12-10-2018

Service Function Graph Design And Embedding In Next Generation Internet

Maryamsadat Jalalitabar
Georgia State University

SERVICE FUNCTION GRAPH DESIGN AND EMBEDDING IN NEXT GENERATION INTERNET

by

MARYAM JALALITABAR

Under the Direction of Xiaojun Cao, Ph.D.

ABSTRACT

Network Function Virtualization (NFV) and Software Defined Networking (SDN) are viewed as the techniques to design, deploy and manage future Internet services. NFV provides an effective way to decouple network functions from the proprietary hardware, allowing the network providers to implement network functions as virtual machines running on standard servers. In the NFV environment, an NFV service request is provisioned in the form of a Service Function Graph (SFG). The SFG defines the exact set of actions or Virtual

Network Functions (VNFs) that the data stream from the service request is subjected to. These actions or VNFs need to be embedded onto specific physical (substrate) networks to provide network services for end users. Similarly, SDN decouples the control plane from network devices such as routers and switches. The network control management is performed via an open interface and the underlying infrastructure turned into simple programmable forwarding devices. NFV and SDN are complementary to each other. Specifically, similar to running network functions on general purpose servers, SDN control plane can be implemented as pure software running on industry standard hardware. Moreover, automation and virtualization provide both NFV and SDN the tools to achieve their respective goals.

In this dissertation, we motivate the importance of service function graph design, and we focus our attention on the problem of embedding network service requests. Throughout the dissertation, we highlight the unique properties of the service requests and investigate how to efficiently design and embed an SFG for a service request onto substrate network. We address variations of the embedding service requests such as dependence awareness and branch awareness in service function graph design and embedding. We propose novel algorithms to design and embed service requests with dependence and branch awareness. We also provide the intuition behind our proposed schemes and analyze our suggested approaches over multiple metrics against other embedding techniques.

INDEX WORDS: Network Function Virtualization, Virtual Network Embedding, Software Defined Networking

SERVICE FUNCTION GRAPH DESIGN AND EMBEDDING IN NEXT GENERATION INTERNET

by

MARYAM JALALITABAR

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

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2018

SERVICE FUNCTION GRAPH DESIGN AND EMBEDDING IN NEXT GENERATION INTERNET

by

MARYAM JALALITABAR

DEDICATION

To my family back home.

ACKNOWLEDGEMENTS

During my PhD, I have received support, encouragement and smiles from many great people. I would have never been able to finish my dissertation without them.

I would like to express my respect and gratitude to my advisor, Dr. Xiaojun Cao. He taught me so many things with great patience: how to think out of the box, how to define scientific problems and write papers, how to prepare for a presentation, how to address the criticism, to name a few. He is supportive and always encourages me to explore new topics independently.

I am thankful to my committee members. Thank you Dr. Anu Bourgeois for all of your support as a mentor in teaching, research and life. Thank you Dr. Wei Li for all the encouraging words and suggestions. Thank you Dr. Jing Zhang for your fantastic mathematical statistic class, where I learned many useful mathematical tools for my research.

My special gratitude and appreciation goes to my mentor, Dr. Raj Sunderraman for his tremendous help, care and guidance in teaching and job search. He is a role model for me, especially in teaching and advising the students.

Finally, I want to thank my family back home and my husband Hesam. They always supported me unconditionally.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

- GSU - Georgia State University
- CS - Computer Science
- NFV - Network Function Virtualization
- VNE - Virtual Network Embedding
- VN - Virtual Network
- SDN - Software Defined Networking
- CAPEX - Capital Expenditure
- OPEX - Operational Expenditure
- SFG - Service Function Graph
- SFC - Service Function Chain
- SFCE - Service Function Chain Embedding
- NSR - Network Service Request
- DAG - Directed Acyclic Graph
- BW - Bandwidth
- LB - Load Balancer
- DPI - Deep Packet Inspection
- FW - Firewall

CHAPTER 1

INTRODUCTION

In this chapter, we introduce the basic concepts of Virtual Network Embedding (VNE), Network Function Virtualization (NFV), and Software Defined Networking (SDN).

Network virtualization [1, 2, 3, 4] is one of the promising technologies for the future Internet. In network virtualization, the primary building block is Virtual Network (VN). A virtual network consists of a number virtual nodes that are interconnected with virtual links. Virtual Network Embedding (VNE) problem deals with the allocation of virtual resources both in terms of nodes and links [5, 6, 7, 8, 9]. VNE is an optimization problem, which aims to identify where the virtual networks elements (nodes and links) will be placed on the physical substrate network. As shown in Figure 1.1, VNE allows several different virtual networks to operate on a single physical infrastructure. Virtual network embedding problem includes two processes: node mapping, which is the embedding of the virtual node (with computational capacity requirement) to a physical node with sufficient computational capacity; and link mapping, which is the mapping of the virtual link (with bandwidth capacity requirement) to the substrate path(s) with enough bandwidth [10, 11, 12, 13, 14]. It is worth noting that when multiple virtual nodes are embedded onto of a single physical node, the process is called revisitation [15].

1.1 Network Function Virtualization

In today's networks, instantiating a new network service such as firewall or deep packet inspection is becoming increasingly difficult. This is due to the proprietary nature of existing hardware appliances, the cost of offering the space and energy for a variety of middleboxes, and the lack of skilled professionals to integrate and maintain these services [16, 17, 18, 19, 20]. Network Function Virtualization (NFV) provides an effective way to decouple network

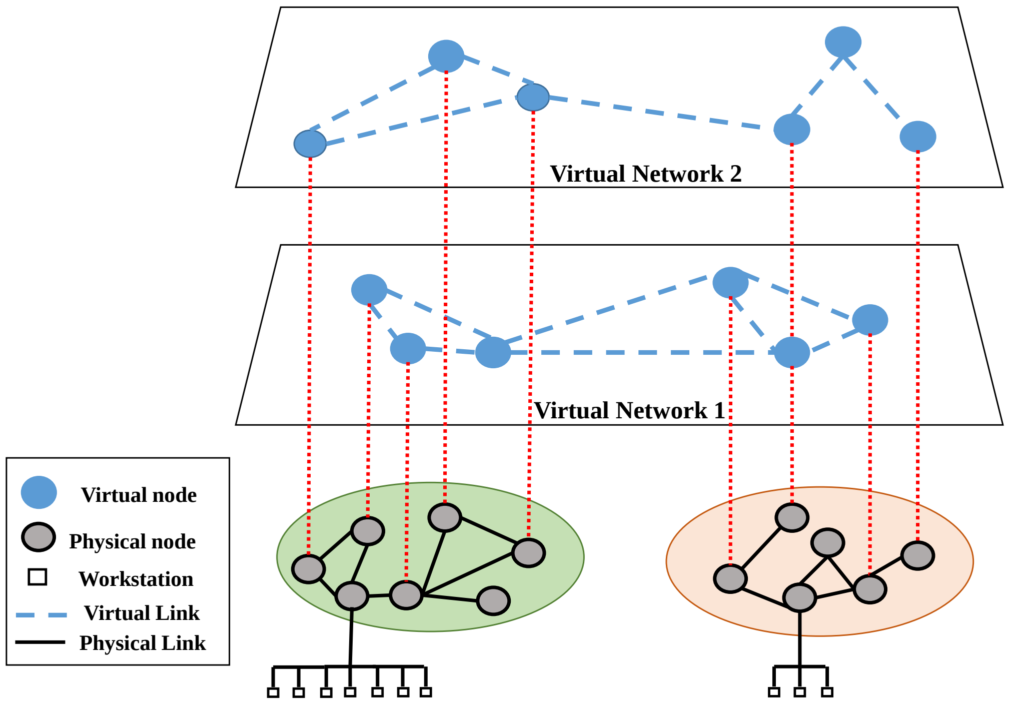


Figure (1.1) An example of virtual network embedding

functions from the vendor specific hardware appliances, enabling the network providers to implement network functions in software running on commodity hardware (i.e., standard servers, storage, and switches). Unlike traditional enterprise networks that deploy network services using the vendor specific proprietary hardware or middleboxes, NFV networks allow these network services or functions to be deployed as Virtual Network Functions (VNFs). Figure 1.2 illustrates the high-level NFV framework. The main three components of the NFV framework are:

1. Virtual Network Function: the software implementation of a network function (e.g., firewall) running on the NFVI.

2. NFV Infrastructure (NFVI): general purpose physical resources that VNFs can be implemented on.

3. NFV Management and Orchestration: which covers all the virtualization-specific man-

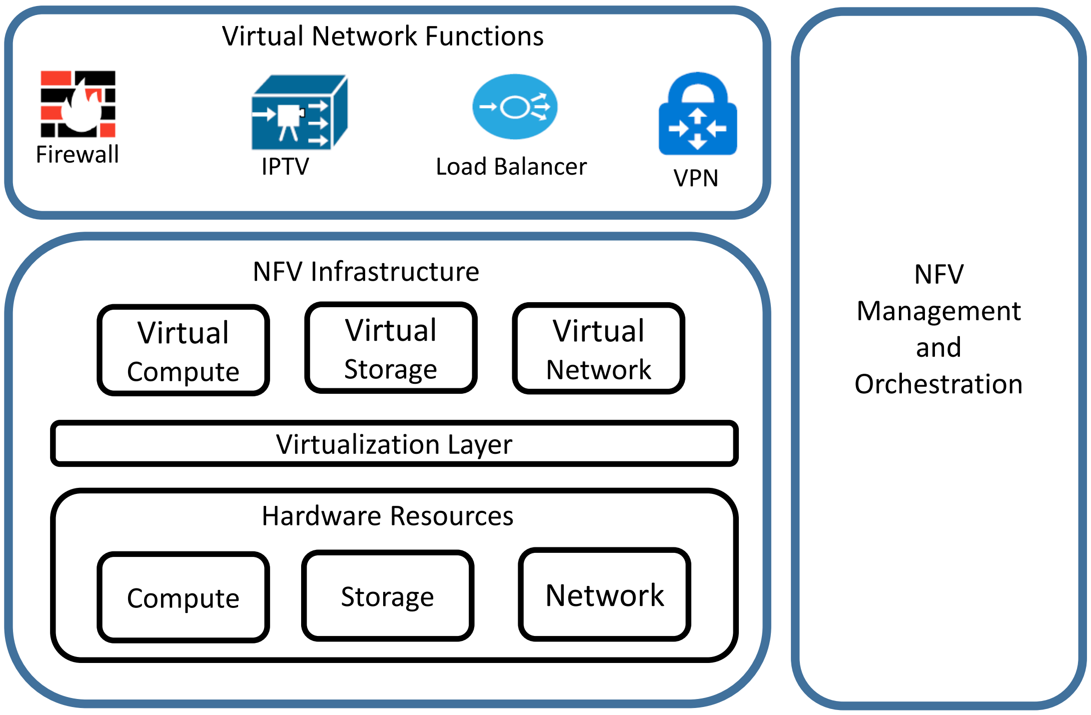


Figure (1.2) High-level NFV framework

agement tasks that are necessary in the NFV framework.

One of the major motivations of Network Function Virtualization is that NFV provides an innovative step toward implementing a network infrastructure with lower CAPEX/OPEX cost by consolidating networking hardware appliances and decreasing the time to market of a new service. Figure 1.3 shows an example of transition from dedicated hardware appliances for network services to software based NFV solutions.

Service Function Graph In the NFV environment, an NFV service request is provisioned in the form of a Service Function Graph (SFG) [21]. The SFG, which can also be in form of a chain (Service Function Chain (SFC)) [22, 23], defines the exact set of actions or Virtual Network Functions (VNFs) that the data stream from the service request is subjected to. These actions or VNFs need to be mapped onto specific physical networks to provide network services for end users. Figure 1.4 shows an example of an SFG, a chain of

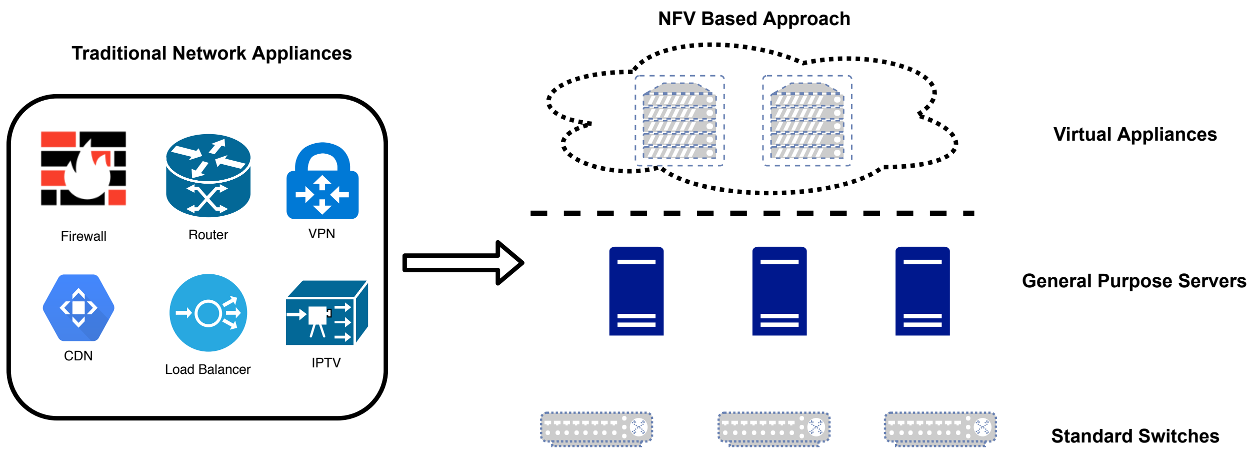


Figure (1.3) Traditional to NFV based approach transition

VNFs, which consists of network functions firewall, NAT and a load balancer.

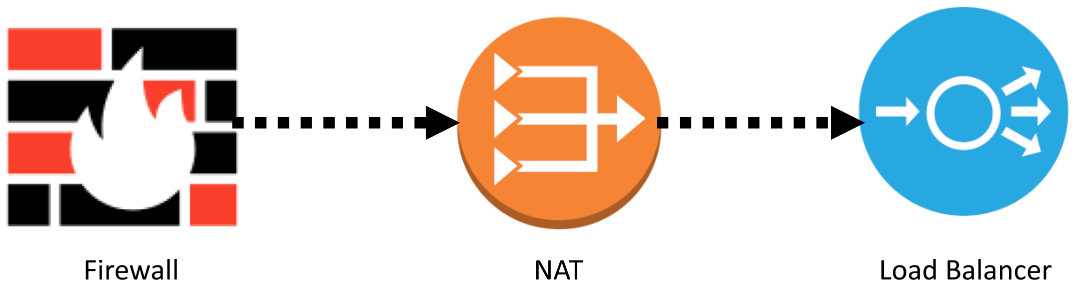


Figure (1.4) An example of a service function graph

1.2 Software Defined Networking

Software Defined Networking (SDN) is a networking framework that separates the network's control logic or control plane from the underlying data plane [17, 24, 25, 26, 27, 28]. With the separation of the control and data planes, network switches become simple forwarding devices and control plane is deployed as a centralized controller [29]. The separation of the planes results in simplifying policy enforcement and network configuration. To separate the control and data planes, well-defined programming interfaces (API) are used. Using API, the controller has direct control over the states of the devices in the data plane. Example of such an API is OpenFlow [30]. An OpenFlow switch has one or more tables (flow tables) for handling the incoming packets. The entries at one flow table specify rules for a subset of the traffic and perform a certain action such as dropping or forwarding the traffic. Figure 1.5 illustrates the high-level view of an SDN framework. Northbound API provides abstraction to the upper layer to develop applications while southbound API defines the instruction set for the forwarding devices.

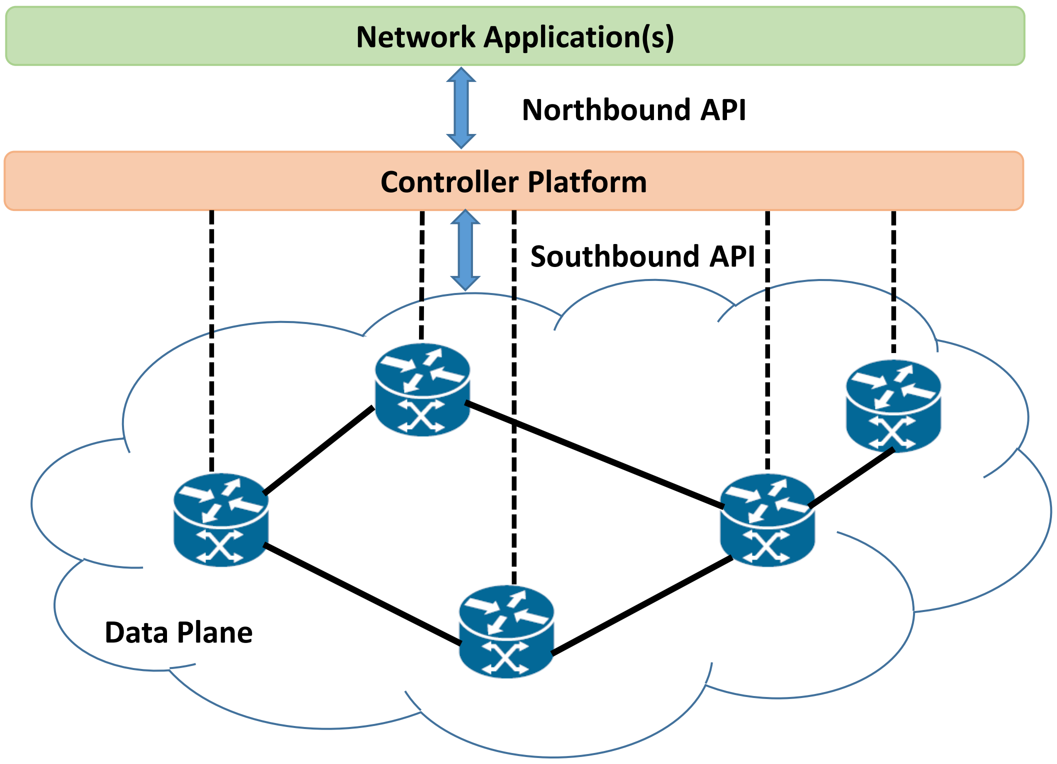


Figure (1.5) High-level view of an SDN architecture

The rest of this dissertation is organized as follows. In Chapter II, a classified overview of network virtualization is presented. In Chapter III, we introduce and model the service function graph design and embedding and present the related challenges. In Chapter IV, V and, VI we present our recent research on the aforementioned challenges. Finally, we highlight the impact of the overall study and conclude the dissertation in Chapter VII.

CHAPTER 2

NETWORK VIRTUALIZATION

In this chapter, we review the related work focusing on three lines of research: virtual network embedding, service function chain embedding and network virtualization in optical networks.

2.1 Virtual Network Embedding

In Virtual network embedding (VNE), given a virtual network consists of virtual nodes and virtual links, the problem is how to embed the given virtual network onto the physical network [31, 32, 33, 34, 35, 36]. The optimization of node/link mapping in VNE has been proven to be NP-hard [37]. Given its NP-hardness, three different types of optimization approaches have been proposed to solve VNE.

1. **Optimal:** The optimal solutions solve the small instances of the problem [38, 39, 40, 41] by creating the Integer Linear Programming (ILP) solutions. Software tools, generally known as solvers ((e.g., CPLEX [42], GLPK[43]).), are used to obtain these optimal solutions. Chowdhury *et al.* formulate the virtual network embedding problem as a mixed integer program through substrate network augmentation [44]. The authors further propose an approach to relax the integer constraints to obtain a linear program. Houidi *et al.* introduce a mixed integer program for the VNE problem [45] that aims to minimize the embedding cost and increase the acceptance ratio rate. Hu *et al.* present path based model for VNE problem, namely P-VNE [46]. By analyzing the dual formulation of the P-VNE model, the authors propose a column generation method [47] to offer the optimal solution. Botero *et al.* define the virtual network embedding energy aware problem. The optimization goal is to embed the virtual network requests in a reduced set of physical devices. The authors use a mixed integer program to

optimally solve the problem. Hu *et al.* present a path based ILP model for the virtual network embedding, namely P-VNE [48]. Based on the dual formulations of the P-VNE model, the authors present a column generation process. This process can be embedded into a branch-and-bound framework to resolve the VNE problem optimally in practice. Hu *et al.* provide an optimal (or near optimal with a per-instance guarantee on its closeness to the optimal solution) method based on iterative process enabling feedbacks between node and link mapping processes [49].

2. **Metaheuristic:** As the optimal solutions are hard to find for large instances of optimization problems, metaheuristics approaches such as ant colony optimization [50] or particle swarm optimization [51] are used to find near-optimal solutions [52, 53, 54]. Fajjari *et al.* propose Ant-Colony-based algorithm to solve the virtual network embedding problem with the goal of minimizing the physical resource usage [55] in terms of bandwidth, power processing and memory. Zhang *et al.* introduce particle swarm optimization-based VN embedding algorithm [56] via the evolution process of particles.

3. **Heuristic:** Heuristic-based methods do not guarantee the optimal solution but they try to find an acceptable solution while achieving low execution time [57, 58, 59, 60, 61, 62]. Zhu *et al.* study on-demand VN assignment problem [63]. The virtual network requests arrive in an online manner and are assigned to the substrate network while minimizing the load balance on the substrate nodes and links. Yu *et al.* study how to simplify the problem of virtual-link embedding [64]. The authors propose an approach to allow the substrate network to embed a virtual link onto a number of substrate paths with a flexible path-splitting ratio. Also, to efficiently handle the online requests, the substrate network periodically re-optimize the mapping of existing virtual links, either by updating the previously selected paths or using the splitting ratios for the existing paths.

2.2 Service Function Chain Embedding

The problem of Service Function Chain Embedding (SFCE) has to design the topology of a service function chain or graph together with the process of node/link mapping. The node mapping and link mapping processes in SFCE are correspondent to the ones in the traditional virtual network embedding problem which is NP-Hard [37]. Hence, similar to VNE, optimization methods have been proposed to solve SFCE [65, 66, 67, 68, 69, 70, 71, 72].

1. **Optimal:** Luizelli *et al.* formulate the network function placement and chaining problem as an integer linear programming [73]. The optimization goal is to minimize the number of VNF instances mapped on the infrastructure. The authors propose a heuristic approach to solve the optimization problem based on a binary search in terms of the number of network function instances. Three models for the topology of the chain are considered: (i) line, (ii) bifurcated path with different endpoints, and (iii) bifurcated path with a single endpoint. Basta *et al.* investigate the virtualization of the Serving Gateway (SGW) and PDN Gateway (PGW) in LTE networks [74]. The authors study the influence of virtualizing or decomposing those two central functions on the data-plane delay as well as the transport network load. The authors formulate the problem as an optimization problem and propose a model for placing the network functions minimizing the network load overheads introduced by the SDN control plane interactions.

