A scalable architecture for federated service chaining

A Scalable Architecture for Federated Service Chaining

by

Chen Chen

A Doctoral Thesis

Submitted in partial fulfilment
of the requirements for the award of

Doctor of Philosophy
of
Loughborough University

15th July 2022

Acknowledgement

I am overwhelmed in all humbleness and gratefulness to show gratitude to all those who have helped me in this journey. This endeavor would not have been possible without the support, suggestions, knowledge and expertise that my supervisor Dr. Fung Po Tso shared with me. Words cannot express my gratitude to him for his invaluable patience and hard work. I am extremely grateful that he took me as a student and continued to have faith in me over the years. I would also like to express my deepest gratitude to my supervisor Dr. Lars Nagel who generously offers me plenty of constructive feedback, inspiration and moral support.

Special thanks to Dr. Iain Phillips, for his generous support on providing the Lobster server, invaluable assistance and insights leading to the success of my study. Thanks should also go to Dr. Asma Adnane, especially for her valuable comments and suggestions. I also had the pleasure of working with Dr. Lin Guan, who is an inspiring colleague. I would like to express my sincere gratitude to all other members of the Computer Science Department for their direct and indirect assistance at every stage of my study.

Lastly, I would be remiss in not mentioning my family and friends, especially my parents and my partner. Without their tremendous encouragement and support in the past few years, it would be impossible for me to complete this journey.

Abstract

The orchestration of Service Function Chain in multiple clouds calls for low-cost, low-latency and scalability. In the existing literature, several techniques have been proposed to meet these requirements. However, how to federate service chains across geo-distributed clouds in light of these requirements remains open.

This thesis aims to study how to compose service chains across multiple clouds by considering several factors such as domain autonomy, domain confidential information, scalability, deployment cost, end-to-end latency and dynamic traffic demands. In particular, the proposed schemes in this thesis are devised to improve the most crucial performance metrics: the deployment cost and the end-to-end latency.

First, we propose a distributed architecture that jointly considers domain autonomy, domain confidential information and scalability. This architecture enables service chains across multiple administrative domains without revealing sensitive network information such as the domain topology. The proposed architecture significantly reduces the deployment cost which consists of resource and traffic routing costs. Moreover, the proposed architecture remarkably reduces the execution time which suggests that it processes the SFC requests timely.

Second, the network traffic is dynamic in nature. To accommodate the varying traffic demand in edge clouds, it is important to dynamically scale VNFs in an agile and efficient manner by considering the resource scarcity at the edge. Hence, we propose a bottleneck-aware VNF scaling and traffic routing algorithm to effectively handle the incoming traffic. The proposed algorithm uses vertical and horizontal scaling in light of the VNF category. The experimental results show that the proposed algorithm efficiently shortens the end-to-end latency, improves the VNF utilization rate and reduces the running time.

List of Publications

Conference Publications

1. **Chen Chen**, Lars Nagel, Lin Cui, and Fung Po Tso. Distributed federated service chaining for heterogeneous network environments. In Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing (UCC), December 2021.

2. **Chen Chen**, Lars Nagel, Lin Cui, and Fung Po Tso. B-Scale: Bottleneck-aware VNF scaling and flow routing in edge clouds. In Proceedings of 27th IEEE Symposium on Computers and Communications (ISCC), June 2022.

Journal Publications

1. **Chen Chen**, Lars Nagel, Lin Cui, and Fung Po Tso. Distributed federated service chaining: A scalable and cost-aware approach for multi-domain networks. Computer Networks, 212:109044, 2022.

Contents

List of Abbreviations

5G	Fifth Generation
ADC	Application Delivery Controller
AppFW	Application Firewall
AR	Augmented Reality
DNN	Deep Neural Network
EdgeFW	Edge Firewall
ETSI	European Telecommunications Standards Institute
GPU	Graphics Processing Unit
IDS	Intrusion Detection System
ILP	Integer Linear Programming
IoT	Internet of Things
ISP	Internet Service Provider
MON	Monitoring
NAT	Network Address Translation
NFV	Networking Function Virtualization
NFVI	Networking Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
SDN	Software Defined Networking
SegFW	Segment Firewall
SF	Serivce Function
SFC	Service Function Chain
SFP	Service Function Path
VIM	Virtualized Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VR	Virtual Reality
WAN	Wide Area Network
WL/DB	Workload

WOC Web Optimization Control

List of Figures

List of Tables

Chapter 1

Introduction

The introduction chapter consists of four sections. Section 1.1 presents the motivation. Section 1.2 introduces the research aims and objectives. The original contribution of this thesis is summarized in Section 1.3. Finally, Section 1.4 summarizes the structural organization.

1.1 Motivation

Due to the emerging edge computing and Network Function Virtualization (NFV) technologies, the network becomes increasingly fragmented when more and more network operators join the market. With the ever-growing network administrative domains and applications, we envision that different kinds of networks such as public cloud data centers, ISP networks and edge networks will span across numerous domains in the future network. The end-user experience will increasingly rely on the collaboration of these networks through resource sharing and cooperation.

As illustrated in Figure 1.1, networks consist of multiple domains which contain public cloud data centers, ISP networks or edge clouds. Network operators are notoriously known for restricting the network information exchange as doing so could mean revealing their competitive business advantages. Similarly, distinct business objectives of different network operators mean that they have proprietary network policies and configurations which exacerbate the difficulty of domain federation. To deliver flexible and efficient network services across multiple network domains, there is a need to devise a scalable architecture for service chaining. In this thesis, scalability refers to not only the ability to incorporate a large number of cloud data centers but also the capacity to accommodate time-varying traffic.

However, existing works primarily focus on either a centralized framework or an inter-cloud federation. In centralized frameworks, it is assumed that a centralized controller has full control and visibility of all domains which is not viable. Sim-

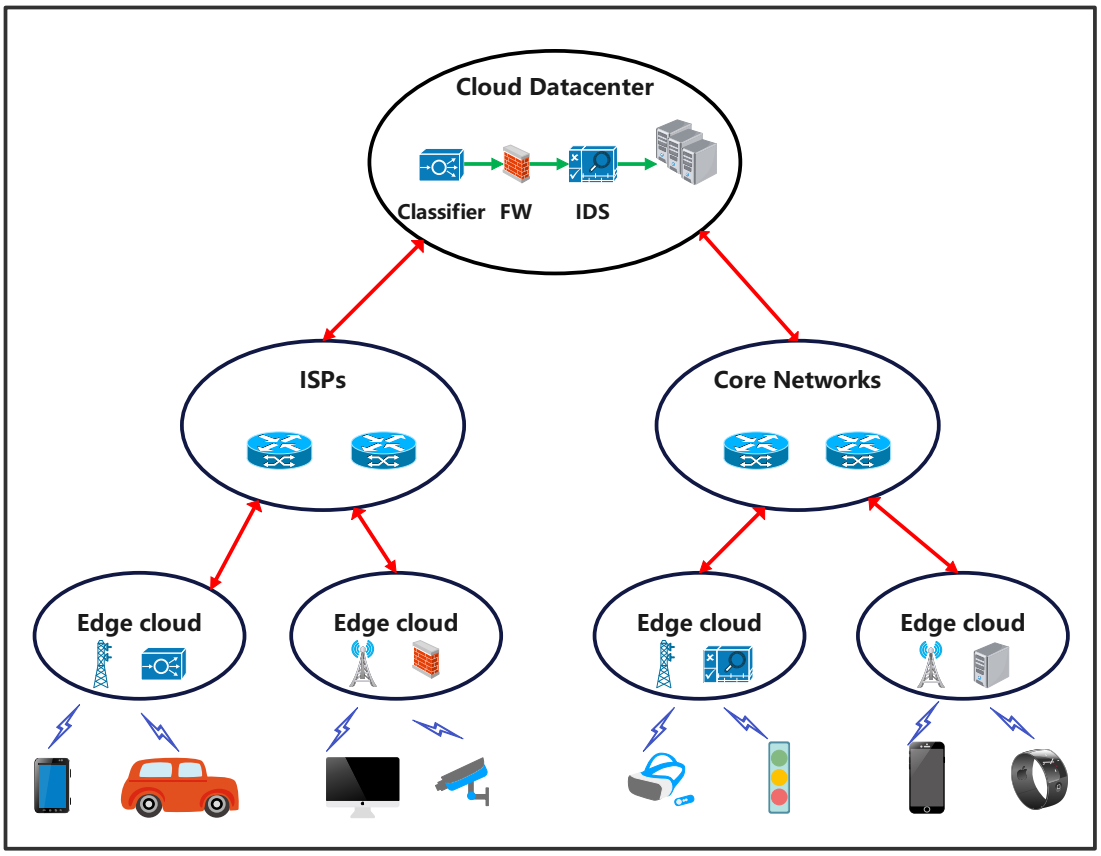


Figure 1.1: Use case: deploying service chains across multiple networks

ilarly, inter-cloud federation usually overlooks the domain autonomy and domain confidential information. Furthermore, current service orchestration approaches are often static and overlook the inherent nature of Service Functions (SFs). This imposes significant limitations on the management of time-varying network traffic and hence deteriorates the end-user experience. Although Software Defined Networking (SDN) and NFV reduce the management complexity of multiple clouds, there is still no efficient mechanism for the federation of different domains. Hence, this thesis is motivated to fill the gap in service chain federation with respect to the scalability.

To improve the system scalability, it is also crucial to study the scalability of Virtual Network Functions (VNFs). This is because the network traffic is time-varying in nature and hence requires elastic solutions for handling dynamics. VNF scaling technique is a promising approach to achieve minimum performance degradation and minimum resource utilization in light of the time-varying traffic workloads. To improve the system scalability, two challenges need to be addressed. We need a placement policy to determine the priority of vertical and horizontal scaling, aiming to achieve optimal performance for Service Function

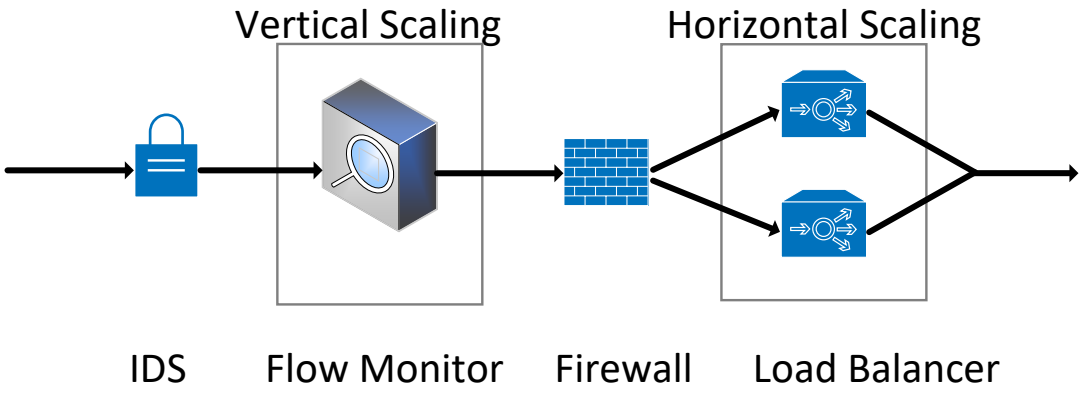


Figure 1.2: An example of VNF scaling

Chains (SFCs). We also need a traffic steering policy that determines how to route traffic flows between multiple VNF instances.

Figure 1.2 illustrates an example of vertical scaling and horizontal scaling. Vertical scaling refers to adding more physical resources to a single VNF instance to improve the processing capacity. Horizontal scaling is creating more than one copy of VNF instances and distributing the network traffic between them.

Keep the above factors in mind, there are a variety of limitations of existing works. Most existing approaches [1], [2], [3] use a centralized controller which assumes that the controller has full control and visibility of all domains. Such a centralized framework cannot alleviate the problem of domain autonomy, domain confidential information and scalability because all the control and information must be given to the centralized controller. Indeed, there are also some approaches [4], [5], [6] that employ a distributed framework to handle the federation of multiple domains. Similarly, many works [7], [8] use the scaling technique to improve the network performance. Meanwhile, a wide range of works [9], [10], [11] reported that the VNF performance is bottlenecked by different types of resources (i.e., CPU, memory or bandwidth). Nevertheless, most of the existing approaches largely overlook the VNF bottlenecks and the resource scarcity. Hence, these approaches cannot jointly achieve excellent performance, time efficiency and scalability.

Therefore, we are motivated to propose a fully-fledged architecture that efficiently federates multiple network domains. In particular, there are two challenges we need to consider.

- First, it is very challenging to jointly consider domain autonomy, domain confidential information and scalability for service chain federation. Domain autonomy refers to that every domain is allowed to implement independent algorithms based on their resource capacities and management policies. Domain confidential information, in this thesis, refers to the network inform-

ation such as topology and network status. Scalability refers to not only the network scale but also the ability to process the sheer volume of traffic demand.

- The time-varying network traffic poses significant challenges to dynamic SFCs. In particular, the key challenge lies in accommodating the fluctuating network traffic because network flows come and leave at any time. Also, most existing works ignore the resource scarcity in edge clouds. Hence, we seek a dynamic and resource-efficient scaling approach for SFC management.

1.2 Aims and Objectives

Motivated by the aforementioned factors, this thesis aims to propose two schemes that build a federated and scalable architecture for SFCs. More specifically, the aims of this thesis are as follows.

- This thesis aims to propose a full-fledged architecture that preserves system scalability and domain autonomy. In light of the sheer volume of requests and the increasing network scale, it is crucial to provide elastic and scalable approaches for service chain federation. With more domains joining the federation, it is vital to enable customized network policies and management strategies as doing so could facilitate every party to optimize its business objective.

- Due to the nature of network traffic, accommodating the time-varying flows is an essential problem for the federated service chaining. To improve the scalability in the VNF-level, this thesis also aims to dynamically handle the fluctuation of the network traffic while minimizing the end-to-end delay.

To achieve the aforementioned aims, we proceed with the following steps.

- Investigate the existing schemes for scalable federation of service chains.

- Formulate mathematical models for SFC placement and flow routing problems.

- Devise a scalable architecture that efficiently copes with the increasing network scale and time-varying traffic demands.

- Devise SFC placement algorithms that achieve near-optimal performance and run in a fast manner.

- Implement the proposed schemes in NS3 and Mininet based testeds to demonstrate the performance and the efficiency.

1.3 Original Contributions

In this section, we summarize our contributions as follows.

- We have proposed a scalable architecture for service chaining that jointly considers scalability and domain autonomy. We have formulated an Integer Linear Programming (ILP) problem, aiming to minimize the deployment cost of SFC requests. Then, we propose Distributed Federated Service Chaining (DFSC) which significantly reduces the overall deployment cost. DFSC distributes the decision process to multiple peer controllers while reducing the amount of required network information. The extensive experiments demonstrate that DFSC significantly reduces the decision-making time, preserves the domain autonomy and improves the scalability of the system. We implemented DFSC and two benchmark approaches from the literature by using not only a simulator NS3 but also an emulator Mininet. In this case, we compare the performance of different testbeds to verify the performance of DFSC.

- We have proposed a VNF scaling algorithm that considers the scalability in the VNF-level. We have formulated an ILP problem which aims to minimize the end-to-end delay in edge clouds. To handle the time-varying network traffic, we propose a hybrid scaling approach. Experimental results show that the proposed scheme reduces 70% execution time in a large-scale network.

1.4 Thesis Organization

The remainder of the thesis is organized as follows.

Chapter 1 illustrates the motivations and aims of this thesis. This chapter highlights the research gaps in the context of service chain federation, including scalability, domain autonomy and confidential information. We show that existing approaches disregard the system scalability as most of them use a centralized controller to manage computing and network resources. Then, we demonstrate the lack of considering the VNF bottlenecks and resource utilization when performing VNF scaling in edge clouds. After that, we describe the overall aims and objectives of this thesis and outline our contributions. Finally, an overview of this thesis is described.

Chapter 2 introduces the background and key concepts of our research including Network Function Virtualization, Service Function Chain, VNF scaling and administrative domains.

Chapter 3 summarizes the existing literature of service chain federation. This chapter demonstrates related works in light of single-domain, multi-domain with centralized architecture, multi-domain with distributed architecture and service chain scaling. The state-of-the-art solutions in the service chaining are summarized. These works aim to improve the system performance including end-to-end latency, deployment cost, energy, resilience and etc.

Chapter 4 presents the system models. This chapter includes the physical network model, the service function chain model, the affinity constraints, the queueing theory to model the latency, the VNF scaling and flow routing. Also, all the symbols and variables for the problem formulation are listed.

Chapter 5 provides a distributed scheme that significantly improves the system scalability in the controller-level. The proposed Distributed Federated Service Chaining approach (DFSC) distributes the decision-making process between multiple controllers. The chapter demonstrates two variants of DFSC, i.e., coarse-grained and fine-grained DFSC. Coarse-grained DFSC provides domain and tier constraints for the SFC granularity while fine-grained DFSC provides these constraints for the VNF granularity. We formulate the SFC placement problem as an Integer Linear Programming problem and prove that it is NP-hard. Extensive simulation results suggest that DFSC remarkably shortens the decision-making time by 70% and reduces the deployment cost by up to 20%.

Chapter 6 presents a hybrid scaling approach which remarkably improves the scalability in the VNF-level. The proposed approach jointly considers the VNF category and resource utilization. This approach aims to improve system scalability by considering the VNF category and resource utilization. We formulate an Integer Linear Programming problem and prove its NP-hardness. The experimental results demonstrate that the proposed algorithm significantly reduces the resource usage while achieving near-optimal latency (9% more than the optimum).

Chapter 7 concludes the thesis. This chapter summarizes the proposed algorithms and discusses the limitations of the proposed schemes. It also demonstrates the future research directions in light of service chain federation.

Chapter 2

Background

In this chapter, we present the key concepts and background in federated service chaining. We envision the widespread network function virtualization and service function chaining as the essential building blocks in multi-domain networks. Also, we present horizontal and vertical scaling because they are pervasive technologies for dynamic SFC orchestration.

2.1 Network Function Virtualization

2.1.1 Definition

In the conventional paradigms of network service, network functions (e.g., firewall or proxy) are usually implemented on the dedicated hardware middleboxes. Due to the emergence of new service paradigms such as edge computing, in-network computing and novel applications such as machine learning functions, hardware middleboxes are facing severe challenges. First, hardware middleboxes are costly due to the expenditure on design and production which significantly deteriorates the revenue of service providers. Moreover, the rigid dedicated middleboxes must be manually configured and managed which makes network management labor-intensive. Therefore, the traditional network service paradigm imposes a variety of limitations on service chain federation in the perspective of operational and capital expenditures.

Network Function Virtualization (NFV), proposed by European Telecommunications Standards Institute (ETSI) [12] in 2012, is a promising approach to leverage software instances of network functions instead of dedicated hardware implementations. In NFV, network services are delivered by a sequence of Virtual Network Functions (VNF) that can run on virtual machines and containers. In contrast to conventional hardware appliances, NFV uses virtualized network functions hosted within virtual machines or containers running on Network Func-

tion Virtualization Infrastructure (NFVI). NFVI refers to all the software and hardware components that provide the environment for NFV deployment. Network Function Virtualization Orchestrator (NFVO) is all functional blocks, data repositories and interfaces that serve the purpose of NFVI and VNF management. NFV decouples the network function from the dedicated hardware which enables flexible and agile network management with significant decrease of expenses. Conventional examples of NFV include firewalls, load balancers, network address translation and etc.

Traffic flows traverse VNFs in a predefined order which is known as the Service Function Chain (SFC). By leveraging the virtualization technologies, the requested services can be deployed in a flexible and efficient manner. Furthermore, if the SFC has been changed, VNFs can be added or deleted at a small cost. NFV technology enables scalable and agile network services, resulting in a more efficient and timely management paradigm. Hence, NFV could advocate service chain federation by reducing the deployment cost and improving the network performance.

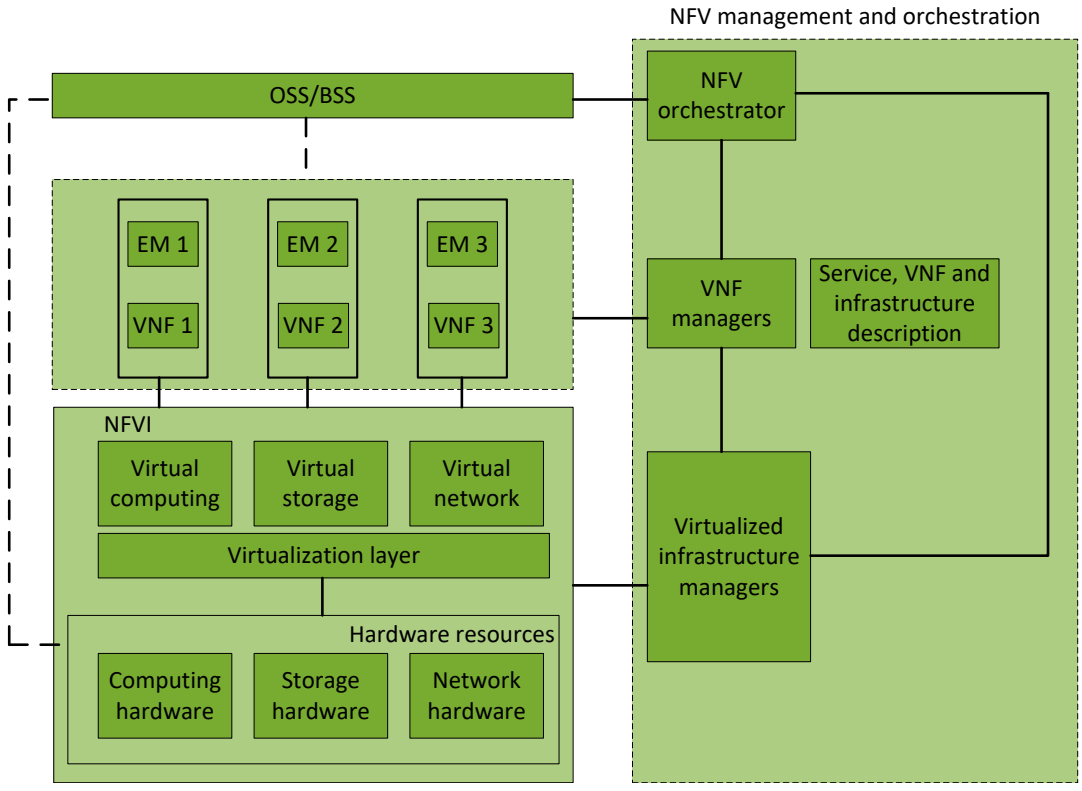


Figure 2.1: The NFVI architecture

Figure 2.1 illustrates the NFVI architecture as defined by the European Telecommunications Standards Institute (ETSI) [13]. The hardware resources include computing, storage and network resources that provide processing, storage and

connectivity capabilities for VNFs by a virtualization layer. The virtualization layer abstracts the hardware resources and provides the virtualized infrastructure for the software. Examples of the virtualized layer include hypervisors and container-based virtualization solutions.

2.1.2 NFV Orchestrator

NFV orchestrator is responsible for managing network services including instantiation, scaling, life cycle and etc. The NFV orchestrator also performs resource management and authorization to resource requests in the NFV.

2.1.3 VNF Managers

The VNF manager is responsible for the VNF lifecycle including instantiation, update, scaling and termination. It also spawns other functions that are essential for the entire VNF lifecycle. Moreover, it coordinates and reports events for other NFVI components.

2.1.4 Virtual Infrastructure Manager

The Virtual Infrastructure Manager (VIM) performs controlling and managing of VNFs in the control of VNFI. It manages the resources and performs resource allocation of NFVI for VNFs. It also performs analysis of the NFVI performance and logs the events. Moreover, it collects and forwards performance and measurement events.

2.2 Service Function Chain

Thanks to the rapid advances in network virtualization, network operators can now deploy many network services upon the virtualized infrastructures by leveraging VNFs. These VNFs replace the dedicated hardware middleboxes such as load balancers, firewalls, deep packet inspectors, intrusion detectors and etc.

Due to the emergence of edge computing and IoT, computation is shifting from public cloud data centers to geo-distributed edge clouds, resulting in deploying the computational and storage resources at the proximity of end-users. This can also reduce the total end-to-end delay and the cost of service providers by reducing the use of pricey WAN links. However, such trends also pose a significant challenge to the service function chaining. Since the network scale and the number of requested services increase exponentially in multi-domain environments [14], it is very challenging to coordinate the services.

2.2.1 SFC Definition

Service Function Chain (SFC) is a set of abstract service functions in a predefined order and the subsequent traffic steering [15]. An example of an abstract service function is a "load balancer". The term "service chain" and "service function chain" are often used interchangeably. Also, a Service Function Path (SFP) is a path that packets of a certain SFC must traverse.

2.2.2 SFC Architecture

Service Function Chain provides a method for deploying SFs in a manner of dynamic ordering and topological independence of SFs as well as the metadata exchange between participating parties. An SFC is an abstracted view of a service at the high level that specifies the ordered set of required SFs. We can use an SFC graph, as illustrated in Figure 2.2, to define an SFC architecture. An SFC starts from the source node and ends at the target node.

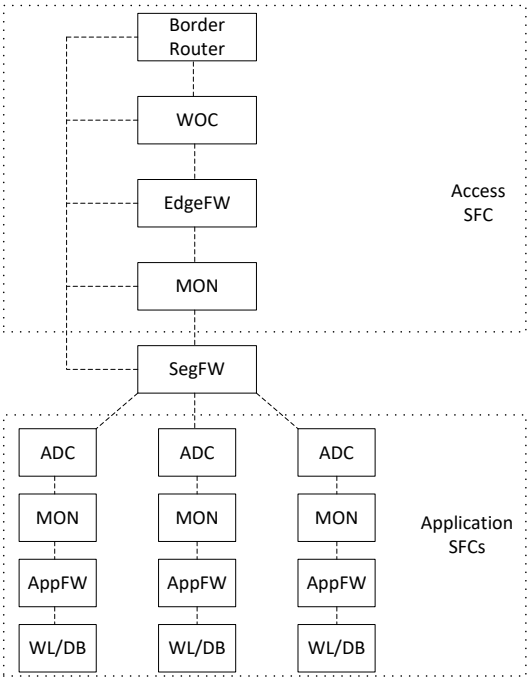


Figure 2.2: Service function chains in the data center

Figure 2.2 shows service function chains in data centers [16]. Access SFCs are responsible for servicing traffic entering and leaving the data center while application SFCs focus on servicing traffic destined to the application. A service chain consists of multiple network functions such as Web Optimization Control (WOC), Edge Firewall (EdgeFW), Monitoring (MON), Segment FW (SegFW), Application Delivery Controller (ADC), Application FW (AppFW), Workload (WL/DB).

Workload refers to a physical or virtual machine that performs a particular task such as web servers, database servers.

Service providers embed a single access SFC and multiple application SFCs for each end-user. Examples of access SFCs include VPN and WOC that support security policies.

SFC includes 6 architectural principles.

- Topological independence: SFC deployment needs no change of the underlay network forwarding topology.
- The service function paths are independent from the packet forwarding.
- Traffic that meets classification rules is steered according to a specific service function path.
- Metadata and context data can be shared between SFs and between external systems.
- The SFC architecture does not rely on the details of SFs.
- The creation, change and deletion of an SFC has no influence on other SFCs.

2.2.3 A List of VNFs

The table 2.1 illustrates a wide range of off-the-shelf VNFs. We use these VNFs in our experimental sections to compose service chains.

2.3 VNF Scaling

2.3.1 VNF Definition

Virtual Network Function (VNF) runs particular network functions on virtual machines or containers in the NFV environment. VNF decouples network functions from dedicated hardware devices with the help of a software layer. Common VNFs include Network Address Translation (NAT), firewalls and routers. VNF increases network scalability and agility. VNF also enables better use of network physical resources. Other advantages include the reduction of power consumption and the reduction of deployment cost. It is common to implement service chains using VNF instances hosted on VMs or containers in a cloud computing platform. VNF scaling is a promising technique to improve system scalability.

Table 2.1: Examples of VNFs

VNFs	Description
Firewall	A network security function that monitors and filters inbound and outbound network traffic based on specific security policies.
NAT	Network address translation. A network function that maps an IP address space into another by changing the network address in the packet header while packets traverse a traffic routing device.
Load balancer	A network function that distributes network traffic across a number of servers. It improves the overall performance of applications by mitigating the burden on servers.
Proxy	An application that performs as an intermediary between clients and servers.
IDS	Intrusion detection system. A network function that monitors the network in light of malicious events or policy violations.
Caching	A network function that stores data to respond faster. The data can be the result of an earlier task or a copy of data stored elsewhere.
WAN-optimizer	A network function that optimizes the data transfer across the wide area networks (WAN).

2.3.2 Vertical Scaling

Vertical scaling (scaling-up) refers to adding more physical resourcs to a single node, typically involving CPUs, memory or storage. In other words, this way of scaling allows resizing of the virtual machine by changing the amount of CPU and memory. Consequently, the processing capacity of the node increases. Vertical scaling is limited by the resource capacity of a single server due to the physical machine capacity.

