DESIGN OF POLICIES MANAGEMENT TOOLS FOR SCALABLE SERVICE PROVISIONING IN CLOUD ENVIRONMENTS

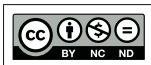
by

Hanan SUWI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN
M.A.Sc.

MONTREAL, AUGUST 08, 2019

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC



BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mrs. Nadjia Kara, Memorandum Supervisor
Département de génie logiciel et des TI, École de technologie supérieure

M. Abdelouahed Gherbi, President of the Board of Examiners
Département de génie logiciel et des TI, École de technologie supérieure

M. Aris Leivadeas, Member of the jury
Département de génie logiciel et des TI,École de technologie supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON JULY 18, 2019

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

CONCEPTION DES OUTILS DE GESTION DES POLITIQUES RESEAUX POUR UN APPROVISIONNEMENT SCALABLE DES SERVICES DANS DES ENVIRONNEMENTS BASES SUR LE CLOUD

Hanan SUWI

RÉSUMÉ

La virtualisation des fonctions réseau (NFV) est un nouveau paradigme qui permet aux fournisseurs de services de communication de proposer des services réseau scalables à moindre coût et avec un déploiement plus agile. Ceci est assuré en dissociant les fonctions réseau de l'infrastructure physique, en les remplaçant par des composants logiciels pouvant s'exécuter sur du matériel physique dédié. NFV encontre de nombreux défis à différents niveaux, tels que le placement des fonctions réseaux ou la gestion et l'adaptation des ressources. En effet, toute stratégie de placement et de routage des fonctions réseaux virtuelles doit respecter les conditions de service, les politiques et restrictions prédéfinies par le client, ainsi que les facteurs relatifs aux ressources disponibles au niveau de l'infrastructure physique du fournisseur tels que la bande passante du réseau, la latence le CPU ... Toutefois, la validation des stratégies de service réseau -pouvant être proposées par différentes parties prenantes (clients, fournisseurs de services, etc.) - doit être envisagée en priorité au cours du processus d'approvisionnement des services. En outre, lorsque ces stratégies présentent des conflits et / ou des redondances entre elles, cela entraîne des conséquences négatives sur l'ensemble du service réseau et une dégradation des performances. En conséquence, le processus d'approvisionnement des services NFV devrait passer par trois étapes principales: 1) détecter les conflits potentiels et les redondances entre les politiques réseaux; 2) résoudre les conflits potentiels et les redondances entre les stratégies et valider les demandes de service; 3) trouver l'emplacement optimal pour mapper les services réseau sur les ressources physiques sous-jacentes tout en satisfaisant le contrat de niveau de service et les politiques spécifiées. Plusieurs approches proposées au cours des dernières années tentent d'assurer les promesses du NFV et permettent la création, l'approvisionnement et la gestion de services réseau de manière flexible, agile et scalable. Mais malgré ces efforts, il est toujours possible d'améliorer les méthodes d'approvisionnement des services dans des environnements virtualisés. Le but de ce travail est de proposer de nouvelles techniques de gestion qui ajoutent de nouvelles fonctionnalités au Framework NFV. Notre proposition permet de détecter les différents conflits potentiels et redondances entre l'ensemble des politiques réseaux et ce de manière continue. De plus, elle permet de résoudre les conflits et les redondances détectées de manière entièrement automatisée et ne requiert pas l'intervention humaine d'un administrateur réseau. De plus, elle permet de découvrir la similarité entre les nouvelles demandes de service entrantes et les services réseau précédemment déployés dans le but de réduire le temps de provisionnement total des services en ignorant une ou plusieurs étapes du processus intégral. Pour atteindre cet objectif, ce travail est divisé en deux volets de recherche complémentaires. La première partie propose un nouveau mécanisme pour la détection et la résolution des conflits et des redondances entre les politiques réseaux, tandis que la deuxième permet de détecter les similarités entre les services réseau (SFC) afin de réduire le temps total de provisionnement des services demandés.

Mots clés: virtualisation des fonctions réseau, apolitique d'affinité, politique anti-affinité, détection de conflit, similarité, apprentissage-machine.

DESIGN OF POLICIES MANAGEMENT TOOLS FOR SCALABLE SERVICE PROVISIONING IN CLOUD ENVIRONMENTS

Hanan SUWI

ABSTRACT

Network Function Virtualization (NFV) is a new paradigm that promises communication service providers to offer scalable network services with lower cost and agile deployment. This is achieved by decoupling the network functionality from the physical infrastructure and replace them into software image that can run on top of commodity hardware. NFV encounters many challenges at different levels such as placement, management, and adaptation of resources. For example, any resource placement and chaining strategy should satisfy the service level agreement (SLA) and the pre-defined policies and restrictions, as well as the minimization of several factors including hardware resource utilization, network bandwidth and latency. However, validating the network service policies that could be proposed by many sources (e.g., customers, service provider, etc.) should be considered one of the most and prime steps in service provisioning process. Moreover, when these policies exhibit conflicts and/or redundancies with each other, this entails negative impacts on whole network service and performance degradation. Accordingly, NFV service provisioning process should be passed through three main steps: (1) detect the potential conflicts and redundancies among policies; (2) resolve the potential conflicts and redundancies among policies and validate the service request; (3) find the optimal location to map the network services into the underlying resources while satisfying the SLA and policies. Several approaches proposed over the past few years help to fulfill NFV promises and enable the creation, provisioning and managing of network service in flexible, agile and scalable way. Despite these efforts, there is still room to improve service provisioning in cloud settings. The aim of this work is to propose novel managing tools which add new functionalities to NFV framework. These tools enable to provide a continuous detection of different potential conflicts and redundancies among various types of policies. Moreover, it allows to resolve the detected conflicts and redundancies in fully automated way without network administrator assistance. Yet, it allows to discover the similarity between the new incoming service requests to deploy and network services that have been already deployed while reducing the total service provisioning time by skipping one or more of provisioning process steps. To achieve this aim, this work is divided into two complementary research tracks. The first track proposes a novel mechanism for the detection and resolution of conflicts and redundancies among policies, while the second track enables finding the similarity between network services to reduce total service provisioning time.

Keywords: Network Function Virtualization, affinity/anti-affinity policy, conflict detection, similarity, machine learning

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ALGORITHMS

Page

LIST OF ABREVIATIONS

AS	Autonomous System
CapEx	Capital Expenditures
CPU	Central Processing Unit
DC	Datacenter
MANO	Management And Orchestration
NFV	Network Function Virtualization
NFVO	Network Function Virtualization Orchestrator
OpEx	Operational Expenditures
PM	Policy Manager
PV-DBOW	Paragraph Vector-Distributed Bag of Words
SFC	Service Function Chain
SLA	Service Level Agreement
SSM	Service Similarity Manager
VNF	Virtual Network Function

LISTE OF SYMBOLS AND UNITS OF MEASUREMENTS

GB	Gigabyte
ms	Millisecond
s	Second
φ	Bijection function
δ	Minimum support threshold
Φ	graph embedding function

INTRODUCTION

0.1 Context

Network service providers aim to find cost-efficient and flexible approaches to create, deploy and manage the network services. Network Function Virtualization (NFV) is a new concept that enables designing effective solutions that allow service providers to achieve their goals. The main idea of NFV is to replace the dedicated hardware network components with software components that offer the same functionality and run on top of commodity hardware (e.g., standard x86 servers). Indeed, a set of network functions can be provided as Virtual Network Functions (VNFs). For example, instead of a dedicated firewall device in a network service, NFV can provide the same functionality of firewall through a VNF (i.e., virtual firewall) deployed on virtual machines or containers. Moreover, a set of VNFs can be chained to provide services through Network Function Virtualization Infrastructure (NFVI), which is called Service Function Chain (SFC).

To automatically deploy a set of VNFs composing complex network services (SFCs) and to steer traffic among these VNFs, an orchestrator is needed to deploy VNFs into shared substrate infrastructures and chain them with other VNFs to produce complex network services to serve different customers and tenants. Therefore, NFV management and orchestration (MANO) has been proposed by the European Telecommunication Standards Institute (ETSI) specification group for NFV. As illustrated in Figure 0.1. NFV-MANO has three main functional blocks (Quittek *et al.* (2014)):

a. NFV Orchestrator (NFVO): allows to manage the complete life-cycle of network service. The life-cycle management includes handling the incoming service requests, creating and instantiating the network service, configuring, monitoring the service and handling the scaling and migration policies.

b. Virtualized Infrastructure Manager (VIM): is responsible for managing the NFV-Infrastructure compute, network and storage resources.

c. VNF Manager (VNFM): manages the life-cycle of VNF instances. It manages the VNF requests their creations, termination, and monitoring.

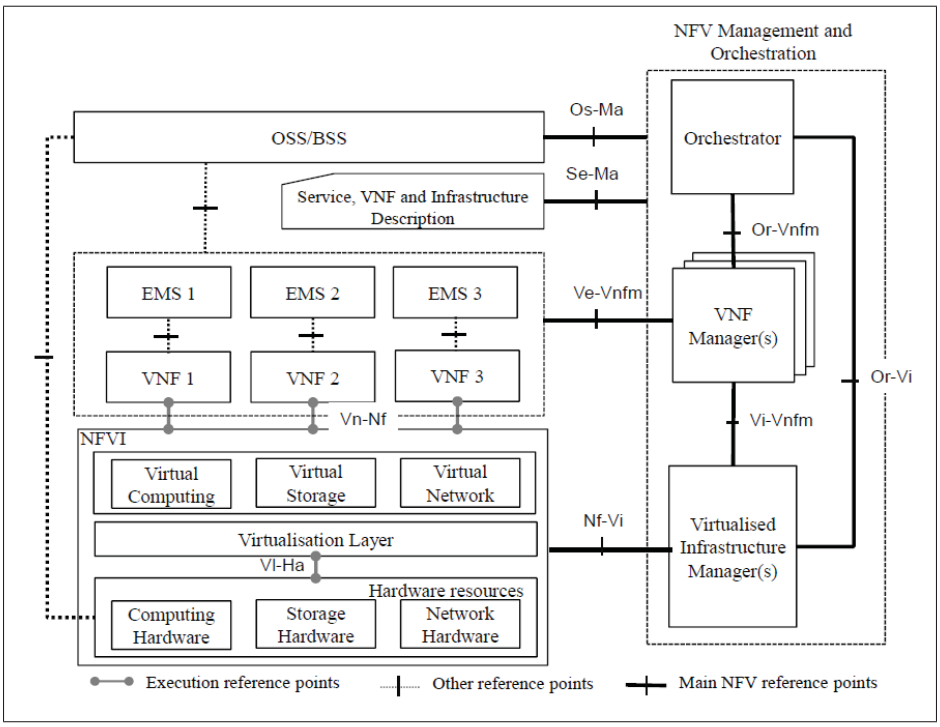


Figure 0.1 NFV framework architecture proposed by ETSI.

To provide a flexible way to deploy and manage the network services, NFV add the ability to multiple sources (e.g., service provider (SP), network administrator) to attach their custom policies concerning the placement routing, and VNF life-cycle management operations (e.g., scaling, migration). These Policies are added to the SFC request for many reasons such as efficiency, resilience, security and economic reasons. Accordingly, some affinity and/or anti-affinity policies are proposed to define placement restrictions of allocating the virtual network functions (VNFs) and virtual paths onto physical resources (data center DC, autonomous system AS, city, etc.) and links. For example, to increase the efficiency of SFC, SPs may decide

that the VNFs that exchange a lot of data should be co-located close to one another (e.g., within the same data center, or even on the same physical server). In another case, the SP may want to spread instances of the same VNF across multiple data centers in order to improve resilience in case a failure occurs in one of the data centers. Whereas, some other affinity and/or anti-affinity used to specify and manage the VNF resources (e.g., storage, compute) and manage the VNF life-cycle operations such as scaling (i.e., adding/removing VNF instances) and migration (i.e., moving a VNF to another location). For example, one policy may be added to SFC state that a certain VNFs should be scaled-out (i.e., adding more VNF instance) if the CPU utilization exceeds 80%. Another policy may be added to SFC request to prevent certain VNFs to migrate to another location. These policies are one of the key enablers to construct a flexible management and orchestration function in NFV-MANO layer (EISGI (2014)).

Regarding the placement policies, "affinity" is defined as "co-location," which specifies that a VNF is to be placed in a physical location or share a physical resource with another VNF. "anti-affinity" is defined as "non-co-location," which specifies that a VNF should not be placed in a same physical location or does not share a physical location with another VNF. With respect to VNF life-cycle management, "affinity" is defined as "allow/permit," which indicates that the mentioned policies are applicable to the specific VNF. "Anti-Affinity" is defined as "not allow/deny," which indicates that the mentioned policies are not applicable to a specific VNF.

However, the problem arises when these policies exhibit some conflicts and/or redundancies with one another, this leads to inconsistent network service provisioning and entails several negative impacts such as Service-Level Agreement (SLA) violation and performance degradation. Accordingly, NFV service provisioning process passes through three main steps: (1) detect the potential conflicts and redundancies among the affinity/anti-affinity policies; (2) resolve the potential conflicts and redundancies among policies and validate them; (3) find the

optimal location to map the network services into the underlying resources with satisfying the SFC policies such as affinity and anti-affinity policies.

NFV promised to decrease costs and complexity of the provisioning and the management of new and already deployed services, and the management of available resources in NFVI. However, one of the most important challenges in NFV is having smart, scalable, and fast functionality provided by NFV-MANO layer for the provisioning of VNFs and new network services while meeting the validated pre-defined policies, service level agreement (SLA), and fault recovery.

0.2 Problem

When a set of attached policies into the new incoming SFC request exhibit some conflicts and/or redundancies with one another, the service provider may face serious problems that might entail several negative impacts such as Service-Level Agreement (SLA) violation and performance degradation. Hence, when the NFV placement and chaining algorithm try to deploy the requested SFC, none of the resulting will lead to a feasible deployment of the SFC request. The service provider should however be able to differentiate between a non-acceptance of the SFC request caused by the leakage of appropriate physical resources or because of the conflicting policies.

Therefore, the NFV-MANO needs to have a novel detection mechanism to check the consistency of SFC requests and detect and filter out any potential conflicts and redundancies between policies (EISGI (2014)). Moreover, after the network service mapping is complete, each VNF is continuously monitored by the MANO layer, since there is a need to guarantee that the existing policies are not violated with the arrival of new policies that may be added by the NFV-Orchestrator or network administrator in response to the changes that might occur at the virtual and/or substrate network levels.

Although, many research works had been addressed the problem of network service provisioning and highlighted the strong need for an orchestration layer that is able to manage the network service provisioning and fulfill NFV promises (Herrera & Botero (2016)) and even if, several research works had addressed the issue of the detection and resolution of the potential conflicts and/or redundancies among policies before instantiating the network service, there is still room to improve these mechanisms and to address a set of persistent limitations of the proposed solutions.

Indeed, the proposed approaches in the literature mainly focus on placement and chaining algorithms (e.g., exact and heuristic approaches) which proposed to deploy the new network service by finding the efficient locations to map sequence of VNFs and steer the traffic among these VNFs considering a set of validated (i.e., free of conflicts) affinity and/or anti-affinity policies. To the best of our knowledge, none of the proposed research work proposes a policy conflict and redundancy detection and resolution algorithms before and after network service instantiation and/or a method to skip detecting the conflicts and redundancies among policies, and/or placement and chaining processes in order to reduce the processing load induced by these algorithms. Indeed, accordingly, we wanted to accelerate network service provisioning process by skipping one or more service provisioning steps (i.e., detecting and resolving the conflicts/redundancies among policies, placement and chaining) while satisfying the validated affinity/anti-affinity constraints, service level agreement (SLA), performance, and fault recovery. Therefore, we propose a new tool that enables to detect the similarity between a new SFC and the already deployed SFCs in order to minimize the NFV network service provisioning time. In another word, instead of consuming the whole NFV provisioning time required both policy conflict detection and resolution algorithms and the placement and chaining algorithm.

0.3 Objectives

To address the issues and fill the gaps outlined in the previous section, this research project proposes to design two manager tools that can be integrated into NFV-MANO layer in order to provide a scalable, elastic and rapid service provisioning. These manager tools must satisfy the following design requirements:

a. must have the ability to detect different potential conflicts and redundancies among both placement and resource management policies (e.g., scaling, migration);

b. must provide a continuous policy detection and resolution;

c. must provide an automated way to resolve the potential conflicts and redundancies;

d. must have the ability to check if the new network service has a similarity with any network service/ sub-service that have been deployed in NFVI.

0.4 Methodology

In order to achieve these objectives, this project will be divided into two research tracks. The first track focuses on the definition and validation of a Policy Manager (PM) tool integrated into VNF manager (VNFM) in NFV-MANO framework. This tool provides a novel offline and real-time policy conflict and redundancy detection, and resolution mechanisms for NFV policies. It allows resolving various potential conflicts and redundancies automatically without administrator assistance. The second track focuses on minimizing the cost and complexity of provisioning new SFC services by proposing Service Similarity Manager (SSM) tool integrated into NFV-Orchestrator (NFVO) in NFV-MANO layer. This tool allows to find matching between the new incoming network service request and deployed SFCs to reduce the time of network service provisioning. The two manager tools are complementary to each other and

can be embedded into NFV-MANO to provide a scalable and efficient network service provisioning. In the first research track and to propose a new Policy Manager tool, we started by classifying the affinity and anti-affinity policies that have common characteristics into set of groups and encoding each of a plurality of conflicts and redundancies between two policies into a known binary string. A set of affinity and anti-affinity policies have been defined for both placement and resource management tasks (e.g., scaling and migration), while considering both service providers and customers as potential sources of these policies. Then, we propose offline and real-time policy conflict and redundancy detection mechanisms to enable continuous and comprehensive detection and resolution of issues at both placement and resource management levels. Yet, we propose an automated resolution engine to resolve the conflicts and redundancies between policies automatically based on priority fields values. A policy manager framework has been developed and extensive experiments have been performed to assess its performance. In the second research track and to propose Service Similarity Manager tool, we model a similarity manager that can find exact and partial matching between SFCs. We first analyze various examples of SFC matching by considering different types of SFC graphs (e.g., linear and non-linear SFC topology), sizes (number of requested VNFs per SFC request) and attributes (e.g., CPU, bandwidth, affinity/anti-affinity). The problem of similarity detection between SFCs has been mathematically formulated using skip-gram, Paragraph Vector-Distributed Bag of Words (PV-DBOW) machine learning embedding model and Frequent Sub-Graph Mining (FSM) techniques. A service similarity manager framework has been developed and extensively tested using datasets generated randomly using NetworkX python library.

0.5 Technical Contributions:

Designing policy and service similarity managing tools as new functionalities to add into NFV-MANO layer was not straightforward task. Therefore, we divided this project into two distinct

steps each focusing on the development and validation of one of the proposed tools. This leads us to produce one regular patent and one provisional application for patent as well publish one journal paper and submit a second one. Thus, the main contributions of this thesis are:

1) Validate the SFC request, by:

a. Detecting different potential conflicts and redundancies between placement and resource management policies;

b. Guarantee that newly added policies do not trigger any conflicts and redundancies with the existing ones;

c. Resolving different potential conflicts and redundancies in automated way and without administrator assistance.

2) Discover the similarity between SFC services, by:

a. Finding the similarity between the new incoming network service request and all the existing services that have been already deployed in NFVI;

b. Minimizing the total time required to provision a new service and decrease costs and complexity for the provisioning of new services.

First, our first main contributions involve providing a novel detection and resolution for NFV policies. This was the subject of our first article, entitled “Offline and Real-Time Conflict and Redundancy Detection in Network Function Virtualization (NFV) policies”, submitted to "Future Generation Computer System" in April 2019 and accepted for publication in June 2019. In addition, the proposed approach, named “Semantic Detection and Resolution of Conflicts and Redundancies in Network Function Virtualization Policies”, was subject to a provisional

patent application filled by Ericsson Patent Unit Canada to the United States Patent and Trademark Office (USPTO) in March 2019. You will find the article documenting this approach in Chapter 2. Our last main contributions involve about finding the similarity between the network services to provide a scalable and agile network service provisioning. This was the subject of our second article, entitled “Machine-Learning based Similarity detection of Service Function Chains for agile service provisioning” submitted to "IEEE Transactions on Services Computing" in June 2019. In addition, the proposed approach, named “Similarity detection for Service Function Chain management”, was subject to a provisional patent application filled by Ericsson Patent Unit Canada to the United States Patent and Trademark Office (USPTO) in June 2019. You will find the article documenting this approach in Chapter 3.

a. Hanan Suwi, Nadjia Kara & Claes Edstrom . Semantic Detection And Resolution of Conflicts and Redundancies in Network Function Virtualization Policies. Regular Patent reference number P77159, March 2019.

b. Hanan Suwi, Nadjia Kara & Claes Edstrom . Similarity detection for Service Function Chain management. Application Patent reference number P78281, June 2019.

c. Hanan Suwi, Nadjia Kara, Omar Abdel Wahab & Claes Edstrom . Offline And Real-Time Conflict and Redundancy Detection in Network Function Virtualization (NFV) policiesAccepted for publication in Elsevier Journal of Future Generation and Computer Systems, May 2019.

d. Hanan Suwi, Fawaz A. Khasawne, Nadjia Kara & Claes Edstrom. Machine-Learning based Similarity Detection for Service Function Chains for agile service provisioning. IEEE Trans. On Cloud Computing Journal (Submitted), June 2019.

0.6 Organization of thesis

Since this document is structured by articles, we present a general literature review followed by two chapters devoted for each mentioned contributions. Each of these chapters, provides more details about the proposed approaches and algorithms, the performance analysis and a general conclusion. Chapter 4 presents a discussion of the main results achieved. The conclusion and future work are presented at the end of this thesis.

CHAPTER 1

LITERATURE REVIEW

NFV has recently gained significant attention from both industry and academia as an agile technology lead to significantly decrease the Operational Expenditures (OpEx) as well as Capital Expenditures (CapEx). NFV challenges such as detecting and resolving the conflicts and redundancies among various policies, as well as scalable network service provisioning have drawn increased attention of many research teams in recent decades.

In this section, we will provide a literature review concerning the NFV technology and discuss some NFV challenges such as detecting and resolving the conflicts and redundancies among policies in NFV, and network service provisioning.

The virtualization of network functions was first proposed by European Telecommunication Standards Institute (ETSI) as a concept to reduce the cost and accelerate network service provisioning for service providers. NFV aims to change the way that network operators build their network services by using standard IT virtualization technology and to run virtual network function NFV on commodity servers rather than on dedicated hardware (Mijumbi *et al.* (2016)). Designing NFV management and orchestration mechanisms to provide agile network service provisioning remains one of the most critical challenges for NFV concept. NFV management and orchestration layer must provide smart, flexible and fast algorithms to validate and resolve various potential conflicts and redundancies among service policies and map the network services into the appropriate physical resources to steer the traffic among VNFs in less cost and scalable manner.

Affinity/anti-affinity policies have been previously studied in cloud computing field. (Larsson *et al.* (2011)) addressed the challenges of assisting application owners in having a hand in the placement location of service components. This may be beneficial, for example, in situations where application owners want some data to reside in a certain country, some software modules with excessive intercommunication overhead to reside in the same datacenter/physical

host, or some data replication servers to be deployed on different hosts. In such a case, they (1) convert hierarchical graph structures into formalized placement constraints, and (2) devise a mathematical model that can be employed to extend current placement techniques by providing support for service own-controlled instructions. (Chen *et al.* (2013)) proposed a heuristic that helps the model to select the appropriate virtual machines for migration taking into account the affinity and anti-affinity policies. Accordingly, in NFV technology, ETSI standard group identify that attaching such the affinity/anti-affinity policies consider the key enabler to provide a flexible management to the functionality provided by the management and orchestration layer. Affinity/anti-affinity policies are attached to network services for many reasons such improving the efficiency, resilience, legislation, privacy and economic. For example, to improve the resilience of network service, the service provider may want to place the same VNFs across multiple different physical resources (e.g., datacenter, servers). Therefore, many placement and chaining algorithms designed to find the feasible physical location satisfying the pre-defined affinity and anti-affinity polices.

Most of studies focus on proposing placement and chaining algorithms satisfying the affinity/anti-affinity policies. In (Allybokus *et al.* (2018)) proposed a heuristic approach using Integer Linear Program (ILP) formulation to assign each path and VNFs of a network service to the best feasible physical location satisfying the anti-affinity constraints. (Zou *et al.* (2018)) proposed a novel Affinity-Based Allocation (ABA) heuristic approach for placing the service functions to solve the deployment over large networks. (Bouten *et al.* (2017)) proposed to embed location requirements along with a set of co-location constraints into the placement of VNFs and virtual edges. To this end, they proposed a set of affinity and anti-affinity constraints that can be employed by service providers to define placement restrictions. Moreover, a semantic SFC validation method was advanced to allow virtual network function infrastructure providers to verify the validity of the constraints set and provide service providers with the appropriate feedback. This step is important to filter out non-valid SFC requests prior to sending them to the placement algorithm, thus reducing the placement process overhead.

