Beyond 5Gネットワークにおけるネットワーク機能仮想化の動的スケーリングとリアルタイムスケジューリング

Dynamic Scaling and Real-Time Scheduling in Network Function Virtualization for Beyond 5G Network

ZHENKE CHEN

Department of Sciences and Informatics
Muroran Institute of Technology

This dissertation is submitted for the degree of
Doctor of Philosophy of Engineering

February 5th, 2024

Declaration

I hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any award. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature, and acknowledgement of collaborative research and discussions.

ZHENKE CHEN
February 5th, 2024

Acknowledgements

Firstly, I sincerely appreciate my supervisor, Professor He Li. He unwaveringly supported and taught me throughout my Ph.D. time. Without his mentorship, I would not have been able to advance my research so significantly or acquire such cutting-edge knowledge in the field.

Besides my supervisor, I would like to thank my predecessor supervisor, Professor Kaoru Ota. Her unwavering kindness and warm smile, even in moments when I made significant mistakes, have left a deep impression on me. From her, I have learned the invaluable lesson of always treating everyone I encounter with kindness and respect.

I am also deeply grateful to Professor Mianxiong Dong, the director of our Emerging Networks and Systems Laboratory (ENeS) for his invaluable support in both my scientific research and daily living. Without his support on my scholarship application, I would not have been able to gain a decent income during my Ph.D. career. Finally, I would like to thank Professor Tomohiko Taniguchi for joining my Ph.D. dissertation defense and sharing his invaluable suggestions.

ZHENKE CHEN
February 5th, 2024

Abstract

Along with Network Function Virtualization (NFV), Mobile Network Operator started to build the fifth generation (5G) networks while enhancing network flexibility, agility, scalability, and cost-efficiency. However, 5G technology is predicted to be unable to adapt to the huge data traffic generated by massive IoT devices in the future beyond 5G (B5G) and the future 6G era. This has raised the tremendous challenge of upgrading the existing 5G network to adapt to the exponential growth in data demands and stricter requirements of user experience.

To make the 5G network adapt to forthcoming B5G/6G traffic and improve the Quality of Experience (QoE) for upcoming B5G/6G applications, in this dissertation, the author developed three research. First, the author developed a system for scaling the network function capacity to handle massive traffic from B5G/6G applications. Second, the author ensures the Quality of Service (QoS) for 5G applications. The solutions aim to improve the QoS of 5G applications, such as reducing delay and ensuring sufficient throughput for different 5G services. Finally, the author optimized the QoE for real-time B5G/6G applications like Digital twin (DT) and Virtual/Augmented Reality (VR/AR). The author uses a metric called Age of Information (AoI) to measure QoE in B5G/6G applications and solve the traffic routing problem to ensure QoE.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Background

1.1.1 The Development of Mobile Communication

The evolution of mobile communication spans multiple generations, each marking significant advancements in technology and society. Starting with 1G in the 1980s, which introduced analog voice communication, it progressed to 2G in the 1990s, enabling digital voice and text messaging. The 2000s saw 3G, offering mobile internet and data services, followed by 4G in the 2010s, which revolutionized connectivity with high-speed broadband for streaming and applications.

With the trend of 4G reaching its capacity limits due to increasing demands from users and emerging technologies, the development of enhanced Mobile Broadband (eMBB) for faster speeds and higher capacity, massive Internet of Things (mIoT) connectivity to support billions of interconnected devices simultaneously, and Ultra-Reliable Low-Latency Communication (URLLC) to enable real-time applications, is becoming a necessity. To address this situation, the fifth generation (5G) emerged to deliver ultra-low latency, massive device connectivity, and unprecedented speeds. For example, 5G enables faster internet speeds, supporting seamless video streaming and high-quality video calls. 5G also powers smart cities by connecting Internet of Things (IoT) devices, such as traffic lights, sensors, and surveillance cameras, ensuring efficient urban management.

According to an Ericsson Mobility Report [30], global 5G devices will grow exponentially, reaching to 6.3 billion by 2030. Furthermore, IoT-driven applications, including autonomous vehicles, remote robotic surgery, digital twins, and immersive extended reality (XR), are experiencing a sharp rise in demand for lower latency, increased data rates and higher

reliability [33]. Current 5G systems are considered to be inadequate for future trends, which appeals to create a next-generation mobile communication standard.

1.1.2 Network Function Virtualization (NFV)

From the era of 1G to 4G, the mobile telecommunications network was based on physical middlebox infrastructure. Traditionally, Mobile Network Operators (MNOs) relied on dedicated hardware middleboxes to implement specific network functions. Fig. 1.1 shows a physical 4G Evolved Packet Core (EPC) network. Network functions such as Packet Data Network Gateway (PGW), Home Subscriber Server (HSS), switch, Mobility Management Entity (MME) and Online charging system (OCS) are running on different dedicated hardware and perform various functionality.

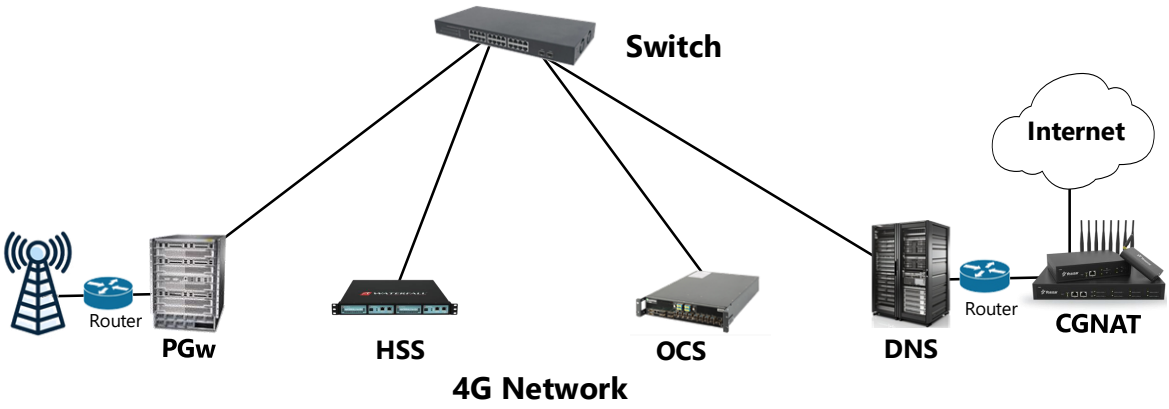

Fig. 1.1 4G Evolved Packet Core (EPC) Network

In the 5G era, more 5G connections bring more diversified user requirements and more automatic control. Traditional middlebox-based network architecture was no longer suitable for the development of 5G due to its lack of flexibility, scalability, and efficiency. As 5G networks require dynamic resource allocation, ultra-low latency, and the ability to support diverse services and massive device connectivity, the static and hardware-dependent nature of middlebox-based architectures fails to meet these demands. Additionally, the high operational costs and limited adaptability of traditional architectures hinder their ability to support the rapid innovation and deployment cycles required for 5G.

Recently, Network Function Virtualization (NFV) [14] was proposed to solve the drawback of traditional middlebox-based networks. The key idea behind NFV is the integration of virtualization technology into network functions. Different from traditional middleboxes, NFV softwarizes network functions so that run them as Virtual Network Functions (VNF) on commodity hardware, shown in Fig. 1.2. As compared with traditional middleboxes,

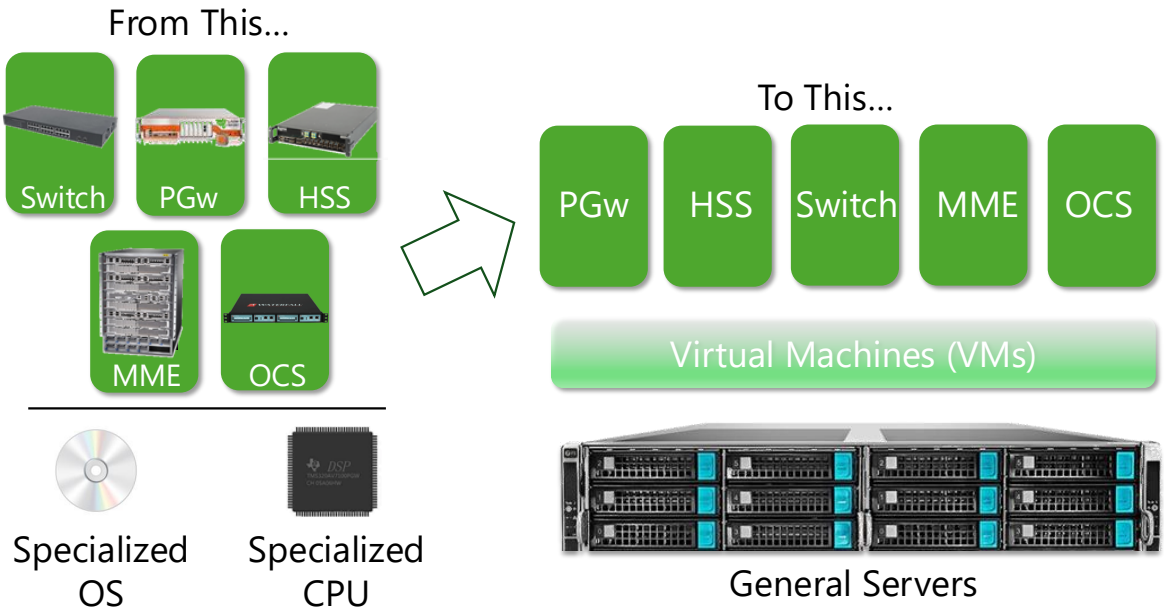

Fig. 1.2 Network Function Virtualization (NFV)

NFV brings benefits as follows: (1) agility and scalability: NFV enables MMOs to deploy and scale network functions as virtualized instances rapidly. With the ability to spin up or down VNFs on demand, MMOs can adapt to changing network requirements quickly and efficiently; (2) cost efficiency: By eliminating the need for dedicated hardware appliances, NFV reduces capital expenditure and lowers operational costs. VNFs can be run on standard servers, reducing hardware costs, power consumption, and maintenance efforts; (3) network optimization: With NFV, hardware resources are virtualized into virtual resources that can be flexibly allocated based on demand. This flexibility improves network performance, enhances Quality of Service (QoS), and optimizes resource utilization.

In summary, NFV facilitates service provision and makes networks more flexible, agile and cost-efficient. Leveraging these benefits, MMOs have adopted this technology in building 5G core networks and EPC networks to provide users with the requested 5G network service.

1.1.3 5G, Beyond 5G and 6G

Recently, many MMOs started to deploy 5G networks. Several MMOs, e.g., Rakuten, Softbank, AT&T, Vodafone and China Mobile explore the opportunities that 5G development would induce new business models and user requirements. The 5G feature is classified into three categories, each driving new tendency for use cases:

- Ultra-Reliable Low Latency Communication (URLLC): URLLC provides low-latency and highly reliable communication for real-time applications. For example, smart

industries use URLLC to monitor the environment of factories and the status of machines to keep workers safe and improve production.

- enhanced Mobile Broadband (eMBB): eMBB is a 5G service that delivers high-capacity broadband with speeds of up to 1 Gbps and high reliability. For example, eMBB supports 3D 8K Virtual Reality (VR) at frame rates of 90 to 120 frames per second.

- massive Machine Type Communication (mMTC): mMTC empowers the network with the ability of massive device connectivity. With mMTC, a 5G network supports millions of simultaneous IoT connections while maintaining a low latency of around 50 ms.

As 5G communication is predicted to be unable to catch up with increasing data traffic generated by massive IoT devices in the future beyond 5G (B5G) and the future 6G era, academics and industry are paying more attention to the next generation communication standard.

Compared with 5G, B5G/6G further enhances connectivity by integrating advanced technologies like artificial intelligence, quantum computing, and terahertz communication. It aims to provide ultra-high-speed data transfer, seamless global coverage, and support for intelligent networked systems. This next-generation network is expected to enable revolutionary applications such as holographic communication, immersive XR/VR/AR, and autonomous systems, pushing the boundaries of innovation and transforming industries worldwide.

1.2 Challenge

The 6G network is expected to support more advanced applications. However, to support these applications, MMOs are facing three main challenges in the current 5G NFV network.

1.2.1 Large 6G Traffic

6G traffic is significantly greater than 5G due to its support for advanced applications and services. For example, holographic communication in B5G/6G requires data rates of several terabits per second (tbps) to transmit 3D images in real-time, far exceeding the bandwidth demands of 5G applications like 4K video streaming.

While the maximum throughput of 5G eMBB reaches up to 20 Gbps, 5G still falls short of meeting the substantial traffic demands up to tbps level from 6G applications. With the advent

of B5G and the anticipated 6G traffic, our experiment shows that the hardware resources of a single NFV server are not adequate to handle huge traffic from B5G/6G applications. An upgrade to increase network capacity and support B5G/6G application is essential for MMOs to consider.

1.2.2 Quality of Service Guarantee

5G NFV network is constructed of servers, each server has certain resources to execute multiple VNFs simultaneously. In the 5G NFV network, a network service is implemented as a service chain consisting of several VNFs that are chained together.

MMO must strategically place VNFs onto the Resource-constrained servers and route traffic to pass through a set of VNFs in the pre-defined order. By optimizing VNF-to-node mapping and traffic path, MMOs can significantly reduce operational/capital costs and enhance network services performance. However, determining server selection and managing traffic path is challenging, as multiple solutions are available and each brings distinct performances. Adding to this complexity, 5G services come with varying requirements for latency, throughput, and reliability, making the problem even more difficult to address. An efficient VNF placement approach is important and indispensable to achieve better overall performance of the 5G network.

1.2.3 Stricter User Experience

To ensure better user experience, stricter Quality of Experience (QoE) standards are necessary to support B5G/6G applications. For example, AR and VR in 6G require lower latency to ensure that digital elements are seamlessly integrated into the real world, providing a more natural and convincing experience. In Digital Twin (DT), it needs real-time updates from the real world to create a digital replica of the physical world. A small lag

However, 5G features (i.e., URLLC, eMBB and mMTC) are insufficient to support the critical requirement of maintaining data freshness for those applications. Therefore, how to optimize the QoE for those applications in the 5G NFV network has also become a critical challenge.

In short, those applications not only produce massive data traffic but also require stricter Quality of Experience (QoE). Toward B5G/6G, MNOs face significant challenges in adapting to the exponential growth in data demands and stricter QoE.

1.3 System Outline

In this dissertation, the author proposes a system to solve the above challenge so that makes the 5G network adapt to forthcoming B5G/6G traffic and improves QoE for upcoming B5G/6G applications.

First, in the precious research, NFV was proposed for building flexible network management in the data center network. In comparison with them, my research focuses on mobile networks where users have stricter requirements on performance. When NFV comes to mobile networks, I also consider user mobility. Second, my research is the first attempt to apply NFV to adapt to the trend of beyond 5G (B5G) and the future 6G area. In the B5G/6G area, user traffic extremely increases to the tb/s level. Furthermore, users require a better user experience than the 5G area. To adapt to these trends, my research scales the network function capacity and uses a new metric called Age of Information (AoI) to measure user experience, which makes my research completely different from the prior.

The system optimizes the 5G network in each level of NFV, as shown in Fig. 1.3.

