the cost-efficient redundancy scheme. *IEEE International Conference on Communications*, 1:1–6, 2017.

[40] Meitian Huang, Weifa Liang, Xiaojun Shen, Yu Ma, and Haibin Kan. Reliability-Aware Virtualized Network Function Services Provisioning in Mobile Edge Computing. *IEEE Transactions on Mobile Computing*, X(X):1, 2019.

[41] Marco Casazza, Mathieu Bouet, and Stefano Secci. Availability-driven NFV orchestration. *Computer Networks*, 155:47–61, 2019.

[42] Karthik Karra and Krishna M Sivalingam. Providing Resiliency for Service Function Chaining in NFV systems using a SDN-based approach. *2018 24th National Conference on Communications, NCC 2018*, pages 1–6, 2019.

[43] Jingyuan Fan, Xiujiao Gao, Zilong Ye, Kui Ren, Chaowen Guan, and Chunming Qiao. GREP: Guaranteeing reliability with enhanced protection in NFV. *HotMiddlebox 2015 - Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, Part of SIGCOMM 2015*, pages 13–18, 2015.

[44] Saifeddine Aidi, Mohamed Faten Zhani, and Yehia Elkhatib. On Improving Service Chains Survivability Through Efficient Backup Provisioning. *14th International Conference on Network and Service Management, CNSM 2018 and Workshops, 1st International Workshop on High-Precision Networks Operations and Control, HiPNet 2018 and 1st Workshop on Segment Routing and Service Function Chaining, SR+SFC 2*, (Cnsm):108–115, 2018.

[45] Jing Li, Weifa Liang, Meitian Huang, and Xiahua Jia. Providing reliability-aware virtualized network function services for mobile edge computing. *Proceedings - International Conference on Distributed Computing Systems*, 2019-July:732–741, 2019.

[46] Manuel Peuster and Holger Karl. E-State : Distributed State Management in Elastic Network Function Deployments. *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 6–10.

[47] Haotong Cao, Yongxu Zhu, Longxiang Yang, and Gan Zheng. A Efficient Mapping Algorithm with Novel Node-Ranking Approach for Embedding Virtual Networks. *IEEE Access*, 5:22054–22066, 2017.

[48] Shuopeng Li, Mohand Yazid Saidi, and Ken Chen. Multi-domainvirtualnetwork embedding with coordinated link mapping. *Advances in Science, Technology and Engineering Systems*, 2(3):545–552, 2017.

[49] J Martin-Perez and Carlos J Bernardos. Multi-Domain VNF Mapping Algorithms. *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, BMSB*, 2018-June, 2018.

[50] Qi Xu, Deyun Gao, Huachun Zhou, Wei Quan, and Wenfeng Shi. An energy-aware method for multi-domain service function chaining. *Journal of Internet Technology*, 19:1727–1739, 2018.

[51] Toktam Mahmoodi, Huub Van Helvoort, and Scott Mansfield. Management and orchestration. *IEEE Communications Standards Magazine*, 1(4):60, 2017.

[52] Multi-operator Orchestration, Balázs Sonkoly, Associate Member, Róbert Szabó, Balázs Németh, and Graduate Student Member. 5G Applications From Vision to Reality :. 38(7):1401–1416, 2020.

[53] Jorge Baranda, Josep Mangues-bafalluy, Ricardo Mart, Luca Vettori, Kiril Antevski, Carlos J Bernardos, and Xi Li. 5G-TRANSFORMER meets Network Service Federation : design , implementation and evaluation.

[54] O N F Tr. SDN A rchitecture. (1):1–59, 2016.

[55] The M E F Forum and M E F Forum. Service Operations Specification MEF 55 Lifecycle Service Orchestration (LSO): Reference Architecture and Framework March 2016. (March), 2016.

[56] Tarik Taleb, Ibrahim Afolabi, Konstantinos Samdanis, and Faqir Zarrar Yousaf. On multi-domain network slicing orchestration architecture and federated resource control. *IEEE Network*, 33(5):242–252, 2019.