Despite their importance, none of these works covers the definitions of affinity and anti-affinity policies for both placement and resource-oriented actions. Our work, on the other hand, covers both actions by enabling the definition of placement, horizontal scaling, vertical scaling, and migration policies. Moreover, in the aforementioned approaches, only service providers and/or network administrators are allowed to set and customize policies. Our solution provides better flexibility in this regard by allowing customers as well to express their own policies and constraints. This enriches the set of defined policies and provides the policy anomaly detection engine with a broader set of policies on which to operate. Finally, none of the approaches discussed provides both offline and real-time conflict and redundancy detection and proposes an automated resolution mechanism. Our solution fills this gap by introducing a real-time conflict and redundancy detection and resolution mechanism that allows the VNF manager to comprehensively and constantly detect conflicts and redundancies and resolve them automatically. This is of prime importance in ensuring that newly added policies do not entail any conflict or redundancy with existing policies (that already underwent the offline policy verification process).

Although detecting conflicts and redundancies among policies is very important for service validate, detecting and resolving the conflict and redundancies among the affinity/anti-affinity policies might slow down the network service provisioning process. One way to reduce time and cost consumed in detecting and resolving the conflicts and redundancies among policies as well as the total provisioning time is to detect similarity between new SFC request and already deployed SFC and use such information in order to decide if one of these process could be skipped. To the best of our knowledge, none of the proposed work propose a method to find the similarity between network services to reduce the total service provisioning time by skipping the detection and resolution process and/or placement and chaining process. The next two chapters describe a more detailed literature review for the two proposed solutions. You will find a review of literature dedicated to detecting and resolving the conflicts and redundancies among policies in the "Related Work" section of Chapter 3, then a literature review devoted to finding the similarity between network services in the "Related Work" section of Chapter 4.

CHAPTER 2

OFFLINE AND REAL-TIME CONFLICT AND REDUNDANCY DETECTION IN NETWORK FUNCTION VIRTUALIZATION (NFV) POLICIES

Hanan Suwi[a], Nadjia Kara[b], Omar Abdel Wahab[c], Claes Edstrom[d], Yves Lemieux[e]

[a b c] Department of Software Engineering and IT, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[d e] Ericsson Canada,
8275 Route Transcanadienne, Ville Saint-Laurent, Québec, Canada H4S 0B6

Paper Accepted for publication in Elsevier Journal of Future Generation and Computer Systems, May 2019.
Regular Patent reference number P77159 filed by Ericsson Patent Unit Canada to the United States Patent and Trademark Office (USPTO) in March 2019

2.1 Abstract

Network Function Virtualization (NFV) is a new technology that allows service providers to improve the cost efficiency and flexibility of network service provisioning. This is accomplished by decoupling the network functions from the physical environment within which they are deployed and converting them into software components that can run on top of commodity hardware. Despite its importance, NFV encounters many challenges at the placement, resource management, and adaptation levels. For example, any placement strategy must take into account the minimization of several factors, including the hardware resource utilization, network bandwidth and latency. Moreover, Virtual Network Functions (VNFs) should continuously be adjusted to keep up with the changes that occur at both the data center and user levels. Over the past few years several efforts have been made to come up with innovative placement, resource management, and readjustment policies. However, a problem arises when these policies exhibit some conflicts and/or redundancies with one another, since the policies are proposed by multiple sources (e.g., service providers, network administrators, NFV-orchestrators and customers). This constitutes a serious problem for the network service as a whole and has several negative impacts such as Service-Level Agreement (SLA) violations and performance degradation. In

this paper, we tackle this problem by defining a set of affinity and anti-affinity policies, which can be proposed by service providers and customers for both placement and resource management actions, such as scaling and migration, using the eXtensible Markup Language (XML). We then propose a conflict and redundancy detection and an automated resolution mechanisms to identify and solve the issues within and between policies. Finally, we integrate a real-time detection component into our solution to provide continuous and comprehensive conflict and redundancy resolution as new policies are introduced. The experimental results show that the proposed policy detection tools could rapidly identify and detect conflicts and redundancies among NFV policies. Moreover, our proposed detection mechanisms have the ability to detect conflicts and redundancies for different types of policies.

Index Terms: NFV, Policy analysis, Affinity, Anti-Affinity, Anomaly detection

2.2 Introduction

Network Function Virtualization (NFV) has recently been gaining increased attention within academic and industrial circles, thanks to its ability to reduce both Capital Expenditures (CapEx) and Operational Expenditures (OpEx). NFV promises to enable telecommunication service providers to find cost-efficient and flexible approaches to create, deploy and manage network services. It has made it possible to shift network functions (e.g., firewalls, load balancing, etc.) from expensive hardware equipment to software components that operate on top of commodity hardware as Virtual Network Functions (VNFs) (Mijumbi *et al.* (2016); Li & Chen (2015); Rotsos *et al.* (2017)). For example, instead of having a dedicated firewall device in a network service, NFV can provide the same functionality of firewalls through a VNF (i.e., a virtual firewall) deployed in virtual machines or containers. Moreover, a set of VNFs can be chained to provide services through a Network Function Virtualization Infrastructure (NFVI), also known as a Service Function Chain (SFC). To go into further detail, with NFV technology, the SFC service will be implemented in software running on top of hardware or physical servers. The

VNFs may then be re-instantiated at different network locations without the need to purchase and install new hardware components.

NFV management and orchestration (MANO) (Quittek *et al.* (2014)) has been proposed by the European Telecommunication Standards Institute (ETSI) specification group to help NFV technology to fulfill NFV promises by providing a set of operations responsible for managing the deployment of a set of VNFs composing complex SFC services, steer traffic among these VNFs and manage the life cycle of SFC service and VNF resources. NFV-MANO consists of three functional blocks: (1) The NFV-Orchestrator (NFVO), which allows managing the complete life cycle of network service; the life cycle management operations include handling incoming service requests, creating and instantiating the network service, configuring, monitoring the service, handling scaling (e.g., adding/removing VNF instance) and migration (i.e., moving the VNF to another location) policies; (2) The Virtualized Infrastructure Manager (VIM), which is responsible for managing the NFV-Infrastructure computing, network and storage resources, and (3) The VNF Manager (VNFM), which manages the life cycle of VNF instances; it manages VNF requests, creation, termination, and monitoring.

Service providers and/or customers may have certain special requirements and restrictions regarding the placement, routing, and VNF life cycle management operations (e.g., scaling, migration, etc.) of their Service Function Chain (SFC) requests. For example, as mentioned in *Service Quality Metrics* by the ETSI NFV standard group (EISGI (2014)), anti-affinity placement constraints are used to allocate the VNFs of the same SFC into different physical hosts (or even separate data centers) in order to improve the resilience and robustness of their services. Some other requirements may also relate to legislative, efficiency, economic, and privacy issues (Bouten *et al.* (2017)). Thus, traditional embedding approaches Fischer *et al.* (2013), in which only node and link requirements can be specified, are no longer adequate. This raises the need to explicitly attach some affinity and anti-affinity policies and constraints to embedding requests. Doing so, however, brings up another challenge, posed by the existence of some conflicts and/or redundancies in the set of affinity and anti-affinity policies received. For example, if the service provider requires two VNFs of the same type to co-locate on the same host,

but also states that all instances of such a VNF type should reside in different data centers, then there is an obvious conflict. Or, if the provider or customer identifies 80% as a threshold value at which a VNF should be scaled out (i.e., add more VNF instances), and then the NFV-Orchestrator adds a new policy with a different threshold value, an obvious conflict therefore arises. Therefore, the NFV technology needs to have a novel detection mechanism to validate and check the consistency of SFC requests and detect and filter out any potential conflicting and redundant policies (ETSI (2017)).

Customers here are wide-ranging in nature and include end users, tenants, or virtual network operators, for instance, who requests and leases services from telecommunication service providers.

2.2.1 Problem Statement

Several approaches take the affinity and anti-affinity placement policies into account in their mapping and embedding algorithms. Others (Bouten *et al.* (2017); Chen *et al.* (2013); Allybokus *et al.* (2018)) are interested in improving the placement strategies of VNFs and the performance of virtualized environments. The latter are useful in situations where network administrators or application owners seek to reduce the intercommunication overhead of software components or want their data to reside in particular countries (for example, for legislative reasons). Yet other approaches (Figueira *et al.* (2015)) are more focused on defining resource management constraints and compelling business rules. The goal in these cases is to improve the overall orchestration process in large systems consisting of multiple sub-systems. Relatedly, some security-oriented approaches (Deng *et al.* (2015)) have been proposed to prevent unauthorized access to valuable services and improve the performance of security tools (e.g., firewalls).

Despite their importance, these approaches are focused on either placement or resource management policies. However, none of them covers both aspects in a simultaneous fashion. In this work, we tackle this limitation by considering both these policy classes together. This

is achieved by allowing the definition of the placement, horizontal scaling, vertical scaling and migration policies; the components of each of these policies could be listed as follows: placement - mapping the virtual node/path onto feasible underlying physical hosts; horizontal scaling - adding/removing VNF instances; vertical scaling - adding/removing CPU cores, and migration - moving the VNF node to another location. The second shortcoming of existing approaches is that they restrict the responsibility for designing and customizing the policies to only providers and/or network administrators, and then pass these policies on to the NFV Management and Orchestration layer (MANO) via the network service descriptor files (Quittek *et al.* (2014)). Our work, on the other hand, allows customers as well to express their own policies and constraints based on SLA and network conditions. This contributes to enriching the set of defined policies, thus providing the detection engine with a more comprehensive set of policies and constraints to analyze and learn from. Finally, and most importantly, most of the current approaches are limited to offline conflict and redundancy detection (i.e., they validate the SFC policies before SFC instantiation) and to resolving conflicts and redundancies with the assistance of the network administrator. We overcome this limitation in our work and propose an automated technique to solve conflicts and redundancies based on the policies' priorities, and without administrator assistance; we further propose a real-time technique that can be used in addition to the offline technique to provide a continuous and comprehensive detection of conflicts and redundancies. This is important to guarantee that newly added policies do not trigger any conflicts and/or redundancies with existing ones.

In the present work, the following sets of terms are respectively used interchangeably: "constraint" and "policy"; "VNF instance", "VNF node" and "functionNode"; and "virtual network path", "virtual link" and "connections".

2.2.2 Contributions

Our proposed solution works as follows. Once a new SFC request containing a set of affinity and anti-affinity policies is received, an XML-based policy file is correspondingly produced. The policy file consists of a set of affinity and anti-affinity policies that describe a set of VNF

actions (e.g., placement, migration, horizontal scaling, or vertical scaling). An offline policy verification process is then executed on the policy file to detect and identify any conflicts and/or redundancies between the affinity and anti-affinity policies. Conflicting and redundant policies are then conveyed to the policy resolution engine to automatically resolve the conflicts and redundancies between the policies without administrator assistance. This is done by comparing the policies' priorities and choosing to retain the ones with the highest priorities. Once all the conflicts and redundancies have been resolved, the actual SFC embedding process is executed to allocate each VNF to the corresponding physical node, while respecting the specified affinity and anti-affinity placement constraints. Finally, a real-time policy verification process is run to ensure that the existing affinity and anti-affinity policies have not been violated with the arrival of new policies. The main contributions of this paper can be summarized as follows:

- Definition of a set of affinity and anti-affinity policies for both placement and resource management actions (e.g., scaling and migration), while considering both the service providers and customers as potential sources of these policies. To the best of our knowledge, this work is the first that considers placement and resource management actions simultaneously.
- Proposal of offline and real-time policy conflict and redundancy detection mechanisms. This allows the continuous and comprehensive detection and resolution of issues at both the placement and resource management levels.
- Proposal of an automated resolution engine to automatically resolve the conflicts and redundancies between policies based on priority field values.

The rest of the paper is organized as follows, In Section 2.3, we present a literature review. In Section 2.4, we give definitions of affinity and anti-affinity policies for both placement and VNF life cycle management operations. An overview of the proposed approach and the algorithms design are provided in Section 2.5. We analyze the performance of the proposed solution in Section 2.6. Finally, the conclusion and future research directions are drawn in Section 2.7.

2.3 Related Work

In this section, we review the most common and recent research works, in which the affinity and anti-affinity policies are identified in cloud computing and NFV systems.

2.3.1 Affinity and Anti-affinity Policy in Cloud Computing

Several recent works in the field of cloud computing have examined affinity and anti-affinity placement policies (Sundararajan *et al.* (2015); Pachorkar & Ingle (2013)). (Konstanteli *et al.* (2014, 2012)) proposed a method to derive optimal service-to-cloud allocation strategies, with a focus on elastic services; with its size being subject to dynamic growth and shrink, based on a varying number of users and patterns of requests. The problem formulation takes into consideration metrics such as trust, eco-efficiency, cost and affinity and anti-affinity constraints that service components might have. The problem is modeled as a General Algebraic Modeling System (GAMS) and solved under real-world provider settings. (Espling *et al.* (2016)) addressed the challenge of assisting application owners in having a hand in the placement location of service components. This may be beneficial, for example, in situations where application owners want some data to reside in a certain country, some software modules with excessive intercommunication overhead to reside in the same data center/physical host, or some data replication servers to be deployed on different hosts. In such a case, they (1) convert hierarchical graph structures into formalized placement constraints, and (2) devise a mathematical model that can be employed to extend current placement techniques by providing support for service own-controlled instructions. (Larsson *et al.* (2011)) proposed a heuristic that helps the model select the appropriate virtual machines for migration, taking into account the affinity and anti-affinity policies. In a bid to decrease the communication overhead and improve the performance of Virtual Machine (VM) allocation, (Chen *et al.* (2013)) proposed an Affinity-Aware Grouping method for Allocation (AAGA) of VMs. Specifically, a grouping method based on the bin packing heuristic algorithm is advanced to deploy VM groups into physical machines. The method was tested in an environment with multiple virtual clusters consisting of 56 VMs against non-affinity-aware grouping-based allocation methods.

To manage SLAs with respect to Business Level policies and cloud provider objectives (e.g., to maximize economic profit) (Macías & Guitart (2014)) define two set of policies: (1) Revenue Maximization and (2) Client Classification according to client affinity and QoS. Affinity is used here to express the relationship between the cloud provider and the client.

2.3.2 Affinity and Anti-affinity Policy in NFV

In the field of NFV, affinity and anti-affinity constraints are used to define a set of placement restrictions to identify how to map virtual nodes and paths onto certain physical hosts and identify the relation between sets of virtual nodes (VNFs) and paths to be co-located or non-co-located into certain physical hosts. (Allybokus *et al.* (2018)) proposed a heuristic approach using an Integer Linear Program (ILP) formulation to assign each path and VNFs of a network service to the best feasible physical location satisfying the anti-affinity constraints. (Bhamare *et al.* (2017)) proposed a novel Affinity-Based Allocation (ABA) heuristic approach for placing the service functions to solve deployments over large networks.

Several works have studied the detection of anomalies (e.g., conflicts) among policies in NFV. (Zou *et al.* (2018)) tackled the challenge of detecting anomalies in independently generated SFC policies in IoT environments. The problem in this case stems from the fact that multiple policy makers design their policies while being unaware of each other's policies. To address this problem, a composition method is advanced to detect and solve anomalies among the SFC policies designed by different subjects in these IoT networks. The solution comprises three main phases: policy composition, anomaly resolution, and dynamic policy adaptation. In the first phase, user-specified policies are given as input to the composition method, which then analyzes them and generates an anomaly-free policy set. In the second phase, the existing anomalies are analyzed and categorized into three types, namely, redundancy, incompleteness, and conflict. The third phase includes a method to detect ambiguities induced by conflicting intents from different policy makers.

(Bouten *et al.* (2017)) proposed to embed location requirements along with a set of co-location constraints into the placement of VNFs and virtual edges. To this end, they proposed a set of affinity and anti-affinity constraints that can be employed by service providers to define placement restrictions. Moreover, a semantic SFC validation method was advanced to allow virtual network function infrastructure providers to verify the validity of the constraints set and provide service providers with the appropriate feedback. This step is important to filter out non-valid SFC requests prior to sending them to the placement algorithm, thus reducing the placement process overhead. (Bouten *et al.* (2017)) proposed a model to detect inter-function anomalies (i.e., the interference between two or more network functions within the same network) in SDN/NFV deployment. To achieve that, they integrate a new component named Policy Analyzer (PA) into the NFV orchestrator for anomalies detection. Then, the PA generates a report containing the results of the analysis to be used by network administrators to reconfigure the network.

In (Figueira *et al.* (2015)), a policy definition framework was advanced to compel business rules and specify resource constraints in NFV settings. The policy framework focused on the overall orchestration requirements for services consisting of various subsystems. The work also provides a profound analysis on policy scope, static versus autonomic policies, global versus local policies, policy conflict detection and resolution, policy actions and translations, and interactions between policy engines. (Deng *et al.* (2015)) proposed VNGuard, a framework for effectively provisioning and managing virtual firewalls, while capitalizing on NFV and SDN technologies. In VNGuard, users are allowed to smoothly define their security policies for virtual networks, while eliminating the need to dig into low-level virtual network deployment. These high-level security policies are then automatically translated into low-level firewall rules, and optimal virtual firewall deployment solutions are derived, while complying with the defined resource and performance constraints.

Despite their importance, none of these works covers the definitions of affinity and anti-affinity policies for both placement and resource-oriented actions (e.g., scaling, migration, etc.). Our work, on the other hand, covers both actions by enabling the definition of placement (mapping

the virtual node/path onto underlying resources), horizontal scaling (adding/removing VNF instances), vertical scaling (adding/removing CPU cores), and migration (moving the VNF node to another location) policies. Moreover, in the aforementioned approaches, only service providers and/or network administrators are allowed to set and customize policies. Our solution provides better flexibility in this regard by allowing customers (e.g., tenant) as well to express their own policies and constraints. This enriches the set of defined policies and provides the policy anomaly detection engine with a broader set of policies on which to operate. Finally, none of the discussed approaches provide both offline and real-time conflict and redundancy detection and propose an automated resolution mechanism. Our solution fills this gap by introducing a real-time conflict and redundancy detection and resolution mechanism that allows the VNF manager to comprehensively and continuously detect conflicts and redundancies and resolve them automatically. This is very important in ensuring that newly added policies do not lead to any conflict or redundancy with existing policies (that already underwent the offline policy verification process).

2.4 Affinity and Anti-Affinity Policies Model and formats

In this section, we define the affinity and anti-affinity placement and VNF life cycle management-oriented policies considered in this work.

- **Placement-oriented policies**: Defined to add a custom restriction on the SFC service mapping process, where the service provider or customers can attach their restrictions on where and how to map the virtual nodes (VNFs) or virtual links (VLs) onto the underlying resources. In addition, with placement policies, the relationship between sets of virtual nodes to be co-located or non-co-located within the same physical space (e.g., physical server, data center DC) is defined.

    a. *Affinity (LocationA, B)*: This policy format pertains to one VNF node, where *B* is a specific VNF instance, VNF type or virtual network interconnecting path, and *LocationA*

is a specific physical location (e.g., physical node, data center (DC), network domain, city etc.). This constraint implies that item *B* **should** be placed at *LocationA*.

b. *Anti-Affinity (LocationA, B)*: This constraint implies that *B* **should not** be placed at *LocationA*.

c. *Affinity (LocationTypeA, B, C)*: This policy format pertains to two VNF nodes, where *B* and *C* are specific VNF instances, VNF types or virtual network paths, and *LocationTypeA* is the type of the physical location (e.g., data center, network domain, or physical host). This constraint implies that both *B* and *C* **should** co-locate at the same physical resource with respect to the mentioned *LocationTypeA*. For example, *Affinity (DC, VNF_1, VNF_2)* indicates that both *VNF_1* and *VNF_2* should be co-located in the same data center.

d. *Anti-Affinity (LocationTypeA, B, C)*: This constraint implies that both *B* and *C* **should not** be placed at the same physical location resource with respect to the mentioned *LocationTypeA*.

- **VNF life-cycle management-oriented policies**: Defined to add a custom restriction on VNF life cycle management operations, such as horizontal scaling, vertical scaling and migration.

a. *Affinity (PolicyType, Conditions, A)*: Where A is a specific VNF instance, *PolicyType* refers to a specific management operation (e.g., vertical scaling, horizontal scaling or migration), and *Conditions* are the restrictions relating to the *PolicyType* (e.g. minimum/maximum number of instances, minimum/maximum number CPU cores, migration type, or the threshold values according to which vertical/horizontal scaling should be triggered). This constraint implies that a certain *PolicyType* is allowed/permitted to *A* by the set of *Conditions*.

b. *Anti-Affinity (PolicyType, Conditions, A)*: This constraint implies that a certain *PolicyType* will be disabled on item *A*. For example, *Anti-Affinity (vertical scaling, null, VNF_3)* implies that the vertical scaling policy should be disabled for *VNF_3*.

2.5 Proposed Approach

The NFV-MANO layer plays an important role in the NFV system, and is responsible for managing and controlling the overall operations of the VNFs (e.g., placement, scaling, migration) (Quittek *et al.* (2014)). Therefore, the conflict and redundancy detection approach of the proposed policy manager framework is integrated as a new functionality into the VNF Manager at the MANO layer, as illustrated in Figure 2.1.

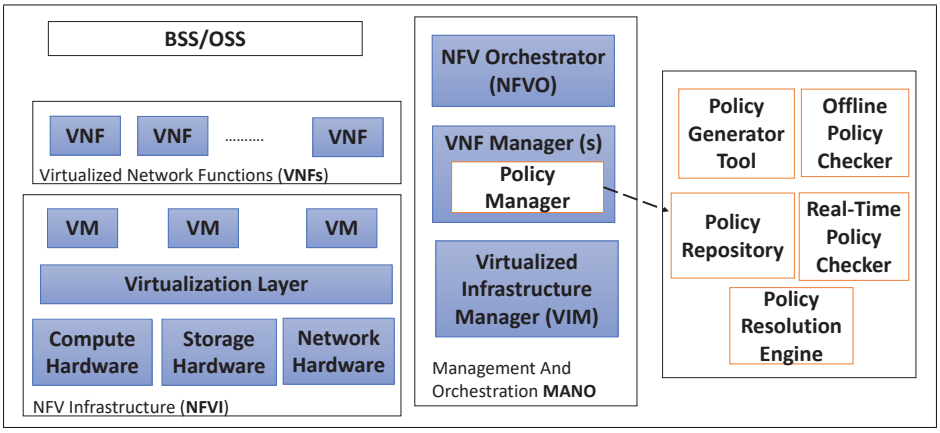


Figure 2.1 Proposed policy manager integrated in NFV architecture

2.5.1 Architecture

As shown in Figure 2.1, the proposed Policy Manager is based on five key components:

a. Policy Generator Tool: Generates an XML policy file containing all the affinity and anti-affinity placement, scaling and migration policies.

b. Policy Repository: Acts as a database and contains all policies that are validated and free of conflicts and redundancy.

c. Offline Policy Checker: Provides a policy detection mechanism to check and validate the policies obtained before the instantiation of SFC network service.

d. Real-Time Policy Checker: Provides a continuous policy detection, where a new affinity or anti-affinity policy is added to the deployed SFC service.

e. Policy Resolution Engine: Provides an automated resolution for the conflicts and redundancies between policies that are detected by the offline or real-time policy detection tools.

2.5.2 Solution Overview

The overall architecture of our solution is depicted in Figure 2.2. Once a new SFC request with a set of affinity and anti-affinity policies (specified by service providers and customers) is received, the Policy Manager obtains all the policies attached to the service request, and an XML-based policy file is generated by the Policy Generator Tool (Figure 2.2, Step 1). The policy file contains the placement, scaling and migration policies translated from their natural language into XML. Each policy P in the policy file could be represented by P= { Id, $PolicyType$, $type$, $globalPriority$, $localPriority$, $Targets$, $Conditions$}. Next, we explain each of P's parameters in detail:

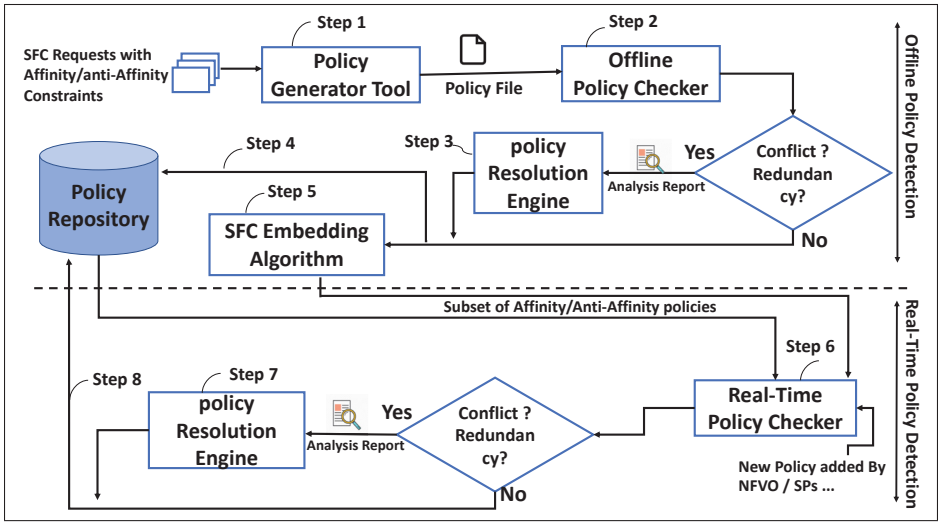


Figure 2.2 High-level overview of the proposed policy manager framework

- *Id* is the policy identifier (e.g., $p_1, p_2, \cdots, p_n$);
- *PolicyType* represents the type of actions for which the policy is created, such as placement, horizontal scaling, vertical scaling, or migration;
- *type* is the type of the underlying policy, which can be either affinity or anti-affinity;
- *globalPriority* defines the global priority of the policy (e.g., $Critical$, $High$, $Medium$, Low, *VeryLow*); the global priority is used to determine the priorities of policies' source. For example, in some cases, the policies defined by the service provider have higher priorities than policies defined by the customer;
- *localPriority* determines the local priority of policies (e.g., $Critical$, $High$, $Medium$, Low, *VeryLow*) defined by the source itself (e.g., service provider, customer);
- *Targets* are a set which depicts the VNF instance, VNF type, virtual path and/or location specification (i.e., *Targets ={FunctionNodes, FunctionTypes, Connections, Location, LocationType}*);
- $Conditions = \{Condition_1, Condition_2, \cdots, Condition_n\}$ are a set of restrictions to the policy actions (i.e., scaling and migration), such as the maximum/minimum number of VNF instances, the migration type, CPU utilization threshold values, etc.