2.3.3 Horizontal Scaling

On the other hand, horizontal scaling is adding or removing new VNF replicas which is also known as scale-out and scale-in. Nevertheless, horizontal scaling may lead to resource over-provisioning and under-utilization because different VNFs are bottlenecked on different resources. Moreover, it is time-consuming to startup new VNF instances which deteriorates the VNF performance.

2.3.4 Comparison of Vertical and Horizontal Scaling

Scaling period: Horizontal scaling approach is limited by the VM or container startup time as it leverages VM or container as the scaling unit. Mao *et al.* [17]

investigated the VM startup time of three common cloud providers, e.g., Microsoft Azure, Amazon EC2 and Rackspace. The average startup time of these 3 providers are 356.6 seconds, 96.6 seconds and 44.2 seconds, respectively. Hence, horizontal scaling needs to be performed ahead of time to handle the time-varying traffic demand. In contrast, vertical scaling dynamically adds more or less number of vCPUs and memory of the VM at runtime. WindRiver [18] reports that it is possible to hotplug a CPU in about 40ms and unplug the CPU in around 20ms. Therefore, the vertical scaling time can be negligible compared with the VM startup time.

Compatibility: VNF instances are usually bounded by at least one type of resource, e.g., CPU, memory and network bandwidth. Hence, some VNFs cannot improve their performance by vertical scaling. A potential solution is running multiple instances and configuring them to handle a few flows.

Robustness: Vertical scaling can avoid instance overloading by adding more computational resources and thus makes the instance resilient to failure. Multiple static instances can be a single-point-of-failure when performing horizontal scaling. However, for vertical scaling, the scalable VNF instance could be a single-point-of-failure.

Management complexity: Horizontal scaling needs to schedule flows among multiple VNF instances, resulting in data synchronization and load balancing. Moreover, the placement of new instances needs to be timely decided. In contrast, vertical scaling processes all the traffic in one VNF instance and thus reduces the management complexity. In summary, vertical scaling has lower management complexity than horizontal scaling.

2.4 Administrative Domains

Currently, service providers need to manually manage and configure the SFC [19]. Figure 2.3 shows a high-level view of the required components to manage SFC in a single administrative domain [20]. The network operator manages all resources and maintains 2 components: service chain management and resource management.

Service management provides interfaces with the network operators and comprises SFC provision and configuration, portability and interoperability.

SFC provision and configuration include automatic deployment of VNFs based on the requested service and resources. Portability and interoperability provide data management for customers and NFs across multiple domains with a unified operation interface.

The resource management maintains a catalog for a set of resources and their attributes. The resources can be required by either containers or app-level services.

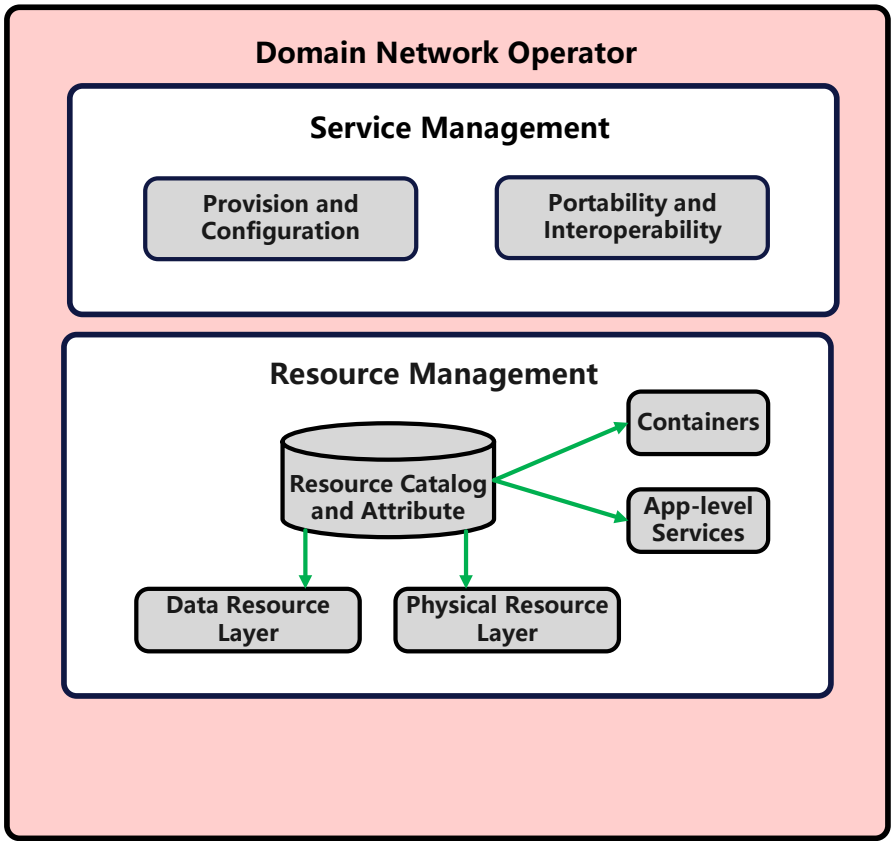


Figure 2.3: Architecture for single-domain networks

Meanwhile, the fragmentation of multiple domains exacerbates the difficulty of delivering such services. In addition, the dynamic nature of networks imposes a variety of limitations on service deployment. The SFC needs to be scaled up or scaled out by considering the incoming traffic flows. Hence, there is a need for an efficient, automatic and dynamic SFC paradigm that saves cost, time and labor to benefit the network operators as well as end-users.

Domain Autonomy

Network operators need to maintain autonomy within their domains. In particular, the domain autonomy in this thesis refers to that every domain can embed different SFC placement algorithms independently. This is of significant importance to network operators as sharing the control of domains leads to severe security risks.

Domain Confidential Information

Service chain federation poses the risk of exposing sensitive information to other network operators. Examples of sensitive information include network topology, link status, resource utilization in data centers and etc. Sharing such information could weaken their business-competing advantages. Hence, domain confidential

information is of significant importance for multi-domain networks.

Domain Scalability

In the service chain federation, the number of participating domains leads to a significant increase in the network scale. Hence, it is vitally important to devise a resource management and SFC placement scheme that scales well to large-scale networks. Moreover, considering the sheer volume of incoming SFC requests compared with the system capacity, it is crucial to improve the system scalability in terms of processing capacities and bandwidth [21].

2.5 Summary

This chapter illustrates the background and key concepts of this thesis. We present the essential definitions of NFV, SFC, VNF scaling and administrative domains in the context of the SFC federation.

We first introduce the NFV technique. In particular, we illustrate the NFV definition and the NFVI architecture. Then, we present the definition of SFCs, the SFC architecture and a list of off-the-shelf VNFs. After that, we present VNF scaling by introducing vertical and horizontal scaling, respectively. We also summarize the advantages and disadvantages of vertical and horizontal scaling. Finally, we illustrate the concept of administrative domains and exemplify the idea by presenting a single-domain architecture.

In the next chapter, we summarize the existing literature on SFC deployment and analyze their benefits and drawbacks.

Chapter 3

Literature Review

In this chapter, we study the topic of federated service chaining by reviewing the existing literature and enabling technologies. Federated service chaining is distributed in nature and hence requires agility, cost optimization, resources optimization, innovative services and optimized performance for tailored services.

In this thesis, administrative domains refer to domains that map to different parities and therefore may include a variety of service providers. Within one administrative domain, multiple technology tiers can coexist based on the type of networks such as public cloud data centers, ISP networks and edge networks. Federated service chaining is referred to as services across the federation of multiple administrative domains.

To support federated service chaining, recent works have proposed approaches that aim for different objectives such as end-to-end latency and deployment cost. Since the existing works pave the way toward the federated service chaining paradigm, we present an overview of the state-of-the-art literature, demonstrating their methods and limitations.

3.1 Single-domain Service Chain

Most of the literature on service chaining focuses on a single domain by considering end-to-end delay, efficient resource allocation and deployment cost.

In this section, we discuss the SFC placement problem for single-domain networks. The single domain, in this thesis, refers to as a single administrative domain that is operated by a single network operator. One domain could contain one or more clouds. The SFC placement problem for single-domain networks has been well studied. An integer linear programming problem is usually used to model such problems with a variety of constraints on network resources, the end-to-end delay and etc. Moreover, the VNFs are required to be placed in a predefined order

which makes the problem more complicated. Related works fall into a number of categories based on different objectives as follows. We have selected two most important objectives, i.e., deployment cost and latency, to formulate our research problems.

3.1.1 Deployment Cost

The deployment cost minimization has received tremendous attention as it is tightly coupled with the revenue of service providers. Hence, many research efforts focus on the minimization of the deployment cost by using means of approaches. The deployment cost ubiquitously contains computational resource cost, traffic routing cost, migration cost and etc.

Chen *et al.* [22] formulate a service placement problem that aims to minimize the overall cost including deployment cost, server usage cost and energy consumption cost. The cited paper jointly considers the service-placing decision, the user allocation decision and the end-to-end latency. Also, the cited paper leverages a local-search based algorithm which starts with a feasible solution and then iterates over local solutions. Extensive experimental results show that the proposed algorithm achieves provable performance with a guarantee. Shang *et al.* [23] investigate the SFC placement problem by considering the operational cost. It is reported that chasing the lowest cost leads to network congestion on popular links and hence deteriorates the performance. To address this issue, a candidate path selection scheme is proposed to jointly reduce the operational cost and avoids network congestion.

Golkarifard *et al.* [24] investigate the VNF placement problem with respect to a variety of practical factors such as request admission, resource activation, resource allocation and traffic routing. A one-shot problem is formulated to optimize the overall deployment cost. Then, an online algorithm based on limited knowledge within each time slot is proposed to solve the problem for large-scale networks. Zheng *et al.* [25] comprehensively study the cost minimization problem for SFCs with provable bounds. It formulates a service chaining and embedding problem that aims to optimize the deployment cost. Then, a cost factor-based algorithm, which guarantees a 2-approximation bound, is proposed.

Zhang *et al.* [26] propose an online adaptive VNF deployment algorithm for 5G networks by leveraging a demand-supply model to represent VNF interference for both edge cloud servers and public cloud servers. Octans [27] aims to maximize the overall throughput of all SFCs for many-core systems. However, cross-node memory access and other factors may deteriorate the network performance. Hence, Octans proposes a performance drop index to evaluate the throughput degrada-

tion and two online algorithms to find near-optimal solutions. Chen *et al.* [28] formulate a mixed integer linear programming problem which considers SFC with different latency requirements. In particular, the nature of services is taken into account. For example, for delay-sensitive services and delay-tolerant services, different placement strategies should be applied. Then, an approach based on a modified shortest path algorithm is proposed to address this problem.

Xu *et al.* [29] formulate an SFC placement problem, aiming to maximize the overall revenue of all stakeholders in the market. Similarly, Song *et al.* [1] study the SFC placement problem on edge and public clouds. The aim is to optimize the maximum link load ratio while fulfilling the service level agreements. Then, a randomized rounding approximation approach is proposed to efficiently solve the problem.

SSCO [30] studies the SFC deployment cost problem by leveraging federated reinforcement learning. By acquiring the local knowledge of residual resources and instantiation costs, the federated-learning scheme efficiently reduces the deployment cost. OKpi [31] formulates a VNF placement problem which considers not only traditional network metrics such as latency and throughput but also some novel metrics such as availability as well as reliability. OKpi advocates not only VNF placement but also the selection of radio access points within polynomial computational time. Okpi incorporates graph theory and optimization in a unique fashion and hence closely matches the optimum.

Gao *et al.* [32] investigate the VNF placement and scheduling problem that aims to optimize the cost of leasing VMs. This cited paper carefully considers several practical factors such as the latency incurred by the start time of VM and VNF instances. Also, due to the inherent nature of different VNFs, models the throughput as a function of allocated resources accordingly. Compared with cost-efficient proactive VNF placement schemes, the proposed approach can achieve lower cost and latency. Zheng *et al.* [33], for the first time, propose the hybrid SFC placement problem that aims to optimize the end-to-end latency cost. The hybrid SFC consists of not only the upstream traffic but also the downstream traffic. This is because remote edge/cloud servers may return the proposed traffic to the customer. An optimal hybrid placement algorithm is proposed based on an auxiliary graph. The experimental results show that the proposed algorithm finds the optimal solution with much less execution time compared with a brutal force algorithm.

Yu *et al.* [34] present ElasticNFV which takes timely resource demand of several service chains into consideration and uses an elastic framework to allocate resources. The aim is to jointly minimize the migration time and deployment cost of SF instances. Meanwhile, many research efforts have focused on the network

policy composition across geo-distributed clouds. Li *et al.* [35] formulate a revenue optimization problem with constraints on the end-to-end latency, resource demand and reliability. To solve this problem, a dynamic algorithm is proposed based on the genetic algorithm and the greedy algorithm. Hope [36] is proposed to overcome the heterogeneity issues in the SFC placement problem. HOPE aims to minimize the network cost by guaranteeing that SFs are optimally placed.

Zhou *et al.* [37] investigate the holistic cost efficiency in cross-edge networks. The deployment cost contains resource cost and traffic routing cost. By leveraging an approximate optimization technique, the proposed problem is decomposed into a number of sub-problems and solves it by rounding the fractional solution. Farkiani *et al.* [38] propose a novel prioritized SFC deployment problem. Then, a heuristic algorithm and an exact method are proposed to solve this problem.

Eramo *et al.* [39] propose a proactive approach to accommodate the fluctuating traffic in peak hours. The approach is based on the Viterbi algorithm that remarkably reduces the computational complexity. FlexShare [40] aims to reduce the SFC deployment cost by sharing the NF instances. The problem is tremendous challenging as sharing such instances needs to balance the workload from different SFCs. FlexShare is proven to be within a constant factor to the optimum and performs well in extensive experiments. CLF [41] aims to provide low-cost and high-performance SFC.

Pei *et al.* [42] propose a binary integer programming problem, aiming to minimize the embedding costs. The resource bottlenecks are regarded as embedding costs and hence the solution successfully avoids resource bottlenecks. By this means, the proposed algorithm significantly improves the request acceptance rate as well as the network throughput. Kariz [43] studies the distributed service function chaining which distributes the SF instances to different locations.

Bari *et al.* [44] investigate the SFC placement, aiming to optimize the network utilization rate and operational costs. The problem is intractable because the required number and placement of VNFs is difficult to determine. The trace-driven simulations justify that the proposed algorithm is within a factor 1.3 of the optimal solution. Cohen *et al.* [45] address the problem of placing SFs within a physical network. The deployment cost comprises two parts: the setup costs of SFs and the distance cost between users. By theoretically proving the performance, the proposed algorithm guarantees a near-optimal performance.

3.1.2 End-to-End Latency

Recently, 5G and edge computing empower the advance of latency-sensitive applications such as Augmented Reality (AR), online machine learning and Virtual

Reality (VR). Many research efforts have focused on the end-to-end latency, aiming to fulfill stringent latency requirements of their applications. In this subsection, we summarize the related works that focus on latency minimization in SFC deployment.

Gharbaoui *et al.* [46] use an experimental prototype to reproduce VNF placement in geo-distributed edge clouds. The cited paper provides a latency-aware algorithm that jointly considers resilience and adaptation. The experiments show that the proposed algorithm achieves a mean of 44.38 seconds with a standard deviation of 4.23 seconds in terms of response time. Liu *et al.* [47] formulate an ILP problem that aims to minimize the end-to-end latency for IoT applications. The cited paper proposes a Lagrangian relaxation heuristic-based approach to approximate the optimal solution. Extensive experimental results prove that the proposed algorithm performs well in terms of the end-to-end latency and acceptance ratio with theoretical bounds.

Pandey *et al.* [48] propose EdgeDQN which aims to simultaneously optimize the underlying resource utilization and the end-to-end latency for multiple SFCs. The authors assume that edge clouds can rent network resources from neighbors and remote clouds in the worst case. The proposed EdgeDQN is built on Q-learning and deep Q-network algorithms in terms of cumulative standard deviation, cumulative reward and learning convergence time for over 400 test cases. Magoula *et al.* [49] study the SFC deployment problem for specific 5G vertical industries such as Industry 4.0 and industrial IoT. A location-aware genetic algorithm based approach is proposed to minimize the end-to-end latency. Evaluation results show that the proposed algorithm achieves a near-optimal solution as well as a low execution time. The rationale is that the proposed algorithm uses an early stopping criterion.

Li *et al.* [50] propose an SFC deployment algorithm based on reinforcement learning. The proposed algorithm aims to learn from the network system and then decide the embedding of SFC requests. Extensive experimental results show that the proposed algorithm adapts to large-scale networks and outperforms other approaches from the literature. Change [51] proposes an online SFC deployment scheme which is managed by users. Change aims to minimize the latency at edge networks in light of user mobility, edge capacity and service migration.

Wang *et al.* [52] formulate an SFC deployment problem by considering the end-to-end latency. The problem jointly considers the server, wired link resources and wireless radio resources. Also, a natural policy gradient is used to train a deep neural network which can avoid local optimum. Masahiro *et al.* [53] focus on jointly resolving the shortest path tour problem and the SFC placement problem.

JOS [54] studies the end-to-end delay minimization problem in heterogeneous

edge clouds. By employing a game-theoretic approach, the proposed algorithm achieves a constant approximation factor 2.62 to the optimum. Liu *et al.* [55] study the SFC placement problem in hybrid edge clouds. Similarly, it aims to minimize the end-to-end delay by leveraging an IoT deep reinforcement learning approach.

FlexChain [56] enables the SFC parallelism to reduce the end-to-end latency. FlexChain jointly considers the SFC parallelism and placement with a performance guarantee. Extensive experimental results show that FlexChain significantly improves the acceptance ratio for latency-sensitive services. Schneider *et al.* [57] propose a scalable approach based on deep reinforcement learning. The proposed algorithm makes rapid deployment decisions in parallel with other nodes. Each agent requires only local knowledge and hence scales independently from the size of the network. The decision time is less than 1 ms, and the algorithm achieves comparable performance in terms of latency and network throughput.

Instead of parallelizing SF instances across different servers, PPC [58] proposes to only parallelize SFs that could indeed shorten the end-to-end delay. Dab *et al.* [59] formulate an ILP problem, aiming to optimize the end-to-end latency as well as the deployment time. Then, a novel online approach is derived from the Maglev algorithm which determines the cloud-native network functions at each level of the network services. The experiments were conducted in a real-world testbed in Kubernetes.

Xie *et al.* [60] formulate an ILP problem that aims to optimize both the latency and the maximum load. The cited paper first conducts real experiments on a wireless router and reports that the communication between VNFs can consume a significant amount of CPU on edge devices. Then, an online algorithm, which distributes the VNFs partially on peer edge devices with low workloads, is proposed. Marcel *et al.* [61] investigate the runtime traffic scheduling for service chains, aiming to optimize system utilization and service quality. The proposed algorithm uses an integer allocation maximum pressure policy to optimize the network throughput.

Jin *et al.* [62] investigate a joint problem of optimizing the resource utilization and the end-to-end latency. By employing a two-stage scheme, a near-optimal solution is achieved with a performance guarantee. EC-HSFP [63], for the first time, addresses the problem of optimizing the end-to-end latency in the hybrid SFC composition and embedding problem. EC-HSFP mainly investigates two scenarios: (a) every physical node only provides one unique SF. (b) every physical node provides multiple SFs.

Dab *et al.* [64] investigate the SFC placement problem for 5G traffic steering in NFV ecosystem. Then, an optimized network-aware load balancing approach,

which is derived from a dynamic round-robin algorithm, is proposed. Gao *et al.* [65] investigate the problem of the access network selection and the service placement for mobile edge computing. The long-term optimization problem is decomposed into a variety of one-shot problems. Then, an iteration-based algorithm is proposed with guaranteed optimal gap.

Poularakis *et al.* [66] study the service placement problem with respect to the data that need to be stored to enable service execution. Multidimensional constraints are considered, including storage, computation and communication for the mobile edge computing networks. A randomized rounding algorithm is proposed which achieves a near-optimal performance among low-latency edge cloud servers. Yang *et al.* [67] study the problem of placing totally ordered and partially ordered SFCs, aiming to minimize the end-to-end delay.

Dimitrii *et al.* [68] study the integer multi-commodity-chain flow (MCCF) problem, aiming to optimize network quality of service constraints. A meta path composite variable approach, that stays within 99% to the optimum, is proposed to solve this problem. Natif [11] studies the SFC placement by considering the performance of SFs. Since software applications are ubiquitously bottlenecked by either CPU or I/O resources, a VNF-aware scheme is proposed to dynamically instantiate SFs and steer traffic.

Sun *et al.* [69] propose a holistic workflow-like SFC request scheme. A dynamic minimum response time considering same level algorithm is proposed to deploy the SFC requests onto edge clouds. Cziva *et al.* [70] study the SFC placement problem in edge clouds, aiming to minimize the end-to-end delay from end-users to the SFs. By leveraging an optimal stopping theory, a proposed algorithm schedules the SFC placement based on temporal latency fluctuations.

3.1.3 Resilience

Shang *et al.* [71] study the VNF backup problem with respect to the cost. The aim is to achieve cost-efficient and resilient SFCs in light of the limited resources at edge clouds. Then, this cited paper proposes a resilience-aware adaptive algorithm, which uses both static and dynamic backups, to guarantee the reliability under the limited capacity of edge clouds. Also, rigorous theoretical analysis is provided for the performance bounds. Extensive experimental results show that the proposed algorithm provides remarkably better reliability with lower backup costs.

Alleg *et al.* [72] formulate a mixed integer linear programming problem, aiming to achieve VNF recovery after failures. The proposed algorithm leverages a resource pool and standby backups to achieve selective diversity and tailored policies. Meanwhile, the proposed algorithm aims to reduce the inherent cost incurred by

VNF backup. Extensive experiments justify that the proposed algorithm can not only improve the VNF resilience but also avoid resource over-provisioning. Qu *et al.* [73] study the reliability problem of SFC by considering the VNF backup. A deep reinforcement learning approach is proposed based on priority. The proposed algorithm determines the backup policy for each VNF, aiming to optimize latency, survival rate and load balancing.

Siasi *et al.* [74] investigate the SFC placement problem for fog computing. The aim is to address the service survivability by taking stringent failure recovery into consideration. Also, an elastic protection scheme is proposed to recover failure in nodes and links. Engelmann *et al.* [75] propose Generic SFC which considers heterogeneity, disjointness, sharing, redundancy and failure. The proposed approach is based on combinatorics and a reduced binomial theorem.

Chen *et al.* [76] study the resilient service chaining placement problem, aiming to minimize the overall latency of flows. A solution with a performance guarantee is proposed to make the problem tractable. CMAB [77] investigates a combinatorial multi-armed bandit problem, which focuses on the VNF backup selection.

Thiruvasagam *et al.* [78] investigate the SFC placement problem while fulfilling the latency and reliability requirements. Then, the reliability of virtual monitoring functions is considered in light of the service degradation. The experimental results show that the proposed algorithm achieves a factor 1.05 to the optimum. Jia *et al.* [79] formulate the SFC backup problem for NFV-enabled 5G network as a mixed integer non-linear programming problem. This cited paper assumes that SFC requests arrive randomly and each server can host all types of VNF. The aim is to optimize the acceptance rate while satisfying the reliability and latency requirements.

Sharma *et al.* [80], for the first time, jointly consider the availability and delay guarantees for SFC placement problem from the theoretical perspective. In particular, the focus is how many SFCs can be deployed within a single data center. Wang *et al.* [81] first report that some small-size flows can lead to network congestion and high delays if they are queuing behind large flows. Then, a parallelized SFC placement problem is formulated for data center networks with respect to the availability and network resources. By splitting large data flows into several small sub-flows, the proposed approaches significantly reduce the end-to-end latency and improve the availability.

Sen *et al.* [82] investigate the resilient SFC placement problem by considering that the primary and the backup service providers can be simultaneously unavailable. Hence, a notion of utility is proposed for a routing path between ingress and egress node pairs to provide reliable utilities. Yin *et al.* [83] investigate the VNF backup problem with respect to both VNFs failures and physical node failures.

Then, a dynamic programming-based placement algorithm is proposed to partition the problem into several sub-problems which are solved as one-shot problems.

3.1.4 Energy

Thanh *et al.* [84] aim to provide a resource and energy-efficient solution for IoT SFCs. The IoT SFC is modeled and then an optimization problem is formulated, aiming to strike a nice balance among resource allocation, energy efficiency and resource dynamics. Then, a smart traffic monitoring IP camera is implemented as the use case to investigate the resource and energy efficiency. The proposed algorithm handles dynamic traffic load and outperforms some existing solutions from state-of-the-art. Khoramnejad *et al.* [85] investigate the SFC offloading problem by partially offloading VNFs onto MEC servers. The overall objective is to optimize the energy consumption in user equipment. By using a double deep Q-network algorithm, the proposed algorithm achieves competitive results against an exhaustive search algorithm.

Sun *et al.* [86] propose an energy-efficient routing algorithm by reusing open servers and load balancing. The proposed algorithm achieves a significant reduction in energy consumption. Also, the average deployment time is reduced by a factor 40 compared with an improved Markov algorithm. Zhou *et al.* [87] propose an auction scheme to encourage service providers to submit bids for demand. By leveraging the proximal Jacobian alternating direction method of multipliers, the proposed algorithm optimizes the social welfare in a distributed manner. Sun *et al.* [88] study the energy minimization problem for online SFC requests. The proposed algorithm is based on a low-complexity heuristic which makes it suitable for fast responding to online SFC requests. The domain confidential information of each participating cloud is efficiently preserved.

ESSO [89] aims to optimize the carbon footprint of SFC placement. ESSO leverages an opportunistic SFC adaption to take advantage of surplus renewable energy. The trace-drive simulations demonstrate that ESSO reduces the carbon footprint by 2-3 times in a small-scale network. Mukherjee *et al.* [90] propose a power-aware approach for cloudlet selection in multi-cloud environment.

Tajiki *et al.* [91] investigate the resource allocation problem in SDN-based networks, aiming to minimize the energy consumption. By proposing a near-optimal heuristic approach, the proposed algorithm achieves a factor 1.14 to the optimum. Jang *et al.* [92] study a multi-objective optimization problem that aims to jointly optimize the energy consumption and the acceptable flow rate. By leveraging linear relaxation and rounding algorithms, near-optimal performance is achieved while accommodating more requests.

3.1.5 Resource Management

Finedge [93] proposes a resource management platform to cope with the resource limitation, real-time traffic fluctuations and stringent QOS requirements for edge networks. By considering the flow-level patterns and the effect of resource allocation, the proposed platform efficiently allocates the most suitable CPU core and corresponding quota to each SF. Extensive experiments show that Finedge can handle a variety of flows with the lowest CPU quota while meeting the service level agreement. Le *et al.* [94] propose a non-cooperative game theory that maps the SFC configuration problem to the Nash equilibrium of the formulated game. A realistic NFV model is reported which considers the fluctuation of actual perceived rates. Then, this cited paper derives the algebraic representation and potential function from the semi-tensor-product framework. Finally, the SFC composition is determined by exploiting the properties of the resulting potential games.

3.2 Centralized Service Chain in Multi-domain Networks

With the advent of edge computing and IoT, more and more network operators join the NFV marketplace, resulting in rendering networks more fragmented. The existing research efforts fall into two categories, i.e., centralized and distributed approaches. The current centralized approaches largely rely on a centralized network controller that has full control and visibility of every network domain as shown in Figure 3.1.

Figure 3.1 illustrates a network with 3 domains and a centralized controller. The controller has full control of SFCs and resources across these domains. Each domain consists of 3 layers, i.e., SFC layer, virtualization layer and physical layer.

3.2.1 Network Latency

Much of the literature has focused on the network latency because the end-to-end latency is a key metric to improve the experience of end-users.

Liu *et al.* [95] study the SFC placement problem for edge clouds and IoTs. The intrinsic nature of IoT devices and mobile edge clouds are jointly considered. By leveraging a quantum machine learning approach, a factor 1.1 to the exact solution is achieved while reducing the run time by 8x compared with the Viterbi algorithm. Son *et al.* [96] propose a dynamic service provisioning algorithm to federate both edge and cloud networks. The time-varying network traffic is considered to automatically allocate resources for latency-sensitive applications.