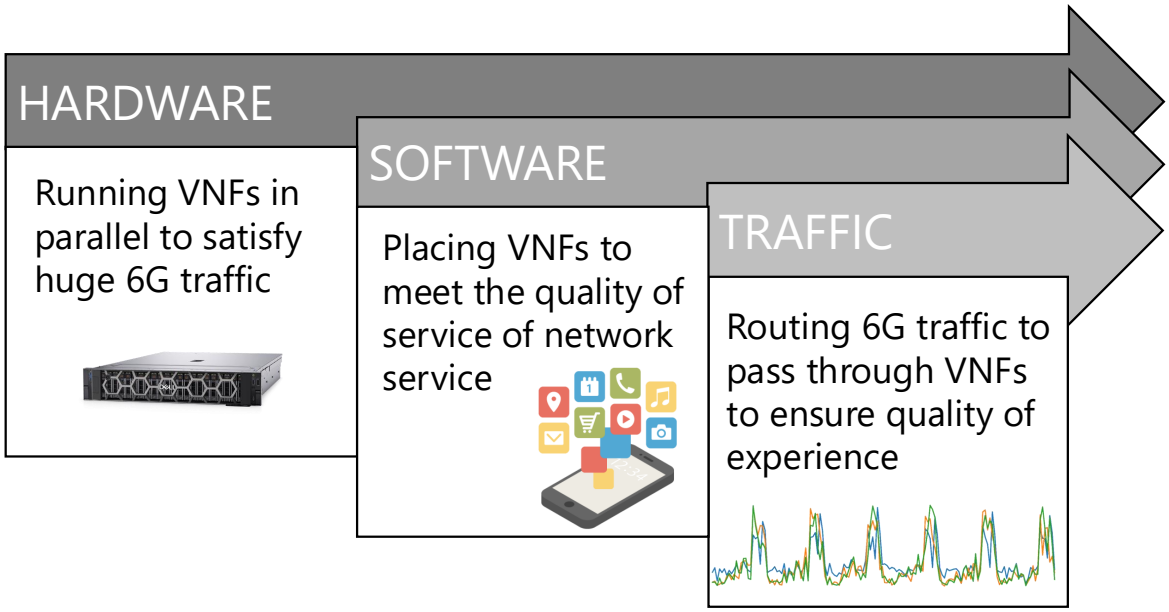

Fig. 1.3 System Outline

1.3.1 Parallel Network Functions Processing for 6G Traffic

The hardware resource of a modern server is becoming insufficient to handle huge 6G data traffic. To accommodate time-varying and huge 6G traffic, scaling VNF computation capacity is an important issue. However, precious research only focuses on scaling VNF

in one machine, limiting the scalability of VNF performance. To build this gap, the author presents a VNF performance scaling system. The system scales the VNF performance by running this VNF in parallel on another server. The process of scaling the VNF performance includes three steps: (1) monitoring the real-time loads of each VNF in the server; (2) deciding on whether a VNF needs to be scaled based on its load; and (3) scaling a VNF performance by running it in parallel on another idle server. Experimental results show that our system surpasses the raw NFV platform on performance by 1.02 times.

1.3.2 Network Function Placement for Quality of Service

In the 5G NFV network, a service is implemented as a service chain consisting of several VNFs that are chained together. A recent survey has revealed that when multiple VNFs are co-located in the same machine, resource contention for shared physical resources will occur and hence degrade the throughput of a VNF and finally increase its processing delays by 50%, as compared to it runs in isolation. However, prior works fail to capture this important characteristic because they simply treat machines as a resource pool without any resource contention happening, making their approach inapplicable to the VNF embedding problem when resource contention is taken into consideration. In this research, the author studied a contention-aware QoS-guaranteed VNF embedding problem. The problem is coded into a mathematical program under a couple of constraints. Given the problem is challenging to solve due to its nature of high complexity, a low-complexity approach is highly desirable for efficient VNF placement.

1.3.3 Data Traffic Routing for 6G User Experience

Compared with 5G, 6G is more sensitive to Quality of Experience (QoE) to deliver a smoother and more seamless experience. To maintain QoE in 6G applications like DT and XR, it is crucial to ensure not only low latency in data traffic but also regular data arrival. In this research, the author uses a novel metric, Age of Information (AoI), to quantify these two requirements, providing a comprehensive measure of the user's QoE. Then, the author studies AoI in a 5G NFV network. Considering that the traffic routing strategy in the NFV network significantly impacts the AoI in the B5G/6G application, the author aims to optimize the AoI by adjusting the traffic routing strategy. The problem is proved to be a Markov Process that can be solved by reinforcement learning. Therefore, the author proposes a Deep Reinforcement Learning (DRL)-based approach to train an intelligent agent that outputs an optimal traffic routing strategy. Finally, simulation results record a 20.3% reduction in AoI for B5G/6G applications.

1.4 Originality and Contributions

This dissertation's main originality and contributions are summarized as follows:

- In the first research, the precious work of NFV platforms in the past five years has been discussed. The author designed and implemented the system based on an open-source NFV platform. This system fully exploits scalability in VNF performance and is implemented on an open-source NFV platform. The author validated the scalability of the proposed system through extensive experiments, which show that it outperforms the other NFV platforms. For all we know, this research is the only study that has designed a VNF performance scaling system on an open-source system.
- In the second research, the author formulates the VNF placement problem as mathematical programming. Compared to the previous works, the author takes the dynamic network scenarios (i.e., user mobility) and resource contention into consideration. The author conducts simulation experiments on a real-world 5G network topology and the results show that the benefit of the proposed algorithms is higher MMO profit and service chain success rate. To the best of our knowledge, a few research focus on optimizing VNF placement in the consideration of resource contentions. This research is one of them.
- In the third research, the author aims to optimize QoE for upcoming B5G/6G applications in the NFV network. The author is the first to use AoI to measure QoE for B5G/6G applications. The author proves the traffic routing problem is a Markov process that can be solved by reinforcement learning. Therefore, the author designs a deep reinforcement learning-based approach to provide an optimal traffic routing strategy in the real-time network environment. This research is the pioneer in improving AoI in the context of NFV network.

1.5 Organization

The dissertation organization is as follows. The author briefly introduces the VNF performance scaling system for huge 6G traffic in Chapter 2. In Chapter 3, QoS optimization for 5G applications in the NFV network is considered. The author discusses the AoI metric in Chapter 4 and designs a traffic routing system to optimize AoI for B5G/6G applications. The author concludes this dissertation in Chapter 5.

Chapter 2

Parallel Network Functions for 6G Traffic

This chapter introduces our research at the hardware level for enabling VNFs to run in parallel on two servers to scale the performance.

2.1 Motivation

6G traffic is significantly greater than 5G due to its support for advanced applications and services. Despite the advantages of NFV, a critical challenge is to leverage its flexibility in scaling VNF Instances (VNFIs) in response to 5G traffic demand. To meet the Service Level Agreement (SLA) of the Service Function Chain (SFC), it's crucial for the VNFIs to have the ability to scale their capacities in a timely and efficient manner. Generally, as shown in Fig. 2.1, we can classify VNF scaling methods into the following three categories.

- **Vertical Scaling (VS)**: VS increases the VNFI capacity by allocating idle hardware resources to it. For example, letting more physical CPU cores and memory space to a running VNFI. While VS can achieve rapid and fine-grained scaling with the help of dockers (for containers) or hypervisors (for VM), VS is limited in its scalability as it is unable to scale outside single physical machines [55].
- **Horizontal Scaling across CPU Cores (HSC)**: HSC creates an additional VNFI by running them on other CPU cores and splits flows across the two instances. However, after HSC scaling, multiple homogeneous VNFIs will run parallelly in a single server with scarce hardware resources (e.g., CPU cores, cache, memory, storage, and NIC). According to recent research [76, 9], embedding two multiple VNFIs at the same server can lose half of their throughput (50.3%) compared to running in isolation because of resource contention.

- **Horizontal Scaling across Servers (HSS)**: HSS creates an additional VNFI on several physical servers to adjust VNF capacity. Unlike VS, which is limited by the capacity of a single server, which is limited by the capacity of a single server, HSS can leverage the hardware resources of all servers in the network, offering better scalability. However, HSS will bring unexpected startup overhead, such as the required time for building a new firewall VM instance [54], increasing the likelihood of SLA violations [38] or even incurring disaster during decision-making.

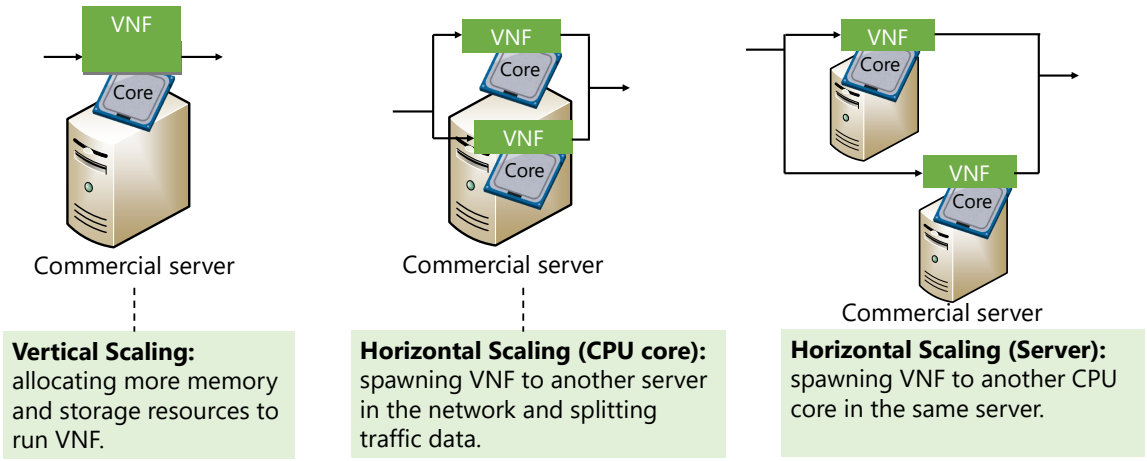

Fig. 2.1 Three Scaling Methods.

In this paper, we propose HyScaler, a dynamic, hybrid VNF scaling system that implements elastic SFCs by leveraging the benefits of all three VNF scaling methods to scale VNFIs. Moreover, HyScaler can thoroughly combine the pros. of vertical and horizontal VNF scaling to scale VNFIs and avoid their cons. We design and implement the prototype HyScaler upon OpenNetVM [81], a popular Data Plane Development Kit (DPDK)-based NFV platform that runs VNFIs in a docker container. Finally, we evaluate the scalability of HyScaler on a testbed and validate the effectiveness of the proposed algorithm by simulations. Overall, the primary contributions of this paper can be summarized as follows.

- We thoroughly investigate the popular high-performance NFV platforms and relevant works on VNF scaling algorithms in recent five years;
- We design and implement `HyScaler`, a fully exploiting NFV scalability to enhance VNFI performance on the real-world NFV platform;
- We validate the scalability of `HyScaler` through extensive experiments, which show that it outperforms the original OpenNetVM platform.

2.2 Related Work

Our proposed `HyScaler` is a hybrid VNF scaling mechanism built upon the DPDK-based OpenNetVM platform. In this section, we first review background knowledge of fast packet processing. Then, we review recent efforts in building scalable NFV platforms. Lastly, we summarize the related works on VNF scaling algorithms in the recent five years, as shown in Table 2.1.

Table 2.1 Summary of recent studies on VNF scaling Algorithms in 5 years

Literatures	Year	VS	HSC	HSS	Algorithm(s)	Implementation
Ghaznavi et al. [20]	2017			✓	heuristic MIP	simulations
Rahman et al. [52]	2018	✓			Machine Learning	simulations
Yu et al. [73]	2018	✓			heuristic	simulations
Yu et al. [74]	2018	✓		✓	heuristic ILP	simulations
Tang et al. [62]	2018		✓	✓	heuristic MILP	Openstack
Fei et al. [17]	2018	✓		✓	heuristic	simulations
Woo et al. [69]	2018		✓	✓	heuristic	simulations
Zhou et al. [85]	2019	✓			heuristic	simulations
Toosi et al. [65]	2019	✓	✓	✓	heuristic	simulations
Hu et al. [25]	2020		✓	✓	heuristic INLP	simulations
Luo et al. [46]	2020		✓		heuristic ILP	simulations
Zhai et al. [77]	2021	✓	✓	✓	ILP	simulations
Liu et al. [45]	2022			✓	heuristic INLP	OpenFlow
HyScaler	2022	✓	✓	✓	heuristic INLP	OpenNetVM

2.2.1 High-performance I/O Libraries

To meet the growing traffic demands, 10Gbps (or 100Gbps) Network Interface Cards (NICs) are today widely equipped in the servers. However, despite the popularity of high-speed NICs, packet processing often struggles to reach the line rate, due to the considerable overhead incurred by the Linux kernel [86]. Faced with this dilemma, several network I/O libraries, such as Netmap [53] and DPDK [1], were introduced as data plane libraries with the aim of accelerating the speed of packet processing. For example, Netmap [53] preallocates memory resources for NIC buffers, reduces system calls in large batches, and eliminates memory copying between userspace and kernel using shared buffers. Intel's DPDK [1] enables fast packet processing by completely bypassing the OS kernel and allowing network packets to be directly processed by the applications in the user space. We do not introduce more technical

details of these libraries, but they are now available in [3]. Because of its open source, rich APIs, high performance, broad community support, etc., DPDK has successfully become a popular standard to implement NFV in servers and is widely used in operator networks [78]. That's why we chose a DPDK-based NFV platform, OpenNetVM [81], to build our `HyScaler`.

2.2.2 Scalable NFV Platforms

While I/O libraries such as DPDK provide a high-performance solution for packet processing, they do not have APIs that are specifically designed to support VNF deployment. To address this issue, several NFV platforms have been developed on top of these I/O libraries. For instance, ClickOS [47] is a Xen-based NFV platform that leverages the netmap [53] data plane and VALE switch to boost the movement of packets between virtual machines. Additionally, NFP [61], HybridSFC [84], and ParaBox [83] take advantage of the parallelism between two VNFs within the same SFC to shorten the SFC length and reduce end-to-end latency. On the other hand, OpenNetVM [81] is a container-based implementation of NetVM [26] that leverages DPDK's high-throughput packet processing capabilities and optimizes packet delivery between VNF instances within a single server.

However, the above NFV platforms cannot build scalable implementations of VNFIs across multiple servers. Without a global SDN controller, they cannot be directly extended to the multi-server environment. Similar to `Hyscaler`, NFV Platforms, such as ScaleFlux [45] and E2, [48] support automatically scaling SFCs across multiple servers to improve SFC performance. For instance, E2 [48] is an NFV platform for placing, scaling, and end-to-end chaining VNFIs for SFCs. If a VNFI is overloaded, E2 scales the VNFI by adding an instance of this VNF and evenly distributes its load across two instances. However, they don't use a hybrid scaling strategy as `Hyscaler` does. Different from the above works that can merely scale a VNFI in a single server or a single data center, Duan et al. [11] and Jia et al. [28] propose a global management system that enables dynamic and flexible SFC placement and deployment. They also adopt a hybrid strategy of scaling SFCs across multiple data centers, but their works are out of the scope of this paper as we focus on scaling VNFIs in a single data center.

2.2.3 Virtual Network Functions (VNF) Scaling Algorithm

Significant efforts have been made in solving the VNF scaling problems. Here, we summarize the related works on VNF scaling algorithms in the recent five years, as shown in Table 2.1. These works adopt various techniques, such as traditional heuristics, machine learning, and

deep learning, to solve the NP-hard mathematical optimization problem of VNF scaling. Rahman et al. [52] develop a proactive QoS- and OpEx-aware VNF scaling algorithm based on machine learning. Yu et al. [73, 74] introduce ElasticNFV, a reactive approach for dynamic fine-grained VNF scaling to achieve high resource utilization and reduce flow migration time during scaling. Zhang et al. [82] designed the POLAR algorithm, an online proactive VNF scaling algorithm, to minimize the effect of inaccurate predictions by modeling the problem as a non-convex objective function and predicting the traffic load. Their goal is to minimize the effect of inaccurate prediction. Zhou et al. [85] define the problem as a bipartite matching problem, which heuristic algorithms solve. Different from HSC and HSS, VS may fail because it can't scale VNF instances outside single physical machines. When the hardware resources of the machine are unable to meet the demands of scaling, the scaling operation may become unsuccessful. As a result, the success ratio of VS is lower than HSC and HSS [77].

