Algorithmes exacts et heuristiques pour le chainage de services réseaux virtualisés

Omar Houidi

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2020IPPAS005

Thèse de doctorat

Algorithms for Virtual Network Functions chaining

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom SudParis

École doctorale de l'Institut Polytechnique de Paris, n°626 (ED IP Paris)
Spécialité de doctorat : Réseaux, Information et Communications

Thèse présentée et soutenue à Évry, le 25/06/2020, par

OMAR HOUIDI

Composition du Jury :

Guillaume Urvoy-Keller Professeur, CNRS, I3S	Président
Lynda Mokdad Professeure, Université Paris-Est Créteil	Rapportrice
Frédéric Giroire Chargé de recherche (HDR), CNRS, I3S	Rapporteur
Marcelo Dias de Amorim Directeur de recherche au CNRS, LIP6/UPMC	Examinateur
Nadjib Aitsaadi Professeur, Université de Versailles Saint-Quentin-en-Yvelines	Examinateur
Michel Kieffer Professeur, CentraleSupélec, Université Paris-Sud	Examinateur
Djamal Zeghlache Professeur, Télécom SudParis, Institut Mines-Télécom	Directeur de thèse
Wajdi Louati Maître assistant, ENIS, ReDCAD	Invité

Abstract

Network Function Virtualization (NFV) is an innovative emerging concept that decouples network functions (such as firewalls, DNS, NATs, load balancers, etc.) from dedicated hardware devices (the traditional expensive middleboxes). This decoupling enables hosting of network services, known as Virtualized Network Functions (VNFs), on commodity hardware (such as switches or servers) and thus facilitates and accelerates service deployment and management by providers, improves flexibility, leads to efficient and scalable resource usage, and reduces costs. This paradigm is a major turning point in the evolution of networking, as it introduces high expectations for enhanced economical network services, as well as major technical challenges.

One of the main technical challenges in this domain is the optimal placement of the VNFs within the hosting infrastructures. This placement has a critical impact on the performance of the network, as well as on its reliability and operation cost. The VNF Placement and Chaining Problem is NP-Hard and there is a need for placement approaches that can scale with problem size and find good solutions in acceptable times. The overarching goal of this thesis is to enable dynamic virtual network resources provisioning to deal with demand fluctuation during the virtual network lifetime, and to enhance the substrate resource usage. Reserving a fixed amount of resources is inefficient to satisfy the VNF resource requirements. To cope with these problems, we propose dynamic resource management strategies.

In this thesis, both exact and heuristic algorithms are designed and evaluated in terms of optimality, complexity, ability to scale, and compared with the state of the art. Elastic mechanisms and scaling algorithms are first presented to improve adaptation and deployment of virtualized network functions in NFV infrastructures to support increasing demand while increasing provider's revenue. Since network providers not only need to control, classify and steer user and application traffic flows in their dedicated slices but also want to extend their already acquired and operational virtual networks or slices with additional service graphs, the thesis proposes extension algorithms of already hosted network functions graphs without disrupting initially deployed and active service instances. The proposed algorithms extend already deployed network services and functions graphs to respond to new demands while taking into account the constraint of minimizing the impact on the original service graphs. The extension algorithms are particularly useful and suitable for situations where already deployed graphs need to be enhanced with new features and properties (adding new functions) and modified to react to degradation and attacks such as removing

a fraction of the graph and replacing with new complex and composed functions into more capable and uncompromised graphs.

The thesis also addresses the VNF placement and chaining problem in an online and in a batch mode to improve performance in terms of longer time reward. An enhanced Reinforcement Learning-based approach is also proposed to improve the long term reward beyond what the previous methods can achieve. This is analyzed and realized for a load balancing objective but can be adjusted for other criteria.

Keywords:

Network Function Virtualization, Resource Allocation, Optimization, Placement, Modeling, and Performance.

Résumé

Cette thèse traite du placement optimal et heuristique de fonctions réseau et de chaînes de fonction réseau dans des infrastructures cloud et réseau virtualisées. L'émergence de la virtualisation des fonctions réseau, connu sous l'acronyme NFV pour Network Function Virtualization, permet de découpler les fonctions réseau en mode logiciel du matériel d'hébergement et de s'appuyer sur des serveurs génériques et d'éviter l'usage de, et la dépendance à, des matériels dédiés voire propriétaires.

Le placement de fonctions réseau virtualisées (représentées par VNF, Virtualized Network Functions) est NP-Difficile puisqu'il s'agit de projeter un petit graphe de ressources virtuelles sur un graphe plus grand (graphe de l'infrastructure d'hébergement). Les solutions optimales, en particulier la programmation linéaire en nombre entier (ILP), ne passent pas à l'échelle. Sachant que la demande est dynamique et peut varier dans le temps et que le réseau est lui même variable dans le temps, il est important de prévoir des adaptations des placements. Cela peut s'effectuer par de l'élasticité sur les ressources d'hébergement réservées à une fonction réseau virtualisée, à un graphe de service réseau, et par une extension du graphe de service lui même en fonction du contexte et des exigences des utilisateurs ou tenants.

La thèse propose une famille d'algorithmes pour le placement de chaînes de services (ou fonctions) réseau avec la possibilité d'étendre les ressources d'hébergement des VNFs (c'est à dire assurer l'élasticité du service d'hébergement en augmentant les ressources allouées ou en générant plusieurs instances de VNFs pour écouler le trafic et répondre à la demande) en plus du placement initial. Une solution en programmation en nombre entier est élaborée et aussi utilisée comme référence pour une comparaison avec l'état de l'art et avec les extensions proposées. L'optimisation étant effectuée en instantanée au fur à mesure de l'arrivée des demandes, une à la fois, une solution qui regroupe plusieurs demandes pour y répondre simultanément, en élaborant un graphe composite, permet d'améliorer les performances. Cette approche connue sous le nom de "batch" n'améliore que partiellement la performance, la récompense sur le long terme (en efficacité, minimisation des ressources consommées, et en équilibrage de charge) est nécessairement limitée.

La thèse s'est penchée aussi sur l'extension de graphes de services réseau, de tenants, déjà déployés, en adoptant une approche de type arbre de recouvrement, Spanning Tree. Plus spécifiquement une modélisation du problème en un "Steiner Tree Problem" a conduit à

des performances proches de l'optimal pour des extensions de graphes au fil des demandes, en les traitant séparément. Les travaux de thèse sont par la suite revenus sur la rentabilité sur le long terme des algorithmes, en approchant le placement de chaînes de services et fonctions réseau comme un objectif long terme via de l'apprentissage par renforcement en se souciant plus de la rentabilité et de l'efficacité long terme des algorithmes contrairement aux approches visant exclusivement l'optimalité instantanée à chaque nouvelle demande.

Cette thèse a permis de faire avancer autant que faire se peut l'état de l'art du placement optimal dans les infrastructures cloud et réseau partagées. Notamment, le problème, et besoin, d'extension de graphes, de tenants, déjà déployés, sans perturber l'hébergement initial, a reçu peu d'attention. Pourtant ce besoin est essentiel pour répondre à des déploiements additionnels de nouvelles fonctions et services réseau, et pour réagir aux dégradations et à l'accroissement de la demande, aux attaques et pour assurer l'introduction de nouveaux services et fonctions de sécurité autour du graphe initial. L'approche peut aussi répondre à des modifications de graphes et d'isolations d'une partie (défectueuse ou compromise) d'un graphe et de son remplacement par d'autres services et graphes fiables et non altérés.

Mots-clés:

Virtualisation des Fonctions Réseau, Allocation des Ressources, Optimisation, Placement, Modélisation, et Performance.

Acknowledgements

At the end of my PhD thesis, I would like to thank all those who made this thesis possible.

I would like to express my sincere gratitude to my supervisor, Pr. Djamal Zeghlache, for supporting me with a continuous stream of intellectual, moral and financial assistance. He has always encouraged me to pursue my own ideas and provided invaluable feedback to improve the quality of my research. I would like to extend my gratitude and acknowledgments to Dr. Oussama Soualah, and Dr. Wajdi Louati for their supports, patient guidances, and insightful comments. I will always be grateful.

My cordial thanks to the reviewers of my thesis, Dr. Frédéric Giroire, and Pr. Lynda Mokdad.

I am grateful for Pr. Guillaume Urvoy-Keller, Dr. Marcelo Dias De Amorim, Pr. Michel Kieffer, and Pr. Nadjib Ait Saadi for accepting to examine my thesis and attend my dissertation defence.

I would like to thank all my family and friends in Tunisia and France, and all my colleagues in the SAMOVAR laboratory. Last but not least, I would like to thank everyone who has contributed to this thesis, directly or indirectly.

Contents

List of Figures

List of Tables

Abbreviations

5G	**Fifth-G**eneration
API	**A**pplication **P**rogramming **I**nterface
BSS	**B**usiness **S**upport **S**ystems
CAPEX	**Cap**ital **Ex**penditure
COTS	**C**ommercial-**O**ff-**T**he-**S**helf
CPE	**C**ustomer **P**remises **E**quipment
CPU	**C**entral **P**rocessing **U**nit
CSP	**C**ommunications **S**ervice **P**rovider
DNS	**D**omain **N**ame **S**ystem
DPI	**D**eep **P**acket **I**nspection
EMS	**E**lement **M**anagement **S**ystem
ETSI	**E**uropean **T**elecommunications **S**tandards **I**nstitute
HVS	**H**igh **V**olume **S**erver
IETF	**I**nternet **E**ngineering **T**ask **F**orce
ILP	**I**nteger **L**inear **P**rogramming
IoT	**I**nternet **o**f **T**hings
ISP	**I**nternet **S**ervice **P**rovider
MDP	**M**arkov **D**ecision **P**rocess
MILP	**M**ixed **I**nteger **L**inear **P**rogramming
NAS	**N**etwork **A**ttached **S**torage
NAT	**N**etwork **A**ddress **T**ranslation
NCT	**N**etwork **C**onnectivity **T**opology
NF	**N**etwork **F**unction
NFP	**N**etwork **F**orwarding **P**ath

NFV	**N**etwork **F**unction **V**irtualisation
NFVI	**N**etwork **F**unction **V**irtualisation **I**nfrastructure
NFVI-PoP	**N**etwork **F**unction **V**irtualisation **I**nfrastructure **P**oint **o**f **P**resence
OPEX	**Op**erational **Ex**penditure
OSS	**O**perations **S**upport **S**ystems
PNF	**P**hysical **N**etwork **F**unction
QoS	**Q**uality **o**f **S**ervice
RL	**R**einforcement **L**earning
SDN	**S**oftware **D**efined **N**etworks
SFC	**S**ervice **F**unction **C**hain
SLA	**S**ervice **L**evel **A**greement
TE	**T**raffic **E**ngineering
VIM	**V**irtualised **I**nfrastructure **M**anager
VNE	**V**irtual **N**etwork **E**mbedding
VNF	**V**irtualised **N**etwork **F**unction
VNFC	**V**irtualised **N**etwork **F**unction **C**omponent
VNF-FG	**V**irtualised **N**etwork **F**unction **F**orwarding **G**raph

Chapter 1

Introduction

Nowadays, Information Technology companies and academics are interested more and more about the Network Function Virtualization (NFV) concept. In fact, Communications Service Providers (CSPs) and Internet Service Providers (ISPs) are facing competition from Over-the-top (OTT) media services and web services, experiencing declining average revenue per user (ARPU) and feeling the pressure to innovate rapidly to respond to new trends such as 5G, IoT (Internet of Things), and edge computing.

Traditional network services are built by chaining together proprietary single-function boxes (middleboxes). The design of these services is custom, the underlying equipments are expensive, require lengthy procurement times and cannot be shared with any other service. Once deployed, the operations and management of these services is largely non-automated, with each box presenting its own management interface. This technique of creating network services is very expensive, and offers no practical way of creating dynamic services.

Network Function Virtualization (NFV) is a technology that can greatly assist in solving these business challenges. Once virtualized, the Virtual Network Functions (VNFs) can be hosted on an industry-standard server or commodity hardware. Virtualization does not stop at replacing physical boxes with virtual machines, but can go further by using microservices, containers, and cloud native architectures. Managing the Lifecycle of these services (such as initial deployment, configuration changes, upgrades, scale-out, scale-in, self-healing, etc.), can also be automated. These VNFs can also be chained and managed in

a dynamic and automated fashion. All these advances enable the creation and management of agile network services.

This thesis addresses the optimal placement of these virtualized network functions and services in hosting infrastructures. That is the problem of mapping VNFs and service function chains (such as chained VNFs) in NFV-Is (Network Function Virtualization - Infrastructures). This problem received a lot of attention from the community and several contributions attempted answering these challenges. Most of the existing solutions focus nevertheless on static placement of VNFs and overlook the dynamic aspect of the problem that arises through ever-changing resource availability in the cloud datacenters and the continuous mobility of the users. Recently there have been some attempts taking into account the encountered dynamic variations. Allocating the VNFs in the *dynamic* NFV environment requires *flexible* and *adaptive* resource provisioning algorithms to deal with the changing demands of the VNFs during their lifetime. This thesis focuses on this specific issue of providing placement algorithms that can cope with some of these variations via elastic VNFs, service chaining, and VNF forwarding graphs (VNF-FGs).

In this chapter we describe the VNF placement and chaining problem and summarizes the contributions of the thesis to facilitate the reading of this manuscript and ease understanding.

1.1 Research Problem and Objectives

- **VNF-Forwarding Graph (VNF-FG) Embedding Problem**

One focus of the thesis is the placement of chains of network services and functions in an NFV context to support multiple tenants and applications. This problem is often referred as Service Function Chaining (SFC in the IETF terminology), also known as Virtualized Network Functions Forwarding Graph (VNF-FG in the ETSI terminology) to emphasize that there is a sequencing and a specific connectivity requested as a complex service graph involving multiple interconnected VNFs. A **Service Function Chain (SFC)** defines an ordered set of abstract service functions (SFs) (VNFs in the case of ETSI) and ordering

constraints that must be applied to packets and/or frames and/or flows selected as a result of classification. The implied order may not be a linear progression as the architecture allows for SFCs that copy to more than one branch, and also allows for cases where there is flexibility in the order in which service functions need to be applied. SFCs may contain cycles, that is traffic may need to traverse one or more SFs within an SFC more than once. A **Virtualised Network Function Forwarding Graph (VNF-FG)** is composed of interconnected VNFs providing a dedicated end-to-end service. It may also contain one or more Network Forwarding Path (NFP). A NFP is an ordered list of connection points forming a chain of Network Functions (NFs), along with policies associated to the list. When the requested service concerns a dedicated virtual network with embedded virtualized functions and services, the dedicated functionality and feature enhanced virtual network is called a slice. A **network slice** is composed of a collection of resources that, appropriately combined, meet the service requirements of the use case that such a slice supports. Each slice provides a set of network capabilities and performance levels that may be suitable for a set of service types. The services can be mapped onto different slices, according to operator policies and business strategies. Each slice has a certain set of attributes, for example, end-to-end low latency or high bandwidth, which can make it more appropriate for a certain types of services. The objective of the thesis is to propose placement algorithms for such chaining while respecting the VNF dependencies and the forwarding paths or flow paths specified in the request. The proposed algorithms apply also to slices seen as just yet another, more complex, service graph composed of nodes (corresponding to hosted services) and edges (corresponding to links or relations between the services).

SFC and VNF-FG embedding are key requirements identified by the IETF-SFC and ETSI-NFV specifications on resource allocation that consists of finding the best (in some defined optimization sense) hosts for VNFs in the network infrastructure (NFV-I). That is, the resource optimization in question must be accomplished with regard to a specific objective (e.g. maximization of remaining network resources, minimization of power consumption, optimization of a specific QoS metric, etc.). The ETSI-NFV and IETF-SFC working groups propose their own terminologies to express respectively the main concepts of NFV and SFC. The terminologies as usual are different but there is a certain correspondence between their components and concepts. In terms of optimization problems, both VNF-FGs and SFCs can

be addressed by all our approaches, since we use the most general case of VNF-FG (more precisely, the derived NCT: Network Connectivity Topology) as input for the optimization (i.e., the SFCs can be considered as a particular case).

The VNF-FG embedding problem can be seen as a generalization of the well-known Virtual Network Embedding (VNE) problem with additional constraints related to the chaining and dependencies between the VNFs. The VNE is known to be NP-hard [4] and, since a VNF-FG embedding is a generalization of VNE, it is also NP-hard. Generally speaking, the problem consists in mapping the virtual resources to candidate substrate resources. All the VNFs composing a chain must be mapped to consider the mapping successful, the placement concerns thus the entire chain that must be embedded at once meaning that the hosting resources (the substrate network or the NFV-I resources) are also reserved and consumed jointly. The two stage mapping process, virtual node and virtual link mappings, of VNE also apply to VNF-FG embedding. The mapping does not need to follow a two step approach, however, and the mapping can be done jointly and simultaneously for both nodes and links (or equivalently services and relationships between these nodes). In VNF-FG embedding, each VNF is annotated with a type to emphasize the functions hosting requirements their features and artifacts; so the VNFs can be hosted in the appropriate and capable physical hosts.

Figure 1.1 shows the deployment of an end-to-end network service between two substrate nodes in an NFV-I. It illustrates the embedding stage involving: service orchestration, service management, the virtualization layer, and the NFVI. The orchestrator is responsible of the embedding stage; it is in charge of the management and orchestration of software resources and the virtualized hardware infrastructure to realize networking services. The service is composed by a set of VNFs that together provide a specific functionality. The virtualization layer abstracts the physical resources and anchors the VNFs to the virtualized infrastructure. It ensures that the VNF lifecycle is independent of the underlying hardware platforms by offering standardized interfaces. This type of functionality is typically provided in the forms of VMs and of their hypervisors that are typically located in data centers, at edge nodes, network nodes, and in end-user facilities.

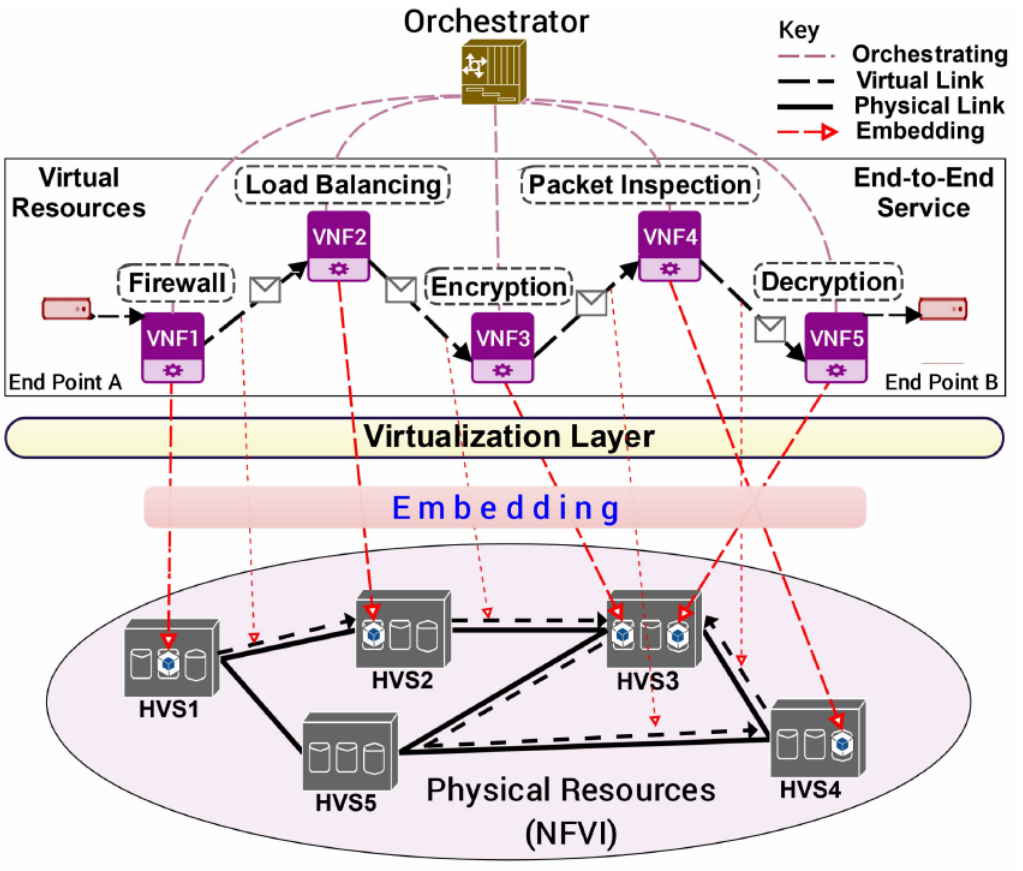


FIGURE 1.1: VNFs forwarding-graph embedding [1]

A High Volume Server (HVS) is considered as a physical network node in an NFV-based network architecture, which uses a hypervisor to manage virtual machines, according to the availability of resources (CPU, Disk, RAM). The NFVI is composed of NFV infrastructure Points-of-Presence (NFVI-PoPs), including resources for computation (such as HVS, RAM), storage (such as disk storage), and networking (such as switches and routers). VMs running on top of HVSs can host one or more VNFs of the same type. An example to illustrate the embedding stage is shown in Figure 1.1. The orchestrator runs a VNF-FG embedding algorithm which makes embedding decisions, according to the adopted optimization objectives.

This thesis addresses these stated embedding needs while taking into account the unavoidable variations in demands in such environments during the lifetime of the VNFs and their associated chains and the ensuing hosting resources elasticity requirements. To describe the problem, illustrated in Figure 1.2, we split the VNF-FG placement and chaining problem

into two sub-problems:

- Initial VNF-FG placement (Initial VNF placement and chaining): The aim of this stage is to efficiently map the initial VNF-FG request onto the substrate network.
- Elastic VNF-FG placement (Dynamic resource management): This stage deals with i) the resource demand fluctuation of the embedded VNF-FG (scaling, migration, extension, etc.) and ii) the re-optimization of the substrate network usage (The resources have to be re-allocated to adapt to dynamic changes in traffic volumes).

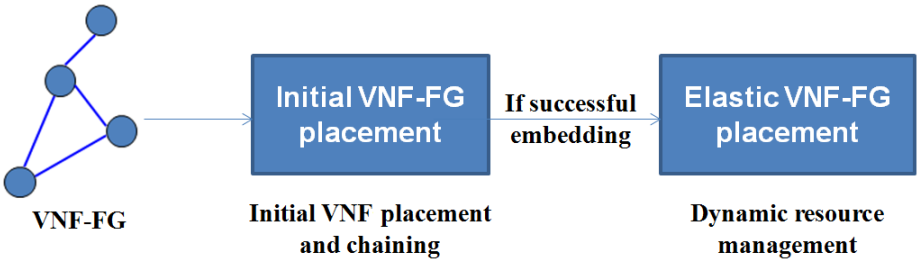


FIGURE 1.2: The VNF-FG placement and chaining sub-problems

- **Scope of the thesis**

In line with the thesis scope, the work focuses on VNF-FG embedding and related key challenges:

- VNF placement and chaining:
  - Where to embed the clients' requests ?
  - How to select the substrate resources ?
  - How to "optimally" manage the NFV-I ?
- Extension of already deployed forwarding graphs in the NFV-I (by adding new forwarding paths, new service chains, new services, and virtualized network functions).
- Horizontal scaling (Add/remove virtualized resources) and Vertical scaling (Reconfigure the capacity/size of existing virtualized resources).
- Migration of VNFs instances (where VNFs are moved to more appropriate servers especially servers with more resources).

1.2 Research Contributions

We summarize for convenience the contributions of the thesis with respect to the originally identified challenges and defined research goals.

- **Virtual Network Function Scaling (Houidi et al. [5], GLOBECOM 2017)**

NFV has been revolutionizing the way networking services are deployed but also requires the dynamic scaling of resources during the lifecycle management of VNFs with increasing service demand. The first contribution proposes an Integer Linear Programming (ILP) approach and a Greedy heuristic to address this NP-Hard scaling problem. Both horizontal (scale out/in) and vertical (scale up/down) scaling are considered and the scalability of the ILP is addressed by reducing the number of candidate hosts in the search space. Extensive simulations evaluate the performance of the algorithms in successful scalings and VNF migrations and highlight the importance of using the algorithms to tidy up the infrastructure to accept more users. The results show that the ILP can outperform the Greedy heuristic if appropriate actions are selected.

- **Dynamic VNF Forwarding Graph Extension Algorithms (Houidi et al. [6], NCA 2018, and Houidi et al. [7], TNSM 2020)**

The second contribution proposes a set of algorithms to extend initially deployed Virtualised Network Functions Forwarding Graphs (VNF-FGs) or Service Function Chains (SFCs) previously requested and acquired by tenants. The latter are likely to request extension of their already acquired, deployed and running dedicated slices to embed additional networking functions to handle for instance new application flows requiring additional classification, authentication, authorization, processing, traffic steering, etc. The tenants (or end-user service providers) are also likely to introduce gradually their services as needs arise and as their consumer base and profiles evolve. Increasing traffic load often will also require extension of the service graphs and chaining such as introducing additional load balancers and servers, new security functions and simply new VNFs related to new tenant services. The related service graph extension problem is addressed first through an ILP algorithm that

serves as a reference for performance comparisons with a set of proposed heuristic algorithms. A heuristic directly inspired by and based on the ILP and the proposed alternative approaches are shown to scale much better than the ILP while providing good solutions. The solutions include an adaptation of an Eigendecomposition-based approach that provides very good performance for highly connected graphs. The problem is also cast into the well know Steiner Tree search problem whose complexity and performance are formally characterized in the literature. The Steiner based algorithm is shown to provide the best overall tradeoff (in rejection rate, complexity, quality, cost, and revenue) with performance closest to optimal.

- **Enhanced Reinforcement Learning Approach for VNF-FG Embedding (Houidi et al. [8], PIMRC 2020)**

The third contribution presents a set of approaches to address the VNF-FG embedding problem to increase provider long-term revenue as a general objective. The placement is achieved through an Integer Linear Programming (ILP) algorithm that serves as a reference for performance comparisons with a set of proposed heuristic algorithms. Since ILP based approaches do not scale well with problem size, the proposed algorithm R_ILP (for reduced exploration) selects a limited number of candidate hosts from the infrastructure to control complexity. Since the online R_ILP treats the requests sequentially, a batch strategy (BR_ILP) that operates on a set of requests is also proposed to improve performance. However, these methods are efficient only in finding instant solutions (operating in short-term to make real-time decisions). They usually make decisions only based on network knowledge for a snapshot or a short time past. This contribution proposes a Reinforcement Learning (RL) based approach, called EQL (Enhanced Q-Learning), aiming at balancing the load on hosting infrastructures. EQL employs RL to learn the network and control it based on the learned experience (learning the usage of the physical resources used in each node). A key point of EQL algorithm is that the time needed for decision making is reduced. In fact, the training phase is accelerated by injecting expert knowledge as side and driving knowledge to the RL algorithm. Results from extensive simulations, based on realistic and large scale topologies, report the performance in terms of acceptance rate of service requests, quality of the mapping of each request, scalability and achieved revenues.

1.3 Thesis Organization

This dissertation is organized into six core chapters. Besides the present chapter introducing the context, objectives and contributions of the thesis, the manuscript is organized as follows:

- **Chapter 2** provides the background information related to this thesis. It presents an overview of NFV and SDN. The chapter also gives an overview of the state of the art of the resource allocation problem in NFV and reviews networking challenges in cloud environments.

- **Chapter 3** addresses the problem of VNF reconfiguration (scaling, migration) with the main objective of reducing the rejection of VNF scaling requests when faced with rising demand and traffic load. An ILP model is proposed and a new Greedy reconfiguration algorithm is also reported for the purpose. Extensive evaluation for the proposed models shows that the solutions can efficiently scale VNFs and forwarding graphs with good execution time and scaling requests acceptation performance over distributed and dynamically varying cloud environments.

- **Chapter 4** addresses the problem of VNF-FG extension with the main objective of maximizing provider revenue by reducing the rejection of extension requests when faced with dynamic demand. An ILP model is presented as an exact algorithm acting as a baseline for comparison and as the solution for small problem sizes. To improve scalability, two heuristic algorithms are also proposed: a new Steiner Tree-based algorithm and an Eigendecomposition-based approach. A comparative performance evaluation with the exact model is reported to assess the efficiency and scalability of the proposed algorithms.

- **Chapter 5** proposes approaches for the VNF-FG placement and chaining problem in an online and a batch mode. The online strategy exploits an ILP derived from a mathematical model of the VNF-FG embedding problem. The ILP model is derived and used to find optimal placement solutions to host the requests when problem size is small. To control complexity and improve scalability for larger graph sizes, the

ILP explores only a reduced subset of candidate hosts (R_ILP). A batch mode algorithm (BR_ILP), to enhance further overall performance, uses the R_ILP algorithm to process a group of requests, queued in a batch window, by serving in priority requests that generate higher revenues for the providers while aiming at maximizing the number of served requests. An Enhanced Q-Learning-Based Approach (EQL) is also proposed and compared with the ILP solutions. The training phase of this approach is improved and accelerated by injecting expert knowledge to increase provider revenue. The performance of these algorithms is compared in terms of acceptance rate of requests, quality of the solutions, scalability and achieved revenues.

- **Chapter 6** summarizes the thesis contributions and presents future research directions and perspectives.

Chapter 2

State of the Art

2.1 Introduction

Software-Defined Networking (SDN) and Network Function Virtualization (NFV) are enabling network programmability and the automated provisioning of virtual networking services. Combining these new paradigms can overcome the limitations of traditional clouds and networks by enhancing their dynamic networking capabilities. Since these evolutions have motivated this thesis and our investigations, this chapter on the state of the art will provide an overview of NFV architecture, SDN, resource allocation challenges and reflect the convergence trend between cloud computing, software networks, and the virtualization of networking functions.

This chapter provides in the first part an overview of the NFV and SDN architectures. The convergence between cloud computing and both SDN and NFV is discussed in the second part. Finally, the third part describes the relation between the VNF placement and chaining problem and the traditional VNE problem and surveys some existing models and algorithms of resources mapping in network environments.

It is important to clarify that the two sub-problems of VNF-FG embedding: i) Initial VNF-FG placement, and ii) Elastic VNF-FG placement (Dynamic resource management) are investigated.