The policy file then undergoes an offline policy check. This step is important to detect any conflict and/or redundancy between the affinity and anti-affinity policies described in the policy file (Figure 2.2, Step 2). The output of the offline policy checker would be an analytical report comprising the detection process results in terms of the constraints that are redundant and/or that conflict with one another. These conflicting and redundant policies are then communicated to the policy resolution engine, which proposes to automatically resolve the conflicts and redundancies between policies by choosing the policies with the highest priorities to keep, and deleting those with lower priorities (Figure 2.2, Step 3). When all the conflicts and redundancies are resolved, all the approved and validated policies are inserted into the policy repository

as groups (Figure 2.2, Step 4). Then, the SFC embedding process can be launched to map each VNF onto the appropriate physical resource, taking into consideration the affinity and anti-affinity placement constraints specified as inputs to the process (Figure 2.2, Step 5). After the placement is complete, each VNF is continuously monitored by the MANO layer (Yang *et al.* (2016)), and the real-time policy checker is invoked to guarantee that the existing affinity and anti-affinity policies are not violated with the arrival of new policies that may be added by the NFV-Orchestrator or network administrator (Figure 2.2, Step 6). If conflicting and/or redundant policies are detected, the policy resolution engine will resolve them (Figure 2.2, Step 7).

To further help readers understand our approach, an example of a policy file is given in Figure 2.9 in Appendix A. Further, snapshots of the XML Schema Definition (XSD) document that defines the formal description of the XML-based policies are given in Figure 2.10 and Figure 2.11, in Appendix A.

2.5.3 Conflict and Redundancy Detection

In general, Policy $P1$ and Policy $P2$ cause a conflict if both policies belong to the same virtual functionality (e.g., functionNodes, functionTypes, connections) and the same policyType (e.g., placement, horizontal scaling etc.), but with different types (affinity, anti-affinity), different Conditions and/or different Locations. Moreover, $P1$ and $P2$ cause a redundancy if both policies have the same parameters (i.e., if they are bound to the same virtual functionality, policyType, Conditions, type and/or the Location specifications). Let us consider the following set of affinity and anti-affinity policies:

- P_1 = *Affinity (DC, VNF_A, VNF_B)*
- P_2 = *Anti-Affinity (DC, VNF_A, VNF_B)*
- P_3 = *Affinity (AS, VNF_1, VNF_2)*
- P_4 = *Affinity (AS3, VNF_1)*

- P_5 = *Affinity (AS1, VNF_2)*
- P_6 = *Affinity (Horizontal-Scaling, cpu_utilization greater than 80 % , VNF_5)*
- P_7 = *Affinity (Horizontal-Scaling, cpu_utilization greater than 50 % , VNF_5)*
- P_8 = *Affinity (DC,e4, e6)*
- P_9 = *Affinity (DC,e4, e6)*
- P_{10} = *Anti-Affinity (AS3, VNF_1)*

P_1 causes a conflict with P_2 since both policies have the same functionTypes (i.e., VNF_A, VNF_B), the same policyType (i.e., placement), the same locationType (DC), but different types (P_1 is affinity and P_2 is anti-affinity). P_3 conflicts with P_4 and P_5 since P_3 implies that both VNF_1 and VNF_2 should be allocated within the same autonomous system, but each VNF has a separate policy with a different location (i.e., $AS3$ and $AS1$). Similarly, P_6 and P_7 cause a conflict, with both having the same functionNode (VNF_5) and the same policyType ($Horizontal\_Scaling$), but different conditions. P_8 is a replica of P_9, hence, both cause a redundancy. P_4 causes a conflict with P_{10} since both have the same functionNode, policyType and location, but the types are different.

Table 2.1 and Table 2.2 present the rules used to respectively detect the conflicts and redundancy between policies, according to the relation between the detection fields of policies. They will be discussed in greater detail in the following sections.

Table 2.1 Rules of conflict detection

PolicyType	Condition		Result
Placement	($P_1 \cap P_2$ = {functionNodeA, type}) $\wedge$ (P_1.location != P_2.location)	(Rule 1)	P_1 cause a conflict with P_2
	($P_1 \cap P_2$ = {functionNodeA, location}) $\wedge$ (P_1.type != P_2.type)	(Rule 2)	
	($P_1 \cap P_2$ = {functionNodeA, functionNodeB, locationType}) $\wedge$ (P_1.type != P_2.type)	(Rule 3)	
	($P_1 \cap P_2$ = {type}) $\wedge$ (P_1.location != P_2.location)	(Rule 4)	P_1 cause a conflict with P_2 and P_3
	($P_1 \cap P_2$ = {location}) $\wedge$ (P_1.type != P_2.type)	(Rule 5)	
Horizontal Scaling, Vertical Scaling , Migration	($P_1 \cap P_2$ = {functionNodeA, PolicyType, type}) $\wedge$ (P_1.conditions != P_2.conditions)	(Rule 6)	P_1 cause a conflict with P_2
	($P_1 \cap P_2$ = {functionNodeA, PolicyType, condtitions}) $\wedge$ (P_1.type != P_2.type)	(Rule 7)	

Table 2.2 Rules of Redundancy detection

PolicyType	Condition		Result
Placement	$P_1 \cap P_2$ = {functionNodeA, type, location}	(Rule 8)	P_1 replica of P_2
	$P_1 \cap P_2$ = {functionNodeA, functionNodeB, type, locationType}	(Rule 9)	
Horizontal Scaling, Vertical Scaling , Migration	$P_1 \cap P_2$ = {functionNodeA, PolicyType, type, condtitions}	(Rule 10)	

2.5.3.1 Offline Policy Detection

Our offline policy checker mechanism is proposed to check and validate the proposed policies before the instantiation of the requested SFC service. The actions involved in the offline detection process as follows:

The first step is to **parse** the generated XML policy file. Then, the policy detection fields (i.e., Id, PolicyType, priorities, type, FunctionNodes, FunctionTypes, Connections, Conditions, Location, and LocationType) are **extracted** and **stored** in *CDB*. The Constraints Detection Base (*CDB*) is created to store the policy information (e.g., Id, PolicyType, priorities etc.) used by the detection mechanisms. Each policy will have a record with detection fields to prevent the offline checker algorithm from getting the information from the XML file more than once.

Thereafter, a set of groups is created, and each policy in the *CDB* is allocated to one of these groups. Then, for each of the generated groups, policy **detection and analysis** processes are launched to identify the conflicting and redundant policies by analyzing the relationships between the detection fields. An **analysis report** is then generated to store the conflicting and redundant policies.

It is worth mentioning that the grouping step is very important since it contributes greatly to minimizing the number of comparisons that have to be done, thus reducing the overall detection process execution time. By grouping, we eliminate any non-required comparisons between policies. For example, returning to the definition of conflict and redundancy detection in Table 2.1 and Table 2.2, it is impossible to have a conflict and/or redundancy between placement policies and scaling policies. Hence, the policies will be classified according to the policyFormat, VNF type, VNF nodes and Policy Type values. Policies belonging to the same VNF

type, VNF nodes and Policy Type will be mapped to a single group. The affinity and anti-affinity policies proposed for the virtual links will be grouped according to the pertaining VNF nodes interconnected by the virtual link. Figure 2.3 illustrates an example of such a grouping process. Only policies relevant to the same group are compared with each other.

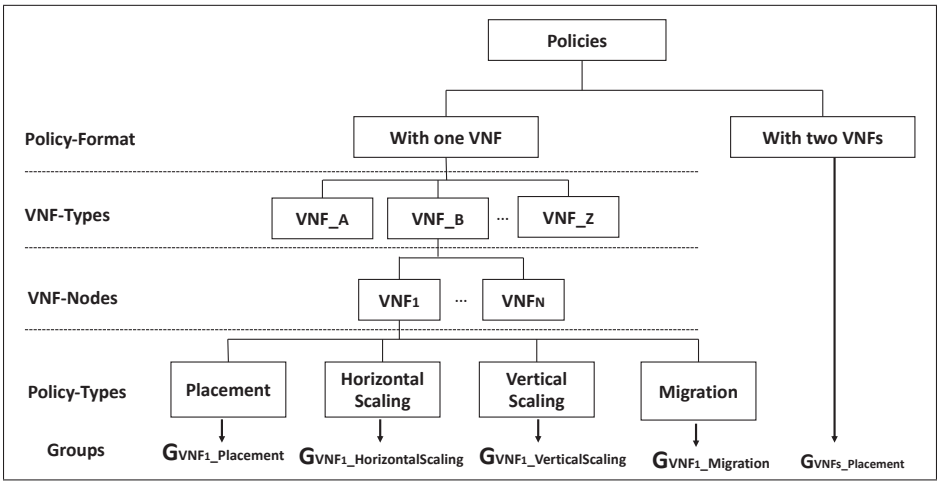


Figure 2.3 Grouping process

A. Algorithms Design

Algorithm 2.1 Offline Policy Checker Algorithm

Input: XML file f containing the policies p.
Output: Analysis report containing conflicting and redundant sets.
procedure OFFLINECHECKER
    Use SAX parser to parse f
    Extract relevant detection fields, i.e., *id*, *PolicyType*, *type*, *priorities* and save them into an array called *CDB*.
        Execute the grouping process using Algorithm 2.2 to obtain the sets $G_{vnfID\_PolicyType}$ and $G_{VNFs\_Placement}$ of groups G.
        for each group $g \in G$ **do**
            Execute the policy analysis using Algorithm 2.4.
        end for
        if group $g \in G_{VNFs\_Placement}$ **then**
            Execute the placement policy analysis using Algorithm 2.5.
        end if
end procedure

Algorithm 2.1 shows the pseudocode of the offline conflict and redundancy detection mechanism. The input of the algorithm is the generated XML policy file containing the VNF placement, scaling and migration policies, and the output is a text file containing the conflicting and redundant policies sets. The first step is to parse the XML file using the SAX method. The main motivation behind choosing SAX as a parser lies in the minimized overhead entailed by such an event-based parser as compared to other parsing approaches (e.g., DOM). Then, the relevant detection fields (e.g., Id, PolicyType, type, FunctionNodes, etc.) are extracted and stored into a detection base called *CDB*.

Next, the grouping algorithm described in Algorithm 2.2 is applied to generate a set of groups ($G_{vnfID\_PolicyType}$ and $G_{VNFs\_Placement}$), each containing a set of relevant policies. For each group obtained in Algorithm 2.2, the Policy Analysis process described in Algorithm 2.4 is performed to detect conflicting and redundant policies. Additionally, the policy analysis process described in Algorithm 2.5 is only performed for the group of type $G_{VNFs\_Placement}$ to extract the set of conflicting and redundant policies among the affinity and anti-affinity placement policies which pertaining to more than VNF.

Algorithm 2.2 Grouping Algorithm

Input: set of VNF Nodes
Input: *CDB* containing the detection fields id, $PolicyType$, $FunctionNodes$, etc.
Output: Set of groups containing a list of policies each.
procedure GROUPING
    for each node $vnfID \in VNF - Nodes$ **do**
        Create a group $G_{vnfID\_PolicyType}$ for each defined PolicyType
    end for
    Create a group $G_{VNFs\_Placement}$ for policies that include more than one VNF
    for each policy $p \in CDB$ **do**
        if p pertains to the VNF-node of group $G_{vnfID\_PolicyType}$
            $G_{vnfID\_PolicyType} = G_{vnfID\_PolicyType} \cup p$
        if p pertain to more than one vnf-node
            $G_{VNFs\_Placement} = G_{VNFs\_Placement} \cup p$
    end for
end procedure

Algorithm 2.2 describes the grouping process. The algorithm takes as inputs a set of VNF-Nodes, and the *CDB* containing the detection fields, and outputs are a set of groups, each containing a list of relevant policies. In the first step of the algorithm, a group for each VNF-node and each PolicyType is created (for example, VNF1_Placement, VNF1_HorizontalScaling). Similarly, a group belonging to the placement policies which are not limited to one VNF-Node is created, furthermore, to the placement policies that have the following format: *Affinity (LocationTypeA, B, C)* or *Anti-Affinity (LocationTypeA, B, C)*. Next, each policy contained in the *CDB* is assigned to a group that is pertaining to.

In Algorithm 2.3, the relationships between the placement, scaling and migration policies are analyzed. The inputs to the algorithm are two policies P_x, P_y and the *CDB* containing the detection fields, and the output is a binary string $relation$ which stores the relationships between policies in binary format by comparing the corresponding detection fields (i.e., 1 in case a relationship exists and 0, otherwise).

Algorithm 2.4 describes the affinity and anti-affinity policy analysis needed to detect the potential conflicts and redundancy issues. The input to this algorithm is a group containing a subset of policies, and the outputs are two sets, namely, a $conflict\_set$ and $redundant\_set$, which contain the policies that are determined to have conflict and redundancy issues. To retrieve the relationships between the policies' detection fields (e.g., policies that have the same conditions), Algorithm 2.3 is called. The relationships are then analyzed according to the rules defined in Table 2.1 and Table 2.2. And if the relationships are of the forms 10100 ($Rule$ 1) (i.e., policies having the same functionNodeA and type, but different locations. Whereas functionNodeB and locationType relationships are "0" since the fields are *null*), 10010 ($Rule$ 2) (i.e., policies having same functionNodeA and location, but having different types), 11001 ($Rule$ 3) (i.e., policies having the same functionNodeA, functionNodesB and locationType, but different types), 1110 ($Rule$ 6) (i.e., policies having the same functionNodeA, policyType, and type but different conditions), or 1101 ($Rule$ 7) (i.e., policies have the same functionNodeA, policyType, and conditions but with different types), then the policies are considered conflicting, and are added to $conflict\_Set$.

Algorithm 2.3 Policy Relation

```
1:  Input: two policies P_x and P_y.
2:  Input: CDB containing the detection fields id, PolicyType, FunctionNodes, etc.
3:  Output: A binary string relation identify the relationship among the policies.
4:  procedure POLICYRELATION
5:      if P_x.PolicyType == "Placement" then
6:          if P_x.functionNodeA == P_y.functionNodeA then
7:              relation.add(1)
8:          else
9:              relation.add(0)
10:         end if
11:         if P_x.functionNodeB == P_y.functionNodeB then
12:             relation.add(1)
13:         else
14:             relation.add(0)
15:         end if
16:         if P_x.type == P_y.type then
17:             relation.add(1)
18:         else
19:             relation.add(0)
20:         end if
21:         if P_x.location == P_y.location then
22:             relation.add(1)
23:         else
24:         relation.add(0)
25:         end if
26:         if P_x.locationType == P_y.locationType then
27:             relation.add(1)
28:         else
29:             relation.add(0)
30:         end if
31:     else
32:         if P_x.functionNodeA == P_y.functionNodeA then
33:             relation.add(1)
34:         else
35:             relation.add(0)
36:         end if
37:         if P_x.policyType == P_y.policyType then
38:             relation.add(1)
39:         else
40:             relation.add(0)
41:         end if
42:         if P_x.type == P_y.type then
43:             relation.add(1)
44:         else
45:             relation.add(0)
46:         end if
47:         if P_x.conditions == P_y.conditions then
48:             relation.add(1)
49:         else
50:             relation.add(0)
51:         end if
52:     end if
53: end procedure
```

Algorithm 2.4 Policy Analysis

Input: a group g containing policies p.
Output: Set $conflict\_set$ of conflicted rules.
Output: Set *redundant_set* of redundant rules.
procedure POLICYANALYSIS
    for i FROM 0 $\longrightarrow$ g.length -1 **do**
        for j FROM i $\longrightarrow$ g.length **do**
            Execute the policy relation process using Algorithm 2.3 with ($g[i]$, $g[j]$) and store the relations into the binary string *relation*
            if $relation \in \{10100, 10010, 11001, 1110, 1101\}$ **then**
                $conflict\_set = conflict\_set \cup \{g[i], g[j]\}$
            else if $relation \in \{10110, 11101, 1111\}$ **then**
                $redundant\_set = redundant\_set \cup \{g[i], g[j]\}$
            end if
        end for
    end for
end procedure

On the other hand, if the relationships are of the forms 10110 (*Rule* 8) (i.e., policies having the same functionNodeA, type and location), 11101 (*Rule* 9) (i.e., policies having the same functionNodeA, functionNodeB, type and locationType), or 1111 (*Rule* 10) (i.e., policies having the same functionNodeA, policyType, type and conditions), then the policies are considered redundant and are added to *redundant_Set*.

Algorithm 2.5 Placement Policy Analysis

Input: a group g containing policies p.
Output: Set $conflict\_set$ of conflicted rules.
procedure PLACEMENTPOLICYANALYSIS
    for each policy $p \in$ group g **do**
        Create a temporary array t_1 such that $t_1 \leftarrow$ placement group of $[p.FunctionNodeA]$
        Create a temporary array t_2 such that $t_2 \leftarrow$ placement group of $[p.FunctionNodeB]$
        for i FROM 0 $\longrightarrow$ $t_1.length$ **do**
            for j FROM 1 $\longrightarrow$ $t_2.length$ **do**
                Execute the policy relation process using Algorithm 2.3 with ($t_1[i]$, $t_2[j]$) as inputs.
            end for
        end for
        if $relation \in \{00100, 00010\}$ **then**
            $conflict\_set = conflict\_set \cup$ {p, $t_1[i]$, $t_2[j]$}
        end if
    end for
end procedure

A further analysis of the placement policies is carried out in Algorithm 2.5 to detect potential conflicts among the policies of the different VNFs. The input to this algorithm is a group containing a subset of policies, and the output is a set $conflict\_set$ that contains the policies that are detected as having a conflict. First, for each policy of the input group, two temporary arrays are created to hold a copy of the appropriate VNF placement groups. For example, if the policy is related to VNF_1 and VNF_2, then the first temporary group will have a copy of placement group policies bound to VNF_1 (obtained from Algorithm 2.2), and the second one will have a copy of placement group policies bound to VNF_2. Next, Algorithm 2.4 is called to analyze the relationships. If the relationships are of the form 00100 $(Rule\ 4)$ (i.e., policies having the same type, but different location), then the policies are deemed to contain conflicts. For example, *Affinity (AS, VNF_A, VNF_B)*, textitAffinity (AS1234, VNF_A) and *Affinity (AS5678, VNF_B).* Similarly, if the relationships are of the form 00010 $(Rule\ 5)$ (i.e., policies have the same locations but with different types). For example, *Affinity (AS, VNF_A, VNF_B)*, *Affinity (AS1234, VNF_A)* and *Anti-Affinity (AS1234, VNF_B)*, then the policies are deemed to contain conflicts. Such these policies are added to $conflict\_Set$.

2.5.3.2 Real-Time Policy Detection

Algorithm 2.6 is invoked to provide a continuous policy detection mechanism. The objective of the real-time policy checker is to prevent conflict and redundancy issues between new and existing policies in real time. Specifically, network administrators or NFV-Orchestrators may need to add new policies in response to the changes that occur at the virtual network level. Before this, the real-time policy checker (Algorithm 2.6) should be invoked to ensure that the new policies should not cause any conflict and redundancy issues.

Note, however, that the real-time policy checker will not compare the new policy with all the existing policies in the policy repository. In fact, only a subset of policies which are bound to the same VNF node and policy type will be selected from the policy repository for comparison. For example, if the new policy pertains to VNF_1, and pertains to horizontal scaling, only the policies pertaining to VNF_1 and relating to horizontal scaling will be extracted from the

policy repository. This significantly reduces the overhead of the real-time policy validation process.

Algorithm 2.6 Real-time policy checker

```
1:  Input: A new Policy P_x.
2:  Input: A group G with a list P of policies.
3:  Output: Analysis Report with conflicted and redundant rule sets.
4:  procedure REALTIMECHECKER
5:      for each policy P in group G do
6:          Execute the policy relation process using Algorithm 2.3 with (P, P_x) as inputs.
7:          if relation ∈ {10100, 10010, 11001, 1110, 1101} then
8:              conflict_set = conflict_set ∪ {p, P_x}
9:          else if relation ∈ {10110, 11101, 1111} then
10:             redundant_set = redundant_set ∪ {p, P_x}
11:         end if
12:     end for
13: end procedure
```

2.5.3.3 Policy Resolution Engine

Algorithm 2.7 describes how the resolution engine solves conflicts and redundancy between policies. The algorithm takes as inputs the analysis report containing the conflicting and/or redundant policy sets detected by the offline policy checker in Algorithm 2.1 or real-time policy checker in Algorithm 2.6, and the *CDB* containing the detection fields of each policy; the outputs are the validated policies which are free of conflicts and redundancies. The Resolution engine resolves any conflicts and redundancies by comparing the priority field values. In the first step of the algorithm, we start by breaking down the analysis report and analyzing each statement; each single statement identifies either a conflict or redundancy between policies detected by the policy checkers. Then, from *CDB*, we get the $globalPriority$ and $localPriority$ values for each policy. As described above, the $globalPriority$ used to determine the priorities of the policies' source (for example, policies defined by customers are more important than policies defined by service providers, or vice versa), and $localPriority$ is used to identify the local priority of each policy defined by the source itself. To decide which policy should be

removed, the resolution engine algorithm attempts to resolve the conflicts and redundancies by:

- Comparing the *globalPriority* values. If they are not equal, then the policy with a lower *globalPriority* will be deleted (if this policy is removed, the conflict or redundancy should be resolved).
- If the *globalPriority* values are equal, the resolution algorithm resolves the conflicts and redundancies by comparing the *localPriority* values. The policy with the lower *localPriority* will then be deleted.
- If there are cases where both the *globalPriority* and *localPriority* values are identical, the resolution engine requests that the policy author decide which policy to remove to resolve the conflicts and redundancies.

Algorithm 2.7 Policy Resolution Engine

```
1:  Input: Analysis report A containing the conflicting and redundant sets.
2:  Input: CDB containing the detection fields id, PolicyType, globalPriority, localPriority, etc.
3:  Output: Policies are free of conflicts and redundancies.
4:  procedure POLICYRESOLUTION
5:      for each statement S in A do
6:          for each Policy P in S do
7:              Get the globalPriority Gp
8:              Get the localPriority Lp
9:          end for
10:         if Gp values are not identical then
11:             Delete P that has lower Gp
12:         else if Gp values are identical then
13:             Delete P that has lower Lp
14:         else if both Gp, Lp are identical then
15:             inform the policy author i.e., customer, provider to decide which policy to remove
16:         end if
17:     end for
18: end procedure
```

2.6 Implementation and Evaluation

In this section, we further discuss the performance evaluation of conflicts and redundancy detection mechanisms among the affinity and anti-affinity policies. We implemented the policy detection tools in Java. Our policy detection mechanism consisted of two components: the offline policy checker module and the real-time policy checker module. Each module is described in detail in Section 2.5.3.1 and Section 2.5.3.2. Our experiments were evaluated on an Intel (R) core (TM) i7-7700 CPU @ 3.60 GHz with 16 GB RAM. All the experiments were repeated 10 times and we report the average values.

Unfortunately, and to the best of our knowledge, none of the studies in the literature review proposes a model for detecting the conflicts and redundancies among NFV placement and VNF life cycle management policies. Hence, there is no baseline for use in comparing the performance of our algorithms. Therefore, we conducted two sets of experiments, the first to assess the efficiency and the performance of the offline conflict and redundancy policy checker algorithm, and the second, to assess the performance of the real-time policy checker algorithm. We evaluated our algorithms by computing the time complexity and studied the affecting factors.