[57] Ibrahim Afolabi, Miloud Bagaa, Tarik Taleb, and Hannu Flinck. End-To-end network slicing enabled through network function virtualization. *2017 IEEE Conference on Standards for Communications and Networking, CSCN 2017*, pages 30–35, 2017.

[58] Francesco Tusa, Stuart Clayman, Dario Valocchi, and Alex Galis. Multi-Domain Orchestration for the Deployment and Management of Services on a Slice Enabled NFVI. *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2018*, pages 1–5, 2018.

[59] Giovanni Baggio, Antonio Francescon, and Riccardo Fedrizzi. Multi-domain service orchestration with X-MANO. *2017 IEEE Conference on Network Softwarization: Softwarization Sustaining a Hyper-Connected World: en Route to 5G, NetSoft 2017*, pages 5–6, 2017.

[60] Antonio Francescon, Giovanni Baggio, Riccardo Fedrizzi, Imen Grida, Ben Yahia, and Roberto Riggio. X – MANO : Cross – domain Management and Orchestration of Network Services. (July), 2017.

[61] Kostas Katsalis, Navid Nikaein, and Andy Edmonds. Multi-Domain Orchestration for NFV : Challenges and Research Directions.

[62] Meng Shen, Ke Xu, Kun Yang, and Hsiao Hwa Chen. Towards efficient virtual network embedding across multiple network domains. *IEEE International Workshop on Quality of Service, IWQoS*, pages 61–70, 2014.

[63] Vincenzo Eramo, Francesco G Lavacca, Tiziana Catena, Marco Polverini, and Antonio Cianfrani. Effectiveness of Segment Routing Technology in Reducing the Bandwidth and Cloud Resources Provisioning Times in Network Function Virtualization Architectures †. (i):1–20, 2019.

[64] Fady Samuel, Mosharaf Chowdhury, and Raouf Boutaba. PolyViNE: Policy-based virtual network embedding across multiple domains. *Journal of Internet Services and Applications*, 4(1):1–23, 2013.

[65] Ying Zhang and Qingfeng Huang. A learning-based adaptive routing tree for wireless sensor networks. *Journal of Communications*, 1(2):12–21, 2006.

[66] Ines Houidi, Wajdi Louati, and Djamal Zeghlache. A distributed virtual network mapping algorithm. *IEEE International Conference on Communications*, pages 5634–5640, 2008.

[67] Shihabur Rahman Chowdhury, Reaz Ahmed, Md Mashrur Alam Khan, Nashid Shahriar, Raouf Boutaba, Jeebak Mitra, and Feng Zeng. Dedicated Protection for Survivable Virtual Network Embedding. *IEEE Transactions on Network and Service Management*, 13(4):913–926, 2016.

[68] Xiaojun Shang, Zhenhua Li, and Yuanyuan Yang. Rerouting Strategies for Highly Available Virtual Network Functions. *IEEE Transactions on Cloud Computing*, PP(c):1, 2019.

[69] Xiaojun Shang, Zhenhua Li, and Yuanyuan Yang. Placement of highly available virtual network functions through local rerouting. *Proceedings - 15th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, MASS 2018*, pages 80–88, 2018.

[70] Shashank Bijwe, Fumio Machida, Shinya Ishida, and Seiichi Koizumi. End-to-end reliability assurance of service chain embedding for network function virtualization. *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2017*, 2017-Janua:1–4, 2017.

[71] Han Qing, Zang Weifei, and Lan Julong. Virtual network protection strategy to ensure the reliability of SFC in NFV. *ACM International Conference Proceeding Series*, pages 5–9, 2017.

[72] Ngoc Thanh Dinh and Younghan Kim. An Efficient Reliability Guaranteed Deployment Scheme for Service Function Chains. *IEEE Access*, 7:46491–46505, 2019.