2.2 Network Function Virtualization (NFV)

2.2.1 Network Services Before NFV

Communication Service Providers (CSPs) go beyond simply providing network connectivity for their enterprise customers. They also offer additional services and network functions like Network address translation (NAT), Firewall, Encryption, Domain Name Service (DNS), Caching, etc. Traditionally, these network functions were deployed using proprietary hardware at the customer premises. This approach provides additional revenue but deploying multiple proprietary devices is costly and makes upgrades difficult (i.e., every time a new network function is added to a service, a truck roll is required to install the dedicated new hardware device). Consequently, service providers began exploring ways to reduce cost and accelerate deployments through Network Function Virtualization (NFV).

2.2.2 What is NFV?

Network Function Virtualization (NFV) [3], [9], [10] is an innovative emerging way to design, deploy, and manage networking services by decoupling functions (such as firewalls, DPIs, load balancers, etc.) from dedicated hardware and moving them to virtual servers.

Several use cases of NFV are discussed in [11]. Note that manageability, reliability, stability, and security are considered in [11] as the key performance parameters in both physical and in software based virtualized networks.

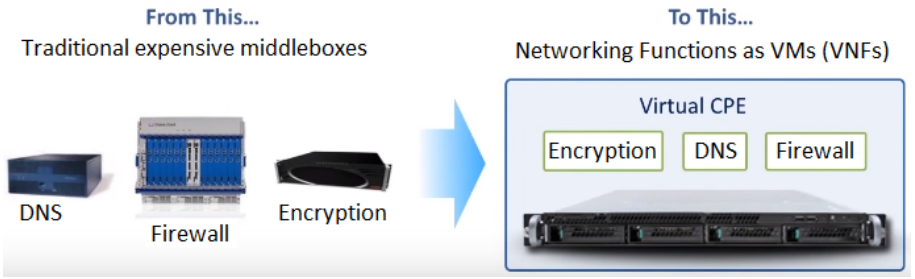


FIGURE 2.1: Network Function Virtualization

NFV provides a number of benefits to network operators, including:

- **Hardware Flexibility**: Because NFV uses regular Commercial-Off-The-Shelf (COTS) hardware, network operators have the freedom to choose and build the hardware in the most efficient way to suit their needs and requirements.

- **Faster Service Life Cycle**: New network services can now be deployed more quickly, in an on-demand and on-need basis, providing benefits for end users as well as the network providers.

- **Scalability and Elasticity**: New services and capacity-hungry applications keep network operators (especially cloud providers), on their toes to keep up with the fast-increasing demands of consumers.

- **Increased Revenue**: The combination of introducing new services faster and existing servers in a more dynamic fashion can jointly result in increased revenue.

- **Reduced Capital Expenditures (CAPEX)**: The use of industry-standard services, increased hardware utilization and adoption of open source software results in reduced capital expenditures.

- **Reduced Operational Expenditures (OPEX)**: Automation and hardware standardization can substantially slash operational expenditures.

- **Improved Customers' Satisfaction**: The combination of service agility and self-service can result in greater customer satisfaction.

- **Reduced Power Consumption and Complexity**: Efficiencies in space, power, and cooling. Communications Service Providers (CSPs) may have finite physical space, electrical power, and cooling capacity in a data center, so they will carefully select equipment to efficiently consume those finite and/or costly resources. NFV provides a better energy efficiency resulting from the consolidation of resources, as well as their more dynamic utilization.

2.2.3 NFV Architecture

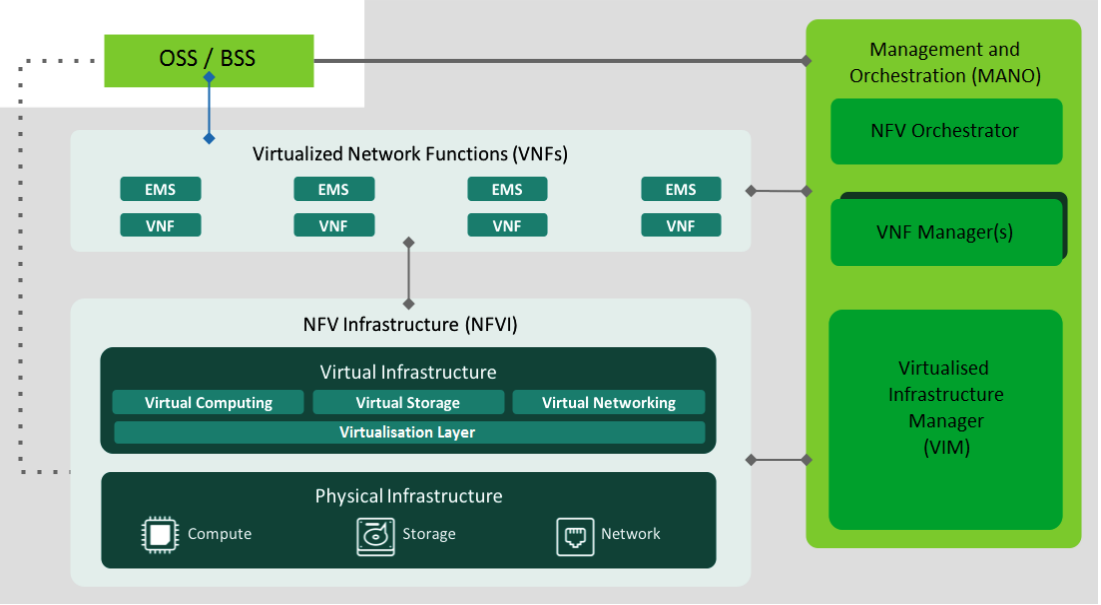


FIGURE 2.2: ETSI NFV reference architecture [2]

The main components of the NFV architectural framework are:

1. **NFV Infrastructure (NFVI)**: is a kind of cloud data center containing the totality of all hardware and software components that build up the NFV environment on which NFV services are deployed, managed and executed. NFVI includes:
   - *Physical Hardware*: this includes computing hardware (such as servers, RAM), storage hardware (such as disk storage, Network Attached Storage (NAS)), and network hardware (such as switches and routers).
   - *Virtualisation Layer*: abstracts the hardware resources and decouples the VNF software from the underlying hardware, thus ensuring a hardware independent lifecycle for the VNFs. We can use multiple open source and proprietary options for the virtualisation layer (such as KVM, QEMU, and VMware).
   - *Virtual Infrastructure*: this includes virtual compute (virtual machines or containers), virtual storage, and virtual networks (overlay networks).
2. **Virtualised Network Functions (VNFs)**: run on top of the NFVI and represent virtualized instances of different network functions. Each VNF has a corresponding

Element Management System (EMS) that provides management and control functionality for that VNF.

3. **NFV Management and Orchestration (MANO)**: NFV MANO does not act in isolation. It interacts with the Operational and Business Support Systems (OSS/BSS) components of the operator to manage the operational and business aspects of the network. MANO includes:

   - *Virtualized Infrastructure Manager (VIM)*: or cloud management software, e.g. OpenStack or Kubernetes. It is responsible for controlling and managing the computing, storage, and network resources, as well as their virtualization.
   - *VNF Manager(s)*: it is responsible for VNF life cycle management, including VNF instantiation, update, query, scaling, and termination.
   - *NFV Orchestrator*: it is in charge of the orchestration and management of NFV infrastructure and software resources, and realizing network services on NFVI. It utilizes resource allocation and placement algorithms to ensure optimal usage of both physical and software resources.

2.3 Software-Defined Networking (SDN)

Software-defined networking (SDN) [12], [13], is one of the most important architectures for the management of complex networks, which may require re-policing or re-configurations from time to time. SDN separates the control plane of traditional networking devices (e.g. switches, routers) from the data plane.

Before SDN, switches or routers were configured using routing protocols that did not allow fine-grained control. The control plane is known as an *SDN controller* and becomes directly programmable via an open interface (e.g., OpenFlow [14], [15] is the most popular SDN protocol/standard and has a set of design specifications). Thus, the underlying infrastructure (network routers/switches) just simply forwards packets by following the flow table rules set by the SDN controller.

SDN has the potential to dramatically simplify network management and to enable innovation and evolution through network programmability [16], [17], and [18].

The Open Networking Foundation (ONF) is taking the lead in SDN standardization, and has defined an SDN architecture model as depicted in Figure 2.3.

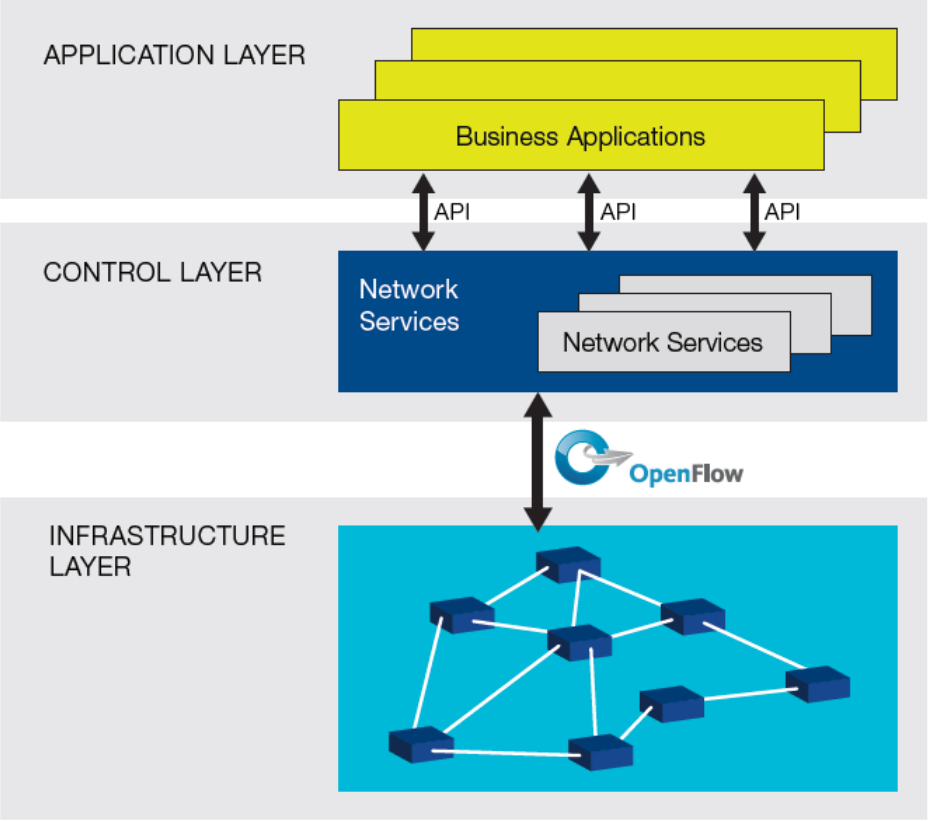


FIGURE 2.3: ONF/SDN architecture

The SDN architecture consists of three distinct layers that are accessible through open APIs:

- **Application Layer**: consists of the end-user business applications that consume the SDN communications services. The boundary between the Application Layer and the Control Layer is traversed by the northbound API.
- **Control Layer**: provides the logically centralized control functionality that supervises the network forwarding behavior through an open interface.
- **Infrastructure Layer**: consists of the network elements (NE) and devices that provide packet switching and forwarding.

2.4 Integration of NFV with other technologies

Over the past years, the integration of NFV with other technologies, such as SDN, Cloud computing, and 5G [19] has attracted significant attention from both the academic research community and industry.

NFV integration with SDN and Cloud computing is beneficial due to the complementary features and distinctive approaches followed by each technology toward providing solutions to today's and future networks [20], [21]. For instance, NFV provides **function abstraction** (i.e., virtualization of network functions) supported by ETSI [22], SDN provides **network abstraction** supported by Open Networking Foundation (ONF) [23], and Cloud computing provides **computation abstraction** (i.e., a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services)) supported by the Distributed Management Task Force (DMTF) [24]. Abstraction is one of the core features of cloud computing which allows abstraction of the physical implementation to hide the background (technical) details from users and developers. To summarize the relationships between NFV, SDN, and Cloud computing, we use Figure 2.4.

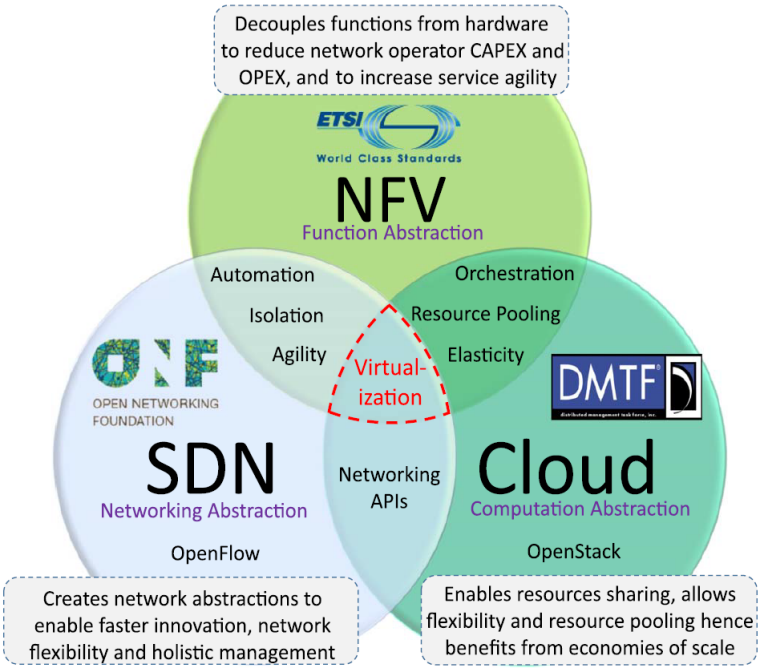


FIGURE 2.4: Relationships between NFV, SDN and cloud computing [3]

SDN, NFV, and Cloud computing technologies are complementary to each other but are independent and can be deployed alone or together. A combination of these technologies

together in a network architecture is more desirable [25]. In fact, the advantages that accrue from each of them are similar: agility, cost reduction, dynamism, automation, resource scaling, etc.

2.5 Resource Allocation in NFV

In NFV, a service is defined as a chain of software functions named Service Function Chain (SFC) [26], [27]. The process of allocating the resources of servers to the services, called Service Placement (or Resource Allocation), is one of the most challenging mission in NFV. The dynamic nature of the service arrivals and departures as well as the meeting Service Level Agreement (SLA) make the service placement problem even more challenging.

This section presents a survey of relevant research in the literature related to resources mapping in network environments. These studies are classified into two related topics. The first topic is the Virtual Network Embedding (VNE) problem [28] and the second one is the Virtual Network Function (VNF) placement and chaining problem [29].

Before entering into details, we add a note on the relation between the VNF placement and chaining problem and the traditional VNE problem. In fact, the task of placing functions is closely related to virtual network embedding [30], [31], [32] and may therefore be formulated as an optimization problem, with a particular objective.

Despite some similarities, VNE and VNF-FG embedding are distinct and have different characteristics [1].

- First, while in VNE we observe one-level mappings (virtual network requests $\rightarrow$ physical network), in NFV environments we have two-level mappings (SFC requests $\rightarrow$ VNF instances $\rightarrow$ physical network). VNE requests are modeled by simple undirected graphs while VNF chains are more complex and contain both the VNFs to place and the traffic flows to steer between the VNF-FG endpoints.

- Second, in NFV environments, a VNF can be shared by multiple demands, while in VNE, different virtual networks are typically independent (i.e., a flow of a virtual network does not traverse through the virtual nodes of another virtual network).
- Third, while the VNE problem considers only one type of physical device (i.e., routers), a much wider number of different network functions coexist in NFV environments.

2.5.1 Virtual Network Embedding (VNE)

2.5.1.1 Initial VNE strategies

The initial VNE problem can be divided into two sub-problems: *Virtual Node Mapping* consists in mapping each virtual node of the VN to one physical node that has enough available resources, and *Virtual Links Mapping* where virtual links connecting the virtual nodes have to be mapped to paths connecting the corresponding nodes in the substrate network under bandwidth resource constraints.

In VNE, the most referenced approach is the one suggested by Chowdhury et al. [33], which introduced a set of algorithms to correlate between node and edge embedding problems to solve the VNE. They embedded the virtual nodes onto physical network nodes based on their residual capacities, but they coordinated the edges embeddings using the multi-commodity flow algorithm to facilitate the embeddings of virtual edges onto physical network paths included the physical network hosting the virtual nodes. However, since nodes were embedded first then edges afterwards, longer paths could be used, causing additional costs and less accepted Virtual Networks Requests (VNRs).

Ogino et al. [34] propose a VNE algorithm based on a greedy approach, which prioritized the virtual edges assignment rather than the virtual nodes. They used the minimum-cost route algorithm to compute the optimum physical path for each virtual edge. The proposed approach focuses on selecting the optimum physical path that will minimize the increase in

the demanded amount of edges' bandwidth and nodes' capacity. However, their methodology could allow to select longer paths containing more physical network resources than requested.

Wang et al. [35] propose a framework to address the VNE problem, employing a branch and bound process to resolve the integrity constraints, which depends on the efficient estimation of the lower and upper bounds of a branch in a rooted tree that represents a subset of the solution set. Then they applied the column generation process to effectively obtain the lower bound for a branch pruning. As the branch and price framework maintains the lower and upper bound of the optimal solution, the authors claimed that their proposed framework can obtain near-optimal solutions with reduced computational time.

Soualah et al. [36] propose another VNE method uses Gomory-Hu tree transformation which provides a compact representation of the network topologies. The initial VNE problem was transformed using successive cuts of the graphs to map the virtual nodes on the tree representing the physical network. The authors claim that mapping virtual trees onto the physical tree fixes the mapping of the nodes and guarantees the mapping of virtual links in a splittable way, where one virtual link is dispatched to a set of physical paths.

It is important to note that VNE problem can be solved in either a *centralized* or in a *distributed* [37] way. Both approaches are fundamentally different [28], each having its own advantages and disadvantages. In a centralized approach, there will be one entity which is responsible for performing the embedding. The advantage of this approach lies in the fact that the mapping entity is at every step of the mapping aware of the overall situation of the network (i.e. it has global knowledge). Moreover, there may be scalability problems in large networks, where a single mapping entity may be overwhelmed by the number of Virtual Network Requests to handle. Contrary to centralized solutions, a distributed approach utilizes multiple entities for computing the embedding. The advantage of such an approach lies in its better scalability. However, one has to pay for this with synchronization overhead.

2.5.1.2 Dynamic/Adaptive resource management strategies

During their lifetime, virtual network resource requirements can evolve according to clients' fluctuating demands. Hence reserving a fixed amount of resources is inefficient to satisfy them. To cope with these problems, some dynamic resource management strategies where proposed. Migration has been also used in the VNE context. Several dynamic algorithms using reconfiguration optimize substrate resources for virtual networks.

Authors of [38] propose a path migration algorithm for reconfiguring and rerouting virtual links, unfortunately do not consider adaptation of the virtual nodes and thus find suboptimal solutions.

An iterative algorithm, called Virtual Network Reconfiguration (VNRe), is presented in [39] to ensure load balancing among substrate nodes and reduce the rejection rate caused by congested substrate links.

In order to supply dedicated virtual node to an end-user typified with a customized traffic, Fajjari et al. [40] suggest a resource provisioning algorithm to guarantee an efficient and flexible share among all the instantiated VNs upon the underlying physical network. For that, the authors proposed a new adaptive VNE algorithm called (Adaptive-VNE), which adopts a backtracking algorithm in order to minimize the VNE cost.

In [41], the authors propose an adaptive optimization algorithm which selects only critical VN requests for reconfigurations. Despite this selection of most critical VN requests, computational cost and service disruptions remain important at large scale.

In [42], the authors propose an adaptive fault-tolerant VN embedding algorithm, which relies on a multi-agent based framework to cope with failures and severe performance degradation.

Authors of [43] propose a dynamic adaptive virtual network resource allocation strategy to deal with the complexity and the inefficiency of resource allocation. The main idea behind the proposal is take advantage of unused bandwidth with respect to the occupancy rate of embedded virtual links. The unused bandwidth will be reassigned to incoming virtual network requests.

Inoue et al. [44] present an adaptive VNE method based on the biological "Yuragi" principle for software-defined infrastructure (SDI). The proposed method works with little information for large and complicated SDI frameworks. The term Yuragi is a Japanese word whose English translation means a small perturbation to the system. Yuragi is a mechanism of adaptability of organisms and is often expressed as an attractor selection model.

Shahriar et al. [45] address the connectivity-aware Virtual Network Embedding problem, which consists in embedding a virtual network (VN) on a substrate network while ensuring VN connectivity against multiple substrate link failures. The proposed method provides a weaker form of survivability incurring less resource overhead than traditional VN survivability models.

The more important body of work on scaling and migrating services in VNE is not directly applicable to the VNF-FG placement problem.

2.5.2 Virtual Network Function (VNF) placement and chaining

Several surveys are now available on NFV, see, e.g., [3], [10], [25], and [1] where the various NFV challenges are discussed.

2.5.2.1 Initial VNF-FG placement strategies

VNF-FG placement and chaining is formulated as an Integer Linear Programming (ILP) in [31], [32], [46], [47], and [48] to find exact solutions for hosting the VNFs of the requested service graph. These works propose partial and exact mathematical formulations for the SFC provisioning problem. Researchers consider different objective functions.

Hybrid mathematical formulations: Martini et al. [49] and Riggio et al. [50] solve the placement and routing for each request independently. Gupta et al. [51] propose a heuristic based on an ILP. They only consider the k-shortest paths for every request in the network and a simplified node capacity constraint, for which only one function per node can be deployed. There has been a recent attempt with a Column Generation model, by [52], and

[53]. Unfortunately, as the resulting column generation model was not well scaling, its ILP solution is not exact.

Exact mathematical formulations: Luizelli et al. [48] provide an exact model minimizing the number of instances of functions in the network. However, they consider only a couple of tens of requests. Savi et al. [54] propose an exact formulation in which the number of VNF nodes is minimized. Their model takes into account additional costs inherent to multi-core environment. However, they only provide results on a small network. Bari et al. [55] consider the operational expenditure (OpEx) for a daily traffic scenario as their objective function. Mohammadkhan et al. [56] propose an exact model along with heuristics aiming at minimizing the maximum usage of CPU and links. The scope of the experiments is limited to the case in which the number of cores per service is limited to one.

However, ILP is only suitable for the situation that all variables are integers. Thus, for some specific situations, the Mixed ILP (MILP) is used instead. For example, Addis et al. [57] proposed a VNF-P model, in which both the NFV goal (minimizing the number of CPUs used by instantiating VNFs) and the Traffic Engineering (TE) goal (minimizing the risk of a sudden bottleneck on network links) are considered. However, in order to jointly achieve the two goals, some non-integer variables have to be introduced. Thus, the VNF-P model proposed was actually a MILP model which described the relationship between VNF placement and traditional routing.

As the addressed problem is NP-Hard [55], the exact solutions do not scale with size and require an excessive amount of time to find the optimal solutions.

Heuristic algorithms: are typically and consequently proposed to scale better with problem size by solving the problem iteratively and to find good solutions much faster. Unfortunately, this is accomplished at the expense of quality of the solutions (proximity with the optimal solutions).

A baseline Greedy algorithm [58], using a bipartite graph construction and matching techniques, for VNF-FG placement and chaining, solves the problem in two steps: by first mapping the VNFs on physical hosts and second by steering the inter-VNF traffic across the hosts. This leads obviously to suboptimal solutions that are often far from optimal.

A heuristic algorithm in [59] uses "Simulated Annealing" (SA) to find solutions faster unfortunately by simplifying the overall problem by considering only one type of VNF and addressing rather small chains.

Authors in [60] focus on the placement of VNFs chains in the NFV context. They propose a matrix-based optimization and a multi-stage graph method that are cost efficient and improve scalability by finding solutions in polynomial times.

Note that some work study the SFC provisioning problem using game theory [61] or approximation algorithms [62], [63], and [64]. Some context-aware placement problems are also studied. Authors in [65] study VNF placement algorithms in virtual 5G network, with the goal of minimizing the path length and optimize the sessions' mobility.

2.5.2.2 Elastic VNF-FG placement strategies

Elastic orchestration of virtual network functions (VNF) is a key factor to achieve NFV goals. However, most existing VNF orchestration researches are limited to offline policy, ignoring the dynamic characteristics of the workload. In fact, the resource that a VNF has may scale due to dynamic traffic (for example, a DPI need less computing resource when the traffic decreases). The QoS demand of a VNF may change also due to changes of service requests (for example, when an established service request asks for low latency, the re-allocation of VNFs is required) [66].

Therefore, in order to overcome those challenges, we should rethink the VNF-FG embedding problem and propose new solutions. Some authors do address virtual network function scaling and migration to ensure dynamic VNF chain placement for rising demand and traffic load.

Authors of [67] propose a consolidation algorithm called Simple Lazy Facility Location (SLFL) that optimizes the placement of the VNF instances in response to on-demand workload. SLFL chooses the VNF instances to be migrated on the basis of the instantaneous reconfiguration but does not assess the impact (induced benefits or penalties) of these decisions on future instants.

Bandwidth guaranteed VNF placement and scaling in Datacenter is considered in [68]. Leveraging the tree-like topology of Datacenter networks, this work proposes an on-line heuristic algorithm that achieves a near-optimal allocation. Note that this approach does not take into account migration cost.

Authors in [69] propose an architecture for the 5G core network using SDN and NFV technologies. Moreover, based on this architecture, they design a framework for determining the Network Functions (NFs) placement and optimizing the NFV Infrastructure (NFVI) resources and network response time during emergency situations. This framework is based on a two-stage method. In the first stage, the placement of the SDN Controllers and VNFs is determined considering not only latency requirements and load levels but also users' mobility and network functions type. Meanwhile, in the second stage, the NFVI resources are dynamically optimized according to variations on network conditions.

Toosi et al. [70] define a unified framework for building elastic service chains. They propose a dynamic auto-scaling algorithm called ElasticSFC to minimize the cost while meeting the end-to-end latency of the service chain. The experimental results show that the proposed algorithm can reduce the cost of SFC deployment and SLA violation significantly.

Gu et al. [71] propose an Elastic Virtual Network Function Orchestration (EVNFO) policy based on workload prediction. They adapt the online learning algorithm for predicting the flows rate of service function chains (SFC), which can help to obtain the VNF scaling decision. They further design the online instance provisioning strategy (OIPS) to accomplish the deployment of VNF instances according to the decision. The simulation proves that EVNFO can provide good performance with dynamic resource provision.

Luo et al. [72] propose a deep learning-based framework for scaling of the geo-distributed VNF chains, exploring inherent pattern of traffic variation and good deployment strategies over time. They novelly combine a recurrent neural network as the traffic model for predicting upcoming flow rates and a deep reinforcement learning (DRL) agent for making chain placement decisions. They adopt the experience replay technique based on the actor-critic DRL algorithm to optimize the learning results.

Pei et al. [73] study the SFC embedding problem (SFC-EP) with dynamic VNF placement in a geo-distributed cloud system. They formulate this problem as a Binary Integer Programming (BIP) model aiming to embed SFC requests with the minimum embedding cost.

Rankothge et al. [74] present an Iterated Local Search (ILS) based framework for automation of resource reallocation that supports the three scaling models. They use this framework to run experiments and compare the different scaling approaches, specifically how the optimization is affected by the scaling approach and the optimization objectives.

Ma et al. [75] present a load balancing methodology for the management of horizontal scaling of NF chains that does not require changes to the NF code. They also develop a prototype reference implementations to illustrate the feasibility of the proposed solution and conduct extensive simulations to assess the performance.

Gouareb et al. [76] aim to minimize latency, considering horizontal scaling of VMs by placing VNFs across physical nodes. Routing and placement of VNFs are linked and have a significant impact on latency especially for delay-critical services. They aim at optimizing the distribution and utilization of available resources and meet user requirements with the objective of minimizing the overall accumulated latency on both the edge clouds (where VNFs are running) and the flow routing paths.

Chen et al. [77] formulate the SFC migration problem as a minimization problem with the objective of total network operation cost under constraints of users' quality of service. They design a deep Q-network based algorithm to solve single SFC migration problem, which can adjust migration strategy online without knowing future information. A novel multi-agent cooperative framework is also proposed to address the challenge of considering multiple SFC migration based on single SFC migration.

Tang et al. [78] propose a real-time VNF migration algorithm based on the deep belief network to predict future resource requirements. But training the practicable neural network is extremely eager for large amounts of training data.

In a recent work, Fei et al. [79] study the proactive VNF provisioning problem for NFV providers, considering the fluctuations of traffic traveling service chains. They formulate

the problem that minimizes the cost incurred by inaccurate prediction and VNF deployment. They first employ an online learning method which aims to minimize the prediction error, to predict the upcoming traffic flows. Then, when launching new instances based on the prediction outcomes, an adaptive scaling strategy is adopted for saving resources and decreasing deployment cost. They also propose two algorithms that are called by the complete online algorithm, for new instance assignment and service chain routing.

2.6 Conclusion

This chapter describes the state of the art that relates directly to the NFV resource allocation problem and that is directly connected to our contributions. In recent years, the problem has received some interest from the community and has produced several valuable contributions. However, the outcome is still incomplete. There are yet important aspects that should be investigated to efficiently manage and allocate the use of the resources in NFV-based network architectures. The next chapters describe the thesis contributions in terms of dynamic resource management strategies, mathematical models, and both exact and heuristic optimization algorithms.

Chapter 3

Virtual Network Function Scaling

3.1 Introduction

In NFV based environment, one of the most important challenges for providers is to efficiently allocate hosting resources to dynamic virtualized network services demands while increasing revenue. Elastic mechanisms and scaling algorithms are essential to improve adaptation and deployment of Virtualised Network Functions (VNFs) in NFV infrastructures to support increasing traffic load and customer demands.

As introduced in [80] and [67], Virtual Network Function scaling is triggered by new client requirements and/or rising traffic load due, for example, to an increase in the user plane traffic or the need of allocating more resources to a VNF to avoid service interruption.

To enhance initial VNF placement with dynamic adaptation, three scaling mechanisms are defined in [67] and [81]:

- *Horizontal scaling (scale out/in)*: Add/remove virtualized resources (e.g., VNF Components (VNFCs)).
- *Vertical scaling (scale up/down)*: Reconfigure the capacity/size of existing virtualized resources.

- *VNF Migration*: move VNF components from one hardware platform onto a better platform while still satisfying the service continuity requirement.

To address this NP-Hard elasticity problem [82], we propose an exact algorithm and a Greedy heuristic. An Integer Linear Programming (ILP) formulation with a search space reduction is adopted to improve scalability. A Greedy algorithm searching for a scaling solution in the neighborhood of an initial placement is also presented. Comparisons between the two approaches show that an adequately tuned and devised ILP can outperform Greedy solutions in terms of proportion of successful scalings.