2. **Metaheuristic:** Cohen *et al.* address the problem of placement of VNFs in a physical network while minimizing the operational cost [75]. The authors prove that the problem is an instance of the facility-location and generalized assignment problems. They provide near optimal approximation algorithms guaranteeing a placement with theoretically proven performance. Fan *et al.* develop an online scheme to minimize the physical resource consumption with the guarantee of high availability for availability-aware SFC mapping [76]. This online scheme calculates the minimum number of off-site backup VNF service providers that are needed to guarantee a certain degree of avail-

ability for a service chain. Rost *et al.* model the service chain embedding problem with the objective of maximizing the profit by embedding an optimal subset of requests or minimizing the costs when all requests need to be embedded [77]. The authors present a polynomial approximation algorithm which is based on linear programming and randomized rounding techniques. Mijumbi *et al.* formulate the problem of online mapping and scheduling of VNFs [78]. The authors propose greedy algorithms and a tabu search-based heuristic to reduce the flow execution time. The proposed algorithms are coordinated that is performing both mapping and scheduling at the same time.

3. **Heuristic:** Sahhaf *et al.* propose the network function decomposition for the service chaining [79]. The network function decomposition converts a virtual function to a number of refined or abstract functions that are interconnected as a graph. The objective is to minimize the cost of mapping for the service chain by choosing a reasonable network function decomposition. Mohammadkhan *et al.* model the network function placement and routing problem [80] as an mixed integer linear programing. The authors also offer a heuristic approach to solve the problem incrementally and consider online flows without affecting the current flows in the network. Bari *et al.* study the optimization problem of VNF placement with goal of minimizing the overall network OPEX cost [81]. The authors formulate the problem as as a multi-commodity, multi-plant, capacitated facility location problem [82], which is an ILP model. They also develop a heuristic algorithm based on the Viterbi algorithm [83]. Mehraghdam *et al.* formulate the specification and placement of the chains [84] such that the data rate and number of used substrate nodes for the designed chain is minimized. Bruschi *et al.* introduce energy-aware Game-Theory-based approach for embedding the VNFs in an NFV environment. In the proposed model, the VNFs are player of the problem which compete for the substrate network node capacity pool [85], with the goal to minimize the embedding cost.

Beside from aforementioned approaches, some works in the literature consider the online

SFC embedding in which the NFV service requests arrive at different points of time and are embedded upon arrival. Zhang *et al.* formulate the joint problem of the chain placement and scheduling as an instance of bin-packing optimization problem [86]. The authors use the technique of Jackson network to model the VNF chain while maximizing the average resource utilization or minimizing the average response latency. To maximize the average resource utilization, a priority-driven weighted algorithm is proposed. To minimize the average response latency, a heuristic algorithm, called Reverse Complete Karmarkar-Karp, is introduced to schedule the requests. Lukovszki *et al.* propose a deterministic online algorithm which is asymptotically optimal in the class of both deterministic and randomized situation [87]. To prove the NP-completeness, authors present an ILP formulation.

Another challenge in SFC embedding problem is that network may change dynamically due to a number of reasons and is not always static. First, the required resources by the VNFs may change over time. For example, when the traffic decreases DPI need less computing resources. Second, due to the changes in service requests the Quality of the Service (QoS) of the VNFs may change as well. For instance, VNFs can be relocated when the established service requests require low latency. Callegati *et al.* use OpenFlow for steering the traffic flow properly [88]. The authors analyze the complexity of the SDN control plane in a cloud-based edge network implementing NFV. According to the case study, namely Layer 2 (L2) and Layer 3 (L3) edge network function implementations, the authors claim that both L2 and L3 approaches are functionally viable to implement dynamic SFC. Wang *et al.* solve the SFC embedding and scaling problem based on preplanned allocation with bandwidth guarantee for the various workloads [89]. They also introduce concept of VNF instance communication graph to describe the bandwidth demand of each VNF instance and explore the placement requirement for bandwidth savings. The authors design an online heuristic algorithm to achieve approximate optimal allocation.

In [90], we present the service graph design and mapping, in which VNF nodes can have order dependence. In [91], we propose the Dependence Aware Service Function Chain with Adaptive Mapping (D_SFC_AM) to design and map the SFC onto the substrate network while minimizing the resource consumption in the substrate network. In [92], we further optimize D_SFC_AM and embed the SFC onto both IP and optical networks. Beck *et al.* propose a heuristic algorithm that performs the chain composition design and mapping in one coordinated step using the backtracking [93]. The proposed algorithm tries to search valid assignments for the node and link mapping. If the assignment is not found, the algorithm backtracks and discards the last embedding phase. Ma *et al.* investigate the Traffic Aware Placement of Interdependent Middleboxes [94]. The authors propose a comprehensive study on the optimal placement of NFV middleboxes that takes into account both traffic changing ratio as well as different types of middlebox dependency relations.

2.3 Network Virtualization in Optical Networks

Recently, Orthogonal Frequency Division Multiplexing (OFDM)-based Elastic Optical Networks (EONs) that can efficiently select the modulation formats and provide flexible spectrum allocation through assigning continuous finer grained subcarriers, are viewed as the promising candidates for next generation substrate networks [95]. Provisioning network function over EONs has obtained attention due to EON's high bandwidth, low cost, and flexible spectrum allocation. For example, Xia *et al.* discuss the possibility of reducing the costly Optical/Electronic/Optical (O/E/O) conversions for NFV chaining in packet/optical datacenters [96]. Mehmeri *et al.* introduce NaaS (Networking as a Service) orchestration platform called Software-Programmed Networking Operating System for service provisioning over packet/optical networks [97]. Riera *et al.* present the analytical model for the VNFs scheduling with the objective of optimizing the execution time of the network services deployed in an optical network [98].

CHAPTER 3

SERVICE FUNCTION GRAPH DESIGN AND EMBEDDING

Most of the service function chain embedding approaches focus on the embedding phase and the topology design for a given request is not considered. However, different topology designs can result in different costs for embedding a service request. We fill this gap in Service Function Graph Design and Embedding problem. We consider the topology design in the form of a graph for which chain is a special case. Service Function Graph Design and Embedding consists of interconnecting a set of network functions through the physical network to ensure network flows are given the correct treatment. These flows must go through end-to-end services traversing a specific set of network functions. This problem can be decomposed into two processes: (i) design, and (ii) embedding. The design phase consists of designing an order for the network functions. The order can be in shape of a linear chain or graph that can satisfy the end-to-end requests. In the embedding process, VNFs are placed onto to virtual machines running on commodity servers. Finally, the requested functions are connected, which consists of creating physical paths that interconnect the network functions.

3.1 Substrate Network

The substrate network is modeled as an undirected graph $G_S = (N_S, E_S)$, where N_S and E_S refer to the set of substrate nodes and links, respectively. Each substrate node in N_S is capable to host some VNF nodes while offering a certain amount of computing capacity. We use $c_s \in \mathbb{Z}^+, (\forall s \in N_S)$ to represent the available computing resources (i.e., CPU) of substrate node s. $f_s \subseteq F$ is the available network functionality of substrate node s, where F is a set of commonly used network functions. For each substrate link $(s, t) \in E_S, (s, t \in N_S)$, $B_{st} \in \mathbb{Z}^+$ represents the available bandwidth. For instance, the available network functions and computing resources for the nodes of a substrate network is shown in Table 3.1.

3.2 NFV Service Request

An NFV service request is represented as 2-tuple $NSR =< N_V, B_V >$, where N_V is the set of VNF nodes and $B_V \in \mathbb{Z}^+$ is the requested bandwidth for the network request. Each VNF node $v \in N_V$ requests some computing resources $c_v \in \mathbb{Z}^+$, and a network function f_v. The function f_v is selected from the set of available functions F offered by the substrate network. Without loss of generality, we assume that each VNF node requests a unique network function. In other words, no two VNF nodes request the same network function in an NSR. As an example, Table 3.2 shows a service request consisting of three VNF nodes and a requested bandwidth for the NSR.

Figure 3.1 shows an example of a constructed SFC: $v_1 \rightarrow v_2 \rightarrow v_3$, which consists of three VNF nodes, i.e., $v_i\,(i = 1, 2, 3)$. These VNF nodes are connected or chained with VNF links. Each VNF node demands a specific network function and certain computing resources (e.g., CPU) as shown in Table 3.2. The available bandwidth of substrate link (for the substrate network in Table 3.1) $s_1 - s_2$, $s_2 - s_3$, $s_3 - s_4$, and $s_2 - s_4$ is $15Gbps$, $15Gbps$, $15Gbps$, and $5Gbps$, respectively. Accordingly, the constructed SFC can be mapped onto the substrate network to form a service function path $s_1 \dashrightarrow s_2 \dashrightarrow s_3 \dashrightarrow s_4$. To accommodate the NFV service request NSR_1, VNF nodes v_1, v_2 and v_3 are mapped to substrate nodes s_1, s_2, and s_4, respectively. The VNF links $v_1 \rightarrow v_2$, and $v_2 \rightarrow v_3$ in the constructed SFC are mapped to the physical paths $s_1 \dashrightarrow s_2$ and $s_2 \dashrightarrow s_3 \dashrightarrow s_4$, respectively, as shown by the dash line in Figure 3.1. Note that substrate link $s_2 - s_4$ cannot be used by NSR_1 due to the lack of enough bandwidth.

Table (3.1) Substrate Network

Substrate Node	s_1	s_2	s_3	s_4
Available CPU	45	55	20	35
Offered Functionality	Firewall	Load Balance	Encryption	Encryption

Table (3.2) $NSR_1 = < v_1, v_2, v_3, 10Gbps >$

VNF Node	v_1	v_2	v_3
Network Function	Firewall	Load Balance	Encryption
CPU Demand	20	25	30

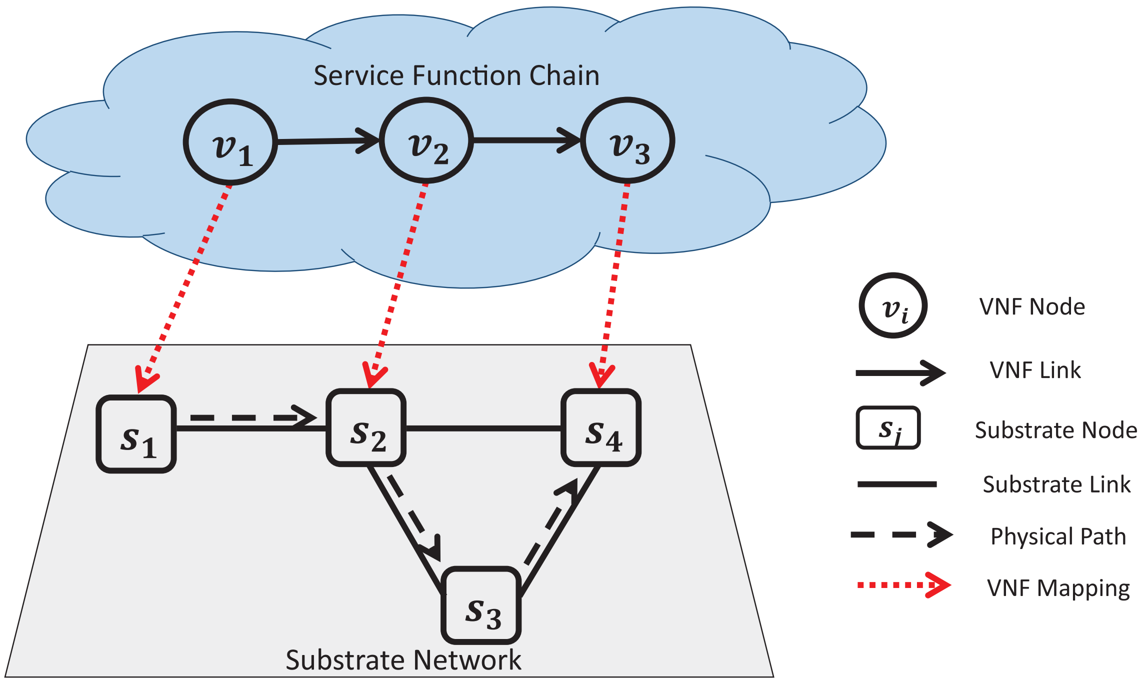


Figure (3.1) An example of an embedded NSR

3.3 Service Function Graph Design and Embedding (SFG_DE) Problem

Definition of SFG_DE problem: Given an NFV service request $NSR =< N_V, B_V >$, and a substrate network $G_S = (N_S, E_S)$, the Service Function Graph Design and Embedding problem can be defined as how to design a graph and map this graph onto the substrate network G_S while satisfying the following constraints:

Node Mapping Constraint Each VNF node $v \in N_V$ must be mapped to exactly one substrate node $s \in N_S$ that has enough computing resources and the required functionality.

We use decision variable X_{ij} to represent the node mapping ($\forall i \in N_V, \forall j \in V_S$).

$$X_{ij} = \begin{cases} 1, & \text{if VNF } i \text{ is allotted to subsrate node } j \\ 0, & \text{otherwise} \end{cases}$$

The following parameters are given for the network function at substrate node i.

- δ_i^f : 1 if substrate node i offers network function f.
- ω_p^f: 1 if network function f is requested by VNF p.

$$\omega_p^f * X_{pi} \leq \delta_i^f, \forall p, f, i$$

Eq. (3.1) ensures that each VNF in a service request is allotted to only one substrate node. Eq. (3.2) states that no more than one VNF resides in the same substrate node.

$$\sum\nolimits_{j \in V_S} X_{ij} = 1, \forall i \in N_V \tag{3.1}$$

$$\sum\nolimits_{j \in V_S} X_{ij} \leq 1, \forall j \in V_S \tag{3.2}$$

Link Mapping Constraint The VNF link between two consecutive VNF nodes in the designed SFG must be mapped to a substrate link or physical path with enough bandwidth in the substrate network. We use decision variable $Y_{i,j}^{p,q}$ to represent the link mapping ($\forall p, q \in N_V, \forall (i,j) \in E_s$).

$$Y_{i,j}^{p,q} = \begin{cases} 1, & \text{link } (i,j) \text{ is on the path between the nodes used for mapping } p \text{ and } q \\ 0, & \text{otherwise} \end{cases}$$

For each link in the substrate network, Eq. (3.3) ensures that the sum of the required bandwidth of all paths using the link is less than or equal to the bandwidth capacity of the link.

$$\sum\nolimits_{\forall p,q \in N_V} \sum\nolimits_{\forall (i,j) \in E_s} Y_{i,j}^{p,q} * B_V \leq B_{ij} \tag{3.3}$$

SFG_DE Objective An optimization objective of SFG_DE problem can be minimizing the bandwidth consumption in the substrate network.

$$\text{Minimize} \sum\nolimits_{\forall (i,j) \in E_S} \sum\nolimits_{\forall p,q \in N_V} Y_{i,j}^{p,q} * B_p$$

3.4 Dependence-awareness in Service Function Graph Design and Embedding

Dependence constraints can significantly affect the system performance during the embedding process. When designing the topology for the service request, if there is a dependence constraint from v_2 to another node v_1, then v_2 must be placed after the node v_1 in the constructed topology so that traffic goes through v_1 before arriving at node v_2. For instance, an IPSec decryptor usually has to be placed before a NAT gateway [99], while a VPN proxy can be placed either before or after a firewall [100]. Hence, the presence of the dependence constraints poses new challenges to the SFCE problem. We use the following example to further explain the importance of the dependence constraint in the SFG_DE problem.

An Example of Dependence-awareness Table 3.3 shows an example of an NFV service request $NSR =< N_V, B_V >$, where $N_V = \{v_1, v_2, v_3\}$, the dependence constraint is $f^1 \mapsto f^3$ and $B_V = 20Gbps$. The service request consists of three VNF nodes and there is one dependence constraint for NSR. The amount of the requested bandwidth for the NSR is 20Gbps. Table 3.4 depicts the available CPU and functionality at each node in the substrate network. The available bandwidth of substrate link $s_1 - s_2$, $s_2 - s_3$, $s_3 - s_4$, and $s_2 - s_4$ is $35Gbps$, $25Gbps$, $30Gbps$, and $45Gbps$, respectively. Based on the given dependence constraint, two of the possible graph options (in the form of a chain) that do not violate the constraint are: $Ch_1 : v_1 \rightarrow v_2 \rightarrow v_3$ and $Ch_2 : v_1 \rightarrow v_3 \rightarrow v_2$. One can embed Ch_1 and Ch_2 onto the substrate network as shown in Figure 3.2(a) and 3.2(b), respectively. The created Service Function Path (SFP) for Ch_1 and Ch_2 is $s_1 \dashrightarrow s_2 \dashrightarrow s_3$ and $s_1 \dashrightarrow s_2 \dashrightarrow s_3 \dashrightarrow s_4$, respectively. One can see that the service function path for Ch_2 uses more substrate links and bandwidth than Ch_1 does.

Table (3.3) NFV service request with dependence constraint NSR

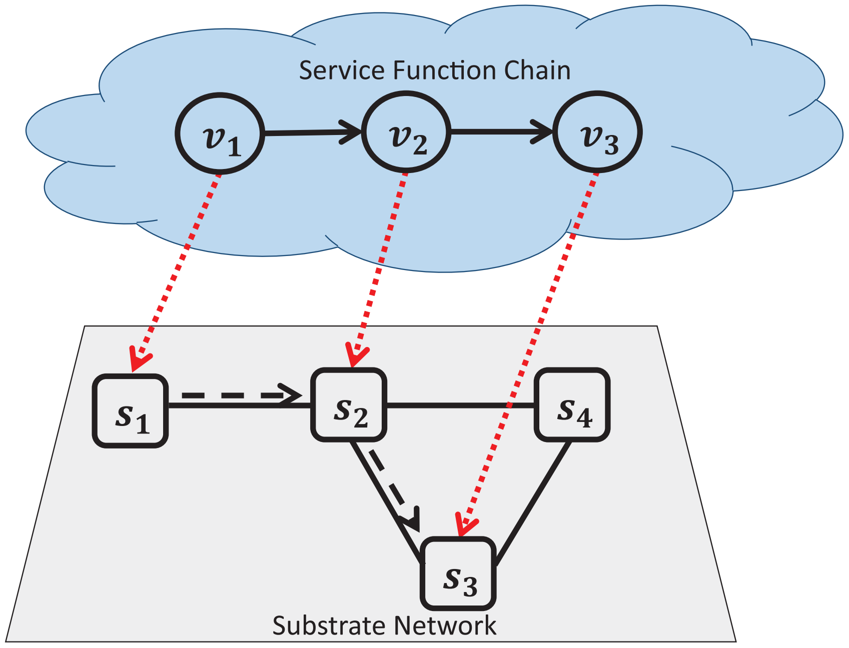
VNF Node	v_1	v_2	v_3
Network Function	f^1	f^2	f^3
CPU	10	30	20
Dependence Constraint	$f^1 \mapsto f^3$		
Requested Bandwidth	20Gbps		

Table (3.4) The substrate network for embedding NSR

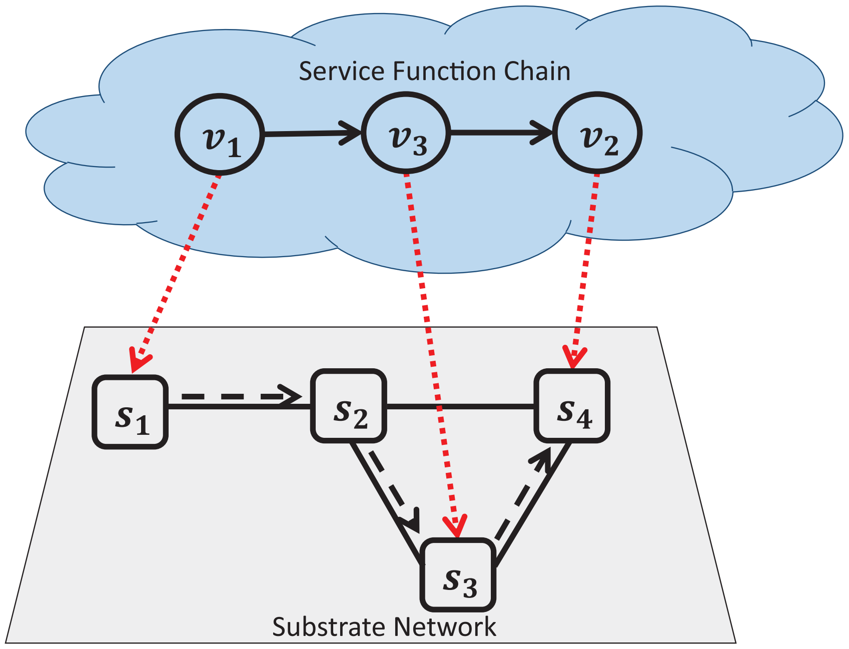
Substrate Node	s_1	s_2	s_3	s_4
Available CPU	40	50	25	45
Offered Functionality	f^1	f^2	f^3	f^2

The above example shows that different design for the service function graph can affect the bandwidth consumption in the substrate network. In fact, how to efficiently construct the graph and map it onto the substrate network while considering the dependence constraints among the VNF nodes is different from the traditional SFCE and VNE. Table 3.5 shows a comparison among VNE, SFCE, and SFG_DE in terms of network topology and node constraints.

The node mapping and link mapping processes in SFG_DE are correspondent to the ones in the traditional virtual network embedding problem which is NP-Hard [37]. Accordingly, we propose efficient heuristic algorithms to solve SFG_DE problem in the following chapters.



(a)



(b)

Figure (3.2) An example of embedding an NFV service request with dependence constraints

Table (3.5) Techniques. VNE, SFCE, and SFG_DE

	Is requested virtual topology given?	Are there node dependencies?
VNE	Yes	No
SFCE	No	No
SFG_DE	No	Yes

3.4.1 Joint Service Function Graph Design and Mapping for NFV with Priority Dependence

How to efficiently construct the graph and map it onto the substrate network while considering the dependence constraints among the VNF nodes is different from the traditional SFCE and VNE. Some of the works in the literature [93] study the problem of SFG_DE with dependence in a sequential manner. That is, in the presence of dependencies, graph is constructed (in the form of a chain) and then mapped onto the substrate network. In chapter 4, we present Service Function Graph Design and Mapping for NFV with Priority Dependence which considers creating the topology of the service request in the form of a graph while embedding the request onto the substrate network.

3.4.2 Joint Dependence-Aware Service Function Chain Design and Mapping

NFV may allow the delivery of end-to-end services through service function chaining. In other words, a service request can be provisioned in the form of a Service Function Chain (SFC) [101, 102, 103]. The SFC defines the exact sequence of actions (i.e., VNF nodes) the data stream from this service request is subjected to.

