5-4-2021

Network Function Virtualization Service Delivery In Future Internet

Danyang Zheng

NETWORK FUNCTION VIRTUALIZATION SERVICE DELIVERY IN FUTURE INTERNET

by

Danyang Zheng

Under the Direction of Xiaojun Cao, PhD

ABSTRACT

This dissertation investigates the Network Function Virtualization (NFV) service delivery problems in the future Internet. With the emerging Internet of everything, 5G communication and multi-access edge computing techniques, tremendous end-user devices are connected to the Internet. The massive quantity of end-user devices facilitates various services between the end-user devices and the cloud/edge servers. To improve the service quality and agility, NFV is applied. In NFV, the customer's data from these services will go through multiple Service Functions (SFs) for processing or analysis. Unlike traditional point-to-point data transmission, a particular set of SFs and customized service requirements are needed to be applied to the customer's traffic flow, which makes the traditional point-to-point data transmission methods not directly used. As the traditional point-to-point data transmis-

sion methods cannot be directly applied, there should be a body of novel mechanisms that effectively deliver the NFV services with customized requirements.

As a result, this dissertation proposes a series of mechanisms for delivering NFV services with diverse requirements. First, we study how to deliver the traditional NFV service with a provable boundary in unique function networks. Secondly, considering both forward and backward traffic, we investigate how to effectively deliver the NFV service when the SFs required in forward and backward traffic is not the same. Thirdly, we investigate how to efficiently deliver the NFV service when the required SFs have specific executing order constraints. We also provide detailed analysis and discussion for proposed mechanisms and validate their performance via extensive simulations. The results demonstrate that the proposed mechanisms can efficiently and effectively deliver the NFV services under different requirements and networking conditions.

At last, we also propose two future research topics for further investigation. The first topic focuses on parallelism-aware service function chaining and embedding. The second topic investigates the survivability of NFV services.

INDEX WORDS: Network Function Virtualization, Software-Defined Networking, Service Function Chaining and Embedding, Approximation Algorithm.

NETWORK FUNCTION VIRTUALIZATION SERVICE DELIVERY IN FUTURE INTERNET

by

Danyang Zheng

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

2021

NETWORK FUNCTION VIRTUALIZATION SERVICE DELIVERY IN FUTURE INTERNET

by

Danyang Zheng

Committee Chair: Xiaojun Cao

Committee: Yi Pan

Shihao Ji

Yi Zhao

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

May 2021

DEDICATION

This dissertation is dedicated to my parents Ling Yang and Dongli Zheng, my step father Tao Zhong and my grandmother Jiuhua Su for their endless support and love during my Ph.D. years. I cannot finish my Ph.D. without their love and encouragement

ACKNOWLEDGMENTS

Pursuing my Ph.D. degree in Georgia State University during the past five years has been a truly life-changing experience for me. I would never have been able to finish my dissertation without the guidance of my committee members, help from my group, and support from my family and my friends. I would like to show my great gratitude to all of them.

First of all, I would like to show my deepest gratitude to my advisor Dr. Xiaojun Cao and Dr. Yi Pan. They provided me with an excellent environment for research, and gave me many opportunities to promote myself. They always inspired me when I felt frustrated, and provided valuable suggestions and guidances when I was trapped in my research. Thanks to their selfless help, I can totally focus on my research and gain a lot of skills on it. Dr. Cao not only has taught me the thorough methodologies to carry out my academic career, but also inspired me to pursue my life and achieve self-actualization. His hard-working, open-mind, brilliant, and warm-hearted has set a good example for me to follow. I am very grateful for all his instruction and priceless help on me. By Dr. Pan's excellent research skills, earnest, and preciseness, he has not only taught me the methodologies used for my study, but also the right attitudes towards my research, career and life. I will always keep the attitudes in my whole life.

It is very grateful and a great honor to have Dr. Shihao (Jonathan) Ji, and Dr. Yi Zhao in my committee, who gave me great supports for my Ph.D. study and spared time to participate in my dissertation committee.

I am also very thankful to the professors and staffs at our department, especially Dr. Yingshu Li, Dr. Zhipeng Cai, Dr. Raj Sunderraman, and Dr. Alex Zelikovsky for their great help in my Ph.D. study. I thank Ms. Tammie Dudley, Ms. Adrienne Martin, Mr. Paul Bryan, Ms. Venette Rice, and Ms. Celena Pittman for their patient help, which makes my study at our department much easier and more convenient.

I also wish to express my gratitude to my former undergraduate advisor Prof. Ling Tian, who initially enlightened me to the world of computer science, and provided me the chance to further study with Dr. Cao at Georgia State University. I also appreciate the help and support from Prof. Guangchun Luo and Prof. Aiguo Chen, who kindly helped me handle many issues in China, and many colleagues in China, especially Dr. Ke Yan.

Also many thanks go to colleagues in my group and department. Special thanks for my group colleagues Dr. Chenguang Kong, Dr. Evrim Guler, Dr. Maryam Jalalitabar, Chengzong Peng, Xueting Liao, my friends Dr. Xu Zheng, Dr. Dongjing Miao, Dr. Yi Liang, Dr. Yan Huang, Xuebing Wu, Shubin Wu, Saide Zhu, Yaxin Deng, Yibin Xie, Jisheng Yang, Kainan Zhang, Honghui Xu, who helped me study and live in U.S..

My parents and my grandmother deserve special mention for their love, encouragement, support, and patience during my life and my study in America. The dissertation is impossible without their support. They shed lights in my heart every time when I felt lonely and tired.

Last but not least, it is a pleasure to thank everybody who made the dissertation possible, as well as express my apologies that I could not mention personally one by one.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

- NFV - Network Function Virtualization
- VNF - Virtual Network Function
- SF - Service Function
- SFC - Service Function Chain
- NSR - NFV/Network Service Request
- SN - Substrate Network
- SON - Substrate Optical Network
- PN - Physical Network
- SFCE - Service Function Chaining and Embedding
- SFP - Service Function Path
- UFSN/UFN - Unique Function Substrate Network
- MFSN/MFN - Multiple Functions Substrate Network

CHAPTER 1
INTRODUCTION

With the advancement in the hardware, software and virtualization techniques, more and more state-of-the-arts networking frameworks and architectures are proposed to facilitate a convenient lives for human beings [1][2][3]. Internet of Things (IoT) techniques enable great number of edge devices connecting to the Internet and create tremendous opportunities for IT developers to design diverse edge applications [1, 4]. Cloud computing systems allow the end-users to offload their computation-intensive tasks to remote servers, thus saving Capital Expenditure (CAPEX) and Operational Expense (OPEX) at the local [2, 5, 6]. Multi-access Edge Computing (MEC) systems empower the latency-sensitive and computation-intensive applications (e.g., Pokemon GO, online machine learning and etc) and provide efficient computing pool at network edges [3, 7, 8]. When designing the above systems, one of the common things that the service providers need take into consideration is how to efficiently satisfy and deliver the services to customers.

To satisfy the service requests from the customers, Network Virtualization enables the possibilities of flexible resource allocations, agile delivery options and smart service scaling.

1.1 Network Virtualization

Network Virtualization (NV) refers to abstracting network resource that was traditionally hosted in hardware to software. NV can combine multiple physical networks to one virtual network, or it can divide one physical network into separate, independent virtual networks [9]. For example, in Fig. 1.1, the physical network is abstracted into two separate virtual networks

with diverse amounts of resource, which can be managed by different service providers. It is worthy of noticing that, the virtual network topology might be different than the physical network. For example, the first virtual link in virtual network 2 represents the connection from Node F to Node C (i.e., F $\rightarrow$ B $\rightarrow$ C) in the physical network. But, the virtual node in the virtual network will share the same position as the physical node does. With the support of the network virtualization, the other problem for the service providers is how to effectively manage those resource and smartly control the service delivery. For this case, we will introduce the concept of the Software-Defined Networking (SDN).

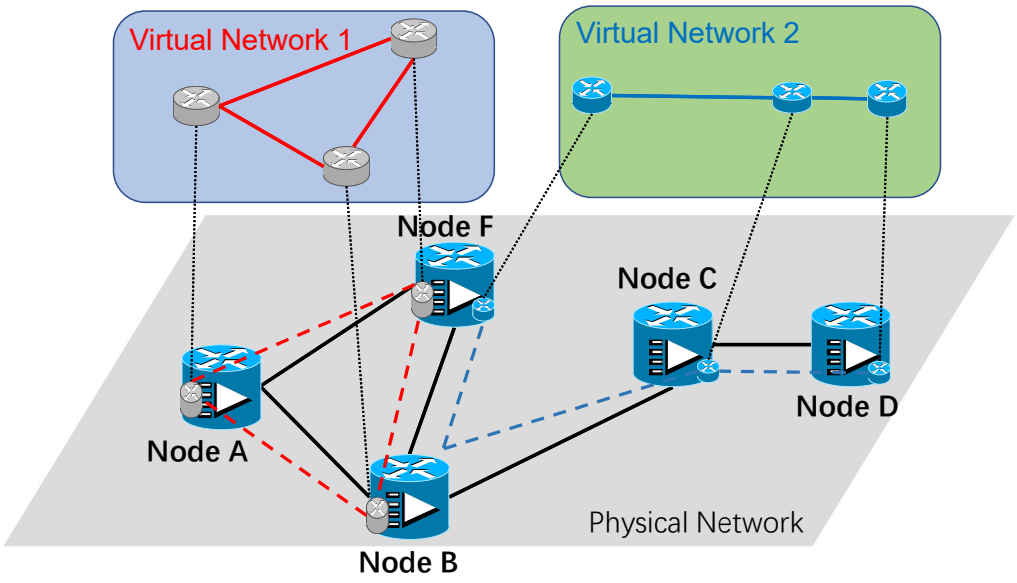


Figure 1.1: An example of network virtualization.

1.2 Software-Defined Network

Software-Defined Network (SDN) decouples the control plane (i.e., managing how to operate the traffic) from the data plane (i.e., forwarding the traffic flow based on decisions of control

plane) [10]. As a result, the SDN control plane is able to directly operate over the state in the network elements of the data plane (e.g., routers, switches or other middleboxes) via a well-defined Application Programming Interface (API) (e.g., OpenFlow [11]).

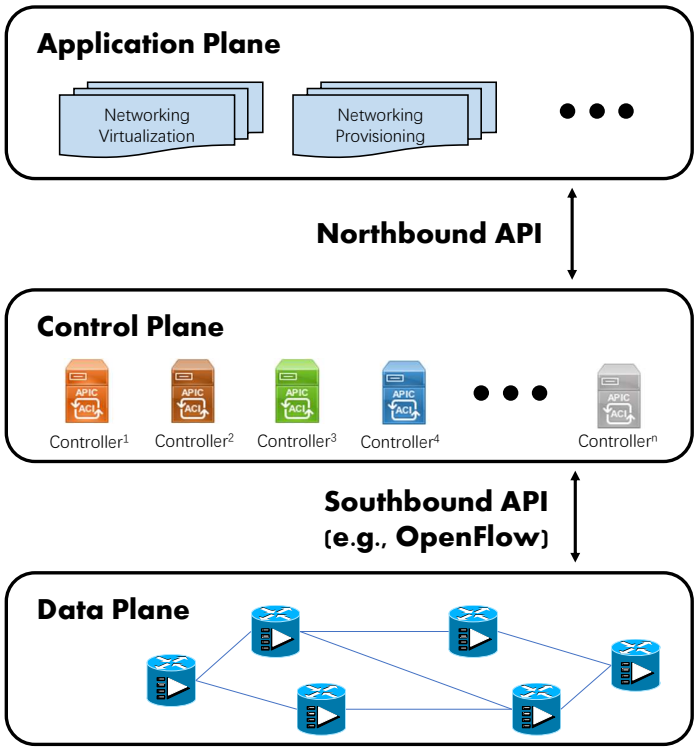


Figure 1.2: Architecture of SDN.

Fig. 1.2 introduces the basic architecture of SDN, which includes three planes and communication interfaces: i) application plane, ii) control plane, and iii) data plane. The application plane runs applications over the network infrastructure, and allows to perform modifications regarding the requirements. The control plane provides logic control policies (e.g., routing schemes) to manage the collected information from the switches of the data plane. It is worthy of mentioning that, the control plane has the global view of the networking conditions (e.g., resource availability). In the data plane, the physical devices are

responsible to forward data when the programmable flow tables can be dynamically configured by the control plane. Meanwhile, the northbound API enables programmable network controllers, and the southbound API allows the communications between the control plane and data plane by using the OpenFlow protocols.

With the advancement in SDN, service providers are able to flexibly provide networking control policies. To provide agile services to the customers, Network Function Virtualization (NFV) is a complement of the SDN techniques, which implements the hardware-based network functions to software modules.

1.3 Network Function Virtualization

Traditionally, the network functions are implemented by the proprietary middle-boxes, which is expensive for the cost and inconvenient for the service delivery [12]. To agilely deliver the service while reducing the CAPEX and OPEX for service providers, Network Function Virtualization (NFV) was proposed by European Telecommunications Standards Institute (ETSI) [13].

Fig. 1.3 shows the architecture of NFV MANagement and Orchestrator (MANO). The NFV Infrastructure (NFVI) abstracts the physical resource to the virtual resource pool and will be used by the virtualized infrastructure managers for supporting the behaviours of the Virtual Network Function Manager (VNFM). With such resource, VNFM will instantiate VNF instances to achieve the certain demands requested by the Element Manager (EM). Then, the information of instantiated VNFs will be informed to the NFV Orchestrator

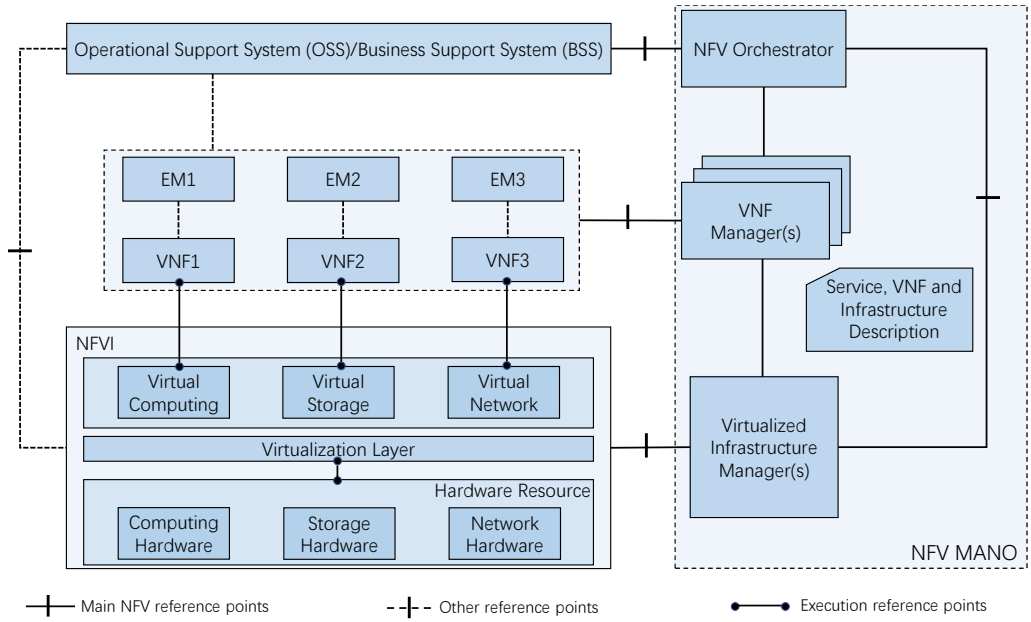


Figure 1.3: Architecture of NFV MANO.

(NFVO). As a result, the NFVO will perform the operations on the Operational Support System (OSS) or Business Support System (BSS).

With NFV, the hardware-based network functions are implemented by the software-based modules called Virtual Network Functions (VNFs) or Service Functions (SFs) [14]. SFs can be flexibly installed on or removed from the commodity servers [15]. In NFV, the NFV Service Request (NSRs) from the customer generally includes source, destination, a set of SFs and the corresponding networking resources [16]. Traditionally, to deliver a service in NFV, the service provider must concatenate the required SFs into a chain structure called Service Function Chain (SFC) and embed the composited chain onto a the Physical Network (PN), while reserving the certain networking resources [16]. The process of compositing and embedding an SFC to meet an NSR is referred to as Service Function Chaining and Embedding (SFCE) [17]. The physical path that hosts the composited SFC is called Service

Function Path (SFP) [14, 15].

1.4 Organization of This Work

With the support of the network virtualization, SDN and NFV techniques, in this dissertation, we plan to investigate how to efficiently deliver the services to the customers for future internet. In chapter 2, we investigate a problem of how to provably deliver a service as an SFC in unique function networks. In Chapter 3, we further study how to provably deliver a service as an SFC in multiple functions networks. In chapter 4, simultaneously considering the forward and backward traffic flows, we investigate a problem for effectively delivering a service as a hybrid SFC. In chapter 5, we further consider the optimization problem of how to minimize the latency of delivering a given hybrid SFC. In chapter 6, when the VNFs have certain executing order constraint, we study how to efficiently composite and embed a dependence-aware SFC. In chapter 7, we list two directions of future work. In chapter 8, we conclude our work.

CHAPTER 2
SERVICE CHAINING AND EMBEDDING IN UFNS

2.1 Motivation

Network Function Virtualization (NFV) provides communication services and replaces the physical proprietary hardware with the software-based modules called Virtual Network Functions (VNFs) or Service Functions (SFs) [18]. SFs can be deployed with the Commercial Off-The-Shelf (COTS) hardware platform or standardized high volume servers [19]. With NFV, a Network Service Request (NSR) from a customer can be a set of SFs with resource demand(s) (e.g., CPU, bandwidth) [20]. To satisfy an NSR, the service provider must chain the SFs with the SF links into a Service Function Chain (SFC), which defines the executing order of the required SFs [14]. At the same time, all the SF nodes and SF links in the SFC must be mapped onto the substrate network to form the actual forwarding path called Service Function Path (SFP) [14]. The processes of SFC composition, SF node mapping, and SF link mapping are referred to as SFC composition and Embedding (SFCE) [21]. When the executing order of the SFs or the SFC is given, the SFCE problem becomes the traditional Virtual Network Embedding (VNE) problem, which is proved as NP-hard [20].

In the literature, there exist many works towards solving the SFCE problem when the executing order of the required SFs are given or partially given [22, 23, 24, 25, 26]. In [22], the authors study SFCE in an optical network when partial executing order is given. When the SFC is given, the authors in [23] study the SFC deployment and adjustment in the online situation while considering the tradeoff between the resource consumption and

operational overhead. The authors in [26] transform the given Substrate Network (SN) into a new graph and find the optimal path to satisfy the required functions in the new graph under the condition that the substrate link provides unlimited bandwidth. In [27], we investigate how to embed the independent SFs onto a unique service function network with a 2-approximation algorithm. When verifying the approximation, we use the length of the Minimum Spanning Tree (MST) as the lower bound instead of comparing the performance of the proposed algorithm with the optimal result.

In this chapter, we study how to composite and embed an SFC onto a specific substrate network, where each substrate node only provides one unique SF. The unique service function network is practical since substrate networks like the Mobile Edge Computing system may have limited computing capacity and each edge substrate node may only perform one specialized/unique SF. We name this problem as SFC composition and Embedding in Unique service Function networks (SFCE-UF). We formulate this problem by applying the technique of Integer Linear Programming (ILP) and prove its NP-hardness. In addition, we propose an algorithm with provable approximation boundary, namely, Spanning Closed Walk based SFC composition and Embedding in unique service function networks (SCW-SFCE). Here, we highlight the differences between this chapter and the one in [27] as:

- We formulate the problem by applying the technique of Integer Linear Programming;
- We improve the approximation boundary as $2^*|1-\frac{2}{V}|$, where $|V|$ is the number of SFs in the NSR.
- In the experiment, we compare the performance of the proposed algorithm with the

optimal results acquired from ILP. We also investigate how the physical link length impacts the approximation boundary.

The rest of this chapter is organized as follows. In Section II, we provide the problem statement of the SFCE-UF. Section III provides the analysis and algorithms for SFCE-UF in unique service function networks. In Section IV, we show the experimental results. In Section V, we conclude our work.

2.2 Problem Statement

We use an undirected graph $G_S = (N, L)$ to represent the physical/Substrate Network (SN), where N represents the set of physical nodes and L is the set of physical links. Each physical node $n \in N$ provides a specific amount of computing capacity and a particular Service Function (SF) denoted as c_n and f_n, respectively. For each physical link $l \in L$, it owns a certain weight denoted as w_l, which can be delay, length and cost of this link. A Network Service Request (NSR) can be represented by a 1-tuple $NSR =< V >$, where V represents the set of SF nodes. For each SF node $v \in V$, it is equipped with a specific amount of computing demand (cd_v) and a SF demand (f_v). Since the physical node only provides one unique SF, the SF nodes mapping process is deterministic and we use $\Delta \subseteq N$ to represent the set of substrate nodes that provide the SF instances required by the NSR. We use $p_{\epsilon_i,\epsilon_j}$ as a binary value to indicate whether the shortest path connecting two physical nodes that provide the required SF instances is used to construct the SFP and $w_{p_{\epsilon_i,\epsilon_j}}$ to denote the length of the path, where $\epsilon_i \in \Delta$ denotes the specific physical node that hosts the in-service

SF instance. The optimization goal of the SFCE-UF can be described as Eq. (2.1).

$$min \sum_{\epsilon_i,\epsilon_j \in \Delta} p_{\epsilon_i,\epsilon_j} * w_{p_{\epsilon_i,\epsilon_j}} \tag{2.1}$$

subject to:

$$p_{\epsilon_i,\epsilon_j} = \begin{cases} 1, & p_{\epsilon_i,\epsilon_j}\ (\forall \epsilon_i, \epsilon_j \in \Delta) \text{ is used} \\ & \text{to construct SFP} \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

$$\sum_{\epsilon_j \in \Delta} f_{\epsilon_i,\epsilon_j} - \sum_{\epsilon_k \in \Delta} f_{\epsilon_k,\epsilon_i} = \begin{cases} 1, & \epsilon_i \text{ is the starting node} \\ 0, & \epsilon_i \text{ is an intermediate node} \\ -1, & \epsilon_i \text{ is the last node} \end{cases} \tag{2.3}$$

$$\sum_{\epsilon_j \in \Delta} f_{\epsilon_i,\epsilon_j} + \sum_{\epsilon_k \in \Delta} f_{\epsilon_k,\epsilon_i} \geq 1, \forall \epsilon_i \in \Delta \tag{2.4}$$

$$f_{\epsilon_i,\epsilon_j} \geq p_{\epsilon_i,\epsilon_j}, \forall \epsilon_i, \epsilon_j \in \Delta \tag{2.5}$$

Eq. (2.2) represents whether path $p_{\epsilon_i,\epsilon_j}$ is used to construct the SFP. Eq. (2.3) determines the position of the node ϵ_i in the constructed SFP by utilizing the flow functions, where $f_{\epsilon_i,\epsilon_j}$ represents whether a flow from ϵ_i to ϵ_j is used to transmit data. When Eq. (2.3) = 1, the physical node ϵ_i is the starting node of the constructed SFP. If Eq. (2.3) = -1, ϵ_i is the last

node of the SFP. Otherwise, ϵ_i is the intermediate node. Eq. (2.4) ensures that for each ϵ_i, there is at least one flow coming/outgoing into/from it. In Eq. (2.5), if there is a flow passing ϵ_i and ϵ_j, path $p_{\epsilon_i,\epsilon_j}$ is used to construct the SFP.

NP-hardness Proof for SFCE-UF: We prove the NP-hardness of the proposed problem by reducing the Travelling Salesman Problem (TSP) [28]. We assume in a given graph $G = (N, L)$, where each pair of nodes has a conditional link $cl_{m,n}$ with the cost as 0. This conditional link can provide the connectivity if and only if the endpoints are the starting node and last node of the constructed SFP. When each node in N provides a required SF instance, solving the SFCE-UF problem in the given graph is the same as the TSP problem, which is NP-hard. Therefore, SFCE-UF is NP-hard.

2.3 SFC composition and Embedding in Unique service Function networks

To optimize the length of the constructed SFP for SFCE-UF, we propose an approximation algorithm with the upper bound as 2, namely, Spanning Closed Walk based SFC composition and Embedding in unique service function networks (SCW-SFCE). In SCW-SFCE, we apply two techniques, Complete Graph Transfer (CGT) and Minimum Spanning Tree-based SFP Construction (MST-SC). For the first technique, CGT transfers the input graph into a complete graph, which is composed of physical nodes that equip with the required SF instances. In the second technique, the SCW-SFCE algorithm constructs the SFP by taking advantage of the MST technique.

2.3.1 Complete Graph Transfer

Fig. 2.1 shows an example of an NSR in a mesh Unique Function Network (UFN), which includes $\{f1, f4, f6\}$. In the UFN, substrate nodes A, B, C, D, E, F support the SF instances for SFs f1, f2, f3, f4, f5, f6, respectively. The physical nodes in dark are the ones to host the required SF nodes in the NSR.

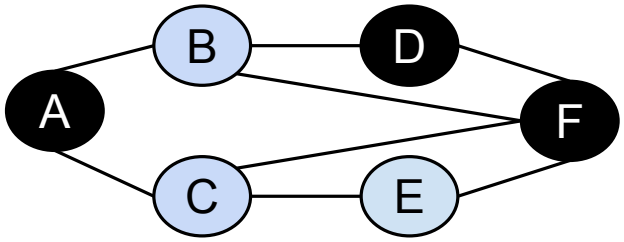


Figure 2.1: NSR = {f1, f4, f6} in a mesh UFN.

Assume that each physical link has enough bandwidth, then we introduce the technique of Complete Graph Transfer (CGT) to transfer the substrate network into a semi-complete graph by connecting all the physical nodes required by the NSR with the shortest paths.

Lemma 2.3.1. *With the assumption that each physical link can provide enough bandwidth for any SF, the optimal SFP exists in the semi-complete graph composed of physical nodes that i) provide the required SFs and ii) connect via the shortest paths or direct links.*

Proof. Here we use the technique of proof by contradiction to prove this lemma. We call the complete graph mentioned in the Lemma 2.3.1 as the SFP candidate complete graph. Assuming that the optimal SFP does not exist in the SFP candidate complete graph, then, there is either i) at least one endpoint of the optimal SFP that provides unrequired SF, or ii) at least one path between a pair of nodes in the optimal SFP that is not connected via

the shortest path. For the former case, if there is an endpoint say v, providing unrequired SF in the optimal SFP, we can get a shorter SFP by deleting v and the link connecting v to other walk components. For the latter case, if there is a path in the optimal SFP that is not connected via the shortest path, this is contradict to the fact that all pair of nodes are connected with the shortest paths. Therefore, the optimal SFP in a given mesh network exists in the SFP candidate complete graph □

2.3.2 Minimum Spanning Tree-based SFP Construction

In this subsection, we propose the Minimum Spanning Tree-based SFP Construction (MST-SC) technique to facilitate the construction of the SFP in the SFP candidate complete graph. Since every node in the SFP candidate complete graph provides one unique required SF instance, each node in the SFP candidate complete graph will be visited at least once. Accordingly, we have the following lemma to support the understanding of the MST-SC technique and the 2-approximation boundary for the algorithm.

Lemma 2.3.2. *The shortest spanning closed walk of a tree visits each link exactly twice.*

Proof. In a spanning walk, every node needs to be visited. Since the fact that there is only one path connecting any pair of nodes in a tree network, every link has to participate in the spanning walk of the tree. In a closed walk, each link has to be used at least twice, one for going forward and one for going back. Thus, each substrate link in the tree network topology has to be traversed at least twice to finish a spanning closed walk on the tree. Therefore, the shortest spanning closed walk visits each link exactly twice. □

Algorithm 1 Spanning Closed Walk based SFC composition and Embedding in unique service function networks (SCW-SFCE)

1: **Input:** G_S, NSR;
2: **Output:** SFP;
3: Set SFP as an empty list;
4: Deploy the SF nodes from NSR onto the corresponding SF instances in G_S and Transfer G_S into SFP candidate complete graph;
5: Find the Minimum Spanning Tree (MST) in the SFP candidate complete graph;
6: Form the shortest closed spanning walk on the MST and generate the Spanning Closed Walk List (SCWL);
7: Find and remove the longest leaf-to-leaf path sequence in the SCWL;
8: Return the remaining component of the SCWL as SFP;

Based on Lemma 2.3.2, we propose the MST-SC technique as shown in Algorithm 1. Line 3-4 initializes the SFP list, embeds the required SFs to their corresponding physical nodes, and transfers the graph into the SFP candidate complete graph. In Line 5-7, the algorithm constructs an SFP from the SFP candidate complete graph.

Theorem 2.3.3. *When given an undirected graph G_S and a request NSR, the SCW-SFCE algorithm can find the SFP, whose length is at most twice as the one from the optimal SFP.*

Proof. We start the proof with an NSR, which only includes two SF nodes. As only two SF nodes are included in the NSR, there are only two physical nodes required to host the two SF instances. As a result, the SFP is a path connecting these two physical nodes. Since these two physical nodes are connected with the shortest path, this constructed SFP is optimal.

Then, we prove the 2-approximation boundary for the SFP constructed by an NSR with more than 2 SF nodes. We denote the length of optimal SFP as SFP_{opt}, the constructed SFP as $SFP_{\text{SCW-SFCE}}$ and the length of the MST as $Length_{MST}$. SCW represents the length of the shortest spanning closed walk of the constructed MST, while $Path_{leaf}$ is the length of

the longest leaf-to-leaf path existing in the shortest spanning closed walk. Based on Lemma 2.3.1, the optimal SFP exists in the constructed SFP candidate complete graph. Hence, the $Length_{MST}$ of the SFP candidate complete graph is smaller or equal to the SFP_{opt} as shown in Eq. (2.6).

$$Length_{MST} \leq SFP_{opt} \tag{2.6}$$

Based on Lemma 2.3.2, the MST is the minimum connected component for a given graph and the constructed shortest spanning closed walk (SCW) has a length twice of the $Length_{MST}$ as shown in Eq. (2.7).

$$SCW = 2 * Length_{MST} \tag{2.7}$$

Since the SCW-SFCE algorithm removes the longest leaf-to-leaf path from the shortest spanning closed walk to create the SFP, Eq. (2.8) holds.

$$SFP_{\text{SCW-SFCE}} = SCW - Path_{leaf} \tag{2.8}$$

As the $Path_{leaf}$ is the longest leaf-to-leaf path in the MST, which at least contains 2-hops (since the NSR contains more than 2 SFs), it is longer than twice of the average length of links in the constructed SFP. Thus, as shown in Eq. (2.9), $Path_{leaf}$ is twice longer than the average length of links in the constructed SFP, where $|V|$ represents the number of SF nodes in the NSR.

$$Path_{leaf} \geq \frac{2}{|V|-1} SFP_{\text{SCW-SFCE}} \tag{2.9}$$

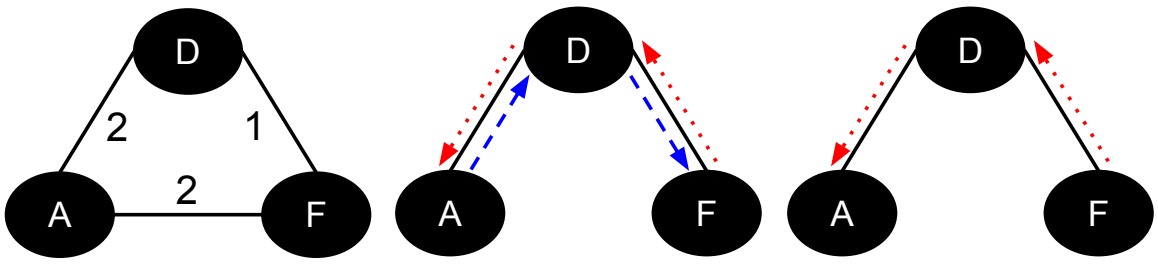
When combining Eq. (2.6)-(2.8), we have Eq. (2.10), which shows that our algorithm finds

the SFP whose length is at most twice of the length of the optimal SFP.

$$SFP_{\text{SCW-SFCE}} \leq 2(1 - \frac{2}{|V|+1})SFP_{opt} \tag{2.10}$$

□

We then use Fig. 2.1 to explain how the proposed SCW-SFCE algorithm works in Fig. 2.2. First, the algorithm transfers the graph into the SFP candidate complete graph, which is shown in Fig. 2.2a, where the number beside the link represents the shortest distance between each two physical nodes required by the NSR. Then, taking Fig. 2.2a as the input, the MST-SC technique creates the MST, and forms the shortest spanning closed walk starting at node F as shown in Fig. 2.2b, where the red-dotted and blue-dashed arrows represent the forwarding and backward paths, respectively. Next, the algorithm finds the longest leaf-to-leaf path segment in the formed SCW, which is $A \rightarrow D \rightarrow F$, and deletes it from the SCW to form the SFP, which is shown in the Fig. 2.2c. As one can see that, the final SFP has the same length as the optimal SFP, which matches Theorem 2.3.3.