To overcome the limitations of VS, recent works concentrated on developing elastic SFC through horizontal scaling (HSC and HSS). For example, Ghaznavi et al. [20] solve the problems using Mixed Integer Programming (MIP) that distributedly places VNFI of the same VNF across multiple servers to minimize resource utilization and CapEx. Tang et al. [62] deploy a dynamic VNF scaling system in China Telecom, which scales VNFIs horizontally based on traffic estimation. However, HSC and HSS scaling at a coarse-grained level may cause unnecessary resource over-provisioning or under-provisioning, degrading resource utilization and increasing OpEx.

While homogeneous VNF scaling has its limitations in adapting to all network scenarios and meeting network service requirements, the hybrid design of VNF scaling is becoming more popular. Researchers, such as Yu et al. [74], assume the problem as an Integer Linear Programming (ILP) model and present the Rubik's algorithm to achieve a balance between performance and resource cost by exploiting fine- and coarse-grained hybrid scaling. Hu et al. [25] propose Palm, a proactive VNF scaling that aims to meet latency guarantees in a multi-tenant cloud environment. However, Rubik and Palm only exploit VS and HSS but ignore the advantage of HSC, which may lead to sub-optimal performance. The mentioned works [52, 73, 74, 82, 77, 20, 62, 74, 25] do not adapt a hybrid scaling that combines all VNF scaling methods as `HyScaler` does. Similar to `HyScaler`, [65] and [77] scale SFC through VS, HSC, and HSS. Toosi et al. [65] and Zhai et al. [77] propose hybrid scaling algorithms with improved vertical and horizontal scaling methods, but lack a real-world implementation.

In summary, existing NFV platforms reviewed don't scale VNFI considering all VNF scaling methods during runtime, while related works on the VNF scaling algorithm consid-

ered the VNF scaling problem as a mathematical optimization problem without real-world implementation.

2.3 System Design

2.3.1 NFV Platform

OpenNetVM acts as a bridge between VNFIs and NICs, facilitating packet delivery between VNFIs and NICs, maintaining the Flow Table, and enabling communication between VNFIs [81]. Fig. 2.2 provides an overview of OpenNetVM. To understand how OpenNetVM works, it's essential to understand its key components. When OpenNetVM starts, it launches a **Manager thread** to create a memory region in 2MB huge pages by invoking relevant DPDK APIs. This memory region is shared by all threads associated with OpenNetVM. After initialization, the manager thread manages all active VNFIs running on OpenNetVM.

Generally, each VNFI runs as an individual thread and is linked to its receive (*rx*) and transmit (*tx*) packet queues. The **Rx thread** is in charge of receiving incoming packets in the NIC Rx buffer and distributing them to the corresponding VNFI's *rx* based on the flow table. On the other hand, **Tx threads** removes packets from VNFI's *tx* and sends them to their next destinations, which could be another VNFI's *rx*, or the NIC Tx buffer. It's important to note that the queues and buffers are implemented as a ring buffer and are managed according to the First-in-First-Out (FIFO) rule.

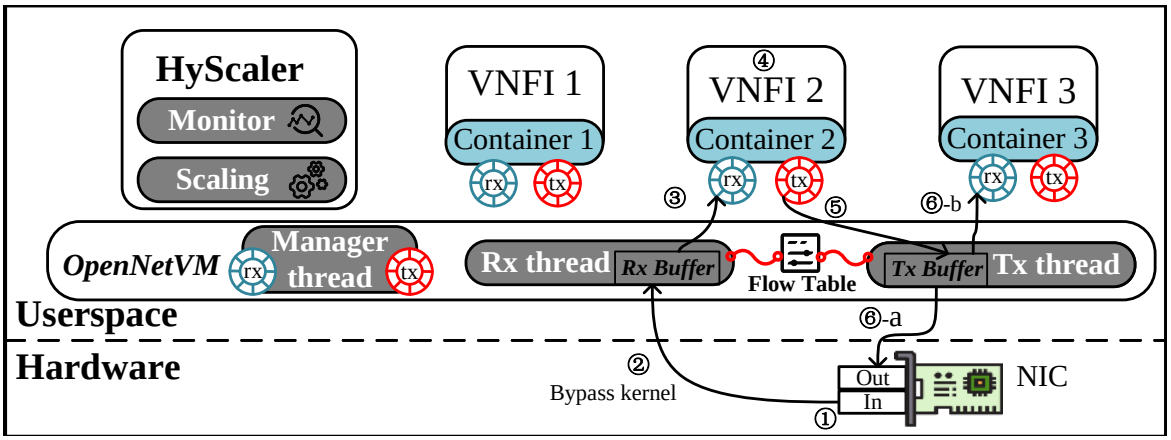

Fig. 2.2 The system overview of HyScaler.

In detail, the process of packets passing through OpenNetVM experiences the following steps, which are shown in Fig. 2.2:

① A batch of packets $\{p\}$ reaches in a NIC port from outside;

② For per p in $\{p\}$, NIC uses Direct Memory Access (DMA) [1] to copy p's data into the shared memory region and insert p's descriptor into the Rx buffer associated with this NIC port;

③ For each p in $\{p\}$, the Rx thread looks up the Flow Table to route p to the corresponding VNFI; Once p is matched to a VNFI, p's descriptor is copied into that VNFI's *rx* by the Rx thread, and then the Rx thread wakes the VNFI up to process the p's descriptor in its *rx*;

④ The VNFI reads and empties its *rx* and then processes the p;

⑤⑥ After finishing processing the p, the VNFI adds the p's descriptor back to its *tx*, where the descriptor will be read by Tx thread and redirected to the next VNFI (see ⑥-b) or send the packet out the machine (see ⑥-a).

Algorithm 1 Procedure for processing a batch of packets in OpenNetVM

Input: $\{p\}$, a batch of packets arrives at a NIC port,
FT, flow table

```
1: for each p in {p} do
2:    DMA_copy(p, NIC, Rx);
3:    /*The packets are transferred from the NIC to the Rx buffer using DMA.*/
4: end for
5: for each p in {p} do
6:    rx, tx = FT(p);
7:    if rx == NULL or tx == NULL then
8:       Continue /* No match of this packet in FT */
9:    end if
10:   enqueue(rx, p);
11:   /*Rx thread enqueues packet's descriptor to VNFI's rx*/
12: end for
13: each p is processed by the corresponding VNFI;
14: for each p in {p} do
15:   dequeue(tx, p);
16:   /*Tx thread dequeues packet's descriptor from VNFI's tx*/
17: end for
```

Algorithm 1 describes the procedure for processing packets in OpenNetVM in pseudo-code. Note that our `HyScaler` runs transparently to deployed VNFs and traffic flows. This section presents how key components work behind regular packet processing. Furthermore,

[1] Direct Memory Access (DMA) is a hardware device that enables input/output (I/O) devices to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations.

the design concept behind HyScaler can be extended to other DPDK-based NFV platforms as the packet processing mechanism is similar to that on other platforms that use DPDK. We chose to implement HyScaler using OpenNetVM, as its container-based implementation of VNFs unlocks the scalability potential of HSS, but the same approach could also be used for other VM-based NFV platforms.

2.3.2 Monitor Module

One of HyScaler's goals is *responsiveness*. To achieve this goal, HyScaler reacts rapidly to scale a VNFI when this VNFI becomes overloaded, avoiding SLA violations of SFCs. The critical problem is *determining whether a running VNFI is overloaded*? The monitor is a fundamental module to answer this problem. We think the VNFI's receive packet queue *rx* is in an ideal position to observe the load of each VNFI since any packets that are processed by this VNFI need to be enqueued into its *rx* at first. If the speed of VNFI processing packets cannot match the traffic load, arrived packets will overstock in the *rx*. The *rx*'s length will increase over time, where rx's length refers to the number of packets currently stored in the rx. Therefore, a sudden increment of the *rx*'s length is a clear semaphore indicating that the traffic load is rising [45].

Threshold. The scaling operation in HyScaler is triggered based on two predefined thresholds: (1) an upper bound threshold to scale the overloaded VNFI and (2) a lower bound threshold to release a redundant VNFI (if it exists). One significant challenge in implementing this is determining the suitable thresholds that can accurately probe changes in traffic load and thus trigger a scaling/releasing operation in a timely manner. However, determining the suitable thresholds is non-trivial, as the traffic load dynamically changes over time. There is a possibility that the *rx*'s length may surpass the threshold for a while and soon fall back to the normal level. When the monitor module gets the *rx*'s length surpasses the upper threshold, it will trigger the VNFI scaling. Then, the monitor module gets the average value, it will release the VNFI, which leads to unnecessary scaling of the VNFI and violates the *efficiency* design goal. Setting a lower bound threshold can avoid this condition. We set the upper and lower bound thresholds as 75% and 25% of the *rx*'s size, where *rx*'s size refers to the maximum capacity of packets that can be held in the rx. If the upper bound threshold is set too high, such as greater than 75%, it may lead to packet loss as the VNFI can not handle the incoming packets in *rx* timely due to overload. The same principle applies to the lower bound threshold, where if it is set too low, HyScaler may not perceive changes in traffic load in a timely manner and may release redundant instances prematurely.

2.3.3 Scaling Module

The scaling module is the basis for reaching the *scalability* design goal. We have the scaling priority: VS $\rightarrow$ HSC $\rightarrow$ HSS. The design of the scaling module strictly follows this priority. As a VNFI runs as a thread in OpenNetVM, it can VS itself by calling resource allocation APIs of DPDK. Therefore, the scaling module is responsible for scaling a VNFI for supporting HSC and HSS.

The scaling module reacts to commands from two sources, the Monitor module, and the Orchestrator:

- When it gets a command from the monitor module, that means a VNF is overloaded and needs to be scaled as soon as possible. According to the scaling priority, then, the Scaling module checks whether there are enough hardware resources in the server to create an additional VNFI. If yes, it creates a new VNFI and migrates an existing flow to the new instance. Contrary, if not, it sends a command to the Orchestrator for making an HSS decision, that is, deciding which server to create an additional VNFI.

- When it gets a command from the Orchestrator, that means the Orchestrator has made an HSS decision on the local server. Therefore, it creates a new VNFI and migrates an existing flow to the new VNFI.

The reactive process of the Scaling module when it receives a command from different sources is shown in Fig. 2.3.

Implementation. We implement the scaling module within the OpenNetVM's manager thread. OpenNetVM provides a communication channel (implemented as ring buffers) to the manager thread so that the monitor module running in each VNFI and Rx thread can use this channel to send a command to the scaling module. The manager thread runs by polling the communication channel to listen to commands from the monitor module. This information will be input for the algorithm to make a VNF placement and chaining decision. The scaling module and orchestrator communicate via Remote Procedure Call (RPC).

2.4 Evaluation

In this section, we demonstrate if `HyScaler` can scale VNF's performance by running it on a testbed, and validate the effectiveness of our proposed algorithm by simulations.

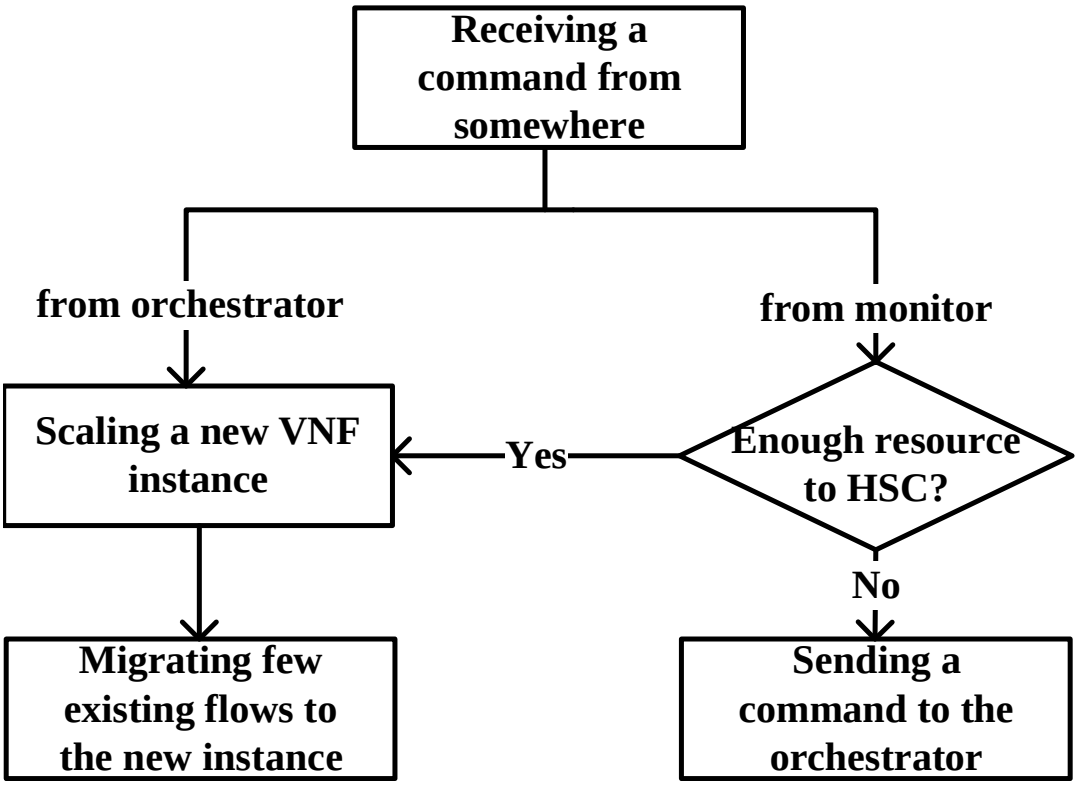

Fig. 2.3 Working process of the scaling module.

2.4.1 Testbed

Our experimental testbed consists of two commercial off-the-shelf servers connected via a 10 Gb switch. The configurations of each server are as follows:

Server A: (1) Dual Intel® Xeon® E5-2630 v3 CPU (Haswell), each of which has eight cores running at 2.4GHz; (2) 512KB L1 and 4MB inclusive L2 cache per core; (3) All cores in the same CPU share 40MB non-inclusive L3 cache; and (4) Each CPU is equipped with one 8GB DDR4 memory.

Server B: (1) One Intel® Core® i9-10900X CPU (Cascade Lake) with a total of 10 cores running at 3.70GHz; (2) 64KB L1 and 1MB L2 cache per core; (3) All cores share 19.25MB non-inclusive L3 cache; and (4) The CPU is equipped with one 64GB DDR4 memory.

Server A and B (5) disable Intel® Turbo Boost and Hyper-threading technologies; (6) are equipped with dual DPDK compatible Intel® I210 and I200 NICs, respectively; and (7) run OpenNetVM with DPDK version 20.10. Ubuntu 20.04 (with Linux kernel 5.4) and Ubuntu 19.10 (with Linux kernel 5.3) are used for Server A's and Server B's operating systems, respectively. Since we offload monitor tasks to the Rx thread and VNFI, there is no need to deploy an additional thread to implement its functionality. Likewise, the Scaling module

runs with the manager thread in OpenNetVM. We implement the Orchestrator as a thread running on Server A.

2.4.2 Experiment Result

Setup. In this experiment, we are to prove if HyScaler can scale VNFI's performance with traffic rising. To achieve this, we first select several basic VNFs in OpenNetVM and then created a simple SFC to test the scalability of our `HyScaler`: (1) Load Generator (LG): This VNF generates and delivers packets with a defined size and rate while measuring the latency of the packets when they are returned to the LG. (2) Firewall: The Firewall is a network security system that monitors and controls incoming and outgoing traffic based on predetermined security rules, acting as a barrier between trusted and untrusted networks. (3) Router: The Router directs data packets between computer networks, ensuring data is sent to its destination efficiently while enabling communication between different network segments. (4) IDS: The IDS monitors network traffic for suspicious activities or known threats, alerting administrators to potential security breaches. (5) Monitor: The Network Monitor continuously observes the performance, availability, and health of a network, providing insights and alerts for troubleshooting and optimization.