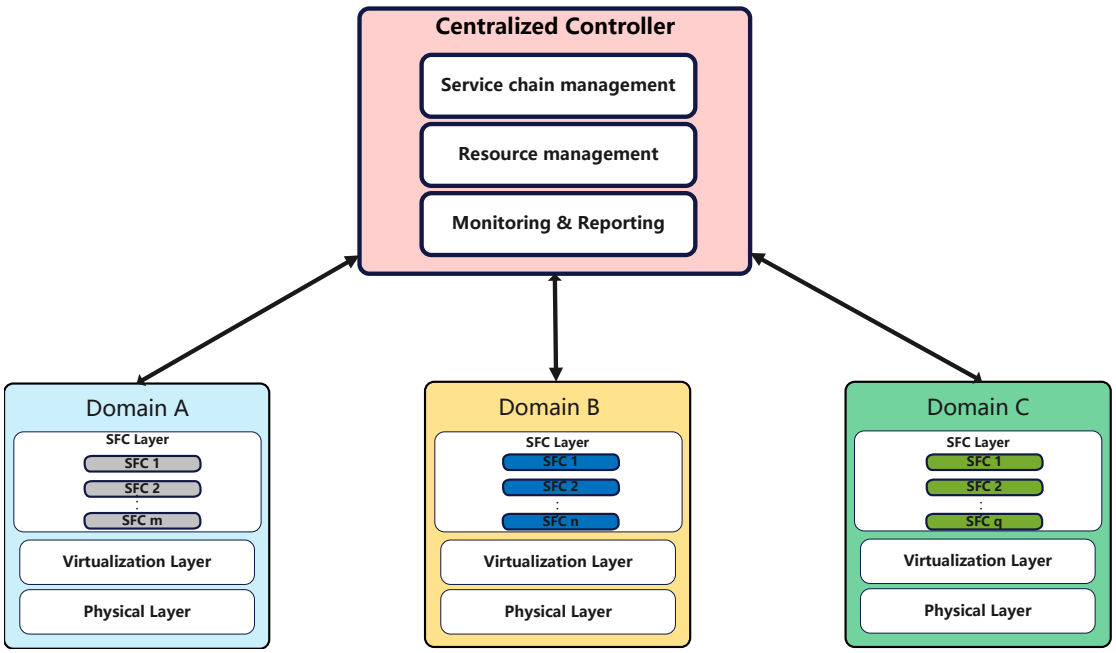


Figure 3.1: Centralized architecture for multi-domain networks

SFCO-AMD [3] studies the SFC orchestration problem across multi-domain networks by considering the confidentiality of network topology. SFCO-AMD first employs a full-mesh aggregation approach to simplify the network topology, and then employs two heuristic approaches to minimize the end-to-end delay. Bhamare *et al.* [97] formulate an ILP problem for the optimal SFC placement in light of inter-cloud response time. The link and computational delays are considered to meet the stringent service level agreements. Finally, an affinity-based approach is proposed to find the near-optimal solution.

Cappanera *et al.* [98] study the SFC placement problem from multiple-stakeholder perspective. By leveraging a layered auxiliary graph, the cited paper aims to fulfill the stringent requirement of subscribers' preferences as well as the quality of services. Xu *et al.* [99] envision that SFC across multi-domain would inevitably increase the end-to-end latency and hence become intractable. An exact approach and a Viterbi based algorithm are proposed to efficiently find near-optimal solutions. pSMART [100] envisages that network operators are notoriously known for inhibiting the network information exchange. As a consequence, the lack of detailed network information renders the multi-domain SFC orchestration intractable. Thus, pSMART aims to balance the privacy and network delay trade-off. pSMART leverages a topology aggregation technique to simplify the network topology and hence preserves the domain confidential information.

Although these approaches pave the way toward service chaining federation, the existing solutions still suffer from the sheer volume of network resources and

network scales. Most of the centralized solutions fail to handle a large number of SFC requests in a fast manner as all the requests are processed in one controller. In other words, centralized approaches pose tremendous limitations on scalability.

3.2.2 Deployment Cost

Since the SFC deployment cost is tightly coupled with the revenue of network operators, many existing works have proposed a variety of methods to minimize the deployment cost while meeting the service level agreements.

Peng *et al.* [101] assume that network applications are divisible in the form of service function chains. Then, Open Jackson queuing network is used to model the optimization problem of long-term deployment cost. A cost-aware algorithm is devised based on Lyapunov drift-plus-penalty function to optimize the cost in a time-slot manner. Samanta *et al.* [102] jointly consider latency-tolerant and latency-constraint services to optimize service latency and revenue. A priority-based management scheme is used to differentiate services and hence reduces the service latency. CRSO [103] investigates the service cost minimization problem of SFC across multi-domain networks. Meanwhile, CRSO provides an economical redundancy approach to guarantee resilient requirements.

Similarly, Zhou *et al.* [37] propose a novel method to jointly reduce the resource cost as well as the traffic routing cost. A regularization approach, that decomposes the formulated problem to a number of one-off fractional problems, is used to provide a near-optimal solution. Zhang *et al.* [104] investigate the SFC placement problem in multi-domain networks by proposing a multi-objective optimization problem. The aim is to minimize the deployment cost while fulfilling the constraint of end-to-end delay.

MOSC [105] studies the problem of minimizing the operational cost by outsourcing SFC to remote public clouds. MOSC assumes that different domains provide different pricing schemes of network functions. David *et al.* [106] investigate the SFC placement problem with respect to the traffic scaling incurred by SFs and SF location dependencies. A network function partitioning technique is used to derive near-optimal solutions.

However, there are some limitations of the aforementioned approaches. These approaches largely overlook the domain autonomy in multi-domain networks. Different network operators are not likely to share the control of their domains as doing so could reveal their business-sensitive competitive advantages.

3.2.3 Resource Management

Due to the multi-domain, geo-distributed, resource-distributed nature of networks, the interest in resource management for federated service chaining remains high. Resource management, in this thesis, refers to a system that allocates and scales network resources based on multiple objectives. Unlike only optimizing a single objective such as the deployment cost or the network delay, a resource management system aims to strike a nice balance between multiple network performances.

He *et al.* [107] study the maximum link load factor minimization problem without service chain graphs. An approximation algorithm is proposed based on the randomized rounding method and proves the efficacy of the proposed algorithm by extensive simulations. Unicorn [108] addresses the issue of discovering and representing the network resources of heterogeneous network domains. Unicorn abstracts the resource availability of every domain to a state and provides a query algorithm to efficiently collect the network information and hence preserves the domain confidential information.

Gupta *et al.* [109] study the SFC deployment problem to minimize the network resource consumption. An ILP problem is formulated with respect to multiple SF replicas. Then, a two-phase column-generation-based approach is proposed to solving the problem with reasonable execution time over large-scale networks. The proposed algorithm helps to analyze the number of SF replicas and the number of servers. X-MANO [110] provides a confidential federation interface for multi-domain service chaining. X-MANO can be applied in peer-to-peer and cascading configuration for multi-domain federation.

3.2.4 Resilience

Niu *et al.* [111] use a sideway cross backup model that jointly considers the resilience of VNFs and physical machines in data center networks. After that, a meteor shower optimization approach is proposed based on heuristics to improve the availability and execution time. Abdelaal *et al.* [112] first propose a problem that is termed as virtual network functions and their replica placement. The cited paper jointly considers load balancing and a number of resource constraints. The aim is multi-fold, including link utilization minimization, energy optimization and deployment cost.

3.3 Distributed Service Chain in Multi-domain Networks

There are also a few works that resort to distributed approaches, aiming to federate multiple domains in an efficient and scalable manner. In general, distributed approaches use multiple domain controllers connected by a secured communication network as shown in Figure 3.2.

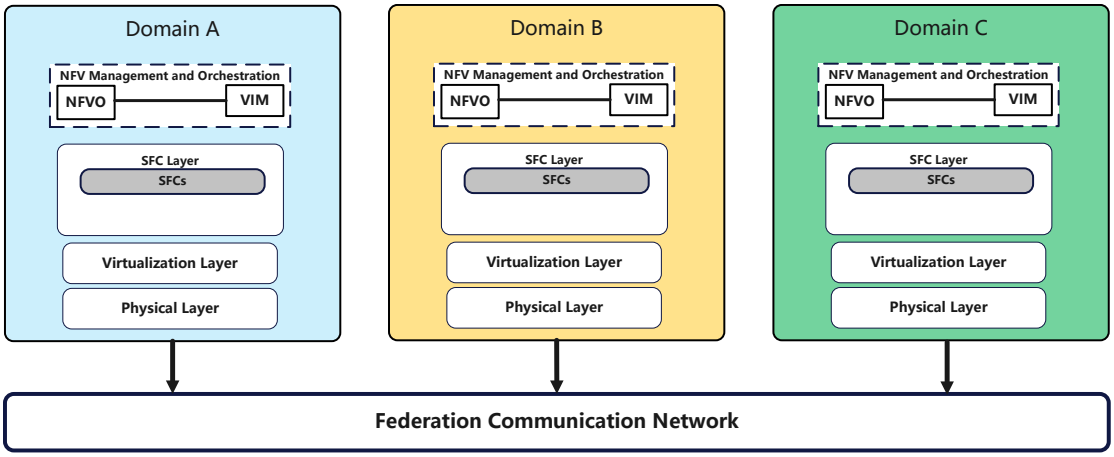


Figure 3.2: Distributed architecture for multi-domain networks

Figure 3.2 illustrates a network with 3 domains that is federated in a distributed manner. Every domain contains a controller that is responsible for NFV management and SFC orchestration. These domains communicate with others via a federated communication network.

3.3.1 Latency

Necklace [4] proposes a distributed framework that cooperates with virtual paths hosting service chains. Necklace uses a fully distributed asynchronous consensus scheme with guaranteed convergence time. By leveraging theoretical analysis, Necklace proves that Necklace achieves a $(1 - 1/e)$ approximation ratio compared with the Pareto optimal chain instantiation. Extensive experimental results verify that Necklace has superior performance. Catena [113] proposes a distributed paradigm for resilient service chains. Even in presence of non-byzantine failures, Catena still achieves guaranteed convergence time and performance with respect to the Pareto optimal approach.

3.3.2 Deployment Cost

Yu *et al.* [114] study an online SFC placement problem by balancing the trade-off between IT and spectrum resources. A two-phase orchestration algorithm is devised for time-efficient SFC placement.

Xu *et al.* [115] investigate the SFC placement problem with respect to a mobile service market of multiple network operators competing for network bandwidth and computation resources. An ILP problem is formulated to optimize the overall social cost of all network operations. Then, a distributed approach based on game theory is used to determine resource sharing among multiple participants. Extensive simulations demonstrate that the social cost can be remarkably reduced by collaborating with multiple network operators. The simulation results demonstrate that the proposed approach significantly reduces the deployment cost and shortens the deployment time.

CRSO [103] investigates the SFC deployment cost minimization problem while considering the reliability requirements. The deployment cost consists of the cross data-center bandwidth cost and service function cost. By leveraging Hidden Markov model, CRSO backups SFs to satisfy a variety of reliability requirements. ADDM [6] investigates the SFC deployment cost problem by considering the nodal resource cost and the link delay cost. ADDM splits the SFC cost minimization problem into multiple sub-problems. Also, ADDM aims to support scalability and confidentiality by using a fully distributed solution.

3.3.3 Resource Management

Toumi *et al.* [21] propose an end-to-end framework for multi-domain networks with respect to the internal communication protocols of each participating domain. Also, the authors conducted extensive simulation in light of a variety of key performance indicators with different encapsulation protocols. Ning *et al.* [116] formulate an optimization problem that maximizes the system utility. The server capacity constraint and the service execution latency are jointly considered. First, the long-term optimization problem is decomposed into a series of one-shot problems by using Lyapunov optimization method. Then, a sample average approximation-based stochastic algorithm is proposed to optimize the expected system utility. The proposed algorithm achieves a minor performance gap compared with the optimum.

Chen *et al.* [117] propose a distributed resource management scheme based on artificial intelligence for edge clouds. A network virtualization technique in IoT scenarios is proposed, and an optimization problem is formulated to optimize the total utility. Also, a dynamic algorithm, which is derived from resource-aware

matching algorithm and averaged multi-step double deep q-network algorithm, is proposed to embed SFC requests in a distributed manner.

Castaneda *et al.* [5] study the NFV dependent reconfiguration problem. In multi-domain networks, there are multiple domain orchestrators that manage the shared network resources. The consistency of the dependent reconfiguration is guaranteed in a distributed manner. Also, the collaboration among multiple parities is enabled as doing so could federate all participants. GDM [118] studies the SFC embedding problem in multi-domain networks. GDM aims to preserve the autonomy and confidential information of the participated domains. By allowing every domain to embed their assigned segments with respect to their policies, GDM successfully achieves domain autonomy. Huff *et al.* [119] propose a distributed scheme to manage SFC across multiple domains. By leveraging SFC segments, Huff enables different segments to communicate with each other.

Mohammad *et al.* [19] propose a dynamic algorithm based tabu search to place VNF managers for large-scale networks in a distributed manner. The algorithm aims to optimize the operational cost with respect to rigorous performance requirements. Extensive experimental results show that the proposed algorithm achieves near-optimal performance and significantly reduces the operational cost. Zhang *et al.* [120] propose a vertex-centric SFC orchestration scheme for multi-domain networks. A fully distributed approach is used to maintain physical information locally. By this means, the management complexity is remarkably reduced.

3.4 Service Chain Scaling

Service chain scaling is of significant importance to the scalability of the architecture. Edge computing has given birth to a variety of novel applications such as augmented reality, virtual reality, self-driving and etc. In general, these applications are sensitive to end-to-end delays. Since edge servers have limited capacity compared with cloud servers, the scarce edge resources need to be managed in a flexible and agile manner. By leveraging VNF instance scaling technology, service providers are able to handle the time-varying traffic demand.

One way to scale VNF is horizontal scaling (scale-out) which allows creating more or less VNF instances. Every instance processes a few traffic flows to adapt to the fluctuation in traffic demands.

Another way is vertical scaling (scale-up) which adds more or less computational resources to an existing VNF instance. By this means, the VNF instance can be resized by changing the CPU or memory capacity. Vertical scaling is usually limited by the resource capacity of the hardware.

3.4.1 Horizontal Scaling

Many research efforts have focused on horizontal scaling as it could avoid resource bottlenecks on a single server. However, horizontal scaling approaches need to schedule the traffic flows among multiple instances, resulting in sophisticated management.

He *et al.* [7] propose a VNF scaling detection scheme in the chain-wide granularity which enables SFC with arbitrary sizes. Then, reinforcement learning technologies are used to schedule the VNF placement based on chain-aware representations. Extensive experimental results demonstrate that the proposed algorithm significantly reduces the overall system cost and optimizes other network performance. Pham *et al.* [121] investigate the SFC placement problem for IoTs. This cited paper first assumes that IoT services can be deployed at edge clouds in the form of VNFs. Then, an operational cost minimization problem is formulated. The proposed algorithm horizontally scales the number of VNFs based on branch-and-bound. Also, the deep neural network model is used to remove the unlikely solutions in the space and hence improves the performance significantly.

Li *et al.* [122] propose an automatic scaling approach to improve the overall cost of the tenanted instances. As the network traffic workload varies over time, it is critical to accommodate the sheer volume of traffic without wasting network resources. Hence, a horizontal scaling approach is used to reduce the overall costs. Zhao *et al.* [123] investigate the VNF scaling problem, aiming to optimize the transmission delay and improve the resource utilization in VMs. Then, a scale-out based algorithm is proposed to efficiently use network resources.

Tong *et al.* [124] report a dynamic VNF scaling algorithm with a proactive traffic prediction method. The proactive prediction method is based on the deep learning network which enables decision-making ahead of time. The proposed algorithm uses multiple agents to explore the solution space and updates network parameters. Hence, this solution is time-efficient. FastScale [125], for the first time, proposes an idea of chain-wide scaling. In short, FastScale considers not only a single VNF scaling but also downstream VNFs. Since scaling an overloaded VNF consequently influences the downstream traffic rate, FastScale carefully considers the SFC scaling by updating a threshold repeatedly.

Luo *et al.* [126] investigate the dynamic scaling of VNF instances within cloud data center. An online scaling algorithm is proposed to scale out VNF instances, aiming to accommodate the time-varying traffic demand. Also, the optimality bound by theoretical analysis is also proved. Hieff [127] jointly considers the VNF scaling and flow mapping problem in a VNF cluster. By leveraging a flow table of heavy flows, Hieff manages light flows with a hash table. Extensive exper-

imental results justify that Hieff accommodates massive flows with low latency while balancing the workload among multiple VNF instances.

Yao *et al.* [128] investigate the horizontal scaling deployment of VNFs, aiming to minimize the operational cost. An offline VNF placement problem is formulated. Subramanya *et al.* [129] investigate the SFC placement and scaling problem for mobile edge computing. A neural-network model, which scales VNFs automatically by forecasting the required number of VNF instances, is proposed. The dynamic traffic patterns, the end-to-end latency and constraints on network resources are jointly considered.

Tang *et al.* [130] study the horizontal VM scaling in data center networks. The traffic patterns in an operator network is first investigated. Then, a traffic prediction method and two VNF placement algorithms are proposed to perform horizontal scaling. Patel *et al.* [131] propose a proactive approach based on deep learning which predicts the auto-scaling of VNFs ahead of time. The SFC placement problem is formulated as a cost minimization problem. Through theoretical analysis and trace-driven simulation, the proposed approach significantly reduces the overall cost with higher service availability.

Fei *et al.* [132] propose a proactive prediction approach for VNF scaling ahead of time. An SFC placement problem is formulated, aiming to minimize the cost incurred by inaccurate traffic prediction. The performance of the proposed algorithm is proven by both theoretical analysis and trace-driven simulation. Wang *et al.* [133] propose a pre-planned resource allocation scheme, aiming to solve the SFC placement and scaling problem. The bandwidth requirement is guaranteed which saves extensive network bandwidth. By leveraging a communication graph, tenants can reserve bandwidth resources between VNFs for service chains. Meanwhile, the proposed algorithm significantly reduces the VM migration overhead.

3.4.2 Vertical Scaling

Many research efforts use vertical scaling to handle the time-varying traffic demand. Vertical scaling has lower complexity compared with horizontal scaling. However, vertical scaling is usually limited by the resource capacity of the hardware. We summarize the existing literature on vertical scaling for the SFC deployment problem.

Qu *et al.* [8] propose a traffic parameter learning method. Change point detection and Gaussian process regression are used to study traffic parameters for every time slot. Then, a Markov decision process is proposed to optimize the migration cost and resource overloading penalty. DYVERSE [134] proposes multiple priority management techniques to achieve dynamic scaling. DYVERSE aims to advocate

workload-aware, system-aware and community-aware service scaling with service level objective constraints. By leveraging an online game and a face detection workload, DYVERSE verifies the performance of DYVERSE.

Zu *et al.* [135] formulate an ILP problem, aiming to deploy SFC requests with minimized latency cost. The vertical capacity scaling in routing commodities is considered. Also, this cited paper derives the optimal capacity for each VM based on the Karush-Kuhn Tucker conditions. Luo *et al.* [136] study the VNF scaling problem for geo-distributed SFC. A deep-learning-based approach is used for predicting the traffic pattern over time. Trace-driven experiments prove that the proposed algorithm timely achieves competitive system costs.

ElasticSFC [137] proposes an auto-scaling approach to minimize the operational cost while fulfilling the service level agreement. ElasticSFC jointly considers the SFC placement, migration and traffic engineering. The proposed algorithm outperforms other approaches in terms of average throughput, acceptance rate and mean instance utilization. Gouareb *et al.* [2] formulate a mixed integer linear programming problem to minimize the network end-to-end latency by leveraging vertical scaling.

Jia *et al.* [138] study the SFC placement problem by considering practical time-varying traffic demand across geographically distributed data centers. The aim is to minimize the operational cost over a variety of time slots. By leveraging a regularization-based approach, the offline optimal problem is converted to a number of one-shot regularized problems. Not only trace-driven simulations but also theoretical analysis are provided. Ghaznavi *et al.* [139] formulate an elastic virtual network function placement problem, aiming to minimize the cost of cloud providers. The aim is to strike a nice balance between the network bandwidth and the deployment cost. Experimental results show that the proposed algorithm accommodates two times more workload compared with a first-fit algorithm.

3.4.3 Hybrid Scaling

There are also a few schemes that use hybrid scaling to accommodate the time-varying traffic demand. Hybrid scaling refers to that the approaches use both vertical and horizontal scaling. We summarize the existing works as follows.

Subramanya *et al.* [140] propose centralized and federated approaches based on deep learning models, aiming to enable SFC scaling for multi-domain networks. A forecasting problem is formulated that predicts the required number of VNF instances ahead of time. Finally, extensive experiments are conducted in a testbed based on Kubernetes and prove the performance of the proposed algorithms. Zhai *et al.* [141] propose a hybrid scaling approach to handle the SFC demands that

frequently change. A key challenge is how to balance the scaling success ratio and the network resources. By leveraging vertical and horizontal scaling approaches, the resource consumption is effectively reduced and the scaling success ratio is improved.

Xie *et al.* [56] study the SFC scaling problem by considering SF parallelism. Since VNFs incur significant processing delay, the flexibility of NFV networks deteriorates. To cope with this, a parallelism-aware approach is proposed to efficiently improve the acceptance ratio and reduce the average delay. Rahman *et al.* [142] propose an auto-scaling scheme based on negotiation-game. It is assumed that tenants always comply with the decision of network operators, and quality of service requirements are homogeneous. After that, a forecasting method is proposed based on a proactive machine-learning approach. Extensive experimental results demonstrate that the proposed algorithm achieves a win-win situation for both tenants and network operators.

ENSC [10] proposes a hybrid scaling approach by jointly considering NFV efficiency and scalability. ENSC investigates the advantages of vertical and horizontal scaling in depth. Then, ENSC formulates an ILP problem and devises a heuristic algorithm called Rubik. Houidi *et al.* [143] formulate an ILP problem in light of the VNF placement and traffic routing problem. Then, a hybrid scaling approach is proposed to optimize the service interruption time incurred by scaling. The simulation results demonstrate that the proposed algorithm outperforms a greedy algorithm in most aspects.

3.5 Summary

In this chapter, we present state-of-the-art works in terms of the SFC deployment. In particular, we summarize the existing literature for single-domain SFC, centralized SFC for multi-domain, distributed SFC for multi-domain and service chain scaling, respectively.

In this context, a wide range of SFC orchestration schemes have been proposed to optimize various objectives such as the deployment cost, the end-to-end latency, energy consumption and etc. Centralized SFC orchestration suffers from scalability issues as the network scale increases significantly in multi-domain networks. Many existing approaches resort to distributed schemes that distribute the decision-making process between multiple controllers. Nevertheless, most of them largely overlook the intrinsic nature of service chain federation which calls for domain autonomy and confidential information. To improve the system scalability, we also need to improve the scalability of an individual VNF. This is because the network traffic is time-varying and dynamic in nature and hence VNFs need to

scale processing capacities to handle the traffic fluctuation. However, most existing approaches ignore the VNF category and hence fail to avoid the resource bottlenecks in VNFs.

In the next chapter, we present the system models.

Chapter 4

System Model

In this chapter, we present the system models including the physical network, service function chain, affinity constraints, queueing theory, VNF scaling and flow routing. These models are used to formulate the research problems and devise solutions.

4.1 Physical Network

We model the physical network as a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, ..., v_V\}$ denotes the set of all nodes and $\mathcal{E} = e_1, e_2, ..., e_E$ denotes the set of all links. In this thesis, every node is regarded as a cloud/edge data center that includes multiple NFV servers. Every cloud can host VNF instances on their Virtual Machines, containers or physical hosts. Every node v contains multiple fields $\{v.cpu, v.mem, v.domain, v.tier\}$. $v.cpu$ and $v.mem$ represent the maximum CPU and memory capacity of node v, respectively. Similarly, let $v.domain$ and $v.tier$ denote the domain and tier of node v, respectively. In this work, we use tier to represent the type of a cloud, e.g., public cloud, edge cloud or ISP cloud. If $e = (u, v) \in \mathcal{E}$, then the inter-cloud ink e traverses the edge clouds u, $v \in \mathcal{V}$. We use $C_v(t)$ to denote the remaining processing capacity at edge cloud v for time slot t. Let $C_{uv}^{bw}(t)$ denote the remaining bandwidth capacity of the inter-cloud link $(u, v) \in \mathcal{E}$ at time slot t. Every edge $e \in \mathcal{E}$ has a bandwidth capacity. Finally, we use $\mathcal{P}$ to represent the paths in the physical network.

4.2 Service Function Chain Model

Service function chain is a set of VNFs in a predefined order. Hence, we use $\mathcal{R}$ to represent the set of SFC requests. Every request is denoted by an 9-tuple $r = \{src, dst, \mathcal{N}, \Psi^{bw}, \Psi^{tr}, l^{td}, T_t, D_d, F\}$, in which src and dst represent the ingress

and egress node, respectively. $\mathcal{N} = \{n_1, n_2, ..., n_N\}$ represents the set of VNFs in a sequential order. Every VNF n consists of multiple fields, i.e., $\{type, cpu, mem\}$. $n.type$ denotes the VNF type such as flow monitor or proxy. $n.cpu$ and $n.mem$ represent the required amount of CPU and memory resources for the SFC, respectively. Ψ^{bw} is the required bandwidth resource, and Ψ^{tr} denotes the required traffic rate. Also, l_{td} denotes the maximum tolerated latency. T_t and D_d denote the set of domain and tier constraints, respectively. We use F to represent the set of traffic flows. Every SFC request consists of multiple flows $f \in F$. We use $n_i^1, n_i^2, ..., n_i^j$ to denote the instances of VNF n_i, which are the VNF replicas or instances.

4.3 Coarse-grained SFC Request Affinity

In this section, we introduce the coarse-grained SFC request affinity constraints. Since network operators could have demands to specify the constraints on domains and tiers, we use such constraints to fulfill these requirements. These constraints support SFC-level policies. By leveraging the idea of affinity and anti-affinity, we define the domain and tier constraints as follows.

$$T_t = \begin{cases} 1 & \text{indicates SFC request} \\ & \text{to be deployed at tier t.} \\ -1 & \text{prevents SFC request to be} \\ & \text{deployed at tier t.} \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

$$D_d = \begin{cases} 1 & \text{indicates SFC request to be} \\ & \text{deployed at domain d.} \\ -1 & \text{prevents SFC request to be} \\ & \text{deployed at domain d.} \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

4.4 Fine-grained SFC Request Affinity

In this section, we show the improved fine-grained SFC affinity constraints which support VNF-level constraints. Such fine-grained constraints enable flexible and agile SFC placement compared with the coarse-grained constraints.

$$T_t^{n_i} = \begin{cases} 1 & \text{indicates VNF } n_i \\ & \text{to be deployed at tier t.} \\ -1 & \text{prevents VNF } n_i \text{ to be} \\ & \text{deployed at tier t.} \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

$$D_d^{n_i} = \begin{cases} 1 & \text{indicates VNF } n_i \text{ to be} \\ & \text{deployed at domain d.} \\ -1 & \text{prevents VNF } n_i \text{ to be} \\ & \text{deployed at domain d.} \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

4.5 Queueing Theory

Queueing theory and queueing models provide a good insight for modeling network delays. We use the $M/M/1$ to model the queues at VNFs and network links because they can be regarded as a single server with a single queue. The $M/M/1$ queueing system includes a single queue and a single server. SFC requests arrive based on a Poisson process with rate λ.

The average delay $D(t)$ in the system contains two parts: the average service time $\frac{1}{\mu(t)}$ and the average waiting time in queue $W(t)$ [144].

$$D(t) = \frac{1}{\mu(t)} + W(t) \tag{4.5}$$

where $\mu(t)$ denotes the processing capacity of the system. By using Little's Theorem [144], the average waiting time is defined as follows.

$$W(t) = \frac{\rho(t)}{\mu(t) - \lambda(t)} \tag{4.6}$$

We use $\lambda(t)$ to represent the traffic arrival rate which is the total traffic rate of all flows in the system. $\rho(t)$ is equal to the ratio of arrival rate and processing rate.

$$\rho(t) = \frac{\lambda(t)}{\mu(t)} \tag{4.7}$$

Hence, the average delay is given as follows.

$$D(t) = \frac{1}{\mu(t) - \lambda(t)} \tag{4.8}$$

4.6 VNF Scaling and Flow Routing

There are usually two ways to scale VNFs. The first one is vertical scaling. Vertical scaling adds more or less physical resources, e.g., CPU and memory, to an existing container or virtual machine. The processing capacity of the scaled VNF instance increases. The second one is horizontal scaling. Horizontal scaling creates more or less virtual instances. Each instance accommodates a few traffic flows.

In vertical scaling, the new processing rate at $\mu_{n_i}^{new}(t)$ for VNF n_i is denoted as follows.

$$\mu_{n_i}^{new}(t) = \begin{cases} 0 & \text{if } \lambda_{n_i}(t) \leq \lambda_{n_i}(t-1) \\ \mu_{n_i}(t) - \mu_{n_i}(t-1) & \text{otherwise.} \end{cases} \tag{4.9}$$

where $\lambda_{n_i}(t)$ and $\lambda_{n_i}(t-1)$ represent the total arrival rate at time slot t and $t-1$, respectively. $\mu_{n_i}(t)$ and $\mu_{n_i}(t-1)$ are the total processing rate of VNF n_i at time slot t and $t-1$, respectively.

In horizontal scaling, the required number of new VNF instances for VNF n_i is computed as follows.

$$z_{n_i}^{new}(t) = \begin{cases} 0 & \text{if } \lambda_{n_i}(t) \leq \lambda_{n_i}(t-1) \\ \lceil \frac{\mu_{n_i}(t) - \mu_{n_i}(t-1)}{\mu_{n_i}^{max}} \rceil & \text{otherwise.} \end{cases} \tag{4.10}$$

where $\mu_{n_i}^{max}$ represents the maximum processing capacity for one instance.

4.7 Symbols and Variables

Table 4.1 presents the symbols for physical network. Symbols for service function chains are illustrated in Table 4.2. Table 4.3 shows the symbols of parameters. Finally, Table 4.4 presents the binary variables for the problem formulation.