(a) SFP Candidate Complete Graph of Fig. 2.1 (b) Spanning Closed Walk (c) Service Function Path

Figure 2.2: Example of the SCW-SFCE Algorithm.

As for the time complexity of the SCW-SFCE algorithm, in the worst case, the first

step (i.e., to create the SFP candidate complete graph) has the time complexity of $|V^2L + V^2NlogN|$. The time complexity of the second step (i.e., finding the MST) is $|L + NlogN|$. Thus, the overall time complexity of the SCW-SFCE algorithm is $|V^2L + V^2NlogN|$.

2.4 Numerical Results and Analysis

In this section, we analyze the performance of the proposed 2-approximation algorithm (i.e., SCW-SFCE). We conduct extensive simulations using the 14-node NSFNET and 24-node USNET as the substrate networks. Unless otherwise specified, we randomly set half of the physical links having the length of 1 and the other links having the length of x, where $x \in 1, 5, 9$. In the NSFNET, we set the requested number of SFs in an NSR in the range of [3,13], while it is in the range of [3, 24] in the USNET. For each series of experiments, we randomly generate more than 100 thousand network service requests and obtain the average results as shown in the following figures. When the substrate network is NSFNET, we apply the Integer Linear Programming (ILP) mentioned above to obtain the optimal results. When the substrate network is USNET, the ILP model is intractable and we compare the performance of the SCW-SFCE algorithm with the Nearest-Neighbour (NN) algorithm. Note that, the NN algorithm has the provable approximation boundary as $|\frac{1}{2}\lceil N \rceil + \frac{1}{2}|$, where N represents the number of nodes in the SFP complete candidate graph [29].

2.4.1 SCW-SFCE vs. ILP in NSFNET

Fig. 2.3 shows the performance of ILP, NN and SCW-SFCE algorithms in the small NSFNET. In Fig. 2.3, the curve of "Average Approximation" is the ratio between the

results of the proposed SCW-SFCE algorithm and the optimal ILP. As one can see that, with the increasing number of SFs in each NSR, all algorithms require more substrate links to construct a longer SFP. The ratio between the proposed SCW-SFCE and the optimal ILP is always less than 2, which demonstrates that the proposed SCW-SFCE does achieve the 2-approximation performance. As one can see that, when the number of required SFs in the request is small, all algorithms have similar performance. When increasing the number of required SFs, ILP has the best performance, while the proposed SFW-SFCE outperforms the NN algorithm due to the spanning closed walk technique.

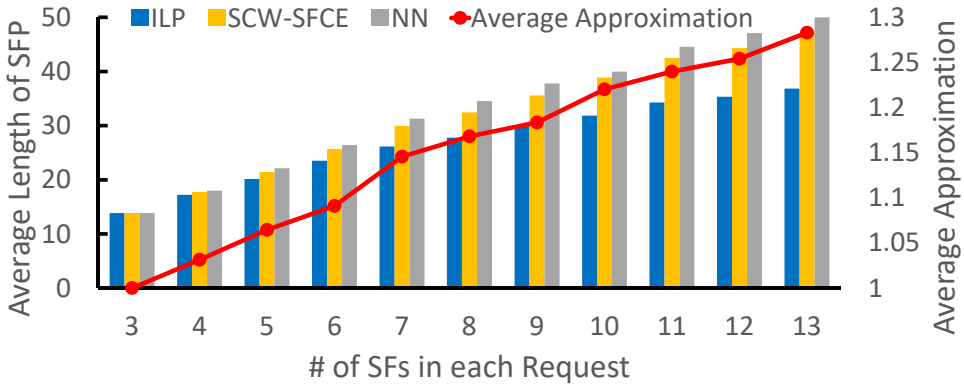


Figure 2.3: ILP vs. SCW-SFCE in NSFNET.

2.4.2 SCW-SFCE Performance Evaluation in USNET

Eq. (2.10) indicates that the length of the Minimum Spanning Tree (MST) is the lower bound of the optimal results. Hence, as shown in Fig. 2.4, we calculate the "Average Approximation" as the ratio between the proposed SCW-SFCE and the MST length. In Fig. 2.4, the red curve (i.e., "1—1") represents the case when the physical link weight are all 1; the blue curve (i.e., "1—5") represents the case when half of the links have a weight

of 1, and the other half links have a weight of 5; and the yellow curve ("1—9") represents the case when half of the links have a weight of 1 and the other half links have a weight of 9. From the results in Fig. 2.4, one can see that in every case, the average approximation value is less than 2.

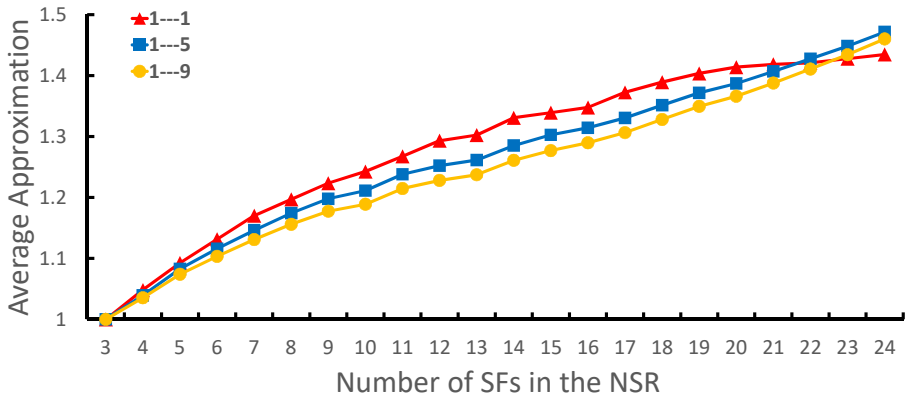


Figure 2.4: The Average Approximation of SCW-SFCE.

2.4.3 SCW-SFCE vs Nearest Neighbour Algorithm in USNET

In Fig. 2.5, the red bar represents the length of the SFP constructed by the SCW-SFCE algorithm, and the blue bar is from the NN algorithm. The grey dashed curve, named as "Difference" and evaluated by the y-axis on the right, represents the difference between the results of SCW-SFCE and NN. As one can see that, when increasing the number of SFs in each NSR, the SCW-SFCE algorithm always outperforms the NN algorithm. However, the difference increases before the number of SFs is less than 10, but it decreases when the number of SFs in each NSR is lager than 10. This is because when the number of SFs is small or large (compared to the number of nodes in the SN), both SCW-SFCE and NN has the similar construction of the SFP due to the MST based SFP construction technique, which

guarantees the performance of the constructed SFP.

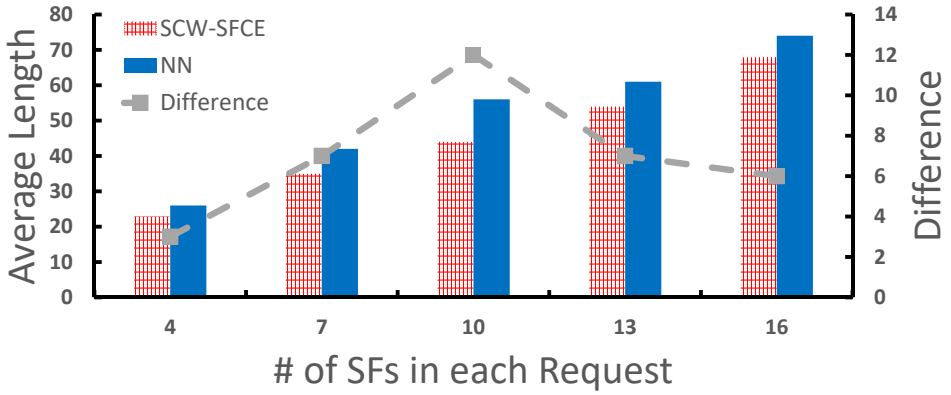


Figure 2.5: SCW-SFCE vs. NN in USNET.

2.5 Summary

In this chapter, we have studied SFC composition and Embedding in the Unique service Function networks (SFCE-UF) in various substrate network scenarios. We have formulated the SFCE-UF problem by applying the technique of Integer Linear Programming (ILP) and proved the NP-hardness of the SFCE-UF. To solve the SFCE-UF problem in a substrate mesh network that each node only offers one unique SF, we have proposed a 2-approximation algorithm, called Spanning Closed Walk based SFC composition and Embedding in unique service function networks (SCW-SFCE). In the next chapter, we will further investigate how to provably deliver the service as an SFC in multiple functions networks.

CHAPTER 3
SERVICE CHAINING AND EMBEDDING IN MFNS

3.1 Motivation

With the advancements in the Internet of things (IoT), billions of devices (e.g., smart phones, wearable devices and sensors) are expected to be connected to the Internet [30, 31]. Due to the limited computing capacity and battery life, IoT devices often face a challenge in dealing with computation-intensive tasks [2]. To mitigate such a challenge, cloud systems (e.g., datacenter) and multiaccess edge computing systems give rise to convenient accesses of abundant computing resource for IoT-based services [32]. In both cloud and multiaccess edge computing systems, many in-network hardware middleboxes such as Firewalls (FWs), Deep Packet Inspection (DPI) and WAN optimizers are employed to ensure the security and performance [33]. On the one hand, these hardware appliances bring benefits for securely and efficiently delivering services. On the other hand, these dedicated middleboxes are expensive and require great efforts for maintenance and management [20].

To reduce the capital expenditures and the operating expense, network function virtualization (NFV) is introduced [13]. NFV transforms the implementation of proprietary hardware appliances (or middleboxes) to software-based modules called virtual network functions or service functions (SFs) [14]. SFs can be flexibly installed on or removed from physical nodes (e.g., edge/cloud servers) [34]. In NFV, the network service request (NSR) consists of a set of SFs with corresponding resource demands (e.g., virtual machine, bandwidth) [15]. To accommodate a network service request, service providers can concatenate the requested

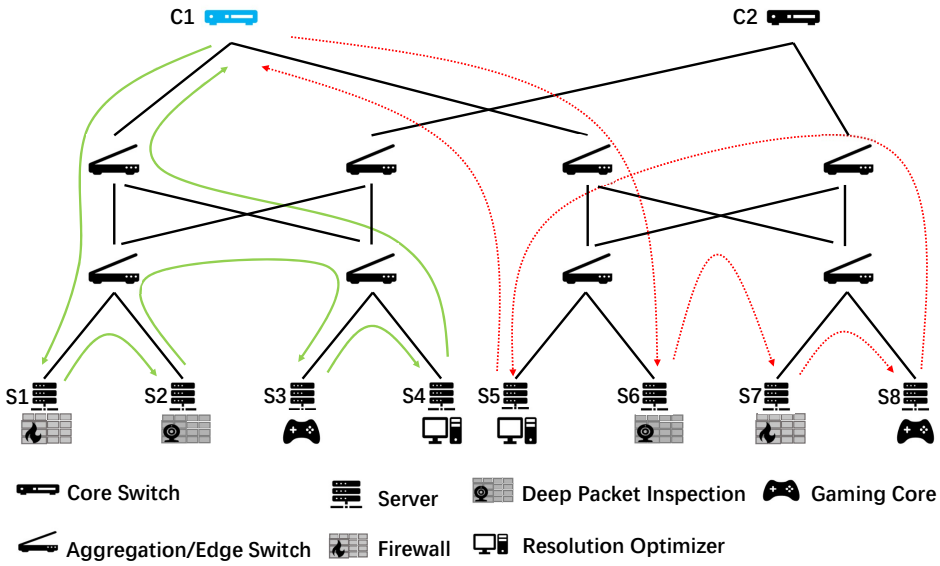


Figure 3.1: Delivering cloud gaming service as an SFC in the datacenter network.

SFs into a linear structure called a service function chain (SFC) and embed it onto the physical network (PN) with enough reserved network resources [35]. If the executing order of the requested SFs is given by the customer, the service provider can construct only one unique SFC with a specific virtual linear topology. When the executing order of the requested SFs is not specified or partially specified, there exist multiple possible combinations to construct the SFC. The process of constructing an SFC and embedding it onto a shared PN is referred to as service function chaining and embedding (SFCE) [15]. The physical path that accommodates the constructed SFC in the PN is called a service function path (SFP).

Fig. 3.1 shows a cloud gaming application running in the datacenter network with an ultralow latency requirement. Inside the datacenter, the customer's traffic comes from the blue core switch (i.e., C1 in Fig. 3.1) requiring the FW and DPI before reaching the gaming core (GC) and the resolution optimizer (RO) afterwards. The green-solid and red-dotted arrows

are two feasible SFPs that satisfy the demands. The green-solid SFP passes the SFs in the order of C1→S1(FW)→S2(DPI)→S3(GC)→S4(RO)→C1 with $3 + 2 + 4 + 2 + 3 = 14$ hops. The red-dotted SFP goes through the SFs as C1→S6(DPI)→S7(FW)→S8(GC)→S5(RO)→C1 and requires $3 + 4 + 2 + 4 + 3 = 16$ hops. Assuming each SF needs the same processing latency, the green-solid SFP outperforms the red-dotted SFP in terms of hop numbers (or transmission latency cost). Hence, it is important for the service provider to optimize the SFCE process and minimize the cost when constructing the SFP [30].

There is some existing work focusing on the cost minimization of the SFCE process [36, 37, 38, 39, 40, 41, 42, 43], and only a few works study the performance guarantee [17, 26, 44, 45, 46, 47]. Specifically, when the executing order of an SFC is given, the authors of [26], [45] and [46] proposed schemes to optimally embed the SFC onto the PN with unlimited bandwidth. Without a given SFC's executing order, the authors of [17] and [47] proposed graph theory-based algorithms to simultaneously construct and embed an SFC with a 2-approximation guarantee under the assumption that each physical node only provides one unique SF. However, the above research work devises the performance approximation guarantees at the expenditure of assumptions, which may not be feasible for practical scenarios. In this paper, for the first time, we provide an approach that comprehensively investigates how to jointly chain and embed an NSR with provable bounds under practical networking conditions. We mathematically define the problem of minimum cost service function chaining and embedding (MC-SFCE). Here, the cost can be identified as the bandwidth, latency or expenditure. We propose a novel algorithm called COst Factor-based SFCE Optimization

with ShortCut (COFO-SC), which is proved to be tightly bounded. In the PNs, where each physical node only provides one unique SF, COFO-SC guarantees a 2-approximation bound. When each physical node provides multiple SFs, COFO-SF can accommodate an NSR with a logarithm-approximation.

The rest of this chapter is organized as follows. In Section II, we mathematically formulate the MC-SFCE problem. In Section III, IV and V, we illustrate the proposed algorithm and analyze its performance in different networking conditions. Section VI demonstrates the experimental results and analysis. We conclude this chapter in Section VII.

3.2 Minimum Cost Service Function Chaining and Embedding (MC-SFCE)

3.2.1 Physical Network Model

The physical network (PN) is denoted by an undirected graph $G = (N, L, F)$, where N represents the set of physical nodes (e.g., edge/cloud server, IoT devices), L is the set of physical links, and F denotes a set of commonly used SFs (e.g., Firewall, deep packet inspection, WAN optimizer). Since installing particular SFs may require significant amounts of resources, licenses or copyrights, in our model, each physical node $n \in N$ can only instantiate a specific set of SF instances $\mathbb{F}_n \subset F$ with a certain amount of computing capacity c_n. Each physical link $l \in L$ has a particular amount of bandwidth bw_l and a certain cost $cost_l$. Here, the cost of a link can be one of the following attributes: bandwidth, latency or expenditure. For simplicity, a physical link l can also be represented by $l_{m,n}$, where $m, n \in N$ are its endpoints. A physical path from m to n is denoted by $path_{m,n}$, where $cost_{path_{m,n}}$ and

$bw_{path_{m,n}}$ represent its cost and bottleneck bandwidth, respectively.

In this work, we investigate two types of PNs: (1) a unique function physical network (UFPN) and (2) a multiple function physical network (MFPN). In UFPN, each physical node only provides one unique SF instance. That is, any two different physical nodes provide different SF instances (i.e., $\mathbb{F}_m \cap \mathbb{F}_n = \emptyset, \forall m, n \in N$). This scenario is practical in the subdomains of a large network [48] or resource-constrained systems [7]. In contrast, each physical node in MFPN is capable of providing various SF instances, while different physical nodes may offer the same SF instance(s) (i.e., $\mathbb{F}_m \cap \mathbb{F}_n$ may not be an empty set).

3.2.2 Network Service Request (NSR)

A network service request (NSR) can be represented as a four-tuple NSR $=< s, d, V, |BW| >$, where s and d represent source and destination nodes, V denotes the set of requested virtual nodes (or SF nodes), and $|BW|$ is the amount of bandwidth demand. For each virtual node $v \in V$, it requires a specific type of SF instance $f_v \in F$, a certain amount of computing demand c_v, and a deployment cost $cost_v$ (e.g., CPU, expenditures, or processing latency). Notably, s and d are regarded as the source and destination, respectively, for both SFP and SFC.

3.2.3 Minimum Cost Service Function Chaining and Embedding (MC-SFCE)

The optimization problem of MC-SFCE is defined as follows: given a PN and an NSR, determine how to accommodate the NSR onto the PN such that (1) the following constraints are satisfied and (2) the cost of the constructed SFP is minimized. Here, the cost of the

constructed SFP means the summation of the links' cost in the SFP and the SFs' deployment cost. The objective function is shown in Eq. (3.1), where $\mu^{u,v}_{path_{m,n}}$ represents the cost of $path_{m,n}$ supporting the traffic from u to v. In Eq. (3.1), the first summation component is the cost for mapping the SFC virtual links, and the second denotes the cost of mapping virtual nodes. Table I describes the primary notations for the variables that we used.

$$min \sum_{v \in V} \sum_{u \in V} \sum_{n \in N} \sum_{m \in N} \mu_{path^{u,v}_{m,n}} + \sum_{v \in V} cost_v \tag{3.1}$$

Table 3.1: Notation Table

Notation	Meaning
N	Set of physical nodes
V	Set of virtual nodes
V_{sub}	Subset of V
m, n	Physical nodes $m, n \in N$
u, v	Virtual nodes $u, v \in V$
$\lvert BW \rvert$	Amount of bandwidth demand
M^v_n	=1 when v is mapped onto n; 0 otherwise
Δ^v_n	=1 when n provides SF instance for v; 0 otherwise
c_v	Computing demand of v
c_n	Computing capacity of n
$cost_v$	Deployment cost of v
$path^{u,v}_{m,n}$	=1 when $path_{m,n}$ supports the traffic from u to v; 0 otherwise
$bw_{path_{m,n}}$	Bottleneck bandwidth of $path_{m,n}$
$cost_{path_{m,n}}$	Cost of $path_{m,n}$
$\mu_{path^{u,v}_{m,n}}$	Cost of $path_{m,n}$ supporting the traffic from u to v

SF node mapping constraints: Eq. (3.2) represents whether a virtual node v is mapped onto physical node n or not, while Eq. (3.3) identifies whether n is capable of instantiating SF instance for v or not. Eq. (3.4) and Eq. (3.5) ensure that if v is mapped onto n, then n must be capable of providing the corresponding SF instance and enough computing capacity. Eq.

(3.6) specifies that each virtual node v will be mapped onto one physical node n.

$$M_n^v = \begin{cases} 1, & \text{Virtual node } v \in V \text{ is mapped} \\ & \text{onto physical node } n \in N. \\ 0, & \text{Otherwise.} \end{cases} \tag{3.2}$$

$$\Delta_n^v = \begin{cases} 1, & \text{Physical node } n \in N \text{ provides} \\ & \text{SF instance for } v. \\ 0, & \text{Otherwise.} \end{cases} \tag{3.3}$$

$$M_n^v \leq \Delta_n^v, \forall n \in N, \forall v \in V \tag{3.4}$$

$$\sum_{v \in V} M_n^v * c_v \leq c_n, \forall n \in N \tag{3.5}$$

$$\sum_{n \in N} M_n^v = 1, \forall v \in V \tag{3.6}$$

Traffic constraints: We use $path_{m,n}^{u,v}$ to denote whether $path_{m,n}$ supports the traffic from virtual node u to v as shown in Eq. (3.7). Eq. (3.8) and Eq. (3.9) ensure that (1) if $path_{m,n}$ is used to support traffic from u to v, then M_m^u and M_n^v must both be 1, and (2) the bottleneck bandwidth of $path_{m,n}$ must not be less than $|BW|$. Eq. (3.10) and Eq. (3.11) require that (1) there must be a physical path starting from each virtual node (including source), and

(2) there must be a physical path ending at each virtual node (including destination). Eq. (3.12) guarantees that there is no circle in the formulated SFP, where V_{sub} is a subset of V. Eq. (3.13) denotes the calculation of $\mu_{path_{m,n}^{u,v}}$.

$$path_{m,n}^{u,v} = \begin{cases} 1, & \text{Path from } n \text{ to } m \text{ supports} \\ & \text{traffic from SF node } u \text{ to } v. \\ 0, & \text{Otherwise.} \end{cases} \tag{3.7}$$

$$path_{m,n}^{u,v} \leq \frac{M_n^v + M_m^u}{2}, \forall m, n \in N, \forall u, v \in V, u \neq v \tag{3.8}$$

$$\sum_{u \in V} \sum_{v \in V} \sum_{m \in N} \sum_{n \in N} path_{m,n}^{u,v} * |BW| \leq bw_{path_{m,n}} \tag{3.9}$$

$$\sum_{v \in V, u \neq v} \sum_{m \in N} \sum_{n \in N} path_{m,n}^{u,v} = 1, \forall u \in \{V \cup s\} \tag{3.10}$$

$$\sum_{u \in V, u \neq v} \sum_{m \in N} \sum_{n \in N} path_{m,n}^{u,v} = 1, \forall v \in \{V \cup d\} \tag{3.11}$$

$$\sum_{u \in V_{sub}} \sum_{v \in V_{sub}} \sum_{m \in N} \sum_{n \in N} path_{m,n}^{u,v} \leq |V_{sub}| - 1, \tag{3.12}$$

$$\forall V_{sub} \subsetneqq V, V_{sub} \neq \emptyset, u \neq v$$

$$\mu_{path_{m,n}^{u,v}} = path_{m,n}^{u,v} * cost_{path_{m,n}^{u,v}}, \tag{3.13}$$

$$\forall u, v \in V, u \neq v, \forall m, n \in N$$

3.3 Complexity Analysis of Minimum Cost Service Function Chaining and Embedding (MC-SFCE)

In this section, we analyze the complexity of the MC-SFCE process in unique function tree networks (UFTNs), unique function mesh networks (UFMNs), multiple function tree networks (MFTNs) and multiple function mesh networks (MFMNs). Note that, since the second summation component in Eq. (1) will be fixed in our assumptions, we illustrate how the first summation component will be optimized.

3.3.1 MC-SFCE in Unique Function Tree Network

We start the analysis with a typical unique function tree network (UFTN), whereas the network topology is a tree and each physical node only provides one unique SF. Note that, even though a physical node provides only one type of SF in UFTN, multiple instances of such an SF can be hosted by this physical node to support multiple NSRs. Fig. 3.2 shows an example of a UFTN, where physical nodes A, B, C, D, E, and F can offer SF instances f1, f2, f3, f4, f5, and f6, respectively. We assume that an NSR starts and ends at A and requires four SFs, f1, f2, f5, and f6. Hence, the dark nodes in Fig. 3.2 will host the required SFs.

From our observation, optimally delivering the above network service (as an SFC) needs

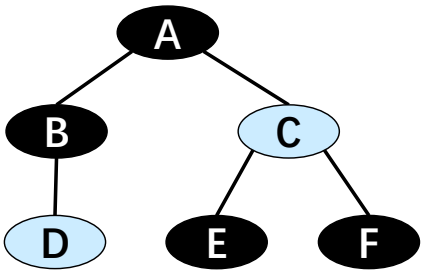


Figure 3.2: Unique function tree.

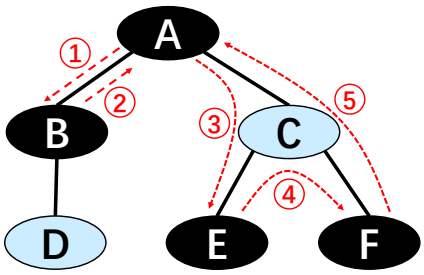


Figure 3.3: SFP for the request.

to find a trace structure that visits all required SFs with the least length[1]. In fact, such a structure is the shortest circle including all dark nodes in Fig. 3.2. In graph theory, a circle can be defined as a nonempty walk where the only repeated nodes are the first and last ones [49]. To form such a shortest circle, one needs to first determine its trace (i.e., the set of nodes and links that will be visited from source to destination). To satisfy the NSR, creating such a trace should take the following **properties** into consideration: (1) all required SFs need to be included in the trace, and (2) the length of the trace is the shortest. As there is only one path connecting any pair of nodes in a tree topology [49], the trace of such a shortest circle is also a tree. In other words, to optimally deliver the network service here, one needs to determine a subtree of the given UFTN, which includes all required SFs (matching the above property (1)), no leaf of this subtree hosts the unnecessary SF (matching the above property (2)). As there is only one path connecting any pair of nodes in a tree topology, such a subtree can be formulated by repeatedly deleting the leaf node that does not host the required SF in the given UFTN. We call the subtree in which every leaf provides one required SF as a basic subtree (BST). Then, to construct the shortest circle in BST, one can follow the spanning closed walk creation methodology in [47] to form

[1]We use the "length" and "distance" interchangeably in this work to represent the cost of a link or path.

the shortest spanning closed walk starting from the source node. Fig. 3.3 demonstrates an SFP created by applying the above methods, where the number beside each red-dotted arrow represents the visiting order. The formulated SFC is s→f1→f2→f5→f6→d along with the SFP A→B→A→C→E→C→F→C→A. Note that, when the destination node is different from the source node, the optimal SFP can be formed by deleting the shortest path connecting the source and destination nodes in the shortest circle.

As the processes of the subtree creation and the spanning closed walk construction can both be finished within polynomial time, the following Lemma holds.

Lemma 3.3.1. *Given an NSR and a UFTN, an optimal SFP can be constructed in polynomial time.*

As the shortest spanning closed walk on a tree is twice the length of the tree [47], Lemma 3.3.2 holds.

Lemma 3.3.2. *When source and destination nodes are same, the cost of a constructed SFP equals twice the cost of the BST.*

3.3.2 MC-SFCE in a Multiple Function Tree Network

In a multiple function tree network (MFTN), each physical node is capable of instantiating a set of SF instances constrained by its computing capacity. Similarly, to optimize the SFCE process in MFTNs, one needs to find a set of physical nodes that support all required SFs, while the BST visiting those physical nodes has the least length. However, unlike the case with UFTN, the set of physical nodes that will host the required SF nodes is not unique in

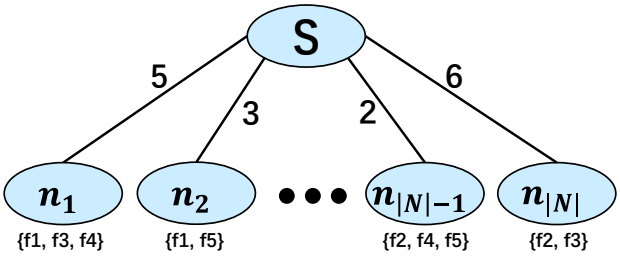


Figure 3.4: MFTN with the star topology; $|N|$ represents the size of the network.

the MFTN. Hence, determining how to properly embed SF nodes in MFTNs would have a large impact on the first summation component in Eq. (3.1).

Theorem 3.3.3. *Given an NSR and an MFTN, constructing an SFP with the minimum cost is NP-hard.*

Proof. We prove the NP-hardness of MC-SFCE in MFTNs via the reduction from the minimum weighted set cover problem [50]. Given a universe and a collection of nonempty subsets with diverse weights, the minimum weighted set cover problem tries to find a collection of subsets with minimum weights such that the elements' union of those subsets is the universe. We consider a special MFTN with a star topology, where all nodes directly connect with the source node as shown in Fig. 3.4. Each leaf node is capable of hosting a set of SF instances, and each link has a specific amount of cost. According to Lemma 3.3.2, given an NSR, finding the optimal SFP in such a topology equals searching a set of leaf nodes that include all required SFs with the minimum cost. We can reduce MC-SFCE in a multiple function star topology to the minimum weighted set cover problem as following: (1) the required SFs in the NSR are the universe in the weighted set cover problem; (2) each leaf node in the star topology represents a subset of the universe; (3) the selected leaf nodes are the selected

subsets; and (4) the cost of the SFP is the sum of the weights of the selected subsets in the weighted set cover problem. Then, MC-SFCE in MFTNs with the star topology is, in fact, equivalent to the minimum weighted set cover problem. Since the minimum weighted set cover is a well-known NP-hard problem [50], the optimization problem of MC-SFCE in MFTNs is also NP-hard. □

3.3.3 MC-SFCE in Mesh Networks

For MC-SFCE in mesh networks, we first examine the unique function mesh network (UFMN).

Theorem 3.3.4. *Given an NSR and a UFMN, constructing an SFP with the minimum cost is NP-hard.*

Proof. We prove the NP-hardness of MC-SFCE in UFMNs via the reduction from the traveling salesman path problem (TSPP) [28]. The TSPP tries to find the shortest path visiting each node in a graph with the minimum distance sum. As each node only provides one unique SF instance, MC-SFCE in UFMN tries to create a path (i.e., SFP) visiting each physical node that provides the required SF instance and requires the minimum path cost. When each physical node provides one required SF, MC-SFCE in UFPN is equivalent to the TSPP, which is NP-hard [28]. □

From Theorem 3.3.3, MC-SFCE in MFTN is NP-hard. Since the tree topology is the special case of the generic mesh network, MC-SFCE in multiple function mesh networks (MFMNs) is also NP-hard.

Lemma 3.3.5. *Given an NSR and an MFMN, constructing an SFP with minimum cost is NP-hard.*

As one can see, MC-SFCE in mesh physical networks is NP-hard. To facilitate the optimization of MC-SFCE in a given physical network, we propose a novel algorithm called COst Factor-based SFCE Optimization with ShortCut (COFO-SC) and analyze its performance bounds in the following sections.

3.4 COst Factor-based SFCE Optimization with ShortCut (COFO-SC)

In this section, based on the analysis above, we first propose novel cost factor (CF) and shortcut (SC) techniques to facilitate MC-SFCE optimization. Then, we combine these two techniques to propose our COst Factor-based SFCE Optimization with ShortCut (COFO-SC) algorithm.

3.4.1 COst Factor-based SFCE Optimization (COFO)

As shown in Eq. (3.1), the cost of constructing an SFP depends on two factors: (1) the number of physical nodes that host the required SF instances and (2) the distances (cost) among these physical nodes. From the analysis in previous section, these two factors need to be jointly taken into consideration to optimize MC-SFCE. For instance, when greedily reducing the number of physical nodes participating the SFP, the selected physical nodes may be farther away from each other, thus increasing the link cost (i.e., the first summation component in Eq. (3.1)). Likewise, when only greedily considering the distance among physical nodes, more physical nodes may be selected to construct the SFP, resulting in more

cost in connecting these physical nodes. Accordingly, we propose the cost factor (CF) to indicate the importance of a physical node for reducing the cost during the process of jointly constructing and embedding an SFC.