In chapter 5, we propose the Dependence-Aware Service Function Chain Design and Mapping algorithm to conduct the (*i*) SFC design, and (*ii*) SFC mapping. The former process will construct a service chain for an NFV service request. The latter process will map the constructed chain onto a substrate network via VNF node/link mapping. The process of node mapping in SFC mapping allocates the substrate computing resource for the VNF nodes. The process of link mapping finds a suitable physical path with enough bandwidth resource for each VNF link.

3.5 Branch-awareness in Service Function Graph Design and Embedding

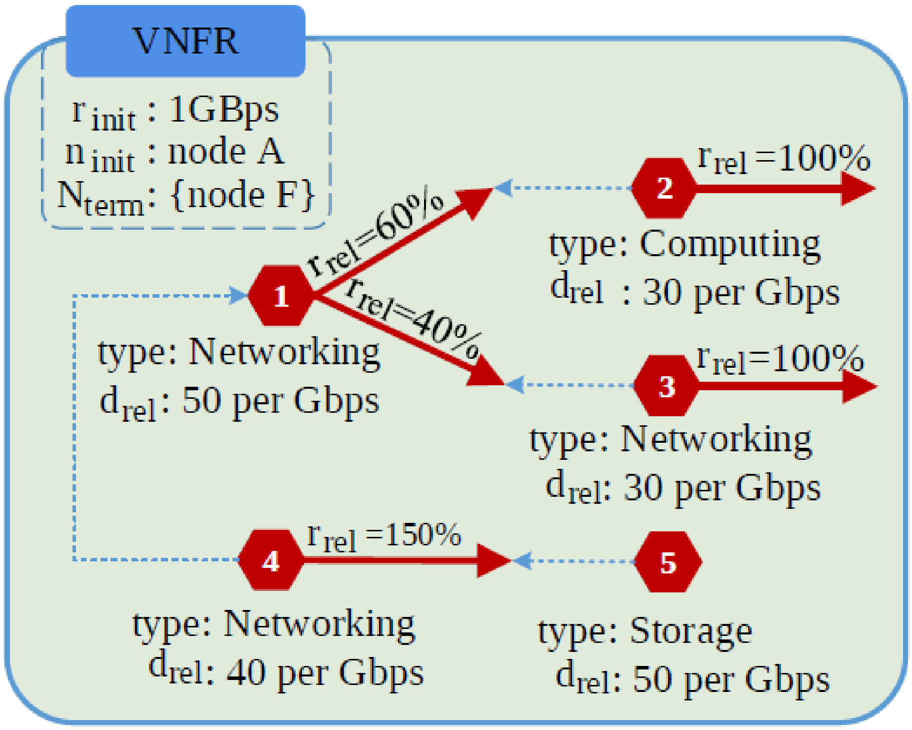
Most of studies in SFG design and embedding implicitly assume that the service functions are ordered in a linear order to allow traffic flow passing through the chained functions in sequence. However, as to be elaborated in chapter 6, the traffic flow could fork due to multiple reasons (e.g., load balancing). The branching of traffic indicates that the packet flow may be directed to multiple different paths, resulting in the need to view the service request as a more complex mesh-like service function graph.

CHAPTER 4

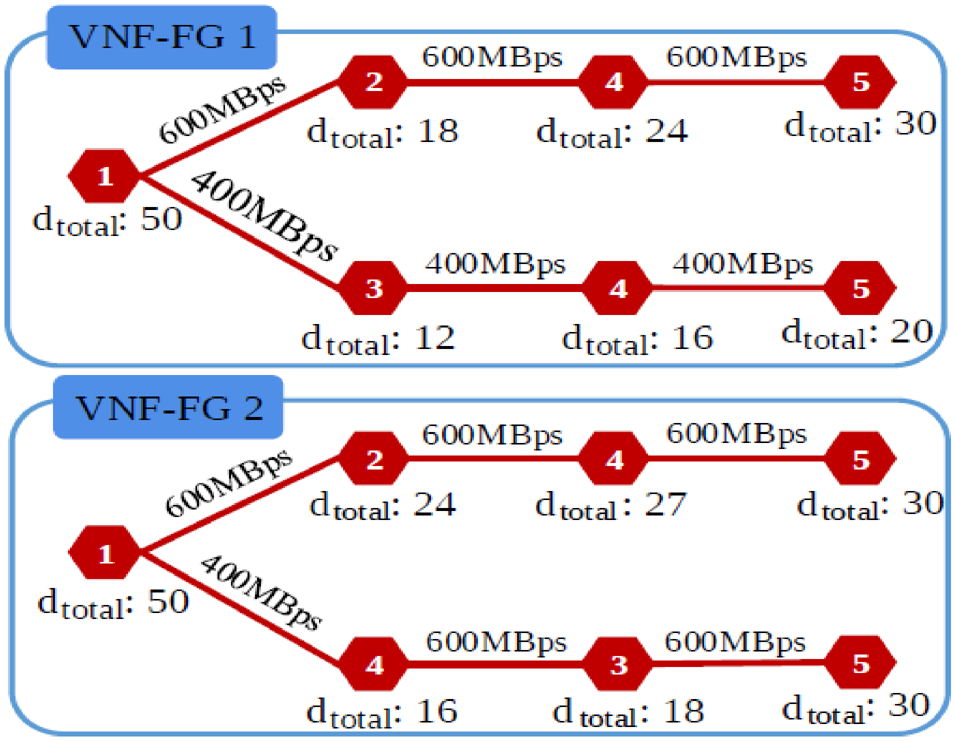
SERVICE FUNCTION GRAPH DESIGN AND MAPPING FOR NFV WITH PRIORITY DEPENDENCE

An Example of Sequential Service Function Graph Design and Embedding with Dependence-awareness Beck et al. propose a heuristic algorithm (CoordVNF) that performs the chain composition design and mapping in one coordinated step using the backtracking [93]. Figure 4.1(a) from [93] shows an VNF Request (VNFR). The initial data rate of the flow is defined as r_{init} and the substrate nodes where the flow initiates and terminates are specified A and F, respectively. d_{rel} specifies the relative processing capacity demands of a VNF. d_{total} shows the total demand that is calculated based on the relative processing capacity and the amount of traffic that is routed to that VNF instance. Depending on the ordering of the VNFs, bandwidth demands of the network flow change. First the chain composition is performed and one valid VNF-FG is chosen (Figure 4.1(b)). Second, CoordVNF is used to solve the embedding problem. CoordVNF recursively tries to find valid assignment options for the VNF instances; if the assignment is not found, the algorithm backtracks and discards the last embedding phase.

In the above example, the chain design and embedding are done sequentially. That is, the chain is fixed before the embedding process. Hence, the resource information of the substrate network is not considered in the chain design phase. However, different design for the service function graph can affect the bandwidth consumption in the substrate network. In this chapter, we present an approach to efficiently construct the graph and map it onto the substrate network while considering the dependence constraints among the VNF nodes.



(a)



(b)

Figure (4.1) Chaining of VNFs

4.1 Substrate Network

The substrate or physical network is modeled as an undirected weighted graph $G_s = (N_s, L_s)$, where N_s is the set of physical nodes and L_s is the set of substrate links. Each substrate node $p \in N_s$ offers a specific amount of computation resources C_p and a unique network function F_p. The physical link between substrate nodes u and v, $e_{uv} \in L_s$, has a certain bandwidth capacity, denoted by $B_{u,v}$.

4.2 Network Service Request

To consider the dependence constraints among the VNFs, we model a service request as a triplet $SR =< N_v, V_{fd}, D_v >$, where N_v represents the set of VNF nodes, V_{fd} is the sink virtual node, D_v represents the dependence requirements between the VNF nodes. Each VNF node $i \in N_v$ requests a specific network function F_i, computing resources C_i, outgoing bandwidth BW_i. Unlike the VNF nodes in N_v, the sink node V_{fd} is designated as the last node to be added in the virtual topology, which requires a certain network function and computing resources, without outgoing bandwidth requirements. A VNF node can have dependence on another node. To show the dependence between two VNF nodes, the greater sign $>$ is used and $i > j$ means that traffic flow will pass through VNF node i before going to the VNF node j. Since the traffic flow will pass through all of the nodes in N_v reaching the sink node V_{fd}, we have $i > V_{fd}$ for all $i \in N_v$. It is assumed that no two VNF nodes in N_v request the same function, which means $F_i \neq F_j$ if $i \neq j$ for all $i, j \in N_v$.

4.3 Service Function Graph Design and Mapping for NFV with Priority Dependence

The problem of optimizing virtual topology design and mapping in the presence of priority dependence (SFG_PD) can be defined as: given the SR request $SR =< N_v, V_{fd}, D_v >$ and substrate network $G_s = (N_s, L_s)$, how to design and map the SFG topology onto the substrate network with minimal resource consumption while satisfying the following constraints:

(*i*) for each VNF node, it is mapped to a substrate node meeting the function and computing requirements; (*ii*) the outgoing link of a VNF node is mapped to a substrate path meeting the bandwidth requirements; and (*iii*) the service function graph is constructed meeting the priority dependence between VNFs. Algorithm 1 show the pseudocode of SFG_PD.

Algorithm 1 SFG_PD mapping with dependent directional acyclic graph (SFG_DAG)

```
Require: SR, $G_s$
Ensure: Mapped functions and constructed virtual topology
 1: $SortedNF$ = Sort VNFs in descending order;
 2: $CandidateList$ = Substrate candidate node list for each VNF;
 3: Update $SortedNF$, initialize $mappedFuncs$ and $G_v$, total_bw_cost = 0;
 4: for all $F$ in $SortedNF$ do
 5:    for all $v$ in CandidateList do
 6:       sum_bw_cost = 0;
 7:       Initialize shortest path list $SP$;
 8:       if $F$ has a dependence on a mapped node $F_v$ (i.e., $F_v > F$) then
 9:          Call DirectDependence();
10:       else if a mapped node $F_v$ has a dependence on $F$ (i.e., $F > F_v$) then
11:          Call PrecedentDependence();
12:       else
13:          Call NoDependence();
14:       end if
15:    end for
16:    Get the candidate node $v_F$ with minimum sum_bw_cost;
17:    Map $F$ to $v_F$;
18:    Find the corresponding virtual function nodes that are mapped to the substrate nodes
       in $SP$;
19: end for
20: Map the sink node to the closest substrate node;
```

As shown in Algorithm 1, the inputs of SFG_PD algorithm are the information of substrate network (G_S) and Network Service Request (NSR). The algorithm starts with identifying the order of VNFs to be mapped onto the substrate network. The order can be generated based on the requested outgoing bandwidth, the requested CPU computing resources or randomly. The sorted VNFs are placed in the *SortedNF* list. In Line 2, the *CandidateList* for each VNF is generated based on the amount of requested computing resources and the network function. The first VNF in this list is then mapped onto the substrate network by selecting a substrate node from its list of candidate nodes. Then, the mapped VNF node is removed from the $SortedNF$ list and is added to the list of mapped VNF nodes (i.e., $mappedFuncs$) and G_v (i.e., the mapped SFG graph) is initialized, as showed in Line 3. For any VNF node, say F to be mapped, the dependence between F and the mapped VNFs (e.g., F_v) in $mappedFuncs$ is checked, which yields three possible cases as follows.

1. F has a dependence on the nodes in the $mappedFuncs$;
2. Some nodes in $mappedFuncs$ have a dependence on F;
3. F has no dependent relationship with the nodes in $mappedFuncs$;

Each of above cases is handled by different procedures as shown in Algorithm 2, 3 and 4. To facilitate the explanation of these algorithms, we elaborate the processes using the example in the next section.

Algorithm 2 DirectDependence()

```
Require: VNF request, $G_s, G_v$
Ensure: Updated $mappedFuncs$, $SortedNF$
1: Identify the dependent directional acyclic subgraph $SG_v$ in $G_v$ with $F_v$ as the destination;
2: Find the mapped substrate subgraph $SG_s$ of $SG_v$;
3: Find the shortest path $P_s$ from $v$ to $SG_s$ that has enough bandwidth;
4: Add $P_s$ to $SP$; update $mappedFuncs$ and $SortedNF$;
```

Algorithm 3 PrecedentDependence()

```
Require: VNF request, $G_s, G_v$
Ensure: Updated $mappedFuncs$, $SortedNF$, $G_v$
1: Identify the dependent directional acyclic subgraph $SG_v$ in $G_v$ with $F_v$ as Source;
2: Find the mapped substrate subgraph $SG_s$ of $SG_v$;
3: Find the shortest path $P_s$ from $SG_s$ to $v$ that has enough bandwidth;
4: Add $P_s$ to $SP$; update $mappedFuncs$ and $SortedNF$;
```

Algorithm 4 NoDependence()

```
Require: VNF request, $G_s, G_v$
Ensure: Updated $mappedFuncs$, $SortedNF$, $G_v$
1: Find the shortest path $P_s$ from $v$ to $G_s$ that has enough bandwidth;
2: Add $P_s$ to $SP$; update $mappedFuncs$ and $SortedNF$;
```

4.4 An Example of SFG_PD

To help explain the proposed SFG_DAG in Algorithm 1, we use the sample service request in Table 4.1 to demonstrate the process of VNF mapping and the virtual topology construction. As shown in Table 4.1, there are four VNF nodes (i.e., F_1, F_2, F_3, F_4) and two dependence constraints. The dependence between nodes F_1 and F_3 is denoted as $F_1 > F_3$, which indicates the traffic flow should pass through F_1 before going to F_3. Similarly, the dependence between F_2 and F_4 indicates F_2 should be located ahead of F_4 in the constructed SFG. We assume that the substrate network is an undirected graph with five nodes (i.e., A, B, C, D, E) offering the requested functions and CPU resources as shown in Figure 4.2. In Figure 4.2, the dash line between a VNF and a substrate node denotes that the corresponding VNF node is mapped onto the connected substrate node.

As shown in Figure 4.2(a), Node F_3 is the first to be mapped and substrate node A can accommodate it. The next VNF to be considered is F_2. As F_2 does not have dependence on the mapped F_3, the procedure in NoDependence() is executed, where the shortest path [104] from the candidate substrate node of F_2 (i.e., D) to node A is identified. As a result, F_2 is mapped onto D and the amount of requested BW by F_2 is reserved along the physical links of the path from D to A. In the mean time, a virtual link from F_2 to F_3 is added onto the virtual SFG graph as shown in Figure 4.2(b).

In the next iteration, VNF F_1 is the current VNF node to be mapped. F_1 has a dependence on the mapped F_3. Thus, Algorithm 2 is called to find a directional path destined to F_3. The algorithm compares the minimum cost paths from the candidate substrate nodes of F_1 (i.e., E) to any substrate node that provides the mapping for VNFs in the directional acyclic subgraph. In this case, the directional acyclic subgraph includes F_2 and F_3 and the corresponding substrate nodes are D and A as shown in Figure 4.2(c). The minimal cost paths for $E- > D$ and $E- > A$ are compared. The path $E- > D$ is then selected, which subsequently adds a directional virtual link from F_1 to F_2 onto the virtual SFG graph.

The last VNF node to address is F_4, which has a dependence from one of previously mapped functions that is F_2. Algorithm 3 is then invoked to check the dependence on the precedent mapped functions. As shown in Figure 4.2(d), a directional acyclic subgraph starting from F_2 is identified, which includes VNF nodes F_2 and F_3. The corresponding substrate nodes are D and A. As the only candidate node of F_4 is B. Hence the minimal cost paths from D or A to B are calculated. As the path from A to B in the substrate network yields less cost than the path from D to B, VNF F_4 is mapped to B. And the algorithm adds a virtual link from the VNF node that resides in A (i.e., F_3) to VNF F_4 as shown in Figure 4.2(d).

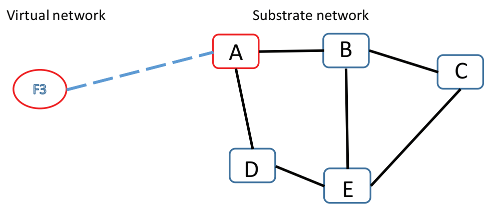
Finally, the sink node is mapped by finding the closest substrate node to all of the substrates nodes that host a VNF node in G_v. For the example in Figure 4.2(e), the sink node will be mapped to node C and a virtual link from F_4 to sink D_v is added to the SFG and mapped along physical path from node B to node C.

Table (4.1) A Service Request Example

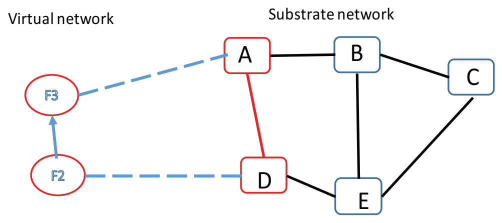
$SR = (F_1, F_2, F_3, F_4, V_{fd})$
$CandidateList = (F_1 : E, F_2 : D, F_3 : A, F_4 : B)$
$SortedNF = (F_3, F_2, F_1, F_4)$
Dependence requirements: $F_1 > F_3$; $F_2 > F_4$;

4.5 Performance Evaluation

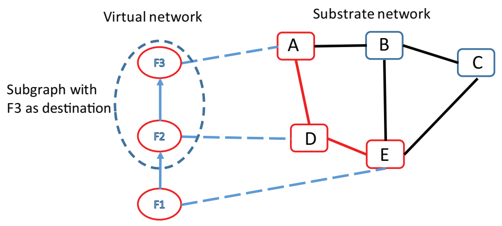
To evaluate the proposed SFG_DAG algorithm, we implement the algorithm by using the NSF network with 14 nodes and 21 edges as the substrate network and conduct extensive simulations to obtain the average results as shown in Figure 4.3, 4.4 and 4.5. We randomly



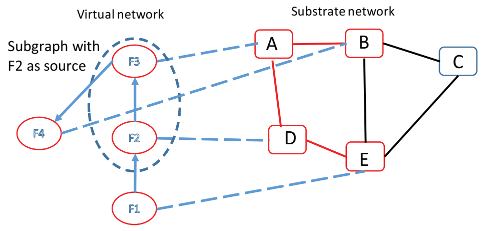
(a) Mapping VNF F_3



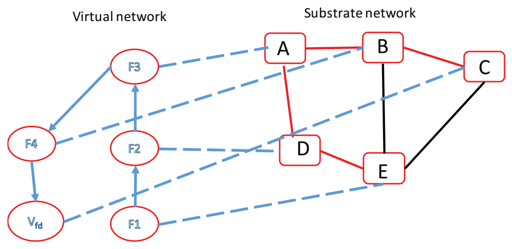
(b) Mapping VNF F_2



(c) Mapping VNF F_1



(d) Mapping VNF F_4



(e) Mapping VNF V_{fd}

Figure (4.2) An Example of SFG_DAG

generate the virtual function service request with N VNF nodes. The function of a VNF node is randomly selected from the functions offered by the substrate network. The dependence relationships between the VNF nodes are also randomly generated. To study how the requested resources impact the performance of the proposed SFG_DAG algorithm, we vary the amount of VNF nodes, requested maximum bandwidth (MBW), and requested maximum CPU (MCPU) resources to compare the total bandwidth consumption by SFG design and mapping process. Accordingly, the CPU request of a VNF node and outgoing bandwidth request are randomly generated, in the range of [0, MCPU] and [0, MBW], respectively. In addition, to investigate the performance of the proposed SFG_DAG algorithm with different order of VNF nodes to be mapped, we sort the VNF nodes according to the following three cases: *i*) randomly; *ii*) requested CPU computing resources; and *iii*) requested outgoing bandwidth. The curves for case $(i), (ii)$, and (iii) are denote by "Random", "CPU" and "BW", respectively.

In Figure 4.3, 4.4 and 4.5, the Y-axis denotes the total bandwidth consumed by the SFG_DAG algorithm in the substrate network. The X-axis in Figure 4.3, 4.4 and 4.5 represents the number of VNF nodes, MCPU and MBW, respectively. From these figures, one can see that sorting the VNF nodes based on the requested bandwidth in case (iii) yields the best performance when compared to case (i) and (ii). This is because accommodating virtual nodes with larger amount of outgoing requested bandwidth at the early iterations of the SFG_DAG algorithm will likely use shorter physical paths, which may lead to less total bandwidth consumed. On the other hand, accommodating virtual VNF node with larger amount of requested CPU resources at the early iterations of the SFG_DAG algorithm may cause some physical links lack of bandwidth to satisfy VNF nodes with high outgoing bandwidth along the shortest path (during the later iterations). As a result, a longer alternative path may have to be employed to accommodate a VNF node with large requested outgoing bandwidth. In addition, each substrate node is equipped with limited CPU resources. Accommodating the VNF nodes in a random order may force some VNF nodes to choose longer paths due to the shortage of CPU in the closed substrate nodes. In other words, in case

(i), the random order of processing VNF nodes may cause longer paths employed by heavy virtual links due to the limited bandwidth and CPU resources in the substrate network. As a result, accommodating the VNF nodes in a random order yields the worst performance.