Next, we create a simple SFC (LG $\rightarrow$ target VNFI) in server A, with the target VNFI as the VNFI is expected to be overloaded and require scaling. To simulate rising traffic load, we used an LG to constantly generate packets at a rate of 10 million packets per second (pps). We simulate traffic load rising by increasing the number of LG instances and observe the scaling action of `Hyscaler`. However, due to our small testbed, we had to colocate other VNFIs on the same server with the LGs. Since the LGs would consume massive resources to generate packets, this could cause resource contention and potentially interfere with the performance of other VNFIs. As such, this experiment is primarily designed to demonstrate the scalability of HSS and discuss the effectiveness of `Hyscaler`. We design two baseline NFV platforms to compare with our system:

- OpenNetVM: The original OpenNetVM, without any optimization, cannot scale overloaded VNFIs across CPU cores or servers.
- OpenNetVM with HSC: The OpenNetVM with the ability to scale overloaded VNFIs across CPU cores through HSC.

Performance. The performance metric used to evaluate VNFI performance is "normalized pps", the ratio of the VNFI pps when it receives traffic from its upstream LGs, to its pps when it receives traffic from only one LG. Fig. 2.4, 2.5, 2.6 and 2.7 show normalized

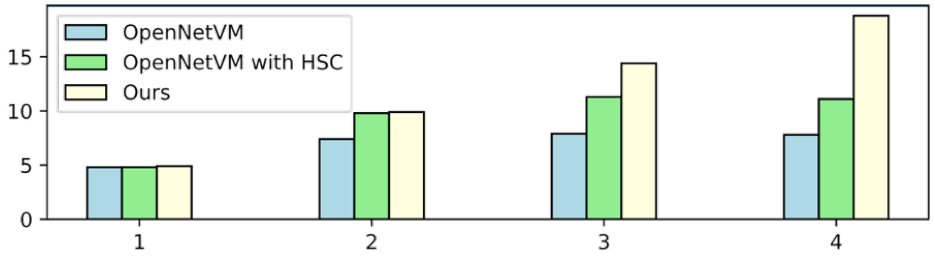

Fig. 2.4 Performance of Firewall.

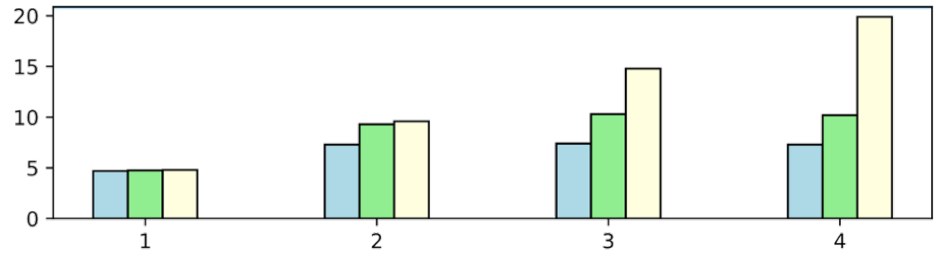

Fig. 2.5 Performance of Router.

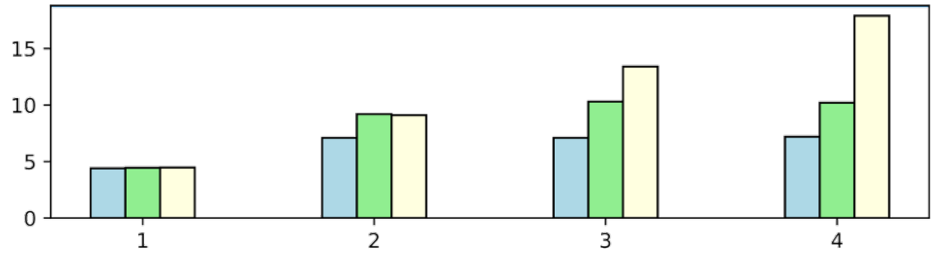

Fig. 2.6 Performance of IDS.

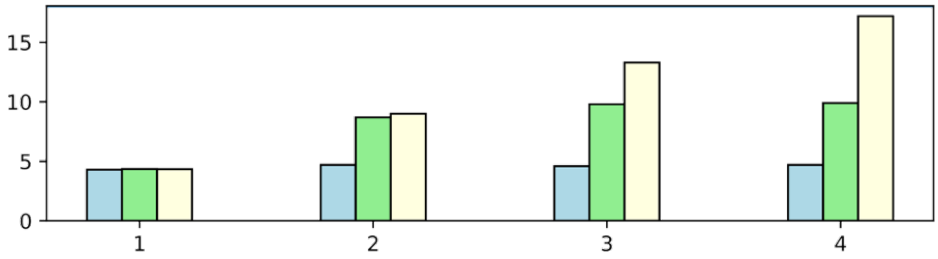

Fig. 2.7 Performance of Monitor.

pps of different target VNFs running on various baseline NFV platforms. As depicted in the figure, we find that resource contention is ubiquitous. All target VNFIs suffer performance degradation, with the number of upstream LG instances increasing when OpenNetVM is without HSS. For example, when a target VNFI IDS runs on OpenNetVM, its performance drops from 9.5×10^6 pps to 7.1×10^6 pps with LG instances increasing from 1 to 4. Even HSC can't play its role in scaling target VNFI performance in this situation, because it cannot scale outside a single physical server. On the contrary, our `Hyscaler` can avoid resource contention since our `Hyscaler` provides OpenNetVM with the ability to create an additional instance for the target VNFI in server B. When the num of upstream LG instances rises to 2, `Hyscaler` can detect the receive packet queue's length of the target VNF exceeding the threshold. `Hyscaler` timely create an additional VNFI to support the overloaded target VNFI and migrate a flow to the new VNFI. Therefore, when the number of upstream LG instances varies from 1 to 4, `Hyscaler` shows superior performance compared with other baseline NFV platforms. For example, when the target VNFI is Firewall, and the number of upstream LG instances is 4, `Hyscaler` outperforms the two counterparts about $1.02\times$ and $0.98\times$, respectively.

2.5 Conclusion

In this research, we introduce `HyScaler`, a dynamic, hybrid VNF scaling system for building elastic SFCs by scaling VNFIs across multiple servers. `HyScaler` provides a monitor module to detect network load changes and launch the scaling process timely. We design a scaling module to forward the scaling request to a global orchestrator, which will run the scaling algorithm to decide on placing and chaining the new VNFI in the network. Finally, we build a prototype of `HyScaler` on OpenNetVM and evaluate the scalability of `HyScaler` on a two-server-based testbed. Extensive experiments are conducted to evaluate the effectiveness of the algorithm in terms of the acceptance ratio, the number of used servers, execution time, resource utilization, and end-to-end delay of flows. From the experiment results, `HyScaler` shows great potential in scalable and flexible VNF scaling for future data center networks.

Chapter 3

Network Function Placement for 5G Quality of Service

This chapter introduces our research at the software level for optimizing VNF placements to ensure QoS.

3.1 Motivation

The development of the fifth-generation (5G) mobile communication networks enables a proliferation of diversified 5G network services, such as Autonomous Vehicles, VR/AR [10], Internet of Things (IoT) and so on. To support these emerging 5G applications, modern Telco Operators (TOs) tend to softwarize their core networks, making it easier and cheaper to customize, manage and orchestrate network services over a single shared physical network infrastructure. The two pivotal technologies to support this trend are: (1) Network Function Virtualization (NFV) to virtualize network services as a set of software instances called Virtual Network Functions (VNFs) and run them on commercial machines. (2) Software-Defined Networking (SDN) separates network control from data forwarding, thus creating the opportunity to apply different routing policies to different traffic flows.

According to the 3GPP 5G standards [21], the 5G core network (5GCN) of TOs, is based on Service-Based Architecture. In 5GCN, each service is generated by the user equipment (UE) and enters the network via the 5G base station (called gNB). Then, the user traffic flow passes processing through a chain of Virtual Network Functions (VNFs) in a specific order to gain an end-to-end customized 5G service. Conceptually, the chain interconnecting these VNFs is called a Service Function Chain (SFC). The ITU [56] has categorized 5G services into three major use cases: ultra-reliable and low-latency communication (URRLC)

for autonomous vehicles, enhanced mobile broadband (eMMB) for VR/AR, and massive machine-like communication (mMTC) for connecting a large number of IoT devices. On the other hand, these 5G services diversified with Quality of Service (QoS) requirements in terms of latency, security, reliability, and complexity. A typical use case of eMMB is watching a video streaming from the Internet, which can be represented by an SFC request: {NAT→FW→TM→VOC→IDPS}, as shown in Fig. 3.1. To fulfill the requirements specified in the Service Level Agreements with users, TOs need to embed the corresponding SFC onto the 5GCN infrastructure and sequentially steer the traffic flows through the SFC. This process, known as SFC embedding, is a critical and challenging task for TOs because the embedding decisions of SFCs can significantly influence the overall network performance and the profitability achieved by the TOs [41]. In this paper, we call this problem the

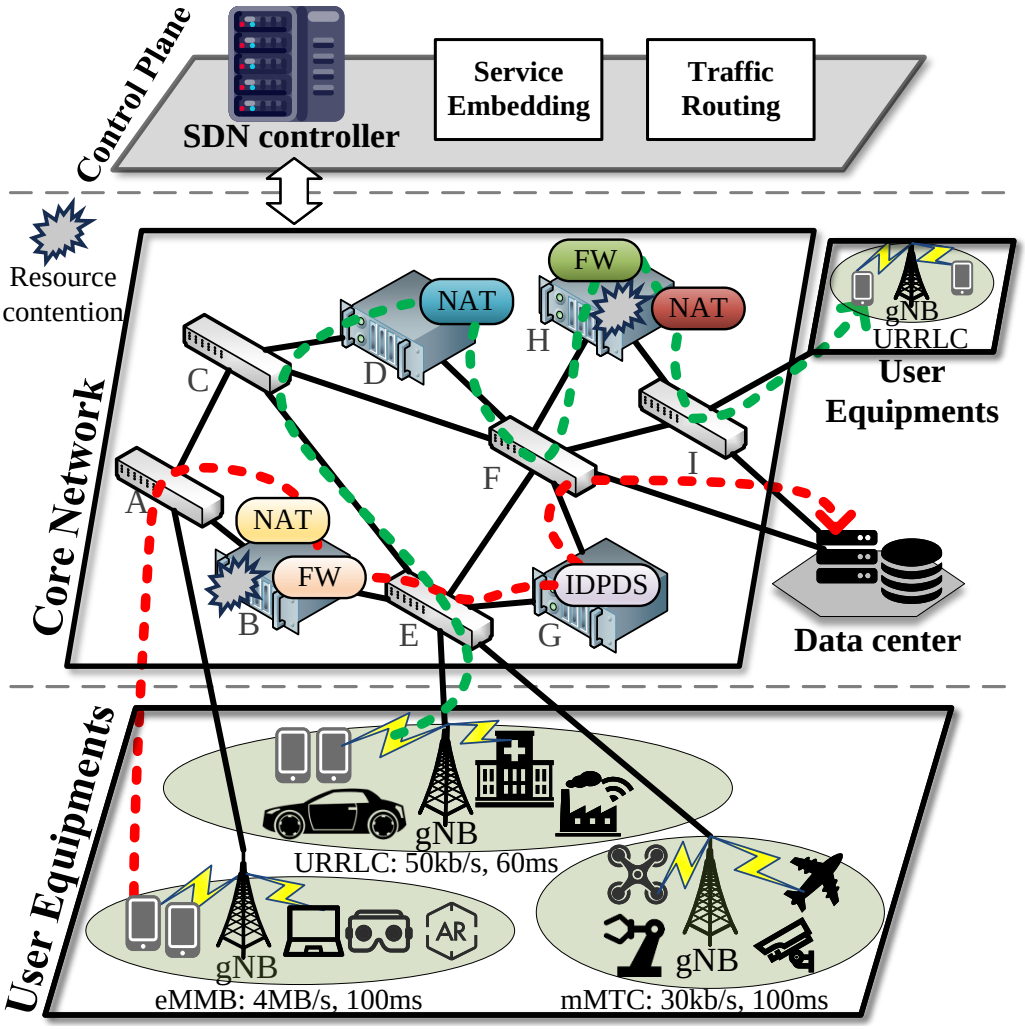

Fig. 3.1 NFV-based 5G core network.

SFC embedding problem and solve it by taking the following two key observations into consideration:

i) ***High mobility of UE***. Generally speaking, mobility is an important characteristic of the 5G UE. Consider a UE that enjoys 5G service via a gNB and then migrates to another gNB due to its location change. In this case, it has to be associated with a new adjacent gNB and reconnect the service. Consequently, the initial SFC embedding decision for this service might no longer remain optimal. TO should migrate the related SFC to maintain user connectivity. Failure to do so may result in a communication breakdown for the user, ultimately leading to QoS requirement violations [68].

ii) ***Resource contention within the NFV nodes and links.*** Modern NFV nodes in 5GCN are usually powered by commercial multi-core CPUs with the capability to operate multiple VNFs concurrently. In a multi-core NFV node, resource contention is ubiquitous and exits due to the limited capacity of shared physical resources, such as CPU cache, memory, and I/O subsystem. According to [31, 58], resource contention can lead to resource conflicts and result in degrading VNF throughput and finally increasing processing delays by nearly 50% as compared to it runs in isolation. Moreover, bandwidth contention in links is also non-negligible, especially in scenarios with high network traffic. The limited bandwidth of links can lead to congestion and performance degradation, affecting the overall end-to-end latency of SFCs. Fig. 3.1 shows that resource contention would happen if two VNFs are embedded in the same node, such as node B and H.

The above observations reveal two critical challenges: i) static SFC embedding approaches are not suitable for dynamic network environments due to the mobility of UE, and ii) the impact of intra-machine/intra-link resource contention on SFC performance must be carefully considered during the SFC embedding. For observation i), the dynamic SFC embedding problem was dedicated to solving in [64, 24, 70, 67, 22] but most of the existing studies make an assumption that all nodes and links are a simple resource pool without any resource contention. This simplification falls short of representing the general network environment, thereby rendering their methodologies inapplicable for the SFC embedding problem when resource contention comes into play. For observation ii), [51, 42, 37, 80] attempt to formulate the SFC embedding problem while taking resource contention into account. However, they fail to provide a comprehensive formulation because neither of them considers both two kinds of resource contention simultaneously. In summary, it remains open to solving the SFC embedding problem with the considerations of both observation i) and ii), and prior approaches are inapplicable to this problem due to their impractical assumptions

and simplified formulations. To the best of our knowledge, none of the existing works have completely captured this important characteristic when making SFC embedding decisions.

Therefore, in this paper, we investigate a QoS-guaranteed SFC embedding problem where both intra-machine and intra-link resource contentions are taken into consideration. We adopt a commercial perspective from TOs, recognizing that the primary goal of TOs is to operate their 5GCNs in the most cost-efficient manner, as it directly affects their financial success and overall business performance.

3.2 Related Work

Table 3.1 shows piror research about VNF placement. Existing efforts solve the SFC embedding problem according to various performance objectives. Although a comprehensive survey about embedding SFC requests in the different network scenarios has been discussed in [2, 23], much less attention has been paid to maximizing the profit of SFC embedding. Since the objective of our SFC embedding problem is to maximize the total profit derived from SFC requests, here we mainly discuss the prior research related to cost and profit optimization while emphasizing the differences between our work and the existing works.

Table 3.1 Piror research about VNF placement.