Cost Factor: Based on the discussion above, a proper physical node that will help reduce the cost of the constructed SFP should (1) provide as many required SF instances as possible (i.e., to minimize the number of physical nodes participating the construction of SFP), and (2) be as "near" as possible to the physical nodes that have been selected. Eq. (3.14) illustrates the calculation of CF value of a physical node n.

$$CF_n = \frac{|\sigma_n|}{|Dis_{(T,n)}|} \tag{3.14}$$

In Eq. (3.14), $|\sigma_n|$ represents the number of required SFs that can be instantiated by physical node n constrained by its computing capacity, while $|Dis_{(T,n)}|$ is the "shortest distance" (minimum cost) between n and the connected component T that is formulated by the selected node(s). Note that $|\sigma_n|$ only counts the number of required SFs that have not been instantiated but can be provided by node n. The connected component T is the minimum spanning tree visiting all selected nodes. Initially, T only includes the source and destination nodes.

To optimize MC-SFCE, we first propose the COst Factor-based SFCE Optimization (COFO), as shown in Algorithm 2. Initially, COFO creates T including s and d (Line 3) to represent the set of selected physical nodes that host the satisfied SFs and Ω to represent the set of unsatisfied SFs. In Lines 4-12, the algorithm repeatedly updates the CF value for each physical node, selects the node x with the highest CF value, and merges the shortest path

Algorithm 2 COst Factor-based SFCE Optimization (COFO)

1: **Input:** G, NSR;
2: **Output:** SFC, SFP;
3: Instantiate $T = \{s, d\}$ and $\Omega = V$;
4: **while** $\Omega \neq \emptyset$ **do**
5: **if** T only includes s and d **then**
6: Update CF values of all nodes by Eq. (3.15);
7: **else** Update CF values of all nodes by Eq. (3.14);
8: **end if**
9: Pick node x with highest CF value;
10: Connect x to T via the bandwidth-aware shortest path $path_{x,T}$, and $T = T \cup path_{x,T}$;
11: Instantiate σ_x at x, and $\Omega = \Omega - \sigma_x$;
12: **end while**
13: Create the shortest spanning closed walk according to T;
14: Form the SFP by deleting the shortest path from s to d in the formed shortest spanning closed walk;
15: Record the corresponding SFC of the constructed SFP;
return SFC, SFP;

$path_{x,T}$ into T, while deleting the instantiated SFs (i.e., σ_x) at x from Ω until all required SFs are satisfied. Note that, if T only includes source and destination nodes, the algorithm applies Eq. (3.15) to calculate the CF value for each physical node. This is because T is not yet a connected component. Thus, the initial CF value is calculated by the average CF value from node n to both source and destination nodes. Otherwise, the CF value is calculated by Eq. (3.14).

$$CF_n^{\text{initial}} = \frac{\frac{|\sigma_n|}{Dis(s,n)} + \frac{|\sigma_n|}{Dis(d,n)}}{2} \tag{3.15}$$

Based on the steps above, T is, in fact, a spanning tree visiting source node, destination node, and all selected nodes. According to Lemma 3.3.2, the SFP can be created by applying the shortest spanning closed walk technique from [47] on T and deleting the shortest path between the source and destination nodes in the walk (Line 14). Finally, the algorithm will

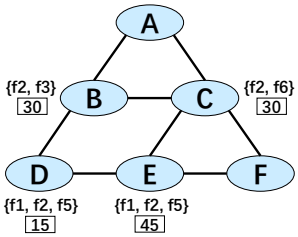


Figure 3.5: An example of MFPN.

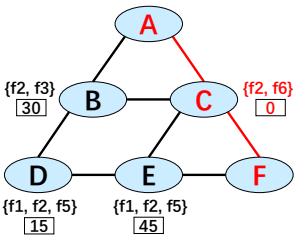


Figure 3.6: 1^{st} iteration result.

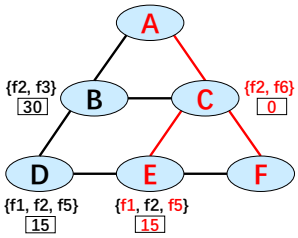


Figure 3.7: 2^{nd} iteration result.

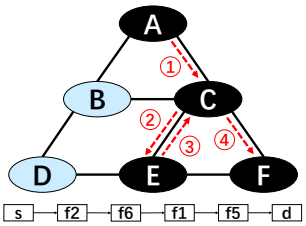


Figure 3.8: Result of COFO.

Table 3.2: CF Value and T Updates

Iteration	B	C	D	E
1	$\frac{\frac{1}{1}+\frac{1}{2}}{2}=\frac{3}{4}$	$\frac{\frac{2}{1}+\frac{2}{1}}{2}=\mathbf{2}$	$\frac{\frac{1}{2}+\frac{1}{2}}{2}=\frac{1}{2}$	$\frac{\frac{3}{2}+\frac{3}{2}}{2}=\frac{3}{2}$
T_1	A, F, $path_{A,C}$, $path_{C,F}$			
Instantiated SFs	f2, f6			
2	0	0	$\frac{1}{2}$	$\frac{\mathbf{2}}{\mathbf{1}}=\mathbf{2}$
T_2	A, F, $path_{A,C}$, $path_{C,F}$, $path_{E,C}$			
Instantiated SFs	f1, f2, f5, f6			

return the SFP and its corresponding SFC.

In Fig. 3.5, the set of SF instances and computing capacity are listed beside each physical node. We assume that there is an NSR starting at A and ending with F, while requiring four SFs, namely, f1, f2, f5 and f6, each of which demands 15 units of computing resources. When applying our proposed COFO, Table 3.2 lists the variation of CF values and T in each iteration. In the first iteration, as node C owns the highest CF value, it is picked by COFO, and paths $path_{A,C}$, $path_{C,F}$ are added to T_1. The red nodes and links in Fig. 3.6 show the paths (links) and nodes in T. In the second iteration, node E will be selected by COFO, and $path_{E,C}$ will be merged into T, as shown in Fig. 3.7. As nodes C and E instantiate all required SFs, COFO then forms the shortest spanning closed walk on T and deletes the shortest path connecting A and F in the walk. Fig. 3.8 shows the constructed

SFP A→C→E→C→F along with the SFC s→f2→f6→f1→f5→d.

As one can see, the step with the highest runtime is updating the CF value for each physical node. When applying the shortest path algorithm with the time complexity $|NL + N^2 logN|$, the time complexity of COFO is $|N^2L + N^3 logN|$.

3.4.2 COst Factor-based SFCE Optimization with ShortCut (COFO-SC)

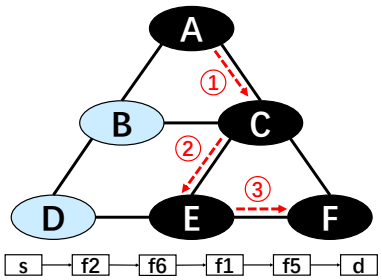


Figure 3.9: Result of COFO after applying the shortcut technique.

In the final iteration of the above COFO, T is the minimum spanning tree (e.g., the red nodes and links in Fig. 3.7) that visits all selected nodes. When applying the shortest spanning closed walk on a minimum spanning tree (i.e., the red nodes and links in Fig. 3.7) to generate the SFP, the walk will follow the order as shown in Fig. 3.8: A→C→E→C→F. This means that, based on COFO, the route going from E to F will have to pass through C

Algorithm 3 COst Factor-based SFCE Optimization with ShortCut (COFO-SC)

Input: G, NSR;
Output: SFP, SFC;
{SFP, SFC} = COFO(G, NSR);
for each adjacent pair of physical nodes m, n in SFP **do**
    if $|path^{G}_{m,n}| \leq |path^{SFP}_{m,n}|$ **then**
        Replace $path^{SFP}_{m,n}$ by $path^{G}_{m,n}$;
    end if
end for
return SFP, SFC;

in the physical network. In fact, in the original PN, there may be a shorter route going from E to F, and this route does not pass through C. We call such a shorter route a shortcut path. Specifically, for two adjacent physical nodes (say, m and n) of the SFP created by COFO, if there is a path in the original physical network G, say $Path^{G}_{m,n}$, which is shorter than the $Path^{SFP}_{m,n}$ in the constructed SFP, then we call this $Path^{G}_{m,n}$ a shortcut path from m to n. For example, for the path E→C→F in Fig. 3.8, there actually is a shortcut path E→ F in the original physical network, as shown in Fig. 3.9. Hence, the cost of an SFP generated from the COFO algorithm can be further optimized via the following processes: for each pair of adjacent physical nodes that host SFs, check whether there is a shortcut path in the physical network. If there is, one can replace the path of these two adjacent physical nodes in the SFP with the shortest shortcut path to further minimize the length of the constructed SFP.

Based on the discussion above, we propose the cost factor-based SFCE optimization with shortcuts (COFO-SC) algorithm, as shown in Algorithm 3. COFO-SC calls COFO first to generate the SFP and the corresponding SFC (Line 3). Then, COFO-SC checks whether there exists a shortcut for any two adjacent physical nodes (say m and n) in the generated SFP (Lines 4-5). If there exists such a shortcut path with enough bandwidth, the algorithm will replace the path connecting m and n in the SFP ($Path^{SFP}_{m,n}$) with the shortcut path connecting m and n in the original PN ($Path^{G}_{m,n}$) (Line 6). Finally, COFO-SC returns the updated SFP. In the worst case, each SF will be independently mapped onto a single physical node, and COFO-SC will check $|V+1|$ pairs of physical nodes in the SFP. Generally, $|V|$ is

much less than $|N|$. COFO-SC has the same time complexity as COFO.

3.5 Bounds Analysis

In this section, we analyze the bounds of our proposed algorithms in unique function physical networks (UFPNs) and multiple function physical networks (MFPNs).

3.5.1 Bound Analysis in UFPNs

Given the topology of a unique function physical network, we can transform it into a complete graph, whereas each node provides a required SF, and each link $l_{m,n}$ is the shortest path between m and n in the given UFPN. We call this complete graph an auxiliary complete graph (ACG). Again, in such a situation, even though a physical node provides only one unique type of SF, multiple SF instances can be hosted to support multiple NSRs. When applying the proposed schemes, the CF factor depends on the distance between the node n and the connected component T as $|\sigma_n|$ will always be 1. Accordingly, in each iteration, the node n in the complete graph with the shortest distance to T will be selected, and the shortest path $path_{n,T}$ will be merged to T. This process is equivalent to the prime algorithm of constructing the minimum spanning tree (MST) in ACG [49].

Lemma 3.5.1. *Given a UFPN and an NSR, the optimal SFP exists in an ACG that is generated from UFPN [17].*

Theorem 3.5.2. *Given an NSR, COFO-SC constructs an SFP that is 2-approximation to the optimal SFP in UFPNs.*

Proof. We use $|SFP_{\text{COFO-SC}}|$ to represent the cost of the SFP generated from the proposed schemes, $|MST|$ to denote the length of the T after the last iteration, $|SFP_{\text{opt}}|$ to present the length of the optimal SFP, and $|Path_{s,d}|$ to represent the length of the shortest path connecting the source and destination.

From Lemma 3.5.1, as the MST (i.e., T after the last iteration) is the minimum length structure that connects all nodes in a graph, Eq. (3.16) holds.

$$|MST| \leq |SFP_{\text{opt}}| \tag{3.16}$$

According to the proposed scheme, the length of the generated SFP equals the length of the shortest spanning closed walk minus the length of the shortest path connecting the source and destination. From Lemma 3.3.2, the length of the shortest spanning closed walk equals twice the length of the MST, thereby, Eq. (3.17) holds.

$$|SFP_{\text{COFO-SC}}| = 2 * |MST| - |Path_{s,d}| \tag{3.17}$$

When combining Eq. (3.16) and Eq. (3.17), Eq. (3.18) holds.

$$|SFP_{\text{COFO-SC}}| \leq 2 * |SFP_{opt}| - |Path_{s,d}| \tag{3.18}$$

As a result, the proposed COFO-SC can accommodate an NSR in UFPNs within a 2-approximation guarantee. □

Lemma 3.5.3. *Given a UFTN and an NSR, the proposed COFO-SC can generate an optimal SFP with the minimum cost.*

Proof. Based on the proposed COFO-SC, after the final iteration, T is the MST that visits

all required SFs, which is equivalent to the basic subtree (BST). BST is the shortest subtree that includes all required SFs. Then, COFO-SC applies the shortest spanning closed walk technique on the BST to generate the SFP. Based on the analysis in Section IV, the SFP generated from COFO-SC is the optimal SFP. □

3.5.2 Bound Analysis in MFPNs

Table 3.3 describes the necessary notations that will be used in the following proof.

Table 3.3: Notations for Approximation Proof

Variable	Notation
$\|\sigma_n^i\|$	Number of SFs instantiated at node n in iteration i
T^i	Connected component T in iteration i
$\|Dis_{n,T^i}\|$	Distance from n to T^i in iteration i
$\|SF_{left}^i\|$	Number of SFs that have not been satisfied in iteration i
SFP_{opt}^i	Optimal SFP to satisfy SF_{left}^i in iteration i
$\|Dis_{n,SFP_{opt}^i}\|$	Distance from n to the SFP_{opt}^i in iteration i
$\|\alpha^i\|$	Number of physical nodes in SFP_{opt}^i
$\|\gamma\|$	Number of physical nodes that host required SFs in SFP_{opt}^1
$\|V\|$	Number of required SFs in NSR
$\|k\|$	Minimum number of SFs satisfied in all iterations
$\|\beta\|$	Total number of iterations to satisfy all SFs

Theorem 3.5.4. *Given an NSR, the proposed COFO-SC is the* $\frac{ln(|V|)}{|k|}$*-approximation to the optimal SFP in MFPNs.*

Proof. Eq. (3.19) is the relationship between the number of satisfied SFs and the unsatisfied SFs in each pair of adjacent iterations (i.e., iteration i and $i+1$).

$$|SF_{left}^i| = |SF_{left}^{i+1}| + |\sigma_n^i|, \forall i \in |\beta| \tag{3.19}$$

As COFO-SC selects the node n with the highest CF value, CF_n is greater than the average CF value of the physical nodes in SFP^i_{opt}. This is because the physical nodes in the optimal SFP can exist: (1) in T_i and (2) out of T_i. For any node m in T_i, node n has a greater CF value than CF_m; otherwise, CF_n is not the greatest. For any node m out of T_i, the CF_m is calculated as $max(\frac{|\sigma^i_m|}{Dis_{m,s}}, \frac{|\sigma^i_m|}{Dis_{m,d}})$, which is also less than CF_n. Therefore, Eq. (3.20) holds.

$$\frac{|\sigma^i_n|}{|Dis_{n,T^i}|} \geq \frac{\sum_{m \in SFP^i_{opt}} \frac{|\sigma^i_m|}{Dis_{m,SFP^i_{opt}}}}{|\alpha^i|}, \forall i \in |\beta| \tag{3.20}$$

As different physical nodes may provide the same SF instance(s), Eq. (3.21) holds.

$$\sum_{m \in SFP^i_{opt}} |\sigma^i_m| \geq |SF^i_{left}|, \forall i \in |\beta| \tag{3.21}$$

For a node m in SFP^i_{opt}, in order to maximize $\frac{|\sigma^i_m|}{Dis_{m,SFP^i_{opt}}}$, the denominator of the CF value is not greater than the half length of the SFP^i_{opt}, as shown in Eq. (3.22).

$$Dis_{m,SFP^i_{opt}} \leq \frac{|SFP^i_{opt}|}{2}, \forall i \in |\beta| \tag{3.22}$$

Initially, no SF has been instantiated by any physical node; thus, Eq. (3.23) holds.

$$|\alpha^i| \leq |\gamma|, \forall i \in |\beta| \tag{3.23}$$

From Eqs. (3.21-3.23), Eq. (3.20) can be transferred into Eq. (3.24).

$$\frac{|\sigma^i_n|}{|Dis_{n,T^i}|} \geq \frac{|\gamma| * |SF^i_{left}|}{|\gamma| * \frac{|SFP^i_{opt}|}{2}}, \forall i \in |\beta| \tag{3.24}$$

Eq. (3.24) can be further transferred into Eq. (3.25).

$$|Dis_{n,T^i}| \leq \frac{|\sigma_n^i|}{|SF_{left}^i|} * \frac{|SFP_{opt}^i|}{2}, \forall i \in |\beta| \tag{3.25}$$

When summing up all iterations, Eq. (3.25) will be transferred into Eq. (3.26).

$$\sum_{i=1}^{|\beta|} |Dis_{n,T^i}| \leq \sum_{i=1}^{|\beta|} \frac{|\sigma_n^i|}{|SF_{left}^i|} * \frac{|SFP_{opt}^i|}{2} \tag{3.26}$$

According to Eq. (3.19), Eq. (3.27) holds.

$$\frac{|\sigma_n^i|}{|SF_{left}^i|} \leq \frac{1}{|SF_{left}^i|} + \frac{1}{|SF_{left}^i| - 1} + ... + \frac{1}{|SF_{left}^{i+1}|} \leq \sum_{\epsilon=SF_{left}^{i+1}}^{SF_{left}^i} \frac{1}{\epsilon}, \forall i \in |\beta| \tag{3.27}$$

Combining Eq. (3.26) and Eq. (3.27), one will have Eq. (3.28), which can be further transferred into Eq. (3.29).

$$\sum_{i=1}^{|\beta|} |Dis_{n,T^i}| \leq \sum_{i=1}^{|\beta|} \sum_{\epsilon=SF_{left}^{i+1}}^{SF_{left}^i} \frac{1}{\epsilon} * \frac{|SFP_{opt}^i|}{2} \tag{3.28}$$

$$\sum_{i=1}^{|\beta|} |Dis_{n,T^i}| \leq \sum_{\epsilon=SF_{left}^{|\beta|}}^{|V|} \frac{1}{\epsilon} * \frac{|SFP_{opt}^i|}{2} \tag{3.29}$$

According to the properties of a harmonic series, Eq. (3.30) holds.

$$\sum_{\epsilon=SF_{left}^{|\beta|}}^{|V|} \frac{1}{\epsilon} = \frac{1}{|V|} + \frac{1}{|V| - \sigma_n^1} + ... + \frac{1}{SF_{left}^{|\beta|}} \leq$$

$$\frac{1}{|V|} + \frac{1}{|V| - |k|} + \frac{1}{|V| - 2 * |k|} + ... + \frac{1}{SF_{left}^{|\beta|}} \leq \frac{ln(|V|) + 1}{|k|} \tag{3.30}$$

As the optimal SFP in the first iteration SFP_{opt}^1 is in fact the overall optimal SFP, for any optimal SFP in other iterations SFP_{opt}^i $(i > 1)$, Eq. (3.31) and (3.32) hold.

$$SFP_{opt}^1 > SFP_{opt}^i, \forall i > 1 \tag{3.31}$$

$$\sum_{i=1}^{|\beta|} |Dis_{n,T^i}| \leq \frac{ln(|V|)+1}{|k|} * \frac{|SFP_{opt}|}{2} \tag{3.32}$$

In fact, the connected component in the last iteration $T^{|\beta|}$ is the spanning tree that connects the source node, destination node and all selected nodes. Based on Lemma 3.3.2, Eq. (3.33) holds.

$$|SFP_{\text{COFO-SC}}| \leq 2 * \sum_{i=1}^{|\beta|} |Dis_{n,T^i}| - path_{s,d} \leq \frac{ln(|V|)+1}{|k|} * |SFP_{opt}| \tag{3.33}$$

Thus, given an NSR, the COFO-SC algorithm is a $\frac{ln(|V|)}{|k|}$-approximation for the MC-SFCE problem in MFPNs. □

3.6 Experimental Results and Analysis

In this section, we show the performance of the proposed algorithms[2] when comparing with the techniques that are directly extended from the schemes [37] and [26].

3.6.1 Simulation Environment

Similar to the state-of-the-art simulation settings in [17, 36, 37, 38], we randomly generate 40-node-180-link mesh networks and 6-fat-tree as physical networks (PNs). For the mesh network, each physical node owns at least four direct neighbors (i.e., each physical node directly connects at least four other nodes). In both mesh and fat-tree networks, each physical node is equipped with the computing capacity in the range of $[30-60]$ and is

[2]The implementations of our methods can be found at github.com/frozenlalala/Function-Embedding-Strategy.

capable of instantiating $[2-5]$ different SF instances. For each physical link, it has available bandwidth in the range of $[5-20]$ and the cost in the range of $[3-8]$.

We set the number of SF nodes required by an NSR in the range of $[6-24]$. For each SF node, the required computing capacity is in the range of $[10-20]$. The source and destination nodes required in the NSR are randomly generated, while the bandwidth demand is set as 5.

3.6.2 Performance Metrics and Benchmarks

We use the following metrics to evaluate the performance of the proposed schemes.

Acceptance Ratio: When the bandwidth provided by the PN is not sufficient, the scheme cannot embed some NSRs. We use the acceptance ratio (AR) to evaluate the performance. AR is calculated based on $AR = \frac{|\text{NSR}_{\text{Accept}}|}{|\text{NSR}_C|}$, where $|\text{NSR}_{\text{Accept}}|$ represents the number of accepted NSRs, while NSR_C is the set of all input requests.

Cost Per Request: We define the cost per request (CPR) to evaluate the performance of the proposed schemes. Here, CPR represents the average cost of the accepted requests and is calculated as $\text{CPR} = \frac{\sum_{\text{NSR}_i \in \text{NSR}_{Accept}} |SFP|}{|\text{NSR}_{Accpet}|}$, where $|SFP|$ represents the length of the SFP for NSR_i.

We extended two state-of-the-art techniques in [37] and [26] as the benchmarks. We extend the method in [37] as follows: (1) calculating the BC value for each node n (BC_n); (2) using the factor $BC_n * \sigma_n$ to evaluate the importance of each physical node, where σ_n represents the number of SF instances that node n can accommodate under the computing resource constraint; (3) selecting the node with the highest factor value; (4) embedding

the set of SFs that maximizes the factor value onto n; (5) repeating (1)-(4) until all SFs are satisfied; and (6) connecting the source, destination and all selected nodes based on the nearest-neighbor algorithm proposed in [29] to form the SFP. Note that, as proved in [29], the proposed nearest neighbor algorithm can visit all required cities (nodes) within a $|\frac{1}{2}\lceil lg(N)\rceil + \frac{1}{2}|$-approximation, where N represents the number of physical nodes in the given network. We name the technique that is extended from [37] "BACON". When given an SFC and the physical network bandwidth is not constrained, the algorithm proposed in [26] can construct an optimal SFP. We extend the technique proposed in [26] as follows: (1) generating ten random SFCs, (2) constructing corresponding SFPs by applying [26], and (3) selecting the SFP with the minimum cost. We call the technique extended from [26] "SP-SFCE".

3.6.3 Performance Analysis in UFPNs

We first evaluate whether the proposed scheme can achieve a 2-approximation performance in unique function physical networks (UFPNs). Under this scenario, each node only provides one unique SF. Based on Eq. (3.16), the length of the MST is less than the length of the optimal SFP, which can be regarded as the lower bound. To test whether the proposed COFO-SC guarantees a 2-approximation in UFPNs, we set the "Upper Bound" as twice the length of MST and extended the branch and bound (B&B) technique proposed in [51] to formulate the "Lower Bound". Figs. 3.10 and 3.11 illustrate the performance of the proposed schemes when increasing the number of required SFs in UFPNs. The yellow-triangle-solid, yellow-triangle-dotted, gray-square-solid and gray-square-dotted curves represent the performance

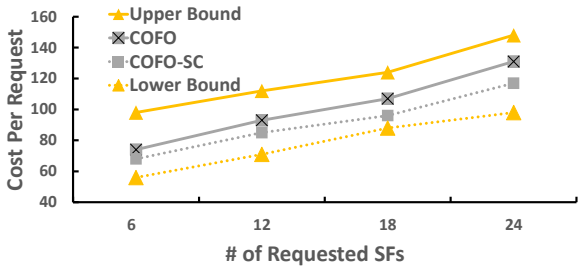


Figure 3.10: Unique function mesh networks.

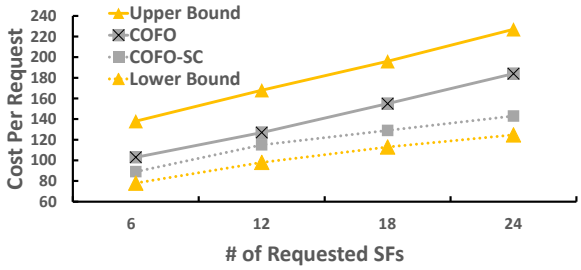


Figure 3.11: Unique function fat-tree.

of "Upper Bound", "Lower Bound", "COFO" and "COFO-SC", respectively.

In Fig. 3.10, both the COFO and COFO-SC algorithms outperform the upper bound, which verifies the correctness of the 2-approximation. In mesh networks, there exists more than one path from one node to another node. As a result, the physical path between two adjacent physical nodes that host the SF instance(s) may not be the shortest path. As COFO-SC applies the shortcut technique to further optimize the cost between the adjacent pair of physical nodes that host SF instances in the constructed SFP, it outperforms COFO.

In Fig. 3.11, when increasing the number of required SFs, both algorithms achieve the 2-approximation performance. When the network is a fat-tree topology, all SFs are supported by the servers locating at the leaf nodes. The minimum spanning tree (MST) is likely created as a star, whereas the source node or the destination node will play the role as the "root". Since the COFO algorithm does not apply the shortcut technique, the path connecting one server to another server may bypass the "root" of the MST and introduce additional cost in COFO. In contrast, as COFO-SC applies the shortcut technique, these detours can be avoided by steering the traffic to the shortest path connecting one server to another directly.

As one can see from Fig. 3.10 and Fig. 3.11, both the COFO and COFO-SC algorithms

achieve the 2-approximation performance. On average, the COFO-SC algorithm outperforms COFO by 10.4% and 18.5% in mesh networks and fat-trees, respectively.

3.6.4 Performance Analysis in MFPNs

We then evaluate the performance of the proposed scheme in multiple function physical networks. The evaluations are carried out by varying the bandwidth in each link (Figs. 3.12 and 3.13), the computing resource in each node (Fig. 3.14), the size of the network (Fig. 3.15), and the number of required SFs (Fig. 3.16). In the following figures, the red-rhombus-solid, blue-circle-dashed and gray-square-dotted curves represent the performance of "SP-SFCE", "BACON" and "COFO-SC", respectively. Specifically, the red-grid, blue-dotted and gray-solid bars in Fig. 3.13 represent the acceptance ratio of "SP-SFCE", "BACON" and "COFO-SC", respectively.

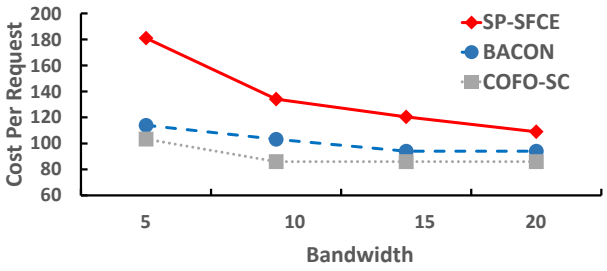


Figure 3.12: CPR vs. bandwidth.

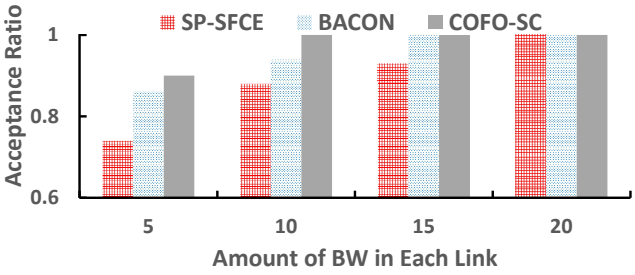


Figure 3.13: Acceptance vs. bandwidth.

Figs. 3.12 and 3.13 show the performances of all schemes when varying the bandwidth in each link in terms of the CRP and accepting ratio. As one can see, all schemes need less cost to accommodate the NSRs and have higher accepting ratios with an increasing bandwidth in each link. When the bandwidth is severely limited, all schemes try to accommodate the NSRs by exploring all possible "longer" paths with enough bandwidth, which leads to the

higher cost. When bandwidth is abundant, the shorter path (i.e., the path with less cost) becomes available, which leads to the lower cost. Since the COFO-SC algorithm jointly composites and embeds the SFC, it effectively steers the traffic even when the bandwidth is severely limited. As a result, the COFO-SC algorithm has a higher accepting ratio than the other two techniques. SP-SFCE has the worst performance because it totally depends on how the SFCs are constructed. Overall, COFO-SC accepts 97.5% of NSRs, and it outperforms SP-SFCE and BACON by an average of 49.4% and 12.3%, respectively.

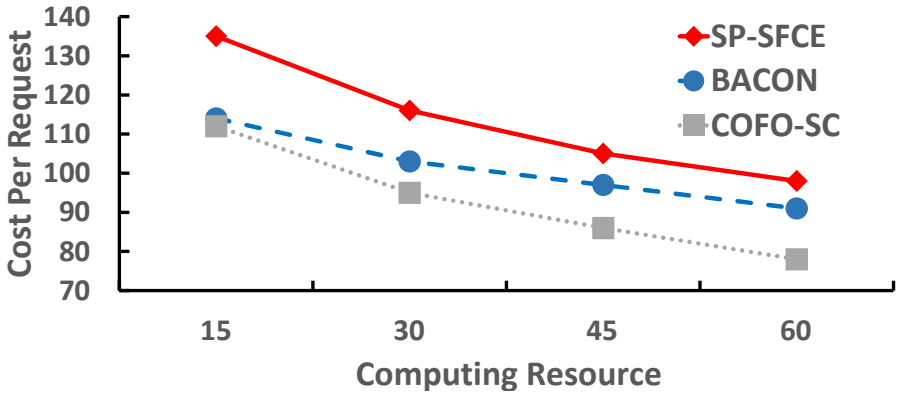


Figure 3.14: CPR vs. computing resource.

To show the impact of the computing resource, we set the computing resource of each physical node in the range of [15, 60]. Fig. 3.14 demonstrates the CPR of all three schemes when varying the amount of computing resources in each physical node. As the computing resources provided by physical nodes increases, the cost needed to construct the SFPs decreases for all schemes. This is because, with more computing resource, the schemes can instantiate more SFs onto nearby physical nodes, which can decrease the number of physical nodes needed for the SFPs and reduce the SFP construction cost. From our experimental results, COFO-SC outperforms the other two schemes. The reason is that the proposed cost

factor can effectively identify the proper physical node to construct the SFP by taking the number of SF instances provided by the physical node and the distance into consideration. On average, COFO-SC outperforms SP-SFCE and BACON by 22.6% and 9.9%, respectively.

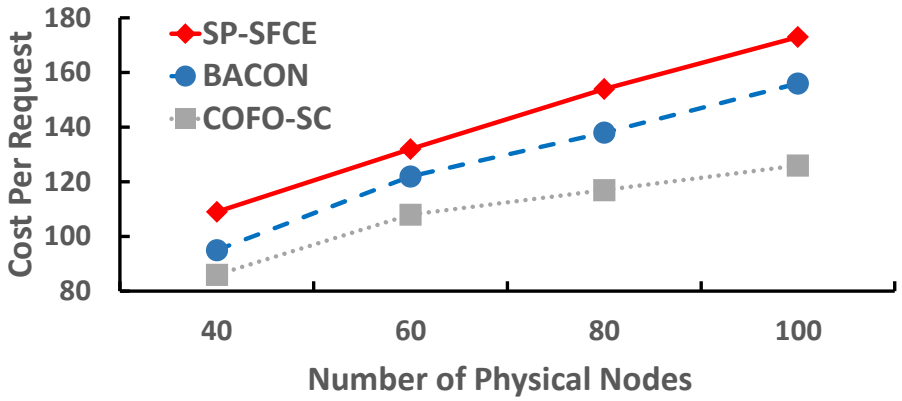


Figure 3.15: CPR vs. number of physical nodes.

As shown in Fig. 3.15, when increasing the size of the network, all algorithms require a longer average cost to construct the SFPs. In a large network, the betweenness centrality (BC) measurement cannot accurately indicate the importance of physical nodes. The reason is that multiple paths may be the "shortest" from source to destination. As a result, the physical nodes selected by BACON might be widely distributed among the network, which can introduce additional cost. As the cost factor takes the distance between the selected physical node and the physical node candidates into consideration, it is able to effectively pick the physical node that can simultaneously host many SF instances while not introducing much additional cost. Hence, COFO-SC outperforms SP-SFCE and BACON by an average of 29.4% and 16.4%, respectively.