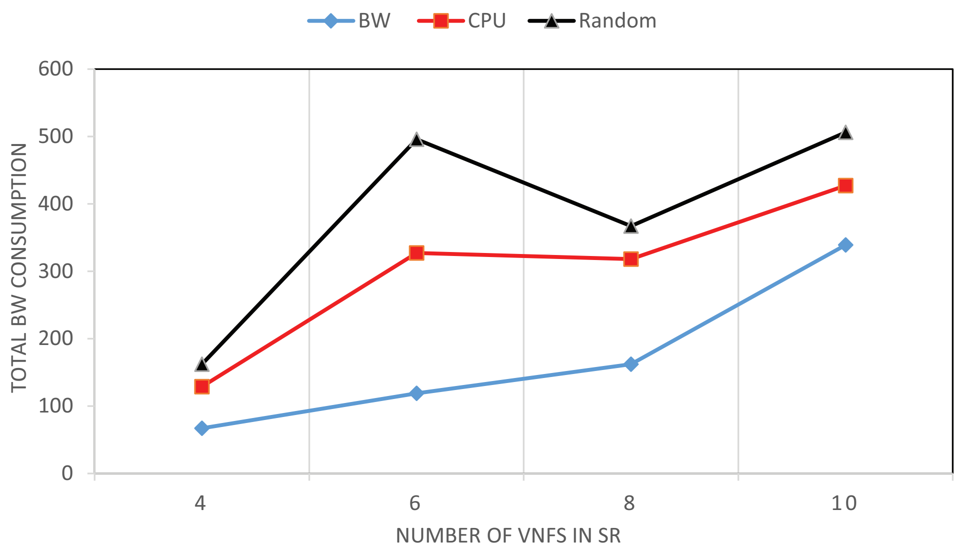


Figure (4.3) The Impact of VNF Counts in SR

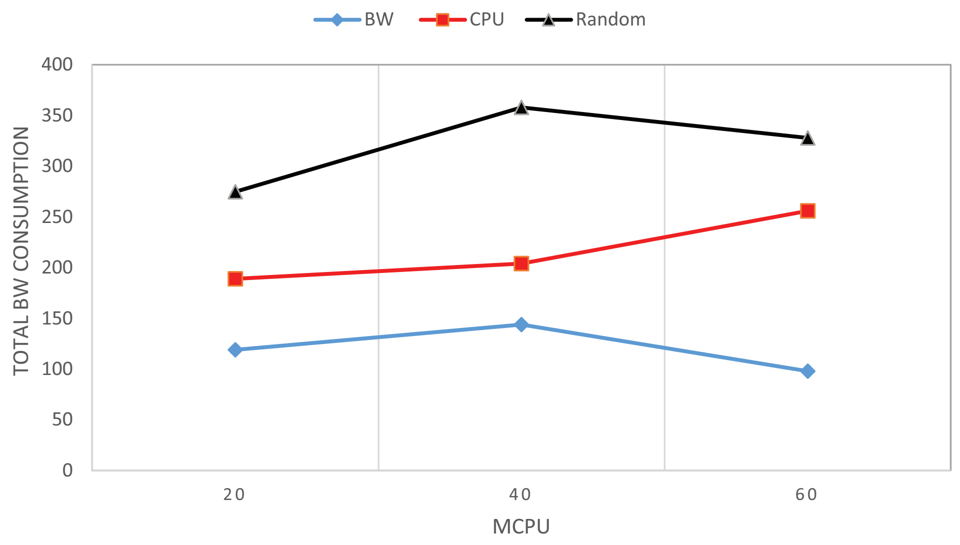


Figure (4.4) The Impact of Requested CPU Resources

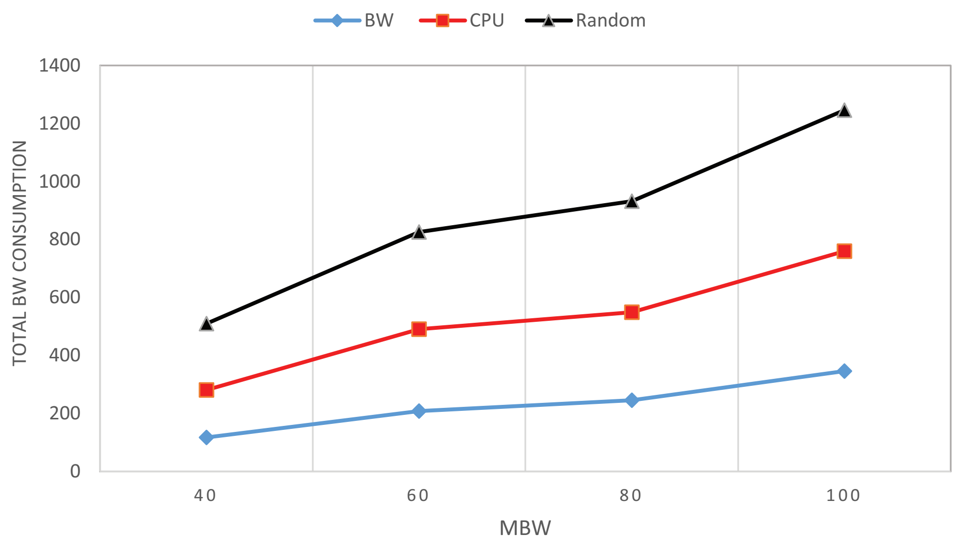


Figure (4.5) The Impact of Requested Outgoing BW

In this chapter, we introduced the problem of SFG design and mapping for NFV with priority dependence. To solve the problem efficiently, we presented the SFG_PD mapping with dependent directional acyclic graph (SFG_DAG) which design and embed the SFG in one coordinated step.

CHAPTER 5

DEPENDENCE-AWARE SERVICE FUNCTION CHAIN DESIGN AND MAPPING

In the NFV envirment, end-to-end services are delivered through service function chaining (SFC). The SFC defines the exact sequence of actions (i.e., VNF nodes) the data stream from this service request is subjected to. In this chapter, we study the problem of SFC design and embedding while considering the dependence constraints among the VNFs.

5.1 Substrate Network

We model the substrate network as an undirected graph $G^s = (V^s, E^s)$, where V^s and E^s are the sets of nodes and links in the substrate network, respectively. We use $c_u \in \mathbb{Z}^+$ as the available computing resource (e.g., CPU) of a substrate node $u \in V^s$. For each substrate link $uv \in E^s$, $(u, v \in V^s)$, $b_{uv} \in \mathbb{Z}^+$ represents the available bandwidth of the substrate link.

5.2 Network Service Request

An NFV service request is represented as a 3-tuple $NR =< B, N, D >$, where $B \in \mathbb{Z}^+$ is the requested bandwidth for the service chain; N is the set of VNF nodes; and D denotes a set of pre-defined dependence relationships between the VNF nodes in the request. Each VNF node $p \in N$ requires some computing resource $c_p \in \mathbb{Z}^+$, and a requested function F_p; Without loss of generality, we assume no two VNF nodes request the same function and each VNF node only requests a single function.

An NFV service request is represented as a 3-tuple $NR =< B, N, D >$, where $B \in \mathbb{Z}^+$ is the requested bandwidth for the service chain; N is the set of VNF nodes; and D denotes a set of pre-defined dependence relationships between the VNF nodes in the request. Each VNF node $p \in N$ requires some computing resource $c_p \in \mathbb{Z}^+$, and a requested function F_p;

Without loss of generality, we assume no two VNF nodes request the same function and each VNF node only requests a single function.

5.3 Dependence-Aware Service Function Chain (D_SFC_AM)

Definition for D_SFC problem: Given a service request $NR =< B, N, D >$, and the substrate network $G^s = (V^s, E^s)$, the Dependence-Aware Service Function Chain (D_SFC) problem can be defined as how to design the SFC and map this chain onto the substrate network while satisfying the following constraints.

(i) Dependence-Aware SFC: In the NR, a set of VNF nodes and dependence relationships among these nodes are given. The process of dependence-aware chain design has to construct an SFC by using all the VNF nodes in N while satisfying the node dependencies. Specifically, if VNF node p is dependent on VNF node q, then VNF p has to be placed behind VNF node q in the designed SFC.

(ii) Node Mapping: Each VNF node $p \in N$ requires $c_p > 0$ computing resource, which is mapped to exactly one substrate node $u \in V^s$ with enough computing resource $(i.e., c_p \leq c_u)$. No two VNF nodes from the same SFC can be mapped onto the same substrate node.

(iii) Link Mapping: Each virtual (or VNF) link between two consecutive VNF nodes in the SFC is mapped to a substrate link or physical path with enough bandwidth in the substrate network.

An optimization objective of D_SFC embedding can be minimizing the bandwidth consumption in the substrate network. The node mapping and link mapping processes in D_SFC are correspondent to the ones in the traditional virtual network embedding problem which is NP-Hard [63]. Accordingly, we propose an efficient heuristic algorithm, D_SFC design and resource allocation with Adaptive Mapping (D_SFC_AM) in the next section.

Heuristic Algorithms for D_SFC Embedding In this section, we first introduce a greedy heuristic, namely D_SFC design with Topological Sorting (D_SFC_TS), for embedding dependence-aware service requests. Then, we propose our D_SFC design and resource allocation with Adaptive Mapping (D_SFC_AM) to jointly optimize the SFC design and node mapping/linking, for embedding dependence-aware service requests.

D_SFC Design with Topological Sorting (D_SFC_TS) In D_SFC_TS, the two components, i.e., SFC design, SFC mapping, are conducted sequentially. The basic idea of D_SFC_TS is to firstly construct a chain based on the topological sorting method [105] and then map the constructed chain. In D_SFC_TS, the topological sorting [105] makes sure the dependence relationships among the VNF nodes are not violated in the constructed chain. Topological sorting output is a linear ordering of the vertices of a Directed Acyclic Graph (DAG). Once the chain is created and the chain topology is fixed, one can treat the constructed chain as a network and call the traditional virtual network embedding algorithms to conduct the node mapping and link mapping. Here, D_SFC_TS adopts the traditional schemes such as giving priorities to substrate nodes with larger available CPU and shorter routing path [64].

The pseudocode of D_SFC_TS is listed in Algorithm 5. In Line 1, the algorithm initializes several variables: $TopChain$ holds the constructed chain from the topological sorting; $LatestMapped$ is the latest VNF node that is mapped; and $LatestUsed$ indicates the latest substrate node that is used for mapping VNF nodes. Line 3 creates a list of candidate substrate nodes for each VNF node in the $TopChain$. Line 4 maps the first node in the chain onto the candidate node that has the highest CPU available. In case there is more than one choice, one node is chosen randomly. Starting Line 5, for each VNF p in the $TopChain$, the shortest path is calculated from $LatestUsed$ to all the candidates of p in S_p. The candidate node u that is the closest to $LatestUsed$ is chosen. In Line 9, the virtual link is mapped to the shortest path from $LatestUsed$ to u.

D_SFC Design and Resource Allocation with Adaptive Mapping If VNF nodes are mapped only by ordering them based on the dependence constraints (e.g., D_SFC_TS) and without considering the CPU requirement, a VNF node, say p, with more

Algorithm 5 D_SFC design with Topological Sorting (D_SFC_TS)

```
Require: Dependency Graph $D$
Ensure: Mapped SFC
 1: Initialize $TopChain$, $LatestMapped$, $LatestUsed$;
 2: Call topological sorting [105] with $D$ to create SFC and save it as $TopChain$;
 3: Create substrate candidate nodes list $S_p$ for each VNF node $p \in TopChain$;
 4: Map the first VNF node in $TopChain$ by choosing the candidate substrate node with
    the highest CPU (say $v$); set $LatestMapped$ as the first VNF node; set $LatestUsed$ as
    $v$;
 5: for each VNF $p \in TopChain$ do
 6:     Add a virtual link from $LatestMapped$ to $p$;
 7:     Find the shortest path from substrate node $LatestUsed$ to each candidate node in $S_p$;
 8:     Choose the substrate node (say $u$) with the lowest distance to $LatestUsed$;
 9:     Map the virtual link to the shortest path from $LatestUsed$ to $u$;
10:     Update $TopChain$, $LatestMapped$, and $LatestUsed$;
11: end for
```

dependencies and less CPU demand may have a higher priority to be mapped in earlier iterations. However, the computing resources occupied by node p may cause that VNF nodes with higher CPU demand cannot be accommodated in the latter iterations. Accordingly, we propose a novel D_SFC design and resource allocation with Adaptive Mapping (D_SFC_AM) algorithm to dynamically sort the VNF nodes for the chain design based on node dependencies, node CPU demand, and the updated resource information from the substrate network. The D_SFC_AM algorithm employs the following three techniques: *Dependence Sorting*, *Independence Grouping*, and *Adaptive Resource Allocation* to jointly optimize the processes of D_SFC design, VNF node mapping and link mapping.

Dependence Sorting To describe the dependence relationships among the VNF nodes in an NFV service request, we define two operators: γ and β, as shown in Eq. 1 and Eq. 2, respectively. Specifically, if a VNF q is dependent on p, we call $q \in \gamma_p$ as a descendant of node p in Eq. 1. On the other hand, if a VNF p is dependent on q, we call

$q \in \beta_p$ as an ancestor of p in Eq. 2.

$$\gamma_p = \{q | (p, q) \in D, p, q \in N\} \tag{1}$$

$$\beta_p = \{q | (q, p) \in D, p, q \in N\} \tag{2}$$

Lemma 1: For any VNF node p, if p is placed behind p's ancestors and ahead of p's descendants in the chain, then the chain does not violate the dependence constraints in D.

To ensure the dependence constraints in D while giving priorities to VNF nodes with higher resource (i.e., CPU) demand, we set the weight of a VNF p, denoted by $\zeta(p)$, as the sum of the CPU demands from p and p's descendants, as shown in the following equation:

$$\zeta(p) = c_p + \sum_{\forall q \in \gamma_p} \zeta(q), \tag{3}$$

where $c_p > 0$ is the CPU demand of node p, γ_p is the set of the descendants of node p.

Lemma 2: For any VNF node p, the weight of node p, $\zeta(p)$, is smaller than the weight of p's ancestors, and $\zeta(p)$ is greater than the weight of p's descendants.

Lemma 3: For any two VNF nodes p and q, if $\zeta(p) \geq \zeta(q)$, placing p ahead of q in the created chain, does not violate the dependence constraints in D.

Based on Lemma 1-3, we propose the Dependence Sorting algorithm as shown in Algorithm 6, which generates a $WeightList$ L, by sorting all the VNF nodes descendingly according to the weight from Eq. (3). The $WeightList$ from the Dependence Sorting algorithm can be denoted as $L = \{v_0, v_1, v_2, ..., v_i, v_{i+1}, ...\}$, where $\zeta(v_i) \geq \zeta(v_{i+1})$.

Independent Grouping Once the Dependence Sorting algorithm calculates the weights and sorts the VNF nodes in $L = \{v_0, v_1, v_2, ..., v_i, v_{i+1}, ...\}$, one can naturally con-

Algorithm 6 Dependence Sorting

```
Require: Dependency Graph $D$
Ensure: WeightList $L$
1: Select a VNF node $p$ which has no descendants;
2: for each $p$'s ancestor $q \in \beta_p$ do
3:     Update the weight of $q$ as Eq. (3);
4: end for
5: Remove node $p$ and the connections from $p$'s ancestor to $p$;
6: Go to step 1 till the weight for all VNFs are calculated;
7: Descendingly sort all VNF nodes based on the weight and save them in $L$;
```

struct an SFC chain with the same order as the VNF nodes in L and map the chain. However, such a constructed chain is fixed for the processes of node/link mapping, which cannot take advantage of the resource information in the substrate network for the SFC design and mapping. In fact, when checking dependence relationships between two consecutive VNF nodes v_i and v_{i+1}, we may find VNF nodes v_i and v_{i+1} have no dependence on each other, which indicates we can place v_i behind or ahead of v_{i+1} in the constructed chain. In other words, if VNF nodes v_i and v_{i+1} have no dependence on each other, we can either add $v_i \rightarrow v_{i+1}$ or $v_{i+1} \rightarrow v_i$ to the chain without violating the dependence constraints in D.

Accordingly, we propose the Independent Grouping algorithm in Algorithm 7 to divide the VNF nodes in L into groups. More specifically, Line 1 creates the first group G_0 and adds L's first VNF node into G_0. As shown in Line 2-9, a new group will be created if a VNF node has any dependencies with a node in the previous group. Thus for a VNF p, with dependence to any of the nodes in the previous group, in line 5-7, a new group is created and p is added to the new group and also removed from WeightList L. As a result, the independence group $G = \{G_0, G_1 ... G_i ...\}$ will be created. Note that VNF nodes in the group with a lower index have higher weight and all VNF nodes in the same group are independent.

Lemma 4: For any two VNF nodes p and q in the same independence group G_i, placing p behind or ahead of q in the chain, does not violate the dependence constraints in D.

Lemma 5: For any two independent group G_i and G_j, if $i < j$, placing all the nodes in G_i ahead of all the nodes in G_j in the created chain, does not violate the dependence constraints in D.

Algorithm 7 Independent Grouping

```
Require: WeightList $L = \{v_0, v_1, v_2, ..., v_i, v_{i+1}, ...\}$ from Dependence Sorting
Ensure: List of the groups $G = \{G_0, G_1...G_i...\}$
1: Set group index $i = 0$ , create group $G_0 = \{v_0\}$, and set $L = L - \{v_0\}$;
2: while $L$ is not empty do
3:     Set $p =$ the first node in $L$ ;
4:     if VNF node $p$ has a dependence on any node in $G_i$ then
5:         Set $i = i + 1$; create a new group $G_i$;
6:         Set $G_i = G_i \cup \{p\}$;
7:         Set $L = L - \{p\}$;
8:     end if
9: end while
```

Adaptive Resource Allocation As the objective of the D_SFC optimization is to minimize the required bandwidth resource in the substrate network, one intuitive strategy is to design and map the chain onto the substrate nodes that are close to each other. Based on Lemma 4 and 5, we propose the Dependence-Aware Service Function Chain design and resource allocation with Adaptive Mapping (D_SFC_AM) algorithm to adaptively create the chain and map the VNF nodes/links based on the available resource from the substrate network. The details of the resource allocation for D_SFC_AM is given in Algorithm 8. Line 1-2 will create G that contains all of the independent groups. Line 7-12 will append the VNF node that has a candidate substrate node closest to the latest mapped substrate node, until all the VNF nodes in an independent group G_i are accommodated. Then D_SFC_AM maps the VNF nodes in the next group G_{i+1}, until all the VNF nodes are added into the chain and mapped onto the substrate network.

5.4 An Example of D_SFC_AM

The example of D in Figure 5.1 illustrates that the service request requires seven VNF nodes and the arrows represent the order dependencies among the nodes. For this depen-

Algorithm 8 D_SFC Embedding with Adaptive Mapping D_SFC_AM

```
Require: Substrate Network $G_s$ and $NR =< B, N, D >$;
Ensure: Mapped SFC;
 1: Call Dependence Sorting in Algorithm 6 to generate $L$;
 2: Call Independent Grouping in Algorithm 7 to generate $G = \{G_0, G_1 ... G_i\}$;
 3: Set the mapped $SFC = \emptyset$; the latest added VNF node $LatestVNF = \emptyset$; the latest
    used substrate node $LatestSub = \emptyset$; and create the list of the candidate substrate node
    list for a VNF node $p$ denoted by $CL(p)$;
 4: Map the first VNF node in $G_0$ onto the candidate substrate node with the highest CPU;
    update $LatestVNF$, $LatestSub$, and $G_0$;
 5: Add VNF node $LatestVNF$ to the mapped SFC;
 6: Set i=0;
 7: for each $G_i \in G$ do
 8:    while $G_i \neq \emptyset$ do
 9:       Sort all the candidate substrate nodes of VNF node in $G_i$ based on the distance to
          substrate node $LatestSub$;
10:       Set $LatestVNF$ as the VNF node that has the candidate node $S$ and node $S$ has
          the shortest physical path to substrate node $LatestSub$;
11:       Add VNF node $LatestVNF$ to the mapped SFC; Map $LatestVNF$ to $S$ and the
          related VNF link;
12:       Update $G_i$, $LatestSub$ and the candidate substrate list;
13:    end while
14:    Set i=i+1;
15: end for
```

dency relationships, $VNF2$ is dependent on three other VNF nodes (i.e. $VNF1$, $VNF4$ and $VNF6$), which implies $VNF2$ has to be located behind $VNF1$, $VNF4$ and $VNF6$ in the SFC design. Without any dependencies among $VNF4$ and $VNF6$, they can be located anywhere in the SFC before $VNF2$.

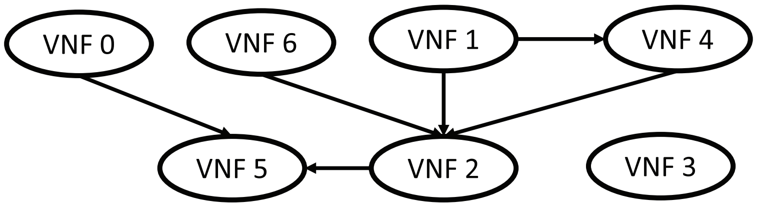


Figure (5.1) An example of dependence relationships (i.e., D)

For the given D in Figure 5.1 the CPU demands of VNF nodes are shown in Table 5.1. The weights are sorted in Table 5.2 and L={VNF1, VNF4, VNF6, VNF2, VNF0, VNF3, VNF5}.

Table (5.1) CPU demand of each VNF

VNF Node	0	1	2	3	4	5	6
CPU	45	40	50	35	25	30	20

Table (5.2) WeightList L

VNF Node	1	4	6	2	0	3	5
Weight	225	105	100	80	75	35	30

The results of independent grouping for VNF nodes in Figure 5.1 is listed in Table 5.3. Specifically, based on the order of the VNF nodes in L, $VNF1$ is assigned to group G_0. The next node in L is $VNF4$ which cannot be added to G_0 since $VNF4$ is dependent on $VNF1$. Hence, a new group, G_1, is created and $VNF4$ is added into G_1. The following node $VNF6$ can be in the same group as $VNF4$ as there is no dependency between them. Then, for $VNF2$, it depends on both $VNF4$ and $VNF6$, therefore G_2 is created and $VNF2$

is a member. Similarly, $VNF0$ and $VNF3$ are added to G_2 because there is no dependency violation. Finally, since node $VNF5$ has a dependence on node $VNF0$ of G_2, a new group G_3 is created and $VNF5$ is added into G_3.

For the adaptive mapping, VNF1 will be firstly added to the chain and mapped onto a substrate node S. In G_1, VNF6 can be behind or ahead of VNF4 in the constructed chain because there is no dependence relationship between VNF4 and VNF6. Hence, the proposed D_SFC_AM will sort the distance from VNF6's and VNF4's candidate substrate nodes to node S, and select the substrate node that is closest to node S to map either VNF6 or VNF4. This way, instead of directly mapping VNF4 ahead of VNF6 due to the weight order, we check to see which VNF node can be mapped closer to the latest mapped node. In other words, for chain design and mapping, D_SFC_AM takes the available resource information from the substrate network into account to adaptively construct the chain and find the closet candidate substrate nodes for the VNF node/link mapping.

Table (5.3) Independent Grouping

Group Index i	VNF
0	{1}
1	{4,6}
2	{2,0,3}
3	{5}

5.5 Performance Evaluation

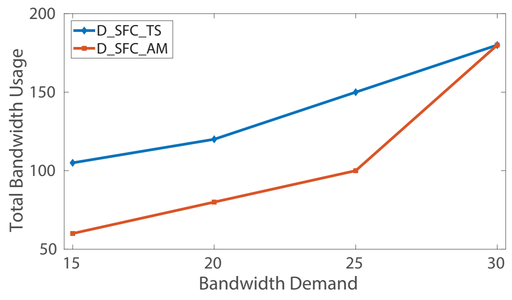
In this section, we analyze the performance of D_SFC design with Topological Sorting (D_SFC_TS) and D_SFC design with Adaptive Mapping (D_SFC_AM). We compare the algorithms by using the NSF network topology with 14 nodes and 21 edges as the substrate network. For each substrate node, the available CPU is randomly generated in the range of $[5, 40]$, and the available bandwidth for each link varies from 5 to 50. For an NFV request, NR, we vary the requested bandwidth (B) in the range of $[15, 35]$ and the CPU demand for each VNF node is randomly generated in the range of $[10, 25]$. The number of VNF nodes in the NR (N) is from 3 to 7. The dependency relationship for VNF nodes is also randomly generated. We conduct extensive experiments to obtain the average results as shown in Figure 5.2.