2.6.1 Evaluation of the Offline Policy Checker algorithm

A. Complexity Analysis

Let R be the total number of policies, M the number of VNF-Nodes, M the total number of groups obtained from Algorithm 2.2, and N the number of policies in a single group. In Algorithm 2.1, the computational time complexity could be represented by $T_{\text{offline}} = t_{\text{parsing}} + t_{\text{extract}} + t_{\text{grouping}} + t_{\text{analysis}}$, where:

t_{parsing}: is the complexity of *Line 4* with the SAX parser function running, and the SAX parser depends on the structure and size of the XML policy file; hence, $t_{\text{parsing}} = C_{\text{policyFile}}$.

t_{extract}: is the complexity of *Line 5*, running R times to extract and store the detection fields of each policy; hence, $t_{\text{extract}} = O(R)$.

t_{grouping}: is the time complexity of Algorithm 2.2 invoked by *Line 6*. In Algorithm 2.2, from *Line 5 - 7*, one "for" loop runs M times to create a set of groups for each VNF-Node, and from *Line 9 - 14* a "for" loop runs R times to allocate each policy to the appropriate group. Therefore, t_{grouping} runs in $O(M+R)$.

t_{analysis}: is the time complexity from *Line 7 - 12*, where Algorithm 2.4 and Algorithm 2.5 are invoked. Each group runs Algorithm 2.4, in Algorithm 2.4, from *Line 5 - 12*, we have nested "for" loops and Algorithm 2.3 runs. Algorithm 2.3 only has one "if" statement without loops, hence, its complexity is $O(1)$, whereas the time complexity of the nested loop is $O(N^2)$. Therefore, the time complexity of Algorithm 2.4 is $O(N^2)$. In Algorithm 2.5, nested triple "for" loops are running. Therefore, the total time complexity of Algorithm 2.5 is $O(N^3)$. t_{analysis} is $O((LN^2)+N^3)$

Accordingly, $T_{\text{offline}} = O(max(C_{\text{policyFile}}, R, (M+N), (LN^2+N^3)))$. Therefore, in the worst case, the offline policy checker algorithm runs in $O(N^3)$. This time complexity includes all the aforementioned steps (i.e., grouping, extracting, analysis).

B. Affecting Factors

In this section, we will evaluate and study the main factors that may have an impact on policy detection algorithms.

1. Impact of number of policies in VNF-groups

As discussed for the offline policy checker algorithm, all the policies will be classified into groups, and the policy analysis function invoked for each group.

To evaluate the impact of the group size (i.e., number of policies in a single group), we used a fixed number of 10 VNFs (i.e., the number of requested VNFs per one SFC request), and

the number of the policies was varied as follows: 40, 80, 160, . . . , 640. These policies were uniformly distributed among the VNF groups, and so the number of policies of each VNF group varied as follows: 4, 8, 16, . . . , 64. It can be seen from Figure 2.4 that the detection time of the policy analysis function grows exponentially by increasing the number of policies in a VNF group.

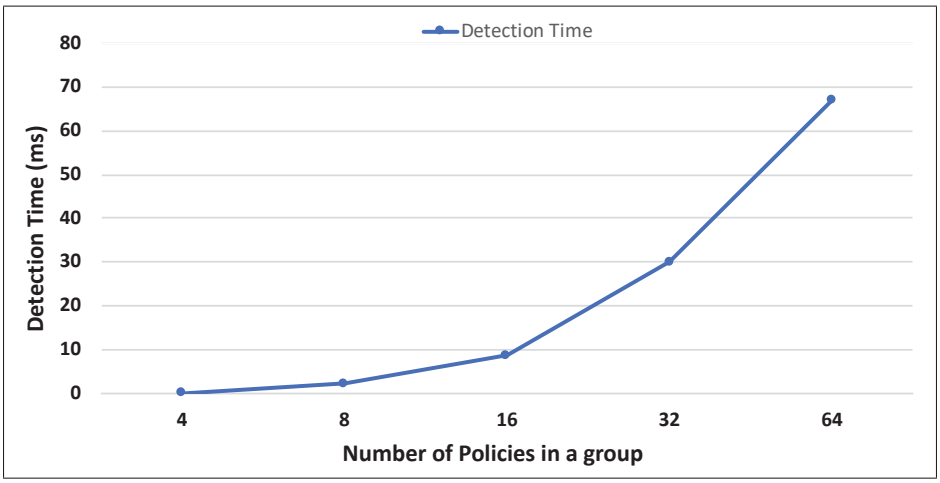


Figure 2.4 Impact of number of policies in a group

2. Impact of the Non-Linear SFC

The SFC may be represented in a linear or non-linear graph. A non-linear SFC usually adds an extra overhead to NFV-related algorithms. To evaluate the impact of non-linear SFCs, we conducted an experiment with a fixed number (10) of VNF nodes, and the number of the policies was varied as follows: 40, 80, 160,. . . , 640. These policies are proposed for a linear SFC and non-linear SFC. It can be seen from Figure 2.5 that the offline policy detection was not affected by the SFC representation.

3. Impact of policy format

As discussed in Section 2.4, the affinity and anti-affinity placement policies have two formats. The first policy format pertains to one VNF (Section 2.4, Placement-oriented policies of type

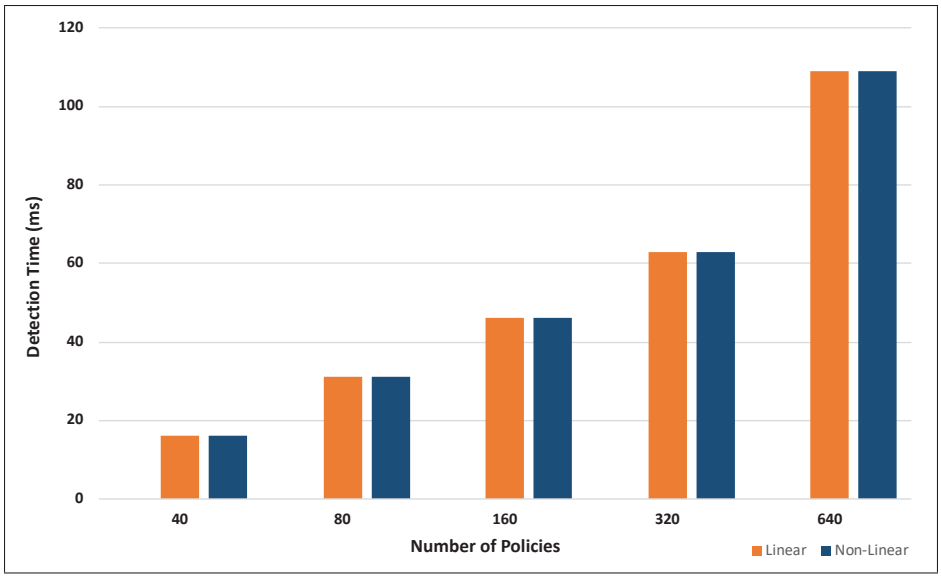


Figure 2.5 Impact of SFC representation

Affinity (LocationA, B) while the second policy format pertains to two VNFs (Section 2.4, Placement-oriented policies of type *Affinity (LocationTypeA, B, C)*.

To evaluate the impact of the policy format, we used a fixed number of 5 VNFs, and the number of placement policies was varied as follows: 15, 25, 35, 45. We conducted test experiments using two different scenarios. In the first scenario, the number of policies pertaining to one VNF was higher than the number relating to two VNFs (e.g., 10 placement policies relating to one VNF and 5 placement policies relating to two VNFs). In the second scenario, the number of policies relating to two VNFs was higher than the number relating to one VNF, as shown in Table 2.3 (e.g., 10 placement policies relating to two VNFs and 5 relating to one VNF). As depicted in Figure 2.6, the algorithm takes a longer time to detect conflicts and redundancies in Scenario 2 than in Scenario 1, even though the total numbers of policies are equal. Indeed, in Algorithm 2.1, the placement policies with more than one VNF involve Algorithms 2.4 and Algorithm 2.5, while for only one VNF, it involves only Algorithm 2.4.

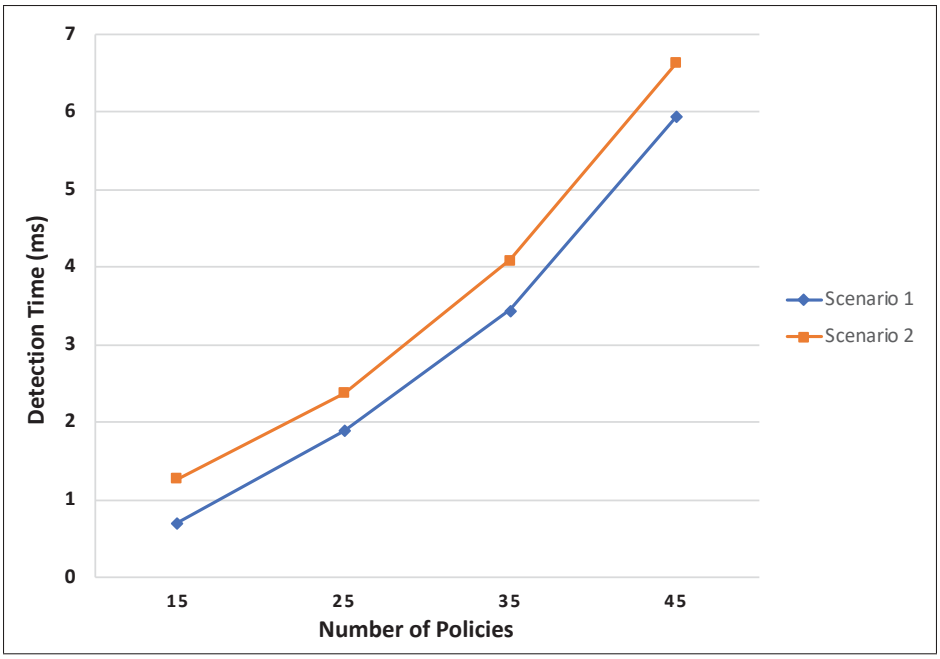


Figure 2.6 Impact of policy format

Table 2.3 Overview of the evaluation parameters to assess the impact of policy format.

Scenario	Number of Policies	Placement policy concerning	
		One VNF	Two VNFs
Scenario 1	15	10	5
	25	15	10
	35	20	15
	45	25	20
Scenario 2	15	5	10
	25	10	15
	35	15	20
	45	20	25

C. Evaluation of combined impact of the relevant factors

To evaluate the combined impact of all the factors discussed and assess the offline policy detection performance, we used a fixed number of 20 VNF nodes, for cases when 5 out of 10 policies cause conflicts or redundancies. The policies were uniformly distributed among the affinity and anti-affinity formats described in Section 2.4, and uniformly distributed among the

relevant VNF groups. Figure 2.7 shows an exponential increase in the processing time as the number of policies is increased. At 200 policies, the offline detection algorithm takes 0.047 s; at 1000 policies, it takes 0.14 s., and at 2000 policies, it takes 0.543 s to complete the detection process. The measured processing time includes the time spent on parsing, storing the detection fields into the CDB, and grouping the policies and detection steps.

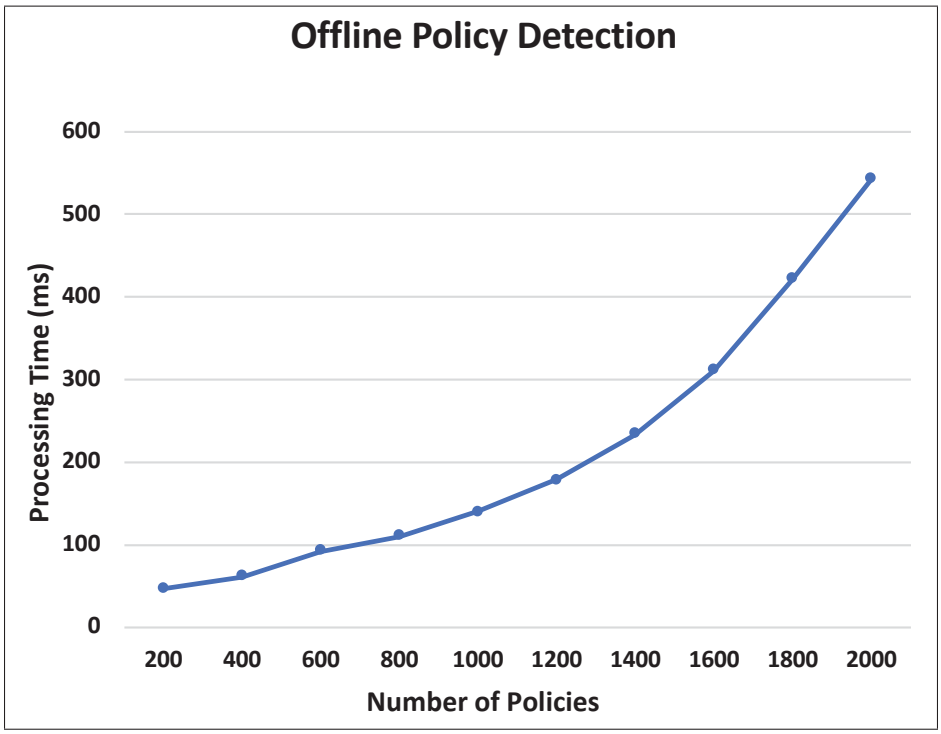


Figure 2.7 Performance evaluation of the Offline policy checker

2.6.2 Evaluation of the Real-Time Policy Checker algorithm

A. Complexity Analysis

Let N be the number of policies in a group. In Algorithm 2.6, from *Line 5 - 12*, there is one "for" loop which runs N times, in *Line 6*, Algorithm 2.3 runs in $O(1)$. As a result, the real-time policy checker algorithm runs in $O(N)$.

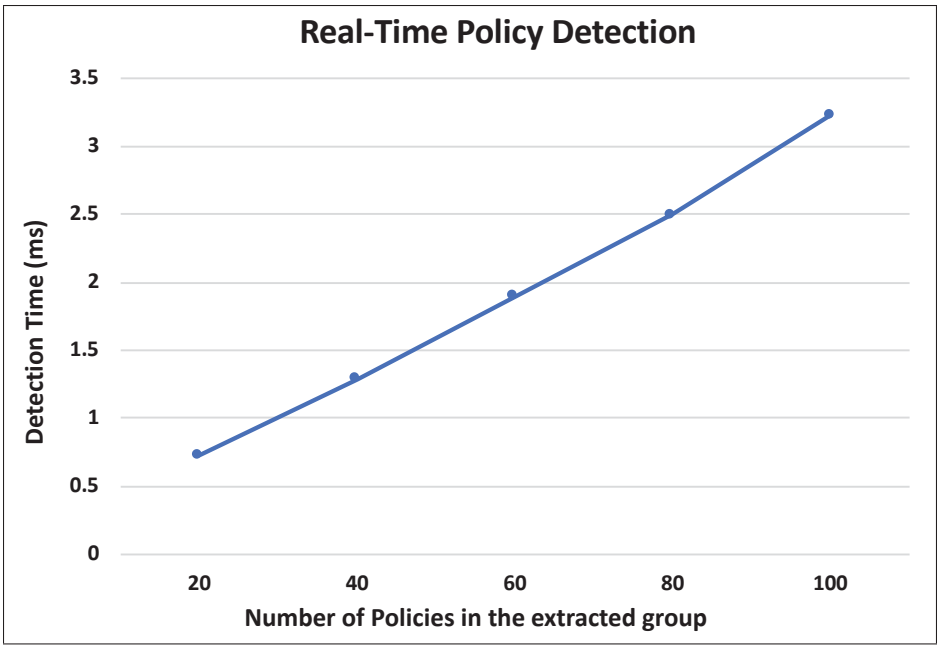


Figure 2.8 Performance evaluation of the Real-time policy checker

B. Evaluation

To assess the performance of the real-time policy checker algorithm, the number of policies extracted from the policy repository was varied as follows: 20, 40, 60, . . . , 100. Figure 2.8 shows the detection time of the real-time policy checker algorithm when the newly added policy causes conflicts or redundancies with 2 out of 10 policies. It can be seen that the real-time detection time grows linearly with the number of policies extracted.

Table 2.4, presents a summary with the factors that may impact the offline and real-time policy detection.

2.7 Conclusion and Future work

We have proposed a policy manager framework for the NFV-MANO architecture. Through this framework, we add the ability by Service Providers (SPs) and customers to add a set of

Table 2.4 Overview of the factors affecting the detection mechanism.

Factors	Detection mechanism	
	Offline Policy Detection	**Real-Time Policy Detection**
Total Number of Policies	Yes	No
Number of policies in a Group	Yes	Yes
SFC representation (Linear, Non-Linear)	No	No
Policy Format (One or Two VNF)	Yes	No
Policy Type (Placement, Horizontal,etc)	No	No
Underlying type of Policy (Affinity, Anti-Affinity)	No	No

affinity and anti-affinity policies for VNF life cycle operations, such as placement, scaling and migration. The framework mainly provides offline and real-time conflicts and redundancy policy detection tools, as well as an automated resolution mechanism to resolve conflicts and redundancies without network administrator assistance.

To improve the efficiency and the performance of the policy detection tools, we classify the policies into a set of groups; only the policies that relate to the same group are compared by the detection algorithms. Our experimental results show that the proposed policy detection tool could rapidly identify and detect the conflicts and redundancies among policies. We were able to enhance the computational time complexity of the real-time policy checker algorithm with reference to the offline policy detection. In the offline detection mechanism, the time is less sensitive since the network service is not yet deployed, whereas when the network service is deployed, and the real-time detection is launched, time becomes very critical, and the detection process should be performed immediately. In the worst case, the offline policy detection runs in $O(N^3)$, whereas the real-time policy detection is $O(N)$.

For future work, we will extend our approach to handle more than one SFC request and detect conflicts and redundancies among their policies, especially when the SFC requests have a set of similar policies and cause similar conflict and redundancy issues. Furthermore, we will further study the conflicts and redundancy detection for a new type of policy involving business rules.

Acknowledgement

This work has been supported by Ericsson Canada and the Natural Sciences and Engineering Research Council of Canada (NSERC).

APPENDIX A. THE PROPOSED XML TEMPLATES

In this appendix, to further help readers understand our approach, an example of an XML policy file is given in Figure 2.9. Moreover, Figure 2.10 and Figure 2.11 show how the affinity and anti-affinity policies are defined using XML Schema Definition (XSD)

```
<?xml version="1.0" encoding="UTF-8"?>
        <PolicySet>
                <Policy Id="P1" type="Affinity" globalPriority="Critical" localPriority="Medium">
                        <PolicyType>Placement</PolicyType>
                        <Tragets>
                                <functionNodeA>VNF_1</functionNodeA>
                                <location>PN_A</location>
                        </Tragets>
                </Policy>
                <Policy Id="P2" type="Affinity" globalPriority="Critical" localPriority="High">
                        <PolicyType>Horizontal-Scaling</PolicyType>
                        <Tragets>
                                <functionNodeA>VNF_3</functionNodeA>
                        </Tragets>
                        <Conditions>
                                <conditionA>min_num_ofInstances: 4</conditionA>
                                <conditionB>max_num_ofInstances: 16</conditionB>

                        </Conditions>
                </Policy>
                <Policy Id="P3" type="Anti-Affinity" globalPriority="High"
localPriority="Critical">
                        <PolicyType>Vertical-Scaling</PolicyType>
                        <Tragets>
                                <functionNodeA>VNF_5</functionNodeA>
                        </Tragets>
                </Policy>
                <Policy Id="P4" type="Affinity" globalPriority="High" localPriority="High">
                        <PolicyType>Placement</PolicyType>
                        <Tragets>
                                <functionTypeA>VNF_A</functionTypeA>
                                <functionTypeB>VNF_B</functionTypeB>
                                <locationType>AC</locationType>
                        </Tragets>
                </Policy>
                <Policy Id="P5" type="Anti-Affinity" globalPriority="Critical" localPriority="Low">
                        <PolicyType>Placement</PolicyType>
                        <Tragets>
                                <connectionC>e1</connectionC>
                                <connectionD>e2</connectionD>
                                <locationType>DC</locationType>
                        </Tragets>
                </Policy>
        </PolicySet>
```

Figure 2.9 Example of XML-generated policy file

```
<xs:element name="Policy" maxOccurs="unbounded" minOccurs="0">
 <xs:complexType>
   <xs:sequence>
   <xs:element name="PolicyType"/>
   <xs:element name="Targets">
     <xs:complexType>
     <xs:choice>
       <xs:sequence>
       <xs:choice>
        <xs:element name="locationType" type="LocationType"/>
        <xs:element name="location" type="Location"/>
       </xs:choice>
       <xs:choice>
        <xs:element name="functionTypeA" type="FunctionType"/>
        <xs:element name="functionNodeA" type="NetworkFunction"/>
       </xs:choice>
       <xs:choice minOccurs="0">
        <xs:element name="functionTypeB" type="FunctionType"/>
        <xs:element name="functionNodeB" type="NetworkFunction"/>
       </xs:choice>
      </xs:sequence>
      <xs:sequence>
       <xs:element name="connectionC" type="virtualLink"/>
       <xs:choice>
        <xs:element name="connectionD" type="virtualLink"/>
        <xs:choice>
         <xs:element name="locationType" type="LocationType"/>
         <xs:element name="location" type="Location"/>
        </xs:choice>
       </xs:choice>
      </xs:sequence>
     </xs:choice>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 <xs:attribute name="Id" type="xs:string"/>
 <xs:attribute name="type" type="ConstraintType"/>
 <xs:attribute name="globalPriority" type="xs:string"/>
 <xs:attribute name="localPriority" type="xs:string"/>
 </xs:complexType>
</xs:element>
```

Figure 2.10 Formal description of VNF placement policies using XML Schema Definition (XSD)

```
<xs:element name="Policy" maxOccurs="unbounded" minOccurs="0">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="PolicyType"/>
   <xs:element name="Targets">
    <xs:complexType>
     <xs:sequence>
      <xs:choice>
       <xs:element name="functionTypeA" type="FunctionType"/>
       <xs:element name="functionNodeA" type="NetworkFunction" />
      </xs:choice>
      <xs:choice minOccurs="0">
       <xs:element name="functionTypeB" type="FunctionType"/>
       <xs:element name="functionNodeB" type="NetworkFunction"/>
      </xs:choice>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="Conditions">
    <xs:complexType>
     <xs:sequence>
      <xs:choice>
       <xs:element name="conditionA" type="xs:string"/>
       <xs:element name="conditionB" type="xs:string"/>
      </xs:choice>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 <xs:attribute name="Id" type="xs:string"/>
 <xs:attribute name="type" type="ConstraintType"/>
 <xs:attribute name="globalPriority" type="xs:string"/>
 <xs:attribute name="localPriority" type="xs:string"/>
 </xs:complexType>
</xs:element>
```

Figure 2.11 Formal description of VNF-Management policies using XML Schema Definition (XSD)

CHAPTER 3

MACHINE-LEARNING BASED SIMILARITY DETECTION OF SERVICE FUNCTION CHAINS FOR AGILE SERVICE PROVISIONING

Hanan Suwi[a], Fawaz A. Khasawneh[b], Nadjia Kara[c], Claes Edstrom[d]

[a b c] Department of Software Engineering and IT, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[d] Ericsson Canada,
8275 Route Transcanadienne, Ville Saint-Laurent, Québec, Canada H4S 0B6

Paper submitted to the journal "“IEEE Transactions on Cloud Computing”" in June 2019.
Provisional patent application P78281 filed by Ericsson Patent Unit Canada to the United States Patent and Trademark Office (USPTO) in June 2019

3.1 Abstract

Network Function Virtualization (NFV) has been identified to improve the flexibility and cost-efficiency in network service provisioning. This is achieved by developing standard IT virtualization technology to integrate many network component types onto industry. The main idea of NFV is to replace the dedicated hardware network components with software components that offer the same functionality and run on top of server hardware. Service Function Chain (SFC) which composed out of a set of Virtual Network Functions (VNFs) chained to create services. With virtualization, NFV promises the network operators to drive fast, agile and scalable deployment and managing for the SFC. Although service provisioning is performed rapidly in NFV infrastructure (NFVI) compared to the traditional hardware-based architecture, but there is always a need/room for improvement to make the network increasingly less reluctant to benefit from the maximum potential of this new technology. This has motivated us to propose a novel approach to reduce the time needed for SFC provisioning, which increases the resiliency level in the NFV infrastructure. To meet those needs, a novel managing tool which helps the NFV technology to reduce the time needed for SFC provisioning. This has been accomplished by computing the similarity between any incoming SFC request and the already existing provisioned SFC services. The problem has been mathematically formulated using a machine

learning technique called Paragraph Vector-Distributed Bag of Words (PV-DBOW) and Frequent Sub-graph Mining (FSM) technique. The results show that our proposed algorithm has a significant improvement in terms of the execution time needed for SFC provisioning by occasionally skipping the following two phases: (1) detecting and resolving the conflicts and redundancy between SFC policies. And/or (2) SFC placement and chaining phases needed for provisioning the SFC based on the similarity existence.

Index Terms: Network Function Virtualization, service provisioning, Similarity, PV-DBOW.