As shown in Fig. 3.16, with an increasing number of SFs, all three schemes require a higher cost to construct the SFPs. When the number of required SFs is small, SP-SFCE

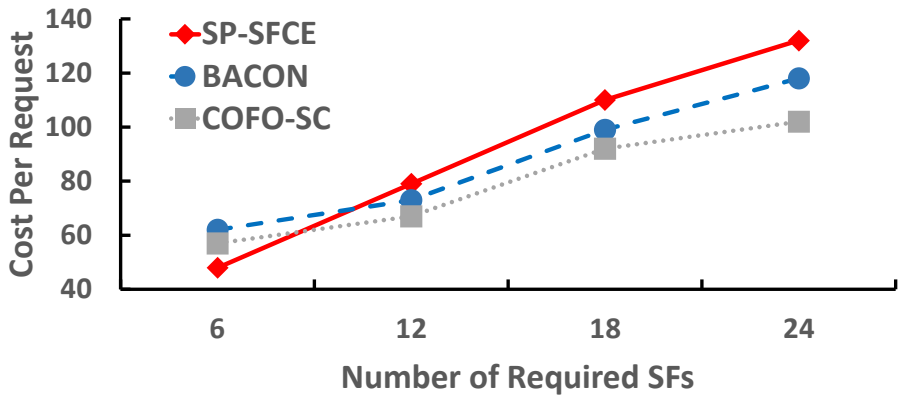


Figure 3.16: CPR vs. number of SFs.

outperforms the other two schemes. The reason is that when the number of required SFs is small, there are relatively high probabilities that the optimized SFCs are constructed and that the SP-SFCE can optimally embed a given SFC. On average, COFO-SC outperforms SP-SFCE by 12.7% and BACON by 10.2%.

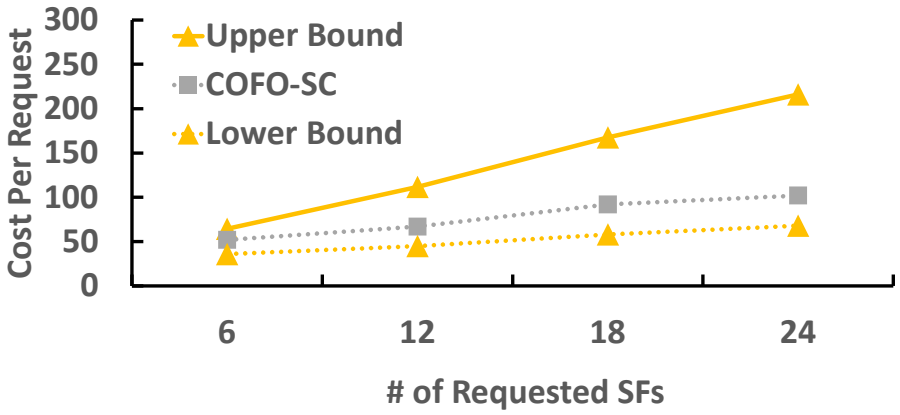


Figure 3.17: Results in multiple function mesh networks.

Last but not least, we verify the approximation performance of the proposed COFO-SC algorithm in multiple function mesh networks. Here, we implemented the method proposed in [26] to find the least length SFP among all SFC compositions as the "Lower Bound". Then, we use the performance of the lower bound that multiplies $\ln(|V|)$ as the "Upper

Bound". In Fig. 3.17, the upper bound, COFO-SC, and lower bound are denoted by the yellow triangle-solid, gray square-dotted, and yellow triangle-dotted curves, respectively. As one can see, the COFO-SC algorithm guarantees the logarithm-approximation. Moreover, the results verify that the proposed CF value can effectively select the physical node to host many SFs while introducing little cost to construct the SFP.

3.7 Summary

In this chapter, we have comprehensively investigated how to deliver IoT-based services by jointly compositing and embedding an SFC with provable bounds. We have mathematically formulated the minimum cost service function chaining and embedding (MC-SFCE) problem. To optimize MC-SFCE problem, we have proposed an efficient algorithm called COst Factor-based SFCE Optimization with ShortCut (COFO-SC), which is the 2-approximation in unique function physical networks and $\frac{ln(|V|)}{|k|}-$approximation in a generic physical network, where $|V|$ is the number of the SF nodes, and $|k|$ denotes the minimum number of SFs satisfied in all iterations during the node selection processes. Through extensive simulations and analysis, we have shown that our proposed COFO-SC outperforms two schemes that are extended from the existing techniques in terms of the acceptance ratio and the average cost for accommodating NSRs.

CHAPTER 4
HYBRID SERVICE CHAIN COMPOSITION AND EMBEDDING

4.1 Motivation

Recently, 5G and Multi-access Edge Computing (MEC) empowers the development of the latency-sensitive or computation-intensive applications such as realtime Virtual Reality (VR), Augmented Reality (AR) games and on-line machine learning [7]. In these applications, the forward traffic from the user and the backward traffic from the MEC server/cloud may require different sets of SFs. The SFC that requires different sets of SFs in the forward and backward directions is referred to as hybrid SFC (h-SFC) [14]. Fig. 4.1 demonstrates an example of an in-service h-SFC for on-line machine learning. The forward traffic (containing data sets) from the source (i.e., user) requires Fire Wall (FW) and Deep Packet Inspection (DPI), while the backward traffic (containing the trained model) has to go through Encryption (Encry), Decryption (Decry) and FW. In the SN, the forward SFP (f-SFP) is Source→A→B→DEST, while the backward SFP (b-SFP) includes DEST⇢D⇢C⇢B⇢A⇢Source. To save the Operating Expense (OPEX) and latency, the SFs required by both directions are generally installed on the same substrate node (e.g., FW in Fig. 4.1) [52] [53].

In this chapter, for the first time, we comprehensively study how to jointly composite and embed an NSR with hybrid traffic onto a shared substrate network. We define a new problem called *Hybrid SFC composition and Embedding* (HSFCE) and propose novel analysis and algorithms for various substrate network scenarios. Our main contributions in this chapter are summarized as follows.

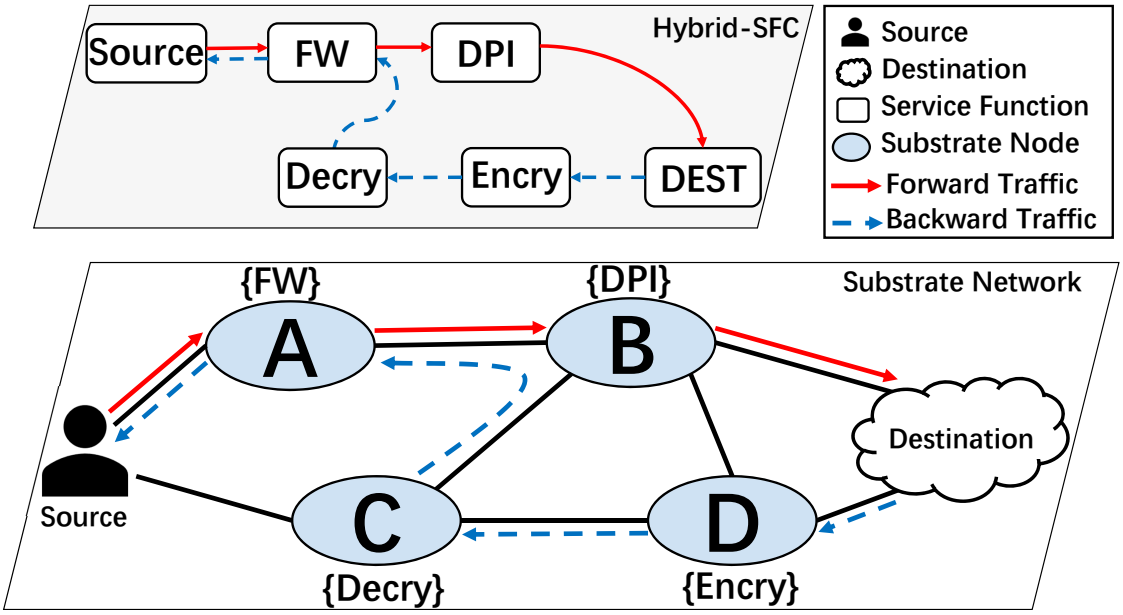


Figure 4.1: An example of in-service h-SFC for one on-line machine learning scenario.

- We mathematically model the problem of Hybrid SFC composition and Embedding (HSFCE) with the objective of minimizing the latency for the constructed hybrid SFP.

- When each substrate node provides only one unique SF, we prove the NP-hardness of HSFCE and propose a 2-approximation algorithm to jointly composite and embed the h-SFC. The proposed algorithm is called Eulerian Circuit based Hybrid SFP optimization (EC-HSFP).

- When a substrate node can provide various SFs, we propose an effective heuristic algorithm called Betweenness Centrality based Hybrid SFP optimization (BC-HSFP).

- Through extensive analysis and simulations, we prove that EC-HSFP guarantees the 2-approximation performance and show that BC-HSFP outperforms the algorithms directly extended from the existing techniques by an average of 20%.

The rest of this chapter is organized as follows. Section II mathematically formulates the HSFCE problem. In Section III and IV, we provide analysis and algorithms for HSFCE in various substrate network scenarios. Section V presents the experimental results and analysis. We summarize this chapter in Section VI.

4.2 Problem Statement

4.2.1 Substrate/physical Network Model

The Substrate Network (SN) is denoted by an undirected graph $G = (N, L, F)$, whereas N represents a set of substrate nodes, L is a set of substrate links, and F denotes a set of available SFs. Each substrate node $n \in N$ provides a specific set of SF instances $\mathbb{F}_n$ ($\mathbb{F}_n \subseteq F$) and a certain amount of available computing capacity c_n. Each physical link $l \in L$ can provide a specific amount of bandwidth bw_l and has a certain latency. For simplicity, a physical link $l \in L$ can also be represented as $l_{m,n}$, where $m, n \in N$ are its endpoints. For a physical path $path_{m,n}$, it has an accumulative latency cost $W_{path_{m,n}}$ depending on the links in this path. In this chapter, we investigate two different network scenarios, Unique Function SN (UFSN) and Multi-Functions SN (MFSN). In UFSN, each substrate node only provides one unique SF instance. That is to say, no two different substrate nodes provide the same SF instance (i.e., $\mathbb{F}_n \cap \mathbb{F}_m = \emptyset$, $\forall m \neq n$). This scenario is practical in the sub-domains of a large network or the resource-constrained systems [32, 48]. In MFSN, each substrate node can provide various SF instances subjecting to the computing capacity and different substrate nodes may offer the same SF instance(s) (i.e., $\mathbb{F}_n \cap \mathbb{F}_m$ may not be empty).

4.2.2 Network Service Request with Hybrid Traffic

A Network Service Request with hybrid traffic can be represented as a 4-tuple $NSR =< s, V_f, V_b, BW >$, where s is the source node, V_f represents the set of required forward SF nodes, V_b specifies the set of backward SF nodes and BW denotes the amount of bandwidth demand. Each SF node $v \in V$ ($V = V_f \cup V_b$) requires a specific SF f_v and a certain amount of computing demand c_v.

4.2.3 Hybrid Service Function Chain composition and Embedding (HSFCE)

The optimization problem of the HSFCE is defined as: given an NSR with hybrid traffic demands, how to composite and embed the hybrid SFC onto a shared SN such that i) the constraints below are satisfied, and ii) the latency of the constructed SFPs is minimized. The objective function is shown in Eq. (5.3), where $\mu_{path_{m,n}^{u,v}}$ represents the latency of the $path_{m,n}$ to support the traffic from SF node u to v.

$$min \sum_{v \in V} \sum_{u \in V} \sum_{n \in N} \sum_{m \in N} \mu_{path_{m,n}^{u,v}} \tag{4.1}$$

SF node mapping constraint: To map an NSR, SF node mapping process needs to follow the constraints in Eqs. (5.4-5.8). We use M_n^v in Eq. (5.4) to represent whether an SF node v is mapped onto the substrate node n and Δ_n^v in Eq. (5.5) to denote whether the substrate n provides the SF instance for the SF node v. Eq. (5.6) ensures that every SF node has to be mapped onto one specific substrate node. In Eq. (5.7), an SF node can only be mapped onto the substrate node that provides the corresponding SF instance. Each substrate node can host a limited number of SF nodes due to the computing capacity as shown in Eq. (5.8).

$$M_n^v = \begin{cases} 1, & \text{SF node } v \in V \text{ is mapped onto} \\ & \text{substrate node } n \in N \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

$$\Delta_n^v = \begin{cases} 1, & \text{substrate node } n \text{ provides} \\ & \text{SF instance for } v \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

$$\sum_{n \in N} M_n^v = 1, \quad \forall v \in V \tag{4.4}$$

$$M_n^v \leq \Delta_n^v, \quad \forall n \in N, \forall v \in V \tag{4.5}$$

$$\sum_{v \in V} M_n^v * c_v \leq c_n, \quad \forall n \in N \tag{4.6}$$

forward/backward SFP (f-SFP/b-SFP) construction constraint: We use $\text{path}_{m,n}^{u,v}$ to denote whether the path from m to n is used to support the traffic from SF node u to v as shown in Eq. (5.10). In Eq. (4.8), only the path whose endpoints are mapped by some SF nodes is able to construct the f-SFP/b-SFP. Note that, we use f_s to represent the function for the source node. Eq. (5.9) guarantees that there must be one path going from the source to the substrate node n that is mapped by one forward SF node. If an SF node u is demanded

in both directions, Eq. (4.10) ensures that there are two paths starting from the substrate node m where u is mapped (one for f-SFP and the other one for b-SFP). Note that, if an SF node u required by both directions is the last SF node in the f-SFP, the first backward SF node is also u, which is represented as $path^{u,u}_{m,m} = 1$. If an SF node u is demanded only in forward/backward direction, Eq. (4.11)/(4.12) ensures that there is only one path starting from the substrate node m where u is mapped. Eq. (4.13) shows that, for any substrate node m, the number of outgoing path(s) from m equals the number of incoming path(s) to m. The number of connections among any proper-subset of the mapped SF nodes V_{sub} should not be more than $|V_{sub}| - 1$ to avoid the circle, which is shown in Eq. (4.14). Eq. (4.15) guarantees that each link of the selected paths should provide enough bandwidth. Eq. (4.16) shows the latency of $\text{path}^{u,v}_{m,n}$.

$$\text{path}^{u,v}_{m,n} = \begin{cases} 1, & \text{path from substrate node } m \text{ to } n \text{ is used} \\ & \text{to support the traffic from SF node } u \text{ to } v \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

$$\text{path}^{u,v}_{m,n} \leq \frac{M^u_m + M^v_n}{2}, \forall m, n \in \{N \cup s\}, \forall u, v \in \{V \cup f_s\} \tag{4.8}$$

$$\sum_{v \in V_f} \sum_{n \in N} \text{path}^{f_s,v}_{s,n} = 1 \tag{4.9}$$

$$\sum_{n \in N} \sum_{v \in V} (\text{path}^{u,v}_{m,n} + \text{path}^{u,f_s}_{m,s} + \text{path}^{u,u}_{m,m}) = 2 * M^u_m, \forall u \in \{V_f \cap V_b\}, u \neq v, \forall m \in N \tag{4.10}$$

$$\sum_{n\in N}\sum_{v\in V}\text{path}_{m,n}^{u,v} = M_m^u, \forall u \in \{V_f - (V_f \cap V_b)\}, u \neq v, \forall m \in N \tag{4.11}$$

$$\sum_{n\in N}\sum_{v\in V}(\text{path}_{m,n}^{u,v} + \text{path}_{m,s}^{u,f_s}) = M_m^u, \forall u \in \{V_b - (V_b \cap V_f)\}, u \neq v, \forall m \in N \tag{4.12}$$

$$\sum_{n\in\{N\cup s\}}\sum_{v\in\{V\cup f_s\}}\text{path}_{m,n}^{u,v} = \sum_{o\in\{N\cup s\}}\sum_{w\in\{V\cup f_s\}}\text{path}_{o,m}^{w,u}, \forall u \in V, \forall m \in N \tag{4.13}$$

$$\sum_{u\in V_{sub}}\sum_{v\in V_{sub}}\sum_{m\in N}\sum_{n\in N}\text{path}_{m,n}^{u,v} \leq |V_{sub}| - 1, \forall V_{sub} \subsetneqq \{V \cup f_s\}, V_{sub} \neq \emptyset, u \neq v \tag{4.14}$$

$$\sum_{u\in V}\sum_{v\in V}\text{path}_{m,n}^{u,v} * BW \leq bw_{l_{a,b}}, \forall l_{a,b} \in \text{path}_{m,n} \tag{4.15}$$

$$\mu_{\text{path}_{m,n}^{u,v}} = \text{path}_{m,n}^{u,v} * W_{\text{path}_{m,n}}, \forall u, v \in V, \forall m, n \in N \tag{4.16}$$

forward/backward SFPs connection constraint: In the f-SFP, only the last substrate node in f-SFP should connect with the substrate node hosting the SF that is only required in the backward direction. Hence, there should be at most one path supporting the traffic from the forward SF (only required by the forward direction) to the backward SF (only required by the backward direction) as shown in Eq. (4.17). Similarly, if a substrate node m hosts an SF node that is only requested by the backward direction, there should be no connection from m to a substrate node n hosting the SF node that is only required by the forward

traffic, which is shown in Eq. (4.18).

$$\sum_{u\in\{V_f-(V_f\cap V_b)\}}\sum_{v\in\{V_b-(V_f\cap V_b)\}} \text{path}_{m,n}^{u,v} \leq 1, \forall m,n \in N \tag{4.17}$$

$$\sum_{u\in\{V_b-(V_f\cap V_b)\}}\sum_{v\in\{V_f-(V_f\cap V_b)\}} \text{path}_{m,n}^{u,v} = 0, \forall m,n \in N \tag{4.18}$$

4.3 Hybrid SFCE in UFSNs

In this section, we investigate how to jointly composite and embed the hybrid SFCs onto the UFSN. We prove the NP-hardness of HSFCE in UFSN, and propose a 2-approximation algorithm, namely, Eulerian Circuit based Hybrid SFP optimization (EC-HSFP), which includes two proposed techniques: i) Hybrid Trace Construction (HTC), and ii) Hybrid Eulerian Circuit Construction (HECC).

4.3.1 Complexity Analysis of HSFCE in UFSN

When the UFSN is a complete graph and every node provides a requested SF, creating an SFP is equivalent to finding a path that visits each node exactly once. When the UFSN is a random graph, we can convert it to a complete graph and efficiently apply the proposed HTC and HECC techniques. Fig. 4.2 shows an example of a UFSN, where the substrate nodes A, B, C, D, E provide SF instances: f1, f2, f3, f4, f5, respectively. We assume that an NSR requires f1, f2, f3, f4, where f1 and f2 are required by the forward traffic while f3 and f4 are demanded in both directions. The dark nodes in Fig. 4.2 will host the required SF instances. We can then generate a complete graph with only dark nodes by connecting the dark nodes via the shortest paths in the UFSN. This complete graph is called SFP Complete

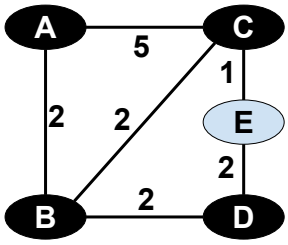


Figure 4.2: An example of UFSN

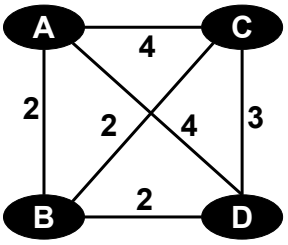


Figure 4.3: SFP complete graph

Graph (SFP-CG). Fig. 4.3 shows the SFP-CG generated from Fig. 4.2, where the number beside the link represents the smallest latency between that pair of substrate nodes.

Lemma 4.3.1. *Given a UFSN, an optimal hybrid SFP for an NSR with hybrid traffic exists in the SFP-CG.*

Theorem 4.3.2. *Hybrid SFC composition and embedding in unique function substrate network is NP-hard.*

Proof. We prove the NP-hardness of HSFCE in UFSN via the reduction of the Travelling Salesman Path Problem (TSPP) [54]. The TSPP tries to find the shortest path that visits each city (node) in the graph exactly once. The HSFCE in UFSN tries to find the shortest SFP connecting the substrate nodes that provide the required SF instances in the UFSN for the forward and backward traffic, respectively; which is equivalent to finding two travelling salesman paths sharing the same endpoints and connecting the substrate nodes in the SFP-CG. Thus, HSFCE in UFSN is NP-hard. □

However, the HSFCE in UFSN cannot be simply optimized by applying the TSPP algorithms twice (one for f-SFP and the other one for b-SFP). This is because, when the optimal f-SFP and b-SFP are generated by the TSPP algorithms, the connection between f-SFP

and b-SFP may be required, which may introduce large latency for the constructed hybrid SFP. Thus, HSFCE in UFSN cannot be directly optimized by applying the existing TSPP techniques.

4.3.2 Hybrid Trace Construction (HTC)

In graph theory, a Hamiltonian path on a connected graph is a path of minimal length which visits every node of a graph exactly once [55]. Accordingly, the optimal unidirectional SFP is a Hamiltonian path of the SFP-CG. Based on the number of connections in the Hamiltonian path, the substrate nodes involved in the construction of a unidirectional SFP can be two types: i) *endpoints*, and ii) *intermediate nodes*. The endpoints include source and destination, whereas the number of connections is odd. For intermediate nodes, they own even number of connections.

Lemma 4.3.3. *The trace for the optimal unidirectional SFP is a path on the SFP-CG.*

Proof. In Lemma 4.3.3, a trace is defined as the set of substrate nodes and links that will be visited from the source to destination in the unidirectional SFP. Since each pair of nodes are connected via the shortest path, no node and link in the SFP-CG will be revisited by the optimal unidirectional SFP. □

Logically, a hybrid SFP can be treated as two inter-related unidirectional SFPs. These two unidirectional SFPs share the same source, destination and common SF instances. Therefore, the hybrid trace (i.e., the trace for a hybrid SFP) in the UFSN can be created by merging the traces of the forward SFP (f-SFP) and backward SFP (b-SFP). With

Lemma 4.3.3, the hybrid SFP starts and ends at the source node, and visits each link in the hybrid trace exactly once, which is in fact a Eulerian circuit of the hybrid trace [55]. That is to say, when the hybrid trace is given a priori, creating a hybrid SFP is equivalent to construct a Eulerian circuit of the hybrid trace. Thus, we propose the Hybrid Trace Construction (HTC) technique to firstly create the hybrid trace as shown in Algorithm 4. HTC first creates the SFP-CG according to the given SN and NSR. Since the Minimum Spanning Tree (MST) has the least length and connects all substrate nodes, to optimize the latency of the constructed hybrid SFP, HTC takes the MST that includes the required forward/backward SF instances as the trace for the f-SFP/b-SFP. We denote these two MSTs by forward and backward MST, respectively. A hybrid trace is then generated by merging the constructed MSTs. As a connected graph has a Eulerian circuit if and only if every node has even degree (connections) [55], HTC doubles the number of links in the hybrid trace to guarantee the existence of the Eulerian circuit.

Algorithm 4 Hybrid Trace Construction (HTC)

1: **Input:** G, NSR;
2: **Output:** hybrid trace;
3: Discarding the links with less than BW bandwidth resource in G;
4: Generate the SFP Complete Graphs (SFP-CGs) for forward SFs and backward SFs;
5: Construct the Minimum Spanning Tree (MSTs) according to the generated SFP-CGs;
6: Create the hybrid trace by merging the forward and the backward MSTs;
7: **for** Each link in the hybrid trace **do**
8: Create one more link with the same endpoints;
9: **end for**
10: **Return** hybrid trace;

4.3.3 Hybrid Eulerian Circuit Construction (HECC)

With the hybrid trace generated from the HTC technique, to construct a Eulerian circuit, we need to visit each link in the hybrid trace exactly once while optimizing latency [55]. To this end, we need to consider: i) the traffic direction for each link (i.e., the link is used for the forward direction or backward direction), ii) the f-SFP ends at which substrate node, and iii) the order of the links to be visited. Accordingly, we introduce the Hybrid Eulerian Circuit Construction (HECC) technique in Algorithm 5, which includes: i) Forward & Backward Link Label (FBLL), ii) Endpoint Determination and iii) Priority Constraint.

Forward & Backward Link Label: Since HTC doubles the number of links in the hybrid trace, the links between each pair of nodes can be either *two-links set* or *four-links set*. For two links set, the FBLL process labels the links with the same direction (i.e., forward or backward) according to the MST that includes this link. That is to say, if a link belongs to the forward MST and is in the two-links set, the FBLL process labels it as forward, vice verse. For four-links set, both endpoints support the SF instance required in both directions. Thus, two links in the four-link set are labelled as forward, while the other two are labelled as backward.

Endpoint Determination: When there is no common SF in both directions, the only substrate node that connects the forward and backward traffic in the hybrid trace is the source node. Thus, the hybrid SFP created from the hybrid trace must firstly visit all required SFs in the f-SFP, go back to the source node, and start the b-SFP. This is also possible for the situation where the forward and backward traffic share some common SF(s).

However, additional latency may be added to the constructed hybrid SFP. This is because, the b-SFP can start with the substrate node that hosts a common SF instance instead of going back to the source node. To reduce the latency, the endpoint determination process (i.e., Line 4-9 in Algorithm 5) finds the common SF accommodated by the substrate node x that has the largest sum-distances of the path $s \to x$ with forward label and path $x \to s$ with backward label and deletes these two paths.

Priority Constraint: To form a hybrid SFP in the hybrid trace, one needs to start and end at s, while each link is visited exactly once. We can label the link that has involved in the construction of the Eulerian circuit as "visited". The priority constraint ensures that, in each direction, the node with even number of unvisited forward/backward links has a higher priority to be visited. If both directions share some common SFs, there is only one path connecting s to x with the forward label and x to s with the backward label. Thus, the forward traffic ends at x, while the backward traffic ends at s.

Algorithm 5 Hybrid Eulerian Circuit Construction (HECC)

1: **Input:** NSR, hybrid trace;
2: **Output:** h-SFC, hybrid SFP;
3: Applying **Forward & Backward Link Label** to the hybrid trace;
4: **if** $V_f \cap V_b \neq \emptyset$ **then**
5: Find the the other endpoint x for f-SFP that hosts the SF $f_v \in V_f \cap V_b$ with the longest shortest path from s to x;
6: **else** x is s;
7: **end if**
8: Delete the $s \to x$ path with forward label;
9: Delete the $x \to s$ path with backward label;
10: Start with s, visit forward links while considering the **Priority Constraint**;
11: Start with x, visit the backward links while considering the **Priority Constraint**;
12: Record the visiting trace as the hybrid SFP and generate the corresponding h-SFC;
13: **Return** hybrid SFP, h-SFC;

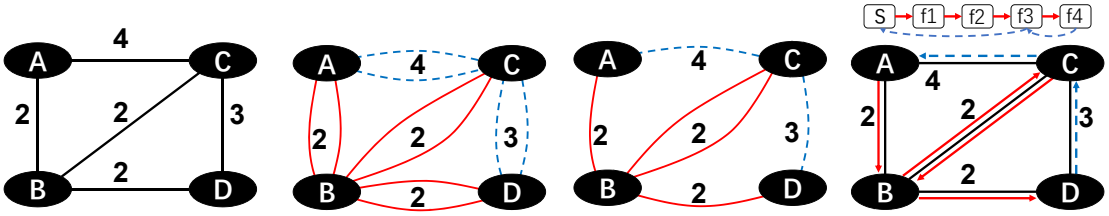
4.3.4 Eulerian Circuit based Hybrid SFP optimization

Based on the proposed HTC and HECC techniques, we present the EC-HSFP algorithm in Algorithm 6. As one can see that, the step with the highest time complexity is to generate the SFP Complete Graph (SFP-CG), which needs to run the shortest path algorithm for multi-times. In the worst case ($|V| = |N|$), when applying the shortest path algorithm whose time complexity is $|NL + N^2 logN|$, the time complexity of EC-HSFP is $|N^2L + N^3 logN|$.

Algorithm 6 Eulerian Circuit based Hybrid SFP optimization (EC-HSFP) Algorithm

1: **Input:** G, NSR;
2: **Output:** h-SFC, hybrid SFP;
3: Create empty sets for hybrid trace, h-SFC and hybrid SFP;
4: hybrid trace = HTC(G, NSR);
5: {hybrid SFP, h-SFC} = HECC(NSR, hybrid trace);
6: **Return** hybrid SFP, h-SFC;

To elaborate how EC-HSFP works, we use Fig. 4.3 as the input. First, EC-HSFP applies the HTC technique to generate the hybrid trace as shown in Fig. 4.4a. Next, the HTC technique doubles the number of links in the hybrid trace and applies the HECC technique. During the process of HECC, FBLL labels the link as forward and backward in Fig. 4.4b, where the red links and blue dotted links represent the forward and back-



(a) Hybrid trace (b) Labelled hybrid trace (c) Delete $x \rightarrow s$ and $s \rightarrow x$ path (d) Hybrid SFP and h-SFC

Figure 4.4: An example of EC-HSFP.

ward labels, respectively. Since D supports the common SF instance (i.e., required in both directions) and has the largest sum-distances to the source node A, D is selected as the endpoint x of the f-SFP. After deleting the $A \rightarrow B \rightarrow D$ path with the forward label and $D \dashrightarrow C \dashrightarrow A$ path with the backward label, Fig. 4.4b is converted to Fig. 4.4c. Next, HECC technique generates the hybrid SFP and the corresponding h-SFC by visiting the substrate nodes with the priority constraint as shown in Fig. 4.4d. In the end, the generated hybrid SFP is $A \rightarrow B \rightarrow C \rightarrow B \rightarrow D \dashrightarrow C \dashrightarrow A$, while the h-SFC is $s \rightarrow f1 \rightarrow f2 \rightarrow f3 \rightarrow f4 \dashrightarrow f3 \dashrightarrow s$.

4.3.5 EC-HSFP is 2-Approximation

Theorem 4.3.4. *EC-HSFP generates the hybrid SFP within a 2-approximation boundary of the optimal hybrid SFP.*

Proof. We denote the length of the forward and backward MSTs by |f-MST| and |b-MST|. The length of the hybrid SFP generated by EC-HSFP is represented as $|\text{SFP}_{\text{EC-HSFP}}|$, while the length of the optimal SFP is denoted by $|\text{h-SFP}_{\text{OPT}}|$. The lengths of the optimal f-SFP and b-SFP are represented as $|\text{f-SFP}_{\text{OPT}}|$ and $|\text{b-SFP}_{\text{OPT}}|$, respectively. The length of the hybrid trace is |HT|. Since MST is the least length connected structure for a connected graph, Eq. (4.19) and Eq. (4.20) hold.

$$|\text{f-MST}| \leq |\text{f-SFP}_{\text{OPT}}| \tag{4.19}$$

$$|\text{b-MST}| \leq |\text{b-SFP}_{\text{OPT}}| \tag{4.20}$$

Eq. (4.21) shows the relationship between the optimal hybrid SFP, f-SFP and b-SFP.

$$|\text{f-SFP}_{\text{OPT}}| + |\text{b-SFP}_{\text{OPT}}| \leq |\text{h-SFP}_{\text{OPT}}| \tag{4.21}$$

According to EC-HSFP, the length of the hybrid trace equals the length sum of f-MST and b-MST as shown in Eq. (4.22).

$$|\text{f-MST}| + |\text{b-MST}| = |\text{HT}| \tag{4.22}$$

After doubling the number of links in hybrid trace, the generated hybrid SFP is the Eulerian circuit of the hybrid trace. Therefore, Eq. (4.23) holds.

$$|\text{SFP}_{\text{EC-HSFP}}| = 2 * |\text{HT}| \tag{4.23}$$

When x is not s, one needs to delete two paths in the hybrid trace. Thus, from Eq. (4.19)-(4.23), we have Eq. (4.24).

$$|\text{SFP}_{\text{EC-HSFP}}| \leq 2 * |\text{h-SFP}_{\text{OPT}}| \tag{4.24}$$

Hence, EC-HSFP achieves a 2-approximation boundary. □

Lemma 4.3.5. *If the UFSN is a tree structure (e.g., fat tree), EC-HSFP generates the optimal hybrid SFP.*

4.4 Hybrid SFCE in MFSN

In this section, we extend EC-HSFP to the Multi-Functions Substrate Network (MFSN). The MFSN allows multiple SF nodes mapped onto the same substrate node (subject to the substrate node's computing capacity and the SF instance availability). Fig. 4.5 and Table 4.1

show an example of the MFSN, where the number beside the link represents the link latency. The NSR starts at node B and requires $f2$, $f3$ and $f4$, each of which demands 20 computing resource. Additionally, SFs $f2$ and $f4$ handle the traffic in both directions.