Table 4.1: Physical Network

Physical Network	**Description**
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Physical network graph.
$\mathcal{R} = \{r_1, r_2, ..., r_R\}$	All SFC requests.
$\mathcal{V} = \{v_1, v_2, ..., v_V\}$	All nodes in the network.
$\mathcal{E} = \{e_1, e_2, ..., e_E\}$	All physical links in the network.
$\mathcal{P} = \{p_1, p_2, ..., p_P\}$	All paths in the network.
C_v^{cpu}, C_v^{mem}	Remaining CPU and memory capacity in node v.
C_p^{bw}	Remaining bandwidth capacity on path p.
$C_v(t)$	Remaining processing capacity in the edge cloud v.
$C_{uv}^{bw}(t)$	Remaining bandwidth capacity in the inter-cloud link uv.

Table 4.2: SFC

SFC	**Description**
src/S	Ingress node of the SFC request.
dst/T	Egress node of the SFC request.
$\mathcal{N} = \{n_1, n_2, ..., n_i\}$	All NFs in the SFC request.
n_i^j	jth instance of the VNF n_i.
$c_{n_i^j}(t)$	Amount of processing capacity required by the VNF instance n_i^j.
$v.cpu, v.mem$	Total capacity of CPU and memory on node v.
$c_{n_i}^{cpu}$, $c_{n_i}^{mem}$	Amount of CPU and memory required by NF n_i.
Ψ^{bw}	Required bandwidth.
Ψ^{tr}	Required traffic rate.
$b_{fuv}(t)$	Traffic rate of flow f in link uv.
l^{td}	Maximum tolerated delay.
l^d	End-to-end delay of a given SFC.
$D_d^{n_i}$	The domain constraint.
$T_t^{n_i}$	The tier constraint.

Table 4.3: Parameters

Parameters	**Description**
β_d^{cpu}	Resource cost of one unit of CPU in domain d.
β_d^{mem}	Resource cost of one unit of memory in domain d.
c_p	Traffic routing cost parameter of path p.
$\lambda_{n_i^j}(t)$	Arrival rate at VNF instance n_i^j.
$\mu_{n_i^j}(t)$	Processing rate at VNF instance n_i^j.
$\lambda_{n_i}(t)$	Total arrival rate at VNF n_i.
$\mu_{n_i}(t)$	Total processing rate at VNF n_i.

Table 4.4: Binary Variables

Binary Variables	**Description**
$y_v^{n_i}$	Indicator if NF n_i is hosted on node v.
$y_p^{n_i,n_j}$	Indicator if traffic from NF n_i to n_j is routed through path p.
$x_{fv}^{n_i^j}(t)$	Indicator variable equals 1 if VNF instance n_i^j of flow f is hosted on edge cloud v.
$y_{fuv}^{n_i^j n_{i+1}^j}(t)$	Indicator variable equals 1 if flow f traverses the virtual link $n_i^j n_{i+1}^j$ which is mapped to physical link uv.

4.8 Summary

In summary, this chapter presents the system models to formulate the research problems.

First, we present the physical network model for multi-domain networks and edge cloud networks. Then, the service function chain model is given by using a 9 tuple. After that, the coarse-grained and fine-grained affinity constraints are provided. These constraints include constraints for the domain and the tier. Moreover, we provide a queueing model for the latency analysis. Then, we provide models for VNF scaling and flow routing. Finally, all the symbols and variables are provided.

In the next chapter, we propose two novel SFC orchestration schemes for multi-domain networks in a distributed manner.

Chapter 5

Distributed Federated Service Chaining

In this chapter, we propose a distributed architecture for the SFC placement problem in multi-domain networks, aiming to minimize the deployment cost for service chains. The proposed Distributed Federated Service Chaining (DFSC) algorithm jointly considers the domain autonomy, scalability and domain confidential information to bridge the gap in the multi-domain networks. In particular, we propose a coarse-grained algorithm and a fine-grained algorithm to adapt to differently tailored policies for network operators. We first formulate an ILP problem for SFC placement. Then, we present the proposed algorithms. Finally, we show the experimental results and compare the performance with the optimum as well as other approaches from the existing literature.

5.1 Introduction

As illustrated in Chapter 1, federated service chaining calls for scalability due to the significantly increasing network scale [145]. For example, network services are shifting from centralized clouds to distributed edge clouds. If more and more edge service providers join the market, the network scale will significantly increase. In the meantime, the number of administrative domains will rise sharply and makes the networks fragmented. Hence, it is of significant importance to federate service chains across multiple domains.

However, federating such services calls for a high degree of service federation and resource sharing among multiple domains. Existing research efforts are unable to support these as they either only consider inter-cloud federation or resort to centralized resource management, both of which assume that all the network and resource information are known and that a centralized controller has visibility and

full control of all network domains.

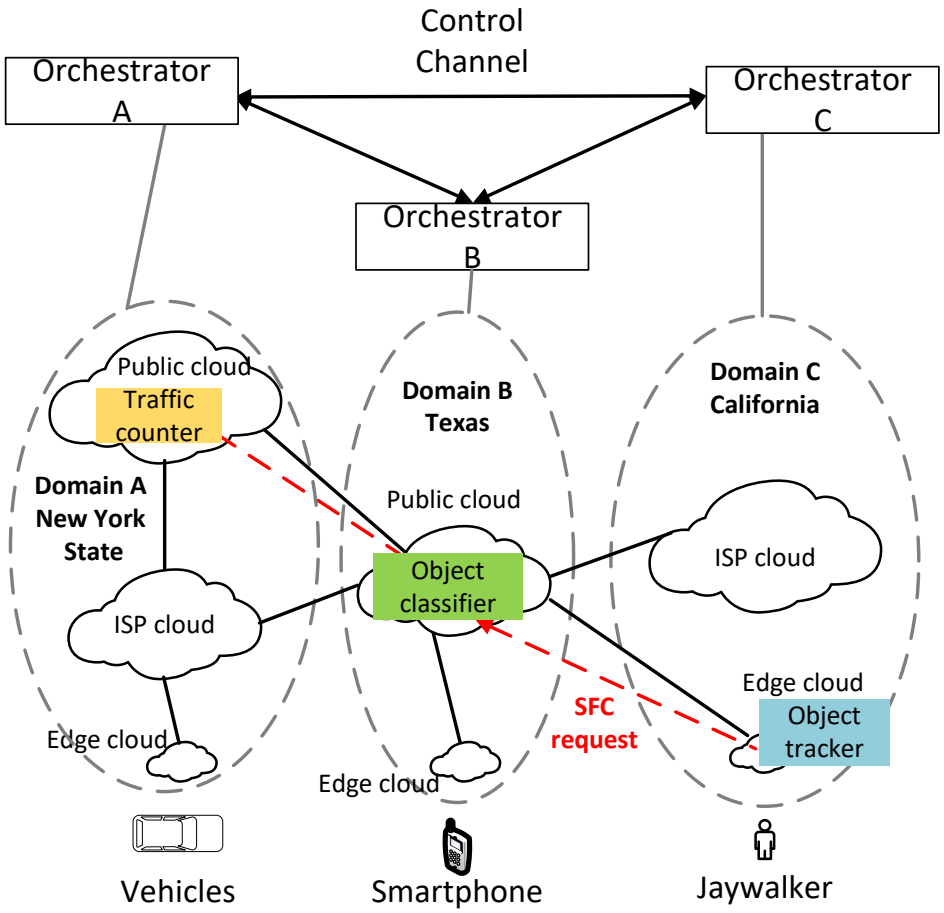


Figure 5.1: An example of SFC in multi-domain networks

Further, there are demands for service providers to specify domains and tiers of VNFs. The term "tier" refers to different types of clouds such as public clouds, ISP clouds and edge clouds. Figure 5.1 illustrates a service chain that detects jaywalkers. The service chain consists of an object tracker, an object classifier and a traffic counter. Certain VNFs are location and hardware dependent. For example, the object tracker must be embedded in domain California when collecting jaywalking data in California. Hence, there is a demand to specify the domain for VNFs. Similarly, the object classifier, which recognizes jaywalkers and cars, usually requires GPU for supporting a deep neural network (DNN). A public cloud can be equipped with certain hardware such as GPUs, which is why the object classifier needs to be deployed in a public cloud rather than an edge cloud. Therefore, there is also a demand to specify the tier.

To overcome the above challenges, in this chapter, we propose the Distributed

Federated Service Chaining scheme (DFSC) which preserves domain autonomy and domain confidential information of every participating domain. Each domain uses an independent controller which takes full advantage of the underlying infrastructure. Also, we reduce the amount of required network information by using the full-mesh aggregation. The scheme shows good scalability as it runs in a distributed manner with low complexity and reduces the execution time by 70%.

We formulate the SFC placement problem as an ILP problem, aiming to optimize the deployment cost in monetary form. We also devise domain and tier constraints to specify the location of SFC requests. Then, we propose two algorithms, i.e., coarse-grained and fine-grained DFSC to provide different granularity for constraints. Coarse-grained DFSC provides domain and tier constraints in the SFC granularity. In contrast, fine-grained DFSC provides domain and tier constraints in the VNF granularity.

We conducted extensive simulations in Network Simulator 3 (NS3) and Mininet. In a small-scale network, we compare DFSC with the offline optimum, showing the optimality gap. The experiment shows that DFSC achieves excellent performance within a factor of 1.15 to the offline optimum. In a large-scale network, we use a centralized approach (SFC Orchestration Across Multiple Domains) and a distributed approach (Distributed Network Service Embedding) to benchmark DFSC. DFSC achieves supreme performance by reducing up to 20% deployment cost and 70% decision-making time compared to the baseline approaches .

5.2 Problem Description

In this section, we formulate the SFC placement problem as an ILP problem that aims to minimize the overall deployment cost for multi-domain networks. This problem needs to determine the placement of VNFs in a sequential order and the traffic routing paths with respect to the resource capacities, the deployment cost and the end-to-end latency.

5.2.1 Problem Statement

The usage of computing and network resources comes with a monetary cost for service providers. When provisioning a service chain across multiple geographically distributed clouds, the consumption of computing resources incurs a resource cost in clouds. Similarly, the traffic routing cost is incurred by the usage of bandwidth resources in network links among clouds. In particular, the aim is to optimize the overall deployment cost for a given SFC request. We use C to denote the overall deployment cost per second. Let C_R represent the resource cost per second

and C_T denote the traffic routing cost per second. Then, the deployment cost is formulated as follows.

$$C = C_R + C_T. \tag{5.1}$$

5.2.2 The Resource Cost

The resource cost comprises two parts, i.e., the CPU and memory costs. Hence, the resource cost C_R is denoted as follows.

$$C_R = \sum_{n_i \in \mathcal{N}} \sum_{v \in \mathcal{V}} (\beta_d^{cpu} c_{n_i}^{cpu} y_v^{n_i} + \beta_d^{mem} c_{n_i}^{mem} y_v^{n_i}) \tag{5.2}$$

where β_d^{cpu} and β_d^{mem} denote the monetary costs of provisioning one unit of CPU and memory in the domain d, respectively. $c_{n_i}^{cpu}$ and $c_{n_i}^{mem}$ represent the required amount of CPU and memory resources for VNF n_i, respectively. Also, $y_v^{n_i}$ is an indicator variable that equals 1 if the VNF n_i is hosted on node v, and 0 otherwise.

5.2.3 The Traffic Routing Cost

When network traffic is routed along a service chain, traffic routing costs are incurred by the usage of network bandwidth. The costs are caused by the total bandwidth consumed on virtual links that are mapped to physical links. We use a unified cost parameter c_p to represent the cost of steering one unit of traffic along with path p because the routing cost is determined by the routing path. Therefore, the routing cost C_T for one SFC request is denoted as follows.

$$\begin{aligned} C_T = &\sum_{p \in \mathcal{P}} c_p \Psi^{tr} y_p^{src,n_1} + \sum_{p \in \mathcal{P}} \sum_{n_i, n_j \in \mathcal{N}} c_p \Psi^{tr} y_p^{n_i,n_j} \\ &+ \sum_{p \in \mathcal{P}} c_p \Psi^{tr} y_p^{n_N,dst}. \end{aligned} \tag{5.3}$$

where Ψ^{tr} represents the traffic rate of the request. $y_p^{n_i,n_j}$ denotes an indicator variable that equals 1 if the path p is used between VNFs n_i and n_j, and 0 otherwise. Specifically, y_p^{src,n_1} denotes whether the path p is mapped to the virtual link between the ingress node and the first VNF, and $y_p^{n_N,dst}$ indicates whether path p is mapped to the virtual link between the last VNF and the egress node.

5.2.4 SFC Orchestration Problem for Heterogeneous Multi-domain Networks

In this work, we consider the SFC placement as an online problem which means that SFC requests arrive one by one.

Problem 1 *Given the amount of computing resources available at each node, the amount of network resources at each link, the required resources by the SFC, the problem is to embed the VNFs and the subsequent traffic routing paths that minimize the deployment cost.*

$$\text{minimize} \quad C = C_R + C_T \tag{5.4}$$

This problem is subject to multiple constraints.

$$\sum_{n_i \in \mathcal{N}} c_{n_i}^{cpu} y_v^{n_i} \leq C_v^{cpu}, \quad \forall v \in \mathcal{V} \tag{5.5}$$

$$\sum_{n_i \in \mathcal{N}} c_{n_i}^{mem} y_v^{n_i} \leq C_v^{mem}, \quad \forall v \in \mathcal{V} \tag{5.6}$$

$$\sum_{p \in \mathcal{P}} \sum_{n_i, n_j \in \mathcal{N}} \Psi^{bw} y_p^{n_i, n_j} \leq C_p^{bw} \tag{5.7}$$

$$\sum_{v \in \mathcal{V}} y_v^{n_i} \leq 1 \tag{5.8}$$

$$l^d \leq l^{td} \tag{5.9}$$

Constraints 5.5 and 5.6 ensures that the required CPU and memory resources must not exceed the remaining CPU capacity C_v^{cpu} and memory capacity C_v^{mem} at the node v. Similarly, constraint 5.7 guarantees that the required bandwidth resources cannot exceed the remaining bandwidth capacity C_p^{bw} on the path p. Constraint 5.8 represents that every VNF for the request must only be assigned once. Also, constraint 5.9 ensures that the end-to-end delay l^d must not exceed the maximum tolerated delay l^{td}.

For coarse-grained DFSC, the domain and tier constraints are presented as follows.

$$\textit{if} \quad T_t = 1, \quad \sum_{v \in \mathcal{V}^t} y_v^{n_i} = 1 \tag{5.10}$$

$$\textit{if} \quad T_t = -1, \quad \sum_{v \in \mathcal{V}^t} y_v^{n_i} = 0 \tag{5.11}$$

$$if \quad D_d = 1, \quad \sum_{v \in \mathcal{V}^d} y_v^{n_i} = 1 \tag{5.12}$$

$$if \quad D_d = -1, \quad \sum_{v \in \mathcal{V}^d} y_v^{n_i} = 0 \tag{5.13}$$

Constraints 5.10 and 5.11 are tier constraints that guarantee all VNFs in the SFC request can or cannot be placed at tier t. Constraints 5.12 and 5.13 are domain constraints that guarantee all VNFs in the SFC request can or cannot be placed at domain d.

For fine-grained DFSC, the domain and tier constraints are presented as follows.

$$if \quad T_t^{n_i} = 1, \quad \sum_{v \in \mathcal{V}^t} y_v^{n_i} = 1 \tag{5.14}$$

$$if \quad T_t^{n_i} = -1, \quad \sum_{v \in \mathcal{V}^t} y_v^{n_i} = 0 \tag{5.15}$$

$$if \quad D_d^{n_i} = 1, \quad \sum_{v \in \mathcal{V}^d} y_v^{n_i} = 1 \tag{5.16}$$

$$if \quad D_d^{n_i} = -1, \quad \sum_{v \in \mathcal{V}^d} y_v^{n_i} = 0 \tag{5.17}$$

Constraints 5.14 and 5.15 are tier constraints that guarantee a VNF can or cannot be placed at tier t. Constraints 5.16 and 5.17 are domain constraints that guarantee a VNF can or cannot be placed at domain d.

Problem 1 can be proven to be NP-hard. For this we demonstrate that the Multiple Knapsack Problem (MKP), which is known to be NP-hard [146], can be reduced to this problem. The input of the MKP problem is a set $\mathcal{M}$ of items, where the weight of item $m_i \in \mathcal{M}$ are $m_i.weight$. A set $\mathcal{K}$ of knapsacks, where the capacity of $k_j \in \mathcal{K}$ are $k_j.cap$. The profit of mapping m_i to k_j is f_{ij}. The objective of the MKP problem is to maximize the overall profit.

For the reduction, we map each item m_i to an NF n_i with $m_i.weight = c_{n_i}^{cpu}$. Each knapsack k_j is considered as a node v_j with $k_j.cap = v_j.cpu$. Then, we assume the memory resources are infinite on the node v_j. Consider the special case that each SFC consists of only one NF. The profit f_{ij} is the negative of the deployment cost of assigning n_i to v_j. Assume that there are sufficient nodes to host all NFs, meaning that no nodes are overloaded. Thus, the MKP problem becomes a special case of problem 1. The objective is to minimize the overall deployment cost, or maximize the total profit. Therefore, the MKP problem can be reduced to the SFC

placement problem and, hence, the SFC placement problem is NP-hard. Similarly, we could also assume $k_j.cap = v_j.mem$ and the CPU resources are infinite on the node v_j. In this case, the MKP problem is still reducible to problem 1.

5.3 Distributed Federated Service Chaining Algorithm

In this section, we demonstrate the distributed federated service chaining algorithm. The main difference between coarse-grained and fine-grained DFSC is that the former checks domain and tier constraints for every SFC, and the latter checks these constraints for every VNF in the chain.

The proposed algorithm is devised to find SFC placement for problem 1. DFSC distributes the decision-making progress to several domain orchestrators, in which the domain autonomy and domain confidential information are well preserved. DFSC merely requires a few network information such as inter-domain links and border nodes. Figure 5.2 illustrates the workflow of the proposed DFSC algorithm.

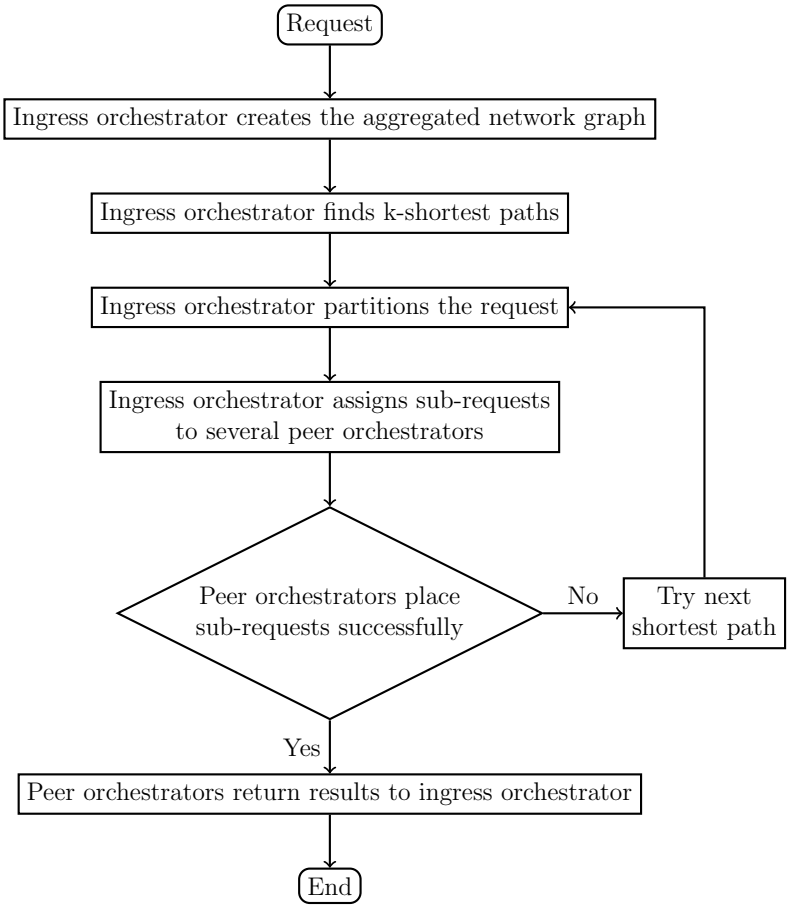


Figure 5.2: The workflow of DFSC

In this work, we assume that every domain is associated with a domain or-

chestrator that manages all the intra-domain information but has only limited information about other domains. If an SFC request arrives at a domain, the request is partitioned by the ingress orchestrator into multiple sub-requests. The ingress orchestrator refers to the orchestrator in a domain that the SFC request arrives. After that, these sub-requests are sent to multiple peer orchestrators. The peer orchestrators process these sub-requests to find a placement, respectively. Finally, the placement results are sent back to the ingress orchestrator.

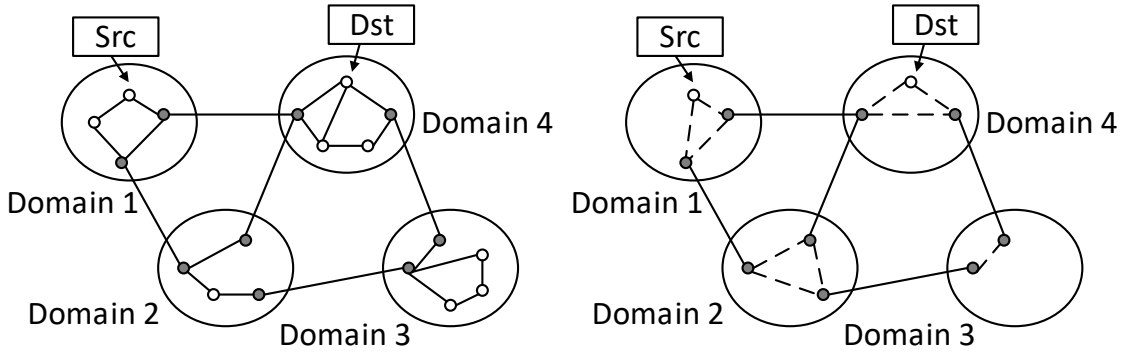
First, the ingress orchestrator creates an aggregated graph by leveraging the full-mesh aggregation [147]. The aggregated graph includes only border nodes and inter-domain links to preserve domain confidential information. Then, the ingress orchestrator leverages a k-shortest path algorithm [148] between source and target pairs in the aggregated graph. The k-shortest path algorithm can provide multiple candidate paths, resulting in exploring the path diversity at the cost of a slight degradation in decision-making time compared to the Dijkstra′s shortest path algorithm. After that, the ingress orchestrator determines how to partition the SFC request into sub-requests and assign sub-requests to multiple peer orchestrators. Finally, every peer orchestrator strives to find a solution within their own domain and sends placement results back to the ingress orchestrator. The ingress orchestrator then assembles the placement of sub-requests and creates the overall placement for the SFC. The details are given in the following sections.

5.3.1 Constructing the Aggregated Graph

Domain confidential information is always a crucial concern for multi-domain networks [100]. To this end, we merely require limited information by leveraging an aggregated graph. The aggregate graph only requires the information of border nodes and inter-domain links. This can be achieved by the Border Gateway Protocol [149]. As shown in Figure 5.3(a), grey nodes represent border nodes and the link that connects two border nodes is an inter-domain link. Consequently, the intra-domain connectivity and topology are preserved as confidential information. Thus, domain internal information is preserved.

As aforementioned, the aggregated graph is only comprised of inter-domain links, border nodes and logical intra-domain links. We derive the aggregated graph from the physical network based on the full-mesh aggregation. The full-mesh aggregation approach is to only keep border nodes and remove other nodes.

Then, the intra-domain links are replaced by a few logical links. Figure 5.3(b) shows that grey nodes are border nodes. The white nodes stand for the ingress and egress nodes of the SFC request. The dotted and solid lines represent the inter-domain and logical links, respectively. The latency l^d and traffic routing cost



(a) The physical network

(b) The aggregated graph

Figure 5.3: Topology aggregation

parameter c_p for every logical link are set to 0 in this phase because we aim to preserve the domain confidential information. There is enough information for the DFSC algorithm to partition the SFC requests into sub-requests in this phase.

5.3.2 Distributed Federated Service Chaining Placement

In this subsection, the distributed framework is presented. Algorithm 1 demonstrates the pseudo-code of DFSC algorithm. First, the ingress orchestrator creates the aggregated network graph when the SFC request arrives. Then, the ingress orchestrator creates several candidate paths based on the k-shortest path algorithm with respect to the traffic routing cost. Then, the SFC request is partitioned into multiple sub-requests based on the resource cost parameters β_d^{cpu} and β_d^{mem}. Finally, sub-requests are assigned to peer orchestrators. Peer orchestrators run Algorithm 2 which searches for solutions for the intra-domain placement. If any of the intra-domain placement fails, DFSC will place the request on the next candidate path until all candidate paths fail.

The DFSC algorithm is initialized in line 1 in algorithm 1. After that, lines 3-17 and lines 24-26 run at the ingress orchestrator. Meanwhile, lines 18-23 are executed on peer orchestrators. The aggregated graph is constructed in line 3. In line 4, the k-shortest path algorithm is employed to compute candidate paths. In lines 6-8, DFSC checks the bandwidth requirement. In line 9, the related domains are sorted based on the resource cost parameters. For simplicity, we regard the sum of β_d^{cpu} and β_d^{mem} as the sorting criteria. After that, the ingress orchestrator checks the available resources of related domains. Next, the ingress orchestrator selects a feasible domain with the lowest resource cost to host the VNF. Note that the main difference is that coarse-grained DFSC checks the SFC-level domain constraints. In contrast, the fine-grained DFSC checks the VNF-level domain constraint. In

Algorithm 1: Distributed Federated Service Chaining Placement

```
1  Input: SFC request r, resource cost parameter β_d^cpu and β_d^mem of each
     domain, traffic routing cost parameter c_p;
2  Output: Routing path, hosted nodes, deployment cost;
3  Construct aggregated graph G^a = (V^a, E^a);
4  Find k-shortest path P^k = p_1, p_2, ..., p_k according to traffic routing cost;
5  while p ∈ P^k ≠ ∅ do
6      if Ψ^bw > C_p^bw then
7          continue;
8      end
9      Sort the set of domains on the path p based on β_d^cpu and β_d^mem;
10     for n_i ∈ N do
11         Find the available domain d with the lowest resource cost;
12         if d meet the constraint D_d then
13             Assign the NF to d;
14         else
15             Find next d;
16         end
17     end
18     for each domain d on path p do
19         Call Algorithm 2 on peer orchestrators;
20     end
21     if Algorithm 2 fails then
22         Continue;
23     end
24     if l^d ≤ l^td then
25         break;
26     end
27 end
```

line 19, algorithm 2 is invoked.

Algorithm 2 demonstrates the intra-domain placement algorithm that is simultaneously executed on multiple peer orchestrators. First, the peer orchestrator employs the k-shortest path algorithm to find candidate paths within its own domain. After that, the peer orchestrator places the VNF to the first available node. In the meantime, the tier and resource constraints are examined. It is worth mentioning that coarse-grained DFSC checks the SFC-level tier constraints, while fine-grained DFSC checks the VNF-level constraints for every VNF. Then, if all VNFs in the sub-request are placed, the peer orchestrator sends the placement results back to the ingress orchestrator.

Algorithm 2: Intra-domain Deployment

Input: Assigned NFs $\mathcal{N}^d$ for domain d, single domain graph $\mathcal{G}^d = (\mathcal{V}^d, \mathcal{E}^d)$;
Output: Routing path p^d in $\mathcal{G}^d$, a set of nodes $\mathcal{V}^h$ that host NFs;
Find k-shortest paths $\mathcal{P}^k = p_1, p_2, ..., p_k$ according to traffic routing cost in $\mathcal{G}^d$;
while $p \in \mathcal{P}^k \neq \emptyset$ **do**
    if $\Psi^{bw} > C_p^{bw}$ **then**
        continue;
    end
    for *Each* n_i *in* $\mathcal{N}^d$ **do**
        Find the node v on p with enough computing resources;
        if t *meet the constraint* T_t **then**
            Assign the NF to v;
        else
            Find next v;
        end
    end
    if *all NFs are assigned* **then**
        return p^d and $\mathcal{V}^h$;
    end
end
return Intra-domain deployment failure;

5.3.3 Compared Algorithms

The SFCO-AMD and DistNSE algorithms have been used to benchmark the performance of DFSC. We describe these compared algorithms as follows.

- SFC Orchestration Across Multiple Domains (SFCO-AMD) [3]

  We select this algorithm because SFCO-AMD also considers the SFC placement for multi-domain networks. It is a good comparison as it uses centralized architecture for the management of service chains. SFCO-AMD partitions the SFC requests based on an aggregated graph in a centralized manner. This approach also uses all the possible abstracted paths that connect the ingress and egress nodes. For an SFC request, SFCO-AMD first partitions the request into several sub-requests on average. Then, VNFs are placed in a first-fit manner.