3.2 Introduction

In traditional telecommunication networks, service provisioning has traditionally based on deploying dedicated hardware network appliance for each Network Function. To create new network services, network operators often require new hardware appliances and finding the space to setup the network service. Moreover, network operators must continuously operate and purchase new hardware-based appliances to adapt and create new network services, especially that hardware-based appliance need maintenance and may reach end of life. Accordingly, the aforementioned problems are magnified with the huge demand for high quality and scalable network services. In recent years, industrial and academic fields have begun focusing on Network Function Virtualization (NFV) as it can solve the problems and decrease the Operational Expenditures (OpEx) as well as Capital Expenditures (CapEx). NFV can help in ensuring the smooth switch from expensive hardware-based appliance, by shifting network functions (e.g., IDS and Firewall) to software-based components running on a top of commodity hardware as Virtual Network Functions (VNFs) (Li & Chen (2015)). With NFV, the network functions can be instantiated in, or moved to different locations in network service without need to install new hardware components. The sequence of set of VNFs called Service Function Chain (SFC).

However, many challenges still need to be tackled in order to push forward the development of NFV technology from its early stage to a more mature one. This can be achieved by designing

a more effective and efficient service and resource orchestration mechanisms that meet a set of constraints (e.g., affinity/anti-affinity, Service Level Agreement (SLA)), accelerate the deployment of the new SFCs and managing the VNF resources. Orchestration is a quite important challenge to be handled, which guarantees the smooth hosting of VNFs in a sharable infrastructure, with an ultimate goal of decreasing the time needed for SFC provisioning. Designing this kind of orchestration is still a daunting task to achieve (Mijumbi *et al.* (2016)).

In addition, there might be some requirements or restrictions imposed by customers and/or service providers concerning the placement, VNF life-cycle management operations, and routing regarding their Service Function Chain (SFC) requests. As stated in ETSI NFV standard group (EISGI (2014)), for example, anti-affinity placement constraints help in allocating the same SFC's VNFs in diverse physical hosts or different data centers for enhancing their service robustness and resilience. Other requirements concerning legislative, economic, privacy, and efficiency concerns (Bouten *et al.* (2017)) might be required by customers and/or service providers. The main limitation of the SFC provisioning approaches (Tajiki *et al.* (2018)), (Wang *et al.* (2016)), (Jang *et al.* (2017)), (Ye *et al.* (2016)) and the affinity/anti-affinity policy enforcement strategies (Sundararajan *et al.* (2015)), (Pachorkar & Ingle (2013)), (Konstanteli *et al.* (2014)) is the lengthy time they take to accomplish those tasks especially in the medium or large scale NFV infrastructures.

In this paper, we addressed the challenge of designing a novel solution in order to efficiently and effectively place and map SFCs into substrate infrastructures while ensuring rapid provisioning for new incoming SFC requests. For the SFC provisioning to be performed, it should undergo two substantial phases, which are: (1) the conflicts and redundancy detection and resolution algorithms to validate the SFC request and to detect and resolve the conflicts and redundancies among the affinity and anti-affinity policies in SFC request, and (2) the joint placement and chaining algorithms run to select the best location where to map the VNFs into the underlying physical resources. Moreover, for the SFC to be placed a set of interdependent phases should be accomplished which are: network function mapping, service scheduling and traffic routing, which have been proven to be an NP-Hard problem (Alameddine *et al.* (2017)).

To successfully resolve the conflict and redundancy among SFC policies and decide where and how to place and map any new incoming SFC request, an amount of processing time is needed to perform all of these tasks. However, this amount of time could be unsuitable for some sensitive applications (e.g., latency-sensitive and availability-sensitive applications). Thus, a novel approach should be introduced to reduce the time needed for SFC provisioning. Our approach enables to take a decision on whether a new SFC request should go through the aforementioned two phases (conflict/redundancy detection and resolution and placement and chaining) based on the similarity existence between any new incoming SFC and the already deployed SFCs in NFVI. Thus, our approach is embedded inside the NFV Orchestration (NFVO) layer, called Service Similarity Manager tool responsible to find similarity between SFCs. This tool includes five main architectural components, namely, service graph decomposition, service graph translator, service similarity checker, translated service graph repository and similarity rate repository. Each SFC is represented as a directed graph composed of a set of VNFs nodes, virtual links as well as attributes for each VNF node (e.g., number of required CPU cores, memory size and storage capacity) and for each virtual link (e.g., required bandwidth, end-to-end delay and affinity/anti-affinity policies). The SFC graph is translated into a vector representation using Paragraph Vector Distributed Bag of Words (PV-DBOW) model and finally the cosine similarity is computed to find a matching between SFCs. The flow chart of our proposed solution and its architectural components are carefully explained in sections 3.6 and 3.7.

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

- We propose a novel SFC service similarity algorithm that can handle different types of SFCs (different SFC graph linear and non-linear, different SFC size, various attributes);
- We model a similarity manager that can find exact and partial matching between SFCs;
- We define a frequent sub-graph mining algorithm that increases the number of potential sub-service that are common with those composing an SFC request and that can be used to find partial matching between SFCs.

- We design a graph-to-vector translation algorithm that enables generating d-dimensional vector space representation of SFC graphs in order to ease similarity detection between SFCs.
- The proposed similarity checker mechanism help reducing time for SFC provisioning and for policy redundancy and conflict detection resolution.
- The similarity detection strategy can be used as a baseline for resource management and prediction.

The rest of the paper is organized as follows: in section 3.3, related works are discussed, followed by background information about the different techniques that are used in our proposed approach. Problem is defined and formulated in section 3.5. In section 3.6, our proposed approach is described and a description of use cases is presented in section 3.7. An experimental setup and a thorough analysis are performed in section 3.8 and finally our work is concluded in section 3.9.

3.3 Related Works

In this section, we review the recent research works, in which detecting the conflicts and redundancies among policies, SFC placement and chaining algorithms are introduced. In addition, we review the most common studies for discovering the similarities in various fields. defining the fields where the used techniques in our approach have been utilized and finally a sub-section to discuss the findings are introduced.

3.3.1 Policy Conflict and Redundancy

One of the placement policies which have been examined in several studies in cloud computing environment is **affinity/anti-affinity placement policies** (Sundararajan *et al.* (2015)), (Pachorkar & Ingle (2013)). (Konstanteli *et al.* (2014)) recommended a method for obtaining

the best service-to-cloud allocation. The problem formulation also considers metrics including trust, cost, eco-efficiency, as well as affinity and anti-affinity constraints which may be present in service components. (Espling *et al.* (2014)) examine the problem of helping application owners to be involved in service components' placement location through transforming hierarchical graph structures into formalized placement constraints as well as creating a mathematical model which can help in extending the existing placement techniques by supporting service own-controlled instructions.

A heuristic is suggested by (Larsson *et al.* (2011)) which can enable the model selecting suitable virtual machines for migration by considering the affinity and anti-affinity policies. On the other hand, (Macías & Guitart (2014)) outline two policy sets, the first is revenue maximization and the second is client classification as per client affinity and Quality of Service (QoS). In this case, affinity indicates the cloud provider's relationship with client. To reduce the communication overhead, an Affinity-Aware Grouping method was suggested by (Chen *et al.* (2013)) for VMs' Allocation (AAGA). In particular, a grouping method was developed as per the bin packing heuristic algorithm so that physical machines can utilize VM groups.

Furthermore, a heuristic approach was recommended by (Allybokus *et al.* (2018)) that used Integer Linear Program (ILP) formulation to find the most suitable physical location for SFC placement while fulfilling the anti-affinity constraints. Moreover, a new Affinity-Based Allocation (ABA) heuristic approach was devised by (Bhamare *et al.* (2017)) to place the service functions into large scale network infrastructures could be resolved. (Deng *et al.* (2015)) also recommended the VNGuard framework to ensure virtual firewall's efficient provisioning and management using SDN and NFV technologies. Users in VNGuard can efficiently define their security policies which are translated automatically into low-level firewall rules and the best virtual firewall deployment solutions are obtained, and the defined performance objectives are satisfied.

(Zou *et al.* (2018)) addressed the problem of identifying anomalies in SFC policies that are set independently in IoT environments by various policy makers. A composition method is devel-

oped to determine and resolve these anomalies. (Bouten *et al.* (2017)) suggested that location requirements should be embedded into the VNFs' and virtual edges' placement with a set of co-location constraints. They, therefore, suggested a set of affinity and anti-affinity constraints which can be implemented through service providers for defining placement restrictions. A semantic SFC validation method was also defined to enable NFV infrastructure providers to substantiate the constraints set's validity as well as to give the suitable feedback to service providers. This step is necessary for eliminating non-valid SFC requests before they are sent to the placement algorithm, which will help decrease the placement process overhead.

A model was also proposed by (Basile *et al.* (2016)) to determine inter-function anomalies concerning SDN/NFV deployment. The authors incorporated a new component called Policy Analyzer (PA) to the NFV orchestrator to find anomalies and then reconfigure the network. A policy definition framework was put forth in (Figueira *et al.* (2015)) to integrate business rules as well as define resource constraints regarding NFV settings. The policy framework emphasized the services' general orchestration requirements including diverse subsystems. The work further comprehensively assessed policy scope, static compared to autonomic policies, policy conflict detection as well as resolution, global compared to local policies, policies engines' interactions, and policy actions and translations.

3.3.2 SFC Placement and Mapping

Numerous attempts have been conducted in the literature to solve the problem of **SFC placement and mapping**. (Tajiki *et al.* (2018)) put forth a new resource allocation architecture that allows energy-aware SFC embedding in SDN-based networks while taking into consideration a set of constraints such as delay, server utilization, and link utilization. They formulate, the VNF placement problems, VNFs' allocation to flows, as well as flow routing as ILP optimization problems. They define a set of heuristic to find a sub-optimal solutions in timescales appropriate for practical applications.

(Wang *et al.* (2016)) examines a joint approach so that NFV resource allocation (NFV-RA) is jointly optimized considering VNFs forwarding graph embedding, VNFs scheduling and VNFs chaining. The joint NFV-RA approach is developed as a Mixed-Integer Linear Programming (MILP). A heuristic-based algorithm is suggested to find a near optimal solution.

On another hand, (Jang *et al.* (2017)) have developed a multi-objective optimization model to solve the problem of dynamic service function placement. This model enables to enhance the flow rate, and routing as well as to reduce the energy cost for different SFCs.

Other studies have explored the joint topology design as well as mapping problem to reduce the total bandwidth consumption. They have put forth the Closed-loop with Critical Mapping Feedback (CCMF) algorithm to solve this problem (Ye *et al.* (2016)). The CCMF algorithm uses the feedback based on the mapping of SFC's critical sub-topologies to enhance the processes of SFC embedding.

Another MILP optimization model has been formulated by (Hawilo *et al.* (2019)) to solve the problem of VNF placement. The solution focuses on the VNF placement and considers the NFV applications' carrier-grade nature while reducing the end-to-end latency and reliability of SFCs.

In (Alameddine *et al.* (2017)), three inter-connected sub-problems which are the Network Functions (NFs) mapping sub-problem, the service scheduling sub-problem, and the traffic routing sub-problem are jointly solved. This study puts forth an SFC Scheduling (SFCS) formulation that uses interactions between NFs mapping onto VNFs, traffic routing, and service scheduling. In order to solve this jointly related problem, authors propose to decompose it into two main sub-problems, which are called the master and pricing sub-problems and column generation technique is used to iteratively solve it to reach a near-optimal solution in a reasonable time.

An online placement algorithm in data centers is proposed by (Moualla *et al.* (2019)). It enables to find the necessary number of replicas of an SFC and its placement in the data center. It also

enables to balance the load between regarding these replicas according to their availability to prevent failure and reduce downtime in the physical infrastructure.

Two factors for enhancing resource utilization are considered by (Li *et al.* (2018)): the time-varying workloads and the Basic Resource Consumptions (BRCs). This helps in instantiating VNFs regarding Physical Machines (PMs). The authors formulate the VNF placement problem as ILP model to reduce the number of PMs used. They solve this problem by devising a Two-StAgeheurisTic solution (T-SAT). This solution includes two stages. The first stage is a correlation-based greedy algorithm regarding SFC mapping while the second stage is an enhanced adjustment algorithm concerning virtual network function requests for every SFC.

3.3.3 SFC Similarity Detection

Skip-gram techniques have been used in various contexts in cloud environment, ranging from solving the data sparsity problem by (Guthrie *et al.* (2006)), clustering huge data based on the topic by (Zobaed *et al.* (2018)), analyzing sensed data collected from social networks in cloud (Zhang *et al.* (2018)) and extracting knowledge from Chinese customer reviews by (Zhao *et al.* (2015)). In addition, this technique has been used in predicting the future resource usage in fifth Generation (5G) networks (Pérez-Romero *et al.* (2018)), and even to predict the intention of the mobile user for a better experience (Wang (2017)). Other usage was to effectively retrieve the encrypted data in the cloud environment (Zhao & Iwaihara (2017)).

To the best of our knowledge, **we are the first** to find the **similarity** between SFCs for the benefit of rapidly provision an SFC and **none of the studies** in the literature have checked the **similarity** between the received SFC requests and the existing ones when studying the SFC provisioning problem. We also have used word embedding Skip-gram model, PV-DBOW in conjunction with FSM technique to find the similarity between the SFCs, **it is worth mentioning** that **we are the first** to use those **three techniques in conjunction** to find the similarity between SFCs in order to reduce the processing overhead for the embedding as well as policy redundancy and conflict detection for new SFC requests.

3.4 Background

In this section, we describe various models that are used as background for our proposed algorithms.

3.4.1 Affinity and Anti-affinity policy

NFV add the ability to multiple source (e.g., service provider) to attach their custom constraints and policies concerning the placement, and VNF lifecycle management operations (e.g., scaling, migration) to the SFC request based on SLA and network conditions. These policies may be added for efficiency, economic, and privacy issues. For example, to increase the efficiency in a service, an affinity policy is proposed to imply that the VNFs that exchange a lot of traffic may should be co-located close to one another (e.g., within the same data center). An anti-affinity policy may propose when the network operator might want to spread instances of the same VNF across different data centers in order to improve resilience of a service in case of failure. Such these policies related to VNF placement policies; affinity herein means that VNF(s) should co-locate within the same physical resource (e.g., data center, physical server) whereas anti-affinity means that the VNF(s) should not co-locate close to each other and within the same physical resource (Bouten *et al.* (2017)).

3.4.2 Word Embedding Skip-gram Model

Word2vec is one of the modern neural methods, which has been proposed as an efficient neural approach to learn the distributed representation of words using a model called "Skip-gram" (Mikolov *et al.* (2013)). Word2vec is used to embed the words into a low-dimensional vector space, it works based on the rationale that the "words appearing in similar contexts tend to have similar meanings and hence should have similar vector representations" (Narayanan *et al.* (2017)).

Skip-Gram (SG) model is used to find word representations that are useful to predict the surrounding words in a given sentence or document. More precisely, given a sequence of words

$\{w_1, w_2, ..., w_T\}$, the target word w_t whose representation need to be learned and the length of the context window c, the objective of SG model is to maximize the log probability of predicting the context words $\{w_{t-c}, ... , w_{t+c}\}$.

$$\sum_{t=1}^{T} log(Pr(w_{t\text{-}c}, ..., w_{t+c}|w_t)) \tag{3.1}$$

Where w_{t-c}, ... ,w_{t+c} are the context words which appears nearby the target word w_t. The probability Pr(w_{t-c}, ... ,$w_{t+c}|w_t$) is computed as

$$Pr(w_{t\text{-}c}, ..., w_{t+c}|w_t) = \prod_{-c \leq j \leq c, j \neq 0} Pr(w_{t+j}|w_t) \tag{3.2}$$

Furthermore, Pr($w_{t+j}|w_t$) is defined as:

$$Pr(w_{t+j}|w_t) = (e^{w_{t+j} \cdot w_t})/(\sum_{i=1}^{\gamma} e^{w_i \cdot w_t}) \tag{3.3}$$

Where w_{t+j} and w_t are the vectors of words and γ is the vocabulary size (i.e., number of words in the vocabulary).

3.4.2.1 NEGative sampling

Equation 3.3. Is very expensive to compute when the vocabulary γ contains large number of words. Hence, NEGative sampling (NEG) is defined to solve this problem and train the Skip-gram model. NEG randomly selects a small subset of words that are not within the target word's context and defines them as negative samples and updates their embedding in every iteration instead of considering all words in γ. Accordingly, NEG ensures that if a word w appears in the context of another word w', then the vector representation of w is closer to that of w' compared to negative samples (Mikolov *et al.* (2013)).

3.4.3 Document Embedding PV-DBOW Model

Inspired from Word2vec method, Doc2vec is proposed to learn paragraph and document embedding vector representation (Le & Mikolov (2014)). Paragraph Vector-Distributed Bag-of-Words (PV-DBOW) is an instance of skip-gram model used by Doc2vec to predict the embedding vectors of words contained in a whole large document. More specifically, given a set of documents D = $\{d_1, d_2, ..., d_n\}$, the target document $d_t \in D$ whose representation needs to be learned, and a sequence of words $w(d_i) = \{w_1, w_2, ..., w_k\}$ to be sampled from the target document d_t. The training objective of PV-DBOW is to maximize the log probability of predicting the words $w_1, w_2, ..., w_k$ which appear in the document d_t.

3.4.4 Frequent Sub-graphs Mining (FSM)

The goal of frequent sub-graph mining process is to extract all the frequent sub-graphs, in a given data set, whose occurrence (i.e., support) counts more than or equal to a defined threshold value. The main idea behind FSM is to 'grow' candidate sub-graphs, in either a breadth-first or depth-first manner (candidate generation), and then determine whether the identified sub-graphs occur frequently enough in the graph data set for them to be considered as interesting candidates(support counting) (Jiang *et al.* (2013)). Many algorithms have been proposed to extract the frequent sub-graph such as, FFSM, Gaston, gSpan, etc.

A potential problem with many FSM algorithms is that the same candidate subgraph discovered more than once. The gSpan algorithm (Yan & Han (2002)) solves this issue by introducing the "rightmost path" technique to avoid duplicated candidate generation, where a straight path starts from the first vertex to the last one using depth-first search. The algorithm of gSpan extracts the frequent subgraphs one by one using a search tree. It starts with a set of frequent containing only one vertex and then, for each 1-node frequent subgraph, it generates a new subgraph by adding one edge at a time.

3.5 Problem Definition and Formulation

Before describing the problem formulation, we list the acronyms and notations used throughout the paper in Table 3.1 and Table 3.2.

Table 3.1 List of Acronyms

Acronyms	Description
5G	Fifth Generation
AAGA	Affinity-Aware Grouping Allocation
ABA	Affinity-Based Allocation
BRC	Basic Resource Consumptions
CapEx	Capital Expenditures
CCMF	Closed-loop with Critical Mapping Feedback
FSM	Frequent Subgraph Mining
ILP	Integer Linear Program
IoT	Internet of Things
MILP	Mixed Integer Linear Program
NEG	NEGative sampling
NF	Network Function
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFV-MANO	NFV-MAnagement and Network Orchestration
NFVO	NFV Orchestrator
OpEx	Operational Expenditures
PA	Policy Analyzer
PM	Physical Machine
PV-DBOW	Paragraph Vector Distributed Bag of Words
QoS	Quality of Service
SDN	Software Defined Network
SFC	Service Function Chain
SFCS	Service Function Chain Scheduling
SG	Skip-Gram
SLA	Service Level Agreement
TOSCA	Topology and Orchestration Specification for Cloud Applications
T-SAT	Two-StAgeheurisTic solution
VNF	Virtual Network Function

Table 3.2 List of Notations

Notation	Description
NS_{g}	Virtualized network service graphs $\{NS_{\mathrm{g1}}, ..., NS_{\mathrm{gn}}\}$
NS_{gi}	Virtualized network service graph i
V_{vnf}	The set of VNF nodes in SFC graph.
E_{vl}	The set of virtual links connecting two VNFs in SFC graph
A	VNF nodes and virtual links set of attributes $\{A_{cpu}, A_{mem}, A_{storage}, A_{BW}, A_{policy}\}$
$f_v : V_{vnf} \rightarrow A_v$	Function to assign a set of attributes for VNF nodes
$f_e : E_{vl} \rightarrow A_e$	Function to assign a set of attributes for E_{vl}
vl	Virtual link
φ:$V_{\mathrm{vnf}} \rightarrow V'_{\mathrm{vnf}}$	Bijection function between two network services (two sets of VNF nodes)
vnf_i	Virtual network function i
$subNS_g$	Network sub service graph
sup_{subNS_g}	Support of sub service graph
δ	Minimum support threshold
ε: $NS_{\mathrm{g}} \rightarrow R^d$	The embedding (translating) function
$\Phi(NSgj)$	The embedding (translating) matrix for network service graph j
$NScatalog$	The catalog containing all the deployed network services
$Freq\_subNS$	The frequent sub services
$\varepsilon(NS_g)$	The vector representation of NS_g
|*|	The total number of * available. * could be NS_g, $Vvnfs$, $Evls$, A_v or A_e

To formulate our problem, a set of definitions are introduced. A new SFC request is represented as a directional graph which is composed of a set of VNFs along with the virtual links connecting these VNFs to form the required service chaining. The attributes for each VNF and each link are also embedded in the graph. The required CPU cores, memory size and storage capacity attributes are defined for each VNF. On the other hand, attributes such as required bandwidth, end-to-end delay and affinity/anti-affinity constraints are also defined for each link. It is worth mentioning that our proposed algorithm can be easily extended to have more than three attributes per VNF and per virtual link.

Definition 1. Virtualized network service is represented as an attributed-directed graph $NS_g = (V_{vnf}, E_{vl}, A)$, where $V_{vnf} = \{vnf_1, ..., vnf_n\}$ is a set of VNF nodes, $E_{\mathrm{vl}} \subseteq (V_{\mathrm{vnf}} x V_{\mathrm{vnf}})$ is a set of virtual links that connect pairs of VNFs, $E_{vl} = \{vl_1, ..., vl_n\}$ and A is a set of VNF nodes and Virtual links attribute types . We define a function $f_v : V_{\mathrm{vnf}} \rightarrow A_v$ that assign a set of attributes for VNF nodes, each VNF node $vnf \in V_{vnf}$ is associated with three attributes in A; the number

of CPU cores (A_{cpu}), Memory size (A_{mem}) and Storage capacity ($A_{storage}$). Similarly, a function $f_e : E_{\text{vl}} \rightarrow A_e$ is defined to assign a set of attributes for E_{vl}, a virtual link $vl \in E_{vl}$ is associated with three attributes; the required bandwidth (A_{BW}), the end-to-end delay (A_{delay}), and affinity or anti-affinity placement policy (A_{policy}).

To find a matching between two SFCs, we have to check if an Isomorphism (i.e., exact similarity) exists between two SFCs.An isomorphism exists between two SFCs if and only if there is bijection function between the two sets of VNFs. Bijection function represents a one-to-one relationship between the two sets of VNFs and to be satisfied, a set of conditions should be met: (1) the number of VNF nodes and virtual links in both SFCs should be equal; (2) the attributes for each VNF node and virtual link are exactly the same; (3) the same links exist and connect the same VNFs nodes in the two Service graph. This yields to definition 2.

Definition 2. Given two NS graphs denoted by $NS_g = (V_{vnf}, E_{vl}, A)$ and $NS'_g = (V'_{vnf}, E'_{vl}, A')$, respectively. NS_g is isomorphic (i.e., Exact service similarity) to another service NS'_g denoted by ($NS_g \equiv NS'_g$). Iff (if and only if) there is a bijection function φ:$V_{\text{vnf}} \rightarrow V'_{\text{vnf}}$ satisfying the following conditions:

a. $|V_{vnf}| = |V'_{vnf}|$, where $|V_{vnf}|$ represent the number of requested VNF nodes in a NS graph.

b. $|E_{vl}| = |E'_{vl}|$.

c. $\forall(vnf) \in V_{\text{vnf}}, (f_v(vnf) = f'_v\ \varphi\ (vnf))$.

d. $\forall(vnf_i, vnf_{\text{j}}) \in V_{\text{vnf}}(vnf_{\text{i}}, vnf_{\text{j}}) \in E_{\text{vl}} \iff (\varphi(vnf_{\text{i}}), \varphi(vnf_{\text{j}})) \in E'_{\text{vl}}$.

e. $\forall(vnf_i, vnf_{\text{j}}) \in E_{\text{vl}}, (f_{\text{e}}(vnf_{\text{i}}, vnf_{\text{j}}) = f'_{\text{e}}(\varphi(vnf_{\text{i}}), \varphi(vnf_{\text{j}}))$.

The bijection φ is an isomorphism between NS_g and NS'_g. NS_g is isomorphic to NS'_g and vice versa. In order to illustrate the exact similarity checking, lets consider the following example.

Example 1. Let consider the three SFC graphs shown in Figure 3.1 According to **Definition 2.** graph (a) is an isomorphic to graph (b) $NS_g \equiv NS'_g$; because both service graphs NS_g and NS'_g

have an identical topology, same number of requested VNF nodes, same connections between VNF nodes and the same attributes (i.e., resource requirements) for both VNF nodes and virtual links). However, graph (a) is **Not** isomorphic to graph (c), although the topology, number of VNF nodes, number of links are identical, but the attributes of the first VNF node and third virtual link are different.