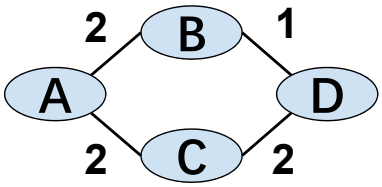


Figure 4.5: An example of MFSN

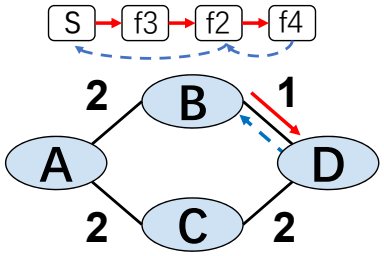


Figure 4.6: BC-HSFP result

Table 4.1: Multi-Functions Substrate Network

Substrate Node	A	B	C	D
Network Function	$f1, f2, f3$	$f2, f3$	$f3, f5$	$f1, f4$
Capacity	40	40	20	20

To map the SF nodes onto the appropriate substrate nodes in MFSN while reducing the latency cost; we propose an efficient heuristic algorithm called Betweenness Centrality based Hybrid SFP optimization (BC-HSFP), which includes the proposed Betweenness Centrality (BC) based node deployment approach and EC-HSFP.

Betweenness Centrality based Node Deployment: Traditionally, the Betweenness Centrality (BC_n) of a substrate node n can be calculated as $BC_n = \frac{path_n}{path_{total}}$, where $path_n$ and $path_{total}$ represent the amount of the shortest path(s) passing node n and the total number of the shortest path(s) in the graph, respectively. The more shortest paths passing a node, the higher probabilities that this node will connect with other substrate nodes via the shortest path rather than a longer detour. That is to say, a substrate node with the higher

BC value will more likely reduce the latency by connecting itself with the other substrate nodes that provide the requested SF instances via the shortest path.

However, this traditional BC technique only considers the connection links between different substrate nodes, which ignores the internal connections within a substrate node in MFSN. In specific, if more than one SF nodes are mapped onto the same substrate node, the connections between these SF instances can also be counted as the potential shortest path(s) passing this substrate node. For example, if $f2$ and $f3$ are embedded onto substrate node B in Fig. 4.5, then the connection between SF instances $f2$ and $f3$ within node B can also be part of the shortest paths. Here, we create a virtual inner connections between SF instances inside a substrate node, while the outer connection indicates the traditional shortest path passing the substrate node. Accordingly, we propose the Inner-connection-included Betweenness Centrality (IBC) to measure the importance of the substrate node in MFSN, which takes both the inner and outer shortest path connections into account. We specify the number of the inner shortest connections of a substrate node n as $path_{n_{in}}$, while the number of the outer paths is $path_{n_{out}}$. We denote the $\sum_{v \in V} |\{f_v\} \cap \mathbb{F}_n|$ by the number of SF nodes that matches the SF instances installed in the substrate node n and δ_n is the number of SF nodes that can be mapped onto n (limited by its computing capacity). Eq. (4.25) calculates $path_{n_{in}}$ while Eq. (4.26) calculates the IBC value of a substrate node n.

$$path_{n_{in}} = min(\sum_{v \in V} |f_v \cap \mathbb{F}_n|, \delta_n) - 1 \tag{4.25}$$

$$IBC_n = \frac{path_{n_{in}} + path_{n_{out}}}{path_{total}} \tag{4.26}$$

With the IBC technique, we propose Algorithm 7 to accommodate an NSR with hybrid traffic onto a shared MFSN. Note that, the substrate candidate of an SF node is the substrate node that can provide the corresponding SF instance.

Algorithm 7 Betweenness Centrality based Hybrid SFP optimization (BC-HSFP) Algorithm

1: **Input:** G, NSR;
2: **Output:** hybrid SFP and h-SFC;
3: Initialize the SFP list as an empty list;
4: Calculate the IBC value for substrate nodes that can provide at least one requested SF instance as in Eq. (4.26);
5: Sort SF nodes in ascending order according to the amount of substrate candidate(s);
6: Map the sorted SF node onto the substrate candidate with the highest IBC value;
7: Generate an induced subgraph G_{induce} including the substrate nodes that host at least one SF nodes;
8: Call EC-HSFP(G_{induce}, NSR) to find the hybrid SFP and h-SFC;
9: Return hybrid SFP and h-SFC;

We use Fig. 4.5 and 4.6 to illustrate how the BC-HSFP algorithm works. In Fig. 4.5, the NSR starts at node B and requires f2, f3 and f4, where f2 and f4 are demanded in both directions. First, the algorithm calculates the IBC value of each substrate node that may participate in the construction of the hybrid SFP as shown in Table 4.2. Then, the algorithm sorts the requested SFs as $\{f4, f2, f3\}$ according to the number of substrate candidate(s).

Table 4.2: IBC Value Calculation

Substrate Node	A	B	C	D
# of Inner Path	1	1	0	0
# of Forward Out Path	3	4	3	3
# of Backward Out Path	1	2	0	2
IBC Value	$\frac{5}{10}$	$\frac{7}{10}$	$\frac{3}{10}$	$\frac{5}{10}$

Based on the IBC values shown in Table 4.2 and the sorted SF nodes, the algorithm deploys $f4$ onto node D, and then $f2, f3$ onto node B. At last, the algorithm generates the induced graph of B and D in Fig. 4.6 and calls the EC-HSFP algorithm to form a hybrid SFP as $B \rightarrow D \dashrightarrow B$ with the h-SFC as $s \rightarrow f3 \rightarrow f2 \rightarrow f4 \dashrightarrow f2 \dashrightarrow s$. As shown in Fig. 4.6, the red lines and dashed blue lines represent the forward and backward SFP, respectively. As one can see in Algorithm 7, the process with the highest running time complexity is to calculate the IBC value for substrate nodes that can provide at least one required SF instance. In the worst case, when every substrate node can provide at least one required SF instance, the time complexity of this algorithm is $|N^2L + N^3logN|$.

4.5 Numerical Results and Analysis

In this section, we analyze and compare the performance of the proposed algorithms with the schemes that are directly extended from the state-of-art techniques [56, 57].

4.5.1 Simulation Environment

We use the 24-nodes US-NET as the Substrate Network (SN), which can be configured as the UFSN or MFSN. For the UFSN, each substrate node supports one unique SF instance, and each physical link has a latency in the range of $[1-5]$. For the MFSN, the number of SF instances supported by a substrate node is in the range of $[2-6]$, while the computing capacity is in the range of $[20-100]$. The link latency in MFSN is in the range of $[1-5]$ and has the bandwidth in the range of $[5-20]$. We set the number of required SF nodes in each NSR within the range of $[4-17]$, while the number of required bidirectional SF nodes

is in the range of $[4-13]$. For each SF node, the required computing capacity is in the range of $[10-20]$. The source node is randomly generated and the bandwidth demand is in the range of $[1-10]$.

In [56], the authors proposed the algorithm to map a given unidirectional SFC onto a shared SN with the shortest length when the bandwidth is abundant. We extend the algorithm in [56] as Shortest Path based HSFP optimization ("SP-HSFP") by generating five random hybrid SFCs, creating five corresponding hybrid SFPs and calculating the average latency. In [57], the authors utilized the Closeness-Centrality (CC) technique in the node mapping process, which aims to determine the substrate candidate with the least average latency cost to connect with others. We combine this CC node mapping with EC-HSFP as "CC-HSFP".

4.5.2 Performance Metrics

We use the following metrics to assess the performance of the proposed algorithms.

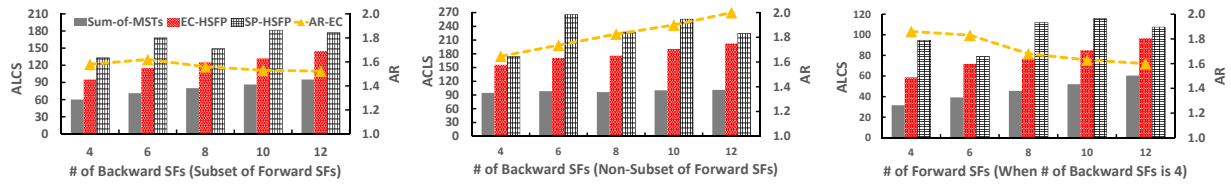
Approximation Ratio (AR): To evaluate the performance of EC-HSFP and SP-HSFP algorithms, we compare their results with the length of the Minimum Spanning Tree (MST), which is proved as the lower boundary in Eq. (4.19) and (4.20). AR can be calculated as $AR = \frac{SFP_{created}}{|MST|}$, where the $SFP_{created}$ represents the latency of the hybrid SFP created by the proposed algorithms and $|MST|$ represents the length sum of the forward MST and backward MST.

Average Latency of the Created SFP (ALCS): $ALCS = \frac{\sum_{NSR_i \in NSR_C} SFP_{created}}{|NSR_C|}$, where NSR_C represents a set of NSRs, and NSR_i is the i_{th} NSR in NSR_C.

4.5.3 Approximation Analysis in UFSN

We evaluate the approximate performance of EC-HSFP under three types of NSR (i.e., Fig. 4.7a, 4.7b, 4.7c). In Fig. 4.7a, 12 forward SFs are required and the backward SFs are a subset of the forward SFs (i.e., $V_b \subseteq V_f$). In Fig. 4.7b, 12 forward and backward SFs are required, while the number of backward SFs that does not belong to the forward SFs varies. In Fig. 4.7c, 4 backward SFs are required and the number of forward SFs varies. In Fig. 4.7, the grey, red and dark gridded bars represent the ALCS of MSTs, EC-HSFP and SP-HSFP, respectively. The yellow dashed curve denotes the AR for EC-HSFP.

When increasing the number of backward SFs that belong to the set of required forward SFs in Fig. 4.7a, EC-HSFP and MSTs need more latency to finish the transmission but the latency required by SP-HSFP fluctuates. This is because the performance of the SP-HSFP totally depends on the given SFCs. Due to the given SFCs, SP-HSFP may take multiple detours leading to fluctuated latency for the constructed hybrid SFP. For EC-HSFP, the AR is under 2 in any situation, which matches Theorem 4.3.4. It is worth noting that, when the number of backward SFs is small (e.g., 4), the lengths of the deleted paths (i.e., $s \rightarrow x$ forward path and $x \rightarrow s$ backward path) may not contribute much to the total latency.



(a) Backward SFs $\subseteq$ Forward SFs (b) Backward SFs $\not\subset$ Forward SFs (c) Fixed # of Backward SFs

Figure 4.7: Approximation analysis in UFSN

Thus, the AR is relatively high when the number of backward SFs is in the range of $[4, 6]$. However, when further increasing the number of backward SFs, there is a higher probability that the $x \to s$ forward path and $s \to x$ backward path become longer. Particularly, when the number of backward SFs is bigger than 10, the deleted paths most likely are the longest one in the generated MSTs. Therefore, the AR is relatively low and the proposed EC-HSFP performs even better when the number of backward SFs is larger than 6.

When the number of backward SFs increases in Fig. 4.7b, EC-HSFP needs more latency while the sum lengths of MSTs do not vary much. Again, since the performance of SP-HSFP totally depends on the constructed SFCs, it is unstable in UFSN. It is worth noting that, when the number of backward SFs that does not belong to the set of forward SFs equals 12, the AR is 2. This is because the only common substrate node that the forward MST and backward MST share is the source node; thus no path will be deleted, whereas $|\text{SFP}_{\text{EC-HSFP}}| = 2 * |HT| = 2 * (|\text{f-MST}| + |\text{b-MST}|)$.

In Fig. 4.7c, when increasing the number of forward SFs, EC-HSFP and the sum lengths of MSTs increase. When the number of forward SFs is small (e.g., 4), the probability that the backward SFs belong to the forward SFs is small. Thus, the common node that the forward and backward MSTs share likely is the source node, which results in a high AR. When increasing the number of forward SFs, the probability that the backward SFs are included in the forward SFs increases, which reduces the latency of the constructed hybrid SFP by potentially increasing the length of the deleted paths.

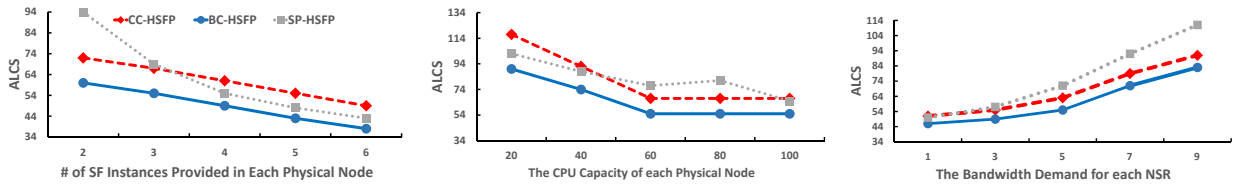
Overall, the AR is no more than 2 in any situation, which verifies that the proposed

EC-HSFP algorithm guarantees the 2-approximation performance.

4.5.4 Performance Analysis in MFSN

Fig. 4.8 shows the performance of CC-HSFP, BC-HSFP and SP-HSFP in MFSNs. In Fig. 4.8, the red dashed curve, grey dotted curve and blue solid curve represent the performance of CC-HSFP, SP-HSFP and BC-HSFP, respectively.

Fig. 4.8a demonstrates that the latency from all three schemes decreases when increasing the number of SF instances in each substrate node. This is because the probability to generate the inner connection becomes larger when the SF instances provided in each substrate node are more. Note that, BC-HSFP outperforms both SP-HSFP and CC-HSFP. This is because BC-HSFP can jointly optimize SFC composition and embedding, whereas SP-HSFP is limited by the given SFCs. The CC technique in CC-HSFP will likely create the MST as a "star". This is because the central node of a graph has the highest CC value, which implies it is the nearest one to other substrate nodes that host the required SFs. Thus, the MST is likely to be created by connecting the central node with the other substrate nodes via the shortest path, which is a "star". However, from the experimental results, the BC technique will likely create the MST as a "path". Fig. 4.9 and 4.10 show an example of the



(a) # of SF instances (b) # of computing resource (c) # of bandwidth demand

Figure 4.8: Performance analysis in MFSN

difference between "star" and "path" topologies. For a star MST in Fig. 4.9, the process in Line 8-9 of Algorithm 6 will delete the paths $A \rightarrow D$ and $D \dashrightarrow A$, removing 4 hops out of final hybrid SFP. However, when a path MST is constructed by BC as shown in Fig. 4.10, the process in Line 8-9 of Algorithm 5 will remove 8 hops, resulting in a shorter hybrid SFP. This difference can be even larger in an SN with more nodes.

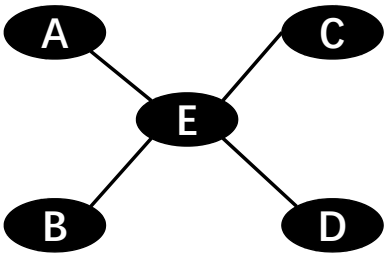


Figure 4.9: A star MST from CC

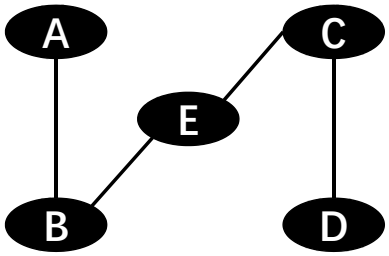


Figure 4.10: A path MST from BC

When the number of SF instances provided by each substrate node is 3, Fig. 4.8b demonstrates that the more computing resource provided by each substrate node, the less latency is required for all algorithms. The performance of BC-HSFP and CC-HSFP is flat when the computing resource provided by each substrate node is larger than 60. This is because 60 computing capacity will allow that all SF instances are available in each substrate node. Thus, further increasing the number of computing capacity does not change the performance of CC-HSFP and BC-HSFP.

Fig. 4.8c shows the performance of the proposed algorithms when varying the number of bandwidth demand. As one can see that, the more bandwidth requested by the NSR, the more latency is required. This is because when increasing the bandwidth demand, some links become unavailable, and a longer path may have to be employed. Note that, CC-HSFP and

BC-HSFP increase slower than SP-HSFP. This is because, as a joint h-SFC composition and embedding process, the routing technique (i.e., EC-HSFP) of BC-HSFP/CC-HSFP does not introduce multi-detours that bring large latency in a bandwidth resource-limited network.

Overall, the proposed BC-HSFP algorithm averagely outperforms CC-HSFP by 20% and outperforms SP-HSFP as much as 50%.

4.6 Summary

In this chapter, for the first time, we have comprehensively studied a new set of *Hybrid SFC composition and Embedding* (HSFCE) problems in different network scenarios. When the computing capacity provided by the Substrate Network (SN) is limited, we have investigated the Unique Function SN (UFSN), where each substrate node only provides one unique type of SF. We have proved the NP-hardness of HSFCE in UFSN and proposed a 2-approximation algorithm to jointly composite and embed a hybrid SFC, called Eulerian Circuit based Hybrid SFP optimization (EC-HSFP). We have also studied HSFCE in Multi-Functions SN (MFSN), where each substrate node provides various SFs, and extended the EC-HSFP with the betweenness centrality technique to optimize HSFCE in MFSN. Our extensive analysis and simulation results have shown that the EC-HSFP algorithm can guarantee the 2-approximation boundary, and the proposed BC-HSFP outperforms the algorithms directly extended from the state-of-art techniques by an average of 20%. In the next chapter, we will further investigate how to optimally embed a given h-SFC onto a shared substrate network.

CHAPTER 5
OPTIMAL HYBRID SERVICE CHAIN EMBEDDING

5.1 Motivation

For many 5G and Internet of Things (IoT) applications such as cloud gaming and on-line machine learning [7][58][59], the forward traffic from the customer is processed at the edge server/cloud, and the backward traffic including the results/models will be transmitted back from the edge server/cloud. As the contents of the forward and backward traffic are different, different SFs may be requested to process the forward and backward traffic. An SFC that requests different sets of SFs for the forward and backward traffic is referred to as hybrid SFC (h-SFC) [14]. Fig. 5.1 depicts an example of an in-service h-SFC of on-line machine learning. The data collected by the IoT devices will be sent to the cloud for further processing after going through the Deep Packet Inspection (DPI) and Firewall (FW). The backward traffic including the machine learning model has to go through the Encryption (Encry), FW,

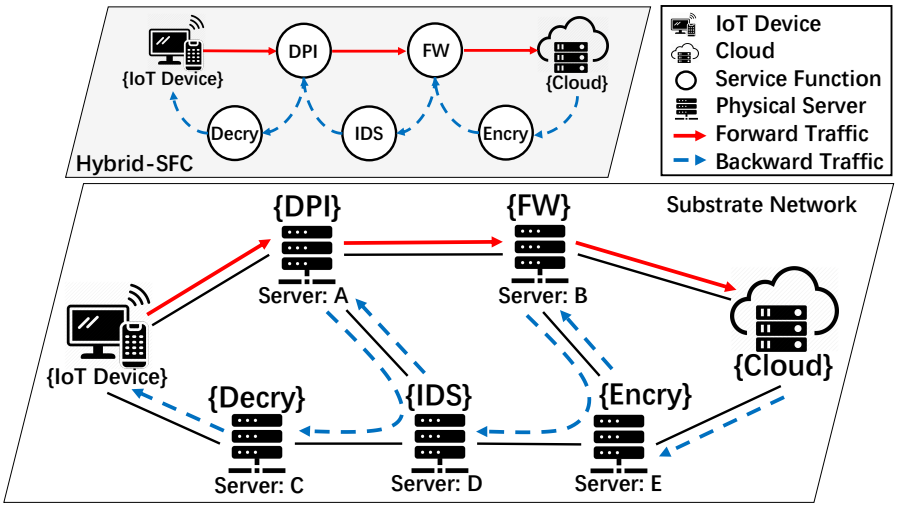


Figure 5.1: An example of in-service h-SFC for on-line machine learning in IoT

Intrusion Detection System (IDS), DPI and Decryption (Decry) before arriving at the IoT devices. In the SN, the forward SFP is constructed as IoT device $\rightarrow$ DPI $\rightarrow$ FW $\rightarrow$ Cloud, while the backward SFP includes Cloud $\dashrightarrow$ Encry $\dashrightarrow$ FW $\dashrightarrow$ IDS $\dashrightarrow$ DPI $\dashrightarrow$ Decry $\dashrightarrow$ IoT Device. For the sake of saving the Operational Expense (OPEX) and latency, the SF that is required along both forward and backward directions are generally instantiated on the same substrate node (e.g., FW and DPI in Fig. 5.1) [52][53].

As MEC related applications (e.g., cloud gaming, on-line AR/VR games) generally require real-time interactions with the customer, in this paper, we study how to embed a given h-SFC onto a shared SN with minimum latency. We define a new problem called Minimum Latency Hybrid SFC Embedding (ML-HSFCE), which is different from our previous work that focuses on hybrid SFC optimization in a special substrate network [27]. In specific, the work in [27] proposed a 2-approximation algorithm to composite and embed an h-SFC onto a substrate network, whereas each substrate node provides only one unique SF instance. In this work, we mathematically model the problem of ML-HSFCE in a generic substrate network, whereas multiple SF instances may be available at one substrate node. We propose an Optimal Hybrid Service Function Chain Embedding (Opt-HSFCE) algorithm to embed a given hybrid SFC onto any substrate network optimally. Opt-HSFCE employs a novel technique of Hybrid SFC embedding Auxiliary Graph (HSAG). Through extensive analysis and simulations, we show the efficiency and effectiveness of the proposed Opt-HSFCE algorithm.

The rest of this chapter is organized as follows. Section II mathematically formulates the HSFCE problem. In Section III and IV, we analyze the complexity of ML-HSFCE and

develop the algorithm to optimize ML-HSFCE. Section V demonstrates the experimental results and analysis. We summarize this chapter in Section VI.

5.2 Minimum Latency Hybrid Service Function Chain Embedding

5.2.1 Substrate/physical Network (SN) Model

We denote the Substrate/physical Network (SN) by an undirected graph $G = (N, E, F)$, where N is the set of substrate nodes, E represents the set of substrate links, and F denotes a set of available SF instances. For each substrate node $n \in N$, it provides a specific set of SF instances $\mathbb{F}_n$ ($\mathbb{F}_n \subseteq F$) and a certain amount of available computing resource c_n. Each link $l_{m,n} \in E$ ($\forall m, n \in N$) has a specific amount of bandwidth resource $bw_{l_{m,n}}$ and a certain amount of latency $la_{l_{m,n}}$. For a path $path_{m,n}$, it has an accumulative latency cost $La_{path_{m,n}}$ as shown in Eq. (5.1) and a bottleneck bandwidth $bw_{path_{m,n}}$ calculated as Eq. (5.2).

$$La_{path_{m,n}} = \sum_{l_{a,b} \in path_{m,n}} la_{l_{a,b}}, \forall a, b, m, n \in N \tag{5.1}$$

$$bw_{path_{m,n}} = min(bw_{l_{a,b}}), \forall l_{a,b} \in path_{m,n}, \forall a, b, m, n \in N \tag{5.2}$$

5.2.2 Hybrid Service function chain Request (HSR)

A Hybrid Service function chain Request (HSR) can be represented as a 2-tuple $HSR =$ <BW, h-SFC>, where BW denotes by the set of bandwidth demands between two adjacent SFs, while h-SFC represents the hybrid SFC. We denote an h-SFC as a set of sequential SFs $\{s, f_{f_1}, ..., f_{f_i}, d, b_{f_1}, ..., b_{f_j}, s\}$. For a specific SF ($f_{f_i}$ or b_{f_j} $\in$ h-SFC), it demands a certain

amount of computing resource ($c_{f_{f_i}}$ or $c_{b_{f_j}}$) to instantiate the corresponding SF at a substrate node. We further use v_i to represent the i_{th} SF in the h-SFC, c_{v_i} to denote its computing demands, and $bw_{v_i,v_{i+1}}$ to denote the bandwidth demand between two adjacent SFs (i.e., v_i and v_{i+1}).

5.2.3 Minimum Latency Hybrid SFC Embedding (ML-HSFCE)

The optimization problem of ML-HSFCE is defined as: given an SN with abundant bandwidth resource and an HSR, how to accommodate the HSR onto the SN such that i) the following constraints are satisfied, and ii) the latency of the constructed SFP is minimized. The objective function is shown in Eq. (5.3), where $\mu_{path_{m,n}^{v_i,v_{i+1}}}$ represents the latency of the $path_{m,n}$ to support the traffic from SF node v_i to v_{i+1}. Note that, in Eq. (5.3), χ is the sequential number of the last SF in the h-SFC. Since the hybrid traffic starts and ends at the source node, both v_0 and v_χ represent the source node. Table 5.1 describes the notations for the variables.

$$min \sum_{i=0}^{i=\chi} \sum_{n \in N} \sum_{m \in N} \mu_{path_{m,n}^{v_i,v_{i+1}}} \tag{5.3}$$

SF node mapping constraint: Eq. (5.4) represents whether an SF node v_i is mapped onto the substrate node n and Eq. (5.5) denotes whether the substrate node n provides the SF instance for SF node v_i. In Eq. (5.6), each SF node must be embedded onto one substrate node. Eq. (5.7) ensures that an SF node can only be embedded onto the substrate node with the corresponding SF instance. Each substrate node can host a limited number

Table 5.1: Notation Table

Notation	Meaning
m, n	Substrate nodes $m, n \in N$
v_i	The i_{th} requested SF in the h-SFC
BW	Requested bandwidth resource
$M_n^{v_i}$	=1 when v_i is mapped on n; 0 otherwise
$\Delta_n^{v_i}$	=1 when n provides SF instance for v_i; 0 otherwise
c_{v_i}	Computing demand of v_i
c_n	Computing capacity of n
$path_{m,n}^{v_i,v_{i+1}}$	=1 when $path_{m,n}$ supports the traffic from v_i to v_{i+1}; 0 otherwise
$bw_{path_{m,n}}$	Bottleneck bandwidth of $path_{m,n}$
$La_{path_{m,n}}$	Latency cost of $path_{m,n}$
$\mu_{path_{m,n}^{v_i,v_{i+1}}}$	Total latency cost of $path_{m,n}$

of SF nodes due to the computing capacity as shown in Eq. (5.8). Eq. (5.9) describes v_0 and v_χ are embedded onto the source node.

$$M_n^{v_i} = \begin{cases} 1, & \text{SF node } v_i \in \text{h-SFC is mapped onto} \\ & \text{substrate node } n \in N \\ 0, & \text{otherwise} \end{cases} \tag{5.4}$$

$$\Delta_n^{v_i} = \begin{cases} 1, & \text{substrate node } n \text{ provides} \\ & \text{SF instance for } v_i \\ 0, & \text{otherwise} \end{cases} \tag{5.5}$$

$$\sum_{n \in N} M_n^{v_i} = 1, \quad \forall v_i \in \text{h-SFC} \tag{5.6}$$

$$M_n^{v_i} \leq \Delta_n^{v_i}, \quad \forall n \in N, \forall v_i \in \text{h-SFC} \tag{5.7}$$

$$\sum_{v_i \in \text{h-SFC}} M_n^{v_i} * c_{v_i} \leq c_n, \quad \forall n \in N \tag{5.8}$$

$$M_s^{v_0} = M_s^{v_\chi} = 1, \tag{5.9}$$

hybrid SFP construction constraint: We use $path_{m,n}^{v_i,v_{i+1}}$ to denote whether the path from m to n is used to support the traffic from SF node v_i to v_{i+1} as shown in Eq. (5.10). Eq. (5.11) ensures that only the path whose endpoints host the SFs will be counted to construct the hybrid SFP. Eq. (5.12) specifies that each SF link is accommodated by one substrate path/link. In Eq. (5.13), if $path_{m,n}$ is used to support the traffic from v_i to v_{i+1}, the bandwidth of each link in the path should be more than $bw_{v_i,v_{i+1}}$. Eq. (5.14) calculates the latency cost of a path that supports the traffic for the SF link.

$$path_{m,n}^{v_i,v_{i+1}} = \begin{cases} 1, & \text{path from node } m \text{ to } n \text{ is used to support} \\ & \text{the traffic from SF node } v_i \text{ to } v_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{5.10}$$

$$path_{m,n}^{v_i,v_{i+1}} \leq \frac{M_m^{v_i} + M_n^{v_{i+1}}}{2}, \forall m, n \in N, \forall v_i, v_{i+1} \in \text{h-SFC} \tag{5.11}$$

$$\sum_{m,n \in N} path_{m,n}^{v_i,v_{i+1}} = 1, \forall v_i, v_{i+1} \in \text{h-SFC} \tag{5.12}$$

$$path_{m,n}^{v_i,v_{i+1}} * bw_{v_i,v_{i+1}} \leq bw_{path_{m,n}}, \forall m, n \in N, \forall v_i, v_{i+1} \in \text{h-SFC} \tag{5.13}$$

$$\mu_{path_{m,n}^{v_i,v_{i+1}}} = path_{m,n}^{v_i,v_{i+1}} * La_{path_{m,n}}, \forall m,n \in N, \forall v_i, v_{i+1} \in \text{h-SFC} \tag{5.14}$$

5.3 Hybrid SFCE Complexity Analysis

In this section, we analyze the complexity of hybrid SFCE by comparing it with the traditional Virtual Network Embedding (VNE) and the unidirectional SFCE.

VNE is a well-known NP-hard problem, which requires the service provider to embed a virtual mesh network onto the substrate network such that a specific goal (e.g., minimize the bandwidth usage) can be achieved [60, 61, 62]. For an SFC, it can be regarded as a special virtual mesh network, whereas the topology of the virtual network is linear. When the bandwidth resource is limited in an SN, embedding a given SFC is still NP-hard since there exists bandwidth competition among virtual links (i.e., links between adjacent SFs) [63]. However, embedding a given SFC onto an SN with abundant bandwidth resource (i.e., no bandwidth competition) is not NP-hard [56][46]. The authors in [56] and [46] have proposed the optimal schemes that construct an auxiliary graph to embed a given unidirectional SFC onto the SN by utilizing the linearity of the SFC. The optimal unidirectional SFP is then equivalent to the shortest path that connects the source and the destination in the composited auxiliary graphs. Similarly, embedding a given h-SFC onto an SN with abundant bandwidth resource can be optimized through the techniques of graph transforming and the shortest path algorithms as shown in the next section.

Next, we discuss whether the optimal unidirectional SFCE techniques [56] [46] can be ap-

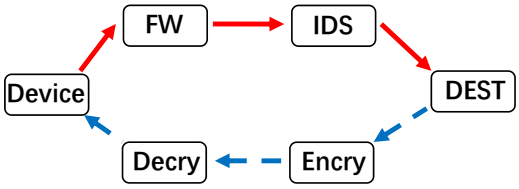


Figure 5.2: Traffic-Independent h-SFC

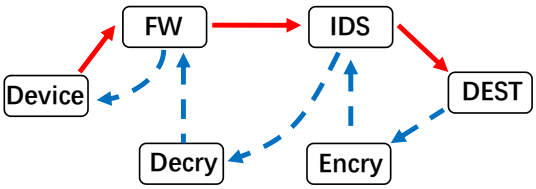


Figure 5.3: Traffic-Dependent h-SFC

plied to solve the ML-HSFCE problem when the bandwidth resource is abundant. According to whether there are any common SFs in the forward and backward SFCs, the h-SFC can be either i) traffic-independent h-SFC or ii) traffic-dependent h-SFC. For the former one, the forward and backward SFCs request different SFs (i.e., f-SFC $\cap$ b-SFC $= \emptyset$). For the latter one, there exists common SFs in both SFCs (i.e., f-SFC $\cap$ b-SFC $\neq \emptyset$). We use the examples in Fig. 5.2 and Fig. 5.3 to demonstrate the traffic-independent h-SFC and traffic-dependent h-SFC. Fig. 5.2 demonstrates the traffic-independent h-SFC. For the forward traffic, FW and IDS are requested, while the backward traffic demands Encry and Decry. Since no common SFs are requested in two traffic directions, the traffic-independent h-SFC can be optimized as two independent unidirectional SFCs that share the same endpoints. For a traffic-dependent h-SFC shown in Fig. 5.3, FW and IDS are requested in both traffic directions. That is to say, the substrate nodes that host FW and IDS will be visited twice by the hybrid SFP (one in the forward SFP, and the other one in the backward SFP). Since both forward and backward traffic visits the substrate nodes that host the common SFs (e.g., FW and IDS in Fig. 5.3), the traffic-dependent h-SFC cannot be treated as two independent unidirectional SFCs. Based on the property of the traffic-independent h-SFC, the following lemma holds.