[73] Oussama Soualah, Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. A link failure recovery algorithm for Virtual Network Function chaining. *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service Management*, pages 213–221, 2017.

[74] Babu Kothandaraman. Centrally Controlled Distributed VNF State Management. pages 37–42, 2015.

[75] Libin Liu, Hong Xu, Zhixiong Niu, Peng Wang, and Dongsu Han. U-HAUL : Efficient State Migration in NFV.

[76] Penghao Sun, Julong Lan, Junfei Li, Zehua Guo, Yuxiang Hu, and Tao Hu. Efficient flow migration for NFV with Graph-aware deep reinforcement learning. *Computer Networks*, 183(June):107575, 2020.

[77] Yang Wang, Gaogang Xie, Zhenyu Li, and Peng He. Transparent Flow Migration for NFV. pages 1–10, 2016.

[78] Godfrey Kibalya, Joan Serrat, Juan-luis Gorricho, Doreen Gift Bujjingo, Jonathan Sserugunda, and Peiying Zhang. A Reinforcement Learning Approach for Placement of Stateful Virtualized Network Functions.

[79] Hua Chai, Jiao Zhang, Zenan Wang, Jiaming Shi, and Tao Huang. A Parallel Placement Approach for Service Function Chain Using Deep Reinforcement Learning. *2019 IEEE 5th International Conference on Computer and Communications, ICCC 2019*, pages 2123–2128, 2019.

[80] Yikai Xiao, Qixia Zhang, Fangming Liu, Jia Wang, Miao Zhao, Zhongxing Zhang, and Jiaxing Zhang. NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning. *Proceedings of the International Symposium on Quality of Service, IWQoS 2019*, (1), 2019.

[81] Manabu Nakanoya, Yoichi Sato, and Hideyuki Shimonishi. Environment-adaptive sizing and placement of NFV service chains with accelerated reinforcement learning. *2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM 2019*, pages 36–44, 2019.

[82] Godfrey Kibalya, Joan Serrat, Juan-Luis Gorricho, Dorothy Okello, and Peiying Zhang. A deep reinforcement learning-based algorithm for reliability-aware multi-domain service deployment in smart ecosystems. *Neural Computing and Applications*, 0123456789, 2020.

[83] Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, Raouf Boutaba, Otto Carlos, and Muniz Bandeira. Orchestrating Virtualized Network Functions. 13(4):725–739, 2016.

[84] Godfrey Kibalya, Joan Serrat, Juan Luis Gorricho, Haipeng Yao, and Peiying Zhang. A novel dynamic programming inspired algorithm for embedding of virtual networks in future networks. *Computer Networks*, 179(May):107349, 2020.

[85] Nicolas Tastevin, Mathis Obadia, and Mathieu Bouet. A Graph Approach to Placement of Service Functions Chains. pages 134–141, 2017.

[86] Peiying Zhang, Xue Pang, Godfrey Kibalya, Neeraj Kumar, Shuqing He, and Bin Zhao. GCMD: Genetic Correlation Multi-Domain Virtual Network Embedding Algorithm. *IEEE Access*, 9:67167–67175, 2021.

[87] Gang Sun, Dan Liao, Dongcheng Zhao, Zhili Sun, and Victor Chang. Towards provisioning hybrid virtual networks in federated cloud data centers. *Future Generation Computer Systems*, 87:457–469, 2018.

[88] Gang Sun, Yayu Li, Hongfang Yu, Athanasios V Vasilakos, Xiaojiang Du, and Mohsen Guizani. Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks. *Future Generation Computer Systems*, 91:347–360, 2019.

[89] Khaled Hejja and Xavier Hesselbach. Online power aware coordinated virtual network embedding with 5G delay constraint. *Journal of Network and Computer Applications*, 124(February):121–136, 2018.

[90] .pdf.

[91] (No Title).