Literatures	Mobility	Delay	Throughput	Cost	Resource contention
Chen et al. [4]	✓			✓	
Peng et al. [16]		✓	✓		
Hantoutiel et al. [23]	✓	✓			
Jin et al. [29]	✓	✓			
Jia et al. [70]	✓	✓			
Zhang et al. [80]	✓				✓
Li et al. [42]	✓				✓
Ours	✓	✓	✓	✓	✓

Regarding resource contention, [58] first discloses that existing placement approaches may result in hardware resource contention of co-located VNFs. Resource contention will cause performance interference between co-located VNFs, leading to performance degradation ranging from 12.36% to 50.3%. [31] shows VNFs that run coresident on the same machine, contend over their hardware resources and, thus, might suffer from reduced performance compared to running alone on the same hardware. Some of the existing VNF placement approaches have taken the performance interference among co-located VNFs into consideration, such as literature [51, 42, 37] have attempted to capture resource contention

based on congestion game formulation. Literature [80] captures performance interference that arises from intra-machine resource contention and proposes an Adaptive Interference-aware Algorithm (AIA) to solve the SFC embedding problem. However, they solely focus on either intra-link or intra-machine resource contention but ignore another one, which does not appear to be comprehensive and practical. In our formulation, we aim to capture performance interference incurred both from intra-link or intra-machine resource contention.

Regarding cost optimization, some existing studies solely focus on minimizing the operational cost or deployment cost of VNF instances when embedding SFC requests. Chen et al. [6] investigated the problem of how to embed SFC requests with a dynamic data rate in the edge cloud with the lowest cost. Lin et al. [43] achieved a cost-efficient SFC embedding in a satellite edge computing network using game theory. Wang et al. [67] proposed an approach to optimize resource costs, covering both communication and computation costs, in a multi-layered network integrating air, ground and space segments. Gao et al. [18] focused on the cost of virtual machines in public cloud networks and proposed an efficient heuristic algorithm while considering the launch cost of virtual machines. However, the above works didn't provide a holistic view of total network costs, since the network cost not only includes deployment cost but also operational cost. Moreover, their approaches are not directly applied to the problem formulated in this research, as optimizing the cost independently cannot guarantee the maximization of profit.

Regarding profit optimization, Liu et al. [44] explored the SFC embedding problem within a multi-tier computing network to maximize the total profit. Yu et al. [72] investigated the orchestration of SFCs in an inter-datacenter network, which is dedicated to profit maximization for TOs. They address the problem by proposing an ILP and two online heuristic algorithms to optimize the deployment cost of SFC requests while guaranteeing the acceptance ratio. With the introduction of deep learning, Fan et al. [15] solved the SFC embedding problem in an online manner. Even though these works considered profit optimization in the SFC embedding problem, it is important to note that the above works didn't fully consider the influence of resource contention on the performance of SFC requests and the total profit. Therefore, to achieve efficient SFC embedding solutions, it is necessary to take into account the contention factor and its implications on performance, in addition to maximizing profit. Different from the above works, in this paper, we introduce a resource contention factor and M/M/1 queue model to capture the impact of resource contention on link accurately.

3.3 Problem Formulation

The system model mainly consists of two parts: i) an NFV/SDN-based 5G core network and ii) service requests with different data rates and QoS requirements. Then we mathematically formulate the SFC embedding problem as an INLP. Moreover, the important notations in our formulation are shown in Table 3.2.

3.3.1 Network Model

Consider a 5GCN as shown in Fig. 3.1 which consists of $\mathbb{N}$ and $\mathbb{L}$. The $\mathbb{N}$ denotes the set of NFV-compliant nodes in 5GCN, and $\mathbb{L}$ denotes the set of 10 Gb/s high-capacity fiber links that interconnect these nodes. There are two kinds of NFV nodes in the network: (1) switching node that runs Open vSwitch [66] to forward and transfer the traffic to neighbor nodes [63] via southbound protocols such as OpenFlow, (2) computing nodes that are equipped with generic-purpose hardware and are responsible for hosting different kinds of VNF instances to accommodate SFC requests. For each computing node $n \in \mathbb{N}$ associated with total computing resources, C_n^{cpu}, which are sufficient to host more than one VNF instance.

In 5GCN, each link $l_{n,n'} \in \mathbb{L}$ is a bi-directional fiber link connecting contiguous nodes n and n' within the network. And $l_{n,n'}$ is associated with the total bandwidth capacity $C_{l_{n,n'},BW}$. The computing resources are measured in terms of Million Instructions Per Second (MIPS). To ensure that our model accurately reflects the real-world network, we allow multiple SFC requests to share a single computing node and a link if their resource capacity is sufficient [40].

3.3.2 Service Chain Requests

We consider a scenario where there is a set of SFC requests, denoted by $\mathbb{R}$, that arrive at the network one by one. Then ISP needs to deploy them onto the network infrastructure and thus each SFC request is equivalent to a chain of VNFs. When a user needs to request a network service, the request will be first sent to an access point, to enter the 5G core network. Generally, an SFC service request is accomplished by an SFC, which is a series of VNFs that are linked together to direct the traffic passing through the VNFs in a specific order. Each SFC request $r \in \mathbb{R}$ encompasses a series of VNF requests denoted by $\mathbb{V}_r$. Each r enters or departs the 5G core network through a different node located near the network edge. In the NFV-based network, the SFC requests from UE are represented by the traffic flows. To obtain the required network service r, the traffic from users originating from the ingress nodes must traverse the VNF instances of $\mathbb{V}_r$ in predefined orders given by $\mathbb{V}_r$ and finally arrive at the

Table 3.2 List of Notations

Physical Network	
$\mathbb{N}$	Set of NFV nodes in the network, $n, n' \in \mathbb{N}$.
$\mathbb{L}$	Set of physical links in the network, $l_{n,n'} \in \mathbb{L}$.
$C_{n,cpu}$	Capacity of computing resource in n.
$C_{l_{n,n'},BW}$	Capacity of bandwidth in link $l_{n,n'}$.
SFC Requests	
$\mathbb{R}$	Set of SFC requests, $r \in \mathbb{R}$.
$\mathbb{V}_r$	Set of VNF requests of SFC request r.
$\mathbb{E}_r$	Set of virtual links connecting two adjacent VNF requests in $\mathbb{V}_r$
λ_r	Data rate of r.
$n_{r,in}$, $n_{r,out}$	Ingress and egress nodes of r.
$Delay_r^{max}$	Maximum tolerated delay of r.
Decision Variables	
x_n^{r,v_i}	A binary variable demonstrating whether VNF request v_i in r is embedded on node n.
$y_{l_{n,n'}}^{r,e_{v_i,v_{i+1}}}$	A binary variable demonstrating whether virtual $e_{v,v'}$ of r is embedded on link $l_{n,n'}$.
Delay-related Variables	
$Delay_{l_{n,n'}}^{tran}$	Transmission delay of link $l_{n,n'}$.
$Delay_{l_{n,n'}}^{prop}$	Propagation delay of link $l_{n,n'}$.
$Delay_{l_{n,n'}}^{que}$	Queuing delay of link $l_{n,n'}$.
$Delay_n^{pro}$	Processing delay in node n.
$Delay_r$	End-to-end delay of r.
$I_{l_{n,n'}}$	Physical distance of the link $l_{n,n'}$.
s	Speed of light.
$load_n$	The load of node n.
$load_{l_{n,n'}}$	The load of link $l_{n,n'}$.
Profit-related Variables	
$\Upsilon_{\mathbb{N}}^r$	Operation cost of VNF instance in r.
$\Upsilon_{\mathbb{L}}^r$	Communication cost of virtual links in r.
Υ	Total profit of embedding all SFC requests in $\mathbb{R}$.
P_r	Revenue of an SFC request r when it is embedded on the network.
P_{cpu}	Price of CPU resource per unit.
P_{BW}	Price of bandwidth per Gb/s.

egress node. Different SFC requests have different QoS requirements. Based on the above description, an SFC request r can be represented by a six-tuple,

$$r = \{\lambda_r; n_{r,in};\ n_{r,out}; \mathbb{V}_r; \mathbb{E}_r; Delay_r^{max}; P_r\}. \tag{3.1}$$

In (3.1), the variables represent the following: λ_r is data rate of r, indicating the amount of data to be processed. $n_{r,in}$ and $n_{r,out}$ are the ingress node and egress node of r, where the traffic flow of r enters and exits the network, respecticely. For each SFC request r, the $\mathbb{V}_r$ represents the set of associated VNF requests in the SFC of r that traffic needs to be traversed in order to gain the service, thus $\mathbb{V}_r = \{v_1, v_2, ..., v_i, ..., v_{|\mathbb{V}_r|-1}, v_{|\mathbb{V}_r|}\}$, $0 < i \leq |\mathbb{V}_r|$. In $\mathbb{V}_r$, each individual VNF request $v_i \in \mathbb{V}_r$ represents the i-th VNF request in r, corresponding to a specific type of VNF, such as DPI, NAT and FW. And $\mathbb{E}_r$ represents the set of virtual links between two adjacent VNF requests of r. In the $\mathbb{E}_r$, each individual virtual link $e_{v_i,v_{i+1}} \in E$ ensures the proper connectivity and sequencing between the i-th and $(i+1)$-th VNF requests, thus ensuring the order of VNF requests in the SFC r. $Delay_r^{max}$ is the maximum tolerated delay of r, indicating the maximum acceptable delay for completing the request r. P_r is the revenue gained by embedding this SFC request r without violating its maximum acceptable delay requirement $Delay_r^{max}$.

The binary variable x_n^{r,v_i} represents if the VNF request v of SFC request r is embedded on node n, as expressed in (3.2).

$$x_n^{r,v_i} = \begin{cases} 1, & i\text{-th VNF request of } r \text{ is embedded on} \\ & \text{node } n, \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

The binary variable $y_{l_{n,n'}}^{r,e_{v_i,v_{i+1}}}$ represents if the virtual link $e_{v,v'}$ of SFC request r is served by the physical link $l_{n,n'}$, as expressed in (3.3).

$$y_{l_{n,n'}}^{r,e_{v_i,v_{i+1}}} = \begin{cases} 1, & \text{virtual link } e_{v_i,v_{i+1}} \text{ of } r \text{ is embedded} \\ & \text{on physical link } l_{n,n'}, \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

3.3.3 Delay Model

Note that different SFCs have distinct QoS requirements based on the network services they support. When an SFC request r within the network, we mainly consider four kinds of delays associated with each SFC request [75, 12] as follows.

1) *Transmission delay*.

$$Delay^{tran}_{l_{n,n'}} = \frac{\lambda_r}{C^r_{l_{n,n'},BW}}. \tag{3.4}$$

where $C^r_{l_{n,n'},BW}$ is the bandwidth allocated to r.

2) *Propagation delay*. When an SFC request r traverses the link $l_{n,n'} \in \mathbb{L}$, its propagation delay depends on the bandwidth allocated to r and the physical distance of the link $l_{n,n'}$. The propagation delay is modeled as follows,

$$Delay^{prop}_{l_{n,n'}} = \frac{I_{l_{n,n'}}}{s}. \tag{3.5}$$

where $I_{l_{n,n'}}$ is the physical distance of the link l, $C^r_{l,BW}$ is the bandwidth allocated to r, and s is the speed that traffic propagates in that medium.

3) *Queuing delay*. The queuing delay at a link can be modeled using a queuing theory model, i.e., M/M/1 queue. In this model, the queuing delay is influenced by the link load, which is determined by the bandwidth utilization of the link before the arrival of SFC request r at node n. The queuing delay in link $l_{n,n'}$, can be expressed as follows.

$$Delay^{que}_{l_{n,n'}} = \frac{load_{l_{n,n'}}}{1 - load_{l_{n,n'}}} Delay^{tran}_{l_{n,n'}}, \tag{3.6}$$

where $load_{l_{n,n'}}$ is the load of the link $l_{n,n'}$ in term of bandwidth resource. The $load_{l_{n,n'}}$ can be calculated as follows.

$$load_{l_{n,n'}} = \frac{\sum\limits_{r \in \mathbb{R}} \sum\limits_{e_{v_i,v_{i+1}} \in \mathbb{V}_r} y^{e_{v_i,v_{i+1}}}_{l_{n,n'}} \lambda_r}{C_{l_{n,n'},BW}}. \tag{3.7}$$

4) *Processing delay*. Recent works [31, 58, 41] reveal that multiple VNF instances embedded on the same node will cause resource contention, resulting in performance degradation and increasing the processing delay. The contention factor captures the impact of resource contention on the processing delay at a node. Specifically, the delay at node n is modeled as follows.

$$Delay^{pro}_{n} = \frac{load_n}{1 - load_n} \frac{\lambda_r}{C^{r,v_i}_{n,cpu}}, \tag{3.8}$$

where $Delay^{pro}_n$ represents the processing delay at node n, $load_n$ is the node load, λ_r is the date rate of r, and $C^{r,v_i}_{n,cpu}$ is the CPU resource allocated to i-th VNF of r at node n, and $C_{n,cpu}$ is the total CPU resource capacity at node n.

The $load_n$ is the load of the node n in term of computation resource, which can be calculated as the proportion of the total computation resource allocated to all SFC requests to

the total computation resource capacity in node n. It's expressed as follows.

$$load_n = \frac{\sum_{r\in\mathbb{R}}\sum_{v_i\in\mathbb{V}_r} x_n^{r,v_i} C_{n,cpu}^{r,v_i}}{C_{n,cpu}}. \tag{3.9}$$

By considering all the mentioned delays, the E2E delay of the r can be calculated as follows.

$$\begin{aligned} Delay_r = &\sum_{v_i\in\mathbb{V}_r}\sum_{n\in\mathbb{N}} x_n^{r,v_i} Delay_n^{pro} + \\ &\sum_{e_{v_i,v_{i+1}}\in\mathbb{E}}\sum_{l_{n,n'}\in\mathbb{L}} y_{l_{n,n'}}^{r,e_{v_i,v'}} \left[Delay_{l_{n,n'}}^{tran} + Delay_{l_{n,n'}}^{prop} + Delay_{l_{n,n'}}^{que}\right]. \end{aligned} \tag{3.10}$$

3.3.4 Profit Model

After giving the delay model, we are now ready to construct the profit model of the TOs. The total profit of TOs is obtained by summing up the individual profits of each SFC request, $\Upsilon_{\mathbb{N}}^r + \Upsilon_{\mathbb{L}}^r$. The profit of each SFC request is composed of three parts: 1) the operation cost of VNF requests, 2) the communication cost of virtual links, and 3) the revenue gained by successful embedding.

1) *Operation cost of VNF instance*.

$$\Upsilon_{\mathbb{N}}^r = \sum_{v_i\in\mathbb{V}_r}\sum_{n\in\mathbb{N}} x_n^{r,v_i} C_{n,cpu}^{r,v_i} P_{cpu}. \tag{3.11}$$

where P_{cpu} represents the cost of computation resource per unit, indicating the cost associated with utilizing CPU resources.

2) *Communication cost of virtual links*.

$$\Upsilon_{\mathbb{L}}^r = \sum_{e_{v,v'}\in\mathbb{E}_r}\sum_{l_{n,n'}\in\mathbb{L}} y_{l_{n,n'}}^{r,e_{v_i,v_{i+1}}} \lambda_r P_{BW}. \tag{3.12}$$

where P_{BW} represents the cost of link bandwidth per unit.

We assign a revenue denoted as P^r to each SFC request r when its embedding decision successfully fulfills its QoS requirements. Therefore, The total profit of TOs can be computed by summing up the profits of all individual SFC requests, which can be represented as follows.

The total profit of TOs can be computed by summing up the profits of all individual SFC requests, which can be represented as follows.

$$\Upsilon = \sum_{r\in\mathbb{R}} (P^r - \Upsilon_{\mathbb{N}}^r - \Upsilon_{\mathbb{L}}^r). \tag{3.13}$$

With the aforementioned system model and decision variables, we now introduce the INLP formulation of the problem. When making embedding decisions, there are several constraints that must be satisfied. Given a physical 5G core network $\mathbb{G}$ and a set of SFC requests $\mathbb{R}$ needed to be embedded in the network, we have the following constraints:

- QoS requirements of SFCs requests must be met, especially in terms of the end-to-end delay and bandwidth.
- Resource capacity constraint of nodes and links must be met.
- The profit of SFC deployment and scheduling is maximized.