In Figure 5.2(a), the y-axis denotes the total bandwidth usage and the x-axis represents the requested bandwidth. From this figure, one can see that when the requested bandwidth is increased, the bandwidth usage by D_SFC_TS and D_SFC_AM increases. D_SFC_AM significantly outperforms D_SFC_TS until B is bigger than 25. This is because the proposed D_SFC_AM can jointly optimize the chain design and VNF node/link mapping with the techniques such as dependence sorting, independent grouping and adaptive resource allocation while D_SFC_TS maps fixed constructed chain without effectively considering the current status of the substrate network. The results can be further verified in Figure 5.2(b).

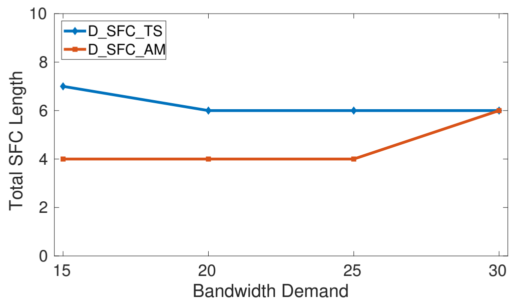
It should be noted that when the bandwidth request B is higher than 25, due to the lack of available links with enough bandwidth, there is not many alternative physical paths for D_SFC embedding and hence both algorithms converged to the similar performance. In Figure 5.2(c) and 5.2(d) the x-axis, represents the CPU demand for the VNF nodes in the NR request. Again, D_SFC_AM yields better results than D_SFC_TS. Figure 5.2(e) and 5.2(f) demonstrate the impact of the number of VNF nodes in the request. These figures show that by increasing the number of VNFs in the request NR , D_SFC_AM uses less bandwidth and less number of hops for the created chain. When the number of VNF nodes exceeds 5, the performance gap between the two algorithms increases, which shows that D_SFC_AM can effectively take advantages of dependence sorting, independent grouping and the available resource information in the substrate network during the process of D_SFC design and mapping to minimize the required bandwidth in the substrate network.

5.6 Shortview Mapping

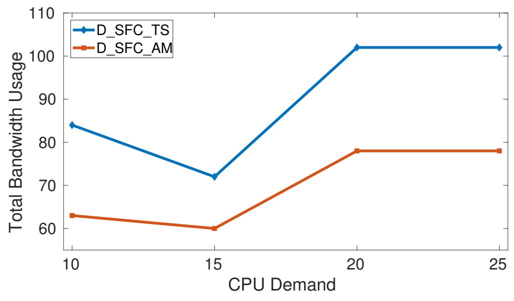
Figure 5.3 shows the results of the D_SFC_AM algorithm for an NFV service request NSR that has three VNF nodes v_1, v_2 and v_3 with the CPU requirement of 10, 20 and 15,



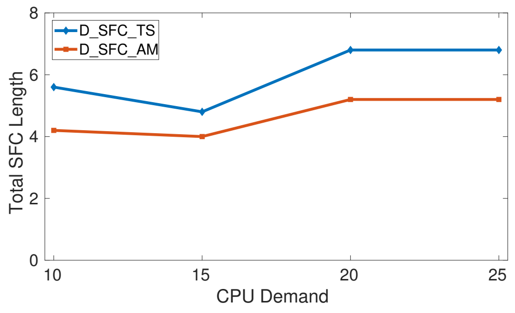
(a) Bandwidth Usage vs B



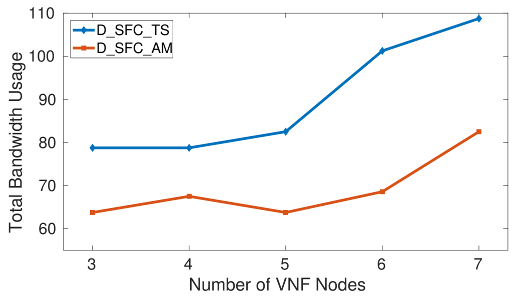
(b) The Length of SFC vs B



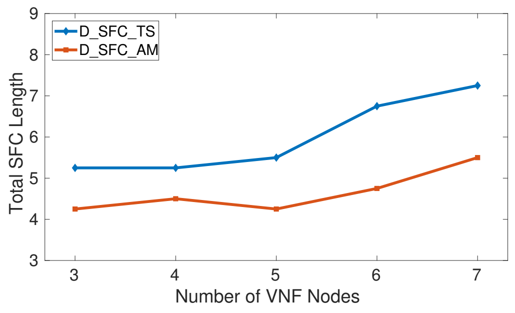
(c) Bandwidth Usage vs CPU



(d) The Length of SFC vs CPU



(e) Bandwidth Usage vs N



(f) The Length of SFC vs N

Figure (5.2) The Performance Comparison between D_SFC_TS and D_SFC_AM.

respectively. The dependence constraints for this NSR include $v_1 \mapsto v_2$ and $v_2 \mapsto v_3$. The substrate candidates for v_1, v_2 and v_3 are $\{s_1\}$, $\{s_2, s_3\}$ and $\{s_4\}$, respectively. The number on each substrate link represents the physical distance (or hop numbers). After the Dependence Sorting and Independent Grouping processes for NSR, the groups are: $g_0 = \{v_1\}$, $g_1 = \{v_2\}$ and $g_2 = \{v_3\}$. According to the D_SFC_AM algorithm, first, v_1 from g_0 is mapped onto s_1. In g_1, as s_2 is the substrate candidate closest to s_1 which is the tail of the created Service Function Path (SFP), v_2 is mapped onto s_2. Similarly, v_3 in g_2 is mapped to s_4. As a result, the created SFC is $v_1 \rightarrow v_2 \rightarrow v_3$. The mapped SFP for this NSR is $s_1 \dashrightarrow s_2 \dashrightarrow s_4$ which has the length as 7.

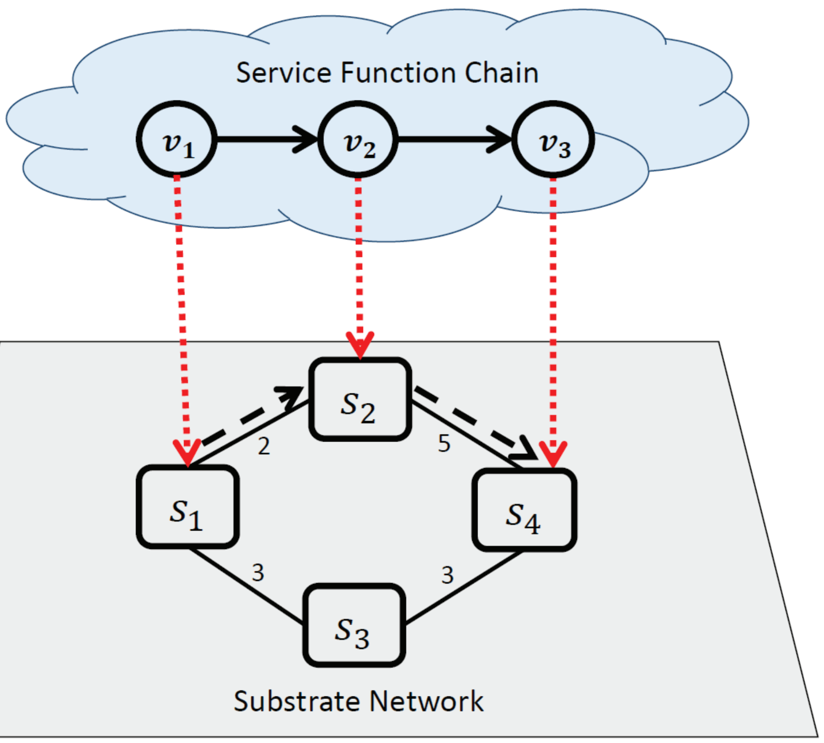


Figure (5.3) The constructed SFC by D_SFC_AM algorithm

However, if v_2 is mapped to s_3 (as shown in Figure 5.4), the total length of the created SFP would be 6 and it requires less bandwidth consumption. This is because D_SFC_AM algorithm uses the strategy that only takes into account the shortest connection between two consecutive VNF nodes in the created SFP. We call this as the *shortview Mapping* by

D_SFC_AM algorithm. This *shortview Mapping* may actually exist for any three consecutive VNF nodes of the constructed SFP. Accordingly, in the next section, we propose the *Tetragon Remapping* technique to further optimize the bandwidth allocation for the constructed chain.

5.6.1 Dependence-aware SFC Embedding with Group Mapping

To avoid the *shortview Mapping* by the D_SFC_AM algorithm, we propose the Tetragon Remapping technique which tries to optimize the VNF node/link mapping for any three consecutive VNF nodes in a chain. In specific, for any three consecutive VNF nodes $v_{i-1} \rightarrow v_i \rightarrow v_{i+1}$ that are mapped to $s_{i-1} \dashrightarrow s_i \dashrightarrow s_{i+1}$, Tetragon Remapping tries to find if there exists a substrate node t_i for VNF node v_i and the following formula is true.

$$\begin{aligned} &\mathit{distance}\,(s_{i-1}, t_i) + \mathit{distance}\,(t_i, s_{i+1}) < \\ &\mathit{distance}\,(s_{i-1}, s_i) + \mathit{distance}\,(s_i, s_{i+1}) \end{aligned}$$

If such substrate nodes t_i are successfully found, we will then remap v_i to the substrate node t_i that yields minimum distance (s_{i-1}, t_i)+distance (t_i, s_{i+1}). Based on this *Tetragon Remapping* technique, we propose a new D_SFCE algorithm, namely, Dependence-aware SFC Embedding with Group Mapping (D_SFC_GM), which employs the aforementioned Dependence Sorting, Independent Grouping, Adaptive Mapping and Tetragon Remapping as shown in Algorithm 9.

When applying the Tetragon Remapping process in Figure 5.3, the Tetragon Remapping process starts from the node v_2 which is the node right ahead of the tail node of the constructed SFC from the D_SFC_AM algorithm. Here, for v_2 there is another substrate candidate s_3 where the sum of the distances between s_1 to s_3 and s_3 to s_4 is less than the existing mapped result. Hence, v_2 is remapped to s_3. The constructed chain after applying the Tetragon Remapping technique for this example is shown in Figure 5.4.

In D_SFC_AM, Dependence sorting and Independent Grouping have the average time complexity of $\mathcal{O}(|N|^2)$, where $|N|$ is the number of nodes in the graph. Next, for each node

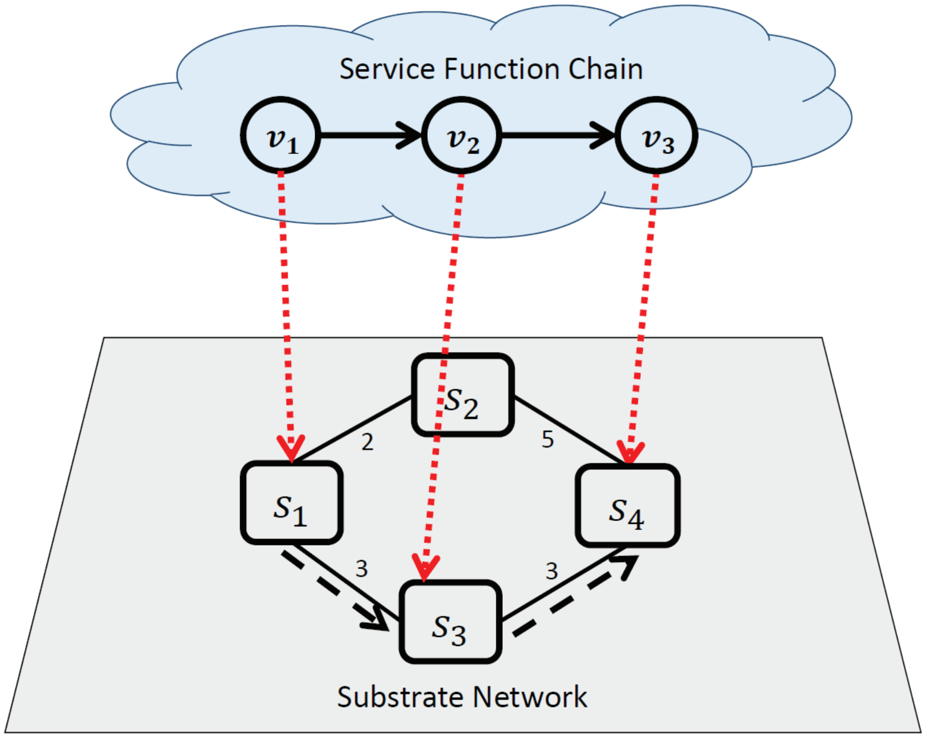


Figure (5.4) The constructed SFC by D_SFC_GM algorithm using Tetragon Remapping

Algorithm 9 D_SFC Embedding with Group Mapping (D_SFC_GM)

```
Require: $G_S$ and $NR =< B, N, D >$;
Ensure: GM_SFC: the constructed SFC, GM_SFP: the constructed SFP;
 1: Call $D\_SFC\_AM(G_S, NR)$ and the outputs are saved as: GM_SFC = $\{v_1, v_2, ..., v_n\}$ and
    GM_SFP = $\{s_1, s_2, ..., s_n\}$;
 2: Set $i = n - 1$;
 3: while $i > 1$ do
 4:    for any substrate candidate node $t_i$ of $v_i$ do
 5:       if $distance(s_{i-1}, t_i) + distance(t_i, s_{i+1})$
          $< distance(s_{i-1}, s_i) + distance(s_i, s_{i+1})$ then
 6:          Remap $v_i$ to substrate node $t_i$;
 7:          Update GM_SFC and GM_SFP
 8:       end if
 9:    end for
10:    Set $i = i - 1$;
11: end while
12: return GM_SFC, GM_SFP;
```

in a group, D_SFC_AM uses the Dijkstra's shortest path algorithm and First-Fit spectrum assignment that has the computing complexity of $\mathcal{O}(|N||L| + |N|^2 \log |N| + |S||L|)$ [106], where $|S|$ is the number of spectrum segments in a substrate fiber link and $|L|$ is the number of links in the graph. Therefore, D_SFC_AM has the average computing time of $\mathcal{O}(|N|^2 * (|N||L| + |N|^2 \log |N| + |S||L|))$. In D_SFC_GM, Tetragon Remapping similarly has the computing complexity of $\mathcal{O}(|N|^2 * (|N||L| + |N|^2 \log |N| + |S||L|))$. Hence, the D_SFC_GM can take the average computing time of $\mathcal{O}(|N|^2 * (|N||L| + |N|^2 \log |N| + |S||L|))$.

It is worth noting that the NFV service requests are given from the beginning, which represents the offline scenario. However, as the time complexity of the algorithms are polynomial, they are suitable for the online scenario.

5.6.2 Performance Evaluation of D_SFC_GM

In this section, we compare the performance of D_SFC_AM and D_SFC_GM algorithms with Topological Sorting (D_SFC_TS) algorithm [91]. In D_SFC_TS, the two processes of SFC design and mapping are conducted sequentially. The basic idea of D_SFC_TS is to first construct a chain by exploiting the topological sorting method [105]. Then, D_SFC_TS maps the constructed chain onto the substrate network. By using topological sorting method, D_SFC_TS guarantees the dependence relationships among the VNF nodes are not violated in the constructed chain. The output of topological sorting is a linearly ordered vertices of a Directed Acyclic Graph (DAG). Once the chain constructed by the topological sorting is fixed, one can treat the constructed chain as a network and call the traditional virtual network embedding algorithms to conduct the node mapping and link mapping. Here, we use the traditional scheme that gives priorities to substrate nodes with larger available CPU and shorter routing paths [64, 107, 108].

We use a 28-node US Backbone network [109] as the substrate IP or optical network. Unless otherwise specified, the available computing resources of substrate nodes is in the range of $[5, 35]$; the offered functionality for each substrate node is randomly generated; the available bandwidth (or number of subcarriers) for each substrate link varies from 5 to 45;

and there is no wavelength conversion in the substrate optical networks. Similarly, unless otherwise specified, the number of VNF nodes (N) in an NFV service request (NSR) is set in the range of $[3, 8]$; the dependent constraints among the VNF nodes are randomly generated; each VNF node requests a computing demand in the range of $[5, 25]$, and each NSR requests the bandwidth (or number of subcarriers) within the range of $[5, 25]$. The NSR is randomly generated and we collect the average bandwidth (spectrum) consumption for embedding a large number of NSRs over 100 different substrate networks, which are denoted as "Average BW (Spectrum) Consumption" in the following figures. We compare and analyze how the number of VNF nodes, bandwidth and computing demand in an NSR impact the performance from the proposed D_SFC_GM, D_SFC_AM and D_SFC_TS algorithms.

The impact of optical constraints

Figure 5.5 shows the impact of optical network constraints (spectrum continuity and consecutiveness). Here, we compare the performance of the three algorithms for two cases where wavelength conversion is available in the substrate network (i.e., D_SFC_GM_Conv) and wavelength conversion is not provided (i.e., D_SFC_GM_NO_Conv). For this experiment, substrate resources are limited as mentioned earlier. In Figure 5.5, the X-axis is the number of VNF nodes in the request and the Y-axis is the average spectrum consumption. As one can see, for all three algorithms, the average spectrum consumption is the highest for the case that no wavelength conversion is available in the substrate network, whereas both spectrum continuity and consecutiveness must be satisfied. When the wavelength conversion is available only consecutiveness constraint should be satisfied. Hence, shorter paths can be found between the nodes on the SFP which results in less spectrum consumption.

The impact of NFV service request size

To study the impact of NSR size, we set each substrate node with unlimited computing resources and set each substrate link with unlimited bandwidth (or number of subcarriers). The NSR is randomly generated with 2 dependencies and the number of VNF

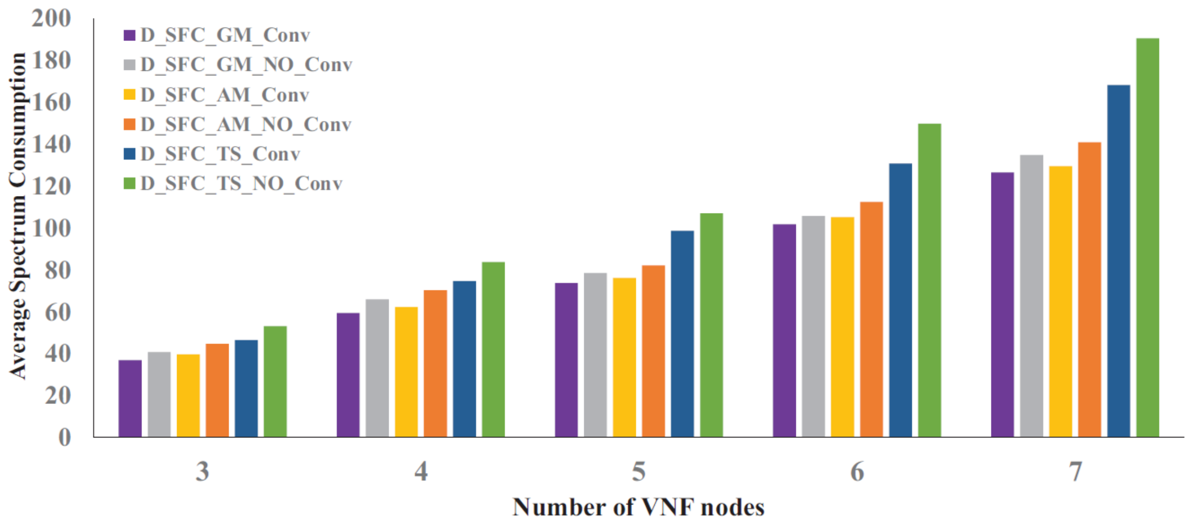
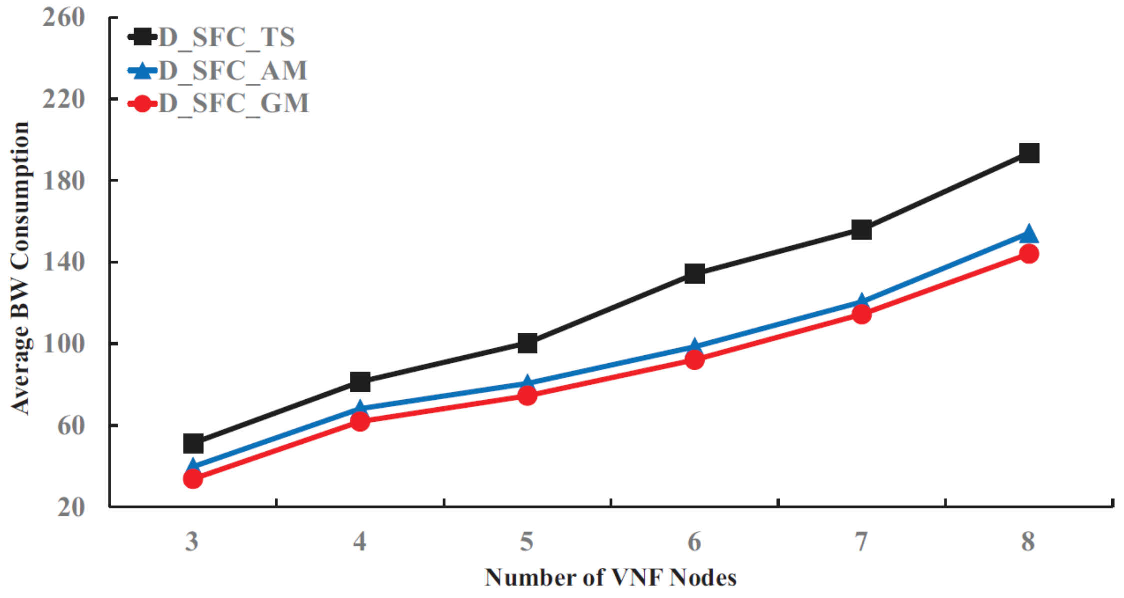


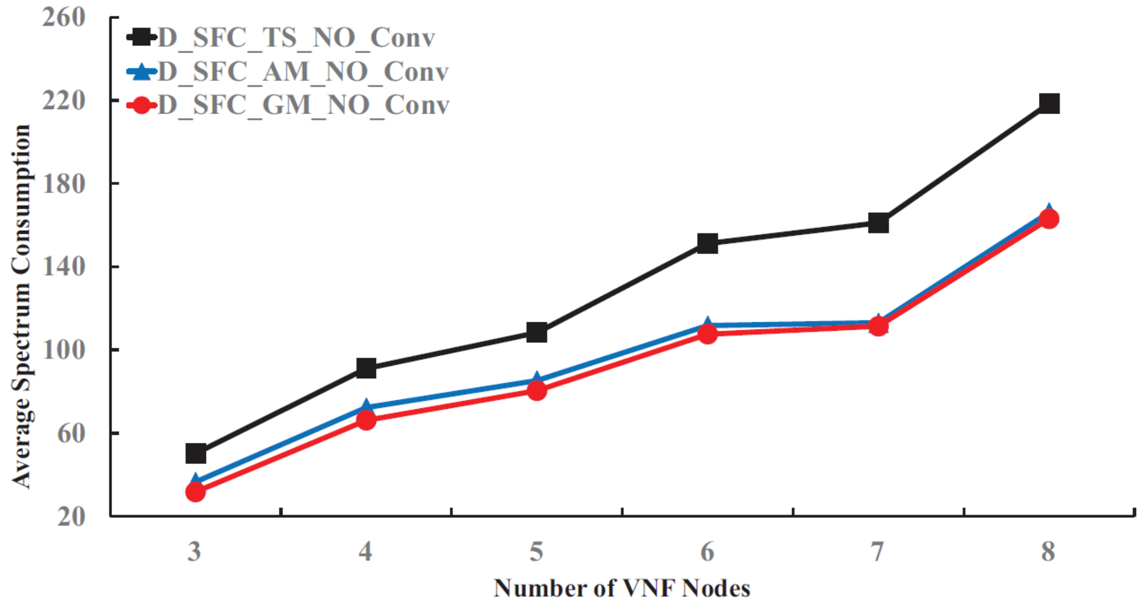
Figure (5.5) Impact of optical constraints

nodes in the NSR varies from 3 to 8. We set the substrate network as IP network to obtain the results in Figure 5.6(a) and set the substrate network as optical network in Figure 5.6(b). Figure 5.6(a) and 5.6(b) show the average bandwidth (spectrum) consumption when increasing the number of VNF nodes in the NSR. As one can see, for all three algorithms, the average bandwidth (spectrum) consumption increases with the number of VNF nodes in the NSR. Both the proposed D_SFC_GM (D_SFC_GM_NO_Conv for optical) and D_SFC_AM (D_SFC_AM_NO_Conv for optical) algorithms significantly outperform the D_SFC_TS (D_SFC_TS_NO_Conv for optical) in IP (optical) substrate networks. This is because the proposed techniques including dependence sorting, independent grouping and adaptive mapping enable D_SFC_GM and D_SFC_AM to design better SFCs. Particularly, here, the substrate networks have abundant CPU and bandwidth (or number of subcarriers) resources, which makes the processes of VNF node and link mapping in all three algorithms to follow similar behavior. As a result, a better SFC design will require less bandwidth (or number of subcarriers) along the Service Function Path (SFP) in the substrate network. Furthermore, the D_SFC_GM algorithm has slightly better results than the D_SFC_AM algorithm due to the Tetragon Remapping technique in D_SFC_GM. The process of Tetragon

Remapping in D_SFC_GM can further optimize the VNF link mapping by identifying closer substrate nodes that the VNF nodes can be relocated to.