[92] Peiying Zhang, Sheng Wu, Miao Wang, Haipeng Yao, and Yunjie Liu. Topology based reliable virtual network embedding from a QoE perspective. *China Communications*, 15(10):38–50, 2018.

[93] Godfrey Kibalya, Joan Serrat, Juan Luis Gorricho, Rafael Pasquini, Haipeng Yao, and Peiying Zhang. A Reinforcement Learning Based Approach for 5G Network Slicing across Multiple Domains. In *15th International Conference on Network and Service Management, CNSM 2019*, 2019.

[94] Alex M. Andrew. Reinforcement Learning: An Introduction. *Kybernetes*, 27(9):1093–1096, 1998.

[95] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems*, pages 1043–1049, 1998.

[96] Haipeng Yao, Sihan Ma, Jingjing Wang, Peiying Zhang, Chunxiao Jiang, and Song Guo. A Continuous-Decision Virtual Network Embedding Scheme Relying on Reinforcement Learning. *IEEE Transactions on Network and Service Management*, 17(2):864–875, 2020.

[97] Magana Mora. Genetic Algorithms for Models Optimization for Recognition of Translation Initiation Sites Thesis by Arturo Magana Mora In Partial Fulfillment of the Requirements for the Degree of Master of Science. pages 1–63, 2011.

[98] Man Soo Han. Optimal path calculation for virtual networks using genetic algorithm. *International Journal of Advanced Computer Research*, 9(40):28–36, 2019.

[99] Isha Pathak and Deo Prakash Vidyarthi. A model for virtual network embedding across multiple infrastructure providers using genetic algorithm. *Science China Information Sciences*, 60(4):1–12, 2017.

[100] Sanghyeok Kim, Sungyoung Park, Kwonyong Lee, Youngjae Kim, and Siri Kim. VNF-EQ : dynamic placement of virtual network functions for energy efficiency and QoS guarantee in NFV. pages 2107–2117, 2017.

[101] Mohammad Ali Tahmasbi Nejad, Saeedeh Parsaeefard, Mohammad Ali Maddah-Ali, Toktam Mahmoodi, and Babak Hossein Khalaj. VSPACE: VNF Simultaneous Placement, Admission Control and Embedding. *IEEE Journal on Selected Areas in Communications*, 36(3):542–557, 2018.

[102] Lidia Ruiz, Ramón J Durán Barroso, Ignacio D E Miguel, Noemí Merayo, Juan Carlos Aguado, Ramón D E L A Rosa, Patricia Fernández, Rubén M Lorenzo, and Evaristo J Abril. Genetic Algorithm for Holistic VNF-Mapping and Virtual Topology Design. 8:55893–55904, 2020.

[103] Long Qu, Chadi Assi, and Khaled Shaban. Delay-Aware Scheduling and Resource Optimization with Network Function Virtualization. *IEEE Transactions on Communications*, 64(9):3746–3758, 2016.

[104] Fan-hsun Tseng, Xiaofei Wang, Li-der Chou, Han-chieh Chao, Senior Member, and Victor C M Leung. Dynamic Resource Prediction and Allocation for Cloud Data Center Using the Multiobjective Genetic Algorithm. 12(2):1688–1699, 2018.

[105] Soumen Swarnakar, Neeraj Kumar, Amit Kumar, and Chandan Banerjee. Modified Genetic Based Algorithm for Load Balancing in Cloud Computing. *2020 IEEE International Conference for Convergence in Engineering, ICCE 2020 - Proceedings*, pages 255–259, 2020.

[106] Yao Zhou. Study on genetic algorithm improvement and application. 2(May):1–73, 2012.

[107] Paul C. H. Chu. A Genetic Algorithm approach for combinatorial optimisation problems. *University of London*, (January):181, 1997.

[108] Harmony Search. Optimization Algorithm :. *Simulation*, 76(2):60–68, 2001.

[109] X. Z. Gao, V. Govindasamy, H. Xu, X. Wang, and K. Zenger. Harmony search method: Theory and applications. *Computational Intelligence and Neuroscience*, 2015, 2015.