Lemma 5.3.1. *The existing SFCE optimization techniques can create the optimal hybrid SFP for a traffic-independent h-SFC.*

Proof. Since the forward and backward traffic requests totally different sets of SFs, the only nodes sharing by both directions are the source and destination nodes. That is to say; the optimal hybrid SFP is a concatenation of the optimal forward SFP and the optimal backward SFP. Therefore, the existing SFC optimization techniques can construct the optimal hybrid SFP for a traffic-independent h-SFC by optimally constructing optimal forward and backward SFPs (as the work in [56] or [46]). □

However, for a traffic-dependent h-SFC, the existing SFCE related techniques cannot be directly applied. This is because if one generates the optimal unidirectional SFP (forward or backward) according to the existing SFCE related techniques and applies the SF node mapping results of the forward SFP for the other traffic direction, the generated hybrid SFP may not be optimal. For example, Fig. 5.4 shows a given SN with enough networking resource (i.e., computing and bandwidth resource), whereas the number beside each link represents the latency cost and the set of SF instances are beside each node. When considering a h-SFC request as $\{A, v_1, v_3, v_4, v_2, v_5, F, v_1, v_3, v_4, A\}$, Fig. 5.5 and Fig. 5.6 shows the optimal hybrid SFP and the hybrid SFP created by applying the technique that directly exetends from the work in [56], [46].

Fig. 5.5 illustrates the optimal hybrid SFP that accommodates the requested h-SFC. SF node v_1 is mapped onto node B, v_3 and v_4 are mapped onto node C, while v_2 and v_5 are mapped onto nodes D and F, respectively. In the forward traffic, the forward SFP employs

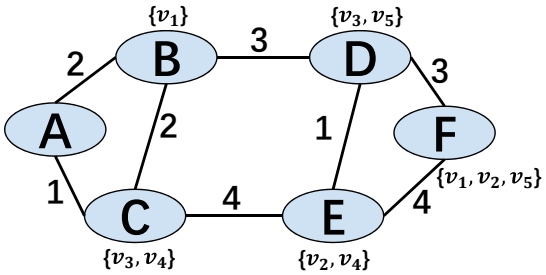


Figure 5.4: A substrate network example

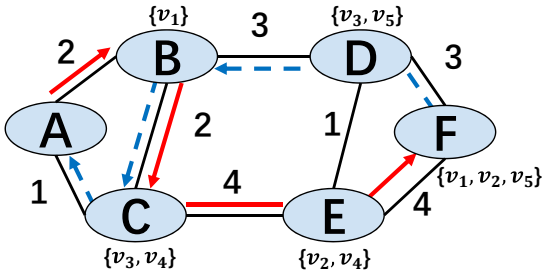


Figure 5.5: The optimal hybrid SFP (latency cost = 21)

the path $A \rightarrow B \rightarrow C \rightarrow E \rightarrow F$ with the length as 12. For the backward traffic, the backward SFP is $F \dashrightarrow D \dashrightarrow B \dashrightarrow C \dashrightarrow A$ with the length of 9. Overall, the optimal hybrid SFP has the latency cost as 21.

Fig. 5.6 demonstrates the hybrid SFP that is generated by i) applying the technique in [46, 56] to embed the f-SFC optimally; and ii) using the common SF nodes embedding results to construct the backward SFP. Accordingly, SF node v_1 is embedded onto node B, v_3 is embedded onto node D, v_2 and v_4 are embedded onto node E, while v_5 is embedded onto node F. The forward SFP is $A \rightarrow B \rightarrow D \rightarrow E \rightarrow F$ with the length of 10, while the backward traffic is $F \dashrightarrow D \dashrightarrow B \dashrightarrow D \dashrightarrow E \dashrightarrow C \dashrightarrow A$, with the length of 15.

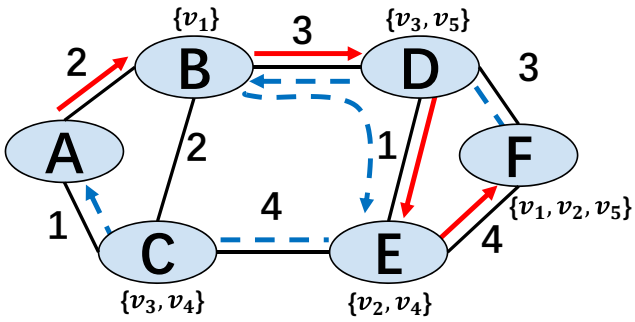


Figure 5.6: The hybrid SFP created by applying the existing SFCE optimization technique on f-SFC and applying the common SF node embedding result on the embedding process of b-SFC (latency cost = 25)

Even though the forward SFP constructed in this way has less length than the forward SFP in Fig. 5.5, the length of the final hybrid SFP generated in this method is larger than the optimal result in Fig. 5.5.

As one can see that, even though the f-SFC is optimally embedded by applying the existing SFCE related techniques, the hybrid SFP generated upon the forward SF node mapping results may not be optimal. In the following sections, we propose the Optimal Hybrid SFC Embedding (Opt-HSFCE) algorithm and the technique of Hybrid SFC embedding Auxiliary Graph (HSAG).

5.4 Optimal Hybrid Service Function Chain Embedding

In this section, when the SN provides abundant bandwidth resource, we propose an Optimal Hybrid Service Function Chain Embedding (Opt-HSFCE) algorithm, which creates the optimal hybrid SFP for a given h-SFC based on the technique of Hybrid SFC embedding Auxiliary Graph (HSAG).

5.4.1 Hybrid SFC embedding Auxiliary Graph (HSAG)

Based on the analysis above, a hybrid SFP needs to meet two constraints: i) executing-order constraint, which requires the SFP visiting the SF instances as the same order they appear in the h-SFC, and ii) re-visit constraint that requires the SF(s) appeared in both forward and backward SFCs will be embedded onto the same substrate node. When an h-SFC is given in an HSR, the executing-order of all SFs is strictly specified. The optimization goal of ML-HSFCE turns out to be finding the shortest SFP, whereas each SF in the HSR is em-

bedded onto a substrate node along the SFP and the re-visit constraint is enforced. Clearly, the traditional shortest path algorithms cannot be directly applied due to this re-visit constraint. A naive brutal-force searching scheme can be time-consuming or even intractable for a large system. In the following, we propose to create an auxiliary graph called Hybrid SFC embedding Auxiliary Graph (HSAG), which effectively takes into account the properties of the given h-SFC and available substrate network resource to facilitate the h-SFC embedding process.

A composited HSAG includes three components i) tier, ii) layer and iii) nodes. As shown in Fig. 5.7, each tier includes at least one layer, and a layer consists of at least one node. Each tier in the HSAG matches with one SF node in the given h-SFC. In other words, the number of tiers in the HSAG equals the number of SF nodes in the h-SFC. Specifically, a composited HSAG can be represented as HSAG $= \{T_1, T_2, ..., T_\delta\}$, where T_i represents the i_{th} tier in the HSAG and δ is the length of the h-SFC. Each tier in HSAG matches with one SF node in the given h-SFC. Note that, T_1 and T_δ only include the source node. We use s and s' to specify the source node in T_1 and T_δ, respectively. As shown in Fig. 5.7, for a tier T_i, it only has direct connections with the adjacent tiers (T_{i-1} and T_{i+1}), which guarantees that a flow from T_1 to T_δ must visit each tier in order. Any path starting from T_1 and ending with T_δ will satisfy the *executing-order constraint*. For tier T_i, it includes at least one layer, which can be represented as $T_i = \{L_1^{T_i}, L_2^{T_i}, ..., L_m^{T_i}\}$. For layer $L_n^{T_i}$ ($n \in [1, m]$), it can be either the set of substrate candidates of v_i or a specific substrate candidate of v_i. For simplicity, we use $SC_{v_i}^{i,n}$ and $sc_k^{i,n}$ to represent the set of substrate candidates and the

k_{th} substrate candidate of v_i in the layer n of the tier i, respectively. An index number v_ik in layer L_n^T indicates that any layer created from L_n^T will use the k_{th} substrate candidate for hosting v_i, which guarantees the *re-visit constraint*.

Table 5.2: Term & Abbreviation Table

Abbreviation	Meaning
T_i	the i_{th} tier
T_d	destination tier
$L_n^{T_i}$	the n_{th} layer in T_i
$sc_k^{i,n}$	the k_{th} substrate candidate of v_i in $L_n^{T_i}$ of T_i
$SC_{v_i}^{i,n}$	the set of substrate candidates of v_i in $L_n^{T_i}$ of T_i
v_ik	index number for the k_{th} substrate candidate of v_i

HSAG for the forward traffic: The forward traffic starts at the source node located at T_1. For every two adjacent tiers T_i and T_{i+1}, according to whether T_i is created for a common SF, there are two different options to create the HSAG.

When T_i is created for a common SF, for each node $sc_k^{i,n}$ in the layer $L_n^{T_i}$, one needs to create a layer $L_m^{T_{i+1}}$ in T_{i+1} to match the re-visit constraint. Each newly created layer $L_m^{T_{i+1}}$ includes the set of substrate candidates ($SC_{v_{i+1}}^{i+1,m}$) for v_{i+1} and extends the index numbers from $L_n^{T_i}$. An index number v_ik is then added to the newly created layer $L_m^{T_{i+1}}$, which indicates that the traffic passes $L_m^{T_{i+1}}$ will re-visit the k_{th} substrate candidate of v_i in the backward traffic. Next, the $sc_k^{i,n}$ connects with all nodes in the newly created layer $L_m^{T_{i+1}}$ by using the shortest path in the SN. Note that, when T_i is created for a common SF, the number of layers in T_{i+1} equals the number of nodes in T_i.

When T_i is created for a non-common SF, for each layer $L_n^{T_i}$ in T_i, a layer $L_n^{T_{i+1}}$ that includes $SC_{v_{i+1}}^{i+1,n}$ and extends the index numbers from $L_n^{T_i}$ is created in T_{i+1}. Then, a complete bipartite graph is created between $L_n^{T_i}$ and $L_n^{T_{i+1}}$ [55], whereas each node in $L_n^{T_i}$ connects with

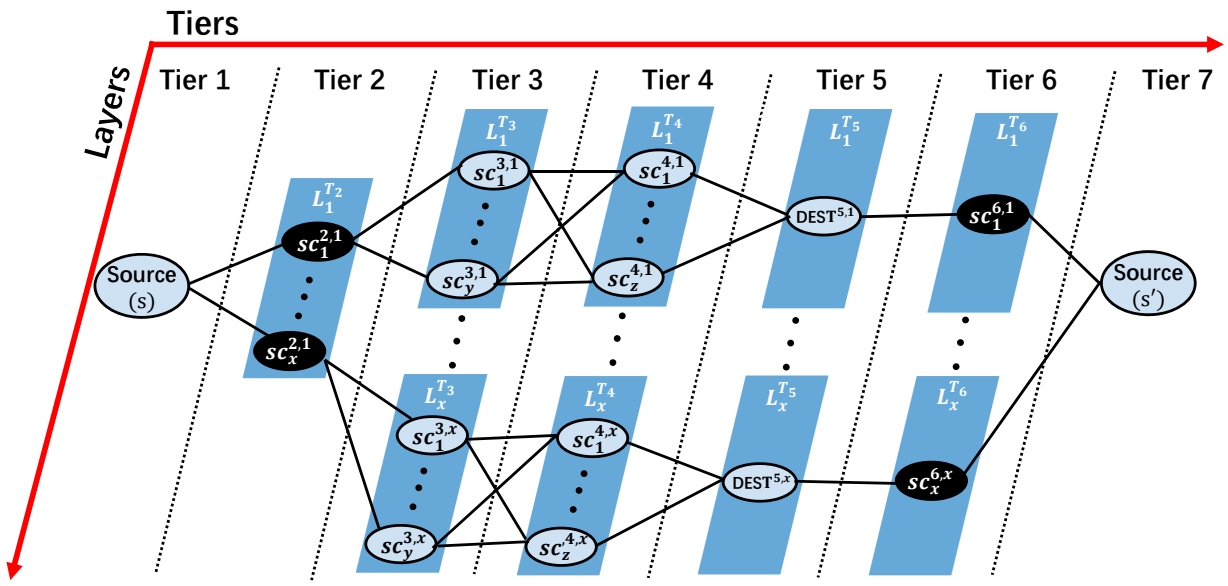


Figure 5.7: Hybrid SFC embedding auxiliary graph for the given h-SFC $\{s, v_1, v_2, v_3, d, v_1, s'\}$.

all nodes in $L_n^{T_{i+1}}$; but there is no connection between any pair of nodes in the same layer. Note that, since each layer in T_i matches a layer in T_{i+1}, the number of layers in T_i equals the number of layers in T_{i+1} when T_i is created for a non-common SF.

As the destination tier is the last tier in the forward traffic, the number of layers in the destination tier equals the product of the number of substrate candidates for all common SFs. For example, if there are three common SFs exists in the h-SFC whose numbers of substrate candidates will be x, y and z, there will be $x * y * z$ layers in the destination tier. For each layer in the destination tier, it includes a unique set of index numbers to indicate that the forward traffic ending at this destination layer visits a specific set of substrate candidates for the common SFs.

HSAG for the backward traffic: The backward traffic starts with the destination tier. Since each destination layer includes a unique set of index numbers that indicate which

substrate candidate needs to be re-visited for every common SF, the HSAG for the backward traffic is created from each destination layer. For a specific destination layer $L_n^{T_d}$, there are two options to create the HSAG according to whether T_{i+1} is for a common SF.

If T_{i+1} is created for a common SF, for each layer $L_n^{T_i}$ in T_i, a layer $L_n^{T_{i+1}}$ is created only with the k_{th} substrate candidate of v_{i+1} according to the set of index numbers of $L_n^{T_d}$. Then, all nodes in $L_n^{T_i}$ connect with the newly created node in $L_n^{T_{i+1}}$.

When T_{i+1} is created for a non-common SF, for each layer $L_n^{T_i}$, a layer $L_n^{T_{i+1}}$ is created with all substrate candidates of v_{i+1}. Next, a complete bipartite graph is formulated between $L_n^{T_i}$ and $L_n^{T_{i+1}}$.

Note that, since the HSAG for the backward traffic is created from each destination layer. Thus, all tiers (other than the last tier) in the backward HSAG own the same number of layers as the destination tier. With a composited HSAG, any path starting at s and ending at s' is a potential hybrid SFP.

Lemma 5.4.1. *Any path connects the source node s to the source node s' in the HSAG satisfies executing-order constraint.*

Proof. In the HSAG, the number of tiers matches the number of SFs in the h-SFC. For tier T_i, it only connects with the adjacent tiers (T_{i-1} and T_{i+1}). There is no connection between any two non-adjacent tiers. That is to say, in the HSAG, any path starting at the source node s and ending with the source node s' has to go through each tier in the order as the corresponding SF appears in the h-SFC. Thus, any path connecting the source node s and s' in the HSAG obeys the executing-order constraint. □

Lemma 5.4.2. *Any path connects the source node s to s' in the HSAG satisfies re-visit constraint.*

Proof. In HSAG, for a specific hybrid SFP, it can be regarded as two SFPs (forward SFP and backward SFP) that share the same endpoints. Each destination layer owns a unique set of index numbers that indicates the substrate candidate for each common SF. Thus, the forward and backward SFPs of a hybrid SFP will visit the same set of substrate candidates for the common SFs. The re-visit constraint is guaranteed for any hybrid SFP in the HSAG. □

Lemma 5.4.3. *The paths from the source node s to the source node s' in the HSAG include all possible hybrid SFPs.*

Proof. The index numbers of all layers include all possible combinations of the visiting substrate candidates for common SFs. Thus, to prove all possible hybrid SFPs are included in the HSAG, we need to prove that all possible combinations of visiting substrate candidates for non-common SFs are also included. For the tier T_i that is not created for a common SF, T_{i-1} is: i) not created for a common SF; or ii) created for a common SF. For the former case, there exists a complete bipartite graph formed by each layer of T_{i-1} and a layer in T_i. Thus, all substrate nodes of v_i are visitable by any path starting at the source. For the latter case, each node in T_{i-1} connects with a layer including the set of substrate candidates of v_i in T_i. Since the substrate candidates of all common SFs are visitable from the source, all substrate candidates for the non-common SF v_i are also visitable from the source. Thus, the HSAG includes all possible hybrid SFPs. □

To explain how the HSAG is created, we assume there is an h-SFC as $\{s, v_1, v_2, v_3, d, v_1, s'\}$, where s and s' represent the first and last node in the forward and backward traffic, respectively. The composited HSAG is shown in Fig. 5.7, which includes 7 tiers. In Fig. 5.7, x, y and z denote the cardinality of the substrate candidates for v_1, v_2 and v_3, respectively. To construct the HSAG in the forward traffic, one needs to start with the source node (T_1). Since T_1 is created for a non-common SF, only one layer $L_1^{T_2}$ is required in T_2, which includes $SC_{v_1}^{2,1}$. The source node in T_1 connects with all nodes in $L_1^{T_2}$. Since T_2 is created for a common SF, for each node $sc_k^{2,1}$ in T_2, a layer $L_k^{T_3}$ that includes $SC_{v_2}^{3,k}$ is created and an index number v_1k is added to the layer. Next, each $sc_k^{2,1}$ connects with all nodes in the corresponding layer $L_k^{T_3}$. As T_4 and T_5 are created for non-common SFs, they follow the same rule as T_1. Since only v_1 is the common SF, there are x (cardinality of the substrate candidates for v_1) layers in the T_5 (tier for the destination). Then, the backward HSAG is constructed from destination layer $L_n^{T_5}$. As T_6 is created for a common SF v_1, for a specific $L_n^{T_5}$, the substrate node $sc_k^{6,n}$ is created according to the index number v_1k. For example, $sc_1^{6,1}$ is created for $L_1^{T_5}$. All nodes in $L_n^{T_5}$ connects with the newly create node $sc_k^{6,n}$. Since there are x layers in the destination tier, for each tier in the backward HSAG (other than the last tier), there are x layers. At last, all layers in T_6 connects with the node in T_7.

5.4.2 Optimal Hybrid SFC Embedding Algorithms

Based on the HSAG, we propose an Optimal Hybrid SFC Embedding (Opt-HSFCE) algorithm in Algorithm 8. Opt-HSFCE prunes the links in the SN that cannot provide enough bandwidth resource. Then, the algorithm finds the available substrate candidates for each

requested SF and constructs the HSAG. At last, the optimal hybrid SFP is created as the shortest path connecting the source node s and s' in the composited HSAG.

Algorithm 8 Optimal Hybrid SFC Embedding (Opt-HSFCE)

1: **Input:** G, HSR;
2: **Output:** *Hybrid SFP*;
3: Discarding the links with bandwidth less than BW in G;
4: **for** each SF v_i in the SFC **do**
5: Find all available substrate candidates and form a list tier i for the SF v_i;
6: **end for**
7: Composite the HSAG according to G, HSR and the substrate candidate list of each SF;
8: Formulate an *SFP* by finding the shortest path connecting s to s' in the composited HSAG;
9: **Return:** *Hybrid SFP*;

Theorem 5.4.4. *Opt-HSFCE can construct the hybrid SFP with the minimum latency when SN provides enough computing and bandwidth resource.*

Proof. The hybrid SFP created by Opt-HSFCE is the shortest path that connects the source node s and s' in the composited HSAG. With Lemmas 5.4.1-5.4.3, the composited HSAG includes all possible hybrid SFPs while guaranteeing the "re-visit constraint" and "executing-order constraint". Therefore, the shortest path that starts at the source node s and ends at s' node is the hybrid SFP that has the minimum latency for the given HSR. □

We then analyze the time complexity of the proposed Opt-HSFCE algorithm. We assume that the length of the h-SFC is δ, and there are ϕ common SFs along with the two directions of the h-SFC. Thus, there are δ tiers in the HSAG. In the worst case, every substrate node can provide all requested SF instances ($\mathbb{F}_n = F, \forall n \in N$). For each tier T_i, there are at most $|N|^{\phi}$ layers, each of which can be a set of substrate candidates of v_i. For each tier,

there are at most $|N|^{\phi+1}$ nodes. Since each pair of layers in T_i and T_{i+1} have at most $|N|^2$ links, there are at most $|N|^{\phi+2}$ links between two adjacent tiers. Accordingly, in an HSAG, there are at most $\frac{\delta}{2}|N|^{\phi+1}$ nodes and $\frac{\delta}{2}|N|^{\phi+2}$ links. To generate an HSAG, the time complexity is $\mathcal{O}(|N||E| + |N|^{\phi+2})$. When applying the Dijkstra's shortest path algorithm, the time complexity is $\mathcal{O}(\frac{\delta}{2}|N|^{\phi+2}log(\frac{\delta}{2}|N|^{\phi+1}))$. As one can see that, the time complexity of the Opt-HSFCE algorithm depends on the number of common SFs in the h-SFC (ϕ). As ϕ is an integer and generally much smaller than $|N|$, the Opt-HSFCE can be regarded as a pseudo-polynomial time algorithm.

5.5 Experimental Results and Analysis

In this section, we show the performance of the proposed Opt-HSFCE algorithm when comparing with the techniques that are directly extended from [46] and [64].

5.5.1 Simulation Environment

We use the 24-nodes-43-links US-NET and a 40-nodes-180-links random network as the Substrate Networks (SNs). For the 40-nodes-180-links random network, each substrate node has at least 4 direct neighbours (i.e., each substrate node directly connects at least four other nodes). For each substrate node, its computing capacity is randomly assigned in the range of $[100 - 200]$. The number of SF instances that a substrate node can handle is randomly distributed in the range of $[2 - 6]$. For each substrate link, its latency cost is in the range of $[1 - 8]$ and its available bandwidth is in the range of $[30 - 50]$.

We set the number of SF nodes required by an HSR in the range of $[6 - 18]$, whereas the

number of common SFs is in the range of $[2-6]$. For each SF node, the required computing capacity is in the range of $[10-20]$. For the h-SFC, the source and destination nodes are randomly generated, and the bandwidth demand for each pair of SFs is in the range of $[1-5]$.

5.5.2 Performance Metrics and Benchmarks

We use the following metrics to assess the performance of the proposed schemes.

Average Latency: We use Average Latency to evaluate the performance of the proposed scheme. Average Latency is calculated as $AvL = \frac{\sum_{HSR_i \in HSR_C} |SFP_{created}|}{|HSR_C|}$, where $|SFP_{created}|$ represents the length of the hybrid SFP generated from the request HSR_i; and HSR_C is the set of HSRs.

Accumulated Latency: When multiple requests are embedded onto the same SN, we use Accumulated Latency to evaluate the performance. Accumulated Latency is calculated as $AcL = \sum_{HSR_i \in HSR_C} |SFP_{created}|$.

Runtime: We employ the runtime to evaluate the time complexity of the proposed Opt-HSFCE algorithm. We build our algorithms in JAVA to validate the performance of our proposed schemes. All measurements are conducted on a Windows 10 system equipped with the Intel i7-4870Q CPU @ 2.50 GHz core and 16 GB RAM.

We implement the brutal force algorithm, which searches the shortest hybrid SFP among all possible combinations for a given h-SFC. In addition, we extend the optimal SFCE technique [46] and the heuristic technique proposed in [64] as the benchmarks. We extend the technique proposed in [46] by i) constructing the optimal forward SFP as [46]; ii) recording the SF node embedding results as the forward SFP; iii) applying the technique in [46] for

each pair of substrate nodes that host the common SFs by the backward direction; and iv) constructing the hybrid SFP by concatenating them. Similarly, we extend the technique proposed in [64] by i) generating the multi-stage graph for both forward and backward SFCs, ii) embedding the common SF(s) onto the substrate node that has the least sum of distances to the substrate candidates of the previous and the next SFs, iii) concatenating the forward and backward multi-stage graph as the hybrid multi-stage, and iv) constructing the hybrid SFP according to the hybrid multi-stage graph. We name the methods extended from [46] and [64] as "2OPT-SFCE" and "Multi-Stage SFCE", respectively.

5.5.3 Performance Analysis of Opt-HSFCE for Single HSR

Figs. 5.8 and 5.9 illustrate the impact of the h-SFC length when setting the number of common SFs as 4. Figs. 5.10 and 5.11 present the impact of the common SFs in an h-SFC when setting the length of the h-SFC as 16 and 24, respectively. Since the brutal force algorithm has the same performance as the proposed Opt-HSFCE, Figs. 5.8-5.11 only show 3 curves, whereas the red dashed curves represent the performance of "Opt-HSFCE", the blue solid curves are from "2OPT-SFCE", and the grey dotted curves show the performance of "Multi-Stage SFCE".

When increasing the number of SFs in the h-SFC, Opt-HSFCE outperforms the other two techniques. For 2OPT-SFCE, this is because the common SFs are embedded according to their positions in forward traffic, which may introduce additional latency for the backward traffic. Even though Multi-Stage SFCE embeds the common SFs onto the substrate nodes that have the least latencies sum to their previous and next SF candidates, it limits the

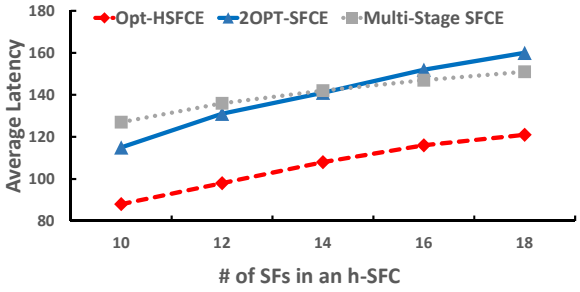


Figure 5.8: The 24-nodes US-NET

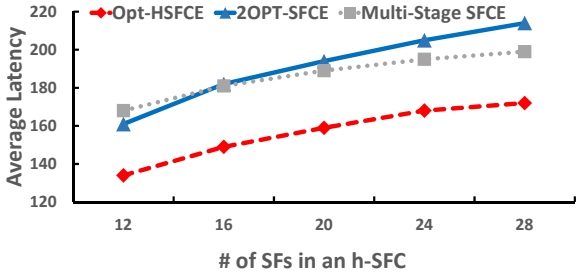


Figure 5.9: The 40-nodes random network

embedding of the non-common SFs. In US-NET, Opt-HSFCE outperforms 2OPT-SFCE and Multi-Stage SFCE by an average of 35% and 37%, respectively. Opt-HSFCE outperforms 2OPT-SFCE and Multi-Stage SFCE by an average of 23% and 19% in the 40-nodes random network.

When increasing the number of common SFs in the h-SFC, all algorithms need less latency, and Opt-HSFCE always has the best performance. This is because increasing the number of common SFs likely reduces the number of substrate nodes that are required to host the requested SFs. With a fixed length of h-SFC, if the number of common SFs is half of the length, the h-SFC becomes a bidirectional SFC, whereas the forward and backward SFCs request the same set of SFs. Hence, when the number of common SFs is 6 in Fig. 5.10, the performance of 2OPT-SFCE is similar to Opt-HSFCE as indicated by Lemma 5.3.1.

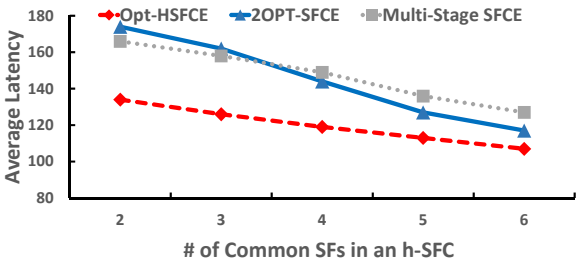


Figure 5.10: The 24-nodes US-NET

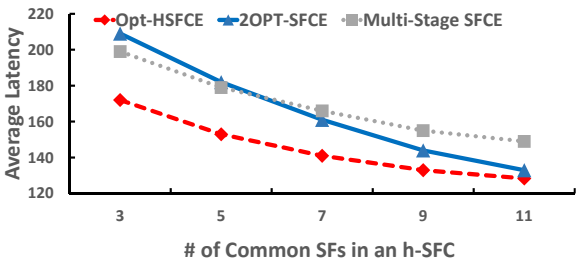


Figure 5.11: The 40-nodes random network

Note that, when the number of common SFs is less than 3 in Fig. 5.10, Multi-Stage SFCE outperforms 2OPT-SFCE, while the latter scheme is better when the number of common SFs is greater than 3. This is because, when the number of common SFs is large, the Multi-Stage SFCE method will embed each SF in a nearest-neighbour manner. In the 24-nodes US-NET, Opt-SFCE outperforms 2OPT-SFCE and Multi-Stage SFCE by an average of 20% and 23%, respectively. In the 40-nodes random network, Opt-SFCE outperforms 2OPT-SFCE and Multi-Stage SFCE by an average of 13% and 17%, respectively.

5.5.4 Performance Analysis of Opt-HSFCE for Multiple HSRs

When embedding multiple HSRs onto the 40-nodes random network, Figs. 5.12 and 5.13 show the accumulated latency required by the proposed scheme. We set the length of the h-SFC as 24 for both figures. The number of common SFs in Fig. 5.12 is 4, while it is 9 in Fig. 5.13. In both figures, the red grid bar represents the performance of "Opt-HSFCE", the blue bar shows the performance of "2OPT-SFCE", and the grey bar represents the performance of "Multi-Stage SFCE". Again, we can see that Opt-HSFCE always outperforms the other two techniques. When HSR requires less common SFs (i.e., Fig. 5.12), Multi-Stage SFCE

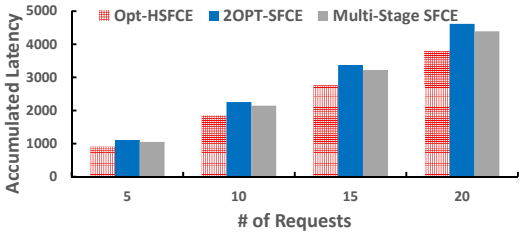


Figure 5.12: Multiple requests in the 40-nodes random network when HSR requests 4 common SFs

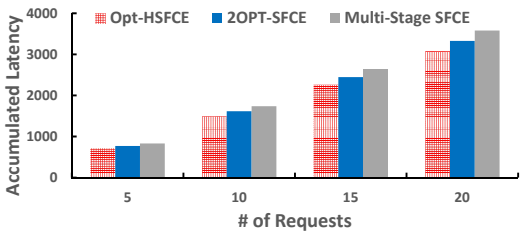


Figure 5.13: Multiple requests in the 40-nodes random network when HSR requests 9 common SFs

outperforms 2OPT-SFCE; while 2OPT-SFCE outperforms Multi-Stage when HSR requires a large number of common SFs (i.e., Fig. 5.13).

5.5.5 Runtime Analysis of Opt-HSFCE

We evaluate the runtime of Opt-HSFCE in the 24-nodes-43-links US-NET under two different scenarios (i.e., Fig. 5.14 and 5.15). In Fig. 5.14, 4 common SFs are in all h-SFCs, while the length of the h-SFC varies. In Fig. 5.15, 16 SFs are requested in the h-SFC, while the number of common SFs varies. In both Fig. 5.14 and 5.15, the green dashed curves, red dashed curves, blue solid curves and grey dotted curves represent the runtime of the "brutal force", "Opt-HSFCE", "2OPT-SFCE" and "Multi-Stage SFCE", respectively. Even though 2OPT-SFCE and Multi-Stage SFCE need less runtime than brutal force and Opt-HSFCE, the latter two significantly outperforms the former two in terms of latency as mentioned above.