3.3.5 Constraint

Thus, we have these constraints as follows:

1) *Node and link capacity constraint*. Since different VNF instances may share the same physical node and link, we ensure that the computation resource occupied by embedded VNF instances on node n must not surpass the capacity $C_{n,cpu}$ in that node. The node resource constraint can be expressed as follows.

$$\sum_{r\in\mathbb{R}}\sum_{v_i\in\mathbb{V}_r} x_n^{r,v_i} C_{n,cpu}^{r,v_i} \leq C_{n,cpu}, \forall n \in \mathbb{N}. \tag{3.14}$$

$$\sum_{r\in\mathbb{R}}\sum_{v_i\in\mathbb{V}_r} x_n^{r,v_i} C_{mem}^{r,v_i} \leq C_{n,mem}, \forall n \in \mathbb{N}. \tag{3.15}$$

Likewise, for each link $l_{n,n'} \in \mathbb{L}$, we have a similar resource constraint as follows.

$$\sum_{r\in\mathbb{R}}\sum_{e_{v_i,v_{i+1}}\in\mathbb{E}_r} y_{l_{n,n'}}^{r,e_{v_i,v_{i+1}}} . \lambda_r \leq C_{l_{n,n'},BW}, \forall l_{n,n'} \in \mathbb{L}. \tag{3.16}$$

2) *VNF placement constraint*. These constraints guarantee that all VNF requests must be embedded on one and only one computing node.

$$\sum_{n\in\mathbb{N}} x_n^{r,v_i} = 1, \forall v \in \mathbb{V}_r \tag{3.17}$$

Likewise, the virtual link between the VNF request has to be embedded onto equal to or greater than one physical link.

$$\sum_{l_{n,n'}\in\mathbb{L}} y_{l_{n,n'}}^{r,e_{v_i,v_{i+1}}} \geq 1, \forall l_{n,n'} \in \mathbb{L}, \forall e_{v_i,v_{i+1}} \in \mathbb{E}_r. \tag{3.18}$$

3) *Single flow constraint*. For simplicity, we ensure the flow of each deployed SFC request r cannot be split into multiple flows.

$$\sum_{l_{n,n'} \in \mathbb{L}} y_{l_{n,n'}}^{r,e_{v_i,v_{i+1}}} = 1, \forall e_{v_i,v_{i+1}} \in \mathbb{E}_r, \forall r \in \mathbb{R} \tag{3.19}$$

4) *Ingress and egress node constraint.* This constraint ensures that the traffic of SFC request r will enter/leave the 5GCN from/through the ingress/egress node.

$$\sum_{n' \in \mathbb{N}} y_{l_{n_{r,in},n'}}^{r,e_{v_1,v_2}} = \sum_{n' \in \mathbb{N}} y_{l_{n_{r,out},n'}}^{r,e_{v_{|\mathbb{V}_r|-1},v_{|\mathbb{V}_r|}}} = 1, \forall r \in \mathbb{R}. \tag{3.20}$$

3.3.6 Objective

By incorporating complete knowledge of the system model, we formulate the SFC embedding problem into an INLP with the objective of achieving the maximum total profit of the TOs, as expressed follows.

$$\begin{aligned} \mathbf{P}_1 :\ & max\ \Upsilon \\ & = max \sum_{r \in \mathbb{R}} (P^r - \Upsilon_{\mathbb{N}}^r - \Upsilon_{\mathbb{L}}^r), \\ & \quad s.t.\ \text{Eq. } (3.2), (3.3), (3.15) - (3.20). \end{aligned} \tag{3.21}$$

By decomposing the original problem $\mathbf{P}_1$ into static ones, the total cost $\Upsilon_{\mathbb{G}}$ now becomes a part of the objective function. According to equation (3.13), the revenue Υ_*^r generated from deploying the SFC requests remains unchanged once they are embedded in the network. We can further transform $\mathbf{P}_1$ into a cost-minimizing SFC-E problem, as represented in $\mathbf{P}_2$.

$$\begin{aligned} \mathbf{P}_2 : & min \sum_{r \in \mathbb{R}} \Upsilon_{\mathbb{N}}^r + \Upsilon_{\mathbb{L}}^r, \\ & \quad s.t.\ \text{Eq. } (3.2), (3.3), (3.15) - (3.20). \end{aligned} \tag{3.22}$$

3.4 Algorithm Design

We first demonstrate the hardness of the problem by proving its NP-hardness. As finding the optimal solution becomes infeasible when the problem size grows, we propose a three-stage heuristic approach with low complexity, tailored to solve the SFC-E problem formulated in Section 3.3.4.

By assuming SFC requests in $\mathbb{R}$ that arrive one by one into an arriving queue, denoted by $\mathbb{Q}$, our approach is focused on deploying and routing a single SFC. Our solution consists

of three steps. First, we introduce the Cost Efficiency Factor Priority Sorting (CEF-PS) algorithm, which prioritizes high-value SFC requests at the head of $\mathbb{Q}$. Second, we embed the SFC requests one at a time, starting from the head of $\mathbb{Q}$. To accomplish this, we devise a K-Cost Minimizing Path (K-CMP) algorithm, which finds out the K paths with the lowest cost and chooses one for embedding the SFC request. Additionally, to avoid the local optimum of K-CMP, we randomly select B paths from the network for comparison with the paths obtained from K-CMP. Finally, with a set of $K+B$ paths at our disposal, we compute the delay and cost of each path in accordance with equations (3.10) and (3.13). Then, we select a path using the Delay-Aware VNF Embedding (DAVE) algorithm, designed to fulfill the requirement of the maximum tolerated delay for SFC.

3.4.1 Hardness Analysis

We present proof demonstrating that the SFC embedding problem is NP-hard. This implies that finding an optimal solution to the problem becomes computationally challenging as the problem size increases. While existing optimization solvers such as CPLEX, Gurobi, or GLPK can find optimal solutions for small-scale cases, their computational complexity becomes prohibitive for larger problem sizes. Therefore, we have developed a heuristic algorithm that strikes a balance between solution quality and computational efficiency, providing efficient solutions for practical scenarios.

Proof: The SFC embedding problem defined in $\mathbf{P}_2$ is an INLP that can be considered as a reduction case of an NP-hard Bin Packing (BP) problem.

The BP problem involves determining whether a given set of items can be packed into a fixed number of bins, for which each bin is with a limited capacity. The objective of BP is to maximize the total value of the packed items while ensuring that the total weight of the items in each bin does not exceed that bin's capacity.

We prove it by following these steps. First, we relax the objective Eq. (3.22) with only resource constraints on nodes Eq. (3.15). Next, we assume the VNF request only consumes CPU resources and sets the computation resource capacity of all nodes to be to a fixed constant. At this point, if we treat the nodes as bins with a fixed capacity and the VNF requests as items with different weights, the problem can be viewed as a reduction from the BP problem. Considering our problem is harder than the BP problem with additional constraints Eq. (3.2), (3.3), (3.16)-(3.20), the SFC embedding problem defined in Eq. (21) is also NP-hard.

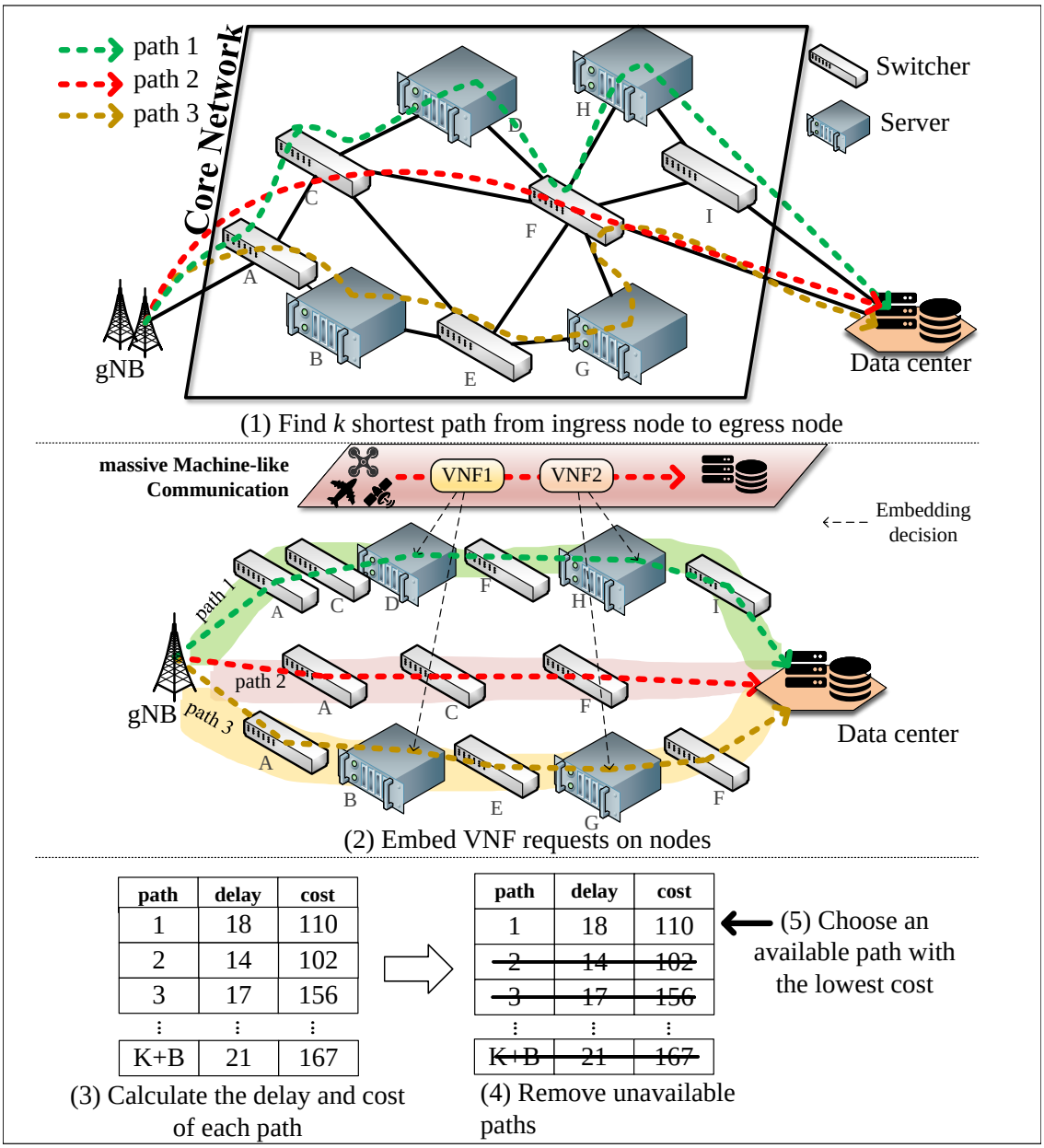

Fig. 3.2 Overview of our approach to embedding an SFC request.

3.4.2 Cost Efficiency Factor Priority Sorting (CEF-PS) Algorithm

Due to the resource constraints in the network, including the limitations on node and link resources, it is possible that not all incoming SFC requests can be accommodated simultaneously. In such cases, it is essential to prioritize the execution of high-profit requests to maximize the overall profit and resource utilization. However, when an SFC request is received by the TOs, the embedding decisions for that request are not known in advance, and therefore, its corresponding profit is also unknown. To measure the expectant profit of each SFC request, we define a new concept, the Cost Efficiency Factor (CEF). The CEF provides a metric that captures the cost efficiency of each SFC request, allowing the network operator to assess the profitability of different requests. By considering the CEF values, the operator can prioritize the execution of SFC requests with higher CEF values, as they indicate a higher revenue-to-data rate ratio and therefore potentially higher profitability.

$$CEF_r = \frac{P_r}{\lambda_r}. \tag{3.23}$$

Therefore, we use an ascending priority queue $\mathbb{Q}$ to store all arrived SFC requests and decide their order of execution. In $\mathbb{Q}$, requests with a higher priority are placed closer to the head of the queue and are executed earlier. Clearly, the design principles of $\mathbb{Q}$ should be:

- The SFC request with a higher CEF should have a higher priority, while those with a lower CEF have a lower priority. This ensures high-value requests are embedded prior to low-cost requests.
- If two or more SFC requests have the same CEF, the request with higher revenue should be placed in front.

It is important to note that the order in which SFC requests $\mathbb{R}$ initially reaches into an arriving queue $\mathbb{Q}$ is arbitrary and not optimized. To address this issue, we design a CEF Priority Sorting (CEF-PS) algorithm with the purpose to ensure that SFC requests with high CEF values are prioritized in $\mathbb{Q}$, allowing them to be embedded earlier than other requests with lower CEF. The CEF-PS algorithm is inspired by the bubble sort sorting algorithm. It iterates through $\mathbb{Q}$ and compares each pair of adjacent requests. If the CEF value of the current request is higher than that of the next request, the two requests are swapped. This process continues until the entire list is sorted in descending order of CEF values.

3.4.3 K-Cost Minimizing Path (K-CMP) Algorithm

Next, we introduce the K-Cost Minimizing Path (K-CMP) algorithm, which is specifically designed to obtain K candidate paths. The K-CMP algorithm follows the steps as follows.

1) Constructing a weighted undirected graph $\mathbb{P}$. Before embedding VNFs on the network, first, we need to find the paths with the lowest cost. We construct the cost-weighted network as a $|\mathbb{N}| \times |\mathbb{N}|$ matrix to indicate the cost of each edge when r traversing them, represented by $\mathbb{P}$. The weighted graph is designed according to the following principles:

- The physical link with a higher cost should have a higher weight, while a link with a lower cost has a lower weight. This means that the algorithm prioritizes links with lower costs when making placement decisions for an SFC request.
- If the unused bandwidth of a link is not sufficient to accommodate the traffic of the SFC request, the weight of that link will be a very large number, which means that the link cannot be considered for the placement of the SFC request. This indicates that the link is not eligible for consideration in the placement of the SFC request. By assigning a weight of a very large number to such links, the algorithm ensures that only links with sufficient bandwidth are considered, thereby avoiding infeasible or inefficient placements.

2) Pruning $\mathbb{P}$. Since the original graph, $\mathbb{P}$, is too large to run our algorithm efficiently, we first prune the unnecessary and redundant links to reduce execution time and computation complexity. We consider redundant links just those that have a bandwidth capacity lower than the data rate of the request. By eliminating these links from the graph, we can significantly reduce the search space for our algorithm and improve its efficiency. To do this, we create a pruned graph, $\mathbb{P}'$, by removing the redundant links from the original graph, $\mathbb{P}$.

3) Obtaining a set of K paths with the lowest cost. To find the K available paths from an ingress node to an egress node in a graph, we employ the K-shortest paths algorithm. This algorithm can constantly generate multiple paths by iteratively removing edges from the previously obtained shortest path. The process can be briefly described as follows: First, the algorithm initializes an empty set to store K paths with the lowest cost and initializes an empty priority queue to store candidate paths. Add the source node as the first path to the candidate paths. The process of path exploration is as follows: while the priority queue is not empty and the path set has not reached the desired size (K), perform the following steps: a) Pop the path with the smallest cost from the priority queue. b) Check if the last node in the path is the target node. If so, add the path to the path list. c) If the last node is not the target node, expand the path by exploring its neighboring nodes. For each neighbor, create a new

Algorithm 2 K-Cost Minimizing Path (K-CMP) Algorithm

Input: $\mathbb{G} = \{\mathbb{N}, \mathbb{L}\}$, the model of the network,
$r = \{\lambda_r; n_{r,in}, n_{r,out}; \mathbb{G}_r; \Psi_r^{max}; P_r\}$, a SFC request,
K, the size of shortest path set.
Output: *shortest_path*, a set of K shortest paths.