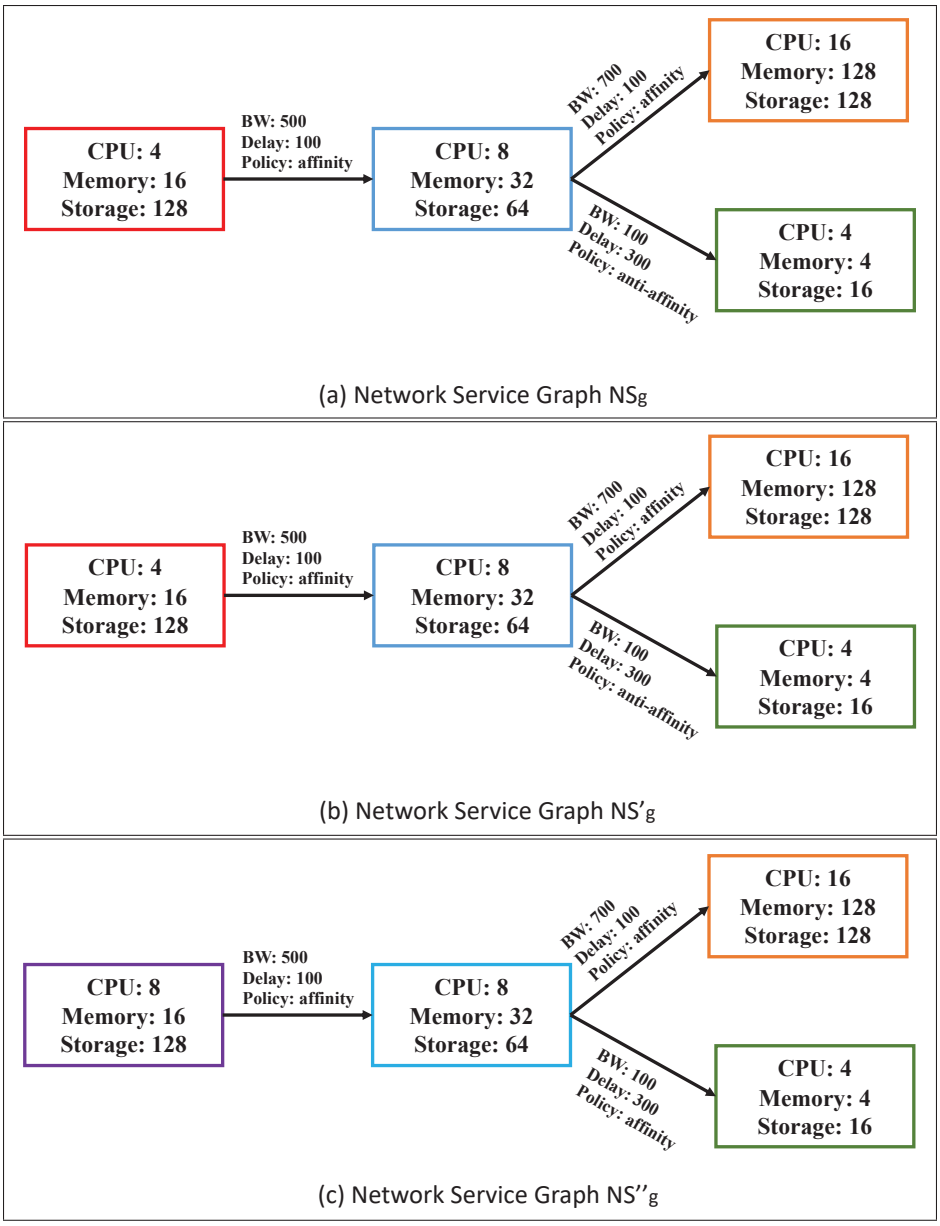


Figure 3.1 Example of a set of network service graphs.

Moreover, one SFC is claimed to be a sub-service of another SFC if and only if the following four conditions are satisfied. Firstly, the set of VNFs composing the first SFC is a subset of the set of VNFs composing the second SFC. Secondly, the set of virtual links connecting VNFs in the first SFC is a subset of virtual links connecting VNFs in the second SFC. Thirdly, the attributes for each matched VNF peers(node and link attributes) in the first and the second SFCs are exactly the same. To support partial matching, we apply definition 3.

Definition 3. Given two NS graphs denoted by $NS_g = (V_{vnf}, E_{vl}, A)$ and $NS'_g = (V'_{vnf}, E'_{vl}, A')$, respectively. NS_g is a sub-service of NS'_g denoted by $NS_{\mathrm{g}} \subseteq NS'_{\mathrm{g}}$, iff:

a. $V_{\mathrm{vnf}} \subseteq V'_{\mathrm{vnf}}$.

b. $E_{\mathrm{vl}} \subseteq E'_{\mathrm{vl}}$.

c. $\forall(vnf) \in V_{\mathrm{vnf}}, (f_v(vnf) = f'_v\,(vnf))$.

d. $\forall(vnf_i, vnf_j) \in E_{\mathrm{vl}}, (f_e(vnf_i,vnf_j) = f'_e\,(vnf_i,vnf_j))$.

The partial similarity checking is described using example 2. **Example 2.** Let consider the two SFC graphs shown in Figure 3.2. According to **Definition 3**, graph (a) is a sub-service of graph (b), $NS_{\mathrm{g}} \subseteq NS'_{\mathrm{g}}$ since all the VNF nodes and virtual links of NS_g are a subset of NS'_g with the same links and attributes.

Isomorphism property can also be applied on sub-services, which we call it a sub-service isomorphism. A network service NS_g is claimed to be isomorphic to another network service NS'_g if and only if NS_g is a subset of NS'_g , or if there is a sub-graph NS''_g in NS'_g which is isomorphic to NS_g and satisfies all the conditions of **Definitions 3 and 4**.

Definition 4. A network service $NS_g = (V_{vnf}, E_{vl}, A)$ is isomorphic to $NS'_g = (V'_{vnf}, E'_{vl}, A')$, denoted by $NS_{\mathrm{g}} \subseteq NS'_{\mathrm{g}}$, iff there exists a sub-graph NS''_g of NS'_g such that NS_g is isomorphic to NS''_g.

To support the partial matching, we also apply definition 5 and 6 which enable to compute the number of occurrence where the sub- service graph $subNS_g$ is contained in each NS_{gi}.

Definition 5. Given a set of virtualized Network service $NS_g = NS_{g1}, ..., NS_{gn}$, the support of sub-service graph $subNS_g$, denoted by sup_{subNS_g} is defined as the fraction of NS graphs in NS_g which contains $subNS_g$.

$$sup_{subNS_g} = \frac{|\{NS_{gi} \in NS_g | subNS_g \subseteq NS_{gi}\}|}{|NS_g|} \tag{3.4}$$

Moreover, a sub-service graph is considered as a frequent one if and only if the support value for this sub- service is greater than or equal to the threshold δ.

Definition 6. Given a minimum support threshold $\delta = (0 < \delta < 1)$, $subNS_g$ is frequent sub-service iff $sup_{\text{subNS}_g} \geq \delta$.

We propose to illustrate the computation of the number of sub-service occurrences using example 3.

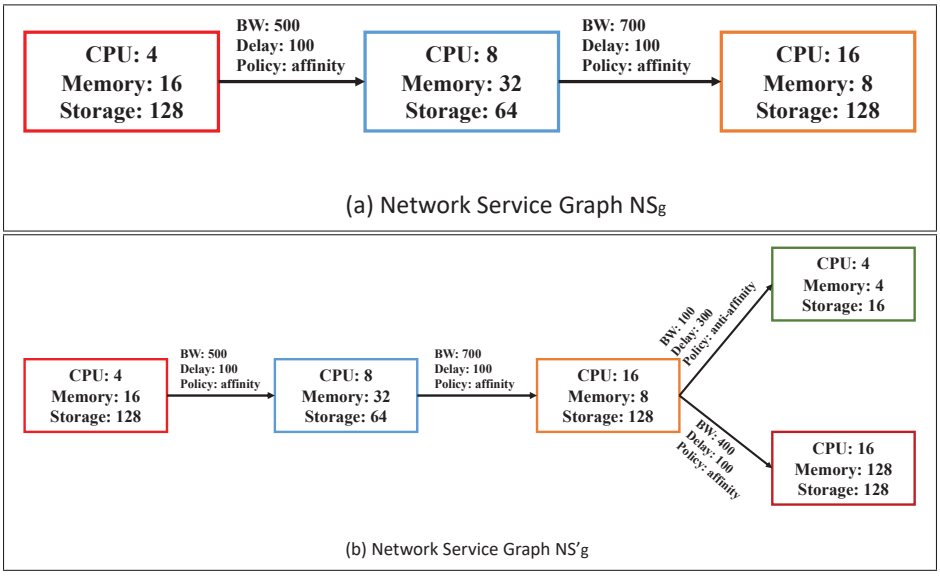


Figure 3.2 Another Example of a set of network service graphs.

Example 3. Let consider the two SFC graphs shown in Figure 3.3 (a) and (b). Let fix $\delta = 1$, the set of extracted sub-service graphs using FSM algorithm and their support values are illustrated in Figure 3.3 (c). For instance, the computed support value of sub-service graph sup_{subNS_5} equals 2, because this sub-service graph exists in both graphs NS_{g1} and NS_{g2}.

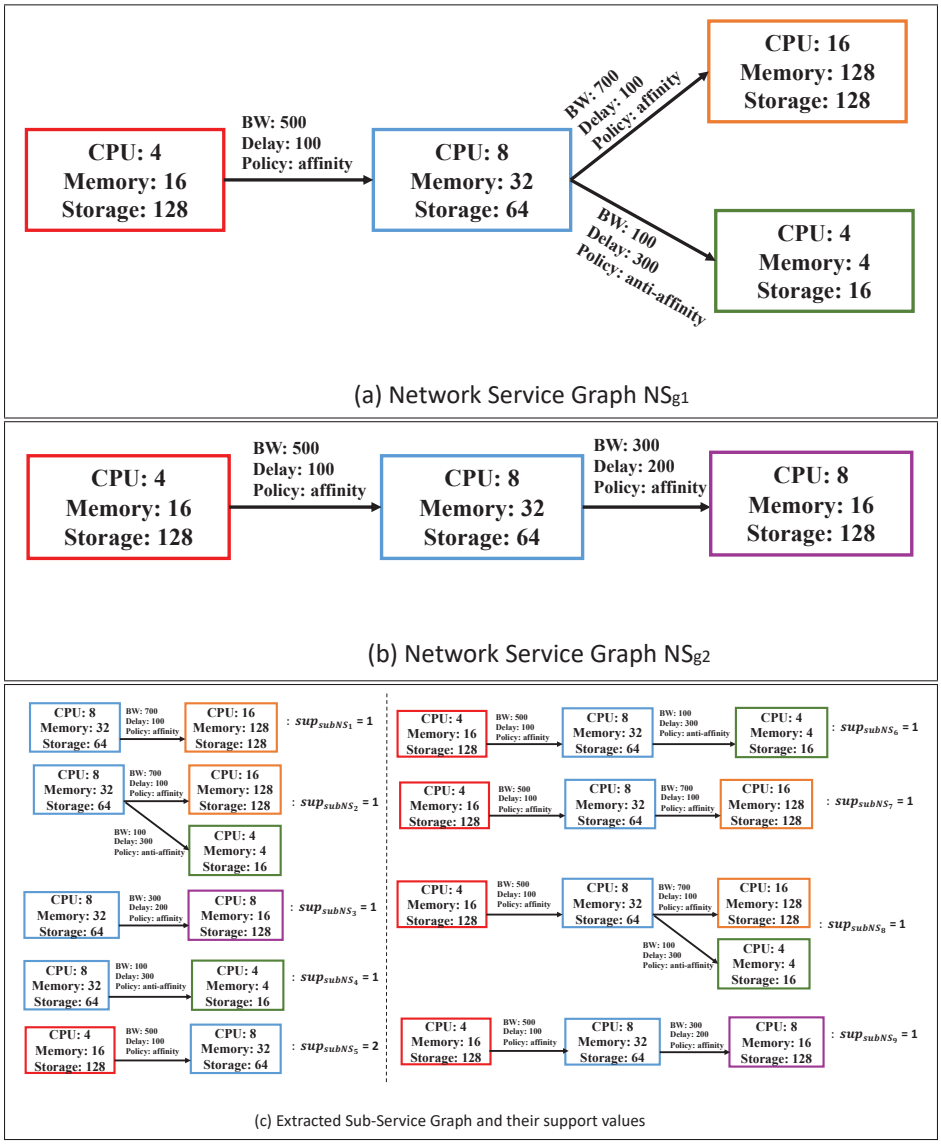


Figure 3.3 Sub-service occurrences based on FSM algorithm.

As previously mentioned, each SFC is represented as a graph. This graph is considered as an input to the PV-DBOW (Doc2vec) model, where each network service graph is treated as a

document and the frequent sub-service graph as a word. The main objective of this embedding model is to convert the network service graph into a low d-dimensional vector representation using definition 7.

Definition 7. Given an NS graph NS_g = (V_{vnf}, E_{vl}, A), the embedding function ε: $NS_{\mathrm{g}} \rightarrow R^d$ is used to map the input graph NS_g into low d-dimensional vector space representation, where d is the pre-defined dimensionality of embedding (i.e., 2-dimensional or 3-dimensional space).

When the matrix Φ, which contains the d-dimensional vectors of the deployed and the new requested NF graphs, is generated, we find the similarity rate between the new SFC request and the deployed ones which can be computed based on definitions 8,9 and 10.

Definition 8. Given a set of NS graphs $NS_{\mathrm{g}} = NS_{\mathrm{g1}}, ..., NS_{\mathrm{gn}}$, the matrix Φ = [$\varepsilon$ (NS_{g1}), ..., $\varepsilon(\mathrm{NS_{gn}})$] contains the embedding representation for every $NS_{\mathrm{gi}} \in NS_{\mathrm{g}}$. The Similarity rate denoted by *SimilarityRate (SR)* between any two NS graphs will be measured using Cosine similarity method. It is given by equation 3.5. Cosine similarity is used to measure the similarity between two vectors using dot product, in another word; it measures the cosine of the angle between the two vectors.

$$SR(NS_{gi}, NS_{gj}) = \frac{\Phi(NS_{gi}) \cdot \Phi(NS_{gj})}{||\Phi(NS_{gi})||||\Phi(NS_{gj}||} \tag{3.5}$$

The range of values for the similarity rate will be between 0 and 1. Two network services are said to be exactly similar if the similarity rate is between the two predefined thresholds 0.99 and 1. In another word, NS_g and NS'_g are similar if $\varepsilon(NS_g)$ and $\varepsilon(NS'_g)$ are close to each other on the vector space and vice versa.

Definition 9. Given two NS graphs denoted by NS_g = (V_{vnf}, E_{vl}, A) and NS'_g = (V'_{vnf}, E'_{vl}, A') respectively, and a similarityRate $(0 < similarityRate \leq 1)$. NS_g is exactly similar to another service NS'_g, $(NS_{\mathrm{g}} \equiv NS'_{\mathrm{g}})$ iff:
$0.99 < similarityRate(\mathrm{NS'_g}, \mathrm{NS_g}) \leq 1$

On the other hand, the predefined thresholds for NS_g to be a sub-service of another sub service NS'_g are between 0.7 to 0.99, as described in the definition 10.

Definition 10. Given two network service graphs denoted by $NS_g = (V_{vnf}, E_{vl}, A)$ and $NS'_g = (V'_{vnf}, E'_{vl}, A')$ respectively, and a similarityRate = $((0 < similarityRate \leq 1))$. NS_g is a sub-service of another service $NS_g \subseteq NS'_g$ iff:
$0.7 < similarityRate(\text{NS'}_g, \text{NS}_g) \leq 0.99$

3.6 Proposed Approach

This section describes the flow chart and the architectural components of the proposed similarity manager tool.

3.6.1 Architecture Overview

As illustrated in Figure 3.4. The proposed Service Similarity Manager tool is integrated as new functionality to NFVO. It is based on five key architectural components:

a. Service Graph Decomposition enables to break-down the SFC service graph into a list of sub-service graphs. These sub-service graphs will be extracted using frequent sub-graph mining technique called “gSpan”.

b. Service graph Translator allows to map the SFC service graph into d-dimensional vector representation using PV-DBOW machine learning embedding model.

c. Service Similarity Checker measures the similarity rate between the new incoming SFC request and the already provisioned/instantiated SFCs . The similarity between two SFCs is measured using Cosine similarity.

d. Translated Service Graphs Repository acts as database, containing all SFC service graph vector representations obtained by Service Graph Translator.

e. Similarity Rate Repository contains all the measured similarity rate between SFC services. This repository might be used by the MANO layer in future for prediction and management of resources.

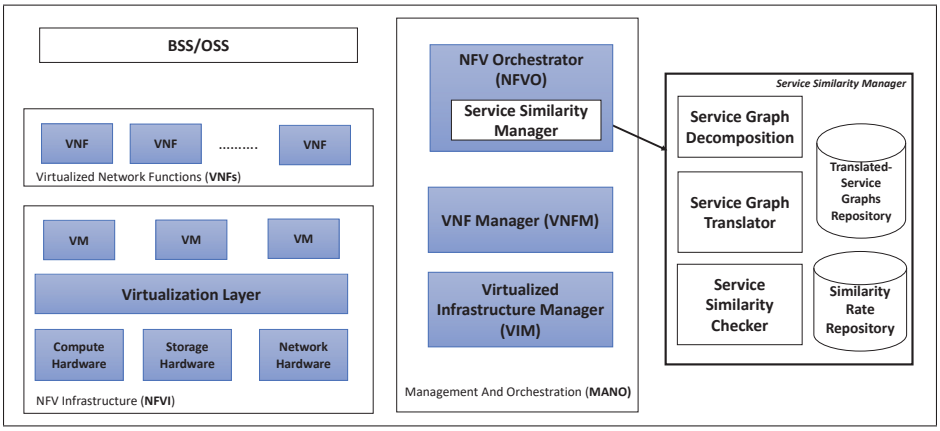


Figure 3.4 Service Similarity Manager tool integrated into NFV framework.

Those five architectural components are communicating interactively to find the similarity between SFCs in a rapid manner.

The overall architecture of the proposed tool is depicted in Figure 3.5. Once a new SFC request graph is received by the NFV Orchestrator, the Service Similarity Manager is launched before invoking the redundancy and conflict detection algorithm (Sundararajan *et al.* (2015)) and the placement and chaining algorithm (Li *et al.* (2018)). These two algorithms are out of the scope of this paper.

Firstly, The Service Graph Decomposition starts by extracting the sub-service graphs $\{SubService_0, ..., SubService_n\}$ of an SFC service request. In another word, the whole SFC is decomposed into a set of sub-service graphs using frequent sub-graph mining algorithm called "gSpan". Each sub-service graph consists of at least two connected VNF nodes with their attributes. Then, the Service Graph Translator converts the new SFC service graph with its sub-service graphs into d-dimensional vector representations using PV-DBOW embedding technique, where

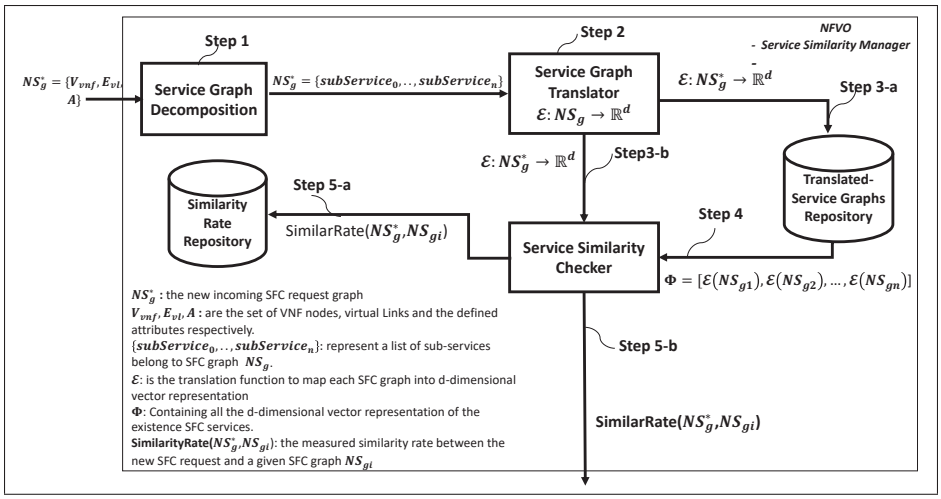


Figure 3.5 High level overview of the proposed approach.

the whole graph is treated as a document and each sub-service graph as a word. The output of the Service Graph Translator is stored into the Translated Service Graph Repository. It is worth mentioning that when any modification occurs for the deployed SFC (e.g., VNF resource scaled in/out, VNF migration, etc.) the Translated Service Graph Repository will update its vector representation. The d-dimensional embedding vector of the new SFC request then undergoes the Service Similarity Checking. The Service Similarity Checker sends a query to Translated Service Graph Repository asking for all the d-dimensional embedding vector representations of the deployed SFC service and starts by computing the similarity between the new SFC request and each deployed SFC. Finally, the output of the Service Similarity Checker is inserted into Similarity Rate Repository. This latest will be used to determine if the policy conflict and redundancy detection and resolution as well as the placement and chaining processes should be launched or not. It can be also used by resource prediction and management mechanisms.

3.6.2 SFC Service Similarity Checker Algorithms

This section describes the SFC similarity checker algorithms. The Similarity Manager is invoked when a new SFC service request is received (Algorithm 3.1). The inputs of the algorithm are the pre-defined minimum support threshold value $(0 < \delta \leq 1)$, the required embedding di-

mension and the translated vector representations of all the deployed SFC services, while the output is a list containing all the measured similarity rates between the new SFC request and all deployed SFCs. The first step is to decompose the SFC request graph and find all the sub-service graphs using gSpan algorithm (Line 7 of algorithm 3.1).

Next, Algorithm 3.2 is applied to map the new SFC request graph into d-dimensional vector representation (Line 8 of Algorithm 3.1). Then, to compute the similarity rate between the new SFC request graph and each deployed SFC, the Service Similarity Checker described in Algorithm 3.3 is invoked (The first for loop in Algorithm 3.1). Finally, according to the output of Algorithm 3.3, if the measured similarity values between the new SFC request and any of the deployed SFC service are in the range of 0.99 and 1, then the new SFC service request is isomorphic (i.e., exact service similarity) to the deployed service (see **Definition 2.** in section 3.5). Whereas if the measured similarity is in the range of 0.7 to 0.99, this indicates that the new SFC service request is a sub-service of the deployed service (see **Definition 3.** in section 3.5). Otherwise, this mean that the new SFC request has no exact or partial similarity with any of the deployed services (The second for loop in algorithm 3.1).

In algorithm 3.2, Service Graph Translator learns the embedding vector representation of a service graph. The algorithm takes as inputs the SFC service graph, the required embedding dimension and the list of sub-service graphs of the given SFC graph obtained from gSpan algorithm. The output is the vector embedding representation with the required d-dimension. Algorithm 3.2 describes the PV-DBOW (see Section 3.4), where the whole SFC service graph and each extracted sub-service graph are treated as a document and as a word respectively. Calculating the $Pr(subNS_j|NS_{gi})$ (Line 8 of Algorithm 3.2) is expensive for large number of sub-service graphs. Therefore, we use the negative sampling method (introduced in section 3.4).

The first step is to select a set of fixed number of randomly chosen sub-service graphs as negative samples $x' = \{nSubSer_1, ..., nSubSer_n\}$. The selected negative samples (i.e., sub-service graphs) is not listed in the SFC service graph whose embedding should be learned. For the

Algorithm 3.1 SFC Service Similarity Checker

Input: New SFC Request NS_g^*
Input: A matrix $\Phi = [\varepsilon\ (NS_{g1}), ..., \varepsilon(\mathrm{NS_{gn}})]$ containing the translated d-dimensional vector representation of the deployed SFC service graphs.
Input: δ minimum support threshold.
Input: d embedding dimension.
Output: a list of *SimilarityRate* identifying the similarity between the new SFC request and the available and deployed SFC services.
procedure SFCSIMILARITYCHECKER
    Execute the sub-graph mining process with (NS_g^*, δ) using "gSpan" Algorithm to Extract the set of sub-service graphs from NS_g^* and store the sub-services into $subNS(NS_g^*)$
    Execute Algorithm 3.2. to learn the vector embedding representation $\varepsilon(NS_g^*)$ of the new SFC request.
    $SimilarityRate = \phi$
    for each $\varepsilon(NS_g) \in \Phi$ **do**
        Execute Algorithm 3.3 to find Similarity with ($\Phi(NS_g^*)$,$\Phi(NS_g)$) as input to compute the similarity rate.
    end for
    /* Check if the new SFC request has an exact similarity service/sub-service with a deployed service */
    for each $s \in SimilarityRate$ **do**
        if $0.99 < s \leq 1$ **then**
            $NS^*_g \equiv NS_g[s]$
        else if $0.7 < s \leq 0.99$ **then**
            $NS^*_g \subseteq NS_g[s]$
        end if
    end for
end procedure

input SFC service graph, the (NS_{gi}, x) is trained and the embedding of all corresponding sub-service graphs are updated (only the embedding of negative samples are updated). For instance, given two SFC service graphs NS_{g1} and NS_{g2}, this training makes their embedding $\Phi(NS_{g1})$ and $\Phi(NS_{g2})$ closer if they contain similar sub-service graphs.