When increasing the number of SFs of the h-SFC in Fig. 5.14, the runtime required by the brutal force algorithm increases sharply from hundreds of seconds to thousands of seconds; while the runtime of Opt-HSFCE does not vary much. This is because, when the

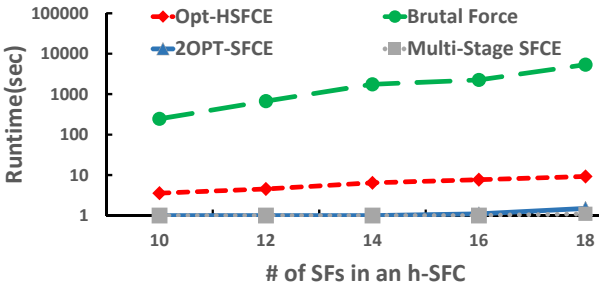


Figure 5.14: Runtime vs length of h-SFC

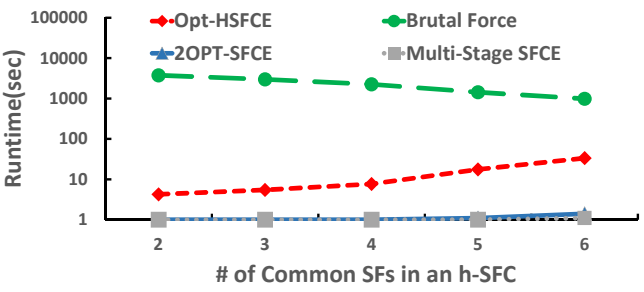


Figure 5.15: Runtime vs # of common SFs

number of common SFs is fixed, increasing the number of SFs in the h-SFC does not affect the complexity of the generated HSAG much. However, for the brutal force, increasing the number of SFs in the h-SFC brings much more potential hybrid SFP combinations, which greatly affects the runtime performance. For example, in the worst case (i.e., every substrate node provides all SF instances), increasing the number of SFs from 10 SF nodes to 12 SF nodes brings $|N|^{12}-|N|^{10}$ more combinations for the brutal force algorithm. Averagely, given an h-SFC with four common SFs, the brutal force needs 2053.45 seconds, while Opt-HSFEC needs 6.282 seconds.

When increasing the number of common SFs in the h-SFC in Fig. 5.15, the runtime for the brutal force algorithm decreases while the runtime of Opt-HSFCE increases. This is because increasing the number of common SFs can effectively decrease the searching combinations for the brutal force algorithm as the backward traffic depends on the forward one. However, the number of common SFs greatly affects the complexity of the HSAG (i.e., the number of nodes and links in the HSAG). Thus, increasing the number of common SFs introduces more runtime for Opt-HSFCE. Nevertheless, with an increasing number of common SFs, Opt-HSFCE still significantly outperforms the brutal force algorithm. Averagely, when the length of the h-SFC is 16, the brutal force algorithm requires 2282.6 seconds, while the Opt-HSFCE needs 13.6 seconds.

5.6 Summary

In this chapter, we have investigated how to optimally embed a given hybrid SFC (h-SFC) in MEC systems. We have mathematically formulated the Minimum Latency Hybrid Service Function Chain Embedding (ML-HSFCE) problem and have proposed an algorithm called Optimal Hybrid SFC Embedding (Opt-HSFCE), which optimally embeds a given h-SFC onto the substrate network. Through extensive simulations and analysis, we have shown that the proposed Opt-HSFCE can find the optimal hybrid SFP with much less runtime compared with the brutal force algorithm and significantly outperforms the schemes that are directly extended from the existing SFC Embedding (SFCE) techniques.

CHAPTER 6
DEPENDENCE-AWARE SERVICE CHAINING AND EMBEDDING

6.1 Motivation

Traditionally,the SFCE problem generally includes two important processes: 1) SFC design and 2) SFC mapping. The former process constructs an SFC for a given Network Service Request (NSR), while the latter maps the constructed SFC onto the substrate network via node/link mapping subprocesses, which correspond to the node/link mapping in traditional Virtual Network Embedding (VNE) [65, 66, 67]. In node mapping, each VNF node is embedded onto a substrate node that has enough available computing resource and provides the requested functionality. To connect the mapped VNF nodes, the link mapping process finds a set of subtrate/physical paths (links) that satisfies the requested bandwidth. This set of substrate/physical paths (links) is also known as Service Function Path (SFP)[14].

As the SFCE is known as NP-hard [18], many researchers have proposed efficient heuristic approaches (e.g., [24, 68, 69]). Recently, how to effectively conduct SFC design and VNF node/link mapping while considering the dependences constraint among VNFs draws research attention [70]. When designing the SFC, if VNF q depends on another VNF p, then q must be placed after p in the constructed chain such that the data stream goes through p before arriving at q. The corresponding optimization problem is called as Dependence-aware Service Function Chain Embedding (D_SFCE). In [70], the authors propose the technique of *Independent Grouping* to guarantee the correctness of SFC construction and the technique of *Adaptive Mapping* to map the created SFC onto IP network via the shortest path

connections.

The rest of the chapter is organized as follows. In Section II, we formulate the Dependence-aware Service Function Chain Embedding in Optical networks (D_SFCE_O) problem. Section III presents our D_SFC_LU algorithm. In Section IV, we analyze the performance of the proposed algorithm. We summarize this chapter in Section V.

6.2 Problem Statement

In this section, we formalize the problem of the Dependence-aware Service Function Chain Embedding in Optical networks (D_SFCE_O).

6.2.1 Substrate Optical Network

We use an undirected graph $SON = (N_S, L_S)$ to represent the Substrate Optical Network (SON), where N_S and L_S denote the set of substrate nodes and substrate fiber links, respectively. Each substrate node is able to host some VNF nodes. The available computing resource (i.e., CPU) of a substrate node s_i is denoted as $CPU_{s_i} \in \mathbb{Z}^+$. F_S is a set of commonly used network functionalities. We use $f_{s_i} \in F_S$ to denote the functionality provided by substrate node s_i. Without loss of generality, each substrate node only provides one type of functionality. For each substrate fiber link $l_{s_i s_j} \in L_S, (s_i, s_j \in N_S)$, we use $subcarrier_i$ to represent the i_{th} subcarrier. For example in Fig. 6.1, the boxes beside the fiber link $l_{s_1 s_2}$ represent subcarriers where the white boxes (i.e., $subcarrier_0$ and $subcarrier_2$) are the available subcarriers and the yellow boxes (i.e., $subcarrier_1$ and $subcarrier_3$) indicate that the subcarriers are occupied.

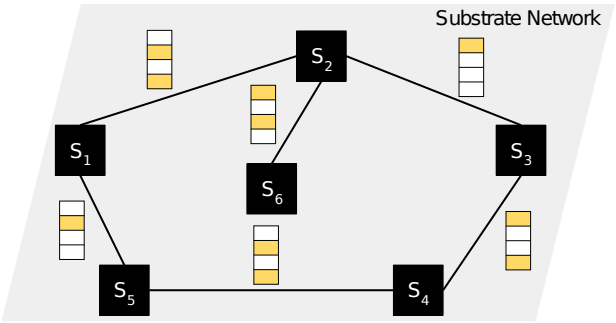


Figure 6.1: An Example of a Substrate Optical Network.

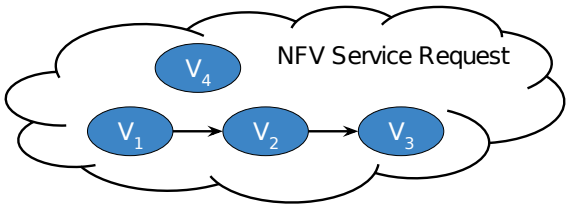


Figure 6.2: Function dependences in NSR_1.

6.2.2 Dependence-aware NFV Service Request

An NFV Service Request (NSR) can be represented by a 3-tuple $NSR =< V, D, B >$, where V is the set of VNF nodes; D represents dependences among VNF nodes, and $B \in \mathbb{Z}^+$ is the requested bandwidth for the data stream. Each VNF node $v \in V$ demands a certain amount of computing resource $CPU_v \in \mathbb{Z}^+$, and a specific network function f_v. To represent dependences among VNF nodes, we use $v_i \rightarrow v_j$ to denote that v_j has a dependence on v_i. Without loss of generality, we assume that each VNF node only requires a unique network function. In other words, there are no two VNF nodes requesting the same network functionality in the same NSR.

Fig. 6.2 illustrates an example of the dependences constraint in NSR_1. There are four VNF nodes in the NSR_1, and the arrows represent dependences among the VNF nodes. Specifically, v_2 depends on v_1 ($v_1 \rightarrow v_2$) and v_3 depends on v_2 ($v_2 \rightarrow v_3$) while v_4 has no dependence on any other VNF node.

6.2.3 Dependence-aware SFC Embedding in Optical Networks

To map an NSR onto a shared SON, we need take the dependences constraint, VNF node mapping constraint and VNF link mapping constraint into consideration.

Dependences Constraint: If the function of v_j depends on that of v_i (i.e., $v_i \rightarrow v_j$), then, in the designed chain, v_i has to be placed ahead of v_j.

VNF Node Mapping Constraint: Each $v \in V$ must be mapped to exactly one substrate node that can provide enough computing resource and the corresponding functionality. To enhance the reliability of the system, we assume that no two VNF nodes in the same NSR are allowed to be mapped onto the same substrate node.

VNF Link Mapping Constraint: Each VNF link between two consecutive VNF nodes in the designed SFC must be mapped (e.g., via the RSA process) onto a substrate fiber link or a physical fiber path that can provide enough consecutive subcarriers. We assume every substrate node is equipped with wavelength converters such that only consecutiveness of subcarriers are taken into account.

Definition of D_SFCE_O problem: The optimization problem of Dependence-aware Service Function Chain Embedding in Optical networks (D_SFCE_O) problem is defined as how to design an SFC and map the created SFC onto a given SON while i) satisfying the aforementioned constraints, ii) minimizing the required bandwidth for the created SFP. A secondary object function for D_SFCE_O is to minimize the number of wavelength converters employed by the NSR. When there is no dependece and spectrum allocation constraints, D_SFCE_O problem can be reduced to SFCE problem. As a special case of D_SFCE_O,

SFCE is known as NP-hard [18]. Therefore, we propose an efficient heuristic algorithm to solve D_SFCE_O problem in the following section.

6.3 Dependence-aware Service Function Chain embedding with Least-Used consecutive subcarriers

In this section, we propose an efficient algorithm, namely, Dependence-aware Service Function Chain embedding with Least-Used consecutive subcarriers (D_SFC_LU). Different from the traditional SFCE algorithms which sequentially create the chain and map the created chain [71], D_SFC_LU algorithm jointly optimizes the chain design and mapping processes while taking into account the process of routing and spectrum allocation in optical networks. The proposed D_SFC_LU algorithm selects and appends a VNF node that has no dependency on the remaining un-mapped VNF nodes, to the tail of the created SFC by applying the technique of Impact Factor based Node Selection (IFNS). When mapping the selected VNF node onto a substrate node, D_SFC_LU takes the advantage of Chain Node Mapping (CNM) technique to identify proper substrate node. At the same time, the Chain-Fit link mapping technique is proposed to connect these substrate nodes with a Service Function Path (SFP).

6.3.1 Impact Factor based Node Selection

For a given NFV service request $NSR =< V, R, B >$, to illustrate the dependences among VNF nodes, we define two operators: *child* and *parent* as shown in Eqs. (6.1)-(6.2). If a VNF node v_j depends on another VNF node v_i, we call v_j as a child of v_i as shown in Eq.

(6.1). Similarly, we call v_j as a parent of v_i in Eq. (6.2) if a VNF node v_i depends on VNF node v_j.

$$child_{v_i} = \{v_j | (v_i \rightarrow v_j) \in D_V, v_i, v_j \in V\} \tag{6.1}$$

$$parent_{v_i} = \{v_j | (v_j \rightarrow v_i) \in D_V, v_i, v_j \in V\} \tag{6.2}$$

We further define two operators: *descendant* and *precedent* in Eqs. (6.3)-(6.4). Among the VNF nodes that have not been added into the created SFC, if a VNF node v_j is a child or grandchild of v_i, then v_j is a descendant of v_i as shown in Eq. (6.3). Similarly, v_j is a precedent of v_i, if v_j is a parent or grandparent of v_i in Eq. (6.4).

$$descendant_{v_i} = \{ \bigcup_{v_j \in child_{v_i}} \{v_j\} \cup descendant_{v_j} \} \tag{6.3}$$

$$precedent_{v_i} = \{ \bigcup_{v_j \in parent_{v_i}} \{v_j\} \cup precedent_{v_j} \} \tag{6.4}$$

Lemma 6.3.1. *If $precedent_{v_i}$ is empty, then appending v_i to the tail of the created SFC does not violate the dependences constraint.*

Lemma 6.3.2. *If both $precedent_{v_i}$ and $precedent_{v_j}$ are empty, then appending v_i ahead of or behind v_j in the SFC does not violate the dependencies constraint.*

According to Lemma 6.3.1-6.3.2, to facilitate the processes of VNFs chain design and mapping, we create a Precedent-mapped List (PL) consisting of VNF node v that has no

dependencies on the remaining un-mapped VNF nodes, i.e., $sizeof(precedent_v) = 0$.

NSR impact factor: In a created SFC, we denote $v_i \dashrightarrow v_j$ as the data stream travelling from v_i to v_j. For example, in Fig. 6.2, initially $precedent_{v_1}$ and $precedent_{v_4}$ are empty. If we select v_4 as the Next VNF node (NV) to append to the created SFC, there would be only one option to design the chain: $v_4 \dashrightarrow v_1 \dashrightarrow v_2 \dashrightarrow v_3$. However, if selecting v_1 as NV, one will have multiple options to construct the chain. From this observation, we define NSR Impact Factor (NIF_v) of a VNF node v as shown in Eq. (6.5), where $distance(v, u)$ stands for the hop numbers between VNF nodes v and u in the dependence graph of an NSR.

$$NIF_v = 1 + \sum_{u \in\ descendant_v} \frac{1}{1 + distance(v, u)} \tag{6.5}$$

Intuitively, mapping a VNF node v that has a larger NIF_v value first can potentially add more un-mapped VNF nodes into PL. As a result, more chain candidates will present, which is broadly equivalent to enlarging the searching space of SFC design. For example, when calculating the NIF of v_1 and v_4 in NSR_1, v_1 has descendants as $\{v_2, v_3\}$, $NIF_{v_1} = 1 + \frac{1}{2} + \frac{1}{3} = \frac{11}{6}$ while v_4 has no descendant, $NIF_{v_4} = 1$. Hence, we propose the Impact Factor based Node Selection (IFNS) technique in Algorithm 9. Line 1 calculates the NIF values of un-mapped VNF node v with $sizeof(precedent_v) = 0$ and selects the one with the highest NIF value as NV. Line 2 appends it to the tail of the created SFC. Line 3-5 updates the PL.

Algorithm 9 Impact Factor based Node Selection (IFNS)

Require: SON, NSR, PL, NV;
Ensure: NV, PL;
1: Calculate NIF values of VNF nodes with $sizeof(Precedent_v) = 0$ in NSR and set NV as the VNF node with highest NIF;
2: Prune NV from NSR and append it at the tail of the SFC;
3: **for** VNF node v that has no precedent in NSR **do**
4: PL = PL $\cup$ $\{v\}$;
5: **end for**

6.3.2 Chain Node Mapping

For a substrate node s, if s can provide enough computing resource and corresponding functionality for a VNF node v, we call s as a substrate candidate (sc^v) of v. As v may have multiple substrate candidates, the candidate set of v is denoted as SC^v. To map the NV onto an appropriate substrate node, we propose an efficient node mapping technique, namely, Chain Node Mapping (CNM), which employs Closeness-Centrality Ratio (CCR) and Linear Property Node Mapping (LPNM) to identify a proper substrate node for the NV.

Closeness-Centrality Ratio: In order to decrease the bandwidth consumption of the ensuing node/link mapping processes, the location to map the first NV is important. We introduce Closeness-Centrality Ratio (CCR) [72] to take advantage of the location relationships between the substrate candidate (sc^{NV}) of the first NV and the substrate candidate of the VNF nodes in PL. The calculation of CCR is shown in Eq. (6.6), where $distance(sc^{NV}, sc^v)$ represents the hop numbers between sc^{NV} and sc^v in the SON.

$$CCR(sc^{NV}, \text{PL}) = \frac{1}{\sum_{sc^v \in SC(v), v \in \text{PL}} distance(sc^{NV}, sc^v)} \tag{6.6}$$

Clearly, the higher the CCR value is, the more substrate candidate nodes are nearby.

Algorithm 10 Chain Node Mapping (CNM)

Require: SON, NSR, SFP, PL, NV;
Ensure: SFC, SFP;
1: **if** SFC only has one VNF node **then**
2: Calculate the Closeness-Centrality Ratio of all substrate candidates of NV by Eq. (6.6);
3: Map NV onto the substrate candidate with the highest Closeness-Centrality Ratio and update the SFP;
4: **else**
5: Set d_2 as the length of the path between the substrate candidate (sc^{NV}) of NV to the tail (s_{tail}) of the current SFP;
6: Set d_1 as the length of the path between the substrate candidate of NV to the substrate candidate (sc^{v}) of the VNF nodes in PL;
7: Find the substrate candidate of NV where $d_1 + d_2$ is minimized and map NV to that substrate node;
8: Update the SFP;
9: **end if**

Hence, mapping NV onto the substrate candidate with the highest CCR value can potentially reduce the overall length of SFP as well as bandwidth consumption in the SON. For example, when mapping NSR_1 onto the SON in Fig. 6.1, v_1 is selected as the NV and PL is updated as $\{v_2, v_4\}$. When calculating the CCR values of v_1's substrate candidates (i.e., s_1 and s_5), $CCR_{s_1} = \frac{1}{1+2+2} = \frac{1}{5}$ while $CCR_{s_5} = \frac{1}{1+2+3} = \frac{1}{6}$. Accordingly, mapping v_1 onto s_1 which has higher CCR value, is a better option, as shown in Fig. 6.3.

Linear Property Node Mapping: The structure of an SFC is different from a Virtual Network (VN) where one virtual node might need multiple (e.g., more than 2) connections with its neighbors. In the SFC, any VNF node (excluding the first and last VNF nodes in the chain) connects two VNF neighbor nodes, one is the tail of the created SFC and the other one is the NV. For example, in Fig. 6.3, after appending v_1 at the tail of the SFC and mapping v_1 onto the SON, v_2 is selected as the NV and the PL is updated as $\{v_3, v_4\}$. When

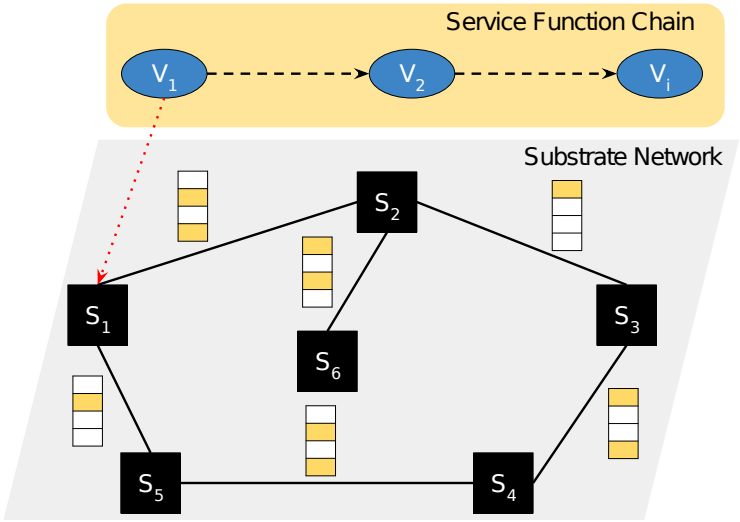


Figure 6.3: An Example of CCR and LPNM.

appending v_2 to the tail of the created SFC, v_2 has two VNF neighbor nodes. One is v_1, the tail of the current SFC, and the other is v_3 or v_4, based on which is selected as the NV in next iteration.

Hence, we propose the Linear Property Node Mapping (LPNM) to find an appropriate substrate candidate for the NV by utilizing this SFC connection property, which is shown in Line 5-8 of Algorithm 10. In LPNM, we define d_2 as the substrate hop number of the path between the tail (s_{tail}) of the current SFP and sc^{NV}. Similarly, we define d_1 as the substrate hop number of the path between sc^{NV} and sc^{v} ($v \in PL$). Then, we search the minimal $d_1 + d_2$ value for all sc^{NV}. The NV is mapped onto the substrate node with the minimal $d_1 + d_2$ among all sc^{NV}. For example, in Fig. 6.3, if v_1 has been mapped onto s_1 and s_1 becomes the tail of the SFP. VNF node v_2 is selected as the NV and the PL is updated as $\{v_3, v_4\}$. LPNM method considers the length of the path between v_2's candidates (s_2 and s_6) to the tail (s_1) of the created SFP and the substrate candidates (s_3 and s_4) of the VNF nodes

in PL. As one can see in Fig. 6.3, when mapping v_2 onto s_2, the path $(s_1 - s_2 - s_3)$ provides the minimal $d_1 + d_2$ as 2. However, when mapping v_2 onto s_6, the path $(s_1 - s_2 - s_6 - s_2 - s_3)$ provides the minimal $d_1 + d_2$ as 4. As a result, v_2 is mapped onto s_2.

6.3.3 Chain-Fit Link Mapping

After the NV is mapped by the LPNM method, a physical/fiber path, which consists one or multiple physical/fiber links, is constructed to support the data transmission between the NV and the tail of the created SFC. Each physical/fiber link along this fiber path needs reserve enough consecutive subcarriers requested by the NSR. When two links along this created SFP use different set of subcarriers, wavelength converter is needed, which increases the cost for the Internet Service Providers (ISPs) [73].

Algorithm 11 Dependence-aware Service Function Chain embedding with Least-Used consecutive subcarriers (D_SFC_LU)

Require: SON, NSR
Ensure: SFP
1: Initialize SFC, SFP, PL as $\emptyset$;
2: Find the least-used consecutive subcarriers which satisfies the bandwidth requirement;
3: **while** NSR is not $\emptyset$ **do**
4: Call IFNS() to calculate the importance of the un-mapped VNF nodes with $sizeof(precedent) = 0$ and construct PL;
5: Call CNM() to map un-mapped VNF nodes onto the substrate optical network;
6: **if** least-used consecutive subcarriers are available in the path between the substrate node where NV is mapped and the tail of the created SFP **then**
7: Reserves least-used consecutive subcarriers for the path;
8: **else**
9: Use First-Fit RSA to reserve enough consecutive subcarriers for the path;
10: **end if**
11: **end while**
12: **return** SFP

Unlike the traditional RSA approaches whereas the fiber path requests are generally

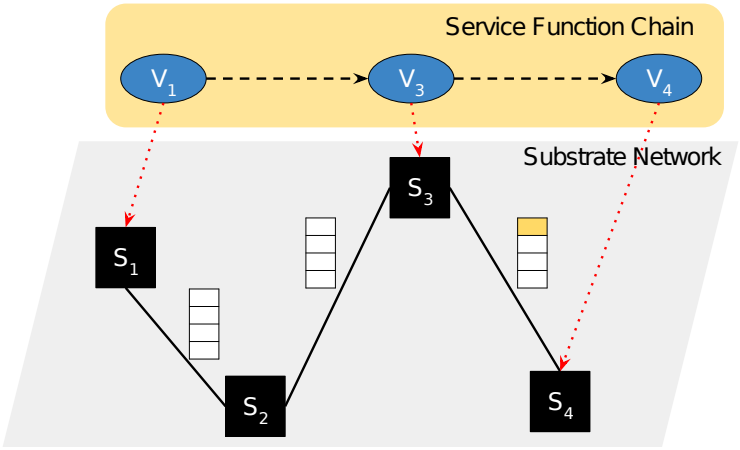


Figure 6.4: An Example of Chain-Fit RSA.

independent to each other, the fiber paths created to form the SFP are correlated. For example, in Fig. 6.4, v_1, v_3 and v_4 have the same requirement as they are in NSR_1 and s_1, s_2, s_3 and s_4 are the same substrate node in Fig. 6.1. The $subcarrier_0$ along $s_3 - s_4$ is not available. If the requested number of subcarriers is 2, to map the virtual link $v_1 \dashrightarrow v_3$, the traditional First-Fit RSA will employ $subcarrier_0$ and $subcarrier_1$, along fiber path $s1 - s2 - s3$. Similarly, to map $v_3 \dashrightarrow v_4$, First-Fit RSA will use $subcarrier_1$ and $subcarrier_2$ along fiber path $s3 - s4$. As a result, for data flow along this SFC, one would have to use wavelength conversion at substrate node s_3. To minimize the used subcarrier resource and wavelength conversion, we propose the Chain-Fit (CF) Routing and Spectrum Allocation, which finds the least-used consecutive subcarriers in the whole substrate nodes for the process of link mapping, as shown in Algorithm 11.

When applying the D_SFC_LU algorithm in Fig. 6.5, the D_SFC_LU algorithm first finds the least-used consecutive subcarriers for the NSR which is $subcarrier_2$ for the SON shown in Fig. 6.1. Continuing with aforementioned processes where v_2 has been mapped onto s_2

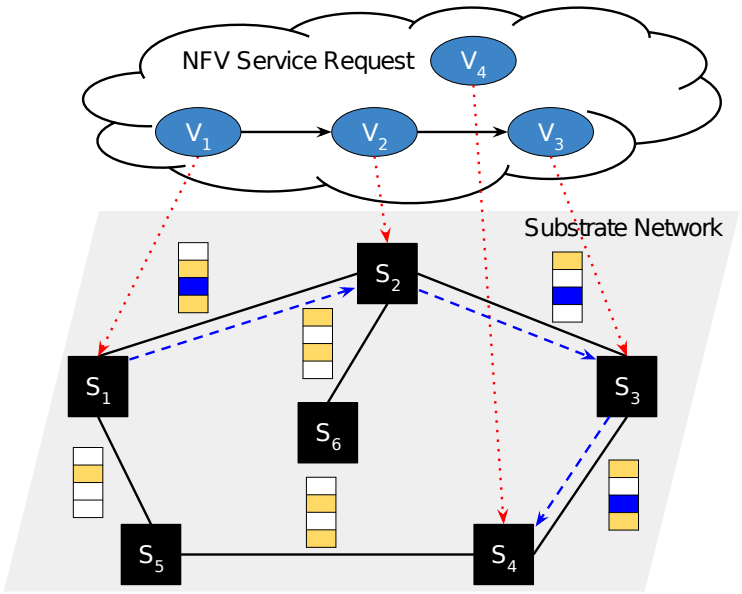


Figure 6.5: An Illustration of D_SFC_LU.

by the CNM technique, we need construct a path to connect s_1 with s_2 via the $subcarrier_3$ by applying the CF technique. The SFC is created as v_1 --➝ v_2 while the created SFP is $s_1 - s_2$. Then, there are only two un-mapped VNF nodes left, v_3 and v_4. As v_3 requires higher computing demand than v_4, v_3 is selected as the NV. As v_3 has only one substrate candidate as s_3, v_3 is mapped onto v_3. To construct a link connecting s_3 with the tail of the created SFP (s_2), the CF reserves $subcarrier_2$ in path $s_2 - s_3$. The created SFC now is v_1 --➝ v_2 --➝ v_3 and the corresponding SFP is $s_1 - s_2 - s_3$. In next iteration, v_4 is selected as the NV and mapped onto s_4. The CF reserves the $subcarrier_2$ for the connection between s_3 and s_4. As a result, the final SFC is created as v_1 --➝ v_2 --➝ v_3 --➝ v_4 and the constructed SFP is $s_1 - s_2 - s_3 - s_4$ with the bandwidth consumption as 3 subcarriers.

6.4 Experimental Results

In this section, we compare the performance of Dependence-aware SFC embedding with Least-Used consecutive subcarriers (D_SFC_LU) and Dependence-aware SFC with Adaptive Mapping (D_SFC_AM) Algorithms [70] by applying the proposed Chain-Fit (CF) and the well-known Random-Fit RSA [74]. In the D_SFC_AM, the algorithm constructs an SFC by applying the independent grouping technique and maps the constructed SFC onto the substrate network using the shortest-path strategy to connect two consecutive VNF nodes in a virtual request. We use "D_SFC_RF" to stand for an algorithm that replaces the Chain-Fit RSA process with RF RSA process in Algorithm D_SFC_LU. Similarly, we use "D_SFC_AM_CF" and "D_SFC_AM_RF" to represent the the coupling of Adaptive Mapping [4] with Chain-Fit and Random-Fit RSA, respectively.

We use a 24-node US Network [75] as the substrate optical network. Unless otherwise specified, the available computing resource of substrate node is in the range of $[5, 25]$; the offered functionality for each substrate node is randomly generated; and the available number of subcarriers for each substrate fiber link varies from 5 to 25. The number of VNF nodes in an NSR is set in the range of $[3, 7]$. Each VNF node requires the computing resource in the range of $[5, 25]$ while each NSR requests the number of subcarriers within the range of $[1, 5]$.

In Fig. 6.6, we set all substrate nodes and links with unlimited available computing capacity and subcarriers, respectively. As one can see in Fig. 6.6, the proposed D_SFC_LU and D_SFC_RF algorithms have better performance than D_SFC_AM_CF and D_SFC_AM_RF due to the techniques of Closeness-Centrality Ratio (CCR) and Linear Property Node Map-

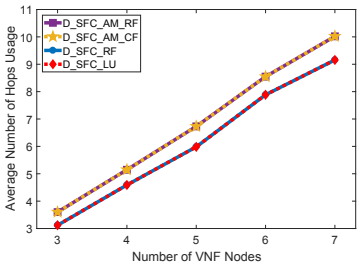


Figure 6.6: SON with unlimited computing resource and unlimited subcarriers

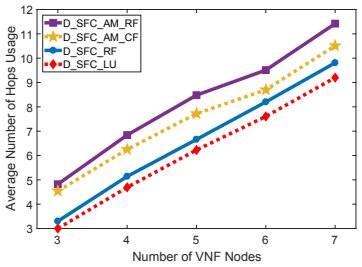


Figure 6.7: SON with unlimited computing resource and limited subcarriers

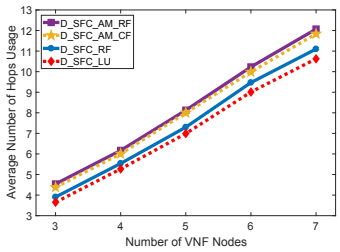


Figure 6.8: SON with limited computing resource and limited subcarriers

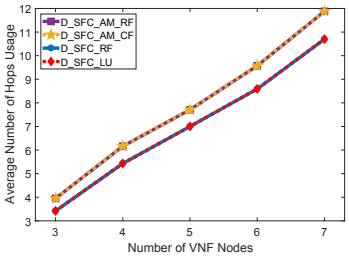


Figure 6.9: SON with limited computing resource and unlimited subcarriers

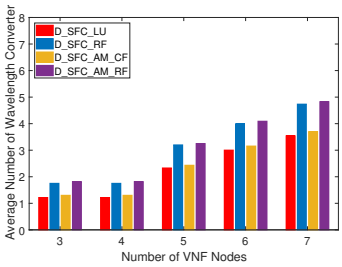


Figure 6.10: SON with unlimited computing resource and limited subcarriers

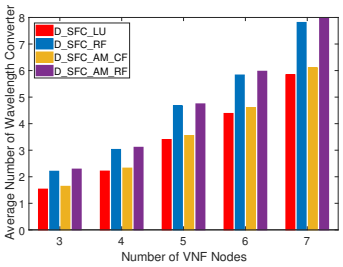


Figure 6.11: SON with limited computing resource and limited subcarriers

ping (LPNM). In other words, if there is no impact of RSA (with unlimited spectrum resource) on the node mapping in the substrate network, the D_SFC_LU and D_SFC_RF algorithms have the same performance by using same node mapping techniques. The techniques of CCR and LPNM embed an un-mapped VNF node onto the substrate node that is the closest node to the constructed SFP in the substrate network. Fig. 6.9 further validates that the D_SFC_LU and D_SFC_RF find better node mapping choice for each VNF node when each substrate node has limited available computing capacity. This is because the D_SFC_LU and D_SFC_RF select the closer available substrate nodes to construct the SFP.