In [82], authors propose an Integer Linear Programming (ILP) model to solve the network function migration problem. In the rest of this chapter, we denote this competitor algorithm by (ILP_C). It proposes a migration cost model and a heuristic algorithm to decrease the migration cost. Note that these algorithms do not consider scaling and elasticity to optimize resource utilization.

The system model is described in Section 3.2. The proposals are introduced in Section 3.3. Section 3.4 reports the performance evaluation results and finally, we summarize the results with some future research directions in Section 3.5.

3.2 Problem Formulation

This section models the VNF and the VNF-FG scaling problem and derives an ILP model that ensures placement and scaling for increasing demands with minimal cost and service interruptions.

3.2.1 Substrate and virtual network models

The physical network (commonly known as substrate or physical infrastructure. The terms are used interchangeably), as defined by the ETSI NFV Infrastructure (NFV-I) [83], is modeled as an undirected weighted graph, denoted by $G_p = (N_p, E_p)$ where E_p is the set of

physical links and N_p is the set of physical nodes. Each substrate node, $k \in N_p$, is characterized by its i) available processing power (i.e., CPU) denoted by CPU_k, and ii) type T_k: switch, server or Physical Network Function (PNF) [84], where PNFs are the traditional physical middleboxes offering network functions. A PNF is a dedicated hardware that implements a network function. Each physical link (i.e., $e \in E_p$) is characterized by its available bandwidth BW_e.

A client request (i.e., requested service function chain, also called VNF-FG) is modeled as a directed graph $G_v = (N_v, E_v)$ where N_v is the set of virtual nodes and E_v is the set virtual links in the graph. Each virtual node, $i \in N_v$, is characterized by its i) required processing power cpu_i and ii) its type t_i: VNF or switch (i.e., ingress or egress). Each virtual link $(ij) \in E_v$ is described by its required bandwidth bw_{ij}. Note that the VNFs can be hosted only in server or PNFs having the same type. The ingress and egress nodes can be hosted only in switches.

3.2.2 Scaling Problem

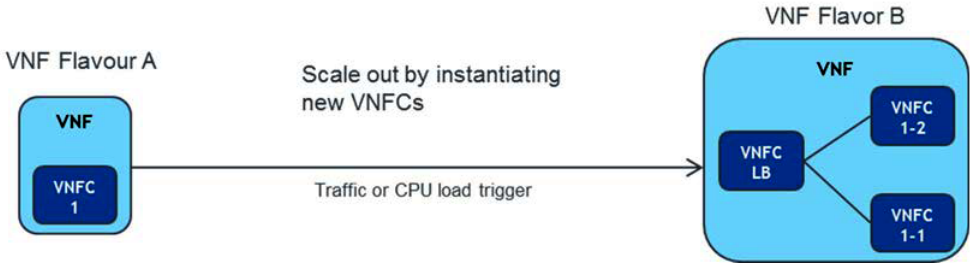


FIGURE 3.1: VNF Autoscaling by instantiating new VNFCs

Before the expiration of a tenant (i.e., a group of users who share a common access with specific privileges) or user VNF-FG lifetime, the resource requirements of the VNFs in the forwarding graph can increase with rising demand. This will entail the spawning of new VNF instances, the allocation of additional hosting resources and possible the migration of the VNFs to other more capable physical hosts. For example, a firewall may need to install more filtering and control rules or handle more applications and users. Figures 3.1 and 3.2 depict examples some of these scaling actions to respond to rising requirements and demand. ETSI [80] defines the types of scalings as follows:

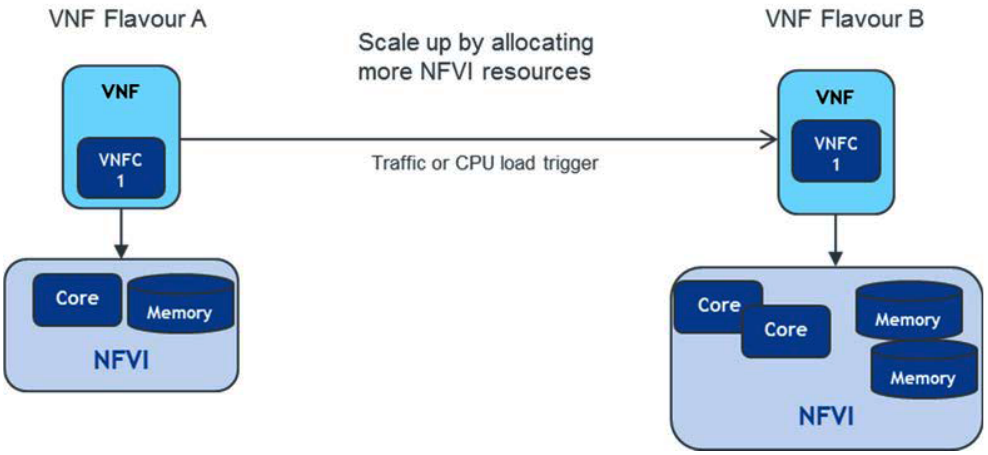


FIGURE 3.2: VNF Autoscaling by allocating additional NFVI resources

1. When a VNF is **scaled out**, new VNF components (VNFCs) are instantiated and added to the VNF. Under such circumstances, the VNF may need a mechanism to distribute the load or traffic among the VNFCs (newly instantiated and existing VNFCs). This distribution can be accomplished through the use of load balancers (see Figure 3.1).

2. When a VNF is **scaled in**, one or more VNFCs of a VNF are terminated.

3. When a VNF is **scaled up**, it is assigned additional NFVI resources such as compute cores, memory, storage, or network resources (see Figure 3.2).

4. When a VNF is **scaled down**, NFVI resources that have been previously allocated to the VNF are de-allocated.

From the point of view of physical resource consumption, scaling out and scaling up can be treated the same way (i.e., increasing VNF scale is accomplished by scaling out or scaling up [80]) despite technical deployment differences. Both actions require additional resources and result in increased physical resources consumption. Scaling out and up are taken into account in our proposed model and optimization algorithms.

3.3 Proposals

We propose an Integer Linear Programming solution and a Greedy algorithm to address the VNF scaling problem. We assume that the scaling concerns the i^{th} VNF in the VNF-FG. Hence, i is fixed (i.e., not variable) for each scaling request. A single VNF scaling request is triggered for each and every generated request.

3.3.1 ILP Formulation

We formulate the ILP model for virtual network function scaling. Table 3.1 summarizes the parameters and the used variables. The model includes integrity constraints associated to the objective function used to achieve optimal scaling.

Node re-mapping constraint: Each scaled VNF i is re-mapped to exactly one physical candidate node k. A VNF (i.e., its components (VNFCs)) can not be split into (distributed across) many physical nodes. This is expressed by:

$$\sum_{k \in \mathcal{C}} x_{ik} = 1 \tag{3.3.1}$$

where:

$$x_{ik} = \begin{cases} 1, & \text{if the VNF } i \text{ is re-mapped to physical node } k; \\ 0, & \text{otherwise.} \end{cases} \tag{3.3.2}$$

Capacity constraint: This constraint ensures that the residual capacity in a physical node k satisfies the required capacity by VNF i. This leads to the following inequality:

$$cpu_i \times x_{ik} \leq CPU_k, \quad \forall k \in \mathcal{C} \tag{3.3.3}$$

Link re-mapping constraint: Each virtual link (ij) will be re-mapped to exactly one physical path $P_{k,m(j)}$ where k is a physical candidate for the i^{th} VNF and $m(j)$ is the initial mapping location of the j^{th} VNF. Note that i and j are neighbors in the VNF-FG request. We use the $\mathcal{K}$-shortest path algorithm to calculate $\mathcal{P}$: the set of $\mathcal{K}$-shortest paths candidates

TABLE 3.1: Main notations

Notation	**Description**
CPU_k	Residual capacity in a physical node k
$m(j)$	Initial mapping location of VNF j
$P_{k,m(j)}$	Physical path interconnecting physical nodes k and $m(j)$
$BW_{P_{k,m(j)}}$	Residual bandwidth in a physical path $P_{k,m(j)}$
$\mathcal{C}$	Set of physical nodes candidates
$\mathcal{P}$	Set of physical paths candidates
min_{CPU}	The minimum capacity in $\mathcal{C}$
BW_e	Residual bandwidth in one physical link e
δ_{ep}	A boolean coefficient determining whether physical link $e \in E_p$ belongs to path $P_{k,m(j)} \in \mathcal{P}$: $\delta_{ep} = 1 \Leftrightarrow e \in P_{k,m(j)}$
$E_N(i)$	Set of virtual links (ij) where j is a virtual neighbor of i
cpu_i	Required capacity by virtual node i
bw_{ij}	Required bandwidth between virtual nodes i and j
x_{ik}	A binary variable indicating whether VNF i is re-mapped to physical node k
$y_{ij,P_{k,m(j)}}$	A binary variable indicating whether virtual link (ij) is re-mapped to physical path $P_{k,m(j)}$
Δ_{t_r}	Remaining time of request before departure
δ_t	Required time to migrate one capacity unit
Rev_u	Revenue unit
Pen_u	Penalty unit

$P_{k,m(j)}$. We extend the Dijkstra algorithm based on residual bandwidth to find more than one candidate path $P_{k,m(j)}$ interconnecting physical nodes k and $m(j)$. Then, we supply the set $\mathcal{P}$ to the ILP.

$$\sum_{k \in \mathcal{C}} y_{ij,P_{k,m(j)}} = 1, \quad \forall (ij) \in E_N(i) \tag{3.3.4}$$

where:

$$y_{ij,P_{k,m(j)}} = \begin{cases} 1, & \text{if the virtual link } (ij) \text{ is re-mapped to physical path } P_{k,m(j)}; \\ 0, & \text{otherwise.} \end{cases} \tag{3.3.5}$$

Bandwidth constraint: Obviously the residual bandwidth in a physical path $P_{k,m(j)}$ has to satisfy the bandwidth requirement of virtual link (ij). We should not violate the remaining bandwidth of each physical link $e \in E_p$ on the physical path $P_{k,m(j)}$. This condition can be formally expressed as:

$$\sum_{(ij)\in E_N(i)} bw_{ij} \times y_{ij,P_{k,m(j)}} \times \delta_{ep} \leq BW_e, \quad \forall e \in P_{k,m(j)},\ \forall P_{k,m(j)} \in \mathcal{P} \tag{3.3.6}$$

Node and link re-mapping constraint: When a VNF i is re-mapped to a physical candidate node k, each virtual link (ij), starting from VNF i, has to be re-mapped to a physical path $P_{k,m(j)}$ having k as one of its endpoint or extremity. The other endpoint is $m(j)$ (i.e., where j is a neighbor of i in VNF-FG). Let us call $\mathcal{P}_{k,m(j)}$ the set of candidate paths between k and $m(j)$. Then, we define a binary variable for each of the candidate path $P_{k,m(j)} \in \mathcal{P}_{k,m(j)}$. This constraint can be written as:

$$x_{ik} = \sum_{P_{k,m(j)}\in\mathcal{P}_{k,m(j)}} y_{ij,P_{k,m(j)}}, \quad \forall (ij) \in E_N(i), \forall k \in \mathcal{C} \tag{3.3.7}$$

Objective function: We are interested in minimizing the service interruption time caused by scaling and favor for this reason new physical hosts with the most available resources. A way to reduce the complexity of the ILP and to scale much better with network size is to explore not all the candidate nodes space. That is cutting the exploration to a subset of the candidates, especially candidate nodes that are less loaded. This should also maximize the revenue of the providers that can maintain services and tidy up their infrastructures to

accept more users.

$$
\begin{aligned}
&maximize\ Z \\
&\text{Where:} \\
&Z = (\Delta_{tr} \times cpu_i \times Rev_u) \times \textstyle\sum_{k \in \mathcal{C}} \frac{CPU_k}{min_{CPU}} \times x_{ik} \\
&\quad - (\delta_t \times cpu_i \times Pen_u) \times \textstyle\sum_{k \in \mathcal{C} \setminus \{m(i)\}} x_{ik}
\end{aligned}
\tag{3.3.8}
$$

The addressed VNF scaling problem is summarized by lumping the objective function and the constraints:

maximize Z

subject to:

$$\textstyle\sum_{k \in \mathcal{C}} x_{ik} = 1$$

$$cpu_i \times x_{ik} \leq CPU_k, \ \ \forall k \in \mathcal{C}$$

$$\textstyle\sum_{k \in \mathcal{C}} y_{ij,P_{k,m(j)}} = 1, \ \ \forall (ij) \in E_N(i)$$

$$\textstyle\sum_{(ij) \in E_N(i)} bw_{ij} \times y_{ij,P_{k,m(j)}} \times \delta_{ep} \leq BW_e,$$

$$\forall e \in P_{k,m(j)}, \forall P_{k,m(j)} \in \mathcal{P}$$

$$x_{ik} = \textstyle\sum_{P_{k,m(j)} \in \mathcal{P}_{k,m(j)}} y_{ij,P_{k,m(j)}}, \ \ \forall (ij) \in E_N(i), \forall k \in \mathcal{C}$$

PROBLEM 1: VNF scaling optimization summary

3.3.2 Heuristic Algorithm

We also propose a Greedy algorithm for VNF scaling and re-mapping to compare with our ILP and gain some insight on achievable performance, benefits and strengths of the exact modeling approach. The proposed Greedy algorithm operates using the following 5 steps:

1. Check initial mapping $m(i)$ capacity: if the remaining resource in node $m(i)$ can satisfy the new required capacity by VNF i, scaling occurs in the same physical node $m(i)$ (i.e., without migration). Else, use migration to scale;

2. Find a set of neighboring physical candidate nodes $\mathcal{C}$ related to m(i). These candidate nodes are also sorted by available resources;

3. Check candidate node re-mapping: if node k capacity and type constraints are met, move to links capacities checking;

4. Compute the shortest path between candidate k and all physical nodes $m(j)$ (where j is a virtual neighbor of i) using Dijkstra's algorithm based on available bandwidth. In fact, this corresponds to computing the best paths that maximize the minimum bandwidth along their route between k and $m(j)$;

5. Check link re-mapping: if links capacities are respected, confirm candidate k as re-mapping solution.

Algorithm 1: Scaling Greedy pseudo-code

```
1  Inputs: G_p, VNF i, cpu_i
2  Output: VNF i scaled
3  m(i) ← GetInitialMapping(VNF i)
4  if CPU_{m(i)} ≥ cpu_i then
5      return Mapping(VNF i, cpu_i, m(i))
6  else
7      C ← ComputeNeighborCandidate(m(i), cpu_i)
8      stack ← Sort(C)
9      while stack ≠ ∅ and solution = False do
10         k ← Pop(stack)
11         if CPU_k ≥ cpu_i and
12         ComputeLinkMapping(k, m(j)) = True then
13             solution ← True
14 return Mapping(VNF i, cpu_i, k)
```

Our Greedy algorithm has a complexity of $O(|\mathcal{C}| \times |N_p|^2 \times |E_N(i)|)$. The algorithm requires in Step 9 (Algorithm 1) $|\mathcal{C}|$ iterations in the worst case and a complexity $|N_p|^2$ (in the worst case) in Step 12 (Algorithm 1) since the Dijkstra's algorithm is used to compute the shortest path needed to map a virtual link (ij) crossing the scaled VNF i. The Fibonacci heaps can improve the complexity of the Dijkstra algorithm to be $O(|E_p| + |N_p| \times \log |N_p|)$. The Dijkstra algorithm is called $|E_N(i)|$ times and this leads to the $O(|\mathcal{C}| \times |N_p|^2 \times |E_N(i)|)$ overall complexity in the worst case.

3.4 Performance Evaluation

The performance of our proposed ILP based is assessed through extensive simulations. Simulator settings and performance evaluation metrics used for evaluation and comparison purposes are presented. The ILP is compared to the Greedy heuristic. The evaluation focuses on the gains reconfiguration (just to tidy up the network) and scaling (during rising demands) provide to the stakeholders by analyzing the proportion of rejected (failed) adaptation attempts.

3.4.1 Simulation Environment

Our simulations are based on a realistic topology as well as extensive simulations for which it is possible to evaluate in depth the scalability of the algorithms using larger infrastructures and request sizes. A 2.50 GHz Quad Core server with 6 GBytes of available RAM is used for the performance evaluation and comparison of the proposed algorithms.

For the realistic topology and the extensive simulations, the VNF-FG requests are generated using a Poisson process with an average arrival rate of 5 requests per 100 time units. The lifetime of each request follows an exponential distribution with a mean of 1000 time units.

The Germany50 network topology [85] is used for the first assessment (with 50 nodes). This topology is defined by the German National Research and Education Network (DFN). The capacity of physical nodes and links are generated randomly in the $[50, 100]$ interval. The size of the VNF-FG requests is set to 5 nodes. The GT-ITM [86] tool is used to generate the requested VNF-FG topologies. The initial VNF-FG computing and bandwidth requirements are set to 10. Scaling in CPU is fixed to 20 with one scaling request at a time per VNF-FG.

To assess the scalability of our proposed algorithms, we generate using the GT-ITM tool a network topology with 100 nodes and a connectivity of 0.2 (or 20%). The physical resource capacities (i.e., CPU and bandwidth) are also drawn randomly in the $[50, 100]$ interval. The VNF-FG request sizes vary between 3 and 15 nodes. The required CPU for each VNF in the VNF-FG is set to 10 units. The required bandwidth between two VNFs, to ensure

communication between them, is set also to 10 units. The connectivity between nodes in the VNF-FG is set to 0.3.

The ILP algorithm is evaluated for three values of the migration penalty Pen_u from 10 to 1000 units in order to tune the penalty according to the needs and provider priority as well as to measure its effect on performance. The migration penalty (Pen_u in the second term of the objective function in Equation 3.3.8) is used to control the re-mapping process and to especially trade-off "in node scaling" with "migration to other hosts". The migration penalties in the performance evaluation correspond to the reported ILP10, ILP100 and ILP1000 results. The performance of the ILP for these penalty values (10, 100 and 1000) are also compared to the Greedy algorithm, to assess the effectiveness and usability of this migration. For the realistic topology and the simulation, 1000 VNF-FG requests are generated and a single VNF scaling request is triggered for each and every generated request. Since, we are focussing on scaling and adaptation of already embedded VNF-FGs, we used the heuristic approach of [58] to generate the initial VNF-FG mapping and initialize the assessment runs.

3.4.2 Performance Metrics

The metrics used for the performance evaluation are described in this section that reports also the results obtained using both the realistic topology and the extensive simulations for the extended performance assessment.

- **Successful scalings** represents the number of VNF scaling requests that have been accomplished. Indeed, due to physical resources limitation the algorithm may reject some scaling requests. From the provider point of view, this number should be maximized in order to improve the global revenue.
- **Scaling ratio** is the ratio of successful scalings defined by the number of successful scalings to the accepted VNF-FG requests. The number of accepted VNF-FG requests depends on the initial provisioning and on how the scaling algorithm deals with scaling requests. Since we use as initial mapping the same basis for all algorithms, the scaling ratio reflects their relative efficiency and enables their comparison.

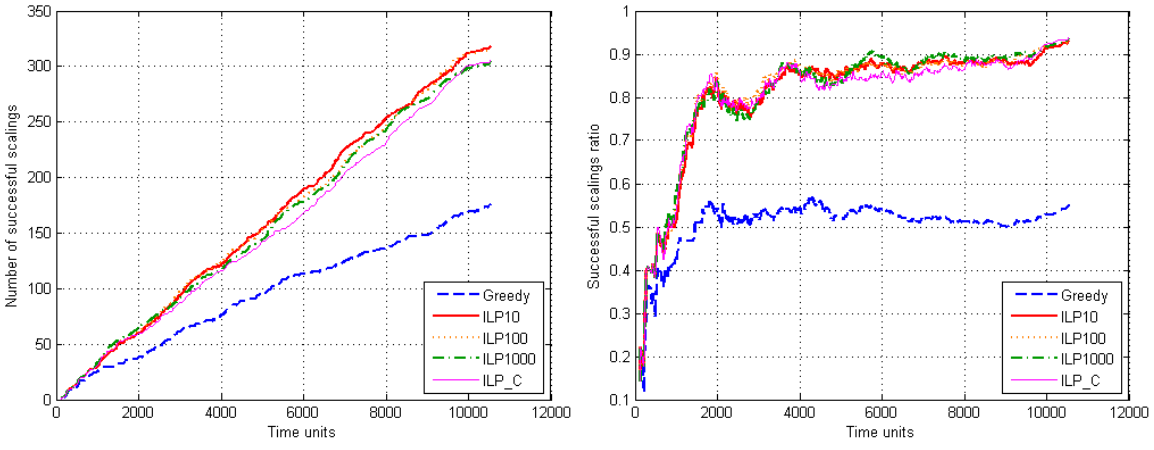
- **Number of migrated VNFs** is the number of the scaled VNFs through migration. Migration involves a temporary service interruption until the concerned VNF is activated in the new physical host.

- **The ratio of migrated VNFs** is the ratio of the number of migrated VNFs to the number of successful scalings. This metric measures the proportion of adaptations accomplished through migration that providers prefer to minimize to avoid or limit the service interruptions due to migrations. Minimizing this measure reduces disruptions to applications.

- **Execution time** is a decisive measure in order to assess the scalability of the algorithms. Service providers prefer efficient and rapid algorithms in order to quickly serve clients.

3.4.3 Simulation Results

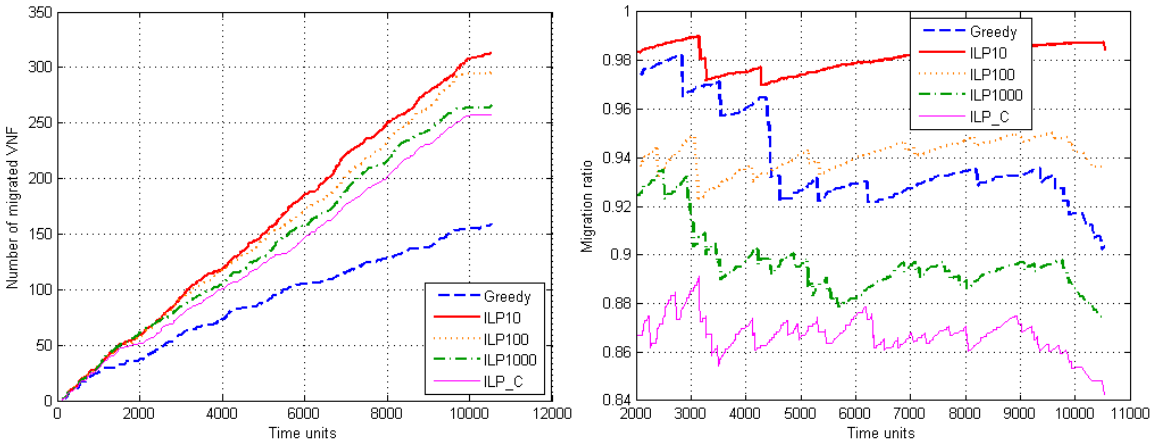
3.4.3.1 Evaluation on a realistic topology



(a) Successful scalings (b) Scalings ratio

FIGURE 3.3: Germany50 topology results: Scalings

We use the Germany50 topology as an NFV-I (i.e., physical network) to conduct the first performance assessment. Figure 3.3(a) shows that the ILP algorithm, using the three migration penalty values, outperforms the Greedy solution. This is accomplished despite the fact that the ILP does not consider all possible candidates but just a subset for scalability



(a) VNF migration (b) Migration ratio

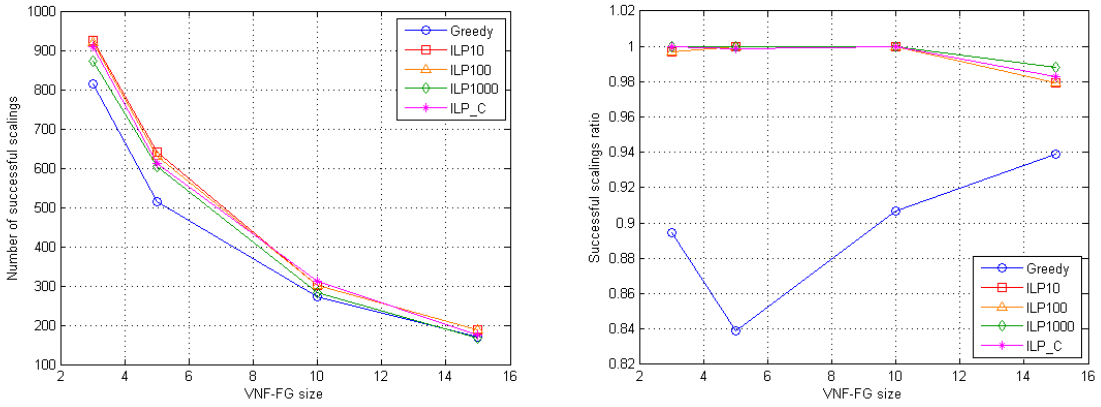
FIGURE 3.4: Germany50 topology results: Migration

reasons. The ILP satisfies in addition more scaling demands as confirmed by figure 3.3(b) where the scaling ratio exceeds 90% for the ILP variants and ILP_C. This proportion is only about 55% for the Greedy strategy.

Figure 3.4(a) depicts the number of the migrated VNFs. The ILP with a migration penalty of 10 performs more migrations than ILP1000 for which a higher migration penalty forces the scaling to occur within the nodes in priority. At the first glance, one could conclude that the Greedy algorithm and ILP_C use less migrations than the ILP1000 but figure 3.4(b) shows that the migration ratio of the Greedy strategy (about 90%) is higher than ILP1000 (roughly 87%) and ILP_C ($84,3\%$). If we analyze only the three ILP configurations, we observe as expected that ILP10 migrates almost all VNFs (more than 98%) to find more hosting resources while ILP100 migrates about 94% of the requests.

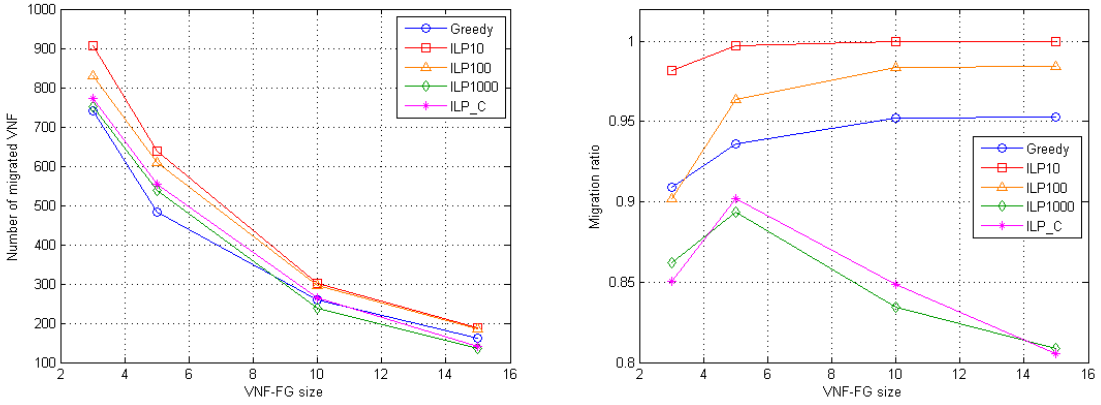
3.4.3.2 Large-scale evaluation

To analyze the behavior of the algorithms and their scalability with problem size, we use larger NFV-Is (with 100 nodes) and initial VNF-FG request sizes with a number of VNFs in the $[3,15]$ interval. Figure 3.5(a) reports the number of successful scalings. The ILP algorithm outperforms the Greedy heuristic by satisfying more scaling demands. Figure 3.5 (b) confirms this with scaling ratios exceeding 97% for the ILP variants and ILP_C. The Greedy strategy achieves ratios only between 84% and 94%.



(a) Successful scalings (b) Scalings ratio

FIGURE 3.5: Large scale scenarios: Scalings



(a) VNF migration (b) Migration ratio

FIGURE 3.6: Large scale scenarios: Migration

Figure 3.6(a) illustrates the number of the migrated VNFs. ILP10 performs more migration than the ILP1000. ILP10 and ILP100 perform more migrations than the Greedy and ILP_C. ILP10 roughly migrates the totality (exactly 100% if VNF-FG sizes exceed 10) of the scaled VNFs compared to ILP100 that migrates $98,4\%$. Figure 3.6(b) shows that the migration ratio of the Greedy strategy (varies between 90% and 95%) is higher than ILP1000 and ILP_C (between 80% and 90%). Clearly the ILP can be tuned to outperform other algorithms as well as tradeoff migrations with in node scaling to fulfill the provider preferences and business interests and policies.

3.4.3.3 Importance of reconfiguration

In this subsection we focus on the gain that the reconfiguration process may provide. The provider can perform re-mapping of some VNFs re-mapping to tidy-up their physical networks and prevent bottlenecks and degradations in the NFV-I and in addition can host more client requests. Figure 3.7 depicts the rejection ratio, based on the initial embedding, of the ILP algorithm, the Greedy heuristic and the baseline solution (i.e., dealing with only the initial mapping). In this scenario there is no extra CPU demand but only the concerned VNF will be reconfigured. The baseline algorithm (i.e., without reconfiguration) rejects more requests (about 14%) compared to the Greedy heuristic and ILP_C (9%) and the ILP model (7%). This result can be explained by the dynamics in the requests arrivals and departures that are exploited more efficiently and especially when departures occur and resources are released. The reconfiguration opportunistically tidies up the network and make room for new requests that would be otherwise rejected because of suboptimal use of the infrastructure.