- Distributed Network Service Embedding (DistNSE) [149]

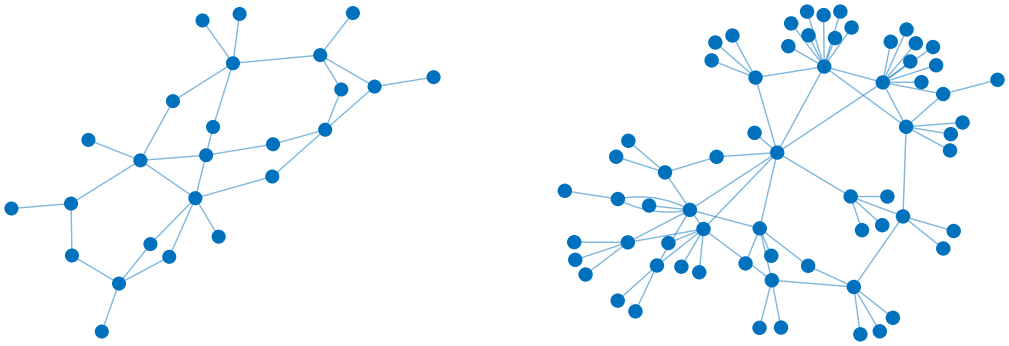
  DistNSE uses a distributed architecture which is a good comparison to the proposed architecture. To demonstrate the performance of DFSC, we compare the proposed DFSC approach to DistNSE in light of a wide range of metrics such as deployment cost, decision-making time and etc. DistNSE

deploys SFC requests in a distributed manner. DistNSE uses a bidding mechanism that allows domains to bid for VNFs. First, DistNSE finds all the paths between ingress and egress nodes. Then, the SFC request is partitioned into a set of sub-requests for domains to compete. Every sub-request is assigned to the domain with the lowest offer. To ensure the correct order of SFC, this algorithm only allows bidding for the last selected sub-request.

5.4 Evaluation for Coarse-grained DFSC

In this section, we show the experimental results for coarse-grained DFSC. We conducted extensive trace-driven simulations to justify the performance of the proposed DFSC algorithm.

5.4.1 Evaluation Environment



(a) Topology Agis (b) Topology Internode

Figure 5.4: Topology in the simulation

We implemented DSFC and SFCO-AMD approaches in Network Simulator 3 (NS3) and conducted the experiments on a server with 105 GB RAM and an Intel(R) Xeon(R) E5645 processor with 24 cores. We use two topologies Agis and Internode from the Internet Topology Zoo [150] as shown in Figure 5.4. The topology Agis consists of 25 nodes, 30 links and 5 domains. The topology Internode consists of 66 nodes, 75 links and 9 domains.

Every SFC request consists of a varying number of VNFs chosen from 5 off-the-shelf VNFs, i.e., firewall, NAT, proxy, load balancer and IDS. We set the number of CPU cores for one public cloud, ISP cloud and edge cloud to 1000, 500 and 200, respectively. Similarly, the memory capacities are set to 1000 GB, 500 GB and 200 GB, respectively. The link bandwidth for inter-domain and intra-domain are

1000 Mbps and 500 Mbps, respectively. The propagation delay is set between 2 ms and 10 ms for every link. Also, the ingress node, the egress node and VNFs are randomly created. For every VNF, the required number of CPU and the required amount of memory are set to (1,10) and (1,20) GB, respectively. The number of VNFs in the SFC request is 3. The required traffic rate is set between 100 kbps and 500 kbps. Also, domain and tier constraints are randomly generated. The value of k for the k-shortest path is empirically set to 8 because the performance is not remarkably improved when k is greater than 8.

The cost settings are derived from the Amazon instance prices [151]. The CPU resource cost per second ranges from \$0.001 to \$0.005. The memory resource cost per second ranges from \$0.001 to \$0.004. The traffic routing cost for one Mbit over a link is between \$0.02 to \$0.05. Finally, the maximum tolerated end-to-end latency ranges from 50 ms to 100 ms.

To justify the effectiveness of DFSC, we first compare DFSC with the offline optimum and SFCO-AMD in a small-scale network. Then, we run DFSC and SFCO-AMD in a large-scale network, wherein achieving the optimum solution from an ILP solver in a reasonable time is impractical.

To prove the performance of the proposed algorithm, the metrics include the deployment cost, the decision-making time, the acceptance rate of request, the cumulative distribution function of deployment cost and end-to-end latency.

5.4.2 Evaluation Results for Agis Topology

To study the optimality gap, we first compare the performance of DFSC and SFCO-AMD with the optimum achieved by the Gurobi solver in a small-scale network.

We first show the cost ratio in Figure 5.5. The cost ratio is the ratio of the average deployment cost attained by DFSC and SFCO-AMD to the offline optimum. The cost ratio of DFSC stays within 1.15 times to the optimum. In contrast, the worst case for SFCO-AMD is approximately 1.3 times to the optimum. This result suggests that DFSC can effectively approach the optimum and has stable performance. The rationale is that DFSC jointly considers the resource and traffic routing cost which leads to a lower overall cost.

Then, we demonstrate the total decision-making time in Table 5.1. The decision-making time is the execution time of the algorithms in NS3 which suggests the time efficiency of the algorithms. The decision-making time of DFSC ranges from 0.58 seconds to 9.6 seconds. In contrast, the decision-making time of SFCO-AMD ranges from 45.2 seconds to 369.7 seconds. The Gurobi solver spends 201 to 5673.9 seconds to find the offline optimum. Overall, DFSC is several orders

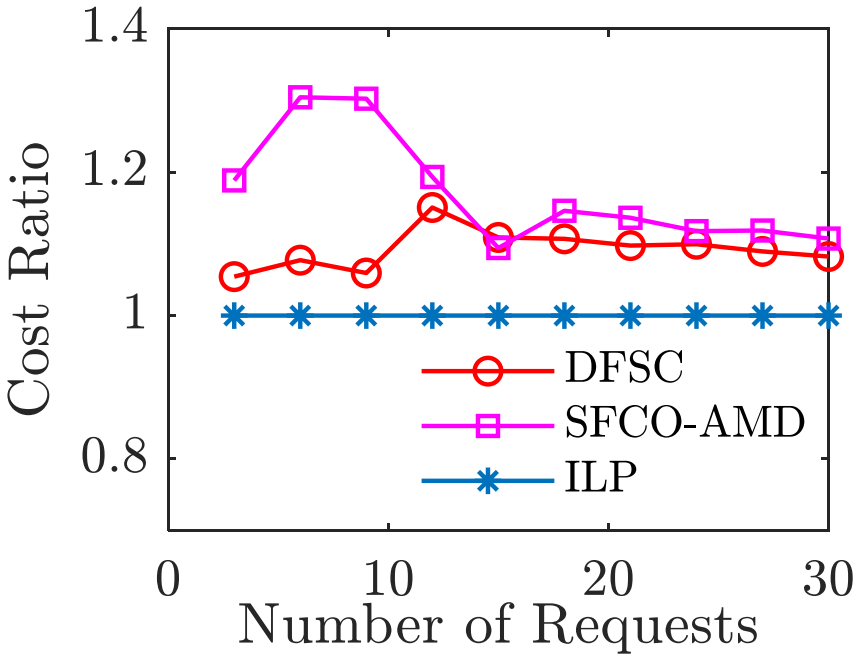


Figure 5.5: Number of requests vs. cost ratio in topology Agis

of magnitude faster than the ILP solver. Similarly, DFSC is also at least one order of magnitude faster than SFCO-AMD. This is because DFSC runs in a distributed manner with low complexity which significantly reduces the run time of the algorithm. The results suggest that DFSC is time-efficient, scalable and capable of processing substantial SFC demands.

5.4.3 Evaluation Results for Internode Topology

For the Internode topology, we compare the proposed DFSC approach with SFCO-AMD approach. The SFCO-AMD approach is regarded as the baseline. We do not implement the ILP solver for this topology because it cannot find the optimum

Table 5.1: Decision-making time

Number of requests	**DFSC**	**SFCO-AMD**	**Gurobi**
3	0.58s	45.2s	201s
6	1.48s	62.6s	505s
9	2.76s	119.2s	903.2s
12	3.8s	137.08s	1441.3s
15	5.08s	176.36s	1978.8s
18	5.55s	211.55s	2958.6s
21	6.43s	250.6s	3783.2s
24	7.62s	271.92s	4857.8s
27	8.42s	291.46s	4192s
30	9.6s	369.7s	5673.9s

in a reasonable time.

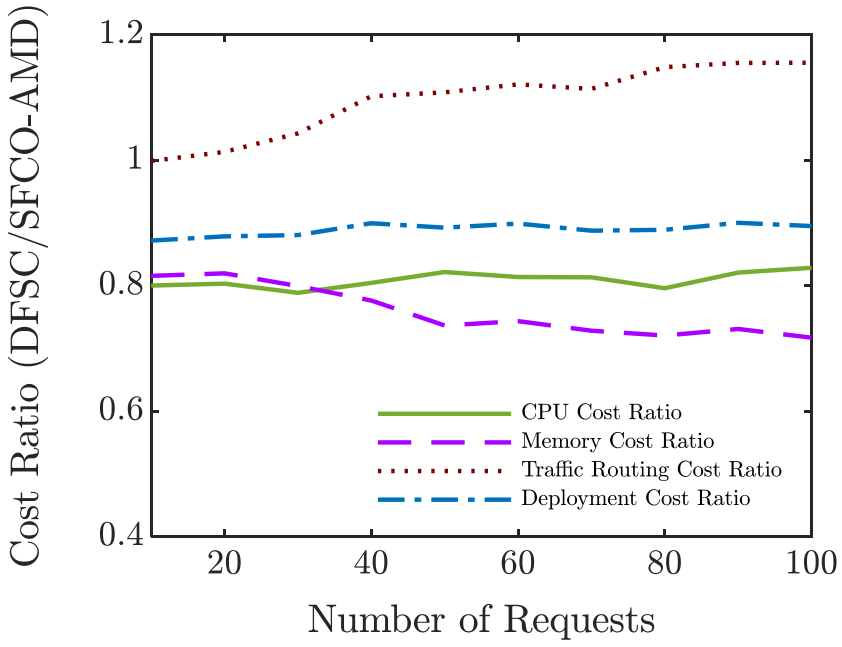
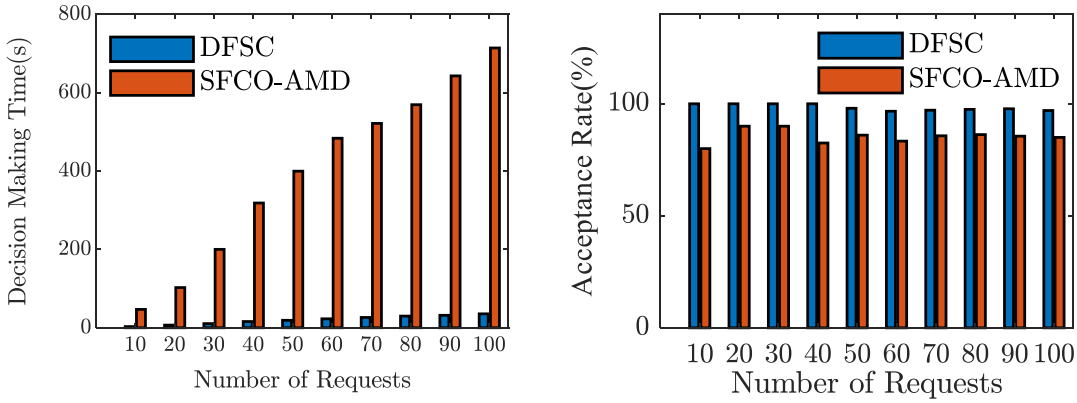


Figure 5.6: The DFSC/SFCO-AMD cost ratio in topology Internode

Figure 5.6 shows the ratio of CPU cost, memory cost, traffic routing cost and deployment cost. We observe that DFSC reduces the overall deployment cost by 12% compared with SFCO-AMD. Similarly, DFSC outperforms SFCO-AMD in terms of CPU and memory cost by up to 22% and 26%, respectively. However, the SFCO-AMD approach achieves 11% less traffic routing cost. The rationale is that DFSC makes a trade-off between traffic routing and resource cost to optimize the overall deployment cost. Hence, DFSC outperforms SFCO-AMD in terms of the overall deployment cost in every case. This result suggests that DFSC is cost-efficient for large-scale networks.

After that, we investigate the total decision-making time of DFSC and SFCO-AMD for the Internode topology. We run both algorithms against a varying number of SFC requests. From Figure 5.7(a) we observe that the decision-making time of DFSC increases slowly while that of SFCO-AMD rises dramatically. The total decision-making of DFSC ranges from 2.98 to 35 seconds, while that of SFCO-AMD ranges from 46 to 714 seconds.

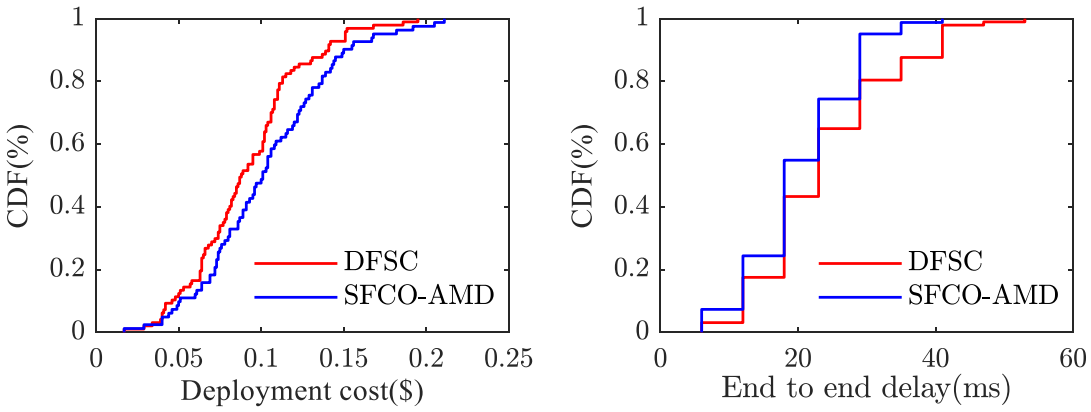
The DFSC algorithm outperforms SFCO-AMD in each case by one order of magnitude. This is because DFSC uses a distributed scheme, in which the decision process is distributed to several orchestrators in parallel. The workload for each orchestrator is significantly reduced. Furthermore, the DFSC approach embeds the k-shortest path algorithm rather than searching all candidate paths in SFCO-AMD approach. By this means, the decision-making time is also shortened.



(a) Decision-making time (b) Acceptance rate

Figure 5.7: The comparison of decision-making time and acceptance rate

Then, we continue to investigate the acceptance rate against a varying number of SFC requests. The acceptance rate is the ratio of successfully deployed SFC requests to the total number of SFC requests. As depicted in Figure 5.7(b), the acceptance rate drops slightly for both algorithms as the number of requests increases. This is due to the emergence of resource bottlenecks. Overall, the DFSC algorithm can still improve the acceptance rate by 12% on average.



(a) CDF of deployment cost (b) CDF of end-to-end delay

Figure 5.8: CDF of deployment cost and delay

Figure 5.8(a) demonstrates the CDF of the deployment cost. DFSC deploys 80% of the requests within \$0.11. In contrast, SFCO-AMD only deploys 60% requests. This is because DFSC strikes a nice balance between resource and traffic routing costs.

Also, we show the CDF of the end-to-end latency in Figure 5.8(b). The end-to-end latency is measured by sending packets along the chain. The SFCO-AMD algorithm achieves 99 percentile SFC requests within 35 ms. However, only 87%

SFC requests of DFSC approach experience less than 35 ms latency. The rationale is that, because DFSC performs better in terms of acceptance rate, more traffic is routed in the network which can incur greater latency. Also, DFSC can select a longer routing path to optimize the overall cost.

5.4.4 Acceptance Rate over Different k Values

Table 5.2 shows the acceptance rate against different k values for the k-shortest path.

Table 5.2: DFSC performance under different k values

Value of k	**Acceptance rate**	**Decision-making time**
1	73%	17.0s
2	74%	17.6s
3	75%	18.3s
4	75%	19.1s
5	75%	20.4s
6	78%	21.1s
7	78%	21.9s
8	79%	21.5s

Since the k value refers to the number of candidate paths, we need to find a suitable value for k. Also, we reduce the number of CPU cores to 100, 50 and 20, respectively. Similarly, we reduce the memory capacities to 100 GB, 50 GB and 20 GB, respectively. The bandwidth capacities for inter and intra-domain links are reduced to 100 Mbps.

We observe that the acceptance rate rises from approximately 73% to 79% when k value increases from 1 to 8. The rationale is that greater k values imply more candidate paths in DFSC approach which can be used to accommodate more SFC requests. Hence, DFSC is more likely to accept more requests. Also, the decision-making time rises inevitably from 17 to 21.5 seconds due to the increase of the searching space.

5.5 Evaluation for Fine-grained DFSC

In this section, we show experimental results for fine-grained DFSC.

5.5.1 Evaluation Environment

To provide in-depth insights into the performance, we embed the fine-grained DFSC not only in NS3 but also in an emulator, i.e., Mininet. Mininet utilizes real

CPU, memory and hardware clock. The experiments are conducted on a server with 105 GB RAM and an Intel(R) Xeon(R) E5645 processor with 24 cores. The topologies are still derived from the Internet Topology Zoo. The cost settings and system parameters are still identical to that in the coarse-grained DFSC section.

5.5.2 System Parameter

Table 5.3: Parameter Settings

Description	**Values**
Number of domains for Agis	5
Number of domains for Internode	9
CPU Capacity in cloud	1000
CPU Capacity in ISP cloud	500
CPU Capacity in edge cloud	200
Memory Capacity in cloud	2000GB
Memory Capacity in ISP cloud	1000GB
Memory Capacity in edge cloud	400GB
Bandwidth Capacity in inter-domain link	1000Mbps
Bandwidth Capacity in intra-domain link	2000Mbps
Link delay	[2,5] ms
Description	**Values**
CPU cost parameter	[0.001, 0.005] $/s
Memory cost parameter	[0.001, 0.004] $/GBs
CPU demand of NF n	[1,10]
Memory demand of NF n	[3, 30] GB
Traffic rate requirement	[100, 500] kbps
Traffic routing cost parameter	[0.02, 0.05] $/Mb
Maximum tolerated delay	[50, 100] ms

All the parameter settings are listed in Table 5.3. The source and target nodes of an SFC request are uniformly selected at random from the set of nodes in the network. The chain length, that is the number of NFs in a request, varies from 3 to 6. The number of cores in one cloud, ISP cloud and edge cloud is 1000, 500, and 200, respectively. The memory capacity of one cloud, ISP cloud and edge cloud is 2000 GB, 1000 GB and 400 GB, respectively. The bandwidth capacities for inter-domain and intra-domain links are set to 1000 Mbps and 2000 Mbps, respectively. The propagation delay is randomly selected between 2 ms and 10 ms for every link. For each NF, the number of CPU cores is randomly selected from {1, 2, ..., 10}. The memory requirement in GB is randomly selected from {1, 2, ..., 20}. Then, we set up some simulation parameters according to some similar papers [42], [152]. Also, the domain and tier constraints are randomly created. The value k for k-shortest path is set to 8 because of empirical studies. The reason behind this setup is that acceptance does not significantly increase when the value

of k is greater than 8. Each SFC request contains of a varying number of network functions.

To prove the performance of the proposed algorithm, the metrics include the deployment cost, the decision-making time, the acceptance rate of request, the cumulative distribution function of deployment cost.

5.5.3 Evaluation Results for Agis Topology

In this subsection, we compare the fine-grained DFSC to the offline optimum, SFCO-AMD and DistNSE for Agis topology. The optimum is still derived from Gurobi solver. Specifically, given an SFC request, Gurobi finds the placement for VNFs and traffic routing paths that optimize the deployment cost. In this experiment, we test DFSC against a varying number of SFC requests with a step size 3.

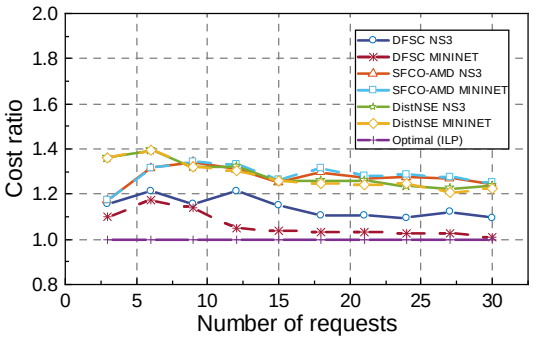


Figure 5.9: Number of requests vs. cost ratio in topology Agis

First, we show the cost ratio of DFSC, SFCO-AMD and DistNSE against the offline optimum which is the ratio of the deployment cost of both algorithms to the optimum. As shown in Figure 5.9, we observe that DFSC achieves 115% deployment cost to the optimum, justifying the effectiveness of our proposed algorithm. Moreover, DFSC outperforms SFCO-AMD and DistNSE in every case. The cost ratio of SFCO-AMD ranges from 1.2 to 1.3 compared with the optimum. Similarly, DistNSE achieves 1.3 to 1.4 in terms of the cost ratio. When the number of SFC request increases, the cost ratio of DFSC only increases slightly, indicating DFSC has stable performance. The results of DFSC in NS3 and Mininet have similar trends, suggesting that DFSC has stable performance. We prove that DFSC achieves near-optimal solutions in a small-scale network.

Then, we continue to analyze the time efficiency of DFSC. The decision-making time refers to the CPU time spent by the algorithm. In Figure 5.10, we observe that: (1) DFSC reduces the decision-making time by at least 70% compared with SFCO-AMD and DistNSE, and DFSC is two orders of magnitude faster than the

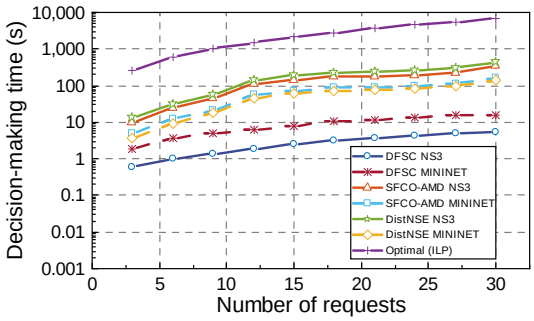


Figure 5.10: Number of requests vs. decision-making time in topology Agis

ILP solver. As the number of SFC requests rises, the decision-making time of DFSC varies slightly while that of SFCO-AMD, DistNSE and ILP solver increases dramatically. The rationale is that DFSC runs in a distributed manner which significantly reduces the workload for every orchestrator. Also, we use the k-shortest path algorithm instead of finding all the candidate paths between ingress and egress pairs. This result suggests that DFSC significantly reduces the decision-making time by 70%.

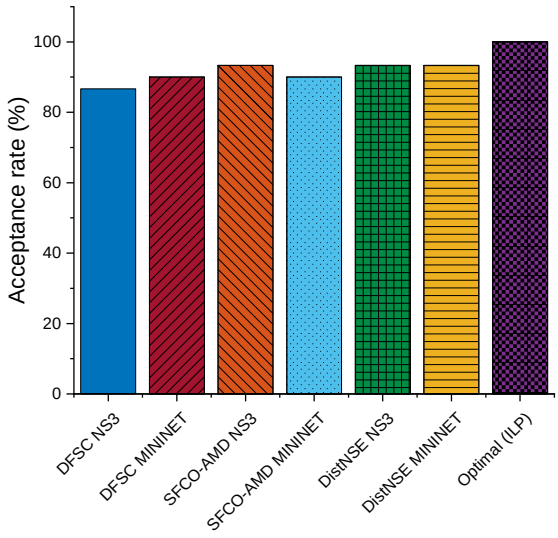
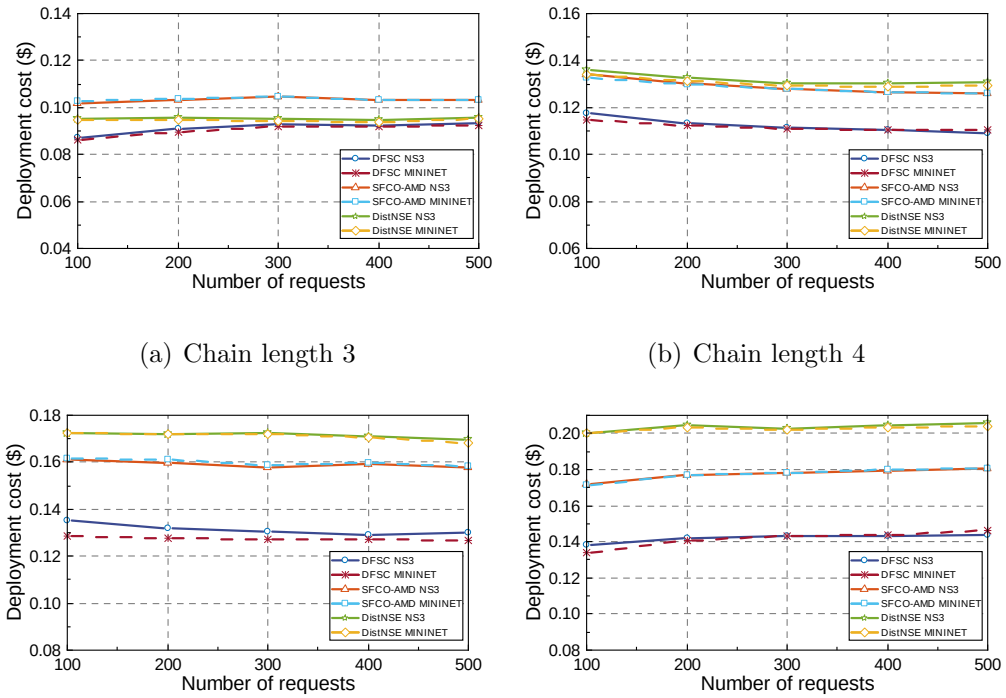


Figure 5.11: Number of requests vs. acceptance rate in topology Agis

Finally, the acceptance rate is shown in Figure 5.11. The acceptance rate is the ratio of the number of successfully placed requests to that of total incoming requests. DFSC achieves about 84 and 85% in NS3 and Mininet, respectively. In contrast, that of SFCO-AMD fluctuates around 85-87%. DistNSE achieves about 86%. We observe that SFCO-AMD and DistNSE only outperform DFSC by about 2%. This is because SFCO-AMD and DistNSE search for all paths between ingress and egress pairs. Although this can slightly improve the acceptance rate, it inevitably leads to a significant increase in the execution time.

5.5.4 Evaluation Results for Internode Topology

Then, we compare the performance of DFSC, SFCO-AMD and DistNSE in a large-scale topology. This topology consists of 66 nodes and 78 links and 9 domains. We do not implement the ILP solver for this topology as it cannot find feasible solutions in a reasonable time.



(a) Chain length 3

(b) Chain length 4

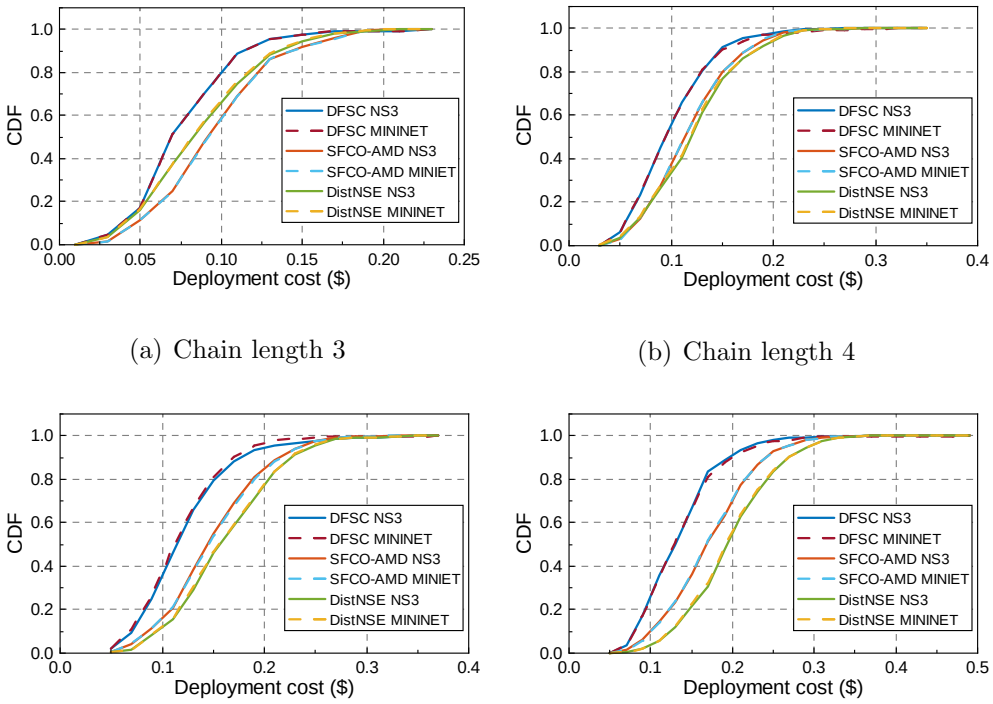
(c) Chain length 5

(d) Chain length 6

Figure 5.12: Number of requests vs. deployment cost for the topology Internode

Figure 5.12 shows the average deployment cost per request in NS3 and Mininet. We observe that DFSC outperforms SFCO-AMD and DistNSE in every case against a different number of requests and different lengths of chains. DFSC achieves the average deployment cost ranging from \$0.08 to \$0.09 for chain length 3, \$0.11 to \$0.12 for chain length 4, \$0.12 to \$0.13 for chain length 5 and \$0.13 to \$0.14 for chain length 6. The average deployment cost of SFCO-AMD ranges from \$0.10 to \$0.11 for chain length 3, \$0.12 to \$0.13 for chain length 4, \$0.15 to \$0.16 for chain length 5 and \$0.17 to \$0.18 for chain length 6. The deployment cost of DistNSE ranges from \$0.09 to \$0.95 for chain length 3, \$0.12 to \$0.13 for chain length 4, \$0.16 to \$0.17 for chain length 5 and \$0.2 to \$0.21 for chain length 6. DFSC effectively reduces the deployment cost by up to 20% compared with SFCO-AMD and DistNSE. The rationale is that DFSC achieves a nice balance between resource and traffic routing costs. The reduction of performance by SFCO-AMD is because of the inefficient partition of SFC requests. Similarly,

DistNSE only allows bidding for last selected sub-requests which leads to an increase of deployment cost. This result suggests that DFSC is cost-efficient and has stable performance.