Construct a weighted undirected graph $\mathbb{P}$ based on $\mathbb{G}$
$\mathbb{P}' \leftarrow$ Prune $\mathbb{P}$
$shortest\_paths \leftarrow \{\}$
$candidate\_paths \leftarrow \{\}$
while $candidate\_paths \neq \emptyset$ and length($shortest\_paths$) == K **do**
  ($cost, path$) = pop the $path$ with the smallest $cost$ from $candidate\_paths$
  $last\_node$ = the last node in $path$
  if $last\_node == n_{r,out}$ **then**
    add $path$ to $shortest\_paths$
  end if
  if $last\_node \neq n_{r,out}$ **then**
    for ($neighbor$, $edge\_cost$) $\in \mathbb{P}'[last\_node]$ **do**
      $new\_path = path + [neighbor]$
      $new\_cost = cost + edge\_cost$
      add (new_cost, new_path) to $candidate\_paths$
    end for
  end if
end while
return $shortest\_paths$

path by appending the neighbor to the current path and calculate the new cost by adding the edge cost. Then store the path and its corresponding cost, to the set of candidate paths.

3.4.4 Delay-Aware VNF Embedding (DAVE) Algorithm

The Delay-Aware VNF Embedding (DAVE) Algorithm, as detailed in Algorithm 2, is designed to embed Virtual Network Function (VNF) requests into the most appropriate path from a pool of candidate paths, under constraints of the VNF order in SFC request, resource capacity of nodes/links and maximum tolerable delay. The algorithm takes as inputs the path set, *shortest_path*, and an SFC request, *r*. The SFC request is represented by a tuple that includes the SFC request's date rate, input node, output node, the graph representing the SFC request, the maximum processing capacity of a node, and the VNF request. The algorithm starts by initializing *VNF_requests* to be the VNF request set in the SFC request and *VNF_placements* to be an empty set. The length of the SFC request *SFC_length*, the length of the path sets *path_length* and the length of VNF requests *SFC_length* are determined. The algorithm then iterates over each *path* in *shorest_paths*. If *path_length* is greater than or equal to the *SFC_length*, the algorithm tries to embed the VNF request set on the nodes along the path. This is done by iterating over nodes of the path and attempting to embed the corresponding slice of VNF requests on it. If the number of successful VNF embeddings equals *SFC_length*, the algorithm returns. If the *path_length* is less than the *SFC_length*, the algorithm calculates how many VNF requests can be embedded on each node of the path (a) and how many VNF requests remain to be embedded (b). The algorithm then attempts to embed a VNF request on each node of the path and the remaining b VNF requests on the last node of the path. The algorithm will return if the total number of successful VNF embeddings equals the *SFC_length*.

3.4.5 Algorithm Analysis

Our approach consists of three heuristic algorithms: CEF-PS, K-CMP, and DAVE. The CEF-PS algorithm uses a bubble sort technique to prioritize SFC requests based on their CEF values. The time complexity of CEF-PS is $\mathcal{O}(|\mathbb{R}|^2)$.

The K-CMP algorithm leverages Yen's K-shortest path algorithm to obtain K paths with the lowest cost. K-CMP's complexity depends on the number of nodes and links, denoted as $|\mathbb{N}|$ and $|\mathbb{L}|$ respectively. The time complexity in best-case is $\mathcal{O}(|\mathbb{L}| + |\mathbb{N}| \log |\mathbb{N}|)$, while that in worst-case is $\mathcal{O}(K \cdot |\mathbb{N}| \cdot (|\mathbb{L}| + |\mathbb{N}| \log |\mathbb{N}|))$.

The DAVE algorithm is specifically designed for efficient VNF embedding with time complexity of $\mathcal{O}(K \cdot |\mathbb{V}_r|)$.

Algorithm 3 Delay-Aware VNF Embedding (DAVE) Algorithm

Input: *shortest_paths*, a set of K shortest paths,
$r = \{\lambda_r; n_{r,in}, n_{r,out}; \mathbb{V}_r; \mathbb{E}_r; \Psi_r^{Delay}; P_r\}$, a SFC request

1: $VNF\_requests = \mathbb{V}_r$, $VNF\_placements = \{\}$
2: $SFC\_length = length(\mathbb{V}_r)$
3: **for** $path \in shortest\_paths$ **do**
4: $path\_length$ = length($path$), $result == False$
5: **if** $path\_length \geq SFC\_length$ **then**
6: $i = 0, result = 0, sum = 0$
7: **for** $i < (path\_length - SFC\_length)$; $i++$ **do**
8: $result \leftarrow$ Embed $VNF\_requests[i : i + path\_length]$ on $path[i : i + path\_length]$
9: $sum = sum + result$
10: **end for**
11: If $sum == SFC\_length$ then return
12: **end if**
13: **if** $path\_length < SFC\_length$ **then**
14: $result == False$
15: $a = SFC\_length/path\_length$, $i = 0$
16: $b = SFC\_length - a \times path\_length$
17: **for** $i < path\_length$; $i++$ **do**
18: $result1 \leftarrow$ Embed $VNF\_requests[i \times a, i \times a + 1]$ on $path[i]$
19: $result2 \leftarrow$ Embed $VNF\_requests[SFC\_length - 1]$ on $path[path\_length - 1]$
20: $sum = result1 + result2$
21: If $sum == SFC\_length$ then return
22: **end for**
23: **end if**
24: **end for**

Considering the experiment parameters described in Section 3.5, the total time complexity of our approach depends on the K-CMP algorithm. Therefore, the upper bound complexity of the approach is $\mathcal{O}(|\mathbb{R}| \cdot (|\mathbb{L}| + |\mathbb{N}| \log |\mathbb{N}|))$ while lower bound is $\mathcal{O}(|\mathbb{R}| \cdot K \cdot |\mathbb{N}| \cdot (|\mathbb{L}| + |\mathbb{N}| \log |\mathbb{N}|))$.

3.5 Simulation

3.5.1 Simulation Setup

We implement our INLP with ILOG CPLEX v12.6 [27] and heuristic algorithms using Python 3.7 in an Intel® NUC with an i7-1260P CPU, 32 GB RAM and 64-bit Windows 11 Pro Education. All the experiments will be executed 10 times and the result represents the average values obtained from these trials. The parameter configurations for the simulation are presented in TABLE 3.3.

Table 3.3 Simluation setting

Parameters	Value	Description
$\lvert\mathbb{N}\rvert$	51	Number of nodes in $\mathbb{G}$.
$\lvert\mathbb{L}\rvert$	68	Number of Links in $\mathbb{G}$.
$\lvert\mathbb{R}\rvert$	$20-120$	Number of SFC requests.
$\lvert\mathbb{V}_r\rvert$	5	Numver of VNF request of r.
$C_{n,cpu}$	$\mathscr{U}(500,1000)$ MIPS	Capacity of CPU resource in n.
$C_{n,cpu}^{r,v_i}$	$\mathscr{U}(50,100)$ MIPS	CPU usage of i-th VNF of r.
$C_{l_{n,n'},BW}$	10 Gb	Bandwidth of link (n,n').
P_r	$\mathscr{U}(50,100)$	Revenue of request r.
P_{cpu}	$\mathscr{U}(6,13)$	Price of CPU resource per unit.
$P_{l_{n,n'}}$	$\mathscr{N}(0.8,0.02)$	Price of bandwidth per Gb/s.
s	3×10^8 m/s	Speed of light.
K	5	Length of the shortest path set.
R	6378 km	Earth radius.

1) *Network Topology:* we conduct our simulation on a 5G real-work network topology extracted from the Internet Zoo Topology [35]: Surfnet (51 nodes and 68 links). Surfnet's network topology is as depicted in Fig. 3.3. Unless specific illustration, all simulation settings are consistent with the aforementioned parameters as shown in TABLE 3.3 and conducted on Surfnet topology. Moreover, Internet Zoo Topology also provides the longitude ρ_n and latitude σ_n of nodes, which enables us to get the physical distance between any two nodes in the network, denoted by $K_{l_{n,n'}}$. We use the widely used coordinate algorithm to calculate the

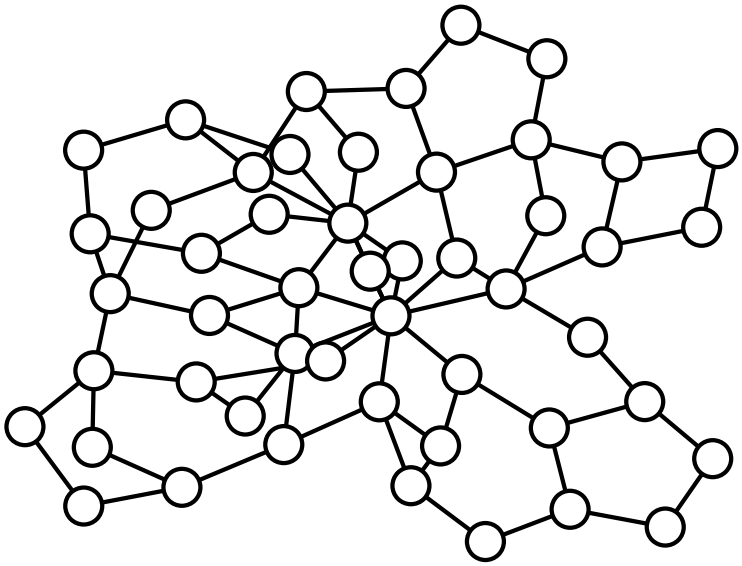

Fig. 3.3 Network topology of Surfnet.

physical distance $K_{l_{n,n'}}$ between two nodes n and n' based on their geographical coordinates. The calculation is expressed below,

$$I_{l_{n,n'}} = 2R \cdot \arcsin \sqrt{\sin^2(\tfrac{\rho_n - \rho_{n'}}{2}) + \cos\rho_n \cdot \cos\rho_{n'} \cdot \sin^2(\tfrac{\sigma_n - \sigma_{n'}}{2})}, \tag{3.24}$$

where R is the earth's radius. This additional information makes our simulation closer to the practical network and achieves a better understanding of the real-world implications of our proposed approach.

Table 3.4 SFC example of 5G services.

service	SFC Request	λ_r	$Delay_r^{max}$
Web Service	NAT→FW→TM→WOC→IDPS	100kbps	500ms
Voice over IP	NAT→FW→TM→FW→NAT	64kbps	100ms
Video Streaming	NAT→FW→TM→VOC→IDPS	4096kbps	100ms
Online Gaming	NAT→FW→VOC→WOC→IDPS	50kbps	60ms

2) *SFC Request:* While there are no specific SFC workloads or datasets publicly available, we use general network traffic datasets or cloud workload datasets as a basis for generating SFC requests. The selection of ingress and egress nodes of an SFC request is done randomly within the network. We choose four kinds of popular 5G services (i.e., Web Service, VoIP, Video Streaming and Online Gaming) as a basic to generate SFC requests (see TABLE 3.4), where each of the 5G services has different QoS requirements in terms of E2E delay $Delay_r^{max}$ and data rate λ_r. This is because End-to-end delay and bandwidth are foundational QoS

requirements for 5G services. For example, video streaming is characterized by the highest bandwidth requirement while video streaming has the strictest E2E delay requirement.

3) *Comparison Approaches:*

- **CPLEX**: CPLEX [27] is a set of software APIs developed by IBM that is often used for solving complex mathematical problems.
- **Ours**: to solve the SFCE problem in the 5G network, we proposed three heuristic algorithms: CEF-PS, K-CMP, and DAVE in Section 3.4. This approach is a combination of these three algorithms.
- **CEF-PS + K-CMP + FF**: this approach adopts a First Fit (FF) principle for its third step, which guides the selection of nodes for SFC embedding. That is, embedding the SFC request on the node first fits the requirement of the SFC request.
- **Adaptive Interference-Aware** (AIA): this approach represents the state-of-the-art SFCE approach from [80] that merely captures intra-machine resource contention but ignores intra-link resource contention.
- **Contention-Unaware** (CU): this approach represents the state-of-the-art SFCE algorithm from [29] with the aim to minimize the E2E delay. However, it does not consider any resource contention, which is a key point of our proposed approach.

3.5.2 Offline Embedding

1) *End-to-end (E2E) SFC Delay*. The author first compare the E2E delay among three kinds of approaches as listed in Section 3.5.1. The results are shown in Fig. 3.4. As the amount of SFC requests increases, the E2E delay also increases for all the approaches. This trend can be attributed to the higher contention for network resources, such as link bandwidth and node capacities when more and more SFC requests are deployed in the network. As a result, the network becomes more congested, leading to longer waiting times for requests in queues and increased transmission delays. Consequently, the E2E delay of the SFC requests grows as the system is required to handle more simultaneous requests. This observation highlights the significance of implementing efficient resource allocation to maintain satisfactory performance levels, especially when dealing with a larger number of SFC requests.

When handling the same amount of SFC requests, the E2E delay of CPLEX is always the smallest one because it can get the optimal solution. In contrast, CU nearly keeps the highest delay among all approaches, this is because its contention-unaware nature makes

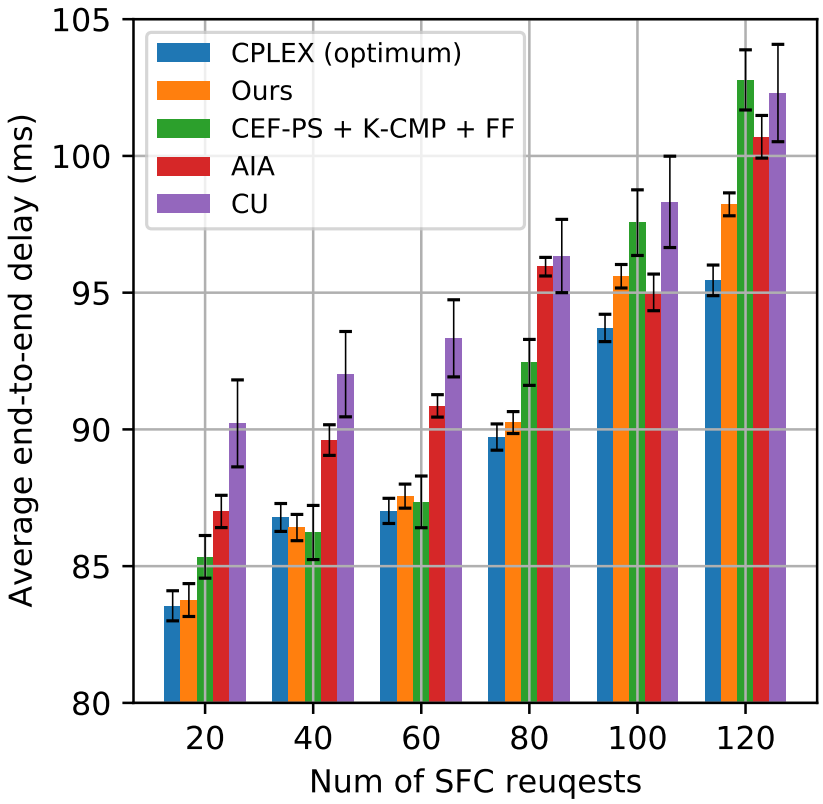

Fig. 3.4 E2E Delay vs number of SFC.

VNF instances highly concentrated in a few nodes, thereby increasing the load of the nodes and increasing the E2E delay. Meanwhile, our proposed approaches show slightly higher E2E delays in comparison with CPLEX. This difference can be attributed to the fact that the proposed algorithms are heuristic in nature, which means they aim to provide a good approximate solution rather than the exact optimal one.