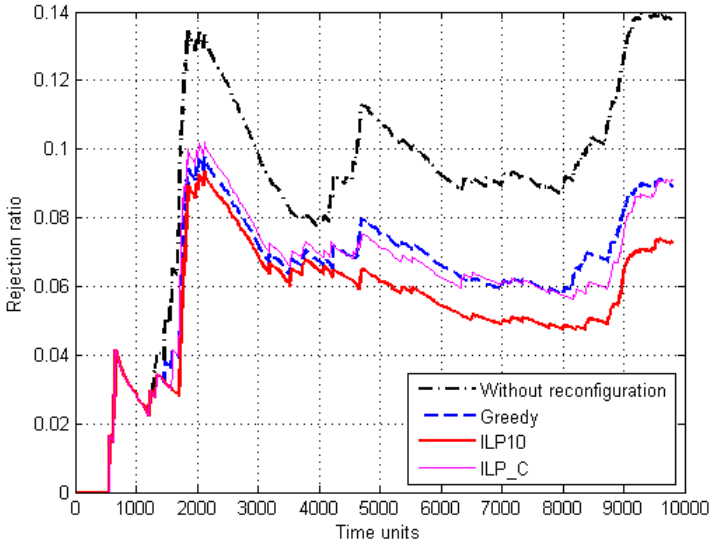


FIGURE 3.7: Rejection Ratio

3.4.3.4 Average time resolution (Execution time)

Table 3.2 reports the execution time performance of the algorithms to gain insight on their scalability and complexity with problem size. In order, to evaluate this metric, the substrate

TABLE 3.2: Execution time (ms)

	$\|\mathcal{C}\|$=5	$\|\mathcal{C}\|$=10	$\|\mathcal{C}\|$=15	$\|\mathcal{C}\|$=20
$Greedy$	$11,65$	$12,07$	$14,64$	$15,43$
ILP	$23,04$	$38,21$	$43,74$	$44,62$
ILP_C	$26,81$	$41,87$	$45,62$	$46,12$

graph size is fixed at 100 nodes. We generate 1000 VNF-FG requests and vary the set of potential candidates size from 5 to 20. These results indicate that the Greedy algorithm has significantly better execution time when compared to our ILP algorithm and ILP_C execution times. However, the faster execution times of the Greedy algorithms are accomplished at the expense of the number of accepted scaling requests. Indeed, the ILP achieves 918 scalings and ILP_C 909 while the Greedy accepts only 815. The extra time taken by the ILP is due to the ability of the ILP to find more solutions and hence accept more scaling requests.

3.5 Conclusion

This chapter addresses the problem of virtual network function reconfiguration with the main objective of reducing the rejection of VNF scaling requests when faced with rising demand and traffic load. We proposed an ILP model and a new Greedy reconfiguration algorithm for the purpose and show that the solutions can efficiently scale virtualized network functions and forwarding graphs with good execution time and scaling requests acceptation performance over distributed and dynamically varying cloud environments.

Chapter 4

Dynamic VNF Forwarding Graph Extension Algorithms

4.1 Introduction

Network Function Virtualization (NFV) coupled with Software-Defined Networking (SDN) is expected to impact the business to business to consumer relationships, current business models and value chains by enabling requests from multiple tenants for virtual infrastructures to deploy and provide their services to other providers and consumers.

Providers that deploy this variety of services and network functions in both physical and virtual infrastructures on behalf of tenants will have to treat via multiple network functions traffic originating, transiting and terminating in their infrastructures. The service provider has dynamic SFC establishment, update, and extension requirements and needs that the infrastructure providers have to meet. The latter will consequently need not only to remove, replace, insert, and add VNF instances, but also to extend already instantiated service chains without disturbing the previously deployed chains.

In this foreseen context, network providers (or physical infrastructure providers and more generally NFV architecture designers and actors) are actually faced with dynamic and variable demands of virtualized network infrastructures from tenants. These actors are indeed

likely to deploy their services, hosted in virtualized infrastructures, gradually as their business grows. These players not only need to control, classify, and steer user and application traffic flows in their dedicated slices but also want to extend their already acquired and operational slices with additional service graphs (adding new forwarding paths, new service chains, new services and virtualized network functions). These extensions of already hosted network function graphs have to be achieved without disrupting initially deployed and active service instances. The extensions must be seamless to applications and services that do not tolerate interruptions, migration or disruptions in quality of service (QoS) and experience.

In order to meet these requirements, this chapter proposes algorithms to extend already deployed network services and function graphs to respond to new demands while taking into account the constraint of minimizing the effect on the original service graphs.

According to [67] and [80], Virtual Network Function scaling and extension are triggered by new client requirements and the rising of network load over time. This is especially true for services with periodic resource demands (e.g., cloud applications such as an online shopping store during holiday seasons, etc.), and managing VNF workload changes [87] (e.g., during peak hours for cellular telecom operators).

In this chapter, we address the more generic and general problem of extending already deployed service function chains (or VNF-FGs) and aim at providing algorithms that can achieve various types of extensions. The latter may be a request to insert a new VNF in an existing graph, a new demand to extend a Network Forwarding Path (NFP) by adding VNFs to an existing chain, and more generally extend an already deployed VNF-FG with a complex service graph.

We refer to the graph extension as a VNF-FG extension to emphasize that we also address networking level extension of already deployed forwarding graphs in addition to the fact that we inherently answer the needs for service function chaining extensions. The extensions, as mentioned, can be triggered by new requests or by the VNF-FG life cycle management that may decide to extend the initial graph to adapt to network and traffic load changes such as adding load balancers, spawning new VNFs to absorb increasing load, etc.

We first propose an Integer Linear Programming (ILP) model as an exact formulation of the VNF-FG extension problem. The ILP can find optimal solutions for reasonable problem sizes and thus can serve as a reference for the performance evaluation and as a way to assess the quality of the solutions for heuristic algorithms at least for small graph sizes. The heuristics should be efficient, find good solutions, and scale with problem size much better than the ILP. To reduce complexity while finding good solutions, we resort to a Steiner Tree based algorithm and an Eigendecomposition algorithm using network adjacency matrices of the requested and hosting graphs as a basis for optimal embedding. The ILP and the heuristic algorithms ensure that the specified connecting points between the previously described graphs and the extensions are respected as specified in the client requests and that there are no disruptions (or minimal disruption) to the initially deployed VNF-FG. We compare the performance of these algorithms in terms of rejection rate of new requests, quality of the solutions and service disruptions as a function of infrastructure size, network connectivity, and system load.

Section 4.2 of this chapter presents the related work. The problem formulation is described in Section 4.3. The proposals are introduced in Section 4.4. Section 4.5 reports the performance evaluation results. Section 4.6 summarizes the main findings.

4.2 Related Work

Since we are addressing the extension of service function chains already deployed in hosting infrastructures, the related work review focusses on the virtual network topology change problem with emphasis on the dynamic expansion and adaptation of Virtualized Network Function-Forwarding Graphs (VNF-FGs) whenever relevant. Indeed, previous work addressed partially or simply did not address extension of already deployed services but rather initial placement, adaptation to changes of an existing graph and/or the hosting infrastructure and reaction to faults. We consequently limit the description to work as close as possible to ours while being aware that extensions of SFCs or VNF-FGs have not been directly and explicitly and fully addressed in the past.

In [88], authors present JASPER, a fully automated approach to jointly optimize scaling, placement, and routing decisions for complex network services. Two algorithms are developed for adaptation of existing services to changes in the demand, a Mixed Integer Linear Program (MILP) and a custom constructive heuristic.

Ayoubi et al. [89] introduced RELIEF, an availability-aware embedding and reconfiguration framework for elastic services in the cloud. The framework comprises two main modules. The JENA sub-system that performs virtual network (VN) embedding to provide just-enough availability guarantees based on the availability of the physical servers hosting the virtual one. The second module, ARES, is a reliable reconfiguration bloc that adapts the embedding of hosted services as they scale (by migrating the VN or adding backup nodes). This work does not address either, explicitly, graph extension requests from already served clients or increasing workloads which is our instead our central concern or objective.

Liu et al. [90] consider the readjustment of VNF chaining in dynamic situations and conditions by trying to optimize jointly the deployment of new SFCs and the readjustment of existing service chains in order to satisfy variable user requirements. This is closer to some extent to our work and stated goal. However, unlike our approach, they use pre-calculated paths in the network. We are focussing on the extension of already deployed chains and service graphs while keeping the initially provided and operating tenant graph unmodified, except for edges involved in the extension. The authors of [90] use first an ILP formulation to solve the problem exactly then reduce time complexity using a Column Generation (CG) heuristic algorithm that approximates the performance of the ILP. The main idea behind the CG based algorithm is to decompose the original problem into a master problem and a sub-problem to solve them iteratively in order to obtain a near-optimal solution.

In [91], authors present a comprehensive analysis of a resource allocation algorithm for VNFs based on genetic algorithms (GA) for the initial placement of VNFs and the scaling of VNFs to support traffic changes. For the scaling of existing policies proposed in [91], the algorithm starts with the current state and searches for the re-assignment of resources for the set of VNFs that need scaling. Therefore, some VNFs change their initial locations and the algorithm tries to minimize the number of server and link configuration changes (to minimize disruptions to existing traffic).

Li et al. [92] implemented a real-time resource provisioning system for NFV called NFV-RT, which integrates timing analysis with several techniques, such as service chain consolidation, and Integer Linear Programming with rounding. Also, the scaling is achieved by duplicating full instances of VNF chains, and by reshuffling chains and migrating traffic.

The proposals defined in this chapter differs from previous work and that of [91] and [92]. They actually address the problem of **Resource and Function Scaling** defined in [93] as the adaptation of the assigned network resources to functions or flows, or adaptation of the number of deployed instances of a specific VNF (by scale out, scale in, or migration). We are instead concerned by another type of flexibility, specifically **Topology Adaptation**, defined also in [93], by extending the graph structure of already active virtual networks that require an extension with additional nodes and links. Our addressed graph extension can contain new network and service functions and is far more complex than simply duplicating the service chain by a graph replication technique as proposed in [92]).

Note that all our algorithms, in this chapter, maintain the initial mapping location of already instantiated service chains (original VNF-FG) to avoid disturbing previously deployed and running services (i.e., without migrations or interruptions), to preserve previous deployments and ensure network stability.

In summary, our work differs significantly from the existing VNFs scaling and adaptation proposals since none really addresses the problem of extending an already deployed tenant VNF-FGs following a new request for seamless (non disrupting) evolutions of these dedicated service chains (i.e., maintaining acceptable service performance).

4.3 Problem Formulation

The problem of extending an already deployed and operating tenant SFC or VNF-FG corresponds to the placement of new requested VNFs and flow paths in the hosting infrastructure. This has to be accomplished while respecting all previous deployments and the specified connectivity between the initial graph and its extension. This is a placement problem with a set of very specific constraints. The placement of virtual functions and path extensions

must cover a subset of previously deployed graph nodes and links, those involved in interconnection of the old and new graph and the ingress and egress switches (or gateways) involved in both graphs. Hence, the objective is to embed a new graph in the infrastructure with a specific node cover requirement with respect to a previous graph deployment. The problem is intuitively identified as some form of node cover problem and a spanning tree problem. The solution must span or cover some specific nodes in the original SFCs and VNF-FGs and find an optimal mapping of the new request nodes and paths. To address this problem we use an ILP formulation that serves as a reference for performance and quality comparisons with two heuristic algorithms, a Steiner Tree solution and an Eigendecomposition approach. Both the Steiner and Eigendecomposition can take into account the old and new graphs inter-connectivity requirements. The eigendecomposition relies on adjacency matrices that reflect the networking topologies of the graphs and for this reason is a natural candidate to address VNF-FG extensions in shared infrastructures. The Steiner Tree approach adopted here corresponds to a generalization of the non-negative shortest path and the spanning tree problems and in fact to the well known Steiner tree problem in graphs.

4.3.1 Substrate Graph or NFV Infrastructure

In order to derive an ILP for the SFC extension problem, we start from a mathematical graph representation of the extension request, the previously deployed service graph (SFC or VNF-FG) and the hosting infrastructure. In addition to this, a description of the interconnection between the previously deployed graph and the extension graph is required, but we embed instead this in the constraints and conditions that must be respected by all algorithms when placing the extension in the rest of the hosting infrastructure.

The physical network (known as substrate and physical infrastructure and used interchangeably), as defined by the ETSI NFV Infrastructure (NFV-I) in [83], is modeled as an undirected weighted graph $G_p = (N_p, E_p)$ where E_p is the set of physical links and N_p is the set of physical nodes.

Each substrate node, $k \in N_p$, is characterized by its i) available processing capacity (i.e., CPU) CPU_k, and ii) type T_k: switch, server or Physical Network Function (PNF) [84]. The PNFs are the traditional physical middleboxes implementing network functions.

Each physical link (i.e., $e \in E_p$) is characterized by its available bandwidth BW_e.

An example of such an infrastructure, known as the NFV-I, including two PNFs and two switches (an ingress switch and an egress switch acting as input and output gateways for the VNF-FG or SFC related Network Forwarding Paths) and some interconnected servers is presented in Figure 4.1.

4.3.2 VNF Forwarding Graph

Similarly a client request (i.e., a requested Service Function Chain (SFC)) is modeled as a directed graph $G_v = (N_v, E_v)$ where N_v is the set of virtual nodes and E_v is the set of virtual links in the requested graph.

Each virtual node, $i \in N_v$, is characterized by its i) required processing power cpu_i and ii) its type t_i: VNF or switch (i.e., ingress or egress). Each virtual link $e_{ij} \in E_v$ is described by its required bandwidth $bw_{e_{ij}}$.

As specified by the SFC IETF working groups [94], we associate a VNF-type to each VNF to represent the network service or function type (e.g., firewall, DPI, NAT, load balancer, SSL, etc.) and their respective requirements. Note that the VNFs can be hosted only by servers or PNFs having the same type. The ingress and egress nodes can be hosted only by switches. Figure 4.1 depicts two Network Forwarding Paths (NFP). Each Forwarding Path NFP describes the ordered VNF sequence the traffic must pass through.

From the VNF forwarding graph G_v, we derive an intermediate request graph called the Network Connectivity Topology graph NCT_v. The $NCT_v = (N_v, E_v)$ is a weighted undirected graph having exactly the same set of nodes and edges than G_v. The key attribute of an NCT node $i \in N_v$ is its requested processing capacity cpu_i. The weight (or requested bandwidth $bw_{e_{ij}}$) of an NCT virtual link $e_{ij} \in E_v$ is the sum of the requested bandwidths of all the VNF flows passing through it. This representation means that there is no path

splitting over links connecting two consecutive VNFs and that the same physical link will be used to host the aggregate demand between any two VNFs for a given VNF-FG.

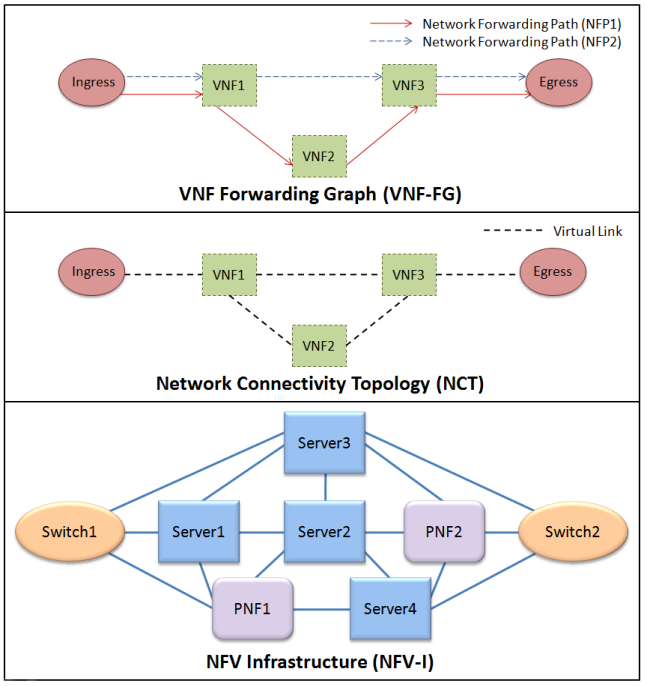


FIGURE 4.1: The VNF-FG and NFV-I topologies

In the ETSI-MANO document [95], the requested graphs are defined as follows:

- **Network connectivity topology (NCT)**: is a graph that specifies the VNF nodes that compose the global network service and the connection between these nodes through virtual links (VL). Each VL is connected to a VNF through a connection point (CP) that represents the VNF interface. Hence, an NCT defines a logical topology among VNFs in a network. Note that this logical topology represented by an NCT may change as a function of user requirements, business policies, and/or network context.

- **VNF Forwarding Graph (VNF-FG)**: is a graph established on top of the NCT. The VNF-FG is composed of network forwarding paths (NFP) that are ordered lists of connection points (CPs) forming a chain of VNFs.

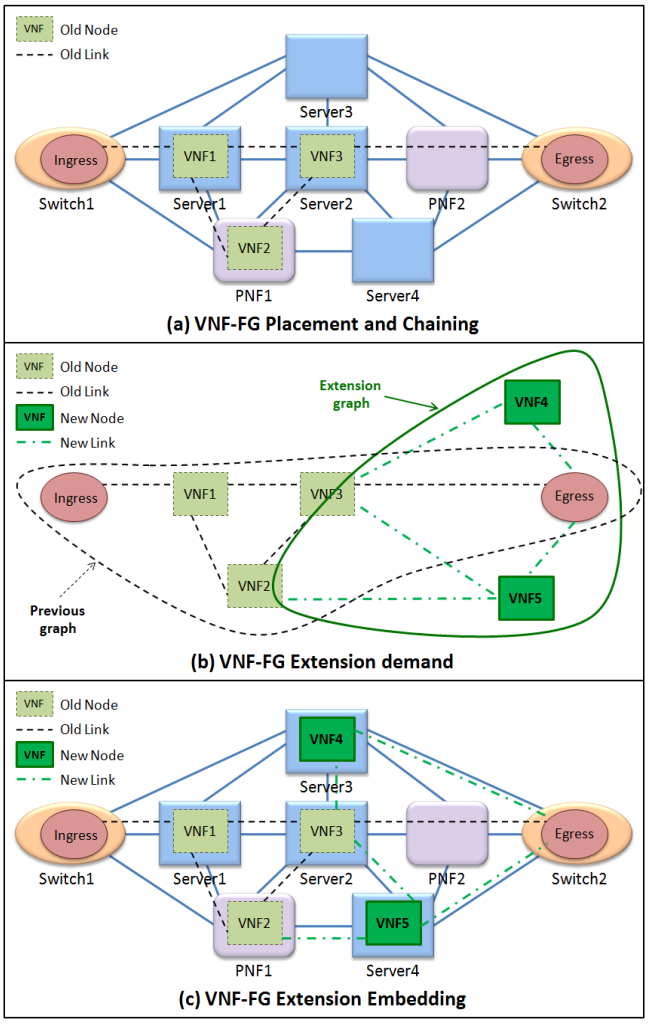


FIGURE 4.2: VNF Forwarding Graph Extension

4.3.3 VNF Forwarding Graph Extension

Figure 4.2 describes the type of VNF-FG extensions considered in our work. Figure 4.2(a) depicts the previously deployed forwarding graph (or slice) before receiving an extension request for two new VNFs (VNF 4 and VNF 5). Figure 4.2(b) shows their required connectivity with the already deployed graph. Figure 4.2(c) depicts a placement solution for the extension with VNF 4 and VNF 5 hosted respectively in Server 3 and Server 4. The newly requested interconnection links and forwarding paths are also highlighted.

4.4 Proposals

To solve the problem of VNF-FG extension placement and chaining, we propose an Integer Linear Programming model, a Steiner Tree based algorithm and an eigendecomposition approach that uses as a basis the adjacency matrices of the request graph and the hosting infrastructure graph.

4.4.1 ILP Model

We now formulate the ILP model for VNF-FG extension. Table 4.1 summarizes the parameters and the used variables. The model includes integrity constraints associated to the objective function used to achieve optimal extension.

Old node mapping constraint: This constraint maintains the initial mapping location of old VNF nodes.

$$x_{im_i} = 1, \quad \forall i \in N_v^{old} \tag{4.4.1}$$

Old link mapping constraint: maintains the initial mapping location of old virtual links.

$$y_{e_{ij},P_{m_i,m_j}} = 1, \quad \forall e_{ij} \in E_v^{old} \tag{4.4.2}$$

New node mapping constraint: ensures that new VNFs, VNF i, is mapped to exactly one physical candidate node k and that the VNF components (VNFCs composing the VNF) cannot be split into (distributed across) many physical nodes. This is expressed by:

$$\sum_{k \in \mathcal{C}} x_{ik} = 1, \quad \forall i \in N_v^{new} \tag{4.4.3}$$

where:

$$x_{ik} = \begin{cases} 1, & \text{if the VNF } i \text{ is hosted on the substrate node } k; \\ 0, & \text{otherwise.} \end{cases} \tag{4.4.4}$$

TABLE 4.1: Table of Notations

Notation	**Description**
CPU_k	Residual capacity in a physical node k
m_i, m_j	Initial mapping location of old VNFs i and j
P_{k_i,k_j}	Physical path interconnecting physical nodes k_i and k_j
$\mathcal{C}$	Set of physical nodes candidates
$\mathcal{P}$	Set of physical paths candidates
min_{CPU}	The minimum capacity in $\mathcal{C}$
BW_e	Residual bandwidth in one physical link e
δ_{ep}	A boolean parameter indicating if the physical link $e \in E_p$ belongs to path $P_{k_i,k_j} \in \mathcal{P}$: $\delta_{ep} = 1 \Leftrightarrow e \in P_{k_i,k_j}$
N_v^{new} E_v^{new}	Set of new virtual nodes Set of new virtual links
N_v^{old} E_v^{old}	Set of old virtual nodes Set of old virtual links
cpu_i	Required capacity by virtual node i
$bw_{e_{ij}}$	Required bandwidth between virtual nodes i and j
x_{ik}	A binary variable indicating whether VNF i is mapped to physical node k
$y_{e_{ij},P_{k_i,k_j}}$	A binary variable indicating whether virtual link e_{ij} is mapped to physical path P_{k_i,k_j}

Capacity constraint: makes sure that the residual capacity in a physical node k satisfies the required capacity by VNF i. This leads to the following inequality:

$$\sum_{i \in N_v^{new}} cpu_i \times x_{ik} \leq CPU_k, \quad \forall k \in \mathcal{C} \tag{4.4.5}$$

New link mapping constraint: *The case where both extremity of virtual link e_{ij} are new nodes:* Each new virtual link e_{ij} is mapped to exactly one physical path P_{k_i,k_j} where k_i is a physical candidate for the i^{th} VNF and k_j is a physical candidate for the j^{th} VNF. Note that

i and j are neighbors in the VNF-FG request.

$$\sum_{k_i \in \mathcal{C}} \sum_{k_j \in \mathcal{C}} y_{e_{ij}, P_{k_i,k_j}} = 1, \ \ \forall e_{ij} \in E_v^{new}, \ \forall i, j \in N_v^{new} \tag{4.4.6}$$

where:

$$y_{e_{ij}, P_{k_i,k_j}} = \begin{cases} 1, & \text{if the virtual link } e_{ij} \text{ is mapped to physical path } P_{k_i,k_j}; \\ 0, & \text{otherwise.} \end{cases} \tag{4.4.7}$$

The case where source extremity i of virtual link e_{ij} is an old node and destination extremity j is a new node: Each new virtual link e_{ij}, starting from old VNF i, is mapped to exactly one physical path P_{m_i,k_j} where m_i is the initial mapping of old VNF i and k_j is a physical candidate for the new VNF j.

$$\sum_{k_j \in \mathcal{C}} y_{e_{ij}, P_{m_i,k_j}} = 1, \ \ \forall e_{ij} \in E_v^{new}, \ \forall i \in N_v^{old}, \ \forall j \in N_v^{new} \tag{4.4.8}$$

The case where destination extremity j of virtual link e_{ij} is an old node and source extremity i is a new node: Each new virtual link e_{ij}, having an old VNF j as destination endpoint, is mapped to exactly one physical path P_{k_i,m_j} where k_i is a physical candidate for the new VNF i and m_j is the initial mapping of the old VNF j.

$$\sum_{k_i \in \mathcal{C}} y_{e_{ij}, P_{k_i,m_j}} = 1, \ \ \forall e_{ij} \in E_v^{new}, \ \forall i \in N_v^{new}, \ \forall j \in N_v^{old} \tag{4.4.9}$$

Bandwidth constraint: Obviously the residual bandwidth in a physical path P_{k_i,k_j} has to satisfy the bandwidth requirement of virtual link e_{ij}. The allocation (or mapping) must not violate the remaining bandwidth of each physical link $e \in E_p$ on the physical path P_{k_i,k_j}. Note that we compute $\mathcal{K}$-shortest paths candidates P_{k_i,k_j} using Dijkstra's algorithm based on residual bandwidth of links. This condition can be formally expressed as:

$$\begin{aligned} \sum_{e_{ij} \in E_v^{new}} bw_{e_{ij}} \times y_{e_{ij}, P_{k_i,k_j}} \times \delta_{ep} \leq BW_e, \\ \forall e \in P_{k_i,k_j}, \ \forall P_{k_i,k_j} \in \mathcal{P} \end{aligned} \tag{4.4.10}$$

Node and link mapping constraint (source type): When a VNF i is mapped to a physical candidate node k_i, each virtual link e_{ij}, starting from VNF i, has to be mapped to a physical path P_{k_i,k_j} with k_i as one of its endpoint or extremity (source). The other endpoint is k_j.

$$\sum_{k_j \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{ik_i}, \quad \forall e_{ij} \in E_v^{new}, \ \forall k_i \in \mathcal{C} \tag{4.4.11}$$

Node and link mapping constraint (destination type): Similarly to the previous constraint (source type), if a VNF j is mapped to a physical candidate node k_j, then each virtual link e_{ij} has to be mapped to a physical path having k_j as one of its endpoints (destination).

$$\sum_{k_i \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{jk_j}, \quad \forall e_{ij} \in E_v^{new}, \ \forall k_j \in \mathcal{C} \tag{4.4.12}$$

Node separation constraint: This constraint corresponds to situations where VNFs have to be separated and mapped onto distinct nodes for security reasons for example or simply due to tenant requirements or application constraints. In this case VNF i and j are mapped into nodes k_i and k_j with $k_i \neq k_j$.

$$x_{ik} + x_{jk} \leq 1, \quad \forall i,j \in N_v^{new}, \ \forall k \in \mathcal{C} \tag{4.4.13}$$

Objective function: The objective function directly depends on the infrastructure providers' interests while taking into account as much as possible the users' or tenants' expectations. The objective function has to be adapted to the provider policies and criteria such as a consolidation, an energy consumption minimization or a cost reduction and revenue maximization. There are consequently multiple possible objective functions, but as far as this chapter is concerned, we will arbitrarily, and for the sake of selecting a given case, assume that the providers aim at naturally balancing the load on their hosting infrastructures. If the providers aim at reducing energy consumption, we would modify the objective function to achieve consolidation with a maximum packing of the requests and hence minimize energy consumption and cost, by for instance selecting the nodes whose residual capacity is sufficient and closest in size to the requests. A simple and efficient way to achieve load balancing is to favor (select in priority) the infrastructure nodes and links that are least loaded to host

the requests, resulting in gradual and distributed filling or loading of the infrastructure. The proposed objective function consists in maximizing Z of Equation 4.4.14.

$$Z = \sum_{i \in N_v^{new}} cpu_i \times (\sum_{k \in \mathcal{C}} \frac{CPU_k}{min_{CPU}} \times x_{ik}) \tag{4.4.14}$$

The variable min_{CPU} is the smallest residual compute power available in all the candidate nodes in set C. This is a normalization factor, used to align all terms in Equation 4.4.14 (dimensionless quantities). This factor is constant at each run of the ILP and it emphasizes in addition nodes with more free resources. The addressed VNF-FG extension problem is finally summarized by lumping the objective function and the constraints:

maximize $\{Z\}$

subject to:

$x_{im_i} = 1, \ \forall i \in N_v^{old}$

$y_{e_{ij},P_{m_i,m_j}} = 1, \ \forall e_{ij} \in E_v^{old}$

$\sum_{k \in \mathcal{C}} x_{ik} = 1, \ \forall i \in N_v^{new}$

$\sum_{i \in N_v^{new}} cpu_i \times x_{ik} \leq CPU_k, \ \forall k \in \mathcal{C}$

$\sum_{k_i \in \mathcal{C}} \sum_{k_j \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = 1, \ \forall e_{ij} \in E_v^{new}, \forall i,j \in N_v^{new}$

$\sum_{k_j \in \mathcal{C}} y_{e_{ij},P_{m_i,k_j}} = 1, \ \forall e_{ij} \in E_v^{new}, \forall i \in N_v^{old}, \forall j \in N_v^{new}$

$\sum_{k_i \in \mathcal{C}} y_{e_{ij},P_{k_i,m_j}} = 1, \ \forall e_{ij} \in E_v^{new}, \forall i \in N_v^{new}, \forall j \in N_v^{old}$

$\sum_{e_{ij} \in E_v^{new}} bw_{e_{ij}} \times y_{e_{ij},P_{k_i,k_j}} \times \delta_{ep} \leq BW_e, \ \forall e \in P_{k_i,k_j}, \forall P_{k_i,k_j} \in \mathcal{P}$

$\sum_{k_j \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{ik_i}, \ \forall e_{ij} \in E_v^{new}, \forall k_i \in \mathcal{C}$

$\sum_{k_i \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{jk_j}, \ \forall e_{ij} \in E_v^{new}, \forall k_j \in \mathcal{C}$

$x_{ik} + x_{jk} \leq 1, \ \forall i,j \in N_v^{new}, \forall k \in \mathcal{C}$

PROBLEM 2: VNF-FG extension optimization summary

4.4.1.1 ILP with Reduced Number of Candidate hosts (RNC_ILP)

The ILP is known not to scale polynomially with problem size, as the problem is NP-Hard [96], [3]. A way to reduce the complexity and to scale much better with size, is to explore not all the "candidate nodes and links" space and accept a suboptimal solution. That is cutting the exploration to a subset of the candidates, especially candidate nodes and links that are less loaded to favor load balancing. In order to avoid local optima, we use a random selection process to diversify choices and move out of local optima (traps) in the problem convex hull. This suboptimal selection process nevertheless takes into account the criteria and constraints expressed in Equation 4.4.14. The performance of this ILP inspired heuristic, using a reduced set of candidates, is compared with the ILP to assess the proximity in achieved solution quality or distance with the optimal solution from the full exploration ILP.

The pseudo-code of the candidate subset selection is depicted in Algorithm 2 where we check the initial mapping of the old VNF-FG graph G_v^{old} and avoid the locations used to host the old VNFs "$UsedNode()$" to favor load balancing. We randomly select eligible hosts "$randomNode$" from the "$queue$" to avoid being trapped in a local optimum. The "$queue$" is sorted by available resources.