(a) Chain length 3

(b) Chain length 4

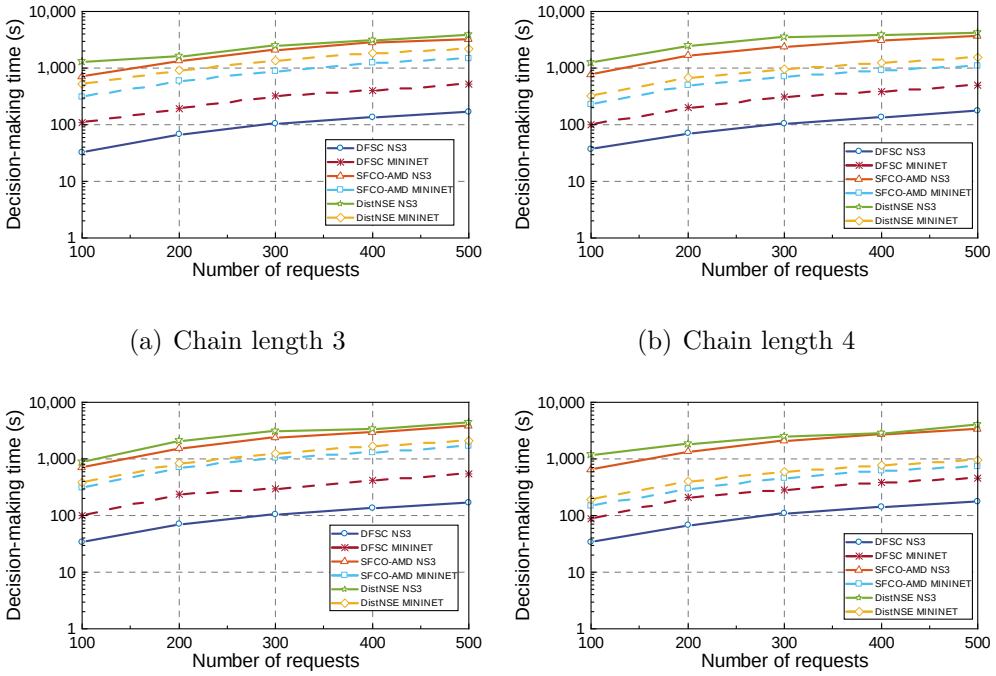
(c) Chain length 5

(d) Chain length 6

Figure 5.13: CDF of deployment cost for the topology Internode

Figure 5.13 shows the distribution of the average deployment cost with 500 SFC requests. DFSC NS3 and DFSC Mininet perform better than SFCO-AMD and DistNSE in NS3 and Mininet in most cases. In chain length 3, DFSC achieves \$0.21 at 99^{th} percentile while that of SFCO-AMD and DistNSE are are \$0.21. In chain length 4, DFSC achieves \$0.22 at 99^{th} percentile while that of SFCO-AMD and DistNSE are \$0.23. In chain length 5, DFSC achieves \$0.29 at 99^{th} percentile while that of SFCO-AMD and DistNSE are \$0.29 and 0.31\$, respectively. In chain length 6, DFSC achieves \$0.29 at 99^{th} percentile while that of SFCO-AMD and DistNSE fluctuates around \$0.33-0.35. Overall, DFSC reduces the deployment cost by up to 20% even at 99^{th} percentile because it consolidates as many as possible VNFs in the low-cost domain while considering the traffic routing cost on the paths.

Then, we demonstrate the total decision-making time for the algorithms in Figure 5.14. We observe that DFSC outperforms SFCO-AMD and DistNSE in both NS3 and Mininet. The decision-making time of DFSC rises much slower than that of SFCO-AMD and DistNSE in most cases. In NS3, the decision-making time



(a) Chain length 3

(b) Chain length 4

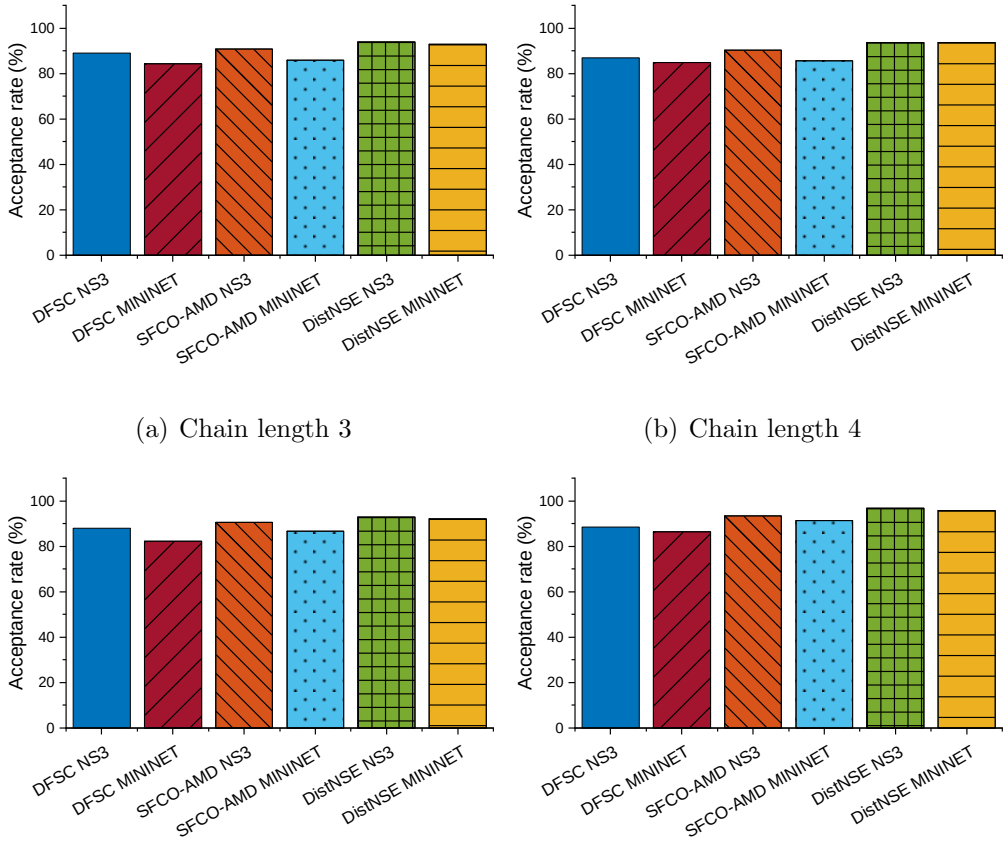
(c) Chain length 5

(d) Chain length 6

Figure 5.14: Number of requests vs. decision-making time for the topology Internode

of DFSC ranges from 33 seconds to 160 seconds on average. In contrast, it takes 10 to 50 minutes for SFCO-AMD to make decisions. It takes DistNSE 20 to 50 minutes to make decisions. Overall, in NS3, the average decision-making time per request of DFSC, SFCO-AMD and DistNSE are about 0.3 seconds, 6.5∼7 seconds and 7∼13 seconds, respectively. We observe the similar trend in Mininet. It takes DFSC about 90 seconds to 7 minutes to make decisions in Mininet. However, it takes 5 to 25 minutes for SFCO-AMD and DistNSE to make decisions. Overall, in Mininet, the average decision-making time per request of DFSC, SFCO-AMD and DistNSE are about 1 second, 1.5∼3 seconds and 2∼5 seconds, respectively. This is because DFSC uses a distributed approach to run the decision-making process in parallel on multiple orchestrators. Another factor is that DFSC uses a k-shortest path algorithm instead of searching all available paths. This result suggests DFSC scales well when the number of SFC requests increases. Although DistNSE also runs in a distributed manner, it needs to process bidding among domains which increases the decision-making time. We also observe that the gap among DFSC, SFCO-AMD and DistNSE in NS3 is larger than that in Mininet. This is because DFSC spends more time making decisions in Mininet than NS3 as DFSC serializes the messages into the disk in Mininet. In NS3, the communication among orchestrators is achieved by global variables. Overall, DFSC is at least 70%

faster than SFCO-AMD and DistNSE either in NS3 or Mininet.



(a) Chain length 3

(b) Chain length 4

(c) Chain length 5

(d) Chain length 6

Figure 5.15: Acceptance rate for the topology Internode

After that, we examine the acceptance rate of the algorithms. Figure 5.15 shows the acceptance rate of the algorithms against different chain lengths. The acceptance rate of DFSC fluctuates around 84∼89%. In contrast, the acceptance rate of SFCO-AMD ranges from 85∼92%. The acceptance rate of DistNSE is about 92∼94%. On average, SFCO-AMD and DistNSE outperform DFSC by 2-3%. The rationale is that, SFCO-AMD and DistNSE embed an algorithm that finds all candidate paths between source and target nodes, resulting in placing more SFC requests than DFSC. Overall, DFSC achieves a comparable acceptance rate.

Overall, fine-grained DFSC reduces the deployment cost significantly by up to 20% compared with SFCO-AMD and DistNSE at the cost of low performance degradation in acceptance rate. Meanwhile, DFSC shortens the decision-making time by at least 70%.

5.6 Summary

In this chapter, we study the SFC placement problem in multi-domain networks. We formulate an ILP problem, aiming to minimize the deployment cost as well as improve the scalability. We introduce two proposed algorithms, i.e., coarse-grained DFSC and fine-grained DFSC. Coarse-grained DFSC enables SFC-granularity constraints while fine-grained DFSC enables VNF-granularity constraints. Hence, fine-grained DFSC allows tailored policies for service providers in a flexible and agile manner. DFSC first builds an aggregated graph by using the full-mesh aggregation which reduces the amount of shared information. Then, DFSC uses a cost-aware algorithm to place VNFs and route the traffic.

We implemented DFSC in Gurobi, NS3 and Mininet, respectively. We benchmark DFSC with the optimum and other approaches against a different number of requests and different chain lengths. Overall, DFSC reduces the deployment cost remarkably as it strikes a nice balance between the resource and routing cost. Similarly, DFSC shortens the decision-making time significantly due to its distributed nature. The decision-making process is distributed to multiple orchestrator and executed parallelly. The trade-off is that DFSC is outperformed by SFCO-AMD and DistNSE in terms of acceptance rate. This is because SFCO-AMD and DistNSE find all candidate paths in the network while DFSC only uses k-shortest path algorithm to save time. Overall, DFSC achieves near-optimal deployment cost while reducing the execution time by at least 70%. This result indicates that DFSC processes the SFC request in a fast manner.

In the next chapter, we present the bottleneck-aware VNF scaling and flow routing algorithm that is a key enabler for the system scalability.

Chapter 6

Bottleneck-aware VNF Scaling

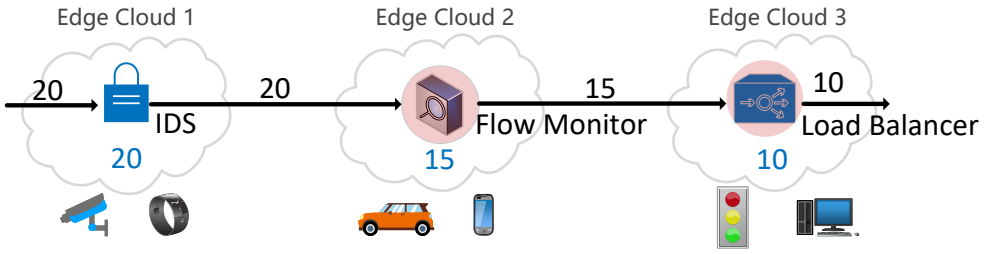
In this chapter, we continue to investigate how to improve the system scalability for service chain federation. To this end, we study the VNF scaling and flow routing problem in edge clouds. Edge computing delivers a wide range of applications with low latency for end-users, which advocates real-time applications such as augmented reality and self-driving. As edge clouds are usually limited in terms of resource capacities, we use VNF scaling technology to better utilize the valuable and scarce resources at the edge. In VNF scaling, every VNF comprises one or multiple VNF instances which are VNF replicas in virtual machines or containers. Every instance accommodates a few incoming traffic flows to avoid resource limitations incurred by computation and network resource constraints. In this chapter, we provide a bottleneck-aware VNF scaling algorithm and a subsequent flow routing algorithm that dynamically accommodate time-varying traffic workloads, shorten the end-to-end latency and reduce the resource utilization. Extensive experiments show that the proposed algorithm achieves near-optimal latency and improves the VNF utilization rate by 15%. Also, the execution time is significantly reduced by up to 87.2%.

6.1 Introduction

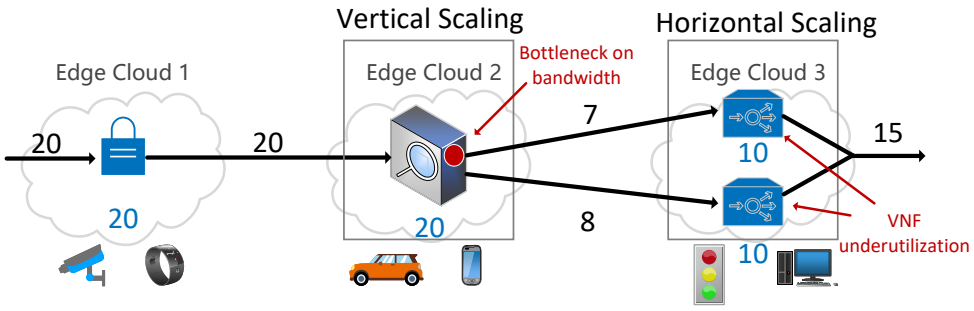
As introduced in Chapter 1, it is vitally important to improve the system scalability as the number of SFC requests will increase in service chain federation. A promising approach is to improve the VNF scalability, aiming to accommodate the time-varying traffic workloads. In the meantime, novel applications such as augmented reality and self-driving cars call for low latency.

Since edge computing handles computational tasks at the proximity of end-users, it can reduce the end-to-end latency and the expenditures for service providers. Nevertheless, edge clouds usually have limited capacity compared with

public clouds. Hence, we consider federating edge clouds to better use the scarce and valuable physical resources. VNF scaling in edge clouds enables multiple VNF instances that are VNF replicas on virtual machines or containers. Every VNF instance processes a few incoming flows to avoid resource limitations incurred by different computation or network resources (i.e., CPU, memory and bandwidth). In this chapter, we focus on a crucial problem: How to minimize the end-to-end latency of SFCs while improving the VNF utilization at edge clouds with VNF scaling?



(a) An overloaded service chain



(b) After scaling by using existing solutions

Figure 6.1: Existing solutions of vertical and horizontal scaling

It is very challenging to address the above problem in edge clouds. Resources need to be shared among geo-distributed edge clouds because the traffic demands are significantly different at different locations or points of time. For example, the traffic demand concentrates at offices in working hours while gaming traffic is aggregated at homes during the evening. Hence, the bandwidth resource on hotspot links can be insufficient in peak hours.

However, most existing approaches largely ignore two issues. First, VNFs cannot be scaled without limitations. Many works report that the performance of VNFs can be easily bottlenecked on CPU, memory or bandwidth [9], [10], [11]. Second, most existing approaches overlook the VNF under-utilization incurred by resource reservation which deteriorates the resource scarcity in edge clouds.

VNFs fall into two categories, i.e., I/O-bound and non-I/O bound. Fig-

ure 6.1(a) shows a flow monitor that is I/O-bound. Since the incoming traffic rate is 20 which is greater than the processing capacity, the flow monitor becomes overloaded. Figure 6.1(b) illustrates the existing solutions. The blue numbers represent the processing capacity of a VNF. The black numbers denote the traffic rate in a link. Existing solutions vertically scale up the flow monitor by adding the processing capacity to 20. Nevertheless, the output rate of the flow monitor is still 15 as flow monitor is limited by bandwidth. In other words, simply scaling up the VNF by increasing the processing capacity cannot improve the performance of I/O-bound VNFs. Similarly, load balancer is also saturated because the incoming traffic is greater than its processing capacity. Existing horizontal solutions scale out by spawning two new instances with a processing capacity of 10. However, the incoming traffic rates are 7 and 8, respectively. Hence, horizontal scaling leads to resource over-provisioning and VNF under-utilization.

In this chapter, we propose a bottleneck-aware VNF scaling and traffic routing approach to overcome the above limitations.

6.2 Problem Description

6.2.1 Delay at VNFs in Edge Clouds

The average delay at VNFs in edge clouds is formulated in Equation (6.1).

$$D_{n_i^j}(t) = \frac{1}{\mu_{n_i^j}(t) - \lambda_{n_i^j}(t)} \tag{6.1}$$

where $\mu_{n_i^j}(t)$ and $\lambda_{n_i^j}(t)$ are the processing rate and the arrival rate at VNF instance n_i^j, respectively.

6.2.2 Link Delay in Inter-cloud Links

Similarly, the average delay in inter-cloud links is defined as follows. We use $C_{uv}(t)$ to denote the processing capacity of the link uv. We use $B_{uv}(t)$ to represent the total traffic rate in the link uv.

$$L_{uv}(t) = \frac{1}{C_{uv}(t) - B_{uv}(t)}, \quad \forall (u,v) \in \mathcal{E} \tag{6.2}$$

6.2.3 Objective Function

We formulate the ILP problem with respect to delays both in edge clouds and inter-cloud links. We approximate these delays by using the average delays.

$$\begin{aligned} min \quad & \sum_{f\in\mathcal{F}}\sum_{v\in\mathcal{V}}\sum_{n_i^j\in\mathcal{N}} x_{fv}^{n_i^j}(t)D_{n_i^j}(t) \\ +\sum_{f\in\mathcal{F}}\sum_{uv\in\mathcal{E}}\sum_{n_i^j,n_{i+1}^j\in\mathcal{N}} & y_{fuv}^{n_i^j n_{i+1}^j}(t)L_{uv}(t) \end{aligned} \tag{6.3}$$

where indicator variable $x_{fv}^{n_i^j}(t)$ equals 1 if flow f traverses the VNF instance n_i^j hosted in edge cloud v. Similarly, indicator variable $y_{fuv}^{n_i^j n_{i+1}^j}(t)$ equals 1 if flow f traverses the virtual link $n_i^j n_{i+1}^j$ mapped to the link uv.

$$\sum_{n_i^j\in n_i}\sum_{v\in V} x_{fv}^{n_i^j}(t) = 1, \quad \forall n_i \in \mathcal{N}, \quad \forall f \in \mathcal{F} \tag{6.4}$$

$$\sum_{f\in\mathcal{F}}\sum_{n_i^j\in\mathcal{N}} x_{fv}^{n_i^j}(t)c_{n_i^j}(t) \leq C_v(t), \quad \forall v \in \mathcal{V} \tag{6.5}$$

$$\sum_{f\in\mathcal{F}}\sum_{n_i^j,n_{i+1}^j\in\mathcal{N}} y_{fuv}^{n_i^j n_{i+1}^j}(t)b_{fuv}(t) \leq C_{uv}^{bw}(t) \tag{6.6}$$

Equation (6.4) guarantees that every flow traverses only one instance of each VNF. Equation (6.5) guarantees that the required processing capacities must not exceed the remaining processing capacity $C_v(t)$. Also, equation (6.6) guarantees that the traffic arrival rate must not exceed the remaining bandwidth capacity $C_{uv}^{bw}(t)$.

6.3 B-Scale Algorithm

The VNF scaling problem has been proved to be NP-hard in many existing works [2], [62]. Assume that every VNF instance n_i^j represents an item m with the size $m_i^j.req$, where the size $m_i^j.req = c_{n_i^j}$. Let every edge cloud v denote a knapsack k with a capacity $k_v.cap = C_v$. Assume that the profit of assigning flows to every VNF instance is the negative of the flow delays. Then, our VNF scaling problem becomes finding VNF placement for every flow that maximizes the total profit. In other words, our problem becomes a Multiple Knapsack Problem (MKP) which is NP-hard[146].

6.3.1 Online Bottleneck-aware VNF Scaling Algorithm

Algorithm 3 shows the online bottleneck-aware VNF scaling algorithm. For each VNF n_i, B-Scale first observes the actual traffic rate. If the traffic rate decreases compared with time slot $t-1$, the algorithm checks the existing VNF instances. If the traffic rate of a VNF instance equals 0, the algorithm removes the instance. If

Algorithm 3: Online bottleneck-aware VNF Scaling Algorithm for Latency Minimization

Input : Request r, Graph $\mathcal{G}$;
for t = *1,2,...,T* **do**
  for n_i *in* $\mathcal{N}$ **do**
    Observe actual traffic rate $\lambda_{n_i}(t)$ at VNF n_i;
    if $\lambda_{n_i}(t) \leq \lambda_{n_i}(t-1)$ **then**
      for n_i^j *in* n_i **do**
        if $\lambda_{n_i^j}(t) == 0$ **then**
          Release n_i^j;
        else if $\lambda_{n_i^j}(t) \leq \mu_{n_i^j}(t)$ **then**
          Scale down $\mu_{n_i^j}(t)$ to $\lambda_{n_i^j}(t)$;
        end
      end
    else
      Call **Place()** to deploy VNFs;
      Call **Traffic steer()** to route traffic;
    end
  end
  Update residual processing $C_v(t)$ and bandwidth capacity $C_{uv}^{bw}(t)$, $\forall v \in \mathcal{V}$, $\forall uv \in \mathcal{E}$;
end

the traffic rate at an instance is smaller than its processing capacity, the algorithm scales down the processing capacity $\mu_{n_i^j}$ to the traffic rate $\lambda_{n_i^j}$. Otherwise, if the traffic rate $\lambda_{n_i^j}(t)$ increases compared with time slot $t-1$, the algorithm invokes Algorithm 4 to embed new VNF instances. If all VNF instances are deployed, the algorithm uses Algorithm 5 to steer the traffic for service chains. Finally, the algorithm updates the bandwidth and processing capacities of the system.

6.3.2 New Instance Placement in Edge Clouds

First, Algorithm 4 computes the k shortest paths between ingress and egress nodes. After that, the algorithm checks the VNF category. If the VNF is non-I/O bound, the algorithm performs vertical scaling (scale-up) based on the required new processing capacity $\mu_{n_i^j}^{new}(t)$. Then, we use the first-fit edge cloud on the path p to host this instance. In this thesis, the term "first-fit" refers to the first edge cloud with sufficient processing capacity. If the vertical scaling fails due to the limited capacity in the edge cloud, the algorithm performs horizontal scaling instead. The algorithm computes the required number of new instances $z_{n_i}^{new}(t)$. Then, the algorithm assigns the new instances to the first-fit edge cloud. In contrast, if the VNF is I/O bound, the algorithm only performs horizontal scaling as vertical scaling cannot improve the performance.

Algorithm 4: Place()

```
Compute the set P of k shortest paths between S and T;
for shortest path p in P do
    if n_i is non-I/O bound then
        Compute new required processing capacity μ_{n_i}^{new}(t);
        Choose the first-fit edge cloud v in p;
        Perform vertical scaling acc. to μ_{n_i}^{new}(t);
        if Vertical scaling failed then
            Compute the number of new instances z_{n_i}^{new}(t);
            foreach newly created instance n_i^j do
                Choose first-fit edge cloud v in p;
                Creating new instance horizontally;
            end
        end
    end
    else if n_i is I/O bound then
        Compute the number of instance z_{n_i}^{new}(t);
        foreach newly created instance n_i^j do
            Choose first-fit edge cloud v in p;
            Creating new instance horizontally;
        end
    end
end
```

6.3.3 Traffic Steer Algorithm

Algorithm 5 shows the traffic steering algorithm. Different from existing solutions that use the shortest path algorithm, we use a k shortest path algorithm to take advantage of non-shortest paths. To avoid the complex state migration problem, we assume that we only steer new coming packets ("new flow") to these newly created instances at time slot t. In other words, the routing of existing packets remains unchanged.

At time slot t, there are several newly created instances for each VNF n_i. For every new flow f in the service chain, the algorithm assigns it to the least used instance n_i^j. Then, between every VNF instance n_i^j and the next-hop VNF instance n_{i+1}^j, the algorithm creates multiple candidate paths P' from the k shortest algorithm. After that, the algorithm selects the first-fit path with sufficient bandwidth to route the traffic.

6.3.4 Compared Algorithms

To benchmark the proposed algorithm, we adopt the Simulated Annealing (SA) algorithm [153]. We select SA because most existing approaches in the literature cannot be directly applied to our scenario. SA can approximate the global optimal

Algorithm 5: Traffic steer()

```
for new flow f do
    foreach VNF n_i do
        Find the least used new instance n_i^j;
        Check the residual processing capacity of n_i^j;
        Assign f to n_i^j;
    end
    foreach VNF n_i do
        for p' in k shortest paths P' between n_i^j and next-hop instance n_{i+1}^j do
            if Check bandwidth succeeds then
                Steer the flow along p';
                Update resource capacities;
                Break;
            end
        end
    end
end
```

solution as it is an efficient searching technique for optimization problems [154]. The SA first generates an initial solution. Then, SA changes the placement of one SFC to generate a new solution. After that, SA makes decisions to accept or reject the new solution. SA only accepts a solution if all capacity constraints are fulfilled. If the objective is not improved after one hundred iterations or the temperature is cooled down to 1, SA stops the termination and accepts the solution. We do not implement other solutions for comparison from the existing literature as they do not consider the VNF category in the scaling problem. This algorithm is implemented by us.

6.4 Performance Evaluation

6.4.1 Simulation Setup

We have conducted extensive simulations with Network simulator 3 (NS3) on a server with 105 GB RAM and an Intel(R) Xeon(R) E5645 processor with 24 cores. We derive the SFC requests from .pcap files of the CAIDA traces [155]. Each SFC comprises a chain of 3 VNFs randomly selected from Table 6.1.

Table 6.1: Processing time and processing capacity of VNFs

VNF	Processing time	Max. Processing capacity	I/O-bound
Firewall	120μs	400Mbps	Yes
IDS	160μs	600Mbps	No
Caching	83μs	580Mbps	No
Flow monitor	200μs	550Mbps	Yes
Load balancer	647.5μs	500Mbps	No

The processing time and capacity of VNFs are derived from related works [132]. Similarly, the VNF category is derived from [10], [9]. To adapt to the resource capacities in the system, we proportionally scale down the VNF processing capacity by a factor of 10. The average packet length is assumed to be 512 bytes. The processing capacity for each edge cloud is 400 Mbps. Also, the link bandwidth is 400 Mbps. The traffic change ratio is obtained from the work [156].

To prove the performance of the proposed algorithm, the metrics include the average latency, the execution time, the acceptance rate of request, the VNF utilization rate, the cumulative distribution function of the end-to-end latency.

6.4.2 Offline B-Scale in Topology Agis

We first compare B-Scale and SA with the offline optimum achieved by the ILP solver Gurobi [157]. As the ILP solver cannot find feasible solutions for large-scale networks or hundreds of SFC requests in a reasonable time, we only implement the ILP solver in a small-scale topology named Agis that consists of 25 nodes and 30 links from Internet Topology Zoo.

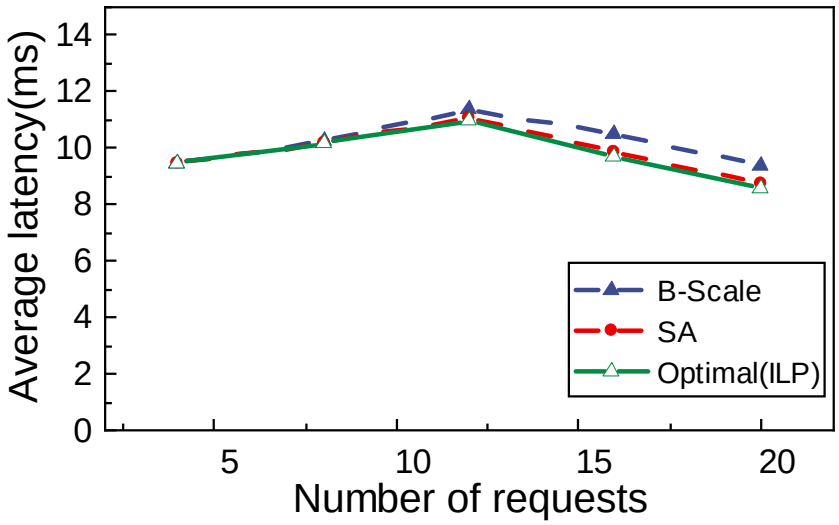


Figure 6.2: Average latency

Figure 6.2 demonstrates that SA achieves only 1% more latency that the offline optimum which suggests that SA approximates the global optimum well. Meanwhile, B-Scale only leads to 4% more latency compared with the optimum because B-Scale balances the network traffic among multiple VNF instances.