Finally, Algorithm 3.3 describes the functionality of the proposed Service Similarity Checker. It computes the similarity between two SFC service graphs.

The inputs are the d-dimensional embedding vector representation obtained by Algorithm 3.2 and the output is a value in range 0 to 1 which identifies the similarity between the two service graphs, where 0 means totally not similar and 1 is 100% similar. Finding the similarity is done by applying the cosine similarity measurement.

Algorithm 3.2 SFC Service Graph Embedding

Input: An SFC network service graph NS_{gi}
Input: $Freq\_SubNS(NS_{gi})$ containing the sub-services obtained using frequent sub-graph mining process "gSpan".
Input: d embedding dimension.
Output: d-dimensional vector representation $\Phi \in R^{|\,NS_{\mathrm{gi}}\,|\,Xd}$ of a network service graph.
procedure SERVICEGRAPHTRANSLATOR
    /* Finding all the sub-service graphs in an SFC request NS_g^* */
    for each $\varepsilon(NS_{\mathrm{g}}) \in \Phi$ **do**
        $\varepsilon(\Phi) = -log(Pr(subNS_{\mathrm{j}} \mid NS_{\mathrm{gi}})$
        $\Phi = \Phi - \alpha * \partial\varepsilon(\Phi)/\partial\Phi$
    end for
end procedure

Algorithm 3.3 Similarity Rate Computation

Input: $\Phi(NS_{gi})$ d-dimensional vector representation of SFC service graph obtained by Algorithm 2
Input: $\Phi(NS_{gj})$ d-dimensional vector representation of SFC service graph obtained by Algorithm 2.
Output: The measured similarity $SimilarityRate$ [0,1] between two SFC service graphs.
procedure SERVICESIMILARITYCHECKER
    /* Finding all the sub-service graphs in an SFC request NS_g^* */
    $$SR = SR \cup \{\frac{\Phi(NS_g^*) \cdot \Phi(NS_{gi})}{||\Phi(NS_g^*)||||\Phi(NS_{gi}||}\}$$
end procedure

3.7 Use Cases

In this section, we will describe the integration of Service Similarity Manager into NFVO to provide agile service provisioning and resource management while minimizing the total processing time required by NFV Orchestrator to deploy new SFC services.

Figure 3.6 illustrates an example of workflow performed by NFV-MANO layer to deploy and manage a new SFC service request. As shown in this figure, when the NFVO receive a new SFC request to deploy, the policy conflict and redundancy detection algorithm is launched to detect all the potential conflicts and redundancies between policies of this SFC. If any conflict and/or redundancy is detected, it should be resolved and filtered out any non-valid policy (phase 1). Then, the placement and chaining algorithm is invoked. It checks the available phys-

ical resources and decides where to map each VNF node and virtual link into the underlying available resources (phase 2). The placement and chaining can be performed jointly or not. When the new SFC is deployed, it is registered into NS catalog which contains all the available network services.

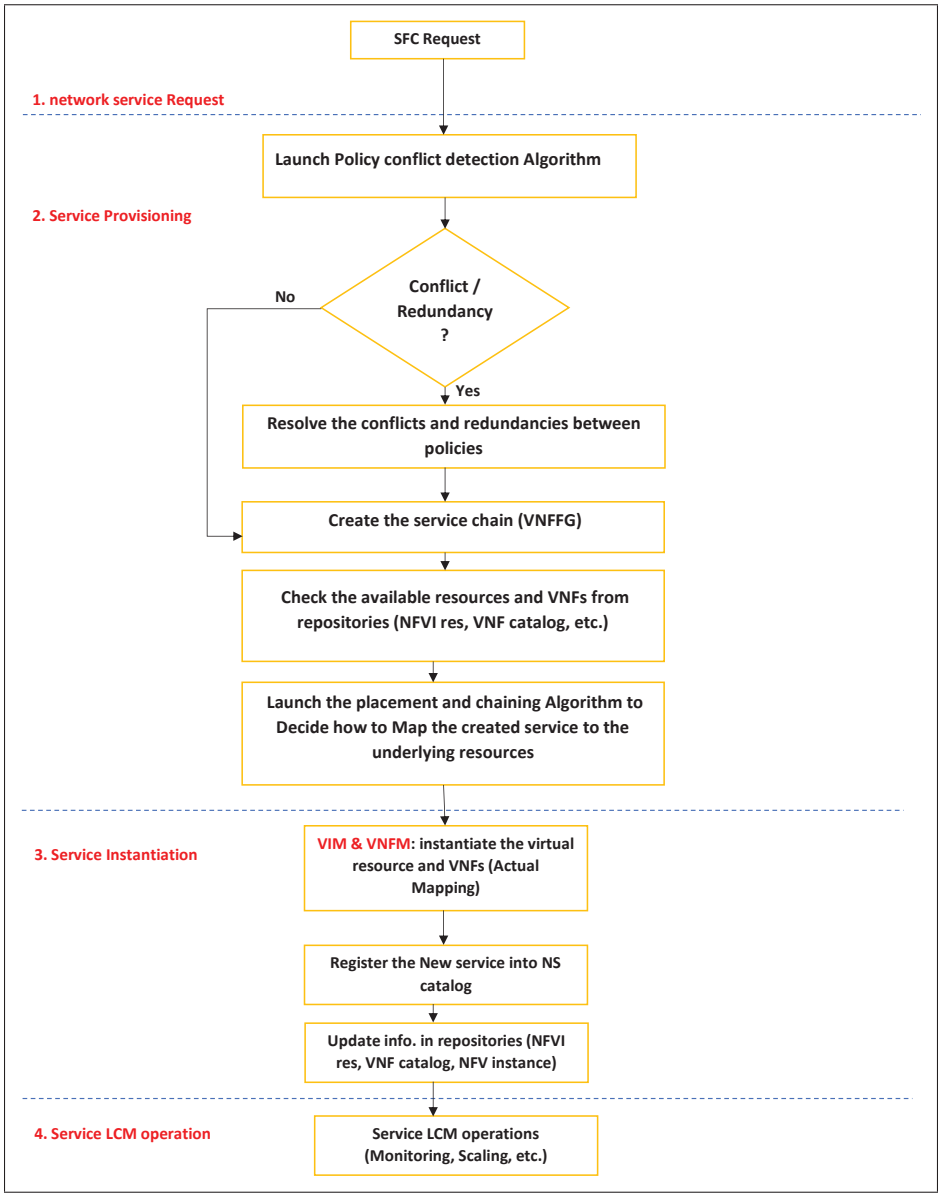


Figure 3.6 the usual workflow of SFC service instantiation.

The SFC deployment process is a daunting task that needs to be optimized and one or both of its phases could be skipped using the similarity manager tool, which allows us, in some cases, to skip the policy redundancy/conflict detection and/or placement/chaining processes.

Therefore and in order to save time, it is suitable to avoid invoking one of the two processes or both to deploy each new SFC request received by NFVO (see Figure 3.6). Hence, using the Service Similarity Manager tool (Figure 3.7), NFVO will be able to skip some processing steps in the two phases to provision the new SFC service request while satisfying all the constraints and SLA.

When the NFVO receive the new SFC request, before launching the policy conflict detection and resolution algorithms, the Service Similarity manager is invoked. If the Similarity Manager tool found that the new SFC service to-deploy has an exact similarity with one of the deployed services or sub-service, there is no need to detect the conflicts and redundancy between policies , this is because, it is impossible to deploy an SFC service into underlying physical resources without validating all the policies and since the new SFC request is an exact similar to a deployed service/sub-service this means that the policies are free of conflict and validated. Then, if the hosted physical servers of the deployed service which is similar to the new SFC request still have enough resources, without launching the placement and chaining algorithm, the new SFC request will be hosted into the same physical servers as the similar one, which significantly reduce the mapping time. When the hosted physical servers have no enough resources, the policy conflict detection and resolution steps are skipped but the placement and chaining algorithm must be invoked.

Nevertheless, the placement and chaining process can be invoked while specifying the type of physical servers and resources that are appropriate to host the new SFC, which reduces the search space of the process, but using the proposed Similarity Manager tool before invoking the two phases is significantly reducing the processing time needed for deploying the new SFC request. In the worst case scenario, when the Service Similarity Manager tool did not find

any matching or similarity between the new SFC request and the deployed SFC service, it will execute the network service provisioning workflow as illustrated in Figure 3.6.

Finally, by checking if the new SFC request is similar to a deployed service/sub-service and logging all such information into a database (e.g., Similarity Rate Repository), it could be used for VNF resource management and VNF and/or network service life-cycle prediction. For example, when a deployed service (A) requests for scaling out (i.e., adding new virtual machines) after processing a certain traffic workload, an exactly similar service (B) can be predicted to behave the same way (scaling out as service A) after processing the same amount traffic workload.

3.8 Experimental Setup and Analysis

In this section, we evaluate and further discuss the performance evaluation of the proposed approach. Our goal is to assess its capability to discover the similarity rate between the new incoming SFC request and the deployed network services by NFVO, in order to reduce the total time required by NFV Orchestrator to provision and deploy a new SFC services by skipping detecting and resolving the conflicts and redundancies between policies algorithms and/or calling the placement and chaining algorithms.

3.8.1 Evaluation Setup

We implemented our algorithm with python 2.7. All the experiments were conducted on Intel (R) core (TM) i7-7700 CPU @ 3.60 GHZ with 16 GB RAM, running Ubuntu 16.04.6 LTS. All the experiments were repeated 10 times and we report the average values.

3.8.2 Virtual Network Service Graph Datasets

To the best of our knowledge, this work is the first that consider the similarity between virtualized network service graphs. Therefore, most of graph datasets available in literature review are related to chemical compound datasets, social networks of YouTube users, and other synthetic

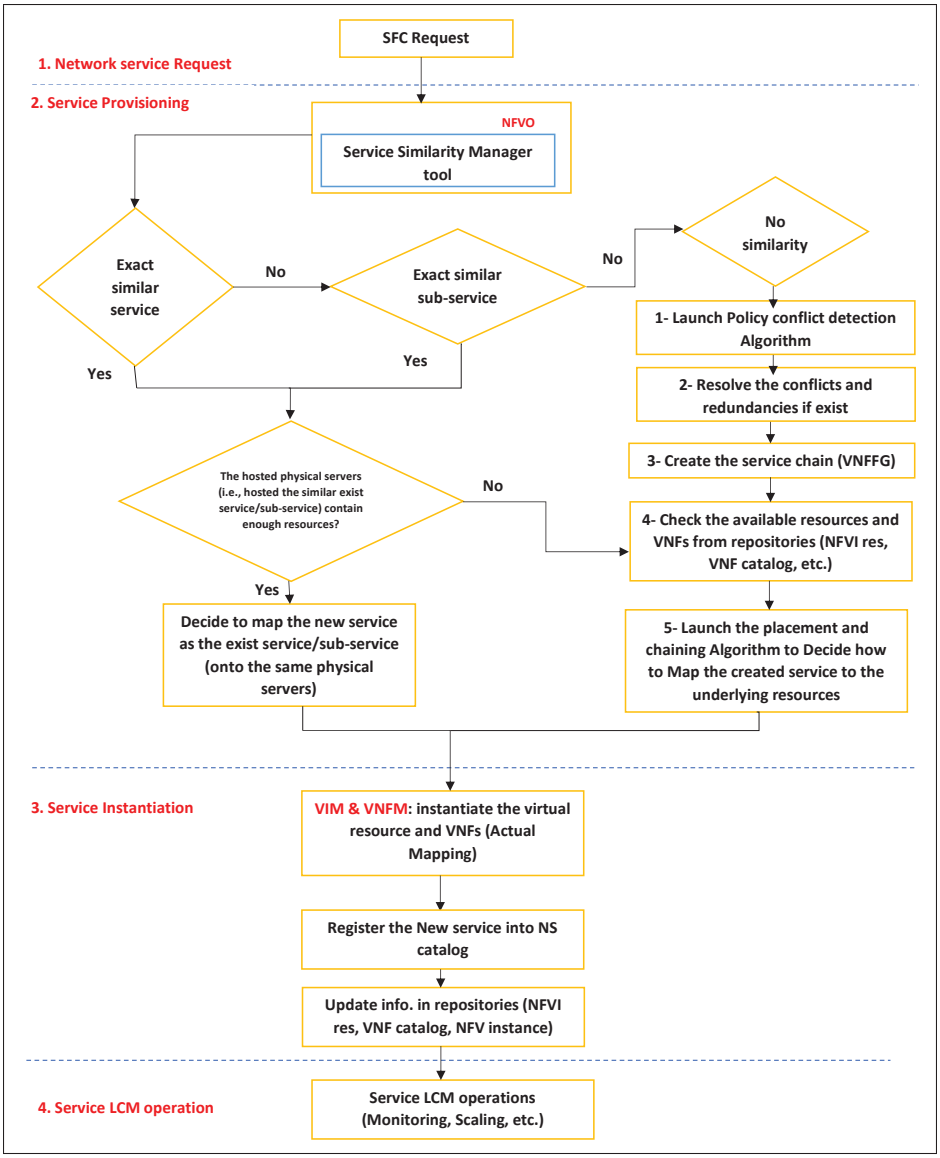


Figure 3.7 The Workflow of SFC service instantiation using our proposed approach.

and real datasets . For our assessment, we have used NetworkX library to generate random virtual service graphs for testing. NetworkX is a python package used for creation, manipulation, and study structures of complex networks. It is worth mentioning that in this paper, we consider a complex virtual network service topologies (linear and non-linear). Table 3.3 describes examples of characteristics and requirements of the virtual network services.

Table 3.3 Virtual Network Service Graph characteristics

Parameter	Range/ Value
Number of VNFs in SFC service	[2, ...,25] virtual node
Required Bandwidth for virtual links	[100, ...,700] Mbps
End-to-End delay for virtual links	[100, ..., 500] ms
Required number of CPU cores for a VNF	[4, ..., 16] cores
Required Memory usage for a VNF	[4, ..., 128] MB
Required Storage	[16, ..., 128] GB
Placement Constraints	[Affinity, Anti-Affinity, Or None]
Topology	[Linear, non-linear]

The new SFC request received by the proposed Service Similarity Manager could be in a different form. For example, it could be a simple SFC request (Linear SFC) or more complex (Non-linear SFC). Therefore, our algorithm will be tested on two different scenarios.

Table 3.4 shows two scenarios, S1 and S2. S1 and S2 identify linear and non-linear SFCs respectively. Moreover, in these scenarios the number of requested VNFs to deploy vary from 2 to 25.

Table 3.4 Scenarios Description

Scenario	SFC Request Topology	Number of Requested VNFs
S1	Linear	[2, ..., 25]
S2	Non-linear	[3, ..., 25]

Moreover, we generate three datasets. Each dataset identifies the available SFC services that have been deployed by the NFVO. The first set containing small number of virtual network service graphs (e.g., the total number of existing services is 10) and the network service itself of small size (e.g., number of VNFs ranging from 2 to 5). The second set is more complex and contains larger number of network services graphs and more complex characteristics (e.g., 500 SFCs), whereas the last set contains a large number of network graphs (e.g., 1000 SFCs). The characteristics of each virtual network service as presented in Table 3.3.

Table 3.5 summarize the characteristics of the generated service graph datasets using NetworkX. Where $|NS_g|$ is the number of the available virtual network services that have been deployed by NFVO to be compared with the new incoming SFC request, avg. $|V_{vnfs}|$ and avg.

$|E_{vls}|$ is average number of VNF nodes and virtual links in NS_g respectively, $|A_v|$ and $|A_e|$ is the total number of attributes assigned for VNF nodes (ex: CPU cores, Memory... etc.) and Virtual Links (ex, BW, delay, ... etc.) respectively.

Table 3.5 Datasets Characteristics

Dataset	$\|NS_g\|$	**Avg.** $\|V_{vnfs}\|$	**Avg.** $\|E_{vls}\|$	$\|A_v\|$	$\|A_e\|$
DS1	10	2.9	1.7	87	43
DS2	500	18.01	17.02	27024	22524
DS3	1000	17.51	16.72	52548	50088

3.8.3 Evaluation and Analysis

As discussed above, our proposed Service Similarity Manager is invoked when a new SFC request is received by the NFVO. Our Algorithm as shown in Algorithm 3.1 has three main steps. Therefore, we will evaluate and study the main factors that may have an impact on each step by running the aforementioned datasets and scenarios.

a. **Step 1:** finding all the sub-service graphs of a given SFC service:
When the proposed similarity manager receives the SFC request, Service Graph Decomposition is responsible to decompose and break-down all the whole SFC service request into a list of sub-service graphs. To evaluate the factors that may have an impact on Service Graph Decomposition. We conduct a set of experiments by running the scenarios presented in Table 3.4 and by varying the SFC request size (i.e., number of requested VNF nodes). As shown in Figure 3.8. The number of sub-service graphs extracted from the SFC request relies on the number of requested VNFs and the topology of the requested SFC. The number of sub-service graphs in scenario S2 is higher than scenario S1 although the number of requested VNF nodes is the same; this is because the topology and the connections between VNF nodes in S2 are more complex than in S1. Hence, the whole SFC service request will be decomposed into a larger number of sub-service graphs.

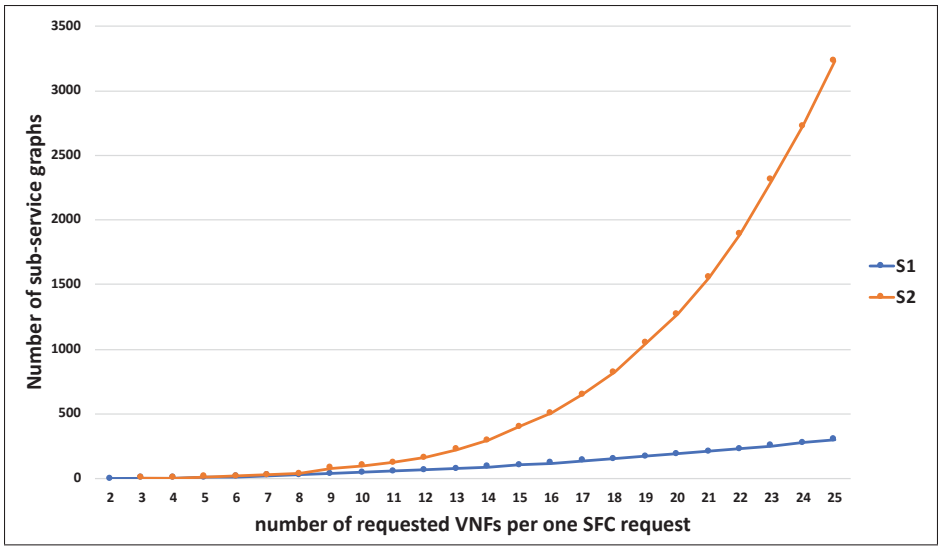


Figure 3.8 Number of sub-service graphs extracted from Service Graph Decomposition algorithm ($\delta = 1$).

Figure 3.9 shows the time required by the Service Graph Decomposition using "gSpan" to find the sub-service graphs depicted in Figure 3.8. Scenario S2 consumed more time than scenario S1, this is because the number of sub-service graphs to be extracted is higher.

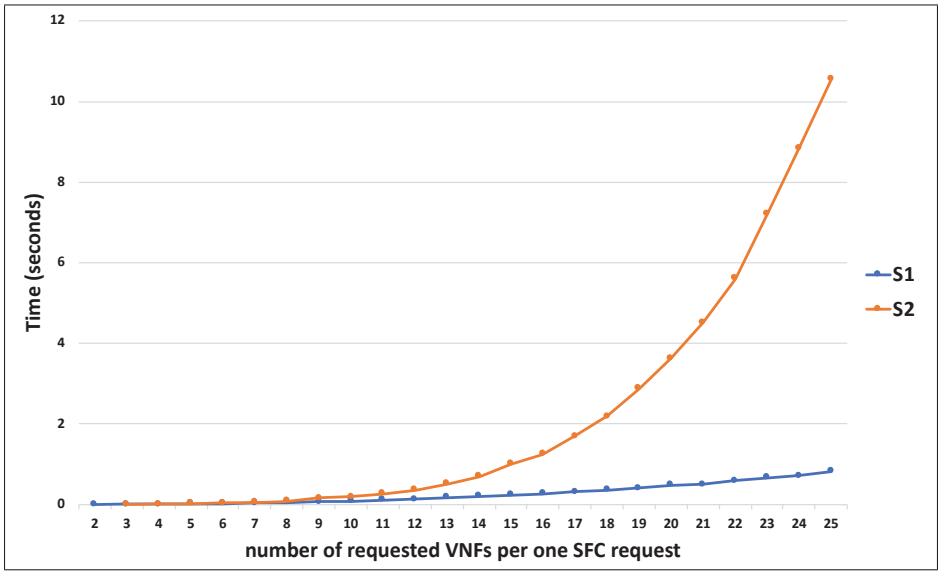


Figure 3.9 Processing time of Service Graph Decomposition algorithm ($\delta = 1$).

b. **Step 2:** learning the SFC service graph embedding:

As presented in Algorithm 3.2. The Service Graph Translator proposed to map the SFC service request graph into d-dimensional embedding vector representation by PV-DBOW machine learning embedding model. To evaluate the performance of Service Graph Translator we conduct a set of experiments by running the scenarios presented in Table 3.4 on different SFC request size (i.e., number of requested VNF nodes).We will follow most graph embedding techniques presented in (Goyal & Ferrara (2018)) and select 128 as the embedding dimension.

It can be seen from Figure 3.10, that the performance of SFC graph embedding relies on the number of requested VNFs and the topology of the requested SFC. Higher is the number of requested VNFs per SFC, higher is the time to provision a new SFC.

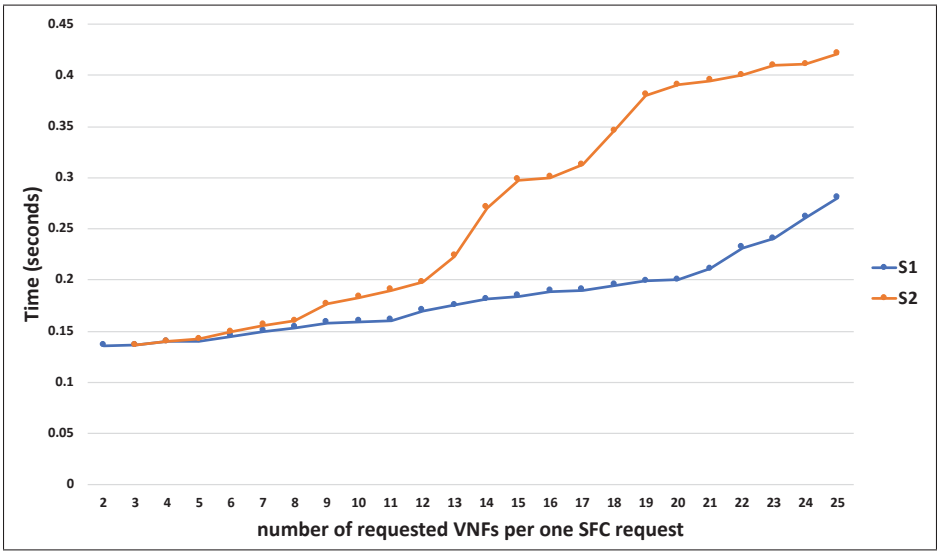


Figure 3.10 Processing time of Service Graph Translator (d = 128).

c. **Step 3:** Finding the similarity rate between SFC service graphs:

The final step in our proposed similarity manager is to measure the similarity rate between the new SFC service request and all SFCs deployed by NFVO using Service Similarity Checker.

To evaluate the performance, we conduct a set of experiments by running both scenarios presented in Table 3.4 on the generated datasets presented in Table 3.5. Each dataset identifies the number of available SFC services to be compared with the new SFC request.

As shown in Figure 3.11 and Figure 3.12. the time required by Service Similarity Checker is not affected by the complexity of SFC service request. Both scenarios S1 and S2 require the same time to process the same SFC request and using the same dataset. This is because, the Service Similarity Checker compare the translated d-dimension vector representation. And as described in Algorithm 3.2, each SFC service graph will be translated into a vector with an embedding size (d = 128) regardless the complexity or the number of requested VNF nodes. Also, it can be seen from Figure 3.11 and Figure 3.12, that the Service Similarity Checker is affected by one metric: the number of available SFC services that the new SFC service request should be compared with to find the similarity (i.e., dataset size DS1, DS2 and DS3).