In Figs. 6.7 and 6.10, each substrate fiber link has limited available subcarriers while each substrate node has unlimited available computing capacity. Fig. 6.7 represents the

average number of hops usage while increasing the requested number of VNF nodes in a virtual request. D_SFC_LU outperforms the D_SFC_RF, D_SFC_AM_CF and D_SFC_AM_RF algorithms. This is because Chain-Fit (CF) RSA provides better spectrum allocation than Random-Fit (RF) RSA. Meanwhile, as CF RSA tries to employ the least-used consecutive subcarriers of the whole network, the number of used wavelength converters is less as shown in Fig. 6.10. Similarly, when the substrate network provides limited available subcarriers and computing capacity, Figs. 6.8 and 6.11 represent the average number of hops usage and wavelength converters while increasing the requested number of VNF node in a virtual request. D_SFC_LU outperforms the D_SFC_RF, D_SFC_AM_CF and D_SFC_AM_RF algorithms due to the contribution of the proposed LPNM and CF RSA techniques.

6.5 Summary

In this chapter, we have investigated the problem of Dependence-aware Service Function Chain Embedding in Optical networks (D_SFCE_O). We have proposed an efficient algorithm, namely, Dependence-aware Service Function Chain embedding with Least-Used consecutive subcarriers (D_SFC_LU) algorithm, which takes the advantage of the proposed techniques such as Impact Factor based Node Choosing (IFNC), Chain Node Mapping (CNM) and Chain-Fit RSA Link Mapping to jointly optimize the SFC design, mapping and RSA processes while minimizing the bandwidth resource and the usage of wavelength converters. Extensive experiments and analysis have shown that the proposed D_SFC_LU algorithm outperforms the existing work.

CHAPTER 7
FUTURE WORK

Even though this work has investigated state-of-the-art problems in delivering NFV services, there still exists open issues for further investigation in this field. We also list two future work in the following sections.

7.1 Future Direction One: Parallelism-aware Service Function Chain Composition and Embedding

With the advancement in Internet of Things (IoT), Multi-access Edge Computing (MEC), online mining and learning techniques, the customer's demands for connectivity, ultra-low latency and reliable services are ubiquitous [30, 76]. To efficiently accommodate these services, we need to take advantages of high-capacity fiber optical networks and the emerging flexible network virtualization technologies. Optical transmission techniques (e.g., dense wavelength-division multiplexing) allow a single fiber link to carry many wavelength channels offering huge spectrum efficiency and bandwidth [77]. In fact, the re-configurable wavelength switching devices have already been widely deployed in long-haul and metro networks to offer ubiquitous and reliable connectivities [78, 79, 80]. Recently, Network Function Virtualization (NFV) was proposed to provide flexible services while reducing the Capital Expenditures (CAPEX) and the Operating Expenditures (OPEX) [13]. NFV implements the network functions that run on traditional proprietary hardware-based middleboxes as software-based modules called Virtual Network Functions (VNFs) or Service Functions (SFs) [14]. The SF can be flexibly installed on or removed from physical nodes (e.g., edge/cloud servers) in

physical network infrastructures [34].

An NFV Service Request (NSR) from the customer includes service source, destination, a set of SFs (e.g., firewall, parental control), and corresponding resource demands (e.g., CPU, bandwidth) [15]. To accommodate an NSR, the service provider concatenates the required SFs into a linear logic structure called Service Function Chain (SFC). Then, the service provider embeds the constructed SFC onto the shared Physical Network (PN) while reserving enough network resource [35]. The processes of accommodating an NSR by compositing and embedding an SFC onto a shared PN is referred to as Service Function Chaining and Embedding (SFCE) [15]. The physical path identified by the SFCE processes to host an in-service SFC in the PN is called Service Function Path (SFP). In the literature, many approaches are proposed to minimize the SFP length or propagation delay through different SF node placement or SFC routing strategies, while the SF processing delay is not considered [17, 26, 37, 38, 39, 81].

The emerging 5G and beyond 5G communication techniques empower the applications with ultra-low latency requirements [82, 83, 84, 85]. Some 5G applications promise to deliver services with about one-millisecond latency [86, 87]. Under such scenarios, the processing delay from consecutively running SFs in an SFC is significant and comparable to the link propagation delay. Thanks to the Network Function Parallelism (NFP) technique, the service provider can run multiple SFs in parallel at one physical node to mitigate the impact of the SF processing delay [86]. With NFP, the processing delay of the SFs that are executed in parallel is reduced from the summation of their processing delay to the highest processing

delay among these SFs. According to the work in [86], two SFs can be executed in parallel only if the operations of these SFs do not conflict with each other. Specifically, the operation of an SF conflicts to the other one's when both SFs change the contents of the customer's traffic stream. For example, both Firewall (FW) and Deep Packet Inspection (DPI) might drop packets from the customer's stream, and they cannot be parallelly executed. While the Parental Control (PC) only monitors the customer's stream without any modification, PC can be executed in parallel with either FW or DPI.

Given that the NFP technique can reduce the processing delay by running multiple SFs in parallel, how to efficiently apply the NFP technique into the SFCE processes to jointly optimizes the SF processing delay and the SFP propagation delay remains open.

7.2 Future Direction two: Survivability of Service Function Chaining and Embedding

The emerging Internet of Things (IoT), 5G, and beyond 5G techniques empower heterogeneous sets of applications with strict Quality of Service (QoS) and ultra-low latency requirements [88, 89, 90]. Limited by its battery life and computing capacity, the IoT device cannot deal with the computation-intensive tasks from the novel applications [91]. To mitigate the above limitations, the computation-intensive tasks are offloaded to the cloud systems that abundant computing resource but may be far away from the customer [92]. As a result, the cloud systems may not be able to satisfy the QoS requirements from the latency-sensitive applications [93]. Thanks to the Multi-access Edge Computing (MEC), the servers with enough computing resource are deployed at the edge of the network to satisfy

the computation-intensive tasks with an ultra-low latency [94].

When Network Function Virtualization (NFV) is applied into MEC, the hardware-based middleboxes, such as firewalls (FWs), Load Balancer (LB), and Intrusion-Detection System (IDS) are replaced by Service Functions (SFs) running on the commodity servers to deliver the customer's service requests [95, 96]. By allowing the instance of SFs to host on the virtual machines and deploy on multiple physical nodes in the PN, NFV can improve network flexibility, scalability, and customize the customer requests, as well as reducing the Capital Expenditures (CAPEX) and the Operating Expenditures (OPEX) [22, 97].

On the one hand, NFV facilitates delivering flexible and cost-efficient services in MEC systems. On the other hand, NFV and MEC's combination confronts challenges in providing reliable real-time services for applications, such as stock exchange and autonomous driving [98]. For example, some MEC applications specify that services should be delivered within one millisecond and above 99.9% reliability [99]. To prevent the network from the physical (e.g., hardware appliances) failures that affect the latency and reliability during the service delivery, the work in [45, 100, 101, 102, 103] proposed several network protection methods. Notably, in terms of the work in [103], a proactive mechanism aims to assign backup nodes for every potential node failure for all the existing nodes and save time to avoid disrupting the rerouting process. Despite the fact that physical failures impact the significant data losses, delays and resource wastage, virtual (e.g., SFs, virtual machines) failures also cannot be ignored when applying the NFV technique to MEC systems.

With NFV, A Network Service Request (NSR) from a customer consists of a set of

SFs and resource demands (e.g., bandwidth, CPU, and storage) [26]. To satisfy an NSR, the service provider creates a Service Function Chain (SFC) by chaining the required SFs into a virtual catenary structure with virtual links, and reserves corresponding networking resources to embed the created SFC onto the shared PN [27]. Even though much research attention has been paid to protecting physical node failure, only little effort has put on the virtual node failure, such as the SF failure of an SFC [104]. When SFs are placed on the servers, the SF failure can rise to potential risk and cannot be underestimated. This is because one of the critical challenges in protecting an SF failure is that the reason of an SF failure is hard to be identified (e.g., misconfigurations, virtual machines crashed, etc.) [105, 106]. Unlike physical failures that will change the topology and connectivity of the PN, virtual failures, such like the SF failure, will interfere with the output of the data flows, and break the execution of multiple SFCs, thereby results in the termination of the entire services in the PN [107, 108]. However, with the ultra-low latency requirements from the 5G and beyond 5G communication techniques, traditional strategies for physical failure cannot be directly applied to solve virtual failures anymore [109], not to mention when physical and virtual failures may occur in the same PN. How to keep the high reliability and follow the ultra-low latency requirement remains open for further investigation.

CHAPTER 8
CONCLUSION

8.1 Dissertation Overview

In this dissertation, for the first time, we comprehensively study and propose a series of problems in how to efficiently deliver the NFV service in future Internet. In the literature, there exists much work on how to deliver the NFV service within the SFCE process [26, 39, 81, 110, 111, 112, 113, 114, 115, 116]. However, within the state-of-the-arts scenarios (e.g., applications in MEC, cloud computing, IoT), the requirements from the customer will significantly impact the SFCE process, and the existing methodologies may not be directedly applied on such scenarios [17, 27, 45]. Furthermore, from the existing work, SFCE is proved to be NP-hard, which can be reduced from many typical NP-hard problem (e.g., travelling salesmen problem and set cover). Another critical problem remains in SFCE is how to deliver the NFV services with a certain provable boundary. In order to address the above problems and challenges, we have done the following work: i) a 2-approximation algorithm to deliver the traditional SFC service in UFNs, ii) a logarithm-approximation algorithm to composite and embed an SFC in MFNs, iii) a 2-approximation algorithm to composite and embed an h-SFC in UFNs and an efficient heuristic algorithm to composite and embed an h-SFC in MFNs, iv) an optimal algorithm to embed a given h-SFC and v) an effective heuristic algorithm to composite and embed a dependence-aware SFC.

8.2 Dissertation Preliminary Work

We studied the provable SFC composition and embedding problem in the second and third chapters, respectively. In the second chapter, we investigated a problem of how to deliver the NFV service in Unique Function Networks (UFNs) with a provable boundary [47]. Note that, UFN is referred to a network whose physical node only provides one unique SF. We proved the SFCE process is NP-hard in UFNs and proposed a 2-approximation algorithm called SFCE with Spanning Closed Walk (SFCE-SCW). In the third chapter we further investigated of how to deliver the NFV service in Multiple Functions Networks (MFNs) with a provable boundary. We proposed a logarithm-approximation algorithm called COst Factor-based SFCE Optimization with ShortCut (COFO-SC) [117].

In the fourth and fifth chapters, we studied the problems of hybrid service function chain composition and embedding. In the fourth chapter, we investigated a problem of service delivery of novel NFV applications in future Internet [17]. 5G and Multi-access Edge Computing (MEC) empower the development of the latency-sensitive or computation-intensive applications such as real-time Virtual Reality (VR), Augmented Reality (AR) games and on-line machine learning [7]. In these applications, the forward traffic from the user and the backward traffic from the MEC server/cloud may require different sets of SFs. The SFC that requires different sets of SFs in the forward and backward directions is referred to as hybrid SFC (h-SFC) [14]. To save the Operating Expense (OPEX) and latency, the SFs required by both directions are generally installed on the same substrate node [52] [53]. As a result, the embedding process of the forward SFC (f-SFC) will impact the embedding process of the

backward SFC (b-SFC), and the traditional SFCE techniques cannot be directly applied. In this part, we comprehensively studied the h-SFC composition and embedding processes. In UFNs, we proved h-SFCE is NP-hard and propose a 2-approximation algorithm based on the graph theory based techniques called Eulerian Circuit based Hybrid SFP optimization (EC-HSFP). Then, we extended the EC-HSFP algorithm to Multiple Functions Networks (i.e., each physical node can provide multiple SFs) via the betweenness centrality technique called Betweenness Centrality based Hybrid SFP optimization (BC-HSFP). In the fifth chapter, when the h-SFC is given, we proposed an optimal algorithm called Optimal Hybrid SFC Embedding (Opt-HSFCE) [45].

Last but not least, in the sixth chapter, we investigated a problem of how to efficiently deliver the NFV service when executing orders of SFs are partially specified [22]. The partially given executing orders among the required SFs are specified as dependence constraint. For example, the encryption function must be processed before the decryption function; otherwise, the contents of the flow will be damaged. The essential challenge in such a scenario is how to composite and embed an SFC while the dependence constraints are always obeyed for the designed SFC and the SFP. In chapter six, we investigate an efficient heuristic algorithm of delivering the NFV service while obeying the given dependence constraints.

REFERENCES

[1] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. U. Rasool, and W. Dou, "Complementing iot services through software defined networking and edge computing: A comprehensive survey," *IEEE Commun. Surv. Tutorials*, vol. 22, no. 3, pp. 1761–1804, 2020.

[2] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 125:1–125:36, 2020.

[3] Q. Pham, F. Fang, V. N. Ha, M. J. Piran, M. Le, L. B. Le, W. Hwang, and Z. Ding, "A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art," *IEEE Access*, vol. 8, pp. 116 974–117 017, 2020.

[4] J. Ding, M. Nemati, C. Ranaweera, and J. Choi, "Iot connectivity technologies and applications: A survey," *IEEE Access*, vol. 8, pp. 67 646–67 673, 2020.

[5] I. Bambrik, "A survey on cloud computing simulation and modeling," *SN Comput. Sci.*, vol. 1, no. 5, p. 249, 2020.

[6] N. E. H. Bouzerzour, S. Ghazouani, and Y. Slimani, "A survey on the service interoperability in cloud computing: Client-centric and provider-centric perspectives," *Softw. Pract. Exp.*, vol. 50, no. 7, pp. 1025–1060, 2020.

[7] A. J. Ferrer, J. M. Marquès, and J. Jorba, "Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 111:1–111:36, 2019.

[8] R. Roman, J. López, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Gener. Comput. Syst.*, vol. 78, pp. 680–698, 2018.

[9] I. Alam, K. Sharif, F. Li, Z. Latif, M. M. Karim, S. Biswas, B. Nour, and Y. Wang, "A survey of network virtualization techniques for internet of things using SDN and NFV," *ACM Comput. Surv.*, vol. 53, no. 2, pp. 35:1–35:40, 2020.

[10] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5g network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges," *Comput. Networks*, vol. 167, 2020.

[11] Z. Shang, "Performance evaluation of the control plane in openflow networks," Ph.D. dissertation, Free University of Berlin, Dahlem, Germany, 2020.

[12] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent nfv middleboxes," in *proc. IEEE INFOCOM*, 2017, pp. 1–9.

[13] "Etsi gs nfv-man 001: Network functions virtualisation (nfv); management and orchestration," *v. 1.1.1*, Dec. 2014.

[14] J. Halpern and C. Pignataro. Service function chaining (SFC) architecture. [Online]. Available: https://tools.ietf.org/html/rfc7665

[15] P. Quinn and T. Nadeau. Problem statement for service function chaining. [Online]. Available: https://tools.ietf.org/html/rfc7498

[16] M. Veeraraghavan, T. Sato, M. Buchanan, R. Rahimi, S. Okamoto, and N. Yamanaka, "Network function virtualization: A survey," *IEICE Trans. Commun.*, vol. 100-B,

no. 11, pp. 1978–1991, 2017.

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

[18] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-art and Research Challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 2016.

[19] M. Chiosi and D. t. Clarke, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges and Call for Action," in *SDN and OpenFlow World Congress*, 2012, pp. 22–24.

[20] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, 2016.

[21] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A Survey on Service Function Chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, 2016.

[22] D. Zheng, E. Guler, C. Peng, G. Luo, L. Tian, and X. Cao, "Dependence-aware service function chain embedding in optical networks," in *proc. IEEE ICC*, 2019, pp. 1–6.

[23] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On Dynamic Service Function Chain Deployment and Readjustment," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 543–553, 2017.

[24] J. Fan, C. Guan, Y. Zhao, and C. Qiao, "Availability-aware Mapping of Service Function Chains," in *proc. IEEE INFOCOM*, 2017, pp. 1–9.

[25] J.-J. Kuo, S.-H. Shen, H.-Y. Kang, D.-N. Yang, M.-J. Tsai, and W.-T. Chen, "Service chain embedding with maximum flow in software defined network and application to the next-generation cellular network architecture," in *proc. IEEE INFOCOM*, 2017, pp. 1–9.

[26] G. Sallam, G. R. Gupta, B. Li, and B. Ji, "Shortest path and maximum flow problems under service function chaining constraints," in *proc. IEEE INFOCOM*, 2018, pp. 2132–2140.

[27] D. Zheng, C. Peng, E. Guler, G. Luo, L. Tian, and X. Cao, "Hybrid Service Chain Deployment in Networks with Unique Function," in *proc. IEEE ICC*, 2019, pp. 1–6.

[28] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, 1991.

[29] D. J. Rosenkrantz, R. E. Stearns, and P. M. L. II, "An analysis of several heuristics for the traveling salesman problem," *SIAM J. Comput.*, vol. 6, no. 3, pp. 563–581, 1977.

[30] L. Chettri and R. Bera, "A comprehensive survey on internet of things (iot) toward 5g wireless systems," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 16–32, 2020.

[31] M. Stoyanova, Y. Nikoloudakis, S. Panagiotakis, E. Pallis, and E. K. Markakis, "A survey on the internet of things (iot) forensics: Challenges, approaches, and open issues," *IEEE Commun. Surv. Tuts.*, vol. 22, no. 2, pp. 1191–1221, 2020.

[32] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 2017.

[33] B. Yi, X. Wang, K. Li, S. K. Das, and M. Huang, "A comprehensive survey of network function virtualization," *Comput. Netw.*, vol. 133, pp. 212–262, 2018.

[34] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 2016.

[35] H. Hantouti, N. Benamar, T. Taleb, and A. Laghrissi, "Traffic steering for service function chaining," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 487–507, 2019.

[36] I. Jang, D. Suh, S. Pack, and G. Dán, "Joint optimization of service function placement and flow distribution for service function chaining," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2532–2541, 2017.

[37] H. Hawilo, M. Jammal, and A. Shami, "Network Function Virtualization-aware Orchestrator for Service Function Chaining Placement in the Cloud," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 643–655, 2019.

[38] Z. Zhou, Q. Wu, and X. Chen, "Online orchestration of cross-edge service function chaining for cost-efficient edge computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 8, pp. 1866–1880, 2019.

[39] R. Yu, G. Xue, and X. Zhang, "Qos-aware and reliable traffic steering for service function chaining in mobile networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2522–2531, 2017.

[40] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp.

543–553, 2017.

[41] T. Taleb, A. Ksentini, M. Chen, and R. Jantti, "Coping with emerging mobile social media applications through dynamic service function chaining," *IEEE Trans. Wireless Commun.*, vol. 15, no. 4, pp. 2859–2871, 2016.

[42] S. D'oro, L. Galluccio, S. Palazzo, and G. Schembra, "Exploiting congestion games to achieve distributed service chaining in nfv networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 2, pp. 407–420, 2017.

[43] Q. Ye, W. Zhuang, X. Li, and J. Rao, "End-to-end delay modeling for embedded vnf chains in 5g core networks," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 692–704, 2019.

[44] M. Rost and S. Schmid, "Service chain and virtual network embeddings: Approximations using randomized rounding," *CoRR*, vol. abs/1604.02180, 2016.

[45] D. Zheng, C. Peng, X. Liao, and X. Cao, "Toward optimal hybrid service function chain embedding in multiaccess edge computing," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6035–6045, 2020.

[46] Z. Cao, M. S. Kodialam, and T. V. Lakshman, "Traffic steering in software defined networks: Planning and online routing," in *proc. ACM SIGCOMM workshop*, 2014, pp. 65–70.

[47] D. Zheng, C. Peng, X. Liao, G. Luo, L. Tian, and X. Cao, "Service function chaining and embedding with spanning closed walk," in *proc. IEEE HPSR*, 2019, pp. 1–5.

[48] D. Dolson, M. Boucadair, S. Homma, and D. Lopez. Hierarchical service function chaining (hsfc). [Online]. Available: https://tools.ietf.org/html/rfc8459

[49] J. L. Gross, J. Yellen, L. W. Beineke, and R. J. Wilson, "Introduction to graphs," in *Handbook of Graph Theory*, 2003, pp. 1–55.

[50] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[51] A. Nikolaev and M. Batsyn, "Branch-and-bound algorithm for symmetric travelling salesman problem," in *proc. IWOCA*, 2018, pp. 311–322.

[52] W. Zhou, Y. Yang, M. Xu, and H. Chen, "Accommodating Dynamic Traffic Immediately: a VNF Placement Approach," in *proc. IEEE ICC*, 2019, pp. 1–6.

[53] A. Boubendir, E. Bertin, and N. Simoni, "A VNF-As-A-Service Design through Micro-Services Disassembling the IMS," in *proc. ICIN*, 2017, pp. 203–210.

[54] F. Lam and A. Newman, "Traveling Salesman Path problems," *Mathematical Programming*, vol. 113, no. 1, pp. 39–59, 2008.

[55] D. B. West *et al.*, *Introduction to Graph Theory*. Prentice hall Upper Saddle River, NJ, 1996, vol. 2.

[56] G. Sallam, G. R. Gupta, B. Li, and B. Ji, "Shortest Path and Maximum Flow Problems Under Service Function Chaining Constraints," in *proc. IEEE INFOCOM*, 2018, pp. 2132–2140.

[57] E. Guler, D. Zheng, G. Luo, L. Tian, and X. Cao, "Virtual Multicast Tree Embedding over Elastic Optical Networks," in *proc. IEEE GLOBECOM*, 2017, pp. 1–6.

[58] L. M. P. Larsen, A. Checko, and H. L. Christiansen, "A Survey of the Functional Splits Proposed for 5G Mobile Crosshaul Networks," *IEEE Commun. Surv. Tuts.*, vol. 21,

no. 1, pp. 146–172, 2019.

[59] J. Dizdarevic, F. Carpio, A. Jukan, and X. Masip-Bruin, "A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 116:1–116:29, 2019.

[60] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual Network Embedding: A Survey," *IEEE Commun. Surv. Tuts.*, vol. 15, no. 4, pp. 1888–1906, 2013.

[61] W. Wei, H. Gu, K. Wang, X. Yu, and X. Liu, "Improving cloud-based iot services through virtual network embedding in elastic optical inter-dc networks," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 986–996, 2019.

[62] H. Wu, F. Zhou, Y. Chen, and R. Zhang, "On Virtual Network Embedding: Paths and Cycles," *CoRR*, vol. abs/1812.06287, 2018. [Online]. Available: http://arxiv.org/abs/1812.06287

[63] M. Rost and S. Schmid, "Charting the Complexity Landscape of Virtual Network Embeddings," in *proc. IFIP*, 2018, pp. 55–63.

[64] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating Virtualized Network Functions," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 4, pp. 725–739, 2016.

[65] 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.

[66] 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.

[67] 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)*, 2013, pp. 410–414.

[68] 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)*, 2016, pp. 1–9.

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

[70] 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)*, 2017, pp. 1–6.

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

[72] E. Guler, D. Zheng, G. Luo, L. Tian, and X. Cao, "Virtual multicast tree embedding over elastic optical networks," in *IEEE Global Communications Conference (GLOBECOM)*, 2017, pp. 1–6.

[73] S. J. B. Yoo, "Wavelength conversion technologies for wdm network applications,"

Journal of Lightwave Technology, vol. 14, no. 6, pp. 955–966, 1996.

[74] B. C. Chatterjee, N. Sarma, and E. Oki, "Routing and spectrum allocation in elastic optical networks: A tutorial," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1776–1800, 2015.

[75] Y. Zhang, X. Zheng, Q. Li, N. Hua, Y. Li, and H. Zhang, "Traffic grooming in spectrum-elastic optical path networks," in *Optical Fiber Communication Conference (OFC)*, 2011, pp. 1–3.

[76] W. Lu, J. Kong, L. Liang, S. Liu, and Z. Zhu, "How Much Can Flexible Ethernet and Elastic Optical Networking Benefit Mutually?" in *proc. IEEE ICC*, 2019, pp. 1–6.

[77] I. V. Loumiotis, P. Kosmides, E. F. Adamopoulou, K. P. Demestichas, and M. E. Theologou, "Dynamic Allocation of Backhaul Resources in Converged Wireless-optical Networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 2, pp. 280–287, 2017.

[78] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takács, "Optical Service Chaining for Network Function Virtualization," *IEEE Commun. Mag.*, vol. 53, no. 4, pp. 152–158, 2015.

[79] A. Minakhmetov, C. Gutterman, T. Chen, J. Yu, C. Ware, L. Iannone, D. C. Kilper, and G. Zussman, "Experiments on Cloud-RAN Wireless Handover using Optical Switching in a Dense Urban Testbed," in *proc. OFC*, 2020, pp. 1–3.

[80] J. A. Hernández, M. Quagliotti, E. Riccardi, V. López, Ó. G. de Dios, and R. Casellas, "A Techno-economic Study of Optical Network Disaggregation Employing Open Source Software Business Models for Metropolitan Area Networks," *IEEE Commun. Mag.*,

vol. 58, no. 5, pp. 40–46, 2020.

[81] A. Hmaity, M. Savi, L. Askari, F. Musumeci, M. Tornatore, and A. Pattavina, "Latency- and Capacity-aware Placement of Chained Virtual Network Functions in FMC Metro Networks," *Opt. Switch. Netw.*, vol. 35, 2020.

[82] R. Katti and S. Prince, "A Survey on Role of Photonic Technologies in 5G Communication Systems," *Photonic Netw. Commun.*, vol. 38, no. 2, pp. 185–205, 2019.

[83] A. A. Ahmed and A. A. Alzahrani, "A Comprehensive Survey on Handover Management for Vehicular Ad-hoc Network based on 5G Mobile Networks Technology," *Trans. Emerg. Telecommun. Technol.*, vol. 30, no. 3, 2019.

[84] E. Yaacoub and M. Alouini, "A Key 6G Challenge and Opportunity - Connecting the Base of the Pyramid: A Survey on Rural Connectivity," *P. IEEE*, vol. 108, no. 4, pp. 533–582, 2020.

[85] L. Ruan, M. P. I. Dias, and E. Wong, "Towards Self-adaptive Bandwidth Allocation for Low-latency Communications with Reinforcement Learning," *Opt. Switch. Netw.*, vol. 37, p. 100567, 2020.

[86] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: Enabling Network Function Parallelism in NFV," in *proc. ACM SIGCOMM*, 2017, pp. 43–56.

[87] I. Tomkos, D. Klonidis, E. Pikasis, and S. Theodoridis, "Toward the 6G Network Era: Opportunities and Challenges," *IT Prof.*, vol. 22, no. 1, pp. 34–38, 2020.

[88] Y. Shih, H. Lin, A. Pang, C. Chuang, and C. Chou, "An nfv-based service framework for iot applications in edge computing environments," *IEEE Transactions on Network*

and Service Management, vol. 16, no. 4, pp. 1419–1434, 2019.

[89] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, p. 106984, 2020.

[90] S. Barmpounakis, G. Tsiatsios, M. Papadakis, E. Mitsianis, N. Koursioumpas, and N. Alonistioti, "Collision avoidance in 5g using mec and nfv: The vulnerable road user safety use case," *Computer Networks*, vol. 172, p. 107150, 2020.

[91] C. Cicconetti, M. Conti, and A. Passarella, "Uncoordinated access to serverless computing in mec systems for iot," *Computer Networks*, vol. 172, p. 107184, 2020.

[92] G. Castellano, F. Esposito, and F. Risso, "A distributed orchestration algorithm for edge computing resources with guarantees," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 2548–2556.

[93] S. Li, M. Y. Saidi, and K. Chen, "Survivable services oriented protection level-aware virtual network embedding," *Computer Communications*, vol. 152, pp. 34 – 45, 2020.

[94] T. Subramanya, D. Harutyunyan, and R. Riggio, "Machine learning-driven service function chain placement and scaling in mec-enabled 5g networks," *Computer Networks*, vol. 166, p. 106980, 2020.

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

[96] K. Han, S. Li, S. Tang, H. Huang, S. Zhao, G. Fu, and Z. Zhu, "Application-driven

end-to-end slicing: When wireless network virtualization orchestrates with NFV-based mobile edge computing," *IEEE Access.*, vol. 6, pp. 26 567–26 577, 2018.

[97] G. P. Sharma, W. Tavernier, D. Colle, and M. Pickavet, "Vnf-aapc: Accelerator-aware vnf placement and chaining," *Computer Networks*, vol. 177, p. 107329, 2020.

[98] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz, "A survey on 5G networks for the internet of things: communicationtechnologies and challenges," *IEEE Access.*, vol. 6, pp. 3619–3647, 2018.

[99] L. Qu, C. Assi, K. Shaban, and M. J. Khabbaz, "A reliability-aware network service chain provisioning with delay guarantees in nfv-enabled enterprise datacenter networks," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 554–568, 2017.

[100] M. Zhu, Q. Sun, S. Zhang, P. Gao, B. Chen, and J. Gu, "Energy-aware virtual optical network embedding in sliceable-transponder-enabled elastic optical networks," *IEEE Access.*, vol. 7, pp. 41 897–41 912, 2019.

[101] Y. Fouquet, D. Nace, M. Pióro, and M. Poss, "An optimization framework for traffic restoration in optical wireless networks with partial link failures," *Opt. Switc. Netw.*, vol. 23, pp. 108 – 117, 2017.

[102] F. He, T. Sato, and E. Oki, "Backup resource allocation model for virtual networks with probabilistic protection against multiple facility node failures," in *proc. DRCN*, 2019.

[103] S. R. Chowdhury, R. Ahmed, M. M. Alam Khan, N. Shahriar, R. Boutaba, J. Mitra, and F. Zeng, "Dedicated protection for survivable virtual network embedding," *IEEE*

Trans. Netw. Service Manag., 2016.

[104] B. Han, V. Gopalakrishnan, G. Kathirvel, and A. Shaikh, "On the resiliency of virtual network functions," *IEEE Commun. Magaz.*, vol. 55, no. 7, pp. 152–157, 2017.

[105] M. Karimzadeh-Farshbafan, V. Shah-Mansouri, and D. Niyato, "A dynamic reliability-aware service placement for network function virtualization (nfv)," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 318–333, 2020.

[106] M. Wang, B. Cheng, and J. Chen, "Joint availability guarantee and resource optimization of virtual network function placement in data center networks," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 2, pp. 821–834, 2020.

[107] D. Yamada and N. Shinomiya, "A solving method for computing and network resource minimization problem in service function chain against multiple vnf failures," in *proc. IEEE CIC*, 2019, pp. 30–38.

[108] J. Fan, M. Jiang, O. Rottenstreich, Y. Zhao, T. Guan, R. Ramesh, S. Das, and C. Qiao, "A framework for provisioning availability of NFV in data center networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2246–2259, 2018.

[109] A. Engelmann and A. Jukan, "A reliability study of parallelized VNF chaining," in *proc. IEEE ICC*, 2018, pp. 1–6.

[110] P. Jin, X. Fei, Q. Zhang, F. Liu, and B. Li, "Latency-aware VNF Chain Deployment with Efficient Resource Reuse at Network Edge," in *proc. IEEE INFOCOM*, 2020, pp. 1–10.

[111] X. Shang, Y. Huang, Z. Liu, and Y. Yang, "Reducing the Service Function Chain

Bakcup Cost over the Edge and Cloud by a Self-adapting Scheme," in *proc. IEEE INFOCOM*, 2020, pp. 1–10.

[112] G. Sallam and B. Ji, "Joint Placement and Allocation of Virtual Network Functions with Budget and Capacity Constraints," in *proc. IEEE INFOCOM*, 2019, pp. 523–531.

[113] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service Placement with Provable Guarantees in Heterogeneous Edge Computing Systems," in *proc. IEEE INFOCOM*, 2019, pp. 514–522.

[114] T. Kuo, B. Liou, K. C. Lin, and M. Tsai, "Deploying Chains of Virtual Network Functions: On the Relation Between Link and Server Usage," *IEEE/ACM Transaction on Networking*, vol. 26, no. 4, pp. 1562–1576, 2018.

[115] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *proc. IEEE INFOCOM*, 2015, pp. 1346–1354.

[116] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF Scaling and Flow Routing with Proactive Demand Prediction," in *proc. IEEE INFOCOM*, 2018, pp. 486–494.

[117] D. Zheng, H. Gu, W. Wei, C. Peng, and X. Cao, "Network service chaining and embedding with provable bounds," *IEEE Internet of Things Journal*, pp. 1–13, 2020.