(a) Substrate IP networks



(b) Substrate optical networks

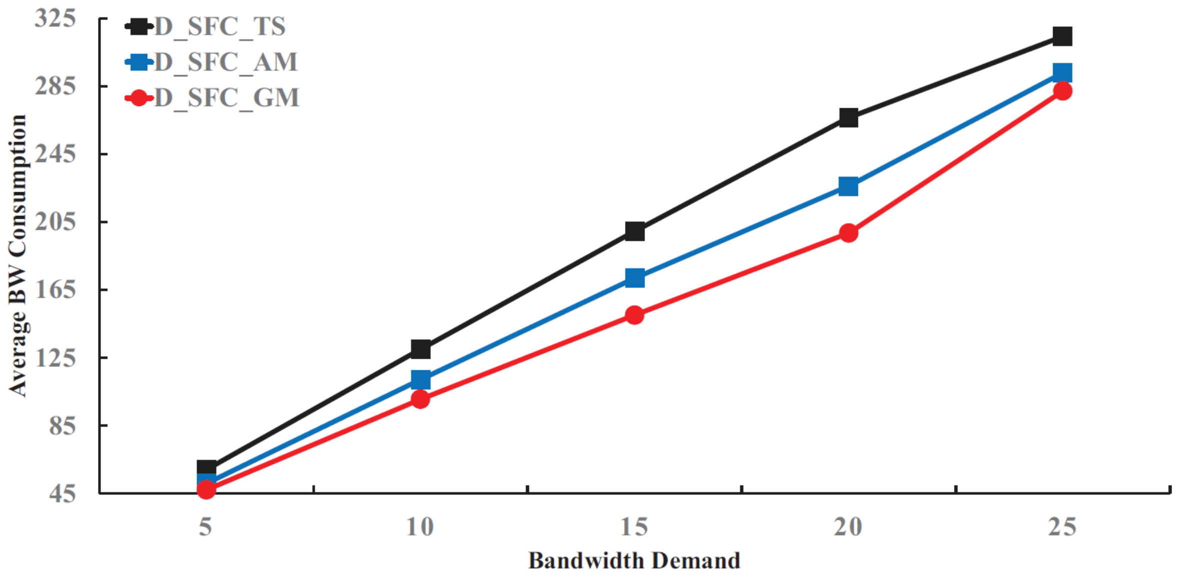
Figure (5.6) Bandwidth consumption vs. Number of VNF Nodes

The impact of NFV service request bandwidth

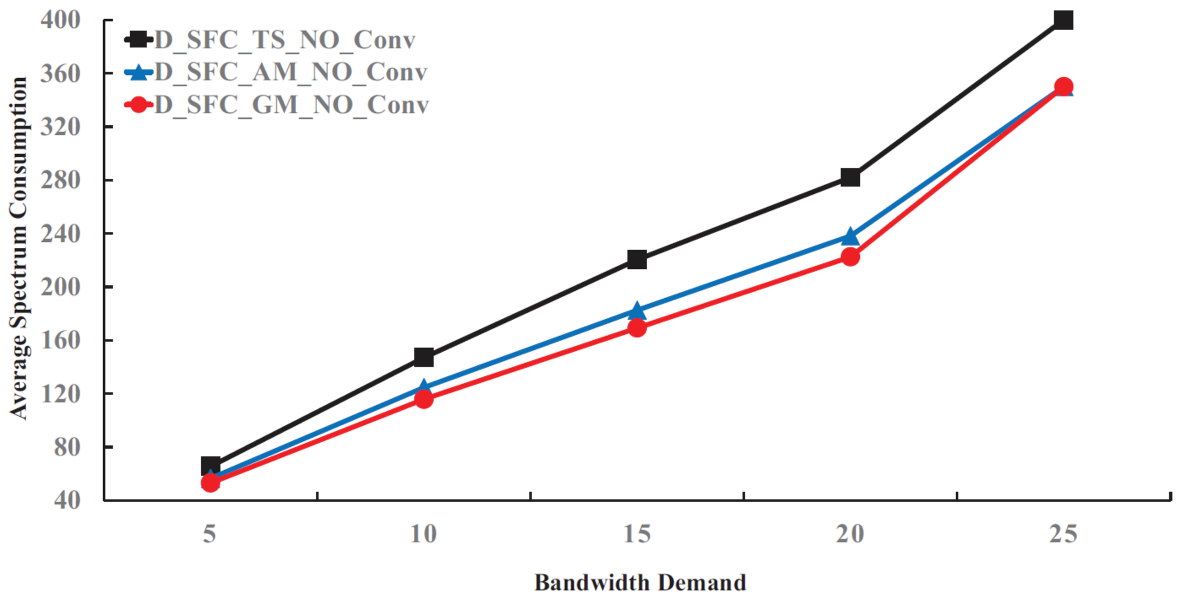
When investigating the impact of NSR bandwidth, we set substrate resource limited as mentioned earlier. Each NSR is randomly generated with 6 VNF nodes and 5 dependencies and the requested bandwidth for the NSR varies from 5 to 25. In Figure 5.7(a), the y-axis denotes the average bandwidth consumption and the x-axis represents the requested bandwidth for the NFV service request. One can see that when the requested bandwidth increases, the average bandwidth consumption of three algorithms increases as well. In particular, D_SFC_AM outperforms D_SFC_TS because the D_SFC_AM algorithm can jointly optimize the chain design and VNF node/link mapping processes by applying dependence sorting, independent grouping and adaptive mapping techniques. As for D_SFC_GM algorithm, it outperforms D_SFC_AM as D_SFC_GM employs the Tetragon Remapping technique to avoid the shortview mapping in D_SFC_AM. It is worthy noting that when the NSR bandwidth request is higher than 25, due to the lack of available links with enough bandwidth, there is not many alternative physical paths for D_SFC embedding and all three algorithms converging to the similar performance. The results from substrate optical network in Figure 5.7(b) further verify these observations.

The impact of CPU in the substrate network

Figure 5.8 shows the impact of the available CPU resources in the substrate nodes. To explore this impact, we set the range for available CPU resources of the substrate nodes as shown in Figure 5.8(a) and 5.8(b). We set the available bandwidth (or number of subcarriers) of the substrate links as unlimited. Each NSR is randomly generated with 7 VNF nodes and 4 dependencies. The CPU requirement for all VNF nodes is 12 and the requested bandwidth (or number of subcarriers) for the NSR is set to 10. One can see that in both IP and optical networks, the average bandwidth (spectrum) consumption decreases when the range for available CPU resources increases. This is because more substrate nodes can satisfy the CPU requirement of the VNF nodes when the range for available CPU resources increases, leading

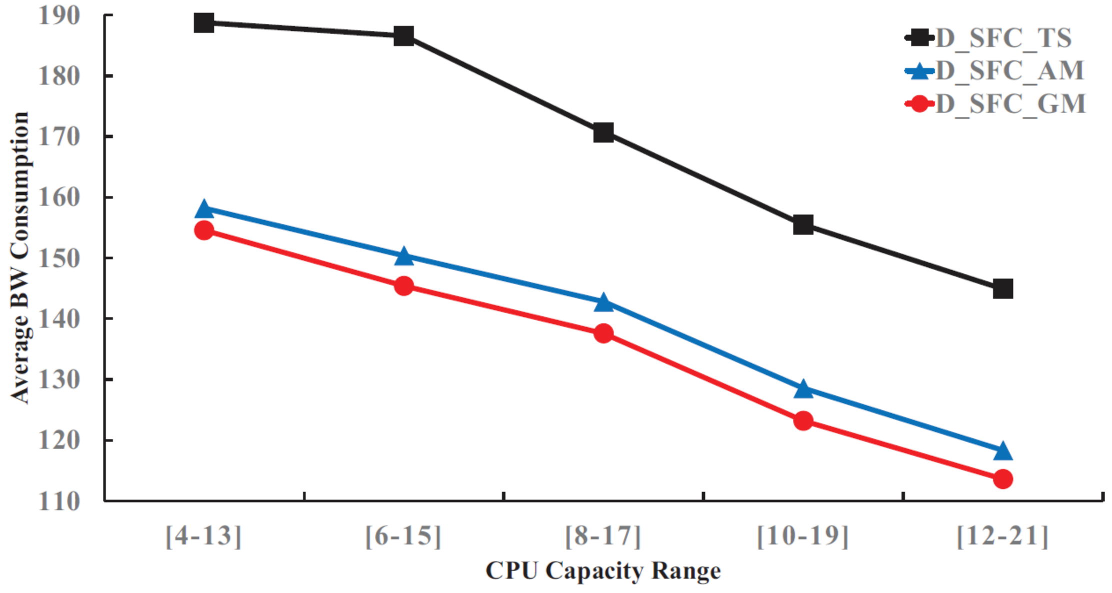


(a) Substrate IP networks

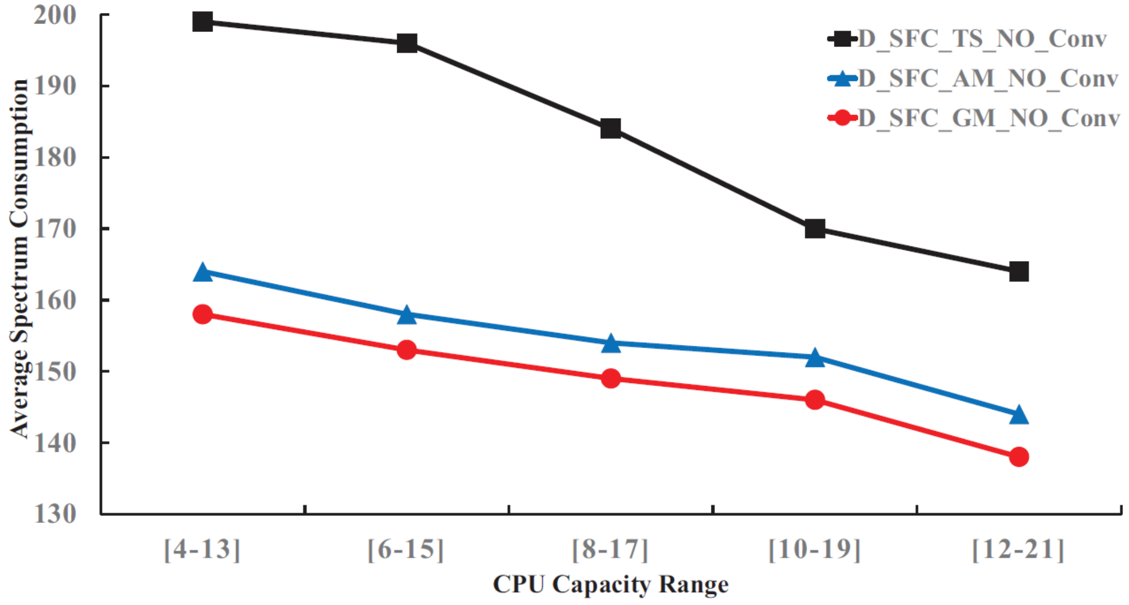


(b) Substrate optical networks

Figure (5.7) Bandwidth consumption vs. NSR bandwidth



(a) Substrate IP networks



(b) Substrate optical networks

Figure (5.8) Bandwidth consumption vs. Substrate CPU capacity to possible shorter service function paths. Once again, the proposed D_SFC_GM yields better results than both D_SFC_AM and D_SFC_TS. In particular, D_SFC_GM outperforms D_SFC_AM and D_SFC_TS as much as 8% and 25%, respectively.

In this chapter, we introduced the problem of dependence-aware SFC design and mapping. To solve the problem efficiently, we presented the D_SFC_AM which uses techniques such as dependence sorting, independent grouping and adaptive resource allocation to construct and map the chain onto the substrate network. To further optimize the bandwidth consumption in the substrate network, we presented D_SFC_GM which uses Tetragon Remapping technique to optimize the mapping for the constructed chain.

CHAPTER 6

BRANCHING-AWARE SERVICE FUNCTION GRAPH EMBEDDING IN NETWORK FUNCTION VIRTUALIZATION

Traffic Flow Branching in NFV Generally, after being processed by one network function, the given (modified) traffic flow continues to the next network function, which leads to a straightforward linear chain. However, in reality, use cases exist that necessitate the branching of the traffic flow, which can be classified into three major categories below.

First, certain network functions may classify traffic flows or treat flows of distinct features in different manners. For instance, a flow classifier may separate the video and non-video traffic, and direct them to respective proceeding network functions. An example of such scenario is given in Figure 6.1. In Figure 6.1, the DPI has advanced classification capabilities to detect whether the traffic is video or not. The video traffic is further forwarded to a Video Optimizer (VO) and later to the LB2 while the non-video traffic is directly sent to LB2.

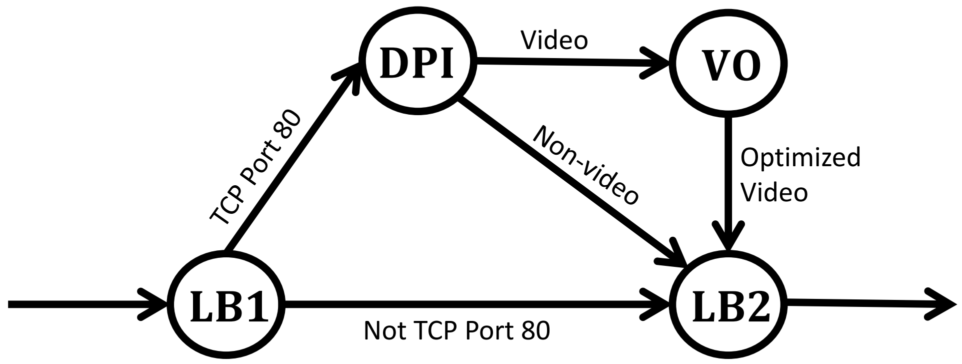


Figure (6.1) An example of branching in a service request

Second, flows can fork in the presence of a load balancer type of network functions. As an example in Figure 6.1, when the incoming traffic from the gateway reaches the LB1, the load balancer investigates whether the traffic is of the type of TCP port 80 or not (i.e., web traffic). Then, the web traffic is directed to another path which contains DPI function. While the non-web traffic is transferred to the LB2.

Third, some network functions with no dependency on each other can operate in parallel rather than sequential, as illustrated in [110]. For example, in Figure 6.2, the functionality of Monitor (MN) NF is to maintain the packet statistics without changing the packets. Therefore, the traffic from VPN can go through the Monitor and the Firewall simultaneously to reduce the end-to-end latency.

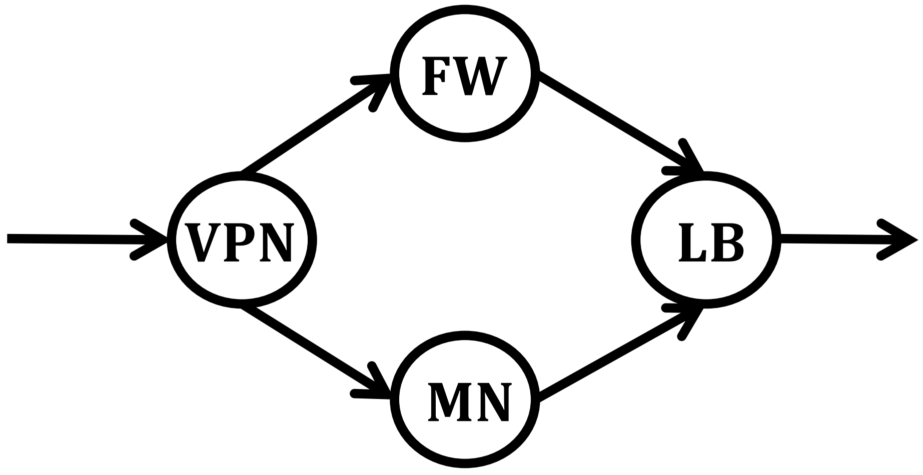


Figure (6.2) An example of parallelism in a service request

With the aforementioned branching, the traffic flow may go through a mesh graph (i.e., a Directed Acyclic Graph (DAG)) as in Figure 6.1 rather than a linear topology. Therefore, one can not employ the traditional SFC chaining and embedding approaches. In the literature, to the best of our knowledge, very limited works [111, 110, 112] explicitly explore the branching in NFV and its impact to the SFC design and embedding. These work discussed and studied the parallelism in SFC, which corresponds to branching in the third case above. However, their focus is on how to enable the parallelism in the SFC. It is worth noting that, the mesh graph, which is the result of branching, is not the same as VNF Forwarding Graph (VNFFG). The latter models how different VNFs from multiple **independent** chains are interconnected [113].

In this chapter, we define the resulting problem of Branching-Aware Service Function Graph (B_SFG) embedding and study how to efficiently accommodate user's B_SFG requests in the substrate network while considering the constraints of function dependence, branching requirements, computing resources of virtual nodes and bandwidth demand of the B_SFG.

6.1 Substrate Network

The substrate network is modeled as an undirected graph $G_p = (V_p, E_p)$, where each node $v_i \in V_p, (i = 1, ..., |V_p|)$ is a dedicated COTS server. The amount of available computing resources (e.g., CPU) in the server v_i is denoted as $cr(v_i) \in \mathbb{Z}^+$. $\Psi \in F$ is the available network function that is supported by substrate node s. For each physical link $uv \in E_p$, $b_{uv} \in \mathbb{Z}^+$ represents the available bandwidth of the physical link.

6.2 Network Service Request

A network service request is defined as a 5-tuple $NSR =< s, d, \beta, N, R >$, where s is the source NF node, d is the destination NF node, β is the initial outgoing traffic from s, N is the set of network functions for the request NSR, and R is the set of policies that specify the requirements that are applied to individual NFs or between NFs. Each NF $v \in NSR$ in the request is associated with multiple attributes: a unique network function f_v; required amount of computing resources $c_v \in \mathbb{Z}^+$; and a traffic modification ratio of t_v, which is the ratio between the outgoing traffic (after applying the network function) to the incoming traffic. We assume that no two VNF nodes in a single request demand the same network function.

6.2.1 Service Function Graph Policy

To accommodate the branching of traffic flow as well as addressing other requirements that need to be applied to NFs, we define a group of policies that can be utilized by the end user to specify the desired service features.

Dependence Policy Dependence policy expresses the need to place two NFs in the required execution order. For instance, if there is a dependence from $v_2 \in N$ to another node $v_1 \in N$, then v_2 must be placed after the node v_1 in the constructed graph. This in turn ensures that traffic flow goes through v_1 before node v_2. The dependence policy is defined as $(NF1 \succ NF2)$ to express the desired execution order of two NFs. For instance, an IPSec decryptor usually has to be placed before a Network Address Translation (NAT) gateway [99], which can be described with the dependence policy (IPSec decryptor $\succ$ NAT). For a service request NSR, the set $\mathscr{D} \in R$ in Eq. (6.1) is the collection of all of the dependence polices for the network functions in the request. It should be noted that, for the dependence policies we only need to focus on the direct dependency and exclude the transitive policies. For example, in Figure 6.1, two of the dependencies are: $(LB1 \succ DPI)$ and $(DPI \succ VO)$. Therefore, we do not need to add $(LB1 \succ VO)$ to the set $\mathscr{D}$.

$$\mathscr{D} = \{(u, v) | u \succ v; u, v \in N\} \tag{6.1}$$

To further describe the dependence relationships among the VNF nodes in an NFV service request, we define two operators: γ and β, as shown in Eq. (6.2) and (6.3), respectively. Specifically, if a VNF q is dependent on p, we call $q \in \gamma_p$ as a descendant of node p in Eq. (6.2). On the other hand, if a VNF p is dependent on q, we call $q \in \beta_p$ as an ancestor of p in Eq. (6.3).

$$\gamma_p = \{q | (p, q) \in \mathscr{D}, p, q \in N\} \tag{6.2}$$

$$\beta_p = \{q | (q, p) \in \mathscr{D}, p, q \in N\} \tag{6.3}$$

Branching Policy To accommodate the branching of the traffic, we next introduce the branching policy. We refer to the NF where the incoming traffic flow can fork as the branching point. We associate each branch leaving the branching point with a percentage of maximum outgoing data rate. The set of all branching points is represented by $\mathscr{B} \in R$ in the Eq. (6.4). Each branch point is further denoted by a set, b_i in Eq. (6.5), where $(NF \Lleftarrow)$ is used to specify the flow branch after being processed by NF. For the elements of the set b_i, i is the node that traffic branches out after that, α specifies the percentage of the incoming traffic out of i and into v. For example, in Figure 6.1, DPI can divide incoming traffic into video type and non-video type. Assume that DPI sends up to 20% of the incoming packets towards a Video Optimizer (VO) and remaining 80% non-video packets towards LB2. Then, the branch policy for the DPI is given as the following two elements:$\{(DPI \Lleftarrow VO, 20), (DPI \Lleftarrow LB2, 80)\}$.