Algorithm 2: Candidate selection module pseudo-code

```
Inputs: G_p, G_v^new, the initial mapping of G_v^old, maxCand
Output: A set of candidate hosts
queue ← ∅
C ← ∅
foreach physical node k ∈ N_p do
    if UsedNode(k) = False then
        queue ← queue ∪ k
stack ← Sort(queue)
repeat
    C ← C ∪ Pop(stack) ∪ randomNode
until size(C) = maxCand;
return C
```

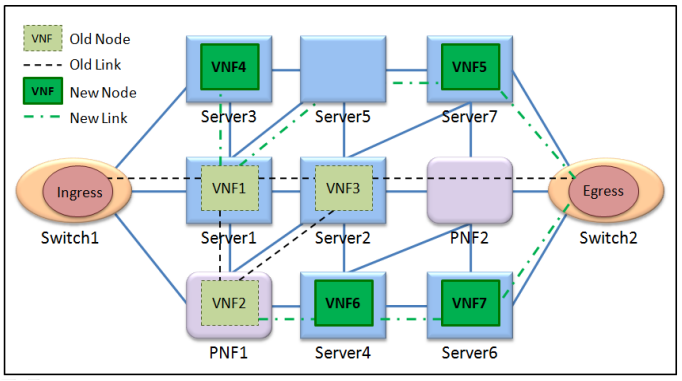


FIGURE 4.3: Example of the improved STVE algorithm

4.4.2 STVE: Steiner Tree based algorithm For VNF-FG Extension

An alternative to the ILP is to view the extension problem as a form of spanning tree problem where the solution for the extension has to cover a number of anchored nodes more precisely those of the previously deployed graph that have to be connected to the new graph (or extension). An equivalent way to address this optimization problem is to view the previous nodes as root nodes in a tree and the objective would be to find the branches and leaves out of these root nodes.

Figure 4.3 depicts a VNF-FG extension request to respond to rising requirements and demands. Server1, hosting VNF1, can be defined as a root tree-node and the objective would be to find a placement solution for the new VNFs, VNF4 and VNF5 part of the extension request, in leaf physical nodes. Similarly, PNF1 is seen as a root tree-node, since it already hosts VNF2 that must be connected to the new VNFs, VNF6 and VNF7. These new VNFs have to find a mapping solution in the ensuing tree leaves.

We nevertheless prefer to view the problem as a Steiner Tree optimization problem that is well known and fully characterized in combinatorial optimization. Indeed, determining a minimum cost connected network that spans a given set of vertices or nodes (known as terminals) is the well identified and studied spanning tree problem in the literature often solved using the cost of a minimum spanning tree (MST)[97] or solved by computing a Steiner Tree of minimal cost [98], [99], [100] and [101].

If a Steiner tree problem in graphs contains exactly two terminals, it reduces to finding the shortest path. If, on the other hand, all vertices are terminals, the Steiner tree problem in

graphs is equivalent to the minimum spanning tree. However, while both the shortest path and the minimum spanning tree problem are solvable in polynomial time, the Steiner tree problem in graphs is NP-complete [102].

The Steiner tree problem in graphs can be seen as a generalization of the shortest path problem (when the problem contains exactly two terminals) and the minimum spanning tree problem (when all vertices are terminals). In the addressed VNF-FG extension problem, the nodes of the previously deployed graph, that the extension graph nodes are connected to, correspond to Steiner Terminals. The goal is to find Steiner Vertices acting as intermediate nodes and hosts of the requested extension graph VNFs. To construct an efficient tree solution to host the new VNFs and place the extension graph while respecting imposed connectivity with the old graph, we use the minimum Steiner Tree approach. The tree cost is defined as the total cost of the links forming the tree.

The problem of computing a minimum cost tree for a given new graph with source s and set of physical nodes candidates $\mathcal{C}$ is modeled as a Steiner tree problem and defined as follows: Given a substrate graph $G_p = (N_p, E_p)$ with a cost associated with each physical link and a subset of the vertices $X \subseteq N_p$, the goal is to find a minimum cost tree that includes all the nodes in X. For finding the minimum cost tree, we set $X = \mathcal{C} \cup \{s\}$ (i.e., source s should have a path to every node in $\mathcal{C}$).

Since in the context of the VNF-FG extension, the substrate network links are undirected, the problem corresponds to the undirected Steiner tree problem found in for instance [103] and [104]. The Steiner tree problem is NP-complete [102] and it is therefore very unlikely that a polynomial algorithm for the Steiner tree problem exists. Several exponential enumeration algorithms have been suggested [101]. Due to the inherent difficulty in solving the Steiner tree problem, much effort has been devoted to the development of approximation algorithms which produce good quality solutions.

An integrative overview of the algorithmic characteristics of three well-known polynomial time heuristics for the undirected Steiner minimum tree problem: shortest path heuristic (SPH), distance network heuristic (DNH), and average distance heuristic (ADH) is given in

[105]. The worst-case time complexity of the SPH, DNH, and ADH heuristics is respectively $O(pn^2)$, $O(m + nlogn)$, and $O(n^3)$ where p is the number of vertices to be spanned, n is the total number of vertices, and m is the total number of edges [105].

In this chapter, we make use of the Shortest Path Heuristic (SPH), originally developed in [106], because of its bounded performance (gap with optimal) and its acceptable complexity (worst-case time complexity of the SPH is $O(pn^2)$). As shown in [106] the worst-case error ratio between the solution T obtained by the SPH and the optimal solution T_o is never greater than 2. More specifically,

$$\frac{c(T)}{c(T_o)} \leq 2 \times \left(1 - \frac{1}{p}\right) \tag{4.4.15}$$

where c is the edge-cost function and p is the number of vertices to be spanned. Furthermore, this bound is tight in the sense that there are problem instances for which the ratio equals the bound.

4.4.2.1 Links Costs for Steiner Tree

The links costs for this computation are determined using the critical link concept to achieve the VNF-FG extension. We define a substrate link to be critical with respect to a physical source s and a set of candidates $\mathcal{C}$ if this link has the highest cost (i.e., very loaded link with few available bandwidth capacity). Once the critical links are identified, we avoid hosting the new virtual links on critical substrate links as much as possible. The link cost is defined in term of the residual bandwidth in the substrate link. These weights are used to select in priority the less critical links (less loaded links) and the nodes with highest available residual capacity. The residual CPU capacities on the physical nodes are also taken into account in the selection.

4.4.2.2 Steiner Tree based algorithm for VNF-FG Extension (STVE)

In the existing literature, many mechanisms can be used to calculate the distances from the source s to a set of physical candidate nodes $\mathcal{C}$ (e.g., Dijkstra's shortest path algorithm, Min-hop shortest path tree (MHT), etc). We choose the Min-hop shortest path with respect to the nodes' and links' capacities and costs to build our Steiner Tree based heuristic algorithm for VNF-FG extension placement and chaining. We run this algorithm starting from a source node s (i.e., s can be a substrate location of an old virtual node) until we reach a node candidate $k \in \mathcal{C}$. Selecting lower cost links and nodes with highest leftover capacity along the paths leads to the nodes k that will host the VNFs in the extension graph. These paths are added to the Steiner tree solution.

The proposed STVE algorithm operates in 5 steps:

1. Check the initial mapping of old VNF-FG graph G_v^{old};
2. Decompose the G_v^{new} to find an efficient mapping and construct the tree solution sequentially;
3. Find a set of physical candidate nodes $\mathcal{C}$ related to a fixed source s that promote the using of lower cost links (Method $GetCandidates$): if node k available capacity and type constraints are met, it can be a candidate;
4. Compute the shortest path between source s and candidate k using Min-hop-shortest-path algorithm;
5. Check link mapping: if links capacities are respected, confirm candidate k as a mapping solution for new VNF and add the solution path from s to k to the Steiner tree built so far;

The pseudo-code of this heuristic is shown in Algorithm 3.

4.4.3 EDVE: Eigendecomposition For VNF-FG Extension

An alternative approach is to address the VNF-FG extension problem by extending and adapting the eigendecomposition method for VNF Placement and Chaining described in

Algorithm 3: STVE algorithm pseudo-code

```
Inputs: G_p = (N_p, E_p) with edge costs, G_v^new = (N_v^new, E_v^new), the initial mapping of
  old VNF-FG graph G_v^old = (N_v^old, E_v^old)
Output:A low-cost Steiner trees hosting the VNF-FG extension graph G_v^new
foreach new virtual link e_ij ∈ E_v^new do
    if (VNF i or VNF j) ∈ N_v^old then
        s ← GetInitialMapping(VNF i or VNF j)
        C ← GetCandidates(s)
        stack ← Sort(C)
        while stack ≠ ∅ and solution = False do
            k ← Pop(stack)
            if Min-hop-shortest-path(s, k) = True then
                solution ← True
                ST ← ST + GetPathSolution(s, k)
    else if (VNF i and VNF j) ∈ N_v^new then
        s ← GetSolutionExtensionMapping(VNF i or VNF j)
        C ← GetCandidates(s)
        stack ← Sort(C)
        while stack ≠ ∅ and solution = False do
            k ← Pop(stack)
            if Min-hop-shortest-path(s, k) = True then
                solution ← True
                ST ← ST + GetPathSolution(s, k)
return ST
```

[107]. This algorithm uses the Umeyama's eigendecomposition approach [108] for optimal matching between weighted graphs. This state of the art method is based on the eigendecomposition of the adjacency matrices of the virtual graph and the physical graph (graph to host the request) and on the Hungarian method [109] to extract mapping results.

4.4.3.1 Eigendecomposition for VNF Placement and Chaining

The proposed method in [107] casts the VNF Placement and Chaining problem in networked cloud infrastructures into the weighted graph matching problem (WGMP) that finds a one to one mapping function ϕ between N_p and N_v that minimizes a distance criterion between G_p (the substrate graph) and NCT_v (requested virtual graph derived from the VNF-FG requests G_v).

This algorithm starts by computing the best paths between any two not directly connected physical nodes (paths that maximize the minimum bandwidth along their route). The substrate graph adjacency matrix A_{G_p} is updated using weights equal to the calculated bandwidths. Information about paths is stored to be used when mapping Forwarding Paths.

Since the eigendecomposition deals with graphs of the same size, the proposed algorithm in [107] adds dummy isolated vertices to the request graph in order to make the graphs of equal size. The algorithm, in fact, adds rows and columns with zeros to the adjacency matrix A_{NCT_v} to line the size to that of the substrate graph (hosting infrastructure).

Starting from the extended adjacency matrix A_{NCT_v} (padded with zeros) and the adjacency matrix of the substrate graph A_{G_p}, the algorithm derives the eigenvectors of these adjacency matrices, U_{G_p} and U_{NCT_v}. The algorithm builds from these eigenvectors a matrix M and a permutation matrix P that contains a 1 in positions corresponding to the VNFs mapping results.

4.4.3.2 Eigendecomposition For VNF-FG Extension (EDVE)

To address the VNF-FG extension problem we also make use of our Eigendecomposition algorithm [107] by extending and adapting it to connect a new request to a previously deployed slice or forwarding graph. The initial deployment is obtained using also the Eigendecomposition so the initial adjacency matrices, traces and permutation matrices are produced and available for the extension when such a request is expressed by the tenant. The extension is realized by considering the new VNF-FG graph G_v^{new} (formulated as a request graph NCT_v^{new}) and the updated substrate graph G_p to find efficient locations for new VNFs. Before searching a mapping for the VNF-FG extension, our proposed algorithm updates the adjacency matrix of the substrate graph A_{G_p} by:

- anchoring the initial mapping NCT_v^{old} by removing reserved (allocated) resources (CPU on nodes and bandwidth on links) for these previously deployed VNFs and fixing (freezing) the location in matrix M and thus in permutation matrix P of the previously deployed nodes and of course the key nodes (in this set) to which some of the extension graph nodes have to be connected to;

- setting deliberately the capacity of some of the used physical nodes to zero, to avoid co-localization of two VNFs in the same substrate node. This also ensures load balanced Eigendecomposition algorithm solutions.

The adjacency matrix of the new VNF-FG Extension request is the second matrix used by the Eigendecomposition algorithm to achieve the mapping of the new graph on the hosting infrastructure while respecting connectivity of the extension with the old graph. This matrix contains the new nodes composing the VNF-FG extension request with the nodes connected to the old graph (previously deployed tenant slice or VNF-FG) especially tagged to achieve the necessary constrained placement. This constrained placement consists in finding the links and paths that will interconnect these new nodes to their corresponding nodes in the previous graph to respect the required inter-graph connections. The other VNF-FG extension nodes are unconstrained and placed as usual according to the selected (desired) optimization criterion.

4.5 Performance Evaluation

The algorithms are compared using extensive simulations (conducted on an experimental cloud and networking platform comprised of physical servers and networking technologies, similarly to our previous work in [107] and [110]) where both the requests and hosting infrastructures are drawn using standard graph generation tools (such as the GT-ITM Tool [86]) and real infrastructure topologies (such as the Germany50 network topology [85]). The comparison includes the ILP algorithm, the reduced candidate set ILP algoithm, the Steiner Tree based algorithm (STVE), the modified Eigendecomposition algorithm as proposed in this work (EDVE) as well as the basic Eigendecomposition algorithm applied to the previous and new graph extension together by combining the two graphs into a common composite graph to be mapped by the unmodified Eigendecomposition (called "ReassignAll"). This unmodified version of the Eigendecomposition is only used to assess how suboptimal the modified Eigendecomposition is. We obviously, do not advocate the use as is of the unmodified Eigendecomposition [107] on the composite graph (i.e., old and extension merged into one composite graph to be mapped at once) since this would require

migration of the networking services or functions and thus interruption of services associated to the previously deployed graph. This version provides nevertheless a means to assess deviation from best achievable performance by the Eigendecomposition.

4.5.1 Simulation Environment

All simulations for all the algorithms run in a dedicated server in the experimental platform with the following processing capabilities: a 2.50 GHz, Quad Core server with 6 GBytes of available RAM. The VNF-FG requests are generated using a Poisson process with an average arrival rate of 2 requests per 100 time units. The lifetime of each request follows an exponential distribution with a mean of 50 time units. The Germany50 network topology [85] is used for the first assessment (with 50 nodes). This topology is defined by the German National Research and Education Network (DFN). The capacity of physical nodes and links are generated randomly in the $[100, 120]$ interval. The size of the initial VNF-FG requests is arbitrarily set to 4 nodes and the size of the extension is set to 3 nodes for each VNF-FG request. The GT-ITM [86] tool is used to generate the requested VNF-FG topologies. The required capacities of the initial VNF-FG are fixed to 10 CPU units for each virtual node and 10 bandwidth units for each virtual link. The extended VNF-FG computing and bandwidth requirements are set to 20 CPU units per node and 20 bandwidth units per link. The connectivity between nodes in the VNF-FG is set to 0.3 (30%).

The performance assessment at larger scale uses the GT-ITM tool to generate network topologies with 200 nodes and a connectivity of 0.3 (or 30%). The physical resource capacities (i.e., CPU and bandwidth) are also drawn randomly in the $[40, 50]$ interval. The VNF-FG requests are generated using a Poisson process with an average arrival rate of 5 requests per 100 time units. The lifetime of each request follows an exponential distribution with a mean of 200 time units. The size of the initial VNF-FG requests is also set to 4 nodes and the size of the extension is set to 3 nodes for each VNF-FG request. The required CPU for each VNF in the initial VNF-FG is set to 15 units. The required bandwidth between two VNFs, to ensure communication between them, is set also to 15 units. The extension VNF-FG computing and bandwidth requirements are set to 20. The connectivity between nodes in the VNF-FG is set to 0.3 (30%).

For the realistic topology and the large-scale evaluations, 1000 VNF-FG requests are generated and an extension request is triggered for each generated request. Since we are focussing on extension and adaptation of already embedded VNF-FGs, without any loss in generality, we used the heuristic approach of [107] to generate the initial VNF-FG mapping. The ILP solver used in our experiments is Cplex [111].

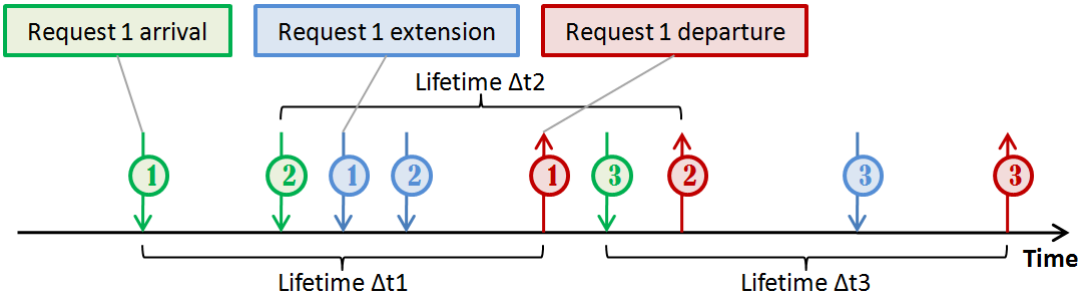


FIGURE 4.4: Processing of VNF-FGs

Arrivals, extensions, and departures occur over time as depicted in the example in Figure 4.4. The time an extension request might take to arrive is a random variable between the arrival and departure time of the corresponding initial request.

4.5.2 Performance Metrics

The metrics used for the performance evaluation are typical indicators for placement algorithms and as defined below:

- **Successful extensions** is the number of VNF-FG extension requests that are fulfilled or successfully placed in the hosting infrastructure while respecting connectivity constraints with the previously deployed graph. This quantity should be maximized since it is equivalent to minimizing rejection rate of requests for the algorithms.

- **Extension ratio** is the ratio of successful extensions to the number of initial VNF-FG graph requests that have been accepted. Since the initial VNF-FG placement is achieved using the Eigendecomposition algorithm to initialize the evaluation with the initial graph before extension for all the algorithms, this metric reflects the relative efficiency of the algorithms, starting from the very same previously deployed graph for all algorithms to compare them on a common and fair basis.

- **Execution time** is the time required by each algorithm to find a placement solution for the extension requests. This metric reflects the algorithms complexity.

- **Acceptance revenue** is the service provider realized (generated) revenue (benefits) at time t when extension requests are successfully fulfilled and accepted. The acceptance revenue is formally expressed as:

$$\mathbb{R}(t) = \sum_{\mathcal{R}eq_{ex} \in \mathcal{AR}_t} \mathbb{R}\left(\mathcal{R}eq_{ex}\right) \tag{4.5.16}$$

where $\mathcal{AR}_t$ is the set of accepted extension requests up to time t and $\mathbb{R}(\mathcal{R}eq_{ex})$ is the extension request $\mathcal{R}eq_{ex}$ gain expressed as in [112] and [113]:

$$\mathbb{R}(\mathcal{R}eq_{ex}) = \alpha \sum_{i \in N_v^{new}} (cpu_i \times U_{cpu}) + \sum_{e_{ij} \in E_v^{new}} \left(bw_{e_{ij}} \times U_{bw}\right) \tag{4.5.17}$$

where U_{cpu} is the realized revenue from allocating one unit of CPU resources and U_{bw} is the earned revenue from the allocation of one bandwidth unit, and $\alpha = \sum_{k \in \mathcal{S}} \frac{CPU_k}{min_{CPU}}$ is a tunable weight that reflects the quality of the solutions. In fact, the revenue increases by choosing the best locations which correspond to nodes with more free resources (i.e., $\mathcal{S}$ is the set of physical nodes solutions, the other parameters CPU_k and min_{CPU} are described in Section 4.4).

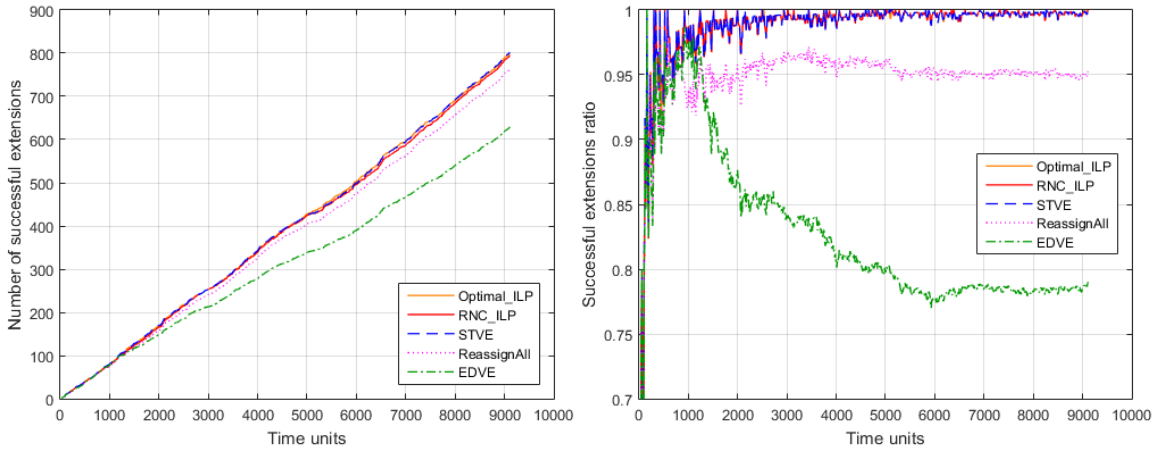
Note that $\mathbb{R}$ is the total revenue taking into consideration all the incoming extension requests.

4.5.3 Evaluation Results

The performance is evaluated on the Germany50 network for all algorithms since the size of this topology is small and the ILP algorithm produces solutions in acceptable time. For larger graph sizes, we resort to random graph generations with hundreds of nodes and links and compare the performance of all other algorithms without the ILP that does not scale with problem size.

4.5.3.1 Evaluation on a realistic topology

The topology used for the evaluation is as stated the Germany50 network with a topology of 50 nodes. The ILP is included in the performance comparison for this topology. Since we also use a modified ILP, a heuristic called the RNC_ILP, that uses a subset of candidates and hence uses partial exploration, we denote the original and optimal ILP as "Optimal" to differentiate explicitly these algorithms and avoid any confusion.



(a) Successful extensions (b) Extensions ratio

FIGURE 4.5: Germany50 network topology results: Extensions

Figure 4.5(a) and Figure 4.5(b) depict respectively the results in terms of successful extensions and extension ratio (two similar metrics with the second normalized metric by a common reference graph, the initially deployed slice or VNF-FG prior to the extension). The Optimal_ILP, the RNC_ILP and the STVE algorithms outperform the eigendecomposition based solutions (i.e., EDVE, ReassignAll) by accepting 800 extension requests versus 629 for the EDVE. The normalized metric confirms these results passed a transient state (passed 2000 time units) where the achieved ratios are around 99% for the Optimal_ILP, the STVE and the RNC_ILP, 95% for the ReassignAll, and only 78% for the EDVE. The Eigendecomposition approach is known to provide much better results for highly connected topology compared with weakly connected graphs as will be shown in the more extensive simulations with randomly generated topologies.

Figures 4.6 and 4.7 compare the quality realized by the proposed algorithms in terms of objective function, achieved maximum in Equation 4.4.14, to provide insight on the relative

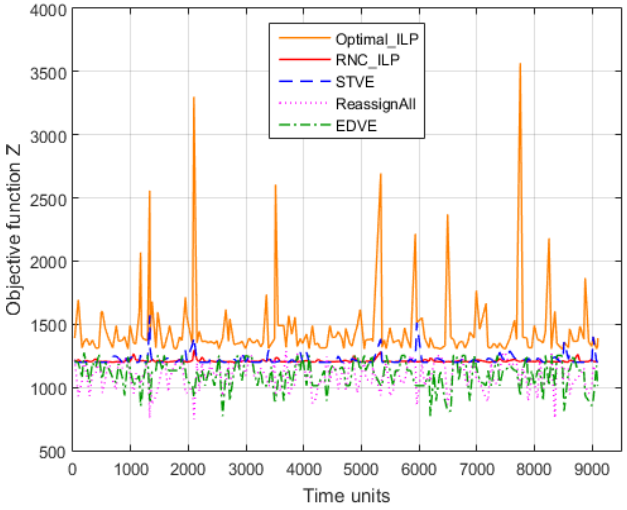


FIGURE 4.6: Quality of the mapping

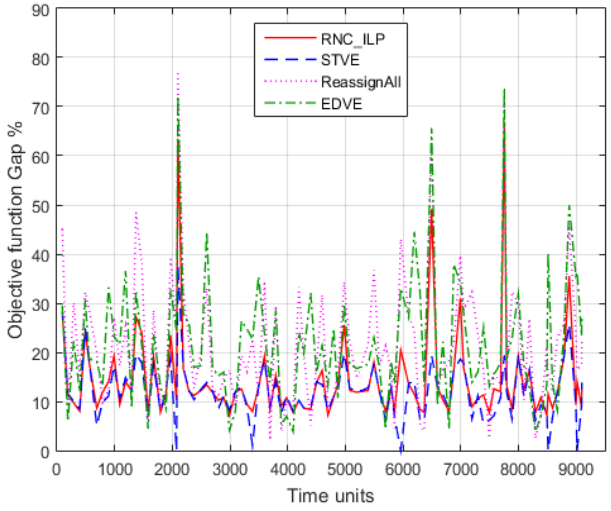


FIGURE 4.7: Objective function gap compared with Optimal_ILP

performance of the heuristics to the optimal solutions found by the ILP. Since we aim in this chapter at maximizing provider revenue, we define the mapping quality based on the value of the objective function Z that describes the algorithms ability to achieve load balancing on the hosting infrastructure.

The gap is computed using Equation 4.5.18.

$$Gap(\%) = \frac{Z\left(Optimal\_ILP\right) - Z\left(ALG\right)}{Z\left(Optimal\_ILP\right)} \times 100 \tag{4.5.18}$$

where ALG represents the selected algorithm for comparison with the ILP, that is: EDVE, ReassignAll, RNC_ILP or STVE.

Figure 4.6 reports the achieved Z values and indicates clearly that the closest in performance to the ILP is the Steiner algorithm (STVE). The other algorithms are outperformed in terms of quality of the solutions. Figure 4.7 depicts the gap of the algorithms in quality of the solutions compared with the ILP, and reveals in an enhanced scale their relative performance. The STVE is shown to be within 10% to 20% of the ILP performance and in some cases achieves the same maximum for the objective function (points touching the abscissa at points 2073, 5961, 8515, 9018 time units).

The next closest algorithm in achieved quality is the RNC_ILP whose gap lies in the range 10% to 30% of the ILP objective function. The Eigendecomposition approaches, ReassignAll and EDVE, are further away in quality from the ILP and the gap can exceed in some cases 70% and spans typically a larger range from 10% to 78%. Obviously ReassignAll, that embeds again all the VNF-FGs (old and new) outperforms the EDVE that embeds only the extension without affecting the previously deployed VNF-FG or service graph.

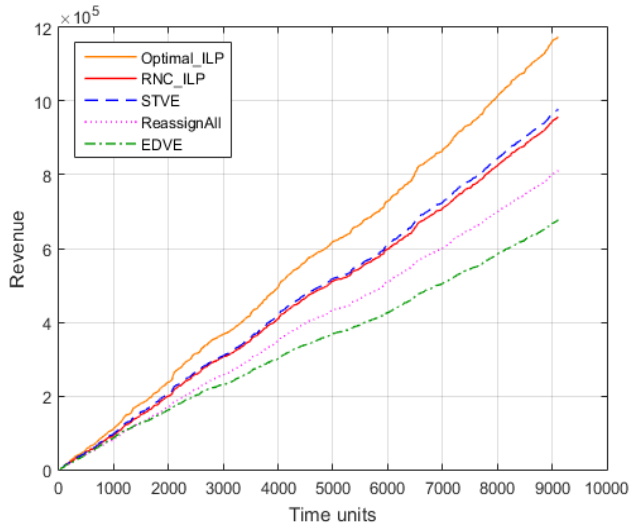


FIGURE 4.8: Acceptance revenue

The computed acceptance revenue reported in (Figure 4.8) corresponds to the total revenue generated when accepting the new extension requests at time t. The Optimal_ILP achieved provider revenue is 16.75% higher than the STVE, and respectively 30.94% and 42.3% better than ReassignAll and EDVE. This interpretation confirms the results obtained in

Figure 4.5 on successful extensions. Summarizing all these results, the STVE stands out as the closest in performance to the ILP and it will be shown that this algorithm scales best with increasing problem size if larger graphs are generated for the performance evaluation.

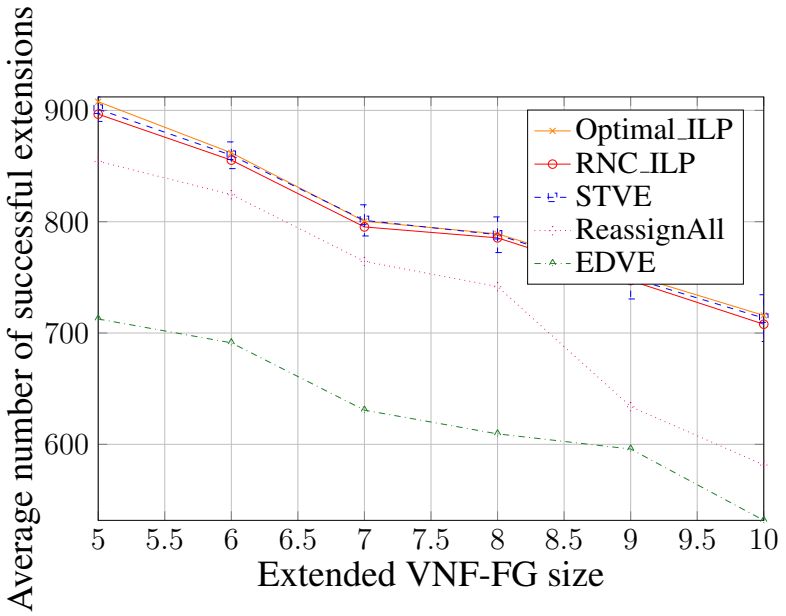
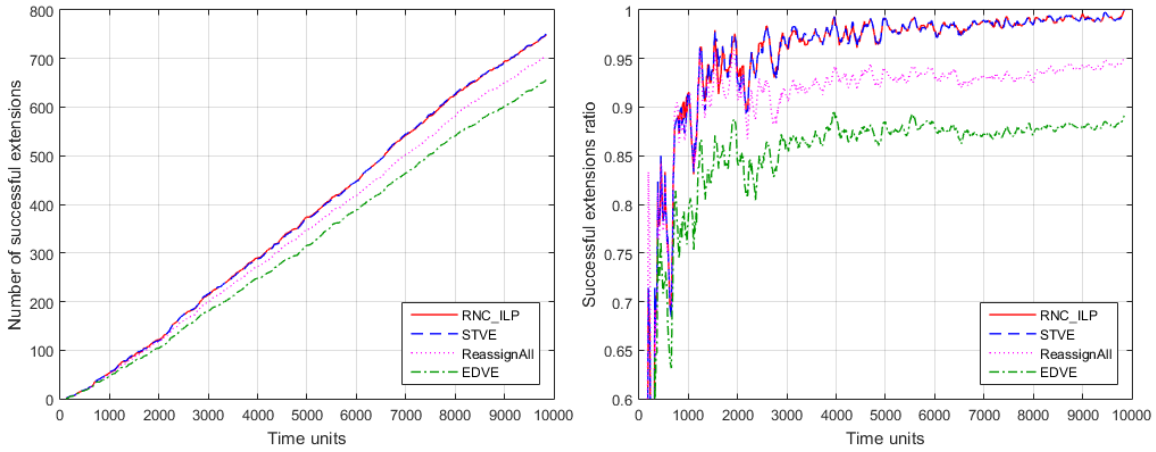


FIGURE 4.9: Successful extensions w.r.t VNF-FG size variation

The results in Figure 4.5 to Figure 4.8 reflect the observed behavior of the algorithms as a function of simulation time over multiple runs. Figure 4.9 evaluates the average number of successful extensions of the algorithms as a function of increasing VNF-FG request sizes while varying the capacity of nodes and links. The results in Figure 4.9, presented with a 95% confidence interval (shown only for the Steiner algorithm for legibility reasons), depict a high similarity with results in Figure 4.5(a) (more precisely for VNF-FG size of 7 nodes).