Figure 6.3 shows the execution time of the ILP solver, B-Scale and SA. Figure 6.3 illustrates that the ILP solver spends over 3400 seconds to find feasible solutions for only 20 requests. This result suggests that the ILP solver cannot scale well for substantial SFC requests. In contrast, B-Scale and SA only spend

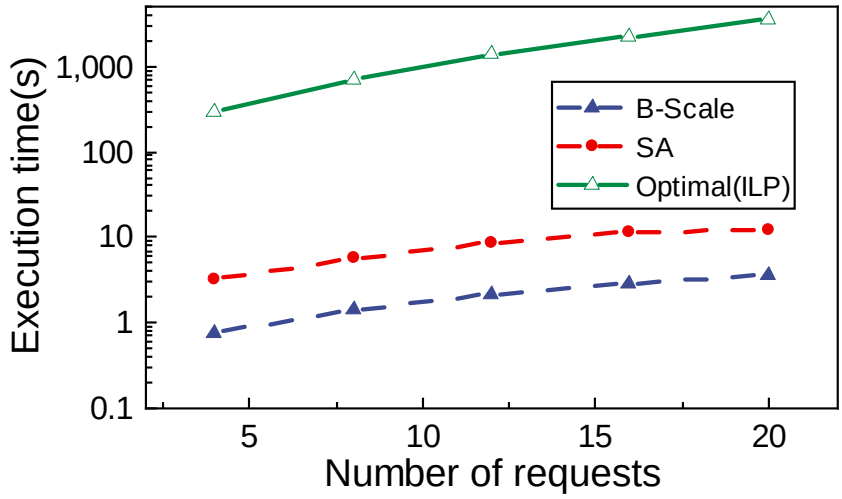


Figure 6.3: Execution time

12 seconds in the worst case. Overall, the results imply that SA can closely approximate the global optimum in a reasonable time and hence we use SA to prove the performance of B-Scale in a larger topology in the following sections.

6.4.3 Offline B-Scale in Topology TW

In this section, we use a US topology named TW consists of 76 nodes and 123 links from the Internet Topology Zoo because it contains a wide range of geo-distributed networks. First, we demonstrate the performance of B-Scale in an offline manner which means that all SFC requests start to run at the beginning and only end when the simulation is ended. We conducted two sets of experiments that simulated light and heavy workloads. Light means that the overall traffic demand is low in light of the processing and bandwidth capacities of the system, heavy means that it is high. The maximum traffic demand of light and heavy workloads are about 5400 and 9500 Mb/s, respectively. We run the B-Scale algorithm over a different number of SFC requests ranging from 20 to 700 with a step size 40. More SFC requests result in more network traffic which requires more VNF instances to process.

Figure 6.4 demonstrates that B-Scale closely matches the optimal solution approximated by SA both in light and heavy workloads. For the light workload, the latency of both algorithms only rises slightly from 12.2 to 17.2 seconds as the bandwidth resources are sufficient on the shortest path between ingress and egress pairs. In contrast, the average latency of SA increases from 11.3 to 14.5 seconds.

The average latency of both algorithms increases sharply for heavy workloads when the number of SFC requests is greater than 340. This is because the band-

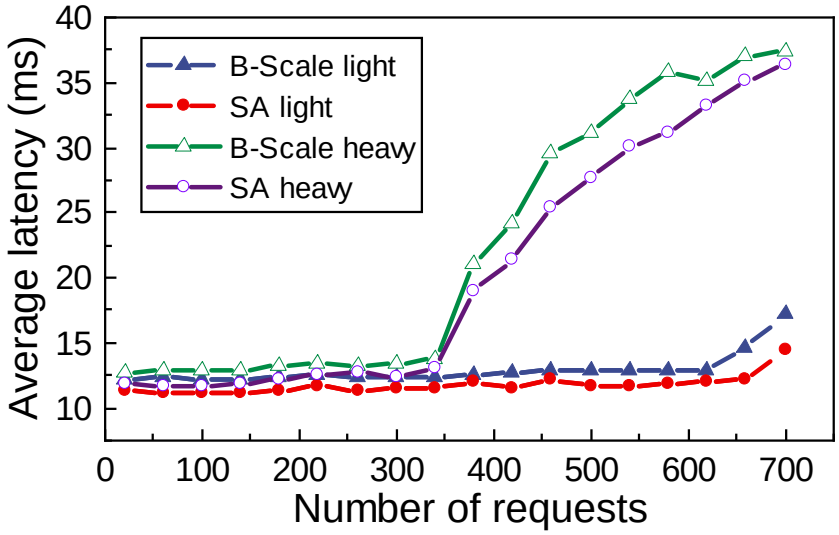


Figure 6.4: Average latency

width resources are gradually insufficient. The average latency of B-Scale ranges from 12.7 to 37.45 seconds. Similarly, SA achieves 11.84∼36.31 seconds. The latency achieved by B-Scale is only 9% higher than that of SA on average. This is because B-Scale balances the traffic among different paths and different VNF instances.

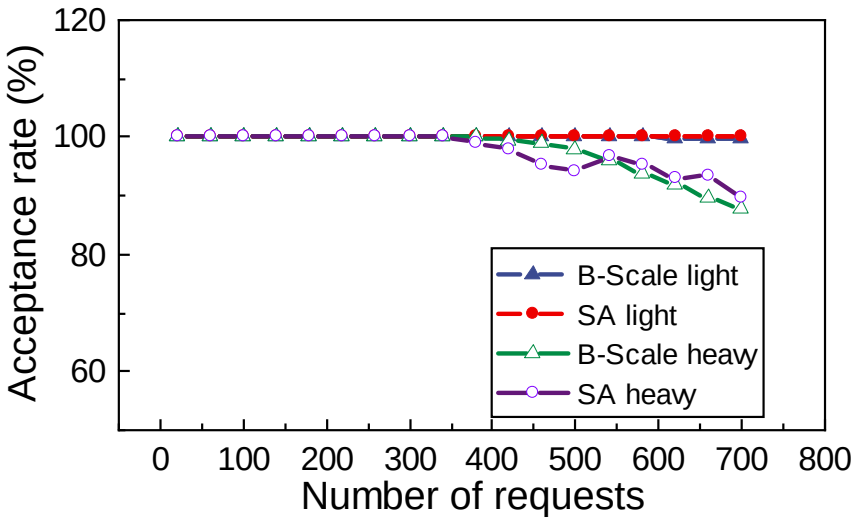


Figure 6.5: Acceptance rate

Figure 6.5 illustrates the acceptance rate which is the ratio between successfully embedded SFC requests and the total number of SFC requests. For light workload, B-Scale and SA both lead to almost 100% acceptance rate. The rationale is that the computation and network resources are sufficient with the light workload. For heavy workload, B-Scale achieves 87∼100% acceptance rate while that of SA

ranges from 89∼100%. B-Scale achieves only 2∼3% lower acceptance rate in a few cases compared with SA. This is because SA spends more time iterating different solutions and hence can find more feasible solutions than B-Scale. Overall, B-Scale achieves comparable acceptance rate to SA.

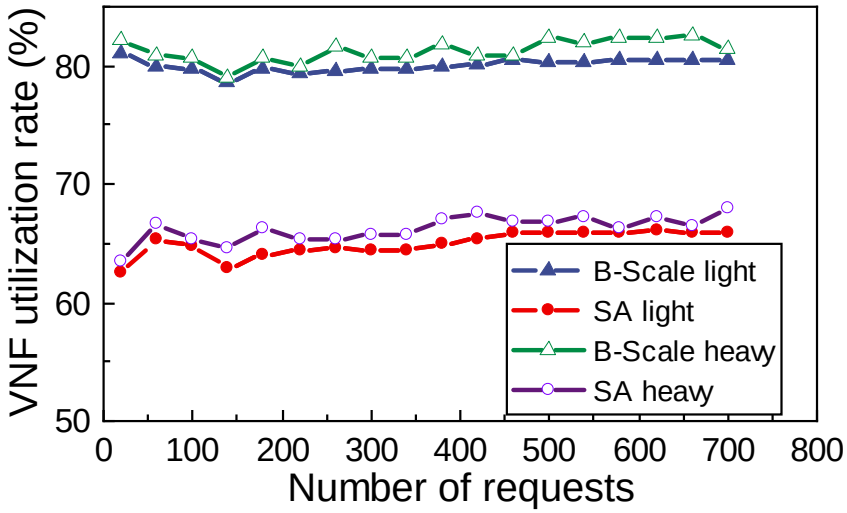


Figure 6.6: VNF utilization rate

Figure 6.6 shows the mean VNF utilization rate. The utilization rate is the ratio of used processing capacity to the total processing capacity of a VNF. The VNF utilization rate of B-Scale fluctuates around 78∼82% while that of SA ranges from 62∼67%. This is because B-Scale applies different kinds of scaling on different types of VNF. For either light or heavy workload, B-Scale performs better by about 15% in most cases. This result suggests that B-Scale effectively uses the computation resources and hence mitigates resource under-utilization in edge clouds.

Figure 6.7 illustrates the total execution time of both algorithms. The average execution time of B-Scale for each request ranges from 0.16 to 0.3 seconds. In contrast, SA achieves 0.3 to 2.9 seconds to process a request. B-Scale remarkably reduces the execution time by up to 87.2%. This is because SA iterates at least over hundreds of iterations to find a near-optimal solution. These results indicate that B-Scale works in a fast manner and hence is suitable for online orchestration.

6.4.4 Online B-Scale in Topology TW

We evaluate the performance of both algorithms in an online manner wherein SFC requests arrive and leave at different time slots. The average life cycle of every flow is 1000 seconds. We simulated the experiments over 18000 seconds. Figure 6.8 illustrates the distribution of the end-to-end latency. We observe that

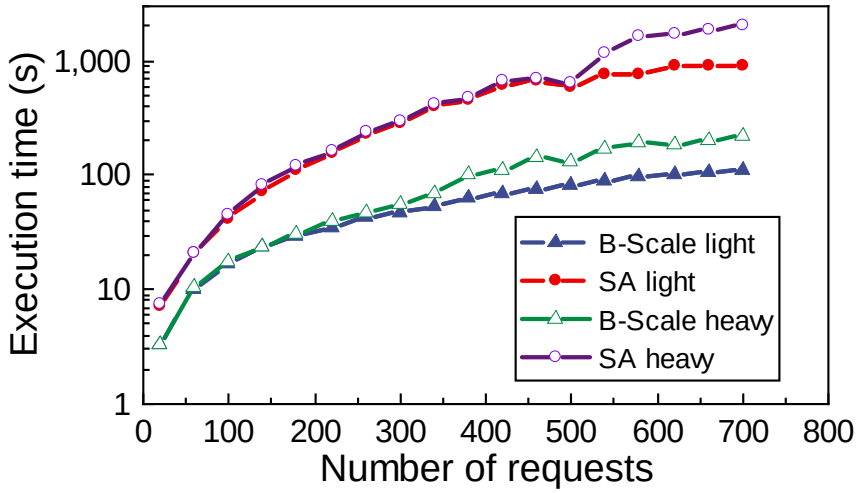


Figure 6.7: Execution time

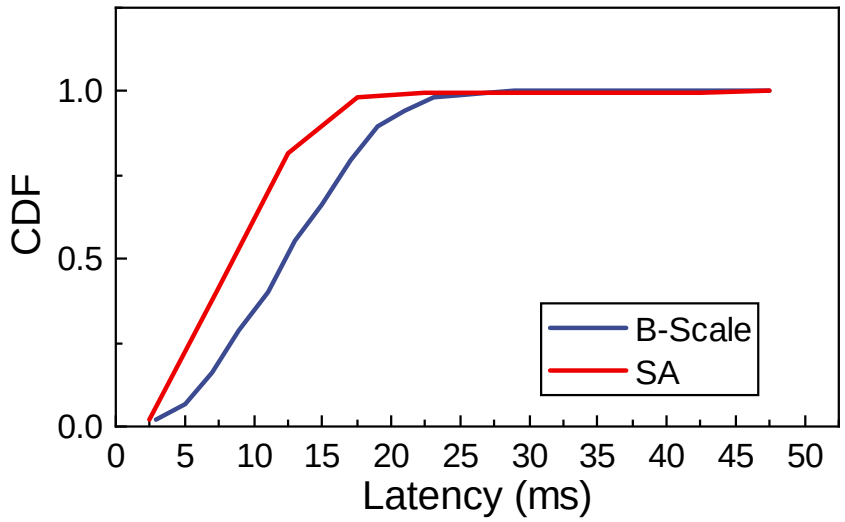


Figure 6.8: CDF of latency

the 99^{th} percentile latency of B-Scale is 25 ms. In contrast, that of SA is 22.5 ms. This result indicates that B-Scale leads to a comparable performance compared with SA. The rationale is that B-Scale distributes the VNF instances on the next shortest path with sufficient bandwidth. By this means, link congestion and VNF overload are avoided. As most SFC requests are deployed within 25 ms, the performance of B-Scale is proved.

Figure 6.9 demonstrates that B-Scale achieves about 79∼85% VNF utilization rate while that of SA fluctuates around 64∼72%. B-Scale significantly improves the VNF utilization rate by up to 15 % which indicates that B-Scale effectively uses the computation resources and mitigates VNF under-utilization. The rationale is that B-Scale uses both vertical and horizontal scaling simultaneously to optimize

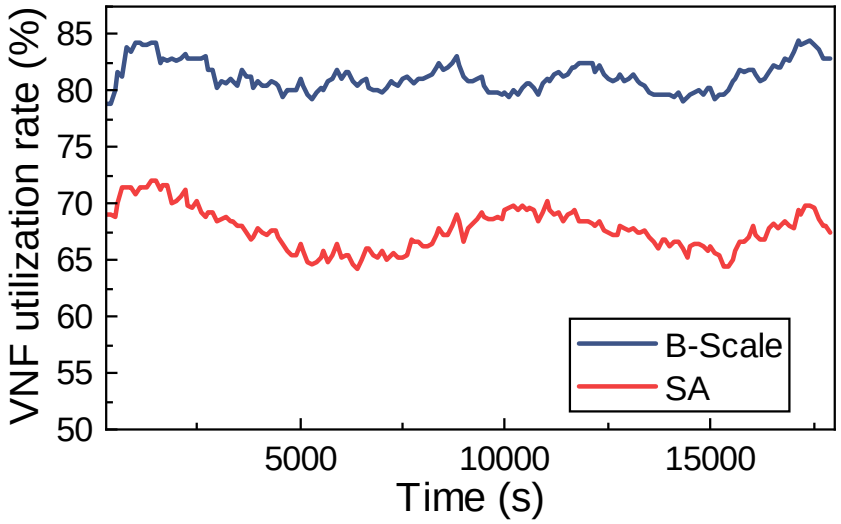


Figure 6.9: VNF utilization rate

the VNF utilization. These findings suggest that B-Scale improves the resource utilization and hence is suitable for SFC orchestration in edge clouds because the resource capacity of edge clouds is limited compared with public clouds.

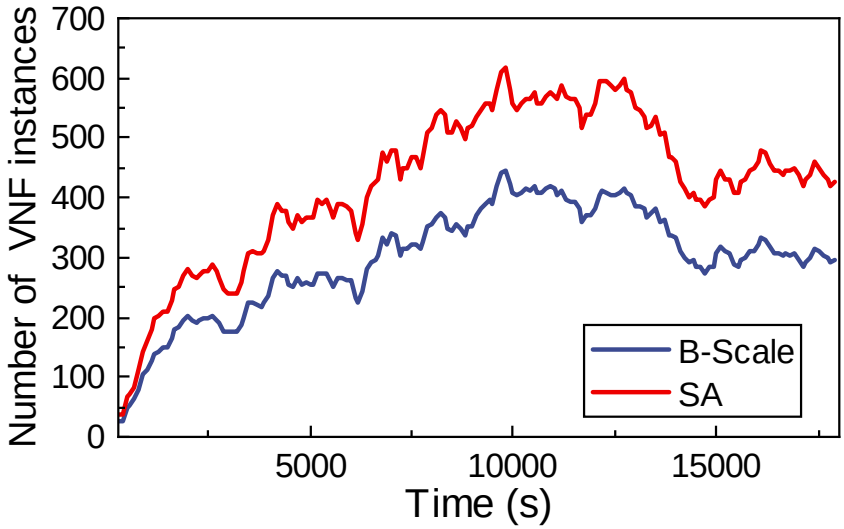


Figure 6.10: Number of instances

Figure 6.10 illustrates the number of VNF instances over a wide range of time slots. B-Scale remarkably reduces the number of VNF instances by about 33 % which suggests that B-Scale uses the system resources in an efficient manner. This is because B-Scale strikes a nice balance between vertical and horizontal scaling based on the VNF category.

Finally, Figure 6.11 shows the number of hops for flows. We observe that 99^{th} percentile number of hops is 10.5 for B-Scale while that of SA is 9.5. This is

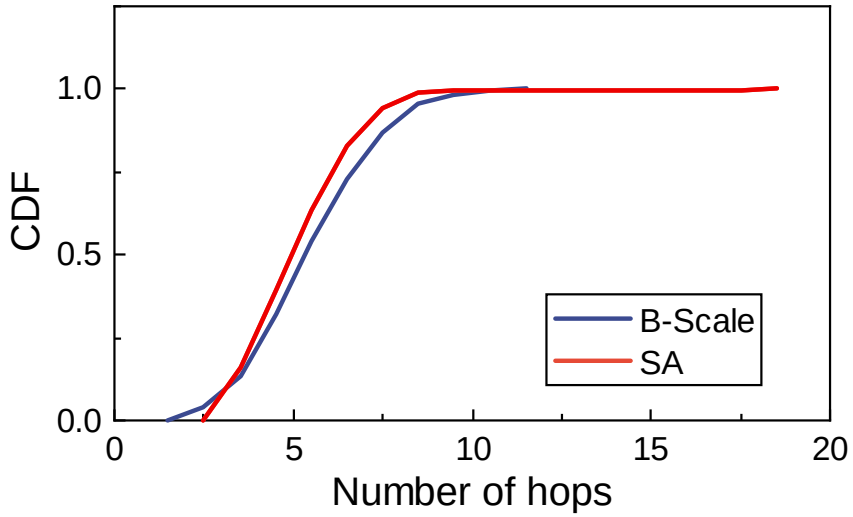


Figure 6.11: CDF of number of hops

because B-Scale strives to consolidate VNF instances on the shortest path and hence avoids using longer paths. This result indicates that B-Scale approximates the number of hops achieved by SA.

Overall, B-Scale achieves near-optimal end-to-end latency (9% higher) approximated by SA at the cost of light degradation in acceptance rate. Also, B-Scale significantly reduces the execution time by about 87.2% which means that it scales well especially when the problem size increases. Moreover, B-Scale significantly improves the VNF utilization rate by 15% which mitigates the resource scarcity in edge clouds.

6.5 Summary

In this chapter, we study the end-to-end latency minimization problem while improving the VNF utilization at edge clouds by using VNF scaling. We first formulate an ILP problem by considering delays at VNFs and links. These delays are approximated by the average delay derived from Little's Theorem.

We propose a novel VNF scaling algorithm with respect to the VNF category. The proposed algorithm checks every VNF's category. If the VNF is non-I/O bound, the proposed algorithm strives to perform vertical scaling before horizontal scaling. If the VNF is I/O bound, the proposed algorithm performs only horizontal scaling as vertical scaling cannot improve the VNF performance in this case. Besides, a k shortest path based algorithm is proposed to steer the network traffic which takes advantage of non-shortest path. Different from most existing works that use the shortest path between VNF pairs, we choose the k shortest

path as the bandwidth becomes insufficient in peak hours. Extensive trace-driven experiments show that the proposed algorithm achieves near-optimal performance. Also, the proposed algorithm efficiently reduces the execution time by up to 87.2% and improves the VNF utilization rate by about 15%. These findings suggest that the proposed algorithm handles SFC requests in a fast manner and mitigates the resource under-utilization in edge clouds. This is of significant importance because edge clouds usually have limited resource capacity.

In the next chapter, we conclude this thesis and point out future works.

Chapter 7

Conclusion and Future work

This chapter summarizes the main conclusions of this thesis. Furthermore, this chapter includes a discussion about promising directions for the future work.

7.1 Conclusion

This thesis proposes a distributed architecture for service chaining in multi-domain networks.

The proposed distributed architecture for service chains significantly reduces the deployment cost by 12∼20% compared to benchmark algorithms. Also, the proposed DFSC achieves a factor of 1.15 compared to the optimal solution. The rationale is that DFSC jointly considers the resource and routing costs. The proposed distributed architecture significantly reduces the decision-making time by 70%. This is because the proposed distributed architecture distributes the decision-making process to multiple orchestrators. The trade-off is that we observe a 2∼3% performance degradation in the acceptance rate of SFC requests. This is because benchmark algorithms search more candidate paths compared to DFSC. These results imply that DFSC makes decisions in a fast manner while reducing the deployment cost for the network operators.

Moreover, we proposed B-Scale algorithm to handle the time-varying network traffic. Extensive simulation shows that the proposed B-Scale algorithm achieves about 9% higher latency than that of SA in a large-scale network. However, B-Scale reduces the execution time by up to 87.2% compared with SA. Also, B-Scale improves the VNF utilization rate by 15% which suggests that B-Scale is suitable for SFC orchestration in edge clouds. The rationale is that B-Scale considers the VNF category when performing VNF scaling. These results suggest that B-Scale processes the SFC request in a fast manner and reduces the resource waste in VNF scaling by improving the VNF utilization rate.

7.2 Future Work

The work presented in this thesis can be extended in the following aspects.

- Regarding the scalability for the system-level, there are many other research objectives such as energy optimization and reliability. Another promising research path is exploring the cloud-edge federation with respect to programmable network devices such as P4 switch. Programmable devices run network applications on network devices and hence offer the speed and agility without altering the existing network architecture. For example, a P4-defined data plane can be used to create service chains between virtual and physical network functions. Also, sophisticated forwarding rules can be implemented and executed for the incoming traffic flows. The matching process can be processed by a combination of packet header fields and user-defined metadata.

- Regarding the scalability for the VNF-level, finding an appropriate approach to jointly achieve VNF scaling and migration still remains open for research. Another promising direction is to predict the traffic demand and the invocation pattern of network functions by using techniques such as Arima model, machine learning and Histogram. For instance, ARIMA model can be used to analyze and forecast time series data. A histogram can be used to exhibit the invocation pattern of network functions. Machine learning approaches such as TensorFlow can also be used to predict incoming workloads. By this means, the algorithm can proactively allocate computational resources for VNFs and hence optimize the SFC deployment.

References

[1] Song Yang, Fan Li, Stojan Trajanovski, Xu Chen, Yu Wang, and Xiaoming Fu. Delay-aware virtual network function placement and routing in edge clouds. *IEEE Transactions on Mobile Computing*, 20(2):445–459, 2021.

[2] Racha Gouareb, Vasilis Friderikos, and Aghvami. Virtual Network Functions Routing and Placement for Edge Cloud Latency Minimization. *IEEE Journal on Selected Areas in Communications*, 36(10):2346–2357, oct 2018.

[3] Gang Sun, Yayu Li, Dan Liao, and Victor Chang. Service function chain orchestration across multiple domains: A full mesh aggregation approach. *IEEE Transactions on Network and Service Management*, 15(3):1175–1191, sep 2018.

[4] Flavio Esposito, Maria Mushtaq, Michele Berno, Gianluca Davoli, Davide Borsatti, Walter Cerroni, and Michele Rossi. Necklace: An architecture for distributed and robust service function chains with guarantees. *IEEE Transactions on Network and Service Management*, 18(1):152–166, 2021.

[5] Josué Castañeda, Saul E. Pomares Hernández, Sami Yangui, Julio C. Pérez Sansalvador, Lil M. Rodríguez Henríquez, and Khalil Drira. Vnf-based network service consistent reconfiguration in multi-domain federations: A distributed approach. *Journal of Network and Computer Applications*, 195:103226, 2021.

[6] Zijun Zhang, Zongpeng Li, Chuan Wu, and Chuanhe Huang. A scalable and distributed approach for nfv service chain cost minimization. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2151–2156, 2017.

[7] Lin He, Lishan Li, and Ying Liu. Towards chain-aware scaling detection in nfv with reinforcement learning. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pages 1–10, 2021.

[8] Kaige Qu, Weihua Zhuang, Xuemin Shen, Xu Li, and Jaya Rao. Dynamic resource scaling for vnf over nonstationary traffic: A learning approach. *IEEE Transactions on Cognitive Communications and Networking*, 7(2):648–662, 2020.

[9] S. Van Rossem, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester. Profile-based resource allocation for virtualized network functions. *IEEE Transactions on Network and Service Management*, 16(4):1374–1388, 2019.

[10] Hui Yu, Jiahai Yang, Carol Fung, Raouf Boutaba, and Yi Zhuang. Ensc: Multi-resource hybrid scaling for elastic network service chain in clouds. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 34–41, 2018.

[11] Wajdi Hajji, Thiago A.L. Genez, Fung Po Tso, Lin Cui, and Iain Phillips. Dynamic Network Function Chain Composition for Mitigating Network Latency. In *Proceedings - IEEE Symposium on Computers and Communications*, volume 2018-June, pages 316–321. Institute of Electrical and Electronics Engineers Inc., nov 2018.

[12] Nfv_white_paper_update_final. https://portal.etsi.org/nfv/nfv_white_paper2.pdf.

[13] Shankar Lal, Tarik Taleb, and Ashutosh Dutta. Nfv: Security threats and best practices. *IEEE Communications Magazine*, 55(8):211–217, 2017.

[14] Sheng Chen, Baochao Chen, Junjie Xie, Xiulong Liu, Deke Guo, and Keqiu Li. Joint service placement for maximizing the social welfare in edge federation. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pages 1–6, 2021.

[15] Ed. J. Halpern and Ed. C. Pignataro. RFC 7665 - Service Function Chaining (SFC) Architecture. *Internet Engineering Task Force (IETF) - Request for Comments: 7665*, pages 487–492, 2015.

[16] Surendra Kumar, Mudassir Tufail, Sumandra Majee, Claudiu Captari, and Shunsuke Homma. Service Function Chaining Use Cases In Data Centers. Internet-Draft draft-ietf-sfc-dc-use-cases-06, Internet Engineering Task Force, February 2017. Work in Progress.

[17] Ming Mao and Marty Humphrey. A performance study on the vm startup time in the cloud. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 423–430, 2012.

[18] Wind river software — safe, secure, reliable. https://www.windriver.com/. (Accessed on 12/22/2021).

[19] Mohammad Abu-Lebdeh, Diala Naboulsi, Roch Glitho, and Constant Wette Tchouati. On the placement of vnf managers in large-scale and distributed nfv systems. *IEEE Transactions on Network and Service Management*, 14(4):875–889, 2017.

[20] L Cui, F Tso, and W Jia. Federated service chaining: Architecture and challenges. *IEEE Communications Magazine*, 58(3):47–53, 2020.

[21] Nassima Toumi, Olivier Bernier, Djamal-Eddine Meddour, and Adlen Ksentini. On cross-domain service function chain orchestration: An architectural framework. *Computer Networks*, 187:107806, 2021.

[22] Yu Chen, Sheng Zhang, Yibo Jin, Zhuzhong Qian, Mingjun Xiao, Jidong Ge, and Sanglu Lu. Locus: User-perceived delay-aware service placement and user allocation in mec environment. *IEEE Transactions on Parallel and Distributed Systems*, 33(7):1581–1592, 2022.

[23] Xiaojun Shang, Zhenhua Liu, and Yuanyuan Yang. Online service function chain placement for cost-effectiveness and network congestion control. *IEEE Transactions on Computers*, 71(1):27–39, 2022.

[24] Morteza Golkarifard, Carla Fabiana Chiasserini, Francesco Malandrino, and Ali Movaghar. Dynamic vnf placement, resource allocation and traffic routing in 5g. *Computer Networks*, 188:107830, 2021.

[25] Danyang Zheng, Huaxi Gu, Wenting Wei, Chengzong Peng, and Xiaojun Cao. Network service chaining and embedding with provable bounds. *IEEE Internet of Things Journal*, 8(9):7140–7151, 2021.

[26] Qixia Zhang, Fangming Liu, and Chaobing Zeng. Online adaptive interference-aware vnf deployment and migration for 5g network slice. *IEEE/ACM Transactions on Networking*, 29(5):2115–2128, 2021.

[27] Heng Yu, Zhilong Zheng, Junxian Shen, Congcong Miao, Chen Sun, Hongxin Hu, Jun Bi, Jianping Wu, and Jilong Wang. Octans: Optimal placement of service function chains in many-core systems. *IEEE Transactions on Parallel and Distributed Systems*, 32(9):2202–2215, 2021.