Lemma 1: A VNF node i in a given service request is a branch point if the following conditions in Eq. (6.4) and (6.5) hold.

$$\mathscr{B} = \{b_i | i \in N; |\gamma_i| \geq 2\} \tag{6.4}$$

$$b_i = \big\{(i \Lleftarrow v, \alpha) | \alpha \in \mathbb{Z}^+; i, v \in N\big\} \tag{6.5}$$

Merging Policy The merging policy is used to specify the merging point (if exists) of the sub-traffic-flows along respective branch. Merge point $\mathscr{M}$ in Eq. (6.6) is the set of all merging points for a service request. To represent a specific merge point, denoted by m_j in Eq. (6.7), we use $(\Rrightarrow NF)$ to describe the merging point by node j. For instance, in Figure 6.1, the LB2 is the merging point $(\Rrightarrow LB2)$ for three sub-traffic-flows: the web traffic, video traffic and non-video traffic.

Lemma 2: A VNF node j is a merging point if the following conditions in Eq. (6.6) and (6.7) hold.

$$\mathscr{M} = \{m_j | j \in N; |\beta_j| \geq 2\} \tag{6.6}$$

$$m_j = \{(v \Rrightarrow j) | j, v \in N\} \tag{6.7}$$

Source/Destination Policy For each service request, two of the VNFs are specified to be the source (s) and the destination (d) of the flow, respectively.

Conflict Policy The conflict policy specifies the cases in which two VNFs cannot be embedded onto the same physical node. For instance, when node-revisitation (mapping multiple virtual nodes onto the same physical node) is not allowed, there is a conflict policy for every pair of VNFs of the same request.

Free node Policy It is possible that for some of the VNFs in a service, there is no given policy. These nodes are referred to as free nodes.

6.2.2 Example of policy definition

For the service request in Figure 6.1, the set of all policies, $R = \{\mathscr{D}, \mathscr{B}, \mathscr{M}\}$, is given below.

$$\begin{aligned}
\mathscr{D} = \{&(LB1 \succ DPI), (LB1 \succ LB2),\\
&(DPI \succ LB2), (DPI \succ VO), (VO \succ LB2)\}\\
\mathscr{B} = \{&\{(LB1 \Lleftarrow DPI, 40), (LB1 \Lleftarrow LB2, 60)\},\\
&\{(DPI \Lleftarrow VO, 20), (DPI \Lleftarrow LB2, 80)\}\}\\
\mathscr{M} = \{&\{(DPI \Rrightarrow LB2), (LB1 \Rrightarrow LB2), (VO \Rrightarrow LB2)\}\}
\end{aligned}$$

6.3 Branching-Aware Service Function Graph Embedding

In this section, we first present a general definition for the Branching-aware Service Function Graph Embedding (B_SFG), which is further classified into two categories: totally-ordered B_SFG and partially-ordered B_SFG.

6.3.1 Definition of the B_SFG Problem

Definition: **Branching-Aware Service Function Graph Embedding (B_SFG)** Given a service request $NSR =< s, d, \beta, N, R >$, and the physical network $G_p = (V_p, E_p)$, the B_SFG problem is a decision problem that determines whether one can instantiate the request NSR over the physical network while satisfying the following constraints:

1. Function Constraint: Each virtual network function $n \in N$ is mapped to one substrate node $v \in V_p$ that has enough computing resources and supports the required functionality; and nodes that are needed for supporting branching and merging are also properly instantiated;

2. Bandwidth Constraint: Between the hosting nodes of two VNF nodes, a physical path with sufficient bandwidth is allocated in the substrate network to connect these two VNF nodes;

3. Dependence Constraint: VNFs are instantiated in the correct order so the traffic flow can be navigated through without violating the dependence policies.

6.3.2 Classification of the B_SFG Problem

When branching/merging points exist, note that the B_SFG problem can be further classified into two cases depending on the completeness of the given dependence policies, which are elaborated below.

Totally-ordered B_SFG In the case of totally-ordered B_SFG, there is a total dependency order on the VNF set in the service request which is given in the form of a DAG.

Figure 6.1 shows one example of such service requests, as specified with the policies in Section III.C.

Partially-ordered B_SFG Partially-ordered case represents the general scenario of a partial dependency order on the VNF set. Here, some of VNFs on one branch do not have any dependencies on each other (e.g., a VPN proxy can be placed either before or after Firewall [100]). In Figure 6.3, VNF 2 and VNF 3 must succeed the VNF 1 on both branches, respectively. Similarly, VNFs 4, 6 and VNFs 5, 7 must be placed on upper and lower branches, respectively. There is no order dependency between VNFs 4, 6 or VNFs 5, 7. As a result, there are two possible options (VNF 2 → VNF 4 → VNF 6, VNF 2 → VNF 6 → VNF 4) and (VNF 3 → VNF 5 → VNF 7, VNF 3 → VNF 7 → VNF 5) for upper and lower branches, respectively.

It is worth noting that the source VNF has a dependence order with all other nodes in the SFG as it has to be placed before all other nodes. Similarly, there is an order constraint between all the VNFs and the destination node. Due to these order constraints, the SFG is either totally-ordered or partially-ordered.

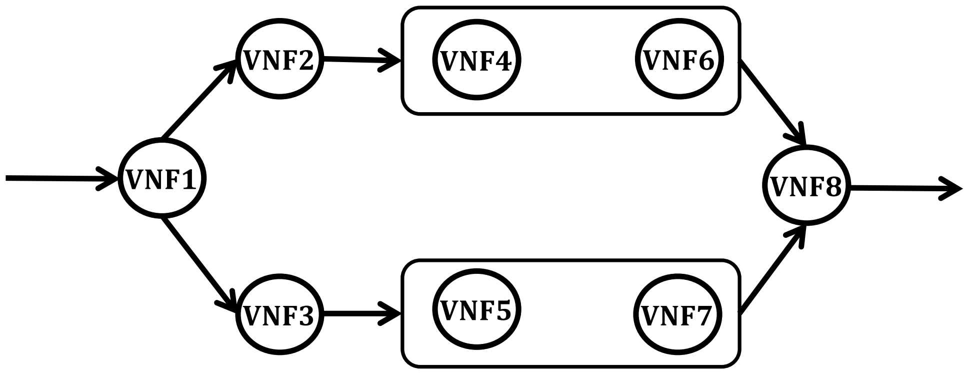


Figure (6.3) An example of a partially-ordered B_SFG

Hereafter, we focus on developing a solution to the first case, the totally-ordered B_SFG problem. Without loss of generality, we assume that for the totally-ordered B_SFG problem, the amount of the bandwidth demand for each virtual link in SFG is given. To solve the totally-ordered B_SFG, one cannot directly apply the traditional VNE methods. That is because in VNE, a fixed undirected virtual network topology is given a priori, whereas in B_SFG, only a set of VNFs with rules and the bandwidth demands for virtual links are given. Hence, B_SFG has to construct the directed SFG and jointly optimize the SFG processes of VNF node/link mapping for an NFV service request.

6.4 Algorithms for B_SFG Problem

In this section, we first introduce a solution, namely B_SFG embedding with Sequential node and link Mapping phases (B_SFG_SM), for embedding branching-aware service requests. Then, we propose our B_SFG embedding with Coordinated Mapping phases (B_SFG_CM) to jointly optimize the SFG node mapping/linking, for embedding branching-aware service requests.

Dependence Sorting To ensure the dependence constraints in $\mathscr{D}$, we set a level (or weight) for a VNF p, denoted by $\ell(p)$, as the following equation, in which β_p is the set of the ancestors of node p.

$$\ell(p) = 1 + \max_{\forall q \in \beta_p} \ell(q), \tag{3}$$

Lemma 3: For any VNF node p, if p is placed after p's ancestors and ahead of p's descendants in the service function graph, then the graph does not violate the dependence constraints in $\mathscr{D}$.

Lemma 4: For any VNF node p, the level of node p, $\ell(p)$, is greater than the level of p's ancestors, and $\ell(p)$ is smaller than the level of p's descendants.

Based on Lemma 3-4, we propose the Dependence Sorting technique to sort all VNF nodes in ascending order based on their *dependence levels*, and save the sorted VNF nodes in a LayerList $\mathbb{L}$. Specifically, each VNF in a request is labeled with a level, and the level of a VNF p, $\ell(p)$, is recursively defined as follows. The level of the entry VNF (a node without any ancestors) is initialized to 0, and the level of every other VNF is computed as the level of its ancestors increased by one. If a VNF has multiple ancestors with different level values, it depends on the ancestor VNF with the maximum level value. It should be noted that all the VNFs from the same level are independent VNFs and, therefore, can be mapped simultaneously. For the example in Figure 6.1, after applying Dependence Sorting, the LayerList of VNF nodes is $\mathbb{L} = \{LB1, DPI, VO, LB2\}$.

Independent Layering To clearly distinguish the VNFs in the same level, we introduce the Independent Layering technique. The details for Dependence Sorting and Independent Layering are presented in Algorithm 10.

More specifically, Line 1-2 conducts the process of Dependence Sorting and Line 3-11 carries on the Independent Layering. Line 3 creates the first layer ℓ_0 and adds $\mathbb{L}$'s first VNF node, v_0, into ℓ_0. In Line 4-10, for a VNF node v_p, if there is any dependence to any node in the previous layer, a new layer is created and v_p is added to the new layer. As a result, the independent layers $\mathbb{L} = \{\ell_0, \ell_1...\ell_j...\}$ are created. As an example, the results of independent layering process for VNF nodes in Figure 6.4.

Algorithm 10 Dependence Sorting and Layering

```
Require: Dependency Graph $\mathscr{D}$
Ensure: List of the levels $\mathbb{L} = \{\ell_0, \ell_1...\ell_j...\}$
1: Calculate level for all VNF nodes in $N$ based on Eq. (3)
2: Ascendingly sort all VNF nodes based on the level and save them in LayerList $\mathbb{L}$;
3: Set layer index $j = 0$ , create layer $\ell_0 = \{v_0\}$, and set $\mathbb{L} = \mathbb{L} - \{v_0\}$;
4: while $\mathbb{L}$ is not empty do
5:     Set $v_p =$ the first node in $\mathbb{L}$ ;
6:     if VNF node $v_p$ has a dependence on any node in $\ell_j$ then
7:         Set $j = j + 1$; create a new layer $\ell_j$;
8:     end if
9:     Set $\ell_j = \ell_j \cup \{v_p\}$;
10:    Set $\mathbb{L} = \mathbb{L} - \{v_p\}$;
11: end while
```

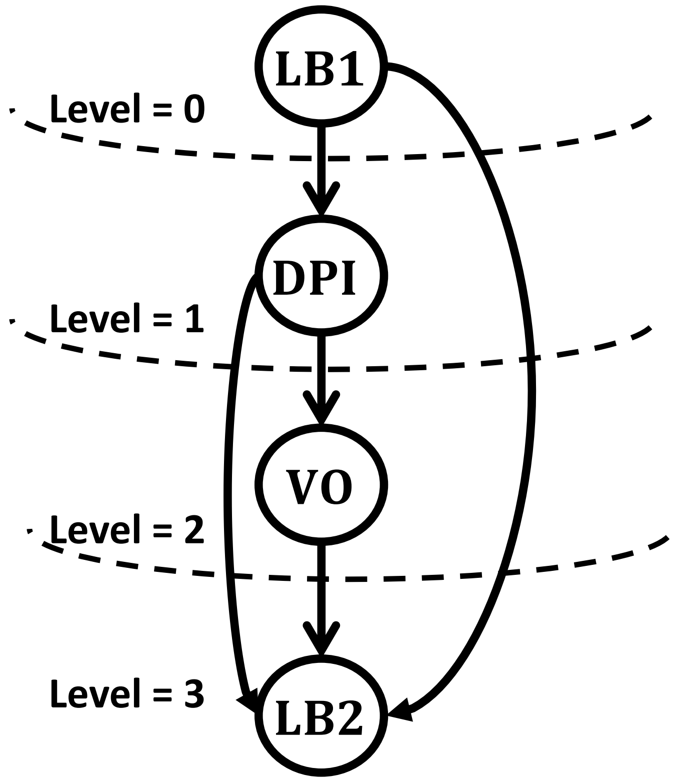


Figure (6.4) Independent layering for the example in Figure 6.1

6.4.1 B_SFG embedding with Sequential Mapping (B_SFG_SM)

In B_SFG_SM, the two components, i.e., node mapping, link mapping, are conducted sequentially. The basic idea of B_SFG_SM is to firstly map all the VNF nodes onto the substrate network and then map the virtual links in the SFG onto the substrate network. In B_SFG_SM, the Dependence Sorting and Layering ensure that the dependence relationships among the VNF nodes are not violated. The details for the B_SFG_SM are presented in Algorithm 11. More specifically, Line 3 creates $\mathbb{L}$, which contains all of the independent layers. Lines 6-13 map all VNF in each level by using the candidate with the highest CPU. Then, the algorithm proceeds to map the VNF links by using the shortest physical path (with sufficient bandwidth) between the hosting nodes of the two ends of each VNF link.

Algorithm 11 B_SFG embedding with Sequential Mapping (B_SFG_SM)

```
Require: Substrate Network, Service Request NSR
Ensure: Mapped Service Request
 1: Initialize Embedded_DAG; LatestVNF;
 2: Create substrate candidate nodes list for each VNF node in the NSR;
 3: Use dependence sorting and layering to generate $\mathbb{L} = \{\ell_0, \ell_1 ... \ell_j ...\}$
 4: Set the Embedded_DAG =$\varnothing$;
 5: Set i=0;
 6: for each $\ell \in L$ do
 7:     while $\ell \neq \varnothing$ do
 8:         Map the VNF node in layer $\ell$ by selecting the candidate with the highest CPU ;
 9:         Add VNF node LatestVNF to the Embedded_DAG;
10:         Update $\ell$ and the candidate substrate list;
11:     end while
12:     Set i=i+1;
13: end for
14: for each VNF node $\in$ Embedded_DAG do
15:     Map the VNF links between the VNF node and its parents using the shortest physical
        path
16: end for
```

6.4.2 B_SFG embedding with Coordinated Mapping (B_SFG_CM)

With sequential node and link mapping, one cannot take advantage of the resource information in the substrate network to efficiently minimize the bandwidth consumption. Accordingly, we propose the B_SFG embedding with Coordinated Mapping (B_SFG_CM) in Algorithm 12 to conduct the node and link mapping processes in one coordinated step. That is, while the VNFs are mapped onto the substrate network, virtual links (directed edges in the SFG) are also allocated onto the substrate network. More specifically, Line 3 creates the independent layers. Line 8-15 map the VNF node with a candidate substrate node that is the closest to its mapped ancestor in substrate node, until all the VNF nodes in a layer ℓ_i are accommodated. Then B_SFG_CM maps the VNF nodes in the next level ℓ_{i+1}, until all the VNF nodes are mapped onto the substrate network.

Algorithm 12 B_SFG embedding with Coordinated Mapping B_SFG_CM)

```
Require: Substrate Network, Service Request NSR
Ensure: Mapped Service Request;
 1: Initialize Embedded_DAG; LatestVNF; LatestSub;
 2: Create substrate candidate nodes list for each VNF node in the NSR;
 3: Use dependence sorting and layering to generate $\mathbb{L} = \{\ell_0, \ell_1 ... \ell_j ...\}$
 4: Set the Embedded_DAG =$\varnothing$;
 5: Map the first VNF node in $\ell_0$ onto the candidate substrate node with the highest CPU;
    update LatestVNF, LatestSub, and L0;
 6: Add VNF node LatestVNF to the Embedded_DAG;
 7: i=1;
 8: for each $\ell \in L$ do
 9:     while $\ell \neq \emptyset$ do
10:         Sort all the candidate substrate nodes of the VNF nodes in $\ell$ based on the distance
            to their parents in the previous levels;
11:         Set LatestVNF as the VNF node that has the candidate node $S$ and node $S$ has the
            shortest physical path to substrate node that holds the parent of the current node;
12:         Add VNF node LatestVNF to the Embedded_DAG; Map LatestVNF to $S$ and the
            related VNF link;
13:         Update $\ell$, LatestSub and the candidate substrate list;
14:     end while
15:     Set i=i+1;
16: end for
```

6.5 PERFORMANCE EVALUATION

We use a 28-node US Backbone network [109] as the substrate IP network. Unless otherwise specified, the available computing resource of substrate nodes changes in the range of $[5, 35]$; the available functionality at each substrate node is randomly generated; and the available bandwidth for each substrate link varies from 5 to 45. Similarly, unless otherwise specified, the number of VNF nodes (N) in an NFV service request (NSR) is set in the range of $[3, 9]$; the dependent constraints among the VNF nodes and the branching points are randomly generated; each VNF node requests a computing resources in the range of $[5, 25]$, and VNF links in each NSR requests the bandwidth within the range of $[5, 25]$. The NSR is randomly generated and we collect the average bandwidth consumption for embedding a large number of NSRs, which is denoted as "Average BW Consumption" in the following figures.

In Figure 6.5, the y-axis denotes the average bandwidth consumption and the x-axis represents the number of VNF nodes in the request. For this experiment, we set each substrate node with unlimited computing resources and set each substrate link with unlimited bandwidth. From this figure, we see the impact of NSR size on the bandwidth consumption. As one can see, for both algorithms, the average bandwidth consumption increases with the number of VNF nodes in the NSR. B_SFG_CM significantly outperforms B_SFG_SM. This is because the proposed B_SFG_CM can jointly optimize the node and link mapping processes.

In Figure 6.6 the x-axis, represents the bandwidth demand for the VNF nodes in the service request. Each NSR is randomly generated with 7 VNF nodes and 8 dependencies and the requested bandwidth for the links in NSR varies from 5 to 20. One can see that when the requested bandwidth increases, the average bandwidth consumption of both algorithms increases as well. In particular, B_SFG_CM outperforms B_SFG_SM because the B_SFG_CM algorithm can jointly optimize the VNF node/link mapping processes. It is worth noting that when the NSR bandwidth request is higher than 20, due to the lack of available physical links with enough bandwidth, there is not many alternative physical paths for B_SFC embedding

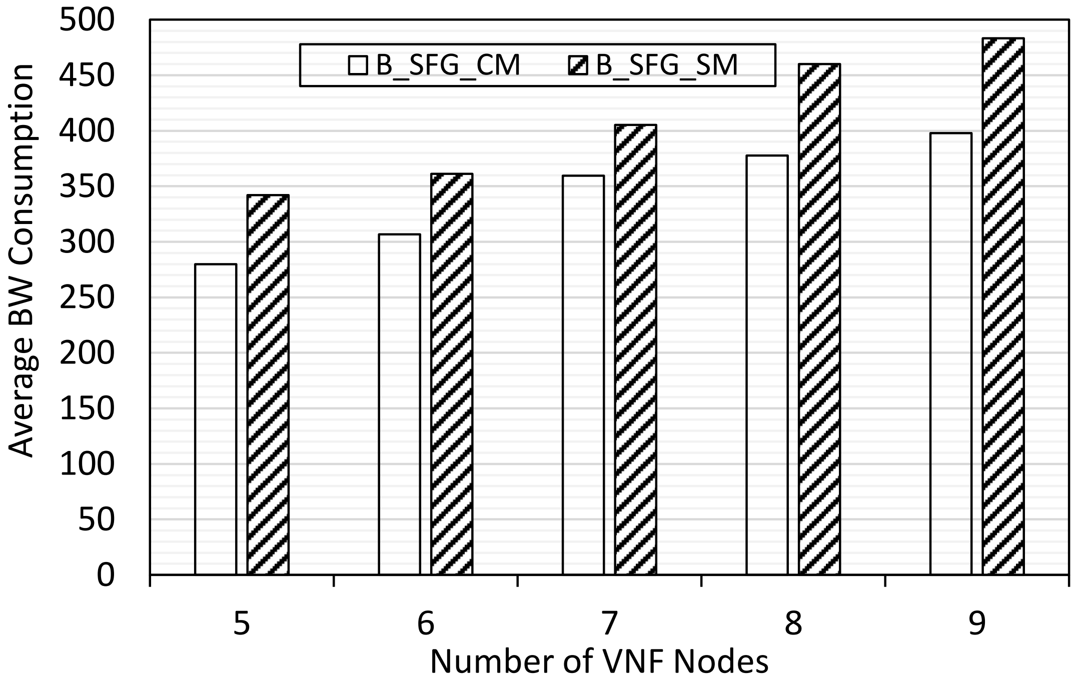


Figure (6.5) Bandwidth consumption versus number of VNF nodes.

and both algorithms converging to the similar performance.

Figure 6.7 demonstrate the impact of the available CPU in the substrate network. For this scenario, we set the range for available CPU resources of the substrate nodes as shown in Figure 6.7. We set the available bandwidth of the substrate links as unlimited. Each NSR is randomly generated with 6 VNF nodes and 10 dependencies. The CPU requirement for all VNF nodes is 12 and the requested bandwidth for the NSR is set to 10. One can see that, the average bandwidth consumption decreases when increasing the range of available CPU resources. This is because more available substrate nodes can be candidates to accommodate the CPU requirement of the VNF nodes when the range for available CPU resources increases, resulting in shorter physical paths. Once again, the proposed B_SFC_CM yields better results than B_SFC_SM.