4.5.3.2 Large-scale evaluation

To analyze the behavior of the algorithms and their scalability with problem size, larger hosting infrastructures (NFV-Is with 200 nodes) are randomly generated using the GT-ITM Tool. This evaluation should confirm the previous results and reveal the ability of the algorithms to scale with increasing problem size. In addition to the previous performance metrics, we report the execution time of each algorithm to shed light on their scalability.



(a) Successful extensions (b) Extensions ratio

FIGURE 4.10: Large-scale network topology extension results

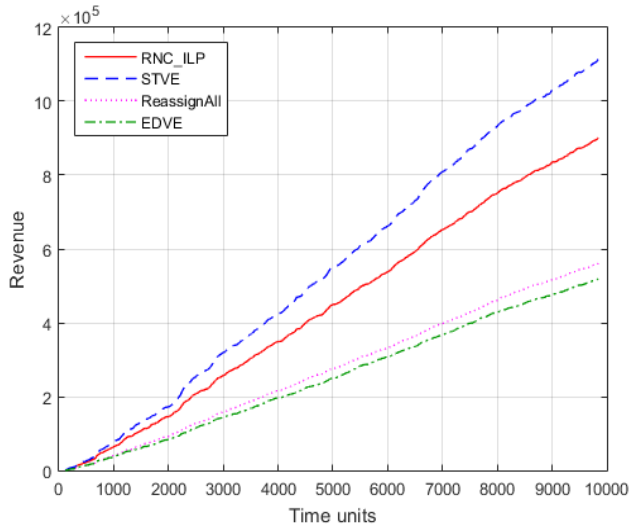


FIGURE 4.11: Acceptance revenue

Figures 4.10(a), 4.10(b), and Figure 4.11 report similar results as for the Germany50 topology. We notice that the RNC_ILP and the STVE algorithms outperform the eigendecomposition based solutions (i.e., EDVE, ReassignAll) by accepting more requests and ensuring higher revenues.

Figure 4.12 depicts the average number of successful extensions as a function of NFV-I size (varying from 100 to 500). Note that several network topologies are considered for each fixed number of physical nodes to ensure that our results are stable and reliable. The results obtained from multiple simulations are averaged and presented with 95% confidence interval on the reported results (shown only for the Steiner algorithm for legibility reasons).

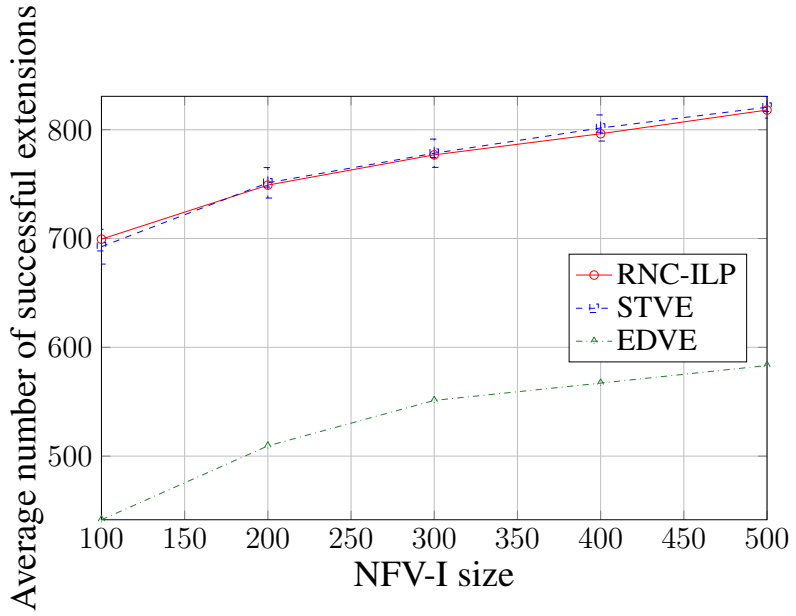


FIGURE 4.12: Successful extensions w.r.t NFV-I size variation

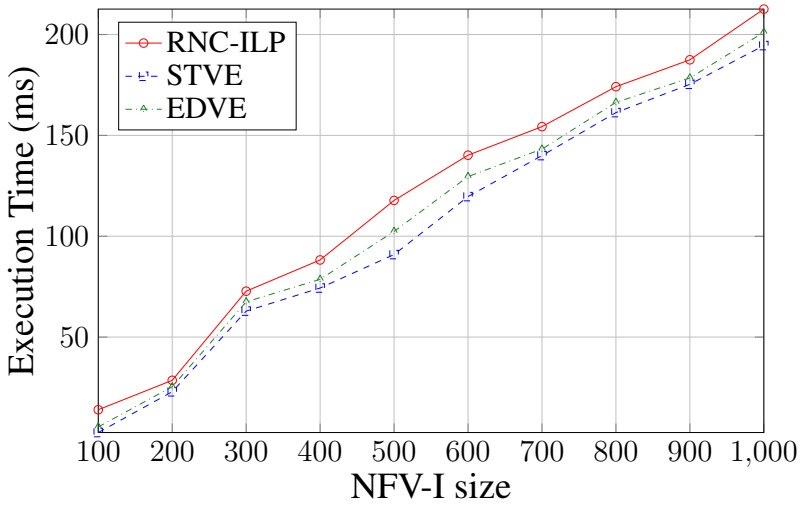


FIGURE 4.13: Execution Time w.r.t NFV-I size variation

4.5.3.3 Execution time (Convergence time)

The aspect that deserves more attention is the algorithms execution time as reported in Figure 4.13 that clearly shows that the Steiner and Eigendecomposition methods have the best performance for this metric since they find solutions for the extension requests placement in few milliseconds for 100 nodes, few tens of milliseconds for 200 nodes and need no more than 60 milliseconds for 300 nodes. In addition, results reported in Figure 4.13 indicate that

the execution times of the Steiner and Eigendecomposition approaches grow linearly like the RNC_ILP. The RNC_ILP has a performance in the vicinity of these two algorithms but requires selection of a reduced set of candidates to realize this gain. Limiting the exploration of the ILP is not necessarily recommended if alternatives, such as the Steiner Tree based algorithm, can find better solutions in a shorter execution time. The eigendecomposition, even if faster, is outperformed by the RNC_ILP in terms of acceptance ratio, revenue, and proximity to the Optimal_ILP. An aspect that also requires additional investigation is the performance of the Eigendecomposition methods for highly interconnected topologies. Eigendecomposition algorithms are known to perform much better when connectivity is higher.

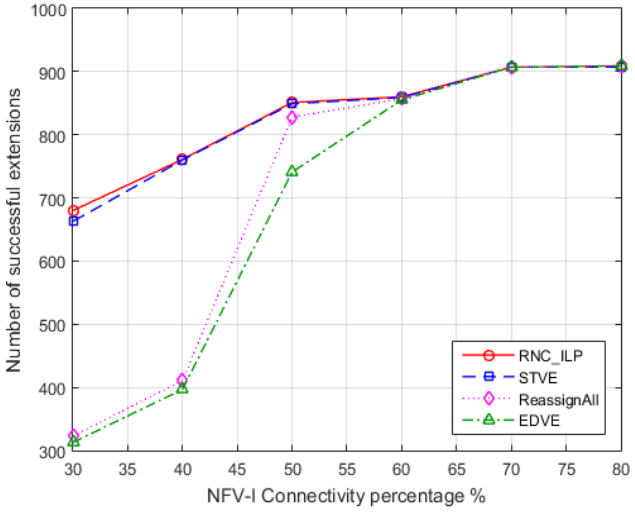


FIGURE 4.14: Successful extensions for varying NFV-I connectivity

4.5.3.4 Variation of NFV-I connectivity

To see the impact of the substrate graph connectivity on the behavior of our algorithms, the results are reported for an NFV-I connectivity percentage spanning a 30% to 80% connectivity in the evaluated hosting infrastructures with a number of nodes fixed at 100 nodes. The substrate graph size is fixed to 100 nodes. As previously hinted, the Eigendecomposition algorithms catch up in performance with all other algorithms as the network connectivity increases, especially when connectivity exceeds 60% as depicted in Figure 4.14. Figure 4.15 confirms the stability of the Eigendecomposition approaches with respect to execution time that remains fairly constant and more importantly rather low compared to the RNC_ILP

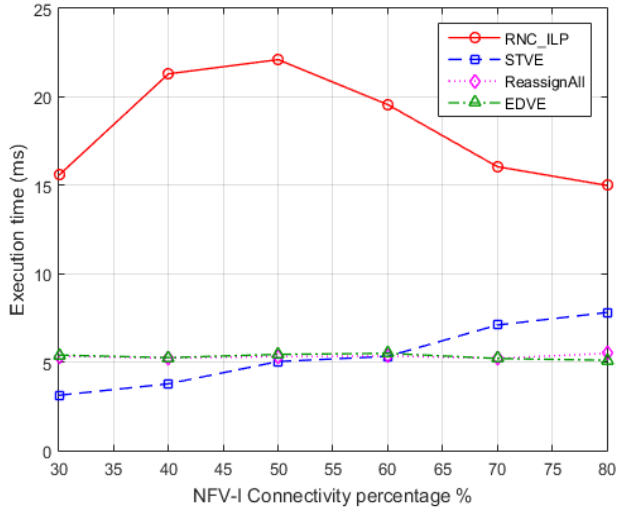


FIGURE 4.15: Execution time when varying the NFV-I connectivity

and very close to the Steiner based approach. The Steiner tends to increase slightly with increasing connectivity but this can be considered marginal with both the Steiner and Eigen-decomposition requiring around 5 milliseconds to find a placement solution for the extension requests. The RNC_ILP takes longer, in the order of tens of milliseconds ($> 15ms$) to find a solution for the evaluated graph sizes (NFV-Is with 100 nodes).

4.6 Conclusion

This chapter, to the best of our knowledge, is probably the first to address the problem of VNF-FG extension with the main objective of maximizing provider revenue by reducing the rejection of extension requests when faced with dynamic and rising demand and traffic load. We propose an ILP model as an exact algorithm acting as a baseline for comparison and as the solution for small problem sizes. To improve scalability, we also propose two heuristic algorithms: a new Steiner Tree based algorithm and an eigendecomposition based approach. We show that the solutions can efficiently extend virtualized network functions forwarding graphs with good execution time and extension requests acceptation performance over distributed and dynamically varying cloud environments. The Steiner Tree is found to be the best overall performance tradeoff solution for the VNF-FG extension problem as it performs uniformly well across all scenarios and can scale for large problem sizes. The Eigendecomposition methods are also quite fast but only recommended if the hosting

infrastructures are highly connected, the only situation where they achieve similar performance to the Steiner and the ILP based solutions.

Chapter 5

Enhanced Reinforcement Learning Approach for VNF-FG Embedding

5.1 Introduction

This chapter, as the previous ones, addresses also the placement of service function chains (SFC) and the placement of virtualized network function forwarding graphs (VNF-FG) in NFV Infrastructures (NFVIs). This NP-Hard placement problem has been extensively investigated in the past, just like in our previous contribution, by mostly achieving instantaneous and sequential placement of the requests. These approaches unfortunately result in optimal placement without consideration for longer term optimization and reward. The algorithms have seldom been combined with past knowledge (using learning) or with prediction of future demands to use resources more efficiently and increase the number of accepted requests. One way to realize this longer term reward is to model and resolve the problem using a Markov Decision Process (MDP) based characterisation. A practical way to resolve the longer term reward placement problem is to use Reinforcement Learning (combined with the MDP modeling or alone) instead of traditional combinatorial and convex optimization leading only to instantaneous optimality and to rather limited long term benefits (or reward).

To highlight the benefits of using a reinforcement learning approach for the VNF-FG and SFC placement problem, we compare the performance to the algorithms proposed in the previous chapters, namely Integer Linear Program (ILP) using an exact formulation for instantaneous optimal placement handling one request at a time. We use a load balancing criterion throughout the infrastructure for all algorithms involved in the performance comparison. Other criteria than load balancing can be selected, such as maximum packing and consolidation to reduce resource usage and favor minimization of energy consumption and operational cost. The relative performance of the algorithms is not expected to change or will not be affected if another criterion is adopted to conduct the comparison. We, hence, limit the study to load balancing objectives. The ILP, that does not scale with problem size, is used as a reference for small hosting infrastructures.

A modified ILP, that operates on a reduced set of candidate hosts, called R_ILP, is proposed to improve scalability and speed compared to the ILP and is part of the overall performance evaluation and assessment. A batch algorithm, called BR_ILP, that operates on a group of requests, handled as a composite graph request, is also included in the comparison since it is expected to perform better than the ILP in terms of placement efficiency over longer time scales. The performance of the algorithms is also compared with an Eigendecomposition algorithm with reasonable complexity (complexity in $O(n^5)$, where n is the hosting graph size) shown to have good placement performance. The Reinforcement Learning solution uses standard Q-Learning enhanced with expert knowledge to accelerate learning during training and when unknown conditions and situations emerge.

Section 5.2 presents related work. The problem formulation is described in Section 5.3. The proposed chain placement algorithms are introduced in Section 5.4. Section 5.5 reports the performance evaluation results and Section 5.6 summarizes the main findings.

5.2 Related work

The need to dynamically deploy virtualized network services on-demand, through VNF-FG embedding, is identified as the core technology of 5G networks. Therefore, this issue has been at the very center of academic and industrial research in recent years. Here, we

give a summary of the use of Reinforcement Learning in networking and especially in the VNF-FG embedding domain.

Since the VNF-FG embedding problem can be well described as a Markov decision process (MDP), then Reinforcement Learning (RL) is a good framework to use to find approximate optimal solutions. Recently, some works proposed RL-based approaches for VNF-FG embedding. With RL, an agent learns by interacting with its environment. The agent learns to perform the best action for each state by performing actions and observing the rewards. Given enough observations, an optimal policy can be learned. Thus, the training data in reinforcement learning is a set of state-action-rewards. Authors in [114] map the VNF-FG in two stages: node mapping stage then link mapping stage. They propose two algorithms for link mapping, based on a Multi-Commodity Flow approach and a shortest-path approach. Regarding the node mapping, the authors propose an MDP-based approach. Consequently, this approach is time consuming and not adapted for real-time embedding. In [115], authors propose a multi-agent-based reinforcement learning approach for virtual network embedding. These agents evaluate the feedback to learn the best policy to adopt in order to optimally allocate the required resources to the virtual nodes and links. In [116], authors propose a deep reinforcement learning-based approach for multi-domain VNF-FG embedding. Authors in [117] tackle the SFC allocation problem and present a learning method that places VNFs on an appropriate node that maximizes the performance of VNFs according to the load condition of the physical network. However, this method takes a huge amount of time to converge under an extensive exploration space. In [118], authors propose an RL-based algorithm to solve the NP-hard VNF scheduling problem.

In some existing studies, the reinforcement learning is also used for VNF migration [78], [77], [119], and scaling [79], [72] problem under dynamic network load. Furthermore, the large action space, when selecting which VNF instance to migrate, leads to high complexity and poor convergence performance.

To the best of our knowledge, we are the first to propose an enhanced RL-based approach combined with an expert knowledge mechanism to avoid a lengthy training process for VNF-FG placement and chaining.

5.3 Problem Description

The VNF-FG embedding optimization problem, extensively addressed in the literature, corresponds to the placement of the requested VNFs and flow paths in the hosting infrastructure.

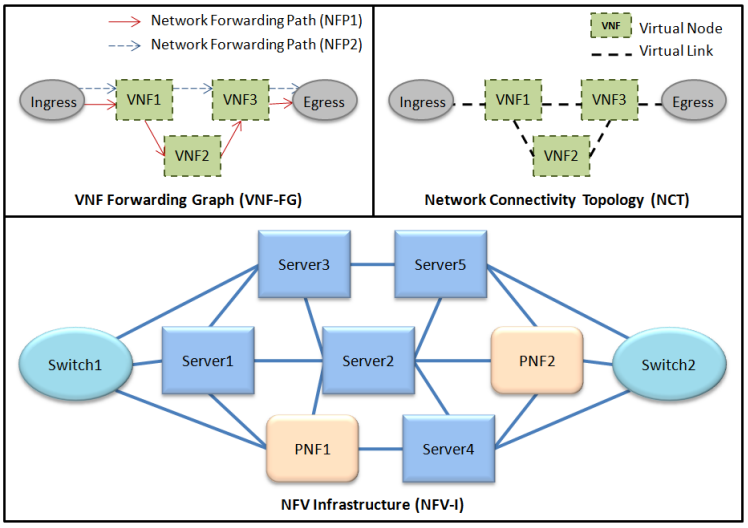


FIGURE 5.1: An example of VNF-FG and NFV-I topologies

5.3.1 Substrate Graph or NFV Infrastructure

Deriving an ILP formulation of the VNF-FG placement problem requires a mathematical graph representation of the service graph (SFC or VNF-FG) and the hosting infrastructure.

The physical infrastructure, as defined by the ETSI NFV Infrastructure (NFV-I) in [83], is modeled as an undirected weighted graph $G_p = (N_p, E_p)$ where E_p is the set of physical links and N_p is the set of physical nodes.

Each substrate node, $k \in N_p$, is characterized by its i) available processing capacity CPU_k (this can be easily extended to other types of resources: such as storage or memory capacity), and ii) type T_k: switch, server or Physical Network Function (PNF). The PNFs are the traditional physical middleboxes implementing network functions. Each physical link (i.e., $e \in E_p$) is characterized by its available bandwidth BW_e.

An example of such an infrastructure, known as the NFV-I, including two PNFs and two switches (serving the ingress and egress flows in the VNF-FG topologies) and some interconnected servers is presented in Figure 5.1.

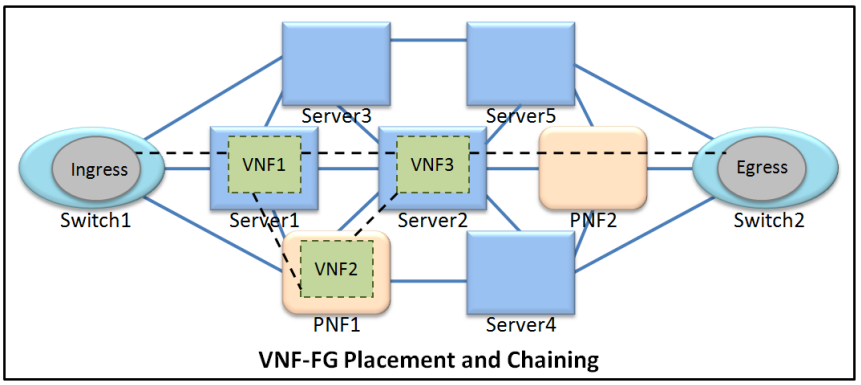


FIGURE 5.2: Mapping of the VNF-FG in the NFV-I

5.3.2 VNF Forwarding Graph or SFC Graph

The client request (i.e., a requested SFC) is modeled as a directed graph $G_v = (N_v, E_v)$ where N_v is the set of virtual nodes and E_v is the set of virtual links in the requested graph. Each virtual node, $i \in N_v$, is characterized by its i) required processing power cpu_i and ii) its type t_i: VNF or switch (i.e., ingress or egress). Each virtual link $e_{ij} \in E_v$ is described by its required bandwidth $bw_{e_{ij}}$. Note that we can also consider instead the end to end latency if this is the criterion to be taken into account. Our model is generic and can be adjusted based on the client criteria and requirements.

As specified by the SFC IETF working groups [94], we associate a VNF-type to each VNF to represent the network service or function type (e.g., firewall, NAT, etc.). The VNFs can be hosted only by servers or PNFs having the same type. The ingress and egress nodes can be hosted only by switches. Figure 5.1 depicts two Network Forwarding Paths (NFP). Each Forwarding Path NFP describes the ordered VNF sequence the traffic must pass through.

From the VNF forwarding graph G_v, we derive an intermediate request graph called the Network Connectivity Topology graph NCT_v. The $NCT_v = (N_v, E_v)$ is a weighted undirected graph having exactly the same set of nodes and edges than G_v. The key attribute of

an NCT node $i \in N_v$ is its requested processing capacity cpu_i. The weight (or requested bandwidth $bw_{e_{ij}}$) of an NCT virtual link $e_{ij} \in E_v$ is the sum of the requested bandwidths of all the VNF flows passing through it.

5.3.3 VNF-FG placement and chaining

Figure 5.2 depicts a placement solution for the VNF forwarding graph (VNF-FG) highlighted by the dashed lines, starting from the ingress switch 1, crossing the VNF 1, VNF 2, and VNF 3 (hosted respectively in Server 1, PNF 1, and Server 2), and ending at the egress switch 2.

The infrastructure providers can map the VNFs using multiple objectives: such as minimizing mapping costs [120], maximizing acceptance ratios and improving provider gains, and improving energy efficiency [121].

5.4 Proposals

To solve the VNF-FG placement and chaining problem, we propose an ILP model and an enhanced Q-Learning based approach, and evaluate and compare their performance.

5.4.1 ILP Formulation

We now formulate the ILP model of the VNF placement and chaining problem using an objective function that aims at finding load balanced solutions. The objective function can be adapted at will depending upon the desired provider optimization objective or policies (such as load balancing to avoid congestion, consolidation, maximize revenue, etc.). We limit the scope of this chapter to a load balancing criterion in the optimization. In Table 5.1 we summarize the parameters and the variables used in the ILP formulation.

Objective function: A pragmatic approach to achieve load balancing is to select in priority candidate hosts that have the highest amount of free resources to gradually load the

TABLE 5.1: Table of Notations

Notation	**Description**
CPU_k	Residual capacity in a physical node k
P_{k_i,k_j}	Physical path interconnecting physical nodes k_i, k_j
$\mathcal{C}$	Set of physical nodes candidates
$\mathcal{P}$	Set of physical paths candidates
min_{CPU}	The minimum capacity in $\mathcal{C}$
BW_e	Residual bandwidth in one physical link e
δ_{ep}	A boolean parameter indicating if the physical link $e \in E_p$ belongs to path $P_{k_i,k_j} \in \mathcal{P}$: $\delta_{ep} = 1 \Leftrightarrow e \in P_{k_i,k_j}$
N_v, E_v	Set of virtual nodes, Set of virtual links
cpu_i	Required capacity by virtual node i
$bw_{e_{ij}}$	Required bandwidth between virtual nodes i and j
x_{ik}	A binary variable indicating whether VNF i is mapped to physical node k
$y_{e_{ij},P_{k_i,k_j}}$	A binary variable indicating whether virtual link e_{ij} is mapped to physical path P_{k_i,k_j}

infrastructure and distribute the load across nodes and links. This is accomplished using an objective function Z, defined in Equation (5.4.1), and combined with a set of equalities and inequalities reflecting the tenant constraints and the providers' obligations and interests. The variable min_{CPU}, in Equation (5.4.1), the smallest amount of available compute resources in all nodes in set C, constant over each run, serves as a normalization factor for the objective function. To maximize Z, the ILP selects in priority hosting nodes with the largest amount of available resources to ensure load balancing across the entire infrastructure.

$$Z = \sum_{i \in N_v} cpu_i \times \left(\sum_{k \in \mathcal{C}} \frac{CPU_k}{min_{CPU}} \times x_{ik}\right) \tag{5.4.1}$$

where the used binary variables are defined as follows:

$$x_{ik} = \begin{cases} 1, & \text{if the VNF } i \text{ is hosted on the substrate node } k; \\ 0, & \text{otherwise.} \end{cases} \tag{5.4.2}$$

$$y_{e_{ij},P_{k_i,k_j}} = \begin{cases} 1, & \text{if the virtual link } e_{ij} \text{ is mapped} \\ & \text{to physical path } P_{k_i,k_j}; \\ 0, & \text{otherwise.} \end{cases} \tag{5.4.3}$$

Furthermore, the following constraints must be satisfied in order to guarantee a feasible solution:

Node mapping constraint: ensures that each VNF i is mapped to exactly one physical candidate node k and its constituent components (VNFCs) are co-located in the same node.

$$\sum_{k \in \mathcal{C}} x_{ik} = 1, \ \ \forall i \in N_v \tag{5.4.4}$$

CPU capacity constraint: ensures that the available resources in a physical node k are sufficient to host VNF i.

$$\sum_{i \in N_v} cpu_i \times x_{ik} \leq CPU_k, \ \ \forall k \in \mathcal{C} \tag{5.4.5}$$

Link mapping constraint: makes sure that each virtual link e_{ij} is mapped to exactly one physical path P_{k_i,k_j} where k_i is a physical candidate for the i^{th} VNF and k_j is a physical candidate for the j^{th} VNF. Note that i and j are neighbors in the VNF-FG request. We compute $\mathcal{K}$-shortest paths candidates P_{k_i,k_j} using Dijkstra's algorithm based on residual bandwidth of links (not based on number of hops). It may find a longer but lightly loaded path better than the heavily loaded shortest path.

$$\sum_{k_i \in \mathcal{C}} \sum_{k_j \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = 1, \ \ \forall e_{ij} \in E_v, \ \forall i,j \in N_v \tag{5.4.6}$$

Bandwidth constraint: The residual bandwidth in a candidate physical path P_{k_i,k_j} has to satisfy the bandwidth requirement of virtual link e_{ij} to host the said link. We should not

violate the remaining bandwidth of each physical link $e \in E_p$ on the physical path P_{k_i,k_j}.

$$\sum_{e_{ij} \in E_v} bw_{e_{ij}} \times y_{e_{ij},P_{k_i,k_j}} \times \delta_{ep} \leq BW_e, \quad \forall e \in P_{k_i,k_j},\ \forall P_{k_i,k_j} \in \mathcal{P} \tag{5.4.7}$$

Node and link mapping constraint (source type): When a VNF i is mapped to a physical candidate node k_i, each virtual link e_{ij}, starting from VNF i, has to be mapped to a physical path P_{k_i,k_j} having k_i as one of its endpoint or extremity (source). The other endpoint is k_j.

$$\sum_{k_j \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{ik_i}, \quad \forall e_{ij} \in E_v,\ \forall k_i \in \mathcal{C} \tag{5.4.8}$$

Node and link mapping constraint (destination type): Similarly to the previous constraint (source type), if a VNF j is mapped to a physical candidate node k_j, then each virtual link e_{ij} has to be mapped to a physical path having k_j as one of its endpoints (destination).

$$\sum_{k_i \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{jk_j}, \quad \forall e_{ij} \in E_v,\ \forall k_j \in \mathcal{C} \tag{5.4.9}$$

Node separation constraint: This constraint corresponds to situations where VNFs have to be separated and mapped onto distinct nodes for security reasons for example or simply due to tenant requirements or application constraints. In this case each VNF i and j are mapped into nodes k_i and k_j with $k_i \neq k_j$.

$$x_{ik} + x_{jk} \leq 1, \quad \forall i,j \in N_v,\ \forall k \in \mathcal{C} \tag{5.4.10}$$

5.4.2 ILP with Reduced Number of Candidate hosts (R_ILP)

In addition to the ILP formulation of Equation (5.4.1), known not to scale since the VNF-FG embedding problem is NP-Hard, we propose a modified ILP that uses a reduced set of candidate hosts from the set of identified candidates, hosting nodes that meet the conditions and constraints in Equation (5.4.1). To accomplish this reduction, we select among the set of

candidates a subset of much fewer candidates (typically 10 to 20 candidates are sufficient to obtain good solutions, i.e., sufficiently close from optimal). Among these subsets, we apply the ILP of Equation (5.4.1) to this subset to make the final mapping of the VNF-FG onto the subset. As mentioned earlier, the objective is to achieve load balancing to maximize the acceptance rate of requests and, thus, improve providers gains. The pseudo-code of the candidate subset selection is illustrated in Algorithm 4, where we select eligible hosts with more available resources (the $queue$ is sorted by available resources) to favor load balancing.

Algorithm 4: Candidate selection module pseudo-code

```
Inputs: G_p, G_v, maxCand
Output: A set of candidate hosts C
queue ← ∅
C ← ∅
foreach physical node k ∈ N_p do
    queue ← queue ∪ k
stack ← Sort(queue)
repeat
    C ← C ∪ Pop(stack)
until size(C) = maxCand;
return C
```

5.4.3 Batch placement and chaining algorithm (BR_ILP)

The current state of the art mainly focuses on online VNF-FG request embedding. Our objective is also to explore batch embedding and to propose a seminal work and analysis for this alternative approach to the VNF-FG placement and chaining problem. With the batch mode more client requests are likely to be accepted and served compared to the online mode. The number of accesses to the NFV-I in the batch mode is smaller. Mapping a batch of n requests requires only one access to the NFV-I while n accesses will be needed for the online mode. Operating in batch mode, by handling multiple requests over a viable time interval, can potentially provide benefits in terms of quality of the optimization with a likelihood of coming closer to optimal solutions. When operating online, the solutions can be instantaneously optimal but are typically suboptimal over longer time durations since there is no look-ahead nor combined look at past, present and future requests. The

online solutions have the advantage of not incurring additional delays in providing embedding solutions on a sequential basis, processing one request at a time. Operating in batch mode enables, for instance, the processing of multiple requests as a group or an equivalent composite request. The batch mode will lead to solutions closer to optimal for the group of requests assuming the size of the batch and the interval over which the group is composed are adequately selected. Providers can reduce the cost of placing and chaining the multiple requests and hence improve their revenues as long as service level agreements with the tenants are met. The batch mode should lead to a better utilization of the provider physical infrastructure and the provider provisioned VNFs. The providers would logically have more freedom to improve or increase their profit and at the same time minimize the rejection rate of tenant requests. This can be beneficial to all stakeholders but we need to verify the conditions for such improvement compared to the online mode. In addition, the improvement must be significant to justify the implementation of the batch mode. The goal of our analysis and study is to set the stage for deeper investigations into the batch mode and facilitate the development of formal performance assessment models in the future.