[110] Javier Rubio-Loyola, Christian Aguilar-Fuster, Gregorio Toscano-Pulido, Rashid Mijumbi, and Joan Serrat-Fernandez. Enhancing metaheuristic-based online embedding in network virtualization environments. *IEEE Transactions on Network and Service Management*, 15(1):200–216, 2018.

[111] Xin She Yang. Harmony search as a metaheuristic algorithm. *Studies in Computational Intelligence*, 191:1–14, 2009.

[112] The Internet Topology Zoo.

[113] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, Felipe Huici, and Implementation Nsdi. ClickOS and the Art of Network Function Virtualization This paper is included in the Proceedings of the. 2014.

[114] Haipeng Yao, Xu Chen, Maozhen Li, Peiying Zhang, and Luyao Wang. A novel reinforcement learning algorithm for virtual network embedding. *Neurocomputing*, 284:1–9, 2018.

[115] Maninderpal Singh, Gagangeet Singh Singh Aujla, Amritpal Singh, Neeraj Kumar, and Sahil Garg. Deep Learning based Blockchain Framework for Secure Software Defined Industrial Networks. *IEEE Transactions on Industrial Informatics*, 3203(c):1, 2020.

[116] Long Gong, Yonggang Wen, Zuqing Zhu, and Tony Lee. Toward profit-seeking virtual network embedding algorithm via global resource capacity. *Proceedings - IEEE INFOCOM*, pages 1–9, 2014.

[117] Deval Bhamare, Mohammed Samaka, and Aiman Erbad. Exploring microservices for enhancing internet QoS. (April):1–18, 2018.

[118] Sangjin Hong, Jason P Jue, Qiong Zhang, Xi Wang, Hakki C Cankaya, Christopher She, and Motoyoshi Sekiya. Virtual optical network embedding in multi-domain optical networks. *2014 IEEE Global Communications Conference, GLOBECOM 2014*, pages 2042–2047, 2014.

[119] Pablo Caballero, Albert Banchs, Gustavo De Veciana, and Xavier Costa-Perez. Network Slicing Games: Enabling Customization in Multi-Tenant Mobile Networks. *IEEE/ACM Transactions on Networking*, 27(2):662–675, 2019.

[120] Gang Sun, Gungyang Zhu, Dan Liao, Hongfang Yu, Xiaojiang Du, and Mohsen Guizani. Cost-Efficient Service Function Chain Orchestration for Low-Latency Applications in NFV Networks. *IEEE Systems Journal*, 13(4):3877–3888, 2019.

[121] D Michalopoulos, C Sartori, V Sciancalepore, N Sastry, O Holland, S Tayade, B Han, D Bega, D Aziz, P Ameigeiras, D Lopez, L Lorca, A Nakao, and H Flinck. 5g n s : p 1 – c , p , a. (May):70–71, 2017.

[122] Oussama Soualah, Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. A Green VNF-FG Embedding Algorithm. *2018 4th IEEE Conference on Network Softwarization and Workshops, NetSoft 2018*, (NetSoft):449–455, 2018.

[123] Khaled Hejja and Xavier Hesselbach. Online power aware coordinated virtual network embedding with 5G delay constraint. *Journal of Network and Computer Applications*, 124(February):121–136, 2018.

[124] Haotong Cao, Yongan Guo, Yue Hu, Shengchen Wu, Hongbo Zhu, and Longxiang Yang. Location Aware and Node Ranking Value-Assisted Embedding Algorithm for One-Stage Embedding in Multiple Distributed Virtual Network Embedding. *IEEE Access*, 6:78425–78436, 2018.

[125] Mahdi Dolati, Seyedeh Bahereh Hassanpour, Majid Ghaderi, and Ahmad Khonsari. DeepViNE: Virtual Network Embedding with Deep Reinforcement Learning. *INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2019*, pages 879–885, 2019.