In this chapter, we introduced the problem of Branching-Aware Service Function Graph

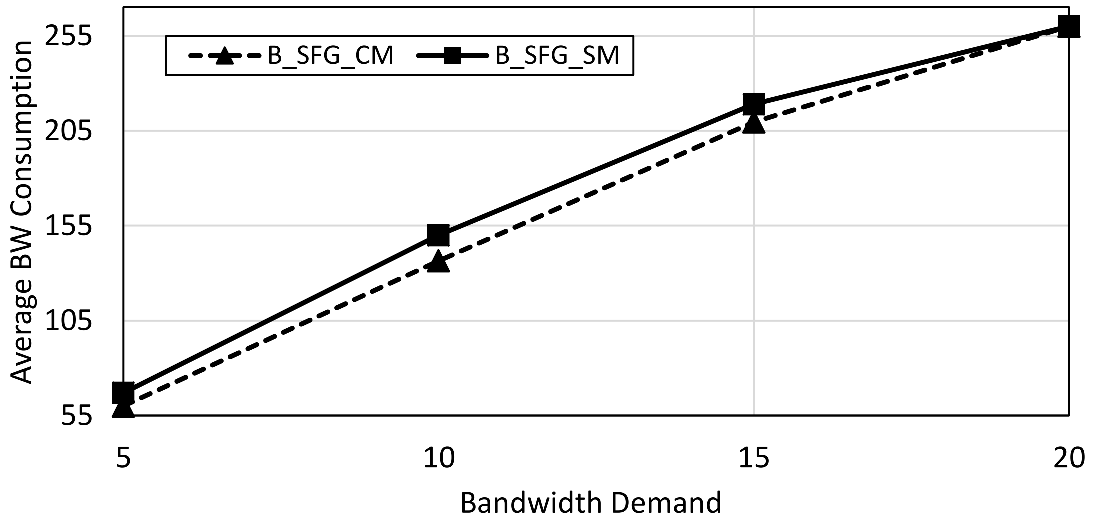


Figure (6.6) Bandwidth consumption versus NSR bandwidth

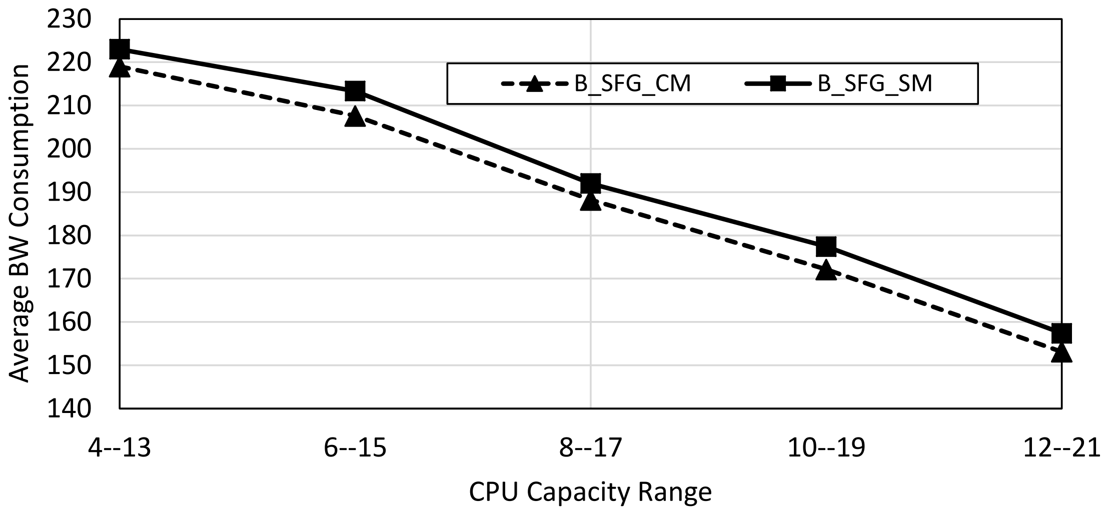


Figure (6.7) Bandwidth consumption versus substrate CPU capacity

(B_SFG) embedding and studied how to efficiently embed the requests in the substrate network while considering different constraints such as branching in the service function graph.

CHAPTER 7

SUMMARY

Network Function Virtualization (NFV) is an emerging technology that promises to address issues in traditional middleboxes, providing service flexibility and reduced cost. NFV decouples network functions from the proprietary middlebox hardware, thus allowing the network providers to implement network functions on virtual machines running in standard servers. Combining NFV with Software-Defined Networking (SDN) technology, future networks such as 5G, mobile networks and optical networks are expected to be operated and utilized at lower cost and higher flexibility. To deliver end-to-end services, it is often required to navigate the traffic to pass through a number of network functions in a pre-defined order, which is referred to as Service Function Graph.

In this dissertation, we have resolved the fundamental problem of how to efficiently provision user demands via resource management in the physical network. We call this problem as Service Function Graph Design and Embedding (SFG_DE). We have extensively reviewed the related work and presented a classified overview of the literature study. More specifically, we study variations of the SFG_DE problem. First, we investigate the service graph design and embedding in the presence of dependence constraint in the service request. We propose a number of algorithms for designing and constructing the service function graph taking into account the dependence requirement. Next, we consider the case where the data stream can branch out to multiple data streams at certain network functions (e.g., a load balancer). This branching indicates that the packet flow may be directed to multiple different paths, resulting in a more complex mesh-like service function graph. We model this problem and provide algorithms for embedding the request onto the physical network. In the future, we plan to study the combination of dependence and branch requirement for a given service request.

REFERENCES

[1] N. M. K. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *IEEE Communications magazine*, vol. 47, no. 7, 2009.

[2] K. Tutschku, T. Zinner, A. Nakao, and P. Tran-Gia, "Network virtualization: Implementation steps towards the future internet," *Electronic Communications of the EASST*, vol. 17, 2009.

[3] A. Berl, A. Fischer, and H. de Meer, "Using system virtualization to create virtualized networks," *Electronic Communications of the EASST*, vol. 17, 2009.

[4] J. Carapinha and J. Jiménez, "Network virtualization: a view from the bottom," in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures.* ACM, 2009, pp. 73–80.

[5] Y. Zhu and M. H. Ammar, "Algorithms for assigning substrate network resources to virtual network components." in *INFOCOM*, vol. 1200, no. 2006, 2006, pp. 1–12.

[6] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures.* ACM, 2009, pp. 81–88.

[7] N. F. Butt, M. Chowdhury, and R. Boutaba, "Topology-awareness and reoptimization mechanism for virtual network embedding," in *International Conference on Research in Networking.* Springer, 2010, pp. 27–39.

[8] S. Zhang, Z. Qian, S. Guo, and S. Lu, "Fell: A flexible virtual network embedding algorithm with guaranteed load balancing," in *IEEE International Conference on Communications (ICC).* IEEE, 2011, pp. 1–5.

[9] S. Zhang, Z. Qian, B. Tang, J. Wu, and S. Lu, "Opportunistic bandwidth sharing for virtual network mapping," in *IEEE Global Communications Conference (GLOBECOM)*, 2011, pp. 1–5.

[10] Z. Zhang, S. Su, J. Zhang, K. Shuang, and P. Xu, "Energy aware virtual network embedding with dynamic demands: Online and offline," *Computer Networks*, vol. 93, pp. 448–459, 2015.

[11] D. Stezenbach, M. Hartmann, and K. Tutschku, "Parameters and challenges for virtual network embedding in the future internet," in *IEEE Network Operations and Management Symposium (NOMS)*, 2012, pp. 1272–1278.

[12] G. Sun, H. Yu, L. Li, V. Anand, Y. Cai, and H. Di, "Exploring online virtual networks mapping with stochastic bandwidth demand in multi-datacenter," *Photonic Network Communications*, vol. 23, no. 2, pp. 109–122, 2012.

[13] J. Inführ and G. R. Raidl, "Introducing the virtual network mapping problem with delay, routing and location constraints," in *Network optimization*. Springer, 2011, pp. 105–117.

[14] W.-L. Yeow, C. Westphal, and U. Kozat, "Designing and embedding reliable virtual infrastructures," in *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*. ACM, 2010, pp. 33–40.

[15] Q. Hu, "Towards a Virtualized Next Generation Internet," 2015.

[16] R. Guerzoni *et al.*, "Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper," in *SDN and OpenFlow World Congress*, vol. 1, 2012, pp. 5–7.

[17] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[18] G. ETSI, "Network functions virtualisation (nfv): Architectural framework," *ETsI Gs NFV*, vol. 2, no. 2, p. V1, 2013.

[19] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "NFV: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Network*, vol. 28, no. 6, pp. 18–26, 2014.

[20] D. Cotroneo, L. De Simone, A. K. Iannillo, A. Lanzaro, R. Natella, J. Fan, and W. Ping, "Network function virtualization: Challenges and directions for reliability assurance," in *Software Reliability Engineering Workshops (ISSREW)*, 2014, pp. 37–42.

[21] J. Halpern and C. Pignataro, "Service function chaining (sfc) architecture," Tech. Rep., 2015.

[22] P. Quinn and J. Guichard, "Service function chaining: Creating a service plane via network service headers," *Computer*, vol. 47, no. 11, pp. 38–44, 2014.

[23] P. Quinn and T. Nadeau, "Problem statement for service function chaining," Tech. Rep., 2015.

[24] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[25] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.

[26] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[27] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE communications surveys & tutorials*, vol. 16, no. 4, pp. 1955–1980, 2014.

[28] M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," *Communications of the ACM*, vol. 57, no. 10, pp. 86–95, 2014.

[29] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[30] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[31] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 38–47, 2011.

[32] S. Herker, A. Khan, and X. An, "Survey on survivable virtual network embedding problem and solutions," in *International Conference on Networking and Services, ICNS*, 2013.

[33] X. Chang, B. Wang, J. Liu, W. Wang, and J. Muppala, "Green cloud virtual network provisioning based ant colony optimization," in *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation.* ACM, 2013, pp. 1553–1560.

[34] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 1, pp. 206–219, 2012.

[35] S. Zhang, Z. Qian, J. Wu, and S. Lu, "An opportunistic resource sharing and topology-aware mapping framework for virtual networks," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2012, pp. 2408–2416.

[36] S. Zhang, Z. Qian, J. Wu, S. Lu, and L. Epstein, "Virtual network embedding with opportunistic resource sharing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 816–827, 2014.

[37] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[38] T. Trinh, H. Esaki, and C. Aswakul, "Quality of service using careful overbooking for optimal virtual network resource allocation," in *8th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON).* IEEE, 2011, pp. 296–299.

[39] X. Zhang, C. Phillips, and X. Chen, "An overlay mapping model for achieving enhanced qos and resilience performance," in *3rd International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT).* IEEE, 2011, pp. 1–7.

[40] C. Wang, S. Shanbhag, and T. Wolf, "Virtual network mapping with traffic matrices," in *IEEE International Conference on Communications (ICC)*, 2012, pp. 2717–2722.

[41] G. Schaffrath, S. Schmid, and A. Feldmann, "Optimizing long-lived cloudnets with migrations," in *IEEE Fifth International Conference on Utility and Cloud Computing (UCC)*, 2012, pp. 99–106.

[42] "ILOG, Inc, "ILOG CPLEX: High-performance software for mathematical programming and optimization," http://www.ilog.com/products/cplex/, 2012.

[43] "Glpk, "gnu linear programming kit,"," http://www.gnu.org/software/glpk, 2008.

[44] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2009, pp. 783–791.

[45] I. Houidi, W. Louati, W. B. Ameur, and D. Zeghlache, "Virtual network provisioning across multiple substrate networks," *Computer Networks*, vol. 55, no. 4, pp. 1011–1023, 2011.

[46] Q. Hu, Y. Wang, and X. Cao, "Resolve the virtual network embedding problem: A column generation approach," in *INFOCOM, 2013 Proceedings IEEE.* IEEE, 2013, pp. 410–414.

[47] M. E. Lübbecke and J. Desrosiers, "Selected topics in column generation," *Operations research*, vol. 53, no. 6, pp. 1007–1023, 2005.

[48] Q. Hu, Y. Wang, and X. Cao, "Resolve the virtual network embedding problem: A column generation approach," in *IEEE International Conference on Computer Communications (INFOCOM)*, May 2013, pp. 410–414.

[49] ——, "Virtual network embedding: An optimal decomposition approach," in *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on.* IEEE, 2014, pp. 1–6.

[50] M. Dorigo, "Optimization, learning and natural algorithms," *PhD Thesis, Politecnico di Milano*, 1992.

[51] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on.* IEEE, 1995, pp. 39–43.

[52] A. Pages, J. Perello, S. Spadaro, and G. Junyent, "Strategies for virtual optical network allocation," *IEEE Communications Letters*, vol. 16, no. 2, pp. 268–271, 2012.

[53] G. Sun, H. Yu, V. Anand, L. Li, and H. Di, "Optimal provisioning for virtual network request in cloud-based data centers," *Photonic Network Communications*, vol. 24, no. 2, pp. 118–131, 2012.

[54] P. Lv, X. Wang, and M. Xu, “Virtual access network embedding in wireless mesh networks,” *Ad Hoc Networks*, vol. 10, no. 7, pp. 1362–1378, 2012.

[55] I. Fajjari, N. A. Saadi, G. Pujolle, and H. Zimmermann, “Vne-ac: Virtual network embedding algorithm based on ant colony metaheuristic,” in *IEEE International Conference on Communications (ICC).* IEEE, 2011, pp. 1–6.

[56] Z. Zhang, X. Cheng, S. Su, Y. Wang, K. Shuang, and Y. Luo, “A unified enhanced particle swarm optimization-based virtual network embedding algorithm,” *International Journal of Communication Systems*, vol. 26, no. 8, pp. 1054–1073, 2013.

[57] H. Di, L. Li, V. Anand, H. Yu, and G. Sun, “Cost efficient virtual infrastructure mapping using subgraph isomorphism,” in *Asia Communications and Photonics Conference and Exhibition.* Optical Society of America, 2010, p. 79890L.

[58] T. Ghazar and N. Samaan, “Hierarchical approach for efficient virtual network embedding based on exact subgraph matching,” in *IEEE Global Communications Conference (GLOBECOM)*, 2011, pp. 1–6.

[59] D. Yun and Y. Yi, “Virtual network embedding in wireless multihop networks,” in *Proceedings of the 6th international conference on future internet technologies.* ACM, 2011, pp. 30–33.

[60] X. Chen, Y. Luo, and J. Wang, “Virtual network embedding with border matching,” in *Fourth International Conference on Communication Systems and Networks (COMSNETS).* IEEE, 2012, pp. 1–8.

[61] J. Liu, T. Huang, J.-y. Chen, and Y.-j. Liu, “A new algorithm based on the proximity principle for the virtual network embedding problem,” *Journal of Zhejiang University SCIENCE C*, vol. 12, no. 11, p. 910, 2011.

[62] A. Razzaq and M. S. Rathore, “An approach towards resource efficient virtual network

embedding," in *Second International Conference on Evolving Internet (INTERNET).* IEEE, 2010, pp. 68–73.

[63] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *IEEE International Conference on Computer Communications (INFOCOM)*, Apr. 2006, pp. 1–12.

[64] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, 2008.

[65] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization." in *CloudNet*, 2015, pp. 171–177.

[66] H. Moens and F. De Turck, "Vnf-p: A model for efficient placement of virtualized network functions," in *10th International Conference on Network and Service Management (CNSM)*, 2014, pp. 418–423.

[67] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *Network Operations and Management Symposium (NOMS).* IEEE, 2014, pp. 1–9.

[68] R. Riggio, A. Bradai, T. Rasheed, J. Schulz-Zander, S. Kuklinski, and T. Ahmed, "Virtual network functions orchestration in wireless networks," in *11th International Conference on Network and Service Management (CNSM).* IEEE, 2015, pp. 108–116.

[69] M. Bagaa, T. Taleb, and A. Ksentini, "Service-aware network function placement for efficient traffic handling in carrier cloud," in *Wireless Communications and Networking Conference (WCNC).* IEEE, 2014, pp. 2402–2407.

[70] M. Bouet, J. Leguay, T. Combe, and V. Conan, "Cost-based placement of vdpi functions in nfv infrastructures," *International Journal of Network Management*, vol. 25, no. 6, pp. 490–506, 2015.

[71] Q. Zhang, X. Wang, I. Kim, P. Palacharla, and T. Ikeuchi, “Vertex-centric computation of service function chains in multi-domain networks,” in *NetSoft Conference and Workshops (NetSoft).* IEEE, 2016, pp. 211–218.

[72] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, “Deploying chains of virtual network functions: On the relation between link and server usage,” in *IEEE International Conference on Computer Communications (INFOCOM).* IEEE, 2016, pp. 1–9.

[73] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, “Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions,” in *IFIP/IEEE International Symposium on Integrated Network Management (IM).* IEEE, 2015, pp. 98–106.

[74] A. Basta, W. Kellerer, M. Hoffmann, H. J. Morper, and K. Hoffmann, “Applying nfv and sdn to lte mobile core gateways, the functions placement problem,” in *Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications,Challenges*, ser. AllThingsCellular ’14, 2014, pp. 33–38.

[75] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, “Near optimal placement of virtual network functions,” in *Computer Communications (INFOCOM), 2015 IEEE Conference on.* IEEE, 2015, pp. 1346–1354.

[76] J. Fan, C. Guan, Y. Zhao, and C. Qiao, “Availability-aware mapping of service function chains,” in *IEEE International Conference on Computer Communications (INFOCOM)*, May 2017, pp. 1–9.

[77] M. Rost and S. Schmid, “Service chain and virtual network embeddings: Approximations using randomized rounding,” *arXiv preprint arXiv:1604.02180*, 2016.

[78] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and S. Davy, “Design and evaluation of algorithms for mapping and scheduling of virtual network functions,” in *1st IEEE Conference on Network Softwarization (NetSoft).* IEEE, 2015, pp. 1–9.

[79] S. Sahhaf, W. Tavernier, D. Colle, and M. Pickavet, "Network service chaining with efficient network function mapping based on service decompositions," in *IEEE 1st Conference on Network Softwarization (NetSoft)*, Apr. 2015, pp. 1–5.

[80] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *IEEE 21st International Workshop on Local and Metropolitan Area Networks*, Apr. 2015, pp. 1–6.

[81] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *11th International Conference on Network and Service Management (CNSM)*. IEEE, 2015, pp. 50–56.

[82] H. Pirkul and V. Jayaraman, "A multi-commodity, multi-plant, capacitated facility location problem: formulation and efficient heuristic solution," *Computers & Operations Research*, vol. 25, no. 10, pp. 869–878, 1998.

[83] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[84] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, Oct. 2014, pp. 7–13.

[85] R. Bruschi, A. Carrega, and F. Davoli, "A game for energy-aware allocation of virtualized network functions," *Journal of Electrical and Computer Engineering*, p. 2, 2016.

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

[87] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," in *International Colloquium on Structural Information and Communication Complexity.* Springer, 2015, pp. 104–118.

[88] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Dynamic chaining of virtual network functions in cloud-based edge networks," in *1st IEEE Conference on Network Softwarization (NetSoft)*, Apr. 2015, pp. 1–5.

[89] F. Wang, R. Ling, J. Zhu, and D. Li, "Bandwidth guaranteed virtual network function placement and scaling in datacenter networks," in *34th IEEE International Performance, Computing and Communications Conference (IPCCC)*, 2015, pp. 1–8.

[90] M. Jalalitabar, G. Luo, C. Kong, and X. Cao, "Service function graph design and mapping for NFV with priority dependence," in *IEEE Global Communications Conference (GLOBECOM)*, Dec. 2016, pp. 1–5.

[91] M. Jalalitabar, E. Guler, G. Luo, L. Tian, and X. Cao, "Dependence-aware service function chain design and mapping," in *IEEE Global Communications Conference (GLOBECOM)*, Dec. 2017, pp. 1–6.

[92] M. Jalalitabar, E. Guler, D. Zheng, G. Luo, L. Tian, and X. Cao, "Embedding dependence-aware service function chains," *Journal of Optical Communications and Networking*, vol. 10, no. 8, pp. C64–C74, 2018.

[93] M. T. Beck and J. F. Botero, "Coordinated allocation of service function chains," in *IEEE Global Communications Conference (GLOBECOM)*, Dec. 2015, pp. 1–6.

[94] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent nfv middleboxes," in *IEEE International Conference on Computer Communications (INFOCOM)*, May 2017, pp. 1–9.

[95] X. Gao, Z. Ye, J. Fan, W. Zhong, Y. Zhao, X. Cao, H. Yu, and C. Qiao, "Virtual network mapping for multicast services with max-min fairness of reliability," *IEEE/OSA*

Journal of Optical Communications and Networking, vol. 7, no. 9, pp. 942–951, Sep. 2015.

[96] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Network function placement for NFV chaining in packet/optical datacenters," *IEEE Journal of Lightwave Technology*, vol. 33, no. 8, pp. 1565–1570, Apr. 2015.

[97] V. Mehmeri, X. Wang, Q. Zhang, P. Palacharla, T. Ikeuchi, and I. T. Monroy, "Optical network as a service for service function chaining across datacenters," in *Optical Fiber Communications Conference and Exhibition (OFC)*, Mar. 2017, pp. 1–3.

[98] J. F. Riera, X. Hesselbach, E. Escalona, J. A. García-Espín, and E. Grasa, "On the complex scheduling formulation of virtual network functions over optical networks," in *16th International Conference on Transparent Optical Networks (ICTON)*, Jul. 2014, pp. 1–5.

[99] "Cisco: NAT order of operation," http://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/6209-5.html.

[100] "Microsoft Technet: VPNs and firewalls," https://technet.microsoft.com/en/us/library/cc958037.aspx.

[101] S. Kumar, M. Tufail, S. Majee, C. Captari, and S. Homma, "Service function chaining use cases in data centers," *IETF SFC WG*, 2015.

[102] J. Napper, W. Haeffner, M. Stiemerling, D. Lopez, and J. Uttaro, "Service function chaining use cases in mobile networks," *Internet Engineering Task Force, Internet-Draft draft-ietf-sfc-usecase-mobility-06*, 2016.

[103] S.-I. Lee and M.-K. Shin, "A self-recovery scheme for service function chaining," in *International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2015, pp. 108–112.

[104] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[105] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Section 22.4: Topological sort," *Introduction to Algorithms (2nd ed.), MIT Press and McGraw-Hill*, pp. 549–552, 2001.

[106] F. S. Abkenar and A. G. Rahbar, "Study and analysis of Routing and Spectrum Allocation (RSA) and Routing, Modulation and Spectrum Allocation (RMSA) algorithms in Elastic Optical Networks (EONs)," *Optical Switching and Networking*, vol. 23, pp. 5–39, 2017.

[107] B. Wang, X. Chang, J. Liu, and J. K. Muppala, "Reducing power consumption in embedding virtual infrastructures," in *IEEE Global Communications Conference (GLOBECOM)*, Dec. 2012, pp. 714–718.

[108] Y. Chen, J. Li, T. Wo, C. Hu, and W. Liu, "Resilient virtual network service provision in network virtualization environments," in *IEEE International Conference on Parallel and Distributed Systems*, Dec. 2010, pp. 51–58.

[109] L. Gong, X. Zhou, X. Liu, W. Zhao, W. Lu, and Z. Zhu, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 5, no. 8, pp. 836–847, Aug. 2013.

[110] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: Enabling network function parallelism in nfv," in *Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 43–56.

[111] Y. Zhang, B. Anwer, V. Gopalakrishnan, B. Han, J. Reich, A. Shaikh, and Z.-L. Zhang, "Parabox: Exploiting parallelism for virtual network functions in service chaining," in *Symposium on SDN Research*. ACM, 2017, pp. 143–149.

[112] X. Lin, D. Guo, Y. Shen, G. Tang, and B. Ren, "Dag-sfc: Minimize the embedding cost of sfc with parallel vnfs," in *47th International Conference on Parallel Processing*. ACM, 2018, p. 15.

[113] ETSI, "Network Functions Virtualisation (NFV); terminology for main concepts in NFV," Dec. 2014.