At this stage, we conduct an initial study based on the batch embedding mode described in Figure 5.3 where there is a time window, denoted by W_b, during which the incoming requests are stored in a dedicated list denoted by $A(W_b)$. Arrivals and departures occur during such windows as depicted in the second batch window example in Figure 5.3. At the end of each batch window W_b there is a required processing period during which the provider executes a management policy to decide for instance i) the embedding order of the already arrived VNF-FG requests (the stored requests in $A(W_b)$) and ii) the requests to be accepted in case there are insufficient physical resources to host all the requests in the batch. This should always be done to achieve provider benefits and maximizing the satisfaction of tenants by keeping rejection rates to a strict minimum, i.e., to the smallest possible value depending on available resources during the batch. Our proposed batch-oriented algorithm applies the R_ILP to the batch and takes into account the scalability of our online R_ILP algorithm by limiting the batch size (retaining only the best candidate hosts) to obtain good solutions in reasonable times. The algorithm handles the queued VNF-FG requests for batch processing in a heuristic way. Instead of trying to place optimally the entire batch

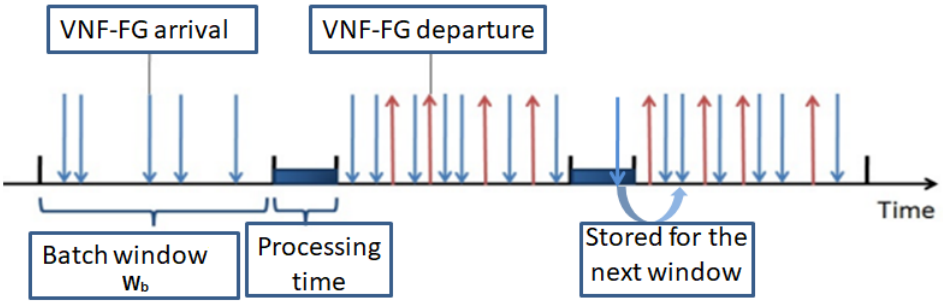


FIGURE 5.3: Processing of VNF-FGs in a batch mode

$A(W_b)$ (i.e., the stored incoming VNF-FGs during the batch window W_b), and end up rejecting all queued VNF-FG requests when there are no optimal placement solutions for the entire batch, the algorithm processes the VNF-FG requests by analyzing the whole batch to determine the order in which the requests must be treated to meet a selected objective such as maximizing provider gain, minimizing energy consumption, maximizing resource utilization, etc. In our performance assessment of the proposed batch-based algorithm, we sort the VNF-FG requests in the batch queue $A(W_b)$ according to the gains generated (foreseen) by their successful placement. The acceptance gain $\mathbb{G}(\mathcal{R}eq)$, of a given request $\mathcal{R}eq$, is formally expressed as defined in [112] and [113]:

$$\mathbb{G}(\mathcal{R}eq) = \sum_{i \in N_v} (cpu_i \times U_{cpu}) + \sum_{e_{ij} \in E_v} \left(bw_{e_{ij}} \times U_{bw}\right) \tag{5.4.11}$$

where U_{cpu} is the realized gain from allocating one unit of CPU resources and U_{bw} is the earned profit from the allocation of one bandwidth unit.

Requests generating the largest gains are placed (served) first in order to maximize overall achieved profit and at the same time avoiding the rejection of all the requests at once when they are treated jointly in one shot. This placement strategy limits rejection rate while simultaneously aiming at maximizing the provider benefits. As long as the rejection rate is kept low, below 5% for example, user satisfaction and quality of experience should remain acceptable.

The batch algorithm, Algorithm 5, operates as follows: upon arrival each incoming request Req_i is stored in the batch list $A(W_b)$. At the expiration of the batch window W_b, the queued VNF-FG requests are sorted based on their (expected) gain and stored in a new list $S(W_b)$.

Algorithm 5: Batch algorithm

Inputs: NFV-I, Batch window W_b
Output: Mapping of the incoming requests during W_b
$A(W_b) \leftarrow \emptyset$
while *(W_b not expired)* **do**
    if *(arrival of new request Req_i)* **then**
        $A(W_b) \leftarrow A(W_b) \cup \{Req_i\}$
$S(W_b) \leftarrow sort(A(W_b))$
$M \leftarrow \emptyset$
while *($S(W_b)$ not empty)* **do**
    $\mathcal{R}eq \leftarrow extractTop(S(W_b))$
    $m \leftarrow$ R_ILP(NFV-I, $\mathcal{R}eq$)
    if *($m \neq null$)* **then**
        $M \leftarrow M \cup \{m\}$
retrun M

The ordered VNF-FG requests in $S(W_b)$ are processed sequentially by the online R_ILP algorithm. Successfully placed VNF-FG requests are stored in a list M along with their placement solution. Requests that cannot be placed are rejected. The placement process continues until all requests have been processed irrespective of the placement outcome in order to place as many requests as possible and, thus, reduce rejection rate. This is the simplest batch mode algorithm, called Batch R_ILP (BR_ILP) in our case.

5.4.4 EQL: Enhanced Q-Learning algorithm for VNF-FG Embedding

We propose an enhanced Q-Learning-based approach (EQL) for VNF-FG placement and chaining to improve long term performance and reward using reinforcement learning combined with an expert knowledge system. The other algorithms provide optimal solutions only for a given instant or over a rather short horizon and are actually quite suboptimal for longer term reward.

5.4.4.1 Preliminaries: Markov Decision Processes (MDP) and Reinforcement Learning (RL)

Markov Decision Processes (MDP) [122] give us a way to formalize sequential decision-making. The most important characteristic of an MDP is that the next state of the system only depends on the current state information and is unrelated to the earlier state. Unlike the Markov Process or Markov Chains, the MDP considers actions, that is, the next state of the system is related not only to the current state, but also to the current action taken. This mathematical formalization is the basis for structuring problems that are solved with Reinforcement Learning (RL) [123]. In fact, an MDP is used to describe an environment for RL, where the environment is fully observable.

The purpose of RL is to solve an MDP when the potentially visited states are unknown. One way to solve such a problem is to first learn the MDP and then solve it using algorithms such as value-iteration or policy-iteration (both using the Bellman equation [124]). The MDP can be learned by simulating different actions from each state until you have a high degree of confidence in the learned transition function and the learned reward function. Unfortunately, this is frequently not possible because the MDP is too large or it would be too expensive to learn the MDP.

Reinforcement Learning algorithms try to do both things at the same time: learn the MDP and solve it to find the optimal policy. To do that the algorithm needs to solve the exploration/exploitation tradeoff. Exploration means trying random actions (this helps discover the underlying MDP). Exploitation means following the so-far optimal policy (this helps maximize rewards). So RL is a technique to learn an MDP and solve it for the optimal policy at the same time. There are many algorithms related to reinforcement learning, such as Q-learning, State Action Reward State Action (SARSA), Deep Q-Network, Generative Adversarial Networks, etc. It has also been found that Q-Learning [125] is the most suitable for the problem studied in this chapter.

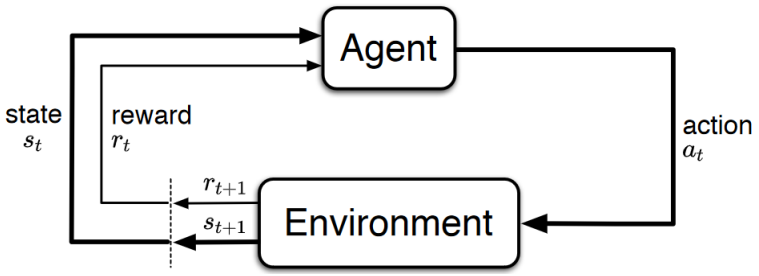


FIGURE 5.4: Operation algorithm of Q-Learning

As shown in Figure 5.4, in an MDP, we have a decision-maker, called an agent, that interacts with the environment it is placed in. These interactions occur sequentially over time. At each time step, the agent will get some representation of the environment state. Given this representation, the agent selects an action to take. The environment is then transitioned into a new state, and the agent is given a reward as a consequence of the previous action. This process of selecting an action from a given state, transitioning to a new state, and receiving a reward happens sequentially over and over again. Throughout this process, the agent's goal is to maximize the total amount of rewards that it receives from taking actions in given states. This means that the agent wants to maximize not just the immediate reward, but the cumulative rewards it receives over time.

5.4.4.2 Enhanced Q-Learning algorithm (EQL)

We optimize the Q-learning algorithm for the VNF-FG embedding problem. In this way, we can not only take advantage of the RL, but also avoid its defects, which can provide new ideas for dynamic SFC deployment. The Q-learning algorithm, shown in Equation (5.4.12), has a function that calculates the quality of a state-action combination:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}} \tag{5.4.12}$$

In Equation (5.4.12), Q is a matrix that stores the recommended values of the executable actions in the current state. Depending on these values, the agent can decide which action to take next. In the Q matrix, t refers to the unit of time prior, $t+1$: a unit of time post, max refers to the maximum value, s_t refers to the state space, which includes the resource requirements of VNFs and the remaining resources of the physical nodes, a_t to the action, s_{t+1} to the future state, and r_t is the reward value, which comes from the reward matrix R (r_t is the reward received when moving from the state s_t to the state s_{t+1}). The Q matrix is updated with reward r_t.

The core of the algorithm is a simple value iteration update, using the weighted average of the old value and the new information. The agent will repeat the above behavior until the Q matrix converges. Here, α and γ are the studied ratio between 0 and 1. The learning rate α determines to what extent newly acquired information overrides old information. A factor of 0 makes the agent learn nothing (exclusively exploiting prior knowledge), while a factor of 1 makes the agent consider only the most recent information (ignoring prior knowledge to explore possibilities). When the problem is stochastic, the algorithm converges under some technical conditions on the learning rate that require it to decrease to zero. In this chapter, a constant learning rate α is fixed to 0.1. The discount factor γ determines the importance of future rewards. A factor of 0 will make the agent short-sighted by only considering current rewards, while a factor approaching 1 will make it strive for a high long-term reward. In our implementation, we use a discount factor γ of 0.9.

In the reward matrix R, we set the value of an element equal to the residual capacity in each physical node (increasing or decreasing the reward r_t value according to the conditions, based on CPU, of each physical node). Through this, the load of nodes (in which each VNF operates) will be smartly distributed, so that each substrate node could exhibit the optimal performance through the efficient use of resources.

The proposed EQL algorithm operates in 5 steps:

1. compute the Q matrix using Equation (5.4.12) and learn the rules during the training process;

2. find a set of physical candidate nodes $\mathcal{C}$ related to a state s which are constituted by the method $getPolicyFromState$: that pick to move to the state that has the maximum Q value based on all possible actions; and the method $getExpertKnowledgeCandidates$: used to select in priority the physical nodes with highest available residual capacity to favor load balancing. This method accelerates the training process time, and also, avoids the random solutions at the beginning of the training process. Note that the node k available capacity and type constraints must be respected to be a candidate;

3. compute the shortest path between s and candidate k using Dijkstra's algorithm based on available bandwidth;

4. check link mapping: if links capacities are respected, confirm candidate k as a mapping solution for the VNF and add the solution path from s to k to the SolutionMapping $SolMap$ built so far;

5. update capacities on G_p, update R matrix (that will impact and update the Q matrix based on network conditions);

The pseudo-code of the EQL is shown in Algorithm 6.

5.5 Performance Evaluation

The algorithms are compared using extensive simulations (conducted on an experimental cloud and networking platform comprised of physical servers and networking technologies, similarly to our previous work in [107] and [110]) where both the request and hosting infrastructure are drawn using standard graph generation tools (such as the GT-ITM Tool [86]) and real infrastructure topologies (such as the Germany50 network topology [85]). The comparison includes the ILP, R_ILP, BR_ILP algorithms, and the enhanced Q-Learning algorithm (EQL). The performance of our proposed algorithms are also compared with the Eigendecomposition state of the art algorithm reported in [107] and a greedy solution [58] based on bipartite matching.

Algorithm 6: EQL algorithm pseudo-code

Inputs: G_p, G_v, $maxTrainCycles$, $maxCand$
Output: SolutionMapping $SolMap$
initialize the Q matrix with all zero elements
initialize the R matrix with the residual capacity in each physical node
$queue \leftarrow \emptyset$
A set of candidate hosts $\mathcal{C} \leftarrow \emptyset$
for $n = 0$ *to* $maxTrainCycles$ **do**
    Compute Q matrix using Equation (5.4.12)
    if *the Q matrix has basically converged* **then**
        Break; //return the Q matrix that can be used
foreach *virtual link* $e_{ij} \in E_v$ **do**
    $s \leftarrow$ SelectInitialState(to host vnf_i or vnf_j)
    repeat
        $\mathcal{C} \leftarrow \mathcal{C} \cup$ getPolicyFromState(s) $\cup$ getExpertKnowledgeCandidates(s)
    until *size*($\mathcal{C}$) = $maxCand$;
    $stack \leftarrow$ Sort($\mathcal{C}$)
    while *stack* $\neq \emptyset$ *and* $solution$ = *False* **do**
        $k \leftarrow$ Pop($stack$)
        if *ComputeLinkMapping*(s, k) = *True and CheckLinks*(s, k) = *True* **then**
            $solution \leftarrow$ True
            $SolMap \leftarrow SolMap$ + GetPathSolution(s, k)
            Update capacities on G_p
            Update R matrix
return $SolMap$

5.5.1 Simulation Environment

All simulations for all the algorithms run in a dedicated server in the experimental platform with the following processing capabilities: a 2.50 GHz, Quad Core server with 6 GBytes of available RAM. The VNF-FG requests are generated using a Poisson process with an average arrival rate λ of 5 requests per 100 time units. The lifetime of each request follows an exponential distribution with a mean of 1000 time units. The Germany50 network topology [85] is used for the first assessment (with 50 nodes). The capacity of physical nodes and links are generated randomly in the $[100, 150]$ interval. The size of the VNF-FG requests is arbitrarily set to 5 nodes. The GT-ITM [86] tool is used to generate the requested VNF-FG topologies. The VNF-FG computing and bandwidth requirements are set to 10 resource units. The connectivity between nodes in the VNF-FG is set to 0.3 (30%).

The performance assessment at larger-scale uses the GT-ITM tool to generate network topologies with NFV-I sizes in the range $[100, 500]$ nodes and a connectivity of 0.3 (or 30%). The physical resource capacities (i.e., CPU and bandwidth) are also drawn randomly in the $[100, 150]$ interval. The VNF-FG requests are generated using a Poisson process with an average arrival rate of 5 requests per 100 time units. The lifetime of each request follows an exponential distribution with a mean of 1000 time units. The size of the VNF-FG requests is also set to 5 nodes. The required CPU for each VNF in the VNF-FG is set to 10 units. The required bandwidth between two VNFs, to ensure communication between them, is set also to 10 units. The connectivity between nodes in the VNF-FG is set to 0.3 (30%).

For the realistic topology and the large-scale evaluations, 1000 VNF-FG requests are generated. The ILP solver used in our experiments is Cplex [111].

5.5.2 Performance Metrics

The metrics used for the performance evaluation are typical indicators for placement algorithms and as defined below:

- **Acceptance ratio** is the ratio of incoming service requests that have been successfully deployed on the network to all incoming requests. Maximizing this quantity is equivalent to minimizing rejection rate of the requests.
- **Acceptance gain** is the service provider realized profit at time t when accepting client requests (in our case successful placement of VNF-FG requests). The acceptance gain is formally expressed as:

$$\mathbb{G}(t) = \sum_{\mathcal{R}eq \in \mathcal{AR}_t} \mathbb{G}\left(\mathcal{R}eq\right) \tag{5.5.13}$$

  where $\mathcal{AR}_t$ is the set of accepted requests up to time t and $\mathbb{G}(\mathcal{R}eq)$ is the request $\mathcal{R}eq$ gain as defined in Eq. (5.4.11). Note that $\mathbb{G}$ is the total profit taking into consideration all the incoming requests.

- **Execution time** is the time needed to find an embedding solution for one VNF-FG request. This metric reflects the ability of the algorithms to scale with problem size and this is especially important for the ILP algorithm assessment.

5.5.3 Simulation Results

The performance of the algorithms is reported for the Germany50 network topology for small scale problems (50 nodes) and for randomly generated network topologies for larger problems involving hundreds of nodes and links.

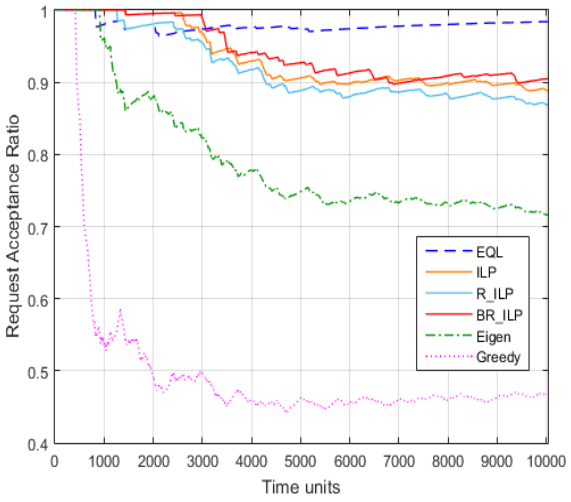


FIGURE 5.5: Acceptance ratio

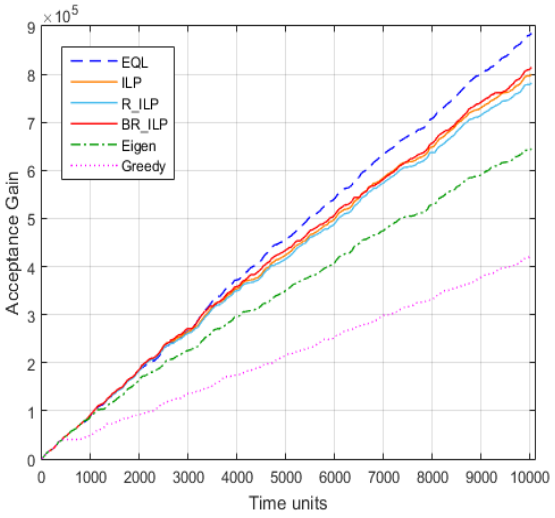


FIGURE 5.6: Acceptance gain

5.5.3.1 Realistic topology evaluation

For the Germany50 network, since the ILP can provide placement solutions in reasonable execution times it can be included in the performance comparisons with the R_ILP, BR_ILP, EQL, Eigen [107], and Greedy [58]. The results provide insight on these five algorithms compared to the optimal solutions provided by the ILP for such small problem sizes. The performance is pushed further for much larger networks for the heuristic algorithms only since they scale much better with problem size. As mentioned, the larger graphs are randomly generated with the GT-ITM Tool.

Figure 5.5 depicts the results in terms of acceptance ratio. The EQL algorithm outperforms the ILP based solutions (i.e., ILP, R_ILP, BR_ILP) by accepting more requests in long-term. The normalized metric (acceptance ratio) confirms these results passed a transient state (passed 3000 time units) where the achieved ratios are around 98.4% for the EQL, 90.5% for the BR_ILP, 88.9% for the ILP, 86.9% for the R_ILP, 71.7% for the Eigendecomposition-based algorithm, and only 46.8% for the Greedy algorithm. The Eigendecomposition approach is known to provide much better results for highly connected topology compared with weakly connected graphs.

The computed acceptance gain depicted in Figure 5.6, that corresponds to the total generated profit at time t, confirm the relative performance of the algorithms observed for the acceptance ratio. The EQL achieved provider gain is 8.02% higher than the BR_ILP, and respectively 9.65%, 11.68%, 27.13%, and 52.43% better than ILP, R_ILP, Eigen, and Greedy.

Figures 5.7 compares the quality realized by the proposed algorithms in terms of objective function, achieved maximum in Equation (5.4.1), to provide insight on the relative performance of the heuristics to the optimal solutions found by the ILP. Since we aim in this chapter at maximizing provider profit, we define the mapping quality based on the value of the objective function Z that describes the algorithms ability to achieve load balancing on the hosting infrastructure. Figure 5.7 reports the achieved Z values and indicates clearly that the closest in performance to the ILP is the enhanced Q-Learning algorithm (EQL). The batch ILP algorithm (BR_ILP) is outperformed by the EQL in terms of quality of the solutions especially for the long-term evaluation.

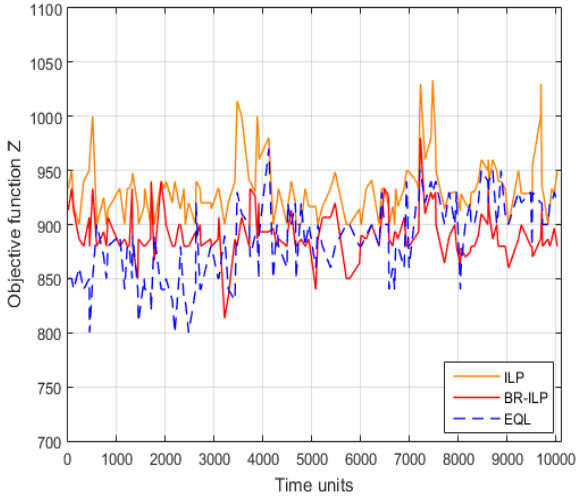


FIGURE 5.7: Quality of the mapping

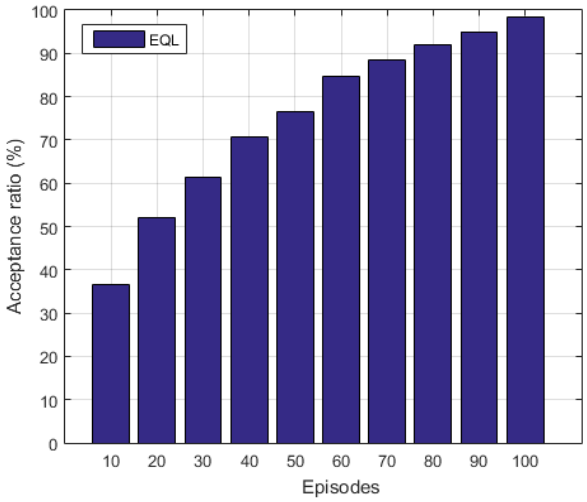


FIGURE 5.8: Percentage of deployed VNF-FGs

Figure 5.8 reports the average acceptance percentage of the enhanced Q-Learning (EQL) algorithm as a function of increasing the number of episodes (from 10 to 100). We observe that the average acceptance percentage under a high number of episodes exhibits much better performance than under a low number of episodes (98.4% for 100 episodes versus 36.5% for 10 episodes).

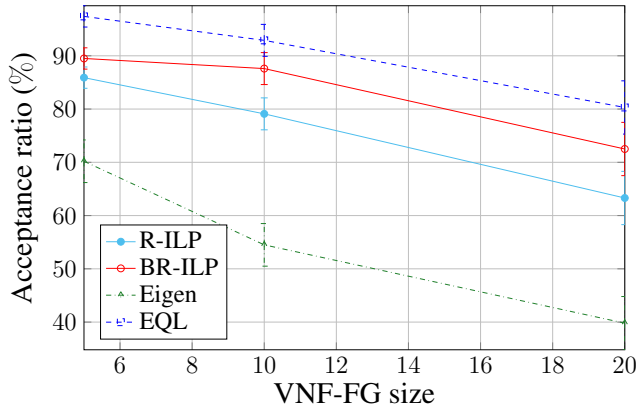


FIGURE 5.9: Acceptance percentage w.r.t VNF-FG size variation

5.5.3.2 Large-scale evaluation

To analyze the behavior of the algorithms for larger problem sizes, NFV-Is with 200 nodes are generated and used in this second performance assessment. Figure 5.9 reports the average acceptance percentage of the algorithms as a function of increasing VNF-FG request sizes (from 5 to 20) and confirms the previous findings. The EQL algorithm outperforms the ILP based solutions (i.e., R_ILP, BR_ILP) and the Eigen approach by accepting more requests. The results in Figure 5.9, presented with a 95% confidence interval, depict a high similarity with results in Figure 5.5 (more precisely for VNF-FG size of 5 nodes).

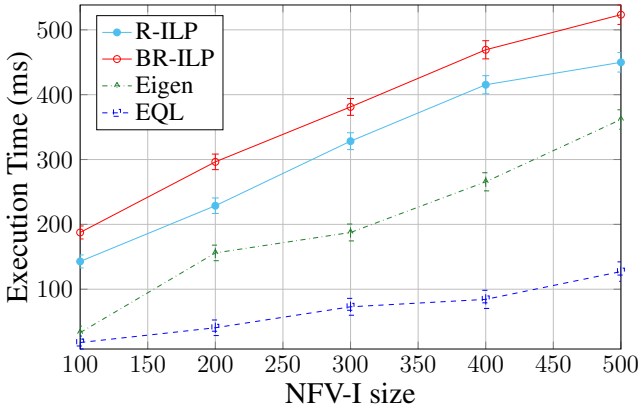


FIGURE 5.10: Execution Time w.r.t NFV-I size variation

5.5.3.3 Execution time (Convergence time)

Figure 5.10 reports the execution time performance of the algorithms with problem size to gain insight on their scalability and complexity. We generate 1000 VNF-FG requests and vary the substrate graph sizes from 100 to 500 nodes. The EQL based algorithm has significantly better execution time compared with the R_ILP, BR_ILP, and Eigen execution times.

5.6 Conclusion

In this chapter, we propose approaches for the VNF-FG placement and chaining problem in an online and a batch mode. The online strategy exploits an Integer Linear Program derived from a mathematical model of the VNF-FG embedding problem. The ILP model is derived and used to find optimal placement solutions to host the requests when problem size is small. To control complexity and improve scalability for larger graph sizes, the ILP explores only a reduced subset of candidate hosts (R_ILP). The introduced batch mode algorithm, to enhance further overall performance, uses our R_ILP algorithm to process a group of requests, queued in a batch window, by serving in priority requests that generate higher profit for the providers while aiming at maximizing the number of served (i.e., placed, chained, hosted) requests. An Enhanced Q-Learning-Based Approach (EQL) is also proposed and compared with the ILP solutions. The EQL algorithm improves the efficiency of addressing this specific type of problem (it not only capitalizes on the decision-making advantages of RL, but also avoids a lengthy training process by injecting expert knowledge). The experimental results show that the performance of the proposed EQL algorithm is superior to that of the ILP solutions, the greedy, and the eigendecomposition algorithms when processing service requests in long-term. In fact, its performance advantages are reflected by the decision time, quality of mapping, deployment success rate and deployment profit when deploying SFCs.

Chapter 6

Conclusion and Future Research Directions

This chapter summarizes the research work and contributions of the thesis on Virtualized Network Function Forwarding graph embedding. The chapter then outlines some perspectives for future investigations.

6.1 Conclusion and Discussion

In this thesis, we addressed the VNF-FG embedding problem, with a focus on the virtual and substrate resource management. In fact, we separate the VNF-FG embedding issue into two sub-problems: the initial VNF-FG placement (that aims at finding an optimal mapping of virtual resources on physical infrastructures), and the dynamic management of resources (that deals with resource demand fluctuation of embedded virtual networks, and the re-optimization of the substrate network usage). The key contributions of the thesis are listed below:

- An exact algorithm for VNF scaling formulated and solved using linear integer programming (ILP) and a new Greedy reconfiguration algorithm is also reported for the problem of VNF reconfiguration (scaling or migration). These algorithms deal

with virtual nodes demand fluctuations (the rising of network load over time). They manage the case where an embedded VNF-FG requires more resources, whereas the hosting substrate node does not have enough available resources. In addition to reducing the rejection rate of VNF scaling requests, our proposals take into account the service interruption during migration and reduce it.

- The second major contribution of the dissertation addresses the problem of VNF-FG extension with the main objective of maximizing the provider revenue by reducing the rejection of extension requests when faced with a dynamic demand. An ILP model is presented as an exact algorithm acting as a baseline for comparison and as the solution for small problem sizes. To improve scalability, two heuristic algorithms are also proposed: a new Steiner Tree-based algorithm (STVE) and an Eigendecomposition-based approach (EDVE).

- Then, we propose approaches for the VNF-FG placement and chaining problem in an online and a batch mode. The online strategy exploits an ILP derived from a mathematical model of the VNF-FG embedding problem. The ILP model is derived and used to find optimal placement solutions to host the requests when the problem size is small. To control complexity and improve scalability for larger graph sizes, the ILP explores only a reduced subset of candidate hosts (R_ILP). A batch mode algorithm (BR_ILP), to enhance further overall performance, uses the R_ILP algorithm to process a group of requests, queued in a batch window, by serving in priority requests that generate higher revenues for the providers while aiming at maximizing the number of served requests. An Enhanced Q-Learning-Based Approach (EQL) is also proposed and compared with the ILP solutions. The EQL algorithm improves the efficiency of addressing this specific type of problems. It not only capitalizes on the decision-making advantages of RL, but also avoids a lengthy training process by injecting expert knowledge.

6.2 Future Research Directions

Suggested future research work resulting from this thesis can be summarized as follows:

- Enhance the previous contributions by making more exhaustive simulations on the proposed algorithms to better evaluate their performance and understand their limits in different conditions.

- Expand the contribution of Chapter 5 to manage the VNF scaling problem. In fact, reinforcement learning can be used to control the scaling decisions in the management and orchestration of network function virtualization.

- Explore other VNF-FG embedding problems, such as the resilient allocation of Service Function Chain problem (consider the possibility of failures in the NFV infrastructure) [126], [127], and [128].

- Combine placement with traffic predictions to anticipate future demands and to check how long term reward and stability could be realized.

Appendix A

Thesis Publications

International Journal

- Omar Houidi, Oussama Soualah, Wajdi Louati, and Djamal Zeghlache, "Dynamic VNF Forwarding Graph Extension Algorithms," IEEE Transactions on Network and Service Management, 2020.