[28] Weihan Chen, Zhiliang Wang, Han Zhang, Xia Yin, and Xingang Shi. Cost-efficient dynamic service function chain embedding in edge clouds. In

2021 17th International Conference on Network and Service Management (CNSM), pages 310–318, 2021.

[29] Zichuan Xu, Haozhe Ren, Weifa Liang, Qiufen Xia, Wanlei Zhou, Guowei Wu, and Pan Zhou. Near optimal and dynamic mechanisms towards a stable nfv market in multi-tier cloud networks. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.

[30] Haojun Huang, Cheng Zeng, Yangmin Zhao, Geyong Min, Yingying Zhu, Wang Miao, and Jia Hu. Scalable orchestration of service function chains in nfv-enabled networks: A federated reinforcement learning approach. *IEEE Journal on Selected Areas in Communications*, 39(8):2558–2571, 2021.

[31] J. Martín-Peréz, F. Malandrino, C. F. Chiasserini, and C. J. Bernardos. Okpi: All-kpi network slicing through efficient resource allocation. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 804–813, 2020.

[32] Tao Gao, Xin Li, Yu Wu, Weixia Zou, Shanguo Huang, Massimo Tornatore, and Biswanath Mukherjee. Cost-efficient vnf placement and scheduling in public cloud networks. *IEEE Transactions on Communications*, 68(8):4946–4959, 2020.

[33] Danyang Zheng, Chengzong Peng, Xueting Liao, and Xiaojun Cao. Toward optimal hybrid service function chain embedding in multiaccess edge computing. *IEEE Internet of Things Journal*, 7(7):6035–6045, 2020.

[34] Hui Yu, Jiahai Yang, and Carol Fung. Fine-grained cloud resource provisioning for virtual network function. *IEEE Transactions on Network and Service Management*, 17(3):1363–1376, 2020.

[35] Yimin Li, Lifang Gao, Siya Xu, Qinghai Ou, Xinyu Yuan, Feng Qi, Shaoyong Guo, and Xuesong Qiu. Cost-and-qos-based nfv service function chain mapping mechanism. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, 2020.

[36] Lin Cui, Fung Po Tso, Song Guo, Weijia Jia, Kaimin Wei, and Wei Zhao. Enabling Heterogeneous Network Function Chaining. *IEEE Transactions on Parallel and Distributed Systems*, 30(4):842–854, apr 2019.

[37] Zhi Zhou, Qiong Wu, and Xu Chen. Online Orchestration of Cross-Edge Service Function Chaining for Cost-Efficient Edge Computing. *IEEE Journal on Selected Areas in Communications*, 37(8):1866–1880, aug 2019.

[38] Behrooz Farkiani, Bahador Bakhshi, S. Ali MirHassani, Tim Wauters, Bruno Volckaert, and Filip De Turck. Prioritized deployment of dynamic service function chains. *IEEE/ACM Transactions on Networking*, 29(3):979–993, 2021.

[39] Vincenzo Eramo and Francesco Giacinto Lavacca. Optimizing the Cloud Resources, Bandwidth and Deployment Costs in Multi-Providers Network Function Virtualization Environment. *IEEE Access*, 7:46898–46916, 2019.

[40] Francesco Malandrino, Carla Fabiana Chiasserini, Gil Einziger, and Gabriel Scalosub. Reducing Service Deployment Cost Through VNF Sharing. *IEEE/ACM Transactions on Networking*, pages 1–14, oct 2019.

[41] Gang Sun, Gungyang Zhu, Dan Liao, Hongfang Yu, Xiaojiang Du, and Mohsen Guizani. Cost-Efficient Service Function Chain Orchestration for Low-Latency Applications in NFV Networks, 2018.

[42] jianing Pei, Peilin Hong, Kaiping Xue, and Defang Li. Efficiently Embedding Service Function Chains with Dynamic Virtual Network Function Placement in Geo-distributed Cloud System. *IEEE Transactions on Parallel and Distributed Systems*, 30(10):2179–2192, 2018.

[43] Milad Ghaznavi, Nashid Shahriar, Shahin Kamali, Reaz Ahmed, and Raouf Boutaba. Distributed service function chaining. *IEEE Journal on Selected Areas in Communications*, 35(11):2479–2489, 2017.

[44] Md Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, Raouf Boutaba, and Otto Carlos Muniz Bandeira Duarte. Orchestrating Virtualized Network Functions. *IEEE Transactions on Network and Service Management*, 13(4):725–739, dec 2016.

[45] Rami Cohen, Liane Lewin-Eytan, Joseph Seffi Naor, and Danny Raz. Near optimal placement of virtual network functions. In *Proceedings - IEEE INFOCOM*, volume 26, pages 1346–1354. Institute of Electrical and Electronics Engineers Inc., aug 2015.

[46] M. Gharbaoui, C. Contoli, G. Davoli, D. Borsatti, G. Cuffaro, F. Paganelli, W. Cerroni, P. Cappanera, and B. Martini. An experimental study on latency-aware and self-adaptive service chaining orchestration in distributed nfv and sdn infrastructures. *Computer Networks*, 208:108880, 2022.

[47] Yicen Liu, Yu Lu, Xi Li, Zhiwei Li, Wenxin Qiao, Yang Zhang, and Donghao Zhao. A lagrangian-relaxation-based approach for service function chain

dynamic orchestration for the internet of things. *IEEE Internet of Things Journal*, 8(23):17071–17089, 2021.

[48] Suman Pandey, Tu Van Nguyen, Jae-Hyoung Yoo, and James Won-Ki Hong. Edgedqn: Multiple sfc placement in edge computing environment. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 301–309, 2021.

[49] Lina Magoula, Sokratis Barmpounakis, Ioannis Stavrakakis, and Nancy Alonistioti. A genetic algorithm approach for service function chain placement in 5g and beyond, virtualized edge networks. *Computer Networks*, 195:108157, 2021.

[50] Wei Li, Haiyang Wu, Chunxia Jiang, Ping Jia, Naling Li, and Peng Lin. Service chain mapping algorithm based on reinforcement learning. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 800–805, 2020.

[51] Lei Wang, Mahdi Dolati, and Majid Ghaderi. Change: Delay-aware service function chain orchestration at the edge. In *2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC)*, pages 19–28, 2021.

[52] Ruijie Wang, Junhuai Li, Kan Wang, Xuan Liu, and Xuan Lit. Service function chaining in nfv-enabled edge networks with natural actor-critic deep reinforcement learning. In *2021 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 1095–1100, 2021.

[53] Masahiro Sasabe and Takanori Hara. Capacitated shortest path tour problem-based integer linear programming for service chaining and function placement in nfv networks. *IEEE Transactions on Network and Service Management*, 18(1):104–117, 2021.

[54] Yu Liu, Xiaojun Shang, and Yuanyuan Yang. Joint sfc deployment and resource management in heterogeneous edge for latency minimization. *IEEE Transactions on Parallel and Distributed Systems*, 32(8):2131–2143, 2021.

[55] Yicen Liu, Hao Lu, Xi Li, Yang Zhang, Leiping Xi, and Donghao Zhao. Dynamic service function chain orchestration for nfv/mec-enabled iot networks: A deep reinforcement learning approach. *IEEE Internet of Things Journal*, 8(9):7450–7465, 2021.

[56] Sihao Xie, Junte Ma, and Jin Zhao. Flexchain: Bridging parallelism and placement for service function chains. *IEEE Transactions on Network and Service Management*, 18(1):195–208, 2021.

[57] Stefan Schneider, Haydar Qarawlus, and Holger Karl. Distributed online service coordination using deep reinforcement learning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 539–549, 2021.

[58] I-Chieh Lin, Yu-Hsuan Yeh, and Kate Ching-Ju Lin. Toward optimal partial parallelization for service function chaining. *IEEE/ACM Transactions on Networking*, 29(5):2033–2044, 2021.

[59] Boutheina Dab, Ilhem Fajjari, Mathieu Rohon, Cyril Auboin, and Arnaud Diquélou. Cloud-native service function chaining for 5g based on network service mesh. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–7, 2020.

[60] An Xie, Huawei Huang, Xiaoliang Wang, Zhuzhong Qian, and Sanglu Lu. Online vnf chain deployment on resource-limited edges by exploiting peer edge devices. *Computer Networks*, 170:107069, 2020.

[61] Marcel Blöcher, Ramin Khalili, Lin Wang, and Patrick Eugster. Letting off steam: Distributed runtime traffic scheduling for service function chaining. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 824–833, 2020.

[62] Panpan Jin, Xincai Fei, Qixia Zhang, Fangming Liu, and Bo Li. Latency-aware vnf chain deployment with efficient resource reuse at network edge. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 267–276, 2020.

[63] 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 *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 1539–1548, 2020.

[64] Boutheina Dab, Ilhem Fajjari, Mathieu Rohon, Cyril Auboin, and Arnaud Diquélou. An efficient traffic steering for cloud-native service function chaining. In *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 71–78, 2020.

[65] Bin Gao, Zhi Zhou, Fangming Liu, and Fei Xu. Winning at the starting line: Joint network selection and service placement for mobile edge computing. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1459–1467, 2019.

[66] Konstantinos Poularakis, Jaime Llorca, Antonia M. Tulino, Ian Taylor, and Leandros Tassiulas. Joint service placement and request routing in multi-cell mobile edge computing networks. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 10–18, 2019.

[67] Song Yang, Fan Li, Ramin Yahyapour, and Xiaoming Fu. Delay-Sensitive and Availability-Aware Virtual Network Function Scheduling for NFV. *IEEE Transactions on Services Computing*, 2019.

[68] Dmitrii Chemodanov, Prasad Calyam, and Flavio Esposito. A Near Optimal Reliable Composition Approach for Geo-Distributed Latency-Sensitive Service Chains. *Proceedings - IEEE INFOCOM*, 2019-April:1792–1800, 2019.

[69] Gang Sun, Yayu Li, Yao Li, Dan Liao, and Victor Chang. Low-latency orchestration for workflow-oriented service function chain in edge computing. *Future Generation Computer Systems*, 85:116–128, aug 2018.

[70] Richard Cziva, Christos Anagnostopoulos, and Dimitrios P. Pezaros. Dynamic, Latency-Optimal vNF Placement at the Network Edge. In *Proceedings - IEEE INFOCOM*, volume 2018-April, pages 693–701. Institute of Electrical and Electronics Engineers Inc., oct 2018.

[71] Xiaojun Shang, Yaodong Huang, Zhenhua Liu, and Yuanyuan Yang. Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme. *IEEE Transactions on Mobile Computing*, pages 1–1, 2021.

[72] Abdelhamid Alleg, Toufik Ahmed, Mohamed Mosbah, and Raouf Boutaba. Joint diversity and redundancy for resilient service chain provisioning. *IEEE Journal on Selected Areas in Communications*, 38(7):1490–1504, 2020.

[73] Hua Qu, Ke Wang, and Jihong Zhao. Reliable service function chain deployment method based on deep reinforcement learning. *Sensors*, 21(8), 2021.

[74] Nazli Siasi, Adrian Jaesim, Aysegül Yayimli, and Nasir Ghani. Service function chain survivability provisioning in fog networks. *IEEE Transactions on Network and Service Management*, pages 1–1, 2021.

[75] Anna Engelmann and Admela Jukan. A combinatorial reliability analysis of generic service function chains in data center networks. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 6(3), dec 2021.

[76] Yang Chen and Jie Wu. Latency-efficient vnf deployment and path routing for reliable service chain. *IEEE Transactions on Network Science and Engineering*, 8(1):651–661, 2021.

[77] Chen Wang, Qin Hu, Dongxiao Yu, and Xiuzhen Cheng. Proactive deployment of chain-based vnf backup at the edge using online bandit learning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 740–750, 2021.

[78] Prabhu Kaliyammal Thiruvasagam, Abhishek Chakraborty, Abin Mathew, and C. Siva Ram Murthy. Reliable placement of service function chains and virtual monitoring functions with minimal cost in softwarized 5g networks. *IEEE Transactions on Network and Service Management*, 18(2):1491–1507, 2021.

[79] Junzhong Jia, Lei Yang, and Jiannong Cao. Reliability-aware dynamic service chain scheduling in 5g networks based on reinforcement learning. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.

[80] Sidharth Sharma, Ashwin Gumaste, and Mallik Tatipamula. High-availability service chain realization theory. In *2020 16th International Conference on the Design of Reliable Communication Networks DRCN 2020*, pages 1–8, 2020.

[81] Meng Wang, Bo Cheng, and Junliang Chen. Joint availability- and traffic-aware placement of parallelized service chain in nfv-enabled data center. In *2020 IEEE International Conference on Web Services (ICWS)*, pages 216–223, 2020.

[82] Arunabha Sen, Sandipan Choudhuri, and Kaustav Basu. Structural dependency aware service chain mapping for network function virtualization. In *2020 16th International Conference on the Design of Reliable Communication Networks DRCN 2020*, pages 1–6, 2020.

[83] Xiaohan Yin, Bo Cheng, Meng Wang, and Junliang Chen. Availability-aware service function chain placement in mobile edge computing. In *2020 IEEE World Congress on Services (SERVICES)*, pages 69–74, 2020.

[84] Nguyen Huu Thanh, Nguyen Trung Kien, Ngo Van Hoa, Truong Thu Huong, Florian Wamser, and Tobias Hossfeld. Energy-aware service function chain embedding in edge–cloud environments for iot applications. *IEEE Internet of Things Journal*, 8(17):13465–13486, 2021.

[85] Fahime Khoramnejad, Roghayeh Joda, and Melike Erol-Kantarci. Distributed multi-agent learning for service function chain partial offloading at the edge. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2021.

[86] Gang Sun, Run Zhou, Jian Sun, Hongfang Yu, and Athanasios V. Vasilakos. Energy-efficient provisioning for service function chains to support delay-sensitive applications in network function virtualization. *IEEE Internet of Things Journal*, 7(7):6116–6131, 2020.

[87] Zhi Zhou, Fangming Liu, Shutong Chen, and Zongpeng Li. A truthful and efficient incentive mechanism for demand response in green datacenters. *IEEE Transactions on Parallel and Distributed Systems*, 31(1):1–15, 2020.

[88] Gang Sun, Yayu Li, Hongfang Yu, Athanasios V. Vasilakos, Xiaojiang Du, and Mohsen Guizani. Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks. *Future Generation Computer Systems*, 91:347–360, feb 2019.

[89] Md. Faizul Bari, Shihabur Rahman Chowdhury, and Raouf Boutaba. ESSO: An Energy Smart Service Function Chain Orchestrator. *IEEE Transactions on Network and Service Management*, pages 1–1, sep 2019.

[90] Anwesha Mukherjee, Debashis De, and Deepsubhra Guha Roy. A Power and Latency Aware Cloudlet Selection Strategy for Multi-Cloudlet Environment. *IEEE Transactions on Cloud Computing*, 7(1):141–154, jan 2019.

[91] Mohammad M. Tajiki, Stefano Salsano, Luca Chiaraviglio, Mohammad Shojafar, and Behzad Akbari. Joint Energy Efficient and QoS-Aware Path Allocation and VNF Placement for Service Function Chaining. *IEEE Transactions on Network and Service Management*, 16(1):374–388, 2019.

[92] Insun Jang, Dongeun Suh, Sangheon Pack, and György Dán. Joint optimization of service function placement and flow distribution for service function chaining. *IEEE Journal on Selected Areas in Communications*, 35(11):2532–2541, 2017.

[93] M. Li, Q. Zhang, and F. Liu. Finedge: A dynamic cost-efficient edge resource management platform for nfv network. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pages 1–10, 2020.

[94] Shuting Le, Yuhu Wu, and Mitsuru Toyoda. A congestion game framework for service chain composition in nfv with function benefit. *Information Sciences*, 514:512–522, 2020.

[95] Yicen Liu, Hao Lu, Xi Li, Donghao Zhao, Weiyi Wu, and Ganqiang Lu. A novel approach for service function chain dynamic orchestration in edge clouds. *IEEE Communications Letters*, 24(10):2231–2235, 2020.

[96] Jungmin Son and Rajkumar Buyya. Latency-aware virtualized network function provisioning for distributed edge clouds. *Journal of Systems and Software*, 152:24–31, 2019.

[97] Deval Bhamare, Mohammed Samaka, Aiman Erbad, Raj Jain, Lav Gupta, and H. Anthony Chan. Optimal virtual network function placement in multi-cloud service function chaining architecture. *Computer Communications*, 102:1–16, 2017.

[98] Paola Cappanera, Federica Paganelli, and Francesca Paradiso. VNF placement for service chaining in a distributed cloud environment with multiple stakeholders. *Computer Communications*, 133:24–40, jan 2019.

[99] Qi Xu, Deyun Gao, Taixin Li, and Hongke Zhang. Low latency security function chain embedding across multiple domains. *IEEE Access*, 6:14474–14484, 2018.

[100] Kalpana D. Joshi and Kotaro Kataoka. psmart: A lightweight, privacy-aware service function chain orchestration in multi-domain nfv/sdn. *Computer Networks*, 178:107295, 2020.

[101] Kai Peng, Jiangtian Nie, Neeraj Kumar, Chao Cai, Jiawen Kang, Zehui Xiong, and Yang Zhang. Joint optimization of service chain caching and task offloading in mobile edge computing. *Applied Soft Computing*, 103:107142, 2021.

[102] Amit Samanta and Zheng Chang. Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint. *IEEE Internet of Things Journal*, 6(2):3864–3872, 2019.

[103] Xuxia Zhong, Ying Wang, and Xuesong Qiu. Cost-aware service function chaining with reliability guarantees in nfv-enabled inter-dc network. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 304–311, 2019.

[104] Chuangchuang Zhang, Xingwei Wang, Yong Zhao, Anwei Dong, Fuliang Li, and Min Huang. Cost Efficient and Low-Latency Network Service Chain Deployment across Multiple Domains for SDN. *IEEE Access*, 7:143454–143470, 2019.

[105] Huan Chen, Xiong Wang, Yangming Zhao, Tongyu Song, Yang Wang, Shizhong Xu, and Lemin Li. MOSC: a method to assign the outsourcing of service function chain across multiple clouds. *Computer Networks*, 133:166–182, mar 2018.

[106] David Dietrich, Ahmed Abujoda, Amr Rizk, and Panagiotis Papadimitriou. Multi-Provider Service Chain Embedding With Nestor. *IEEE Transactions on Network and Service Management*, 14(1):91–105, mar 2017.

[107] Yexiao He, Xiaoning Zhang, Zixiang Xia, Yutao Liu, Keshav Sood, and Shui Yu. Joint optimization of service chain graph design and mapping in nfv-enabled networks. *Computer Networks*, 202:108626, 2022.

[108] Qiao Xiang, X. Tony Wang, J. Jensen Zhang, Harvey Newman, Y. Richard Yang, and Y. Jace Liu. Unicorn: Unified resource orchestration for multi-domain, geo-distributed data analytics. *Future Generation Computer Systems*, 93:188–197, 2019.

[109] Abhishek Gupta, Brigitte Jaumard, Massimo Tornatore, and Biswanath Mukherjee. A Scalable Approach for Service Chain Mapping with Multiple SC Instances in a Wide-Area Network. In *IEEE Journal on Selected Areas in Communications*, volume 36, pages 529–541. Institute of Electrical and Electronics Engineers Inc., mar 2018.

[110] Antonio Francescon, Giovanni Baggio, Riccardo Fedrizzi, Ramon Ferrusy, Imen Grida Ben Yahiaz, and Roberto Riggio. X–mano: Cross–domain management and orchestration of network services. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5, 2017.

[111] Meng Niu, Qingmian Han, Bo Cheng, Meng Wang, Ziqi Xu, Wenyuan Gu, Shuhao Zhang, and Junliang Chen. Hars: A high-available and resource-saving service function chain placement approach in data center networks. *IEEE Transactions on Network and Service Management*, pages 1–1, 2022.

[112] Marwa A. Abdelaal, Gamal A. Ebrahim, and Wagdy R. Anis. Efficient placement of service function chains in cloud computing environments. *Electronics*, 10(3), 2021.

[113] Flavio Esposito. Catena: A distributed architecture for robust service function chain instantiation with guarantees. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–9, 2017.

[114] Hongfang Yu, Zhenrong Chen, Gang Sun, Xiaojiang Du, and Mohsen Guizani. Profit maximization of online service function chain orchestration in an inter-datacenter elastic optical network. *IEEE Transactions on Network and Service Management*, 18(1):973–985, 2021.

[115] Zichuan Xu, Lizhen Zhou, Sid Chi-Kin Chau, Weifa Liang, Qiufen Xia, and Pan Zhou. Collaborate or separate? distributed service caching in mobile edge clouds. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 2066–2075, 2020.

[116] Zhaolong Ning, Peiran Dong, Xiaojie Wang, Shupeng Wang, Xiping Hu, Song Guo, Tie Qiu, Bin Hu, and Ricky Y. K. Kwok. Distributed and dynamic service placement in pervasive edge computing networks. *IEEE Transactions on Parallel and Distributed Systems*, 32(6):1277–1292, 2021.

[117] Handi Chen, Shupeng Wang, Guojun Li, Laisen Nie, Xiaojie Wang, and Zhaolong Ning. Distributed orchestration of service function chains for edge intelligence in the industrial internet of things. *IEEE Transactions on Industrial Informatics*, pages 1–1, 2021.

[118] Yi Liu, Hongqi Zhang, Dexian Chang, and Hao Hu. Gdm: A general distributed method for cross-domain service function chain embedding. *IEEE Transactions on Network and Service Management*, 17(3):1446–1459, 2020.

[119] Alexandre Huff, Giovanni Venâncio, Vinícius Fulber Garcia, and Elias P. Duarte. Building multi-domain service function chains based on multiple nfv orchestrators. In *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 19–24, 2020.

[120] Qiong Zhang, Xi Wang, Inwoong Kim, Paparao Palacharla, and Tadashi Ikeuchi. Vertex-centric computation of service function chains in multi-domain networks. In *IEEE NETSOFT 2016 - 2016 IEEE NetSoft Conference and Workshops: Software-Defined Infrastructure for Networks, Clouds, IoT and Services*, pages 211–218. Institute of Electrical and Electronics Engineers Inc., jun 2016.

[121] Chuan Pham, Duong Tuan Nguyen, Nguyen H. Tran, Kim Khoa Nguyen, and Mohamed Cheriet. Optimized iot service chain implementation in edge cloud platform: A deep learning framework. *IEEE Transactions on Network and Service Management*, 18(1):538–551, 2021.

[122] Chunlin Li, Jun Liu, Bo Lu, and Youlong Luo. Cost-aware automatic scaling and workload-aware replica management for edge-cloud environment. *Journal of Network and Computer Applications*, 180:103017, 2021.

[123] Xin Zhao, Xuan Jia, and Yanpei Hua. An efficient vnf deployment algorithm for sfc scaling-out based on the proposed scaling management mechanism. In *2020 Information Communication Technologies Conference (ICTC)*, pages 166–170, 2020.

[124] Riming Tong, Siya Xu, Bo Hu, Jinghong Zhao, Lei Jin, Shaoyong Guo, and Wenjing Li. Vnf dynamic scaling and deployment algorithm based on traffic prediction. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 789–794, 2020.

[125] Xiang Chen, Yuxin Chen, Qun Huang, Haifeng Zhou, Dong Zhang, Chunming Wu, and Junchi Xing. Fastscale: Fast scaling out of network functions. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 436–442, 2020.

[126] Ziyue Luo and Chuan Wu. An online algorithm for vnf service chain scaling in datacenters. *IEEE/ACM Transactions on Networking*, 28(3):1061–1073, 2020.

[127] Zenan Wang, Jiao Zhang, Haoran Wei, and Tao Huang. Hieff: Enabling efficient vnf clusters by coordinating vnf scaling and flow scheduling. In *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, 2020.

[128] Yifu Yao, Songtao Guo, Pan Li, Guiyan Liu, and Yue Zeng. Forecasting assisted vnf scaling in nfv-enabled networks. *Computer Networks*, 168:107040, 2020.

[129] Tejas Subramanya, Davit Harutyunyan, and Roberto Riggio. Machine learning-driven service function chain placement and scaling in mec-enabled 5g networks. *Computer Networks*, 166:106980, 2020.

[130] Hong Tang, Danny Zhou, and Duan Chen. Dynamic network function instance scaling based on traffic forecasting and vnf placement in operator data centers. *IEEE Transactions on Parallel and Distributed Systems*, 30(3):530–543, 2019.

[131] Yashwant Singh Patel, Deepak Verma, and Rajiv Misra. Deep learning based resource allocation for auto-scaling vnfs. In *2019 IEEE International Con-*

ference on Advanced Networks and Telecommunications Systems (ANTS), pages 1–6, 2019.

[132] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. Adaptive vnf scaling and flow routing with proactive demand prediction. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 486–494, 2018.

[133] Fangxin Wang, Ruilin Ling, Jing Zhu, and Dan Li. Bandwidth guaranteed virtual network function placement and scaling in datacenter networks. In *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, 2015.

[134] Nan Wang, Michail Matthaiou, Dimitrios S. Nikolopoulos, and Blesson Varghese. Dyverse: Dynamic vertical scaling in multi-tenant edge environments. *Future Generation Computer Systems*, 108:598–612, 2020.

[135] Jiachen Zu, Guyu Hu, Yang Wu, Dongsheng Shao, and Jiajie Yan. Resource aware chaining and adaptive capacity scaling for service function chains in distributed cloud network. *IEEE Access*, 7:157707–157723, 2019.

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

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

[138] Yongzheng Jia, Chuan Wu, Zongpeng Li, Franck Le, and Alex Liu. Online scaling of nfv service chains across geo-distributed datacenters. *IEEE/ACM Transactions on Networking*, 26(2):699–710, 2018.

[139] Milad Ghaznavi, Aimal Khan, Nashid Shahriar, Khalid Alsubhi, Reaz Ahmed, and Raouf Boutaba. Elastic virtual network function placement. In *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, pages 255–260, 2015.

[140] Tejas Subramanya and Roberto Riggio. Centralized and federated learning for predictive vnf autoscaling in multi-domain 5g networks and beyond. *IEEE Transactions on Network and Service Management*, 18(1):63–78, 2021.

[141] Dong Zhai, Xiangru Meng, Zhenhua Yu, Hang Hu, and Xiaoyang Han. A fine-grained and dynamic scaling method for service function chains. *Knowledge-Based Systems*, 228:107289, 2021.

[142] Sabidur Rahman, Tanjila Ahmed, Minh Huynh, Massimo Tornatore, and Biswanath Mukherjee. Auto-scaling network service chains using machine learning and negotiation game. *IEEE Transactions on Network and Service Management*, 17(3):1322–1336, 2020.

[143] Omar Houidi, Oussama Soualah, Wajdi Louati, Marouen Mechtri, Djamal Zeghlache, and Farouk Kamoun. An efficient algorithm for virtual network function scaling. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–7, 2017.

[144] D. P. Bertsekas and R. G. Gallager. *Data Networks, 2nd ed.* Upper Saddle River, NJ, USA: Prentice-Hall, 1992.

[145] Chuangchuang Zhang, Xingwei Wang, Anwei Dong, Yong Zhao, Qiang He, and Min Huang. Energy efficient network service deployment across multiple sdn domains. *Computer Communications*, 151:449–462, 2020.

[146] U. Pferschy H. Kellerer and D. Pisinger. Knapsack problems. *Berlin, Germany: Springer*, 2004.

[147] Whay C. Lee. Topology aggregation for hierarchical routing in ATM networks. *Computer Communication Review*, 25(2):82–92, 1995.

[148] Jin Y. Yen. Finding the K Shortest Loopless Paths in a Network . *Management Science*, 17(11):712–716, jul 1971.

[149] Ahmed Abujoda and Panagiotis Papadimitriou. DistNSE: Distributed network service embedding across multiple providers. In *2016 8th International Conference on Communication Systems and Networks, COMSNETS 2016*. Institute of Electrical and Electronics Engineers Inc., mar 2016.

[150] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[151] Amazon. Ec2 on-demand instance pricing – amazon web services. https://aws.amazon.com/ec2/pricing/on-demand/, 2021.

[152] Chuan Pham, Nguyen H. Tran, Shaolei Ren, Walid Saad, and Choong Seon Hong. Traffic-aware and Energy-efficient vNF Placement for Service Chaining: Joint Sampling and Matching Approach. *IEEE Transactions on Services Computing*, pages 1–1, feb 2017.

[153] P. Laarhoven and E. Aarts. Simulated annealing. In *Simulated Annealing: Theory and Applications*. Springer Netherlands, 1987.

[154] Deval Bhamare, Aiman Erbad, Raj Jain, Maede Zolanvari, and Mohammed Samaka. Efficient virtual network function placement strategies for Cloud Radio Access Networks. *Computer Communications*, 127:50–60, sep 2018.

[155] CAIDA. The caida ucsd passive-2019.

[156] Wenrui Ma, Oscar Sandoval, Jonathan Beltran, Deng Pan, and Niki Pissinou. Traffic aware placement of interdependent nfv middleboxes. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017.

[157] Gurobi. Gurobi optimizer reference manual, 2021.