[126] Haotong Cao, Longxiang Yang, and Hongbo Zhu. Novel Node-Ranking Approach and Multiple Topology Attributes-Based Embedding Algorithm for Single-Domain Virtual Network Embedding. *IEEE Internet of Things Journal*, 5(1):108–120, 2018.

[127] N M Mosharaf, Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. *Proceedings - IEEE INFOCOM*, pages 783–791, 2009.

[128] Yang Wang, Qian Hu, and Xiaojun Cao. A branch-and-price framework for optimal virtual network embedding. 94:318–326, 2016.

[129] Zeheng Yang and Yongan Guo. An exact virtual network embedding algorithm based on integer linear programming for virtual network request with location constraint. *China Communications*, 13(8):177–183, 2016.

[130] Haotong Cao, Yongxu Zhu, Gan Zheng, and Longxiang Yang. A Novel Optimal Mapping Algorithm with Less Computational Complexity for Virtual Network Embedding. *IEEE Transactions on Network and Service Management*, 15(1):356–371, 2018.

[131] Haotong Cao, Longxiang Yang, and Jinbo Chen. An exact VNE algorithm based on optimization theory. *2016 IEEE International Conference on Consumer Electronics-Asia, ICCE-Asia 2016*, pages 1–4, 2017.

[132] Roberto Amazonas, Juan Felipe, Miguel Molina, and Xavier Hesselbach-serra. Journal of Network and Computer Applications A novel paths algebra-based strategy to flexibly solve the link mapping stage of VNE problems. 36:1735–1752, 2013.

[133] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding. *ACM SIGCOMM Computer Communication Review*, 38(2):17–29, 2008.

[134] Peiying Zhang, Fanglin Liu, Chunxiao Jiang, and Godfrey Kibalya. Virtual Network Embedding Strategy Based on Load Balancing Genetic Algorithm. pages 1–17.

[135] Zhongbao Zhang, Sen Su, Junchi Zhang, Kai Shuang, and Peng Xu. Energy aware virtual network embedding with dynamic demands: Online and offline. *Computer Networks*, 93:448–459, 2015.

[136] Christian Aguilar-Fuster, Mahboobeh Zangiabady, Jose Zapata-Lara, and Javier Rubio-Loyola. Online Virtual Network Embedding Based on Virtual Links' Rate Requirements. *IEEE Transactions on Network and Service Management*, 15(4):1630–1644, 2018.

[137] Xiang Cheng, Sen Su, Zhongbao Zhang, Kai Shuang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology awareness and optimization. *Computer Networks*, 56(6):1797–1813, 2012.

[138] Fangjin Zhu and Hua Wang. A modified ACO algorithm for virtual network embedding based on graph decomposition. *Computer Communications*, 80:1–15, 2016.

[139] Andreas Blenk, Patrick Kalmbach, Johannes Zerwas, Michael Jarschel, Stefan Schmid, and Wolfgang Kellerer. NeuroViNE: A Neural Preprocessor for Your Virtual Network Embedding Algorithm. *Proceedings - IEEE INFOCOM*, 2018-April:405–413, 2018.

[140] Rashid Mijumbi, Joan Serrat, Juan Luis Gorricho, and Raouf Boutaba. A Path Generation Approach to Embedding of Virtual Networks. *IEEE Transactions on Network and Service Management*, 12(3):334–348, 2015.

[141] Francesco Bianchi and Francesco Lo Presti. A markov reward based resource-latency aware heuristic for the virtual network embedding problem. *Performance Evaluation Review*, 44(4):57–68, 2017.

[142] Shuiqing Gong, Jing Chen, Siyi Zhao, and Qingchao Zhu. Virtual Network Embedding with Multi-attribute Node Ranking Based on. 10(2):522–541, 2016.

[143] D Cotroneo, L De Simone, A K Iannillo, A Lanzaro, R Natella, Jiang Fan, and Wang Ping. Network function virtualization: Challenges and directions for reliability assurance. *Proceedings - IEEE 25th International Symposium on Software Reliability Engineering Workshops, ISSREW 2014*, pages 37–42, 2014.