International Conferences

- Omar Houidi, Oussama Soualah, Wajdi Louati, Marouen Mechtri, Djamal Zeghlache, and Farouk Kamoun, "An Efficient Algorithm for Virtual Network Function Scaling," in 2017 IEEE Global Communications Conference, GLOBECOM 2017, Singapore, December 4-8, 2017, pages 1-7, 2017.
- Omar Houidi, Oussama Soualah, Wajdi Louati, Djamal Zeghlache, and Farouk Kamoun, "Virtualized Network Services Extension Algorithms," in 17th IEEE International Symposium on Network Computing and Applications, NCA 2018, Cambridge, MA, USA, November 1-3, 2018, pages 1-5, 2018.

- Omar Houidi, Oussama Soualah, Wajdi Louati, and Djamal Zeghlache, "An Enhanced Reinforcement Learning Approach for Dynamic Placement of Virtual Network Functions," in 31st IEEE Annual International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2020, London, UK, 31 August-3 September, 2020, pages 1-7, 2020.

Bibliography

[1] Juliver Gil-Herrera and Juan Felipe Botero. Resource allocation in NFV: A comprehensive survey. *IEEE Trans. Network and Service Management*, 13(3):518–532, 2016.

[2] ETSI GS NFV 002: Network Functions Virtualisation (NFV); Architectural Framework. 2014.

[3] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys and Tutorials*, 18(1):236–262, 2016.

[4] Edoardo Amaldi, Stefano Coniglio, Arie M. C. A. Koster, and Martin Tieves. On the computational complexity of the virtual network embedding problem. *Electronic Notes in Discrete Mathematics*, 52:213–220, 2016.

[5] Omar Houidi, Oussama Soualah, Wajdi Louati, Marouen Mechtri, Djamal Zeghlache, and Farouk Kamoun. An Efficient Algorithm for Virtual Network Function Scaling. In *2017 IEEE Global Communications Conference, GLOBECOM 2017, Singapore, December 4-8, 2017*, pages 1–7, 2017.

[6] Omar Houidi, Oussama Soualah, Wajdi Louati, Djamal Zeghlache, and Farouk Kamoun. Virtualized Network Services Extension Algorithms. In *17th IEEE International Symposium on Network Computing and Applications, NCA 2018, Cambridge, MA, USA, November 1-3, 2018*, pages 1–5, 2018.

[7] Omar Houidi, Oussama Soualah, Wajdi Louati, and Djamal Zeghlache. Dynamic VNF Forwarding Graph Extension Algorithms. *IEEE Transactions on Network and Service Management*, 2020.

[8] Omar Houidi, Oussama Soualah, Wajdi Louati, and Djamal Zeghlache. An Enhanced Reinforcement Learning Approach for Dynamic Placement of Virtual Network Functions. In *31st IEEE Annual International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2020*, pages 1–7, 2020.

[9] ETSI, Network Functions Virtualisation - Introductory White Paper. In *SDN and OpenFlow World Congress, Darmstadt, Germany, October 22-24, 2012*, pages 1–16, 2012. URL https://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[10] Bo Yi, Xingwei Wang, Keqin Li, Sajal K. Das, and Min Huang. A comprehensive survey of network function virtualization. *Computer Networks*, 133:212–262, 2018.

[11] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[12] Fei Hu, Qi Hao, and Ke Bao. A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys and Tutorials*, 16 (4):2181–2206, 2014.

[13] Diego Kreutz, Fernando M. V. Ramos, Paulo Jorge Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[14] Nick McKeown, Thomas E. Anderson, Hari Balakrishnan, Guru M. Parulkar, Larry L. Peterson, Jennifer Rexford, Scott Shenker, and Jonathan S. Turner. Openflow: enabling innovation in campus networks. *Computer Communication Review*, 38(2):69–74, 2008.

[15] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network innovation using openflow: A survey. *IEEE Communications Surveys and Tutorials*, 16(1):493–512, 2014.

[16] Hyojoon Kim and Nick Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.

[17] Bruno Astuto A. Nunes, Marc Mendonca, Xuan Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys and Tutorials*, 16(3): 1617–1634, 2014.

[18] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. A survey on software-defined networking. *IEEE Communications Surveys and Tutorials*, 17(1):27–51, 2015.

[19] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, and Taieb Znati. Network function virtualization in 5g. *IEEE Communications Magazine*, 54(4):84–91, 2016.

[20] Michel S. Bonfim, Kelvin Lopes Dias, and Stenio F. L. Fernandes. Integrated NFV/SDN architectures: A systematic literature review. *CoRR*, abs/1801.01516, 2018.

[21] Van-Giang Nguyen, Anna Brunström, Karl-Johan Grinnemo, and Javid Taheri. Sdn/nfv-based mobile packet core network architectures: A survey. *IEEE Communications Surveys and Tutorials*, 19(3):1567–1602, 2017.

[22] ETSI. Network Functions Virtualisation (NFV). URL https://www.etsi.org/technologies/nfv.

[23] ONF. Software Defined Standards. URL https://www.opennetworking.org/software-defined-standards/overview/.

[24] DMTF. Standards and Technology. URL https://www.dmtf.org/standards.

[25] Yong Li and Min Chen. Software-defined network function virtualization: A survey. *IEEE Access*, 3:2542–2553, 2015.

[26] Paul Quinn and Thomas D. Nadeau. Problem statement for service function chaining. *RFC*, 7498:1–13, 2015. URL http://www.rfc-editor.org/rfc/rfc7498.txt.

[27] Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *J. Network and Computer Applications*, 75:138–155, 2016.

[28] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann de Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys and Tutorials*, 15(4):1888–1906, 2013.

[29] Abdelquoddouss Laghrissi and Tarik Taleb. A survey on the placement of virtual resources and virtual network functions. *IEEE Communications Surveys and Tutorials*, 21(2):1409–1434, 2019.

[30] Riccardo Guerzoni, Riccardo Trivisonno, Ishan Vaishnavi, Zoran Despotovic, Artur Hecker, Sergio Beker, and David Soldani. A novel approach to virtual networks embedding for SDN management and orchestration. In *2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014*, pages 1–7, 2014.

[31] Hendrik Moens and Filip De Turck. VNF-P: A model for efficient placement of virtualized network functions. In *10th International Conference on Network and Service Management, CNSM 2014 and Workshop, Rio de Janeiro, Brazil, November 17-21, 2014*, pages 418–423, 2014.

[32] Sevil Mehraghdam, Matthias Keller, and Holger Karl. Specifying and placing chains of virtual network functions. In *3rd IEEE International Conference on Cloud Networking, CloudNet 2014, Luxembourg, Luxembourg, October 8-10, 2014*, pages 7–13, 2014.

[33] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Trans. Netw.*, 20(1):206–219, 2012.

[34] Nagao Ogino, Takeshi Kitahara, Shin'ichi Arakawa, and Masayuki Murata. Virtual network embedding with multiple priority classes sharing substrate resources. *Computer Networks*, 112:52–66, 2017.

[35] Yang Wang, Qian Hu, and Xiaojun Cao. A branch-and-price framework for optimal virtual network embedding. *Computer Networks*, 94:318–326, 2016.

[36] Oussama Soualah, Ilhem Fajjari, Makhlouf Hadji, Nadjib Aitsaadi, and Djamal Zeghlache. A novel virtual network embedding scheme based on gomory-hu tree within cloud's backbone. In *2016 IEEE/IFIP Network Operations and Management Symposium, NOMS 2016, Istanbul, Turkey, April 25-29, 2016*, pages 536–542, 2016.

[37] Ines Houidi, Wajdi Louati, and Djamal Zeghlache. A distributed virtual network mapping algorithm. In *Proceedings of IEEE International Conference on Communications, ICC 2008, Beijing, China, 19-23 May 2008*, pages 5634–5640, 2008.

[38] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *Computer Communication Review*, 38(2):17–29, 2008.

[39] Ilhem Fajjari, Nadjib Aitsaadi, Guy Pujolle, and Hubert Zimmermann. VNR algorithm: A greedy approach for virtual networks reconfigurations. In *Proceedings of the Global Communications Conference, GLOBECOM 2011, 5-9 December 2011, Houston, Texas, USA*, pages 1–6, 2011.

[40] Ilhem Fajjari, Nadjib Aitsaadi, Boutheina Dab, and Guy Pujolle. Novel adaptive virtual network embedding algorithm for cloud's private backbone network. *Computer Communications*, 84:12–24, 2016.

[41] Yong Zhu and Mostafa H. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 23-29 April 2006, Barcelona, Catalunya, Spain*, 2006.

[42] Ines Houidi, Wajdi Louati, Djamal Zeghlache, Panagiotis Papadimitriou, and Laurent Mathy. Adaptive virtual network provisioning. In *ACM SIGCOMM Workshops Proceedings*, 2010.

[43] Ilhem Fajjari, Nadjib Aitsaadi, Guy Pujolle, and Hubert Zimmermann. Adaptive-vne: A flexible resource allocation for virtual network embedding algorithm. In *2012 IEEE Global Communications Conference, GLOBECOM 2012, Anaheim, CA, USA, December 3-7, 2012*, pages 2640–2646, 2012.

[44] Koki Inoue, Shin'ichi Arakawa, Satoshi Imai, Toru Katagiri, and Masayuki Murata. Adaptive VNE method based on yuragi principle for software defined infrastructure. In *17th IEEE International Conference on High Performance Switching and Routing, HPSR 2016, Yokohama, Japan, June 14-17, 2016*, pages 188–193, 2016.

[45] Nashid Shahriar, Reaz Ahmed, Shihabur Rahman Chowdhury, Md Mashrur Alam Khan, Raouf Boutaba, Jeebak Mitra, and Feng Zeng. Virtual network embedding with guaranteed connectivity under multiple substrate link failures. *IEEE Transactions on Communications*, 2019.

[46] Aaron Gember, Anand Krishnamurthy, Saul St. John, Robert Grandl, Xiaoyang Gao, Ashok Anand, Theophilus Benson, Aditya Akella, and Vyas Sekar. Stratos: A network-aware orchestration layer for middleboxes in the cloud. *CoRR*, abs/1305.0209, 2013.

[47] Md. Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, and Raouf Boutaba. On orchestrating virtual network functions. In *11th International Conference on Network and Service Management, CNSM 2015, Barcelona, Spain, November 9-13, 2015*, pages 50–56, 2015.

[48] Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Salete Buriol, Marinho P. Barcellos, and Luciano Paschoal Gaspary. Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions. In *IFIP/IEEE International Symposium on Integrated Network Management, IM 2015, Ottawa, ON, Canada, 11-15 May, 2015*, pages 98–106, 2015.

[49] Barbara Martini, Federica Paganelli, Paola Cappanera, Stefano Turchi, and Piero Castoldi. Latency-aware composition of virtual functions in 5g. In *Proceedings of the 1st IEEE Conference on Network Softwarization, NetSoft 2015, London, United Kingdom, April 13-17, 2015*, pages 1–6, 2015.

[50] Roberto Riggio, Abbas Bradai, Tinku Rasheed, Julius Schulz-Zander, Slawomir Kuklinski, and Toufik Ahmed. Virtual network functions orchestration in wireless networks. In *11th International Conference on Network and Service Management, CNSM 2015, Barcelona, Spain, November 9-13, 2015*, pages 108–116, 2015.

[51] Abhishek Gupta, M Farhan Habib, Pulak Chowdhury, Massimo Tornatore, and Biswanath Mukherjee. Joint virtual network function placement and routing of traffic in operator networks. *UC Davis, Davis, CA, USA, Tech. Rep*, 2015.

[52] Abhishek Gupta, Brigitte Jaumard, Massimo Tornatore, and Biswanath Mukherjee. Service chain (SC) mapping with multiple SC instances in a wide area network. In *2017 IEEE Global Communications Conference, GLOBECOM 2017, Singapore, December 4-8, 2017*, pages 1–6, 2017.

[53] Abhishek Gupta, Brigitte Jaumard, Massimo Tornatore, and Biswanath Mukherjee. A scalable approach for service chain mapping with multiple SC instances in a wide-area network. *IEEE Journal on Selected Areas in Communications*, 36(3):529–541, 2018.

[54] Marco Savi, Massimo Tornatore, and Giacomo Verticale. Impact of processing costs on service chain placement in network functions virtualization. In *IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2015, San Francisco, CA, USA, November 18-21, 2015*, pages 191–197, 2015.

[55] Md. Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, Raouf Boutaba, and Otto Carlos Muniz Bandeira Duarte. Orchestrating virtualized network functions. *IEEE Trans. Network and Service Management*, 13(4):725–739, 2016.

[56] Ali Mohammadkhan, Sheida Ghapani, Guyue Liu, Wei Zhang, K. K. Ramakrishnan, and Timothy Wood. Virtual function placement and traffic steering in flexible and dynamic software defined networks. In *2015 IEEE International Workshop on Local and Metropolitan Area Networks, LANMAN 2015, Beijing, China, April 22-24, 2015*, pages 1–6, 2015.

[57] Bernardetta Addis, Dallal Belabed, Mathieu Bouet, and Stefano Secci. Virtual network functions placement and routing optimization. In *4th IEEE International Conference on Cloud Networking, CloudNet 2015, Niagara Falls, ON, Canada, October 5-7, 2015*, pages 171–177, 2015.

[58] Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. VNF placement and chaining in distributed cloud. In *9th IEEE International Conference on Cloud Computing, CLOUD 2016, San Francisco, CA, USA, June 27 - July 2, 2016*, pages 376–383, 2016.

[59] Xin Li and Chen Qian. The virtual network function placement problem. In *2015 IEEE Conference on Computer Communications Workshops, INFOCOM Workshops, Hong Kong, China, April 26 - May 1, 2015*, pages 69–70, 2015.

[60] Selma Khebbache, Makhlouf Hadji, and Djamal Zeghlache. Virtualized network functions chaining and routing algorithms. *Computer Networks*, 114:95–110, 2017.

[61] Salvatore D'Oro, Laura Galluccio, Sergio Palazzo, and Giovanni Schembra. Exploiting congestion games to achieve distributed service chaining in NFV networks. *IEEE Journal on Selected Areas in Communications*, 35(2):407–420, 2017.

[62] Rami Cohen, Liane Lewin-Eytan, Joseph Naor, and Danny Raz. Near optimal placement of virtual network functions. In *2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26 - May 1, 2015*, pages 1346–1354, 2015.

[63] Yu Sang, Bo Ji, Gagan Raj Gupta, Xiaojiang Du, and Lin Ye. Provably efficient algorithms for joint placement and allocation of virtual network functions. In *2017 IEEE Conference on Computer Communications, INFOCOM 2017, Atlanta, GA, USA, May 1-4, 2017*, pages 1–9, 2017.

[64] Andrea Tomassilli, Frédéric Giroire, Nicolas Huin, and Stéphane Pérennes. Provably efficient algorithms for placement of service function chains with ordering constraints. In *2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018*, pages 774–782, 2018.

[65] Tarik Taleb, Miloud Bagaa, and Adlen Ksentini. User mobility-aware virtual network function placement for virtual 5g network infrastructure. In *2015 IEEE International Conference on Communications, ICC 2015, London, United Kingdom, June 8-12, 2015*, pages 3879–3884, 2015.

[66] Yanghao Xie, Zhixiang Liu, Sheng Wang, and Yuxiu Wang. Service function chaining resource allocation: A survey. *CoRR*, abs/1608.00095, 2016.

[67] Milad Ghaznavi, Aimal Khan, Nashid Shahriar, Khalid Alsubhi, Reaz Ahmed, and Raouf Boutaba. Elastic virtual network function placement. In *IEEE CloudNet*, pages 255–260, 2015.

[68] Fangxin Wang, Ruilin Ling, Jing Zhu, and Dan Li. Bandwidth guaranteed virtual network function placement and scaling in datacenter networks. In *34th IEEE International Performance Computing and Communications Conference, IPCCC 2015, Nanjing, China, December 14-16, 2015*, pages 1–8, 2015.

[69] Irian Leyva Pupo, Cristina Cervelló Pastor, and Alejandro Llorens Carrodeguas. A framework for placement and optimization of network functions in 5g. 2018.

[70] Adel Nadjaran Toosi, Jungmin Son, Qinghua Chi, and Rajkumar Buyya. Elasticsfc: Auto-scaling techniques for elastic service function chaining in network functions virtualization-based clouds. *Journal of Systems and Software*, 152:108–119, 2019.

[71] Yunjie Gu, Yuxiang Hu, Yuehang Ding, Jie Lu, and Jichao Xie. Elastic virtual network function orchestration policy based on workload prediction. *IEEE Access*, 7: 96868–96878, 2019.

[72] Ziyue Luo, Chuan Wu, Zongpeng Li, and Wei Zhou. Scaling geo-distributed network function chains: A prediction and learning framework. *IEEE Journal on Selected Areas in Communications*, 37(8):1838–1850, 2019.

[73] Jianing Pei, Peilin Hong, Kaiping Xue, and Defang Li. Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system. *IEEE Trans. Parallel Distrib. Syst.*, 30(10):2179–2192, 2019.

[74] Windhya Rankothge, Helena Ramalhinho, and Jorge Lobo. On the scaling of virtualized network functions. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 125–133. IEEE, 2019.

[75] Jiefei Ma, Windhya Rankothge, Christian Makaya, Mariceli Morales, Franck Le, and Jorge Lobo. An overview of A load balancer architecture for VNF chains horizontal

scaling. In *14th International Conference on Network and Service Management, CNSM 2018, Rome, Italy, November 5-9, 2018*, pages 323–327, 2018.

[76] Racha Gouareb, Vasilis Friderikos, and A. Hamid Aghvami. Placement and routing of vnfs for horizontal scaling. In *26th International Conference on Telecommunications, ICT 2019, Hanoi, Vietnam, April 8-10, 2019*, pages 154–159, 2019.

[77] Ruoyun Chen, Hancheng Lu, Yujiao Lu, and Jinxue Liu. Msdf: A deep reinforcement learning framework for service function chain migration. *arXiv preprint arXiv:1911.04801*, 2019.

[78] Lun Tang, Xiaoyu He, Peipei Zhao, Guofan Zhao, Yu Zhou, and Qianbin Chen. Virtual network function migration based on dynamic resource requirements prediction. *IEEE Access*, 7:112348–112362, 2019.

[79] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. Adaptive VNF scaling and flow routing with proactive demand prediction. In *2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018*, pages 486–494, 2018.

[80] ETSI GS NFV-TST 001: Network Functions Virtualisation (NFV); Pre-deployment Testing; Report on Validation of NFV Environments and Services. 2016.

[81] Guilherme Galante and Luis Carlos Erpen De Bona. A survey on cloud computing elasticity. In *IEEE Fifth International Conference on Utility and Cloud Computing, UCC 2012, Chicago, IL, USA, November 5-8, 2012*, pages 263–270, 2012.

[82] Jing Xia, Zhiping Cai, and Ming Xu. Optimized virtual network functions migration for NFV. In *22nd IEEE International Conference on Parallel and Distributed Systems, ICPADS 2016, Wuhan, China, December 13-16, 2016*, pages 340–346, 2016.

[83] ETSI GS NFV 001: Network Functions Virtualisation (NFV); Use Cases. 2013.

[84] ETSI GS NFV 003: Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV. 2014.

[85] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. Sndlib 1.0 - survivable network design library. *Networks*, 55(3):276–286, 2010.

[86] Ellen W. Zegura, Kenneth L. Calvert, and Samrat Bhattacharjee. How to model an internetwork. In *Proceedings IEEE INFOCOM, San Francisco, CA, USA, March 24-28, 1996*, pages 594–602, 1996.

[87] Róbert Szabó, Mario Kind, Fritz-Joachim Westphal, Hagen Woesner, David Jocha, and András Császár. Elastic network functions: opportunities and challenges. *IEEE Network*, 29(3):15–21, 2015.

[88] Sevil Dräxler, Holger Karl, and Zoltán Ádám Mann. JASPER: joint optimization of scaling, placement, and routing of virtual network services. *IEEE Trans. Network and Service Management*, 15(3):946–960, 2018.

[89] Sara Ayoubi, Yanhong Zhang, and Chadi Assi. A reliable embedding framework for elastic virtualized services in the cloud. *IEEE Trans. Network and Service Management*, 13(3):489–503, 2016.

[90] Junjie Liu, Wei Lu, Fen Zhou, Ping Lu, and Zuqing Zhu. On dynamic service function chain deployment and readjustment. *IEEE Trans. Network and Service Management*, 14(3):543–553, 2017.

[91] Windhya Rankothge, Franck Le, Alessandra Russo, and Jorge Lobo. Optimizing resource allocation for virtualized network functions in a cloud center using genetic algorithms. *IEEE Trans. Network and Service Management*, 14(2):343–356, 2017.

[92] Yang Li, Linh Thi Xuan Phan, and Boon Thau Loo. Network functions virtualization with soft real-time guarantees. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*, pages 1–9, 2016.

[93] Mu He, Alberto Martínez Alba, Arsany Basta, Andreas Blenk, and Wolfgang Kellerer. Flexibility in softwarized networks: Classifications and research challenges. *IEEE Communications Surveys and Tutorials*, 2019.

[94] Surendra Kumar, Mudassir Tufail, Sumandra Majee, Claudiu Captari, and Shunsuke Homma. Service function chaining use cases in data centers.

Internet-draft, February 2017. URL https://tools.ietf.org/html/draft-ietf-sfc-dc-use-cases-06.

[95] ETSI GS NFV-MAN 001: Network Functions Virtualisation (NFV); Management and Orchestration. 2014.

[96] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[97] Konstantinos Kalpakis, Koustuv Dasgupta, and Ouri Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):628–637, 2001.

[98] Philip N. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *J. Algorithms*, 19(1):104–115, 1995.

[99] Rashid Bin Muhammad. Range assignment problem on the steiner tree based topology in ad hoc wireless networks. *Mobile Information Systems*, 5(1):53–64, 2009.

[100] S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1(3): 195–207, 1971.

[101] Pawel Winter. Steiner problem in networks: A survey. *Networks*, 17(2):129–167, 1987.

[102] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[103] Piotr Berman and Viswanathan Ramaiyer. Improved approximations for the steiner tree problem. *J. Algorithms*, 17(3):381–408, 1994.

[104] Marek Karpinski and Alexander Zelikovsky. New approximation algorithms for the steiner tree problems. *J. Comb. Optim.*, 1(1):47–65, 1997.

[105] Pawel Winter and J. MacGregor Smith. Path-distance heuristics for the steiner problem in undirected networks. *Algorithmica*, 7(2&3):309–327, 1992.

[106] H. Takahashi and A. Matsuyama. An approximate solution for the steiner problem in graphs. *Math. Japonica*, 24(6):573–577, 1980.

[107] Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. A scalable algorithm for the placement of service function chains. *IEEE Trans. Network and Service Management*, 13(3):533–546, 2016.

[108] Shinji Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(5):695–703, 1988.

[109] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[110] Marouen Mechtri, Chaima Ghribi, Oussama Soualah, and Djamal Zeghlache. NFV orchestration framework addressing SFC challenges. *IEEE Communications Magazine*, 55(6):16–23, 2017.

[111] IBM ILOG CPLEX Optimization Studio. URL https://www.ibm.com/products/ilog-cplex-optimization-studio/.

[112] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *Computer Communication Review*, 38(2):17–29, 2008.

[113] N. M. Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*, pages 783–791, 2009.

[114] Soroush Haeri and Ljiljana Trajkovic. Virtual network embedding via monte carlo tree search. *IEEE Trans. Cybernetics*, 48(2):510–521, 2018.

[115] Rashid Mijumbi, Juan-Luis Gorricho, Joan Serrat, Maxim Claeys, Filip De Turck, and Steven Latré. Design and evaluation of learning algorithms for dynamic resource management in virtual networks. In *2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014*, pages 1–9, 2014.

[116] Pham Tran Anh Quang, Abbas Bradai, Kamal Deep Singh, and Yassine Hadjadj Aoul. Multi-domain non-cooperative VNF-FG embedding: A deep reinforcement learning approach. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM Workshops 2019, Paris, France, April 29 - May 2, 2019*, pages 886–891, 2019.

[117] Sang Il Kim and Hwa Sung Kim. A research on dynamic service function chaining based on reinforcement learning using resource usage. In *Ninth International Conference on Ubiquitous and Future Networks, ICUFN 2017, Milan, Italy, July 4-7, 2017*, pages 582–586, 2017.

[118] Junling Li, Weisen Shi, Ning Zhang, and Xuemin Sherman Shen. Reinforcement learning based VNF scheduling with end-to-end delay guarantee. In *2019 IEEE/CIC International Conference on Communications in China, ICCC 2019, Changchun, China, August 11-13, 2019*, pages 572–577, 2019.

[119] Hye-Jin Ku, JH Jung, and Gu-In Kwon. A study on reinforcement learning based sfc path selection in sdn/nfv. *International Journal of Applied Engineering Research*, 12 (12):3439–3443, 2017.

[120] Chrysa A. Papagianni, Aris Leivadeas, Symeon Papavassiliou, Vasilis Maglaris, Cristina Cervello-Pastor, and Álvaro Monje. On the optimal allocation of virtual resources in cloud computing networks. *IEEE Trans. Computers*, 62(6):1060–1071, 2013.

[121] Oussama Soualah, Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. Energy efficient algorithm for VNF placement and chaining. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017, Madrid, Spain, May 14-17, 2017*, pages 579–588, 2017.

[122] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994.

[123] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *J. Artif. Intell. Res.*, 4:237–285, 1996.

[124] Dimitri P. Bertsekas. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific, 2005.

[125] Christopher J. C. H. Watkins and Peter Dayan. Technical note q-learning. *Machine Learning*, 8:279–292, 1992.

[126] Ali Hmaity, Marco Savi, Francesco Musumeci, Massimo Tornatore, and Achille Pattavina. Virtual network function placement for resilient service chain provisioning. In *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM), Halmstad, Sweden, September 13-15, 2016*, pages 245–252, 2016.

[127] Marcus Schöller, Martin Stiemerling, Andreas Ripke, and Roland Bless. Resilient deployment of virtual network functions. In *5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops, ICUMT 2013, Almaty, Kazakhstan, September 10-13, 2013*, pages 208–214, 2013.

[128] Michael Till Beck, Juan Felipe Botero, and Kai Samelin. Resilient allocation of service function chains. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, November 7-10, 2016*, pages 128–133, 2016.

Titre : Algorithmes exacts et heuristiques pour le chainage de services réseaux virtualisés

Mots clés : Virtualisation des Fonctions Réseau, Allocation des Ressources, Optimisation, Placement, Modélisation.

Résumé : Cette thèse traite du placement optimal et heuristique de fonctions réseau et de chaînes de fonction réseau dans des infrastructures cloud et réseau virtualisées. L'émergence de la virtualisation des fonctions réseau (NFV) permet de découpler les fonctions réseau en mode logiciel du matériel d'hébergement et de s'appuyer sur des serveurs génériques et d'éviter l'usage des matériels dédiés voire propriétaires.
Le placement de fonctions réseau virtualisées (VNF) est NP-Difficile puisqu'il s'agit de projeter un petit graphe de ressources virtuelles sur un graphe plus grand (graphe de l'infrastructure d'hébergement). Les solutions optimales, en particulier la programmation linéaire en nombre entier (ILP), ne passent pas à l'échelle. Sachant que la demande est dynamique et peut varier dans le temps et que le réseau est lui même variable dans le temps, il est important de prévoir des adaptations des placements. Cela peut s'effectuer par de l'élasticité sur les ressources d'hébergement réservées à une fonction réseau virtualisée, à un graphe de service réseau, et par une extension du graphe de service lui même en fonction du contexte et des exigences des utilisateurs.
La thèse propose une famille d'algorithmes pour le placement de chaînes de services réseau avec la possibilité d'étendre les ressources d'hébergement des VNFs en plus du placement initial. La thèse s'est penchée aussi sur l'extension de graphes de services réseau, déjà déployés, en adoptant une approche de type arbre de recouvrement. Plus spécifiquement une modélisation du problème en un "Steiner Tree Problem" a conduit à des performances proches de l'optimal pour des extensions de graphes au fil des demandes, en les traitant séparément. Les travaux de thèse sont par la suite revenus sur la rentabilité sur le long terme des algorithmes, en approchant le placement de chaînes de services et fonctions réseau comme un objectif long terme via de l'apprentissage par renforcement en se souciant plus de la rentabilité et de l'efficacité long terme des algorithmes contrairement aux approches visant exclusivement l'optimalité instantanée à chaque nouvelle demande.

Title : Algorithms for Virtual Network Functions chaining

Keywords : Network Function Virtualization, Resource Allocation, Optimization, Placement, Modeling.

Abstract : Network Function Virtualization (NFV) is an innovative emerging concept that decouples network functions from dedicated hardware devices. This decoupling enables hosting of network services, known as Virtualized Network Functions (VNFs), on commodity hardware and thus facilitates and accelerates service deployment by providers, improves flexibility, leads to efficient and scalable resource usage, and reduces costs. This paradigm is a major turning point in the evolution of networking, as it introduces high expectations for enhanced economical network services. One of the main technical challenges in this domain is the optimal placement of the VNFs within the hosting infrastructures. This placement has a critical impact on the performance of the network. The VNF placement and chaining problem is NP-Hard and there is a need for placement approaches that can scale with problem size and find good solutions in acceptable times. The overarching goal of this thesis is to enable dynamic virtual network resources provisioning to deal with demand fluctuation during the virtual network lifetime, and to enhance the substrate resource usage. Reserving a fixed amount of resources is inefficient to satisfy the VNF resource requirements. To cope with these problems, we propose dynamic resource management strategies.
In this thesis, both exact and heuristic algorithms are designed and evaluated in terms of optimality, complexity, ability to scale, and compared with the state of the art. Elastic mechanisms and scaling algorithms are first presented to improve adaptation and deployment of VNFs in NFV infrastructures to support increasing demand while increasing provider's revenue. Since network providers not only need to control, classify and steer user traffic flows in their dedicated slices but also want to extend their already acquired and operational virtual networks with additional service graphs, the thesis proposes extension algorithms of already hosted network functions graphs without disrupting initially deployed service instances. The thesis also addresses the VNF placement and chaining problem in an online and in a batch mode to improve performance in terms of longer time reward. An enhanced Reinforcement Learning-based approach is also proposed to improve the long term reward beyond what the previous methods can achieve.

Institut Polytechnique de Paris
91120 Palaiseau, France