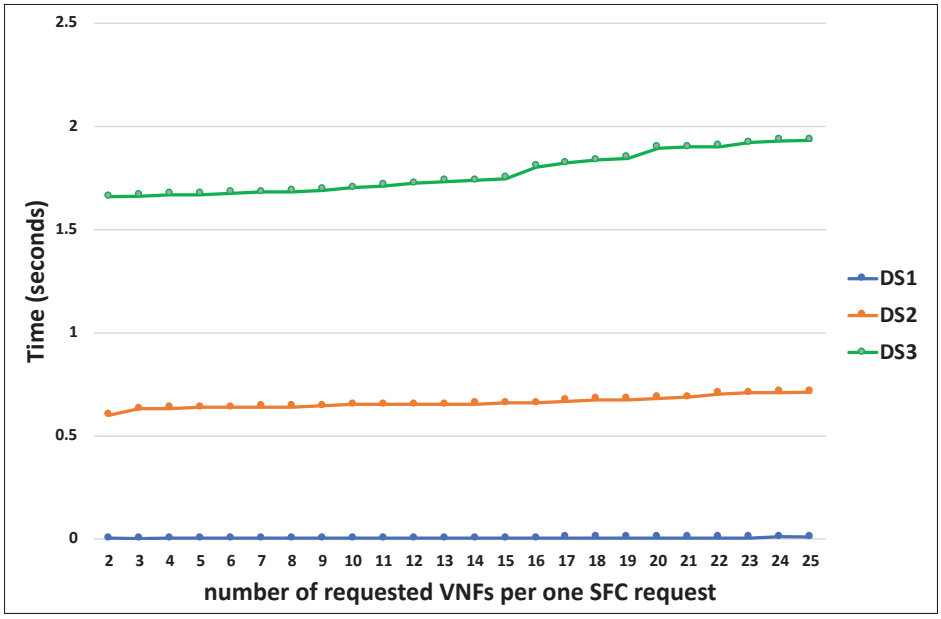


Figure 3.11 Execution time of Service Similarity Checker - scenario S1 with different datasets.

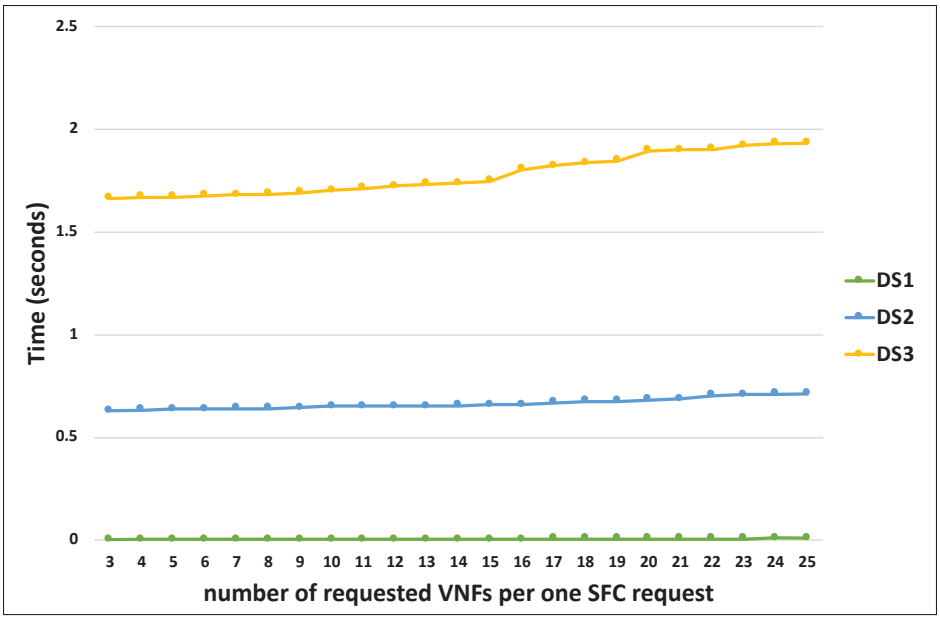


Figure 3.12 Execution time of Service Similarity Checker - scenario S2 with different datasets.

d. Performance of the proposed Service Similarity Manager approach:

Figures 3.13 and 3.14. illustrate the total processing time of the proposed Service Similarity approach. Accordingly, we can conclude that the proposed Service Similarity Manager relies on three main parameters: the number of requested VNFs per SFC service request, the complexity of SFC request (i.e., linear or non-linear service) and the number of deployed SFC services. Based on this analysis, the main factor effecting the proposed similarity manager tool is the number of deployed SFC services used to compare the new SFC services.

Unfortunately, and to the best of our knowledge, none of the studies in literature review proposes a model to check the similarity between SFC services. Hence, there is no base line to compare the performance of the similarity manager tool with it.

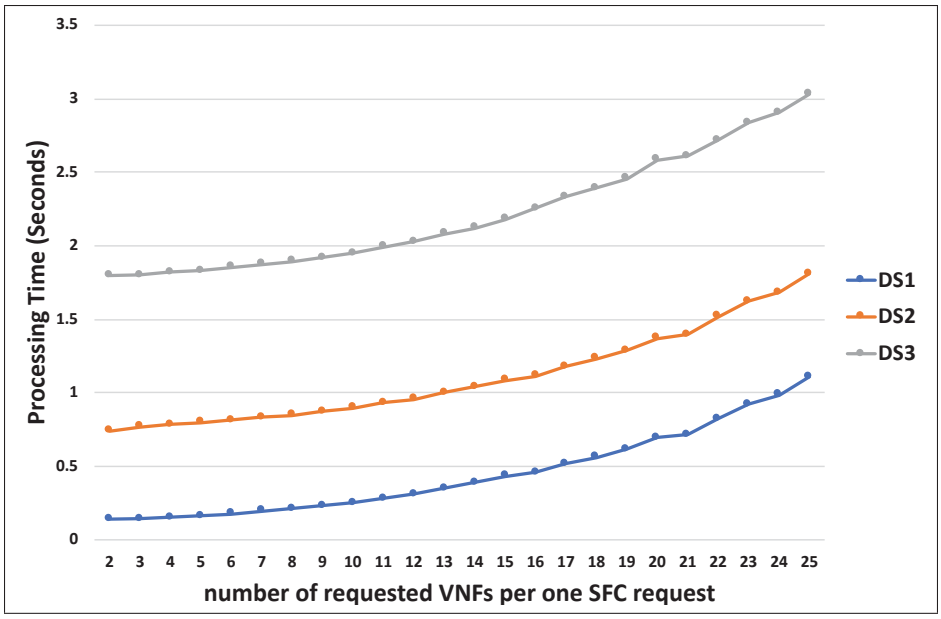


Figure 3.13 Total processing time of the proposed approach - scenario S1.

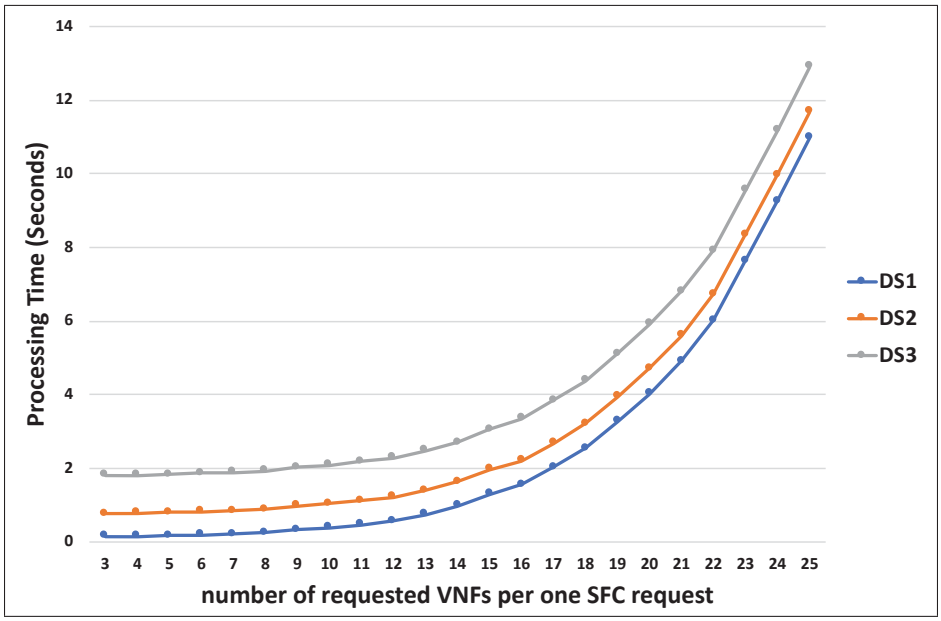


Figure 3.14 Total processing time of the proposed approach - scenario S2.

3.8.4 Evaluation of use case 1: NFV service provisioning

To the best of our knowledge, there is no studies in literature review that provides an evaluation for NFV service provisioning for the complete steps and phases: (1) detecting the conflicts and redundancies between policies (2) resolving the detected conflicts and/or redundancies between policies and (3) running placement and chaining algorithm.

Using our proposed Service Similarity Manager, NFV service provisioning process may face one of the following cases (As shown in Figure 3.15):

1) **The Best case:** where the new incoming SFC request match at least one exact SFC service/sub-service graph **AND** the hosted physical servers where the VNF nodes and virtual links of the similar service have been deployed have enough resource to deploy such new incoming SFC request. In this case, using the proposed algorithm, there is no need to launch the policy conflict detection and resolution as well as the placement and chaining algorithm. Hence, the total time to find the solution is only the time consumed by our proposed similarity algorithms.

2) **The Average case:** where the new incoming SFC request match at least one exact service/sub-service graph **but** the hosted physical servers where the VNF nodes and virtual links of the similar service have been deployed do not have enough resources to deploy the new incoming SFC request. Using the proposed approach, there is no need to launch the policy conflict detection and resolution algorithm. Moreover, we can have an indicator for the type of physical servers that are appropriate to deploy the received SFC request according to the physical servers which host the similar services. Hence, instead of taking into account all the physical servers and infrastructure to find the optimal solution to deploy the new service, the placement and chaining algorithm will consider only a subset of physical servers with specific type.

   Hence,the total time to find the solution is the time required by our proposed algorithm and a smaller time than that needed by the placement algorithm (Smaller search space for substrate resources).

3) **The Worst case:** where the new incoming SFC request did not match any of the service/sub-service graph (i.e., not similar to a service/sub-service graphs). In this case we have to go through all the steps and launch the policy conflict and resolution algorithm as well as the placement and mapping algorithm. Hence, the total time to map a new SFC is the time required to process similarity manager tool and the algorithms for policy conflict and resolution and SFC placement and chaining.

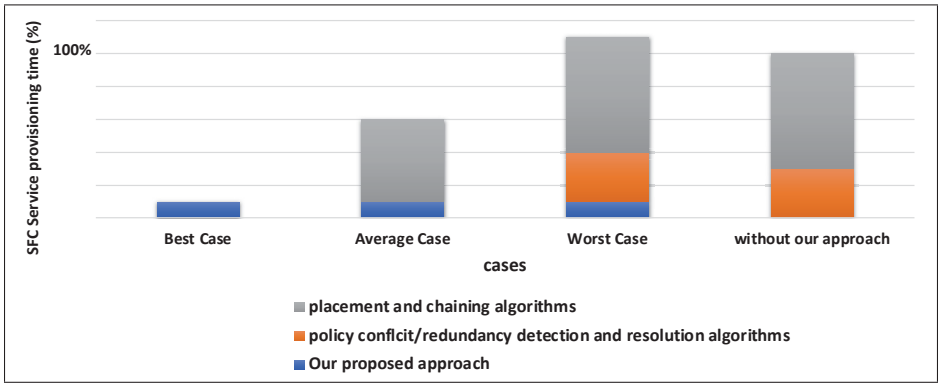


Figure 3.15 SFC provisioning time with and without using our approach.

3.9 Conclusion and Future Work

In this paper, a novel tool has been proposed which is mainly based on finding the similarity between the new incoming SFC request and the already provisioned SFCs in the NFV infrastructure. This tool composed of five main architectural components which are: service graph decomposition, service graph translator, service similarity checker, translated service graph repository and similarity rate repository. The problem has been mathematically formulated by firstly representing each SFC as a directed graph which contains a set of VNF nodes, virtual links and their associated attributes. Then PV-DBOW technique has been used to translate each represented graph into a d-dimensional vector and finally the similarity is found between any two vectors using a cosine similarity technique.

The proposed approach has extensively been tested with different number of VNFs per SFC, various attributes (e.g., required bandwidth and end-to-end delay for virtual links, number of CPU cores, memory and storage needed for each VNF) and with different types of SFCs (linear and non-linear).

The main contribution in the proposed tool is its capability of deciding whether it is really necessary or not to go through the conflict/redundancy and placement/chaining phases in a proactive manner and before starting an SFC provisioning process. This tool has significantly reduced the total processing time taken for SFC provisioning. Using this tool would be highly beneficial small, medium and more noticeably in large scale network infrastructures.

In our future work, the similarity approach can be utilized to predict resource adaptation (e.g., horizontal or vertical scaling, migration) for an SFC based on its similarity with other existing SFCs. Moreover, we propose to adapt the similarity manager tool and apply it on the substrate level. This adaptation will enable to find the best servers to host the new incoming SFC based on its similarity with a deployed SFC.

CHAPTER 4

DISCUSSION OF THE RESULTS

Over the articles presented in previous chapters, we have proposed two innovative managing tools, relevant to virtualized network services (e.g., NFV) domain, being patented. The first managing tool, which is the policy conflict and redundancy detection and resolution mechanisms (discussed in Chapter 2) proposed to check the consistency of the constraints and polices posed by the SFC request and remove potential conflict and redundancy among policies, and guarantee to deliver a validated SFC request to the placement and chaining algorithms that should be mapped and chained into the underlying resources. The experimental results show that the proposed policy detection tools could rapidly identify and detect conflicts and redundancies among NFV policies. Moreover, our proposed detection mechanisms have the ability to detect conflicts and redundancies for different types of policies (e.g., placement, scaling, migration policies).

The second managing tool which proposed to discover the similarity between the new SFC request and all existing provisioned service (discussed in Chapter 3), using document embedding machine learning model, enable to minimize processing time for the provisioning of a new SFC request by skipping some processing steps (e.g., detecting and resolving the conflicts and redundancies among policies, placement and chaining process) while satisfying all the constraints as well as SLA. The results show that this tool significantly improve the execution time needed for SFC provisioning by occasionally skipping the following two phases: (1) resolving the redundancy and conflict between policies if exists. and/or (2) the placement/mapping phases needed for provisioning the SFC based on the type of the detected similarity.

To the best of our knowledge, we are the first who propose such these managing tools to the NFV framework and add such these functionalities to NFV technology. The results show that both managing tools are an integral part of service provisioning module in network virtualization environments and add more flexibility and enhance the performance for the service provisioning in such environments.

CONCLUSION AND RECOMMENDATIONS

In this work, we have proposed two complementary tools for managing the service provisioning in Network Function Virtualization (NFV). Both tools consider innovative approaches and have led to one regular patent and one provisional application for patent and two journal papers: one accepted for publication and one submitted. The first managing tool, which is a Policy Manager, add new functionalities to NFV system by providing novel policy detection mechanisms in order to validate the network services and provide a continuous detection tool to track different potential conflicts and redundancies among several kind of service policies, and resolve them in an automated way. The experiments show its capability to discover the conflicts and redundancies among hundreds of policies with a low execution time. The second managing tool, which is Service Similarity Manager, offer additional functionality to NFV technology by proposing a method to discover the similarity between the new incoming service requests and the existing services that have been deployed by leveraging a PV-DBOW machine learning embedding model. This tool has been tested for many use cases of NFV system deployed. For all the experiments performed, it was able to reduce the total provisioning time using similarity detection strategy based on which an NFV orchestrator may decide to omit one or more steps (e.g., policy conflict and redundancy detection, placement and chaining) in order to reduce the processing overhead. The combination of these two managing tools add more scalability and power to NFV management and orchestrion layer to enable more agile and flexible service provisioning. Moreover, it enables to achieve the pre-defined objectives (1) having the ability to detect different potential conflicts and redundancies among both placement and resource management policies (e.g., scaling, migration), (2) providing a continuous policy detection and resolution, (3) offering an automated way to resolve the potential conflicts and redundancies, (4) discovering the similarity between the new incoming network service to deploy and all the existing service that already deployed by the NFVO in order to reduce the time for SFC provisioning.

The recommended improvement to our managing tools would be adapting it to handle more than one service request, include more type of policies such as security, or business policies, and to improve many operations in NFV (e.g., resource management and prediction) by leveraging the similarity detection between services.

BIBLIOGRAPHY

Alameddine, H. A., Sebbah, S. & Assi, C. (2017). On the interplay between network function mapping and scheduling in vnf-based networks: A column generation approach. *Ieee transactions on network and service management*, 14(4), 860–874.

Allybokus, Z., Perrot, N., Leguay, J., Maggi, L. & Gourdin, E. (2018). Virtual function placement for service chaining with partial orders and anti-affinity rules. *Networks*, 71(2), 97–106.

Basile, C., Canavese, D., Lioy, A., Pitscheider, C. & Valenza, F. (2016). Inter-function anomaly analysis for correct sdn/nfv deployment. *International journal of network management*, 26(1), 25–43.

Bhamare, D., Samaka, M., Erbad, A., Jain, R., Gupta, L. & Chan, H. A. (2017). Optimal virtual network function placement in multi-cloud service function chaining architecture. *Computer communications*, 102, 1–16.

Bouten, N., Mijumbi, R., Serrat, J., Famaey, J., Latré, S. & De Turck, F. (2017). Semantically enhanced mapping algorithm for affinity-constrained service function chain requests. *Ieee transactions on network and service management*, 14(2), 317–331.

Chen, J., Chiew, K., Ye, D., Zhu, L. & Chen, W. (2013). Aaga: Affinity-aware grouping for allocation of virtual machines. *Advanced information networking and applications (aina), 2013 ieee 27th international conference on*, pp. 235–242.

Deng, J., Hu, H., Li, H., Pan, Z., Wang, K.-C., Ahn, G.-J., Bi, J. & Park, Y. (2015). Vnguard: An nfv/sdn combination framework for provisioning and managing virtual firewalls. *Network function virtualization and software defined network (nfv-sdn), 2015 ieee conference on*, pp. 107–114.

EISGI, N. (2014). Network functions virtualisation (nfv); service quality metrics.

Espling, D., Larsson, L., Li, W., Tordsson, J. & Elmroth, E. (2014). Modeling and placement of cloud services with internal structure. *Ieee transactions on cloud computing*, 4(4), 429–439.

Espling, D., Larsson, L., Li, W., Tordsson, J. & Elmroth, E. (2016). Modeling and placement of cloud services with internal structure. *Ieee transactions on cloud computing*, 4(4), 429–439.

ETSI. (2017). Network functions virtualisation (nfv); management and orchestration; report on policy management in mano; release 3. Consulted at https://www.etsi.org/deliver/etsi_gr/NFV-IFA/001_099/023/03.01.01_60/gr_NFV-IFA023v030101p.pdf.

Figueira, N., Krishnan, R., Lopez, D., Wright, S. & Krishnaswamy, D. (2015). Policy architecture and framework for nfv infrastructures. *Active internet-draft, ietf secretariat, internet-draft draft-irtf-nfvrg-nfv-policy-arch-01*.

Fischer, A., Botero, J. F., Beck, M. T., De Meer, H. & Hesselbach, X. (2013). Virtual network embedding: A survey. *Ieee communications surveys & tutorials*, 15(4), 1888–1906.

Goyal, P. & Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-based systems*, 151, 78–94.

Guthrie, D., Allison, B., Liu, W., Guthrie, L. & Wilks, Y. (2006). A closer look at skip-gram modelling. *Lrec*, pp. 1222–1225.

Hawilo, H., Jammal, M. & Shami, A. (2019). Network function virtualization-aware orchestrator for service function chaining placement in the cloud. *Ieee journal on selected areas in communications*, 37(3), 643–655.

Herrera, J. G. & Botero, J. F. (2016). Resource allocation in nfv: A comprehensive survey. *Ieee transactions on network and service management*, 13(3), 518–532.

Jang, I., Suh, D., Pack, S. & Dán, G. (2017). Joint optimization of service function placement and flow distribution for service function chaining. *Ieee journal on selected areas in communications*, 35(11), 2532–2541.

Jiang, C., Coenen, F. & Zito, M. (2013). A survey of frequent subgraph mining algorithms. *The knowledge engineering review*, 28(1), 75–105.

Konstanteli, K., Cucinotta, T., Psychas, K. & Varvarigou, T. A. (2012). Admission control for elastic cloud services. *Ieee cloud*, pp. 41–48.

Konstanteli, K., Cucinotta, T., Psychas, K. & Varvarigou, T. A. (2014). Elastic admission control for federated cloud services. *Ieee transactions on cloud computing*, 2(3), 348–361.

Larsson, L., Henriksson, D. & Elmroth, E. (2011). Scheduling and monitoring of internally structured services in cloud federations. *2011 ieee symposium on computers and communications (iscc)*, pp. 173–178.

Le, Q. & Mikolov, T. (2014). Distributed representations of sentences and documents. *International conference on machine learning*, pp. 1188–1196.

Li, D., Hong, P., Xue, K. et al. (2018). Virtual network function placement considering resource optimization and sfc requests in cloud datacenter. *Ieee transactions on parallel and distributed systems*, 29(7), 1664–1677.

Li, Y. & Chen, M. (2015). Software-defined network function virtualization: A survey. *Ieee access*, 3, 2542–2553.

Macías, M. & Guitart, J. (2014). Sla negotiation and enforcement policies for revenue maximization and client classification in cloud providers. *Future generation computer systems*, 41, 19–31.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F. & Boutaba, R. (2016). Network function virtualization: State-of-the-art and research challenges. *Ieee communications surveys & tutorials*, 18(1), 236–262.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, pp. 3111–3119.

Moualla, G., Turletti, T. & Saucez, D. (2019). Online robust placement of service chains for large data center topologies. *Ieee access*, 7, 60150–60162.

Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y. & Jaiswal, S. (2017). graph2vec: Learning distributed representations of graphs. *arxiv preprint arxiv:1707.05005*.

Pachorkar, N. & Ingle, R. (2013). Multi-dimensional affinity aware vm placement algorithm in cloud computing. *International journal of advanced computer research*, 3(4), 121.

Pérez-Romero, J., Riccobene, V., Schmidt, F., Sallent, O., Jimeno, E., Fernandez, J., Flizikowski, A., Giannoulakis, I. & Kafetzakis, E. (2018). Monitoring and analytics for the optimisation of cloud enabled small cells. *2018 ieee 23rd international workshop on computer aided modeling and design of communication links and networks (camad)*, pp. 1–6.

Quittek, J., Bauskar, P., BenMeriem, T., Bennett, A., Besson, M. & Et, A. (2014). Network functions virtualisation (nfv)-management and orchestration. *Etsi nfv isg, white paper*.

Rotsos, C., King, D., Farshad, A., Bird, J., Fawcett, L., Georgalas, N., Gunkel, M., Shiomoto, K., Wang, A., Mauthe, A. et al. (2017). Network service orchestration standardization: A technology survey. *Computer standards & interfaces*, 54, 203–215.

Sundararajan, P. K., Feller, E., Forgeat, J. & Mengshoel, O. J. (2015). A constrained genetic algorithm for rebalancing of services in cloud data centers. *2015 ieee 8th international conference on cloud computing*, pp. 653–660.

Tajiki, M. M., Salsano, S., Chiaraviglio, L., Shojafar, M. & Akbari, B. (2018). Joint energy efficient and qos-aware path allocation and vnf placement for service function chaining. *Ieee transactions on network and service management*, 16(1), 374–388.

Wang, H. (2017). A mobile world made of functions. *Apsipa transactions on signal and information processing*, 6.

Wang, L., Lu, Z., Wen, X., Knopp, R. & Gupta, R. (2016). Joint optimization of service function chaining and resource allocation in network function virtualization. *Ieee access*, 4, 8084–8094.

Yan, X. & Han, J. (2002). gspan: Graph-based substructure pattern mining. *2002 ieee international conference on data mining, 2002. proceedings.*, pp. 721–724.

Yang, L., Van Tung, D., Kim, M. & Kim, Y. (2016). Alarm-based monitoring for high availability in service function chain. *2016 international conference on cloud computing research and innovations (icccri)*, pp. 86–91.

Ye, Z., Cao, X., Wang, J., Yu, H. & Qiao, C. (2016). Joint topology design and mapping of service function chains for efficient, scalable, and reliable network functions virtualization. *Ieee network*, 30(3), 81–87.

Zhang, Y., Zhang, D., Vance, N., Li, Q. & Wang, D. (2018). A light-weight and quality-aware online adaptive sampling approach for streaming social sensing in cloud computing. *2018 ieee 24th international conference on parallel and distributed systems (icpads)*, pp. 1–8.

Zhao, F., Zhu, H., Jin, H. & Qiang, W. (2015). A skip-gram-based framework to extract knowledge from chinese reviews in cloud environment. *Mobile networks and applications*, 20(3), 363–369.

Zhao, R. & Iwaihara, M. (2017). Lightweight efficient multi-keyword ranked search over encrypted cloud data using dual word embeddings. *arxiv preprint arxiv:1708.09719*.

Zobaed, S., Haque, M. E., Kaiser, S. & Hussain, R. F. (2018). Nocs2: Topic-based clustering of big data text corpus in the cloud. *2018 21st international conference of computer and information technology (iccit)*, pp. 1–6.

Zou, D., Huang, Z., Yuan, B., Chen, H. & Jin, H. (2018). Solving anomalies in nfv-sdn based service function chaining composition for iot network. *Ieee access*.