[144] Guido Marchetto, Riccardo Sisto, Jalolliddin Yusupov, and Adlen Ksentini. Virtual Network Embedding with Formal Reachability Assurance. *14th International Conference on Network and Service Management, CNSM 2018 and Workshops, 1st International Workshop on High-Precision Networks Operations and Control, HiPNet 2018 and 1st Workshop on Segment Routing and Service Function Chaining, SR+SFC 2*, (Cnsm):368–372, 2018.

[145] Estimating Consumption plan costs in Azure Functions | Microsoft Docs.

[146] S Kolliopoulos. Exact and approximation algorithms for network flow and disjoint-path problems. 1998.

[147] Peter Lammich B and S Reza Sefidgar B. Interactive Theorem Proving - 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings. 9807:219–234, 2016.

[148] Gang Sun, Hongfang Yu, Vishal Anand, and Lemin Li. A cost efficient framework and algorithm for embedding dynamic virtual network requests. *Future Generation Computer Systems*, 29(5):1265–1277, 2013.

[149] Defang Li, Peilin Hong, Kaiping Xue, and Jianing Pei. Availability Aware VNF Deployment in Datacenter through Shared Redundancy and Multi-Tenancy. *IEEE Transactions on Network and Service Management*, 16(4):1651–1664, 2019.

[150] Siri Kim, Yunjung Han, and Sungyong Park. An Energy-aware Service Function Chaining and Reconfiguration Algorithm in NFV. pages 54–59, 2016.

[151] Peiying Zhang, Yanrong Hong, X U E Pang, and Chunxiao Jiang. VNE-HPSO : Virtual Network Embedding Algorithm Based on Hybrid Particle Swarm Optimization. pages 213389–213400, 2020.

[152] Ilhem Fajjari, Nadjib Ait Saadi, Guy Pujolle, and Hubert Zimmermann. VNE-AC: Virtual network embedding algorithm based on ant colony metaheuristic. *IEEE International Conference on Communications*, 2011.

[153] Wenting Wei, Student Member, Huaxi Gu, Tong Zhou, Xuanzhang Liu, and Wanyun Lu. Energy Efficient Virtual Machine Placement With an Improved Ant Colony Optimization Over Data Center Networks. *IEEE Access*, 7:60617–60625, 2019.

[154] Ali Hmaity, Marco Savi, Francesco Musumeci, Massimo Tornatore, and Achille Pattavina. Protection strategies for virtual network functions placement and service chains provisioning. *Networks*, 70(4):373–387, 2017.

[155] Mohammad Ali Khoshkholghi, Javid Taheri, Deval Bhamare, and Andreas Kassler. Optimized Service Chain Placement Using Genetic Algorithm. *Proceedings of the 2019 IEEE Conference on Network Softwarization: Unleashing the Power of Network Softwarization, NetSoft 2019*, pages 472–479, 2019.

[156] Qian Li, San Yang Liu, and Xin She Yang. Influence of initialization on the performance of metaheuristic optimizers. *Applied Soft Computing Journal*, 91:106193, 2020.

[157] Tobias Blickle and Lothar Thiele. A Mathematical Analysis of Tournament Selection. *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 9–16, 1995.

[158] Amir Mohamad. On Demonstrating the Gain of SFC Placement with VNF Sharing at the Edge. 2019.

[159] Roohollah Amiri, Hani Mehrpouyan, Lex Fridman, Ranjan K Mallik, Arumugam Nallanathan, and David Matolak. A Machine Learning Approach for Power Allocation in HetNets Considering QoS. *IEEE International Conference on Communications*, 2018-May, 2018.

[160] Se Rim Park and Jin Won Lee. A fully convolutional neural network for speech enhancement. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2017-Augus(2):1993–1997, 2017.