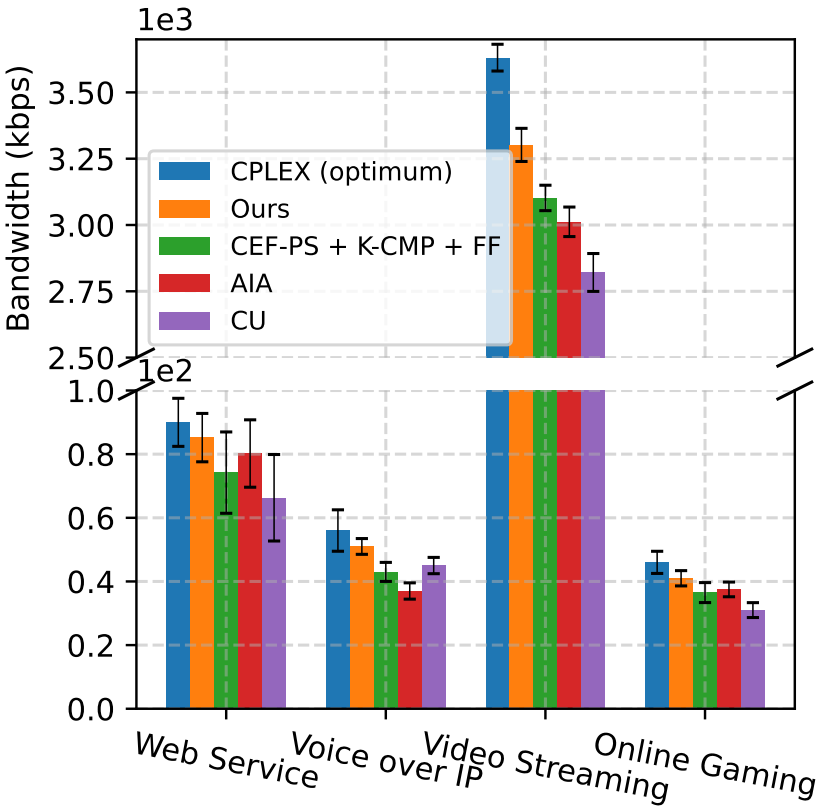

Fig. 3.5 Bandwidth of different services.

2) *Bandwidth.* Since resource contention also impacts the bandwidth received by the destination of the SFC request. This influence is demonstrated in Fig. 3.5, which illustrates the achieved bandwidth for different types of 5G services. As expected, CPLEX achieves the highest bandwidth for all 5G services and our approach achieves sub-optimal performance. The rationale behind this is that our approach is resource contention-aware which greatly alleviates the resource contention in the nodes/links. Our approach, being resource contention-aware, attains sub-optimal but commendable 80% performance, as it actively manages and mitigates resource contention within nodes and links. Conversely, AIA exhibits the second-lowest bandwidth allocation for all 5G services. This low bandwidth allocation might lead to a mismatch with service requirements, resulting in a reduced success rate Low bandwidth

may lead to a violation of QoS requirements, resulting in a low success rate. This aligns with the results shown in Fig. 3.5.

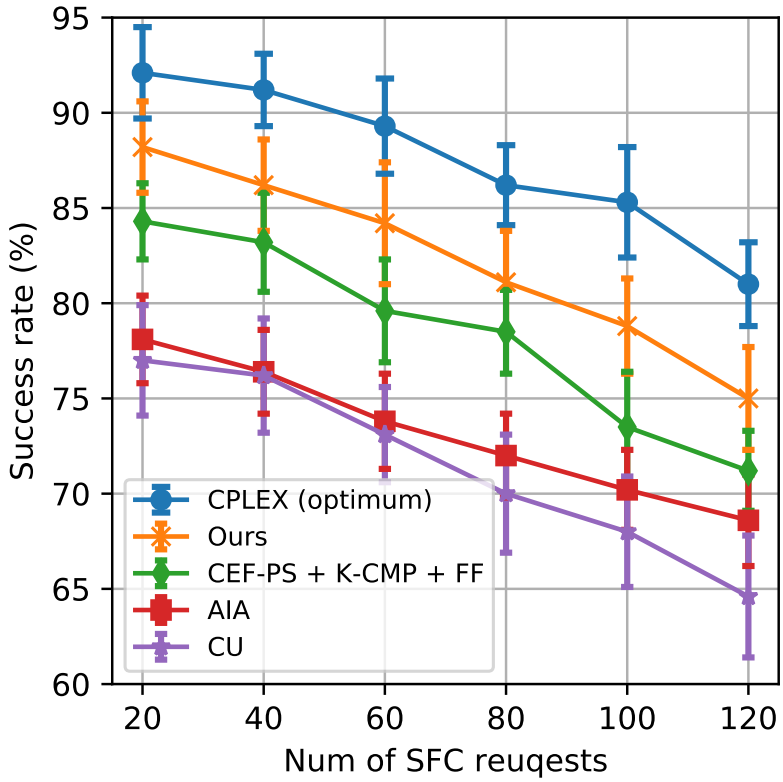

Fig. 3.6 Success rate vs number of SFC.

3) *Success rate.* The success rate is defined as the proportion of SFCs embedded that satisfy the following QoS requirements: i) the E2E delay not exceed the maximum acceptable delay requirement $Delay_r^{max}$, and ii) the bandwidth exceed or equal 75% of the input data rate λ_r. Therefore, the success rate can be expressed as,

$$success\_rate = \frac{\sum_{r \in \mathbb{R}} [\mathbb{I}(Delay_r \leq Delay_r^{max}) \cdot \mathbb{I}(Bandwidth_r \geq \lambda_r \cdot 75\%)]}{|\mathbb{R}|}, \tag{3.25}$$

where $\mathbb{I}$ is indicator function. Fig. 3.6 shows the success rate against the different numbers of SFC requests. As opposed to the CPLEX-based approach that has always been able to find an optimal placement solution, our approach accepted around 87 percent of the SFC requests due to sub-optimal SFC placements. CU gains the lowest success rate as compared with other approaches. This is because CU is completely unaware of resource contention, leading to severe resource contention arising from high node/link utilization, as shown in Fig. 3.9 and Fig. 3.10.

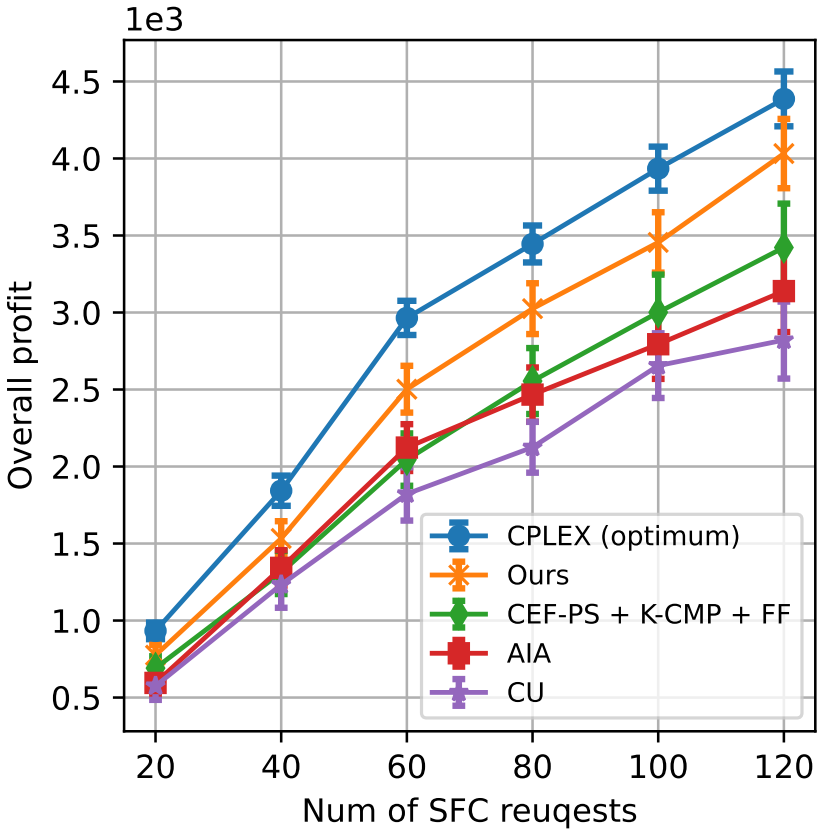

Fig. 3.7 Profit vs number of SFC.

4) *Profit.* Fig. 3.7 presents the comparison of total profits when these approaches handle varying amounts of SFC requests. We find there is a corresponding increase in the total profit when the amount of SFC requests increases. This is consistent with the expectation that an increase in the amount of SFC requests implies a greater amount of opportunities for the successful embedding and execution of service function chains. Each SFC embedding leads to increased revenue, contributing to the overall profit. When comparing the total profit among different approaches for the same amount of SFC requests, it is notable that the CPLEX solver consistently achieves the highest profit. This superior performance can be attributed to CPLEX's ability to find the optimal solution for our formulated problem, allowing it to utilize network resources in the most efficient manner with high possibility. As a result, CPLEX is able to successfully embed a higher amount of SFC requests, which in turn leads to a greater profit. It should be noted that CU shows inferior performance in total profit than other approaches, even worse than AIA. This is because it does not prioritize serving requests with high CEF, thus potentially missing out on more profitable opportunities.

Table 3.5 Running time (s)

No. of SFC request	20	40	60	80	100	120
CPLEX	150.2	320.1	734.5	1534.7	3232.2	11866.4
Ours	25.7	47.9	71.4	95.2	118.3	147.5
CEF-PS + K-CMP + FF	24.6	46.5	69	93.9	112.3	140.6
AIA	23.1	43.1	65.3	90	105.4	132.7
CU	22.2	45	62.5	85.8	102.3	129.2

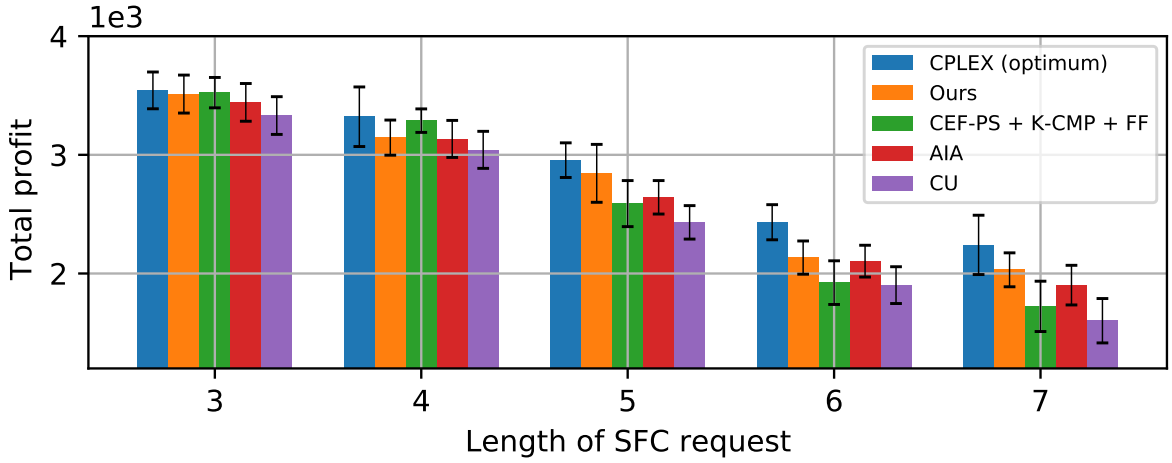

Fig. 3.8 Total profit against different SFC lengths.

5) *SFC length.* Regarding SFC length, we set it as five, which is consistent with the fact that all 5G SFC requests listed in Table III consist of five VNFs. To further enhance the persuasiveness of our approach, we conducted an experiment exploring the total probit against different SFC lengths. The experiment parameters are the same as TABLE 3.3 except for the SFC length, which ranges from 3 to 7, and the number of SFC requests, which is set as 60. The results are depicted in Fig. 3.8. When the SFC length is 3, the total profit of all approaches shows similar results because the number of VNFs can not incur significant influence of resource contention and reduce success rate. Correspondingly, the total profit of all the approaches, tends to decrease, as the SFC length increases. This is because as the SFC length increases, deploying an SFC request will consume more resources of nodes and links, aggravating the level of resource contention. More severe resource contention, in turn, results in longer SFC latency and less throughput, ultimately reducing the success rate in satisfying user requirements. As mentioned earlier, CU doesn't consider resource contention, making its total profit always the least. Our approach is close to the optimal solution in all cases. As compared with CEF-PS+K-CMP+FF, our approach shows superior performance. This is because the third step of our approach helps it escape from the local optimum and get a higher-quality solution.

6) *Running Time.* We also measured the total running time of the approaches to evaluate their scalability in large-scale networks. Although CPLEX can achieve optimal results, it's important to understand the trade-off involved. TABLE 3.5 shows the time required to execute the approaches. The results show that CPLEX incurs very high computational complexity. Therefore, when running in a large-scale network and with an increased amount of SFC requests, finding the optimal solution via CPLEX becomes computationally infeasible. Compared with CPLEX, our proposed approach demonstrates a trade–off between the optimality and the scalability of the solution.

3.5.3 Online Embedding

To conduct a more realistic 5G evaluation, we evaluate our approach in an online manner, where UEs have high mobility and the ingress node of SFC request will change from time to time. As the SFC embedding decisions are made without a priori knowledge, the CEF-PS algorithm can not be used due to its requirement of having all SFC request patterns in advance. Furthermore, CPLEX is not applicable to a dynamic network environment as it incurs high time complexity. First, the system time should be divided into discrete time slots set, donated by $\mathbb{T} = \{t_1, t_2, ..., t_j, ..., t_{|\mathbb{T}|}\}$. We set $|\mathbb{T}| = 100$ and $|\mathbb{R}| = 40$. Similar to the offline scenario, we assume that SFC requests sequentially enter into the network and we embed it at the beginning of each time slot. Then, to simulate the high mobility of UEs, we assume that the

ingress node of each embedded SFC request will change after a fixed interval. We set the interval as 5-time slots in this simulation.

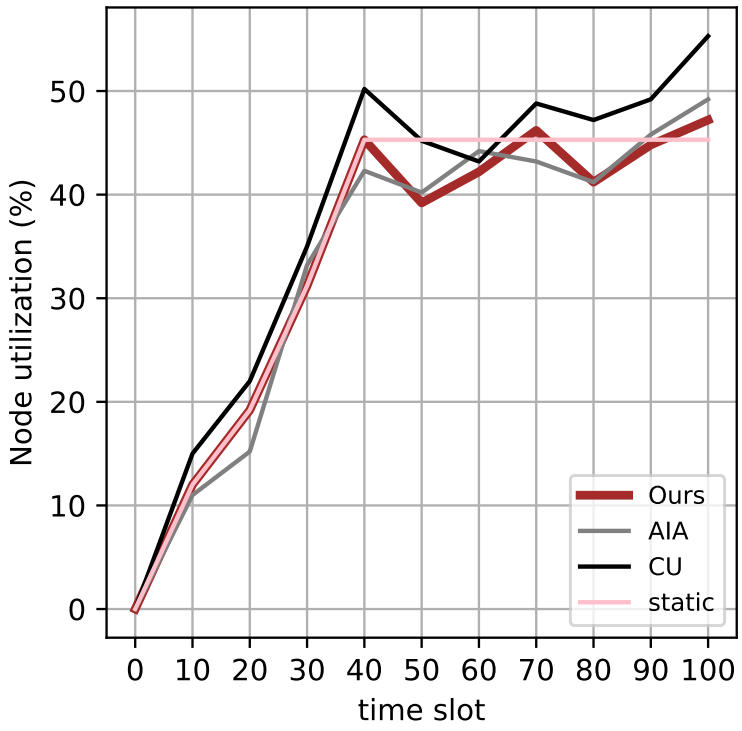

Fig. 3.9 Node utilization over time.

In addition, we assume that once the ingress node of the SFC request r is changed at time slot t_j, the system doesn't immediately reset the placement of r because there is a new SFC request arriving at the network at t_j. The reconfiguration of the placement of r won't occur until the subsequent time slot t_{j+1}, which means the placement of r remains unchanged during the period between t_j and t_{j+1}. This setting would help us observe the aftermath of employing static SFC embedding in a dynamic network environment. We introduce an additional benchmark objective for comparative evaluation:

- **Static**. This represents the conventional static embedding approach. The initial SFC embedding is using our approach. Once the placement of every single embedded SFC request is determined, the placement remains unchanged despite the high mobility of UEs.

1) *Node/link utilization.* The definition of node/link utilization is the same as the load of node/link (see Eq. (3.9) and (3.7)), which can reflect the level of resource contention within the node/link. Fig. 3.9 and 3.10 show the average node and link utilization against the time slots. The results show that before the 40 time slot, all four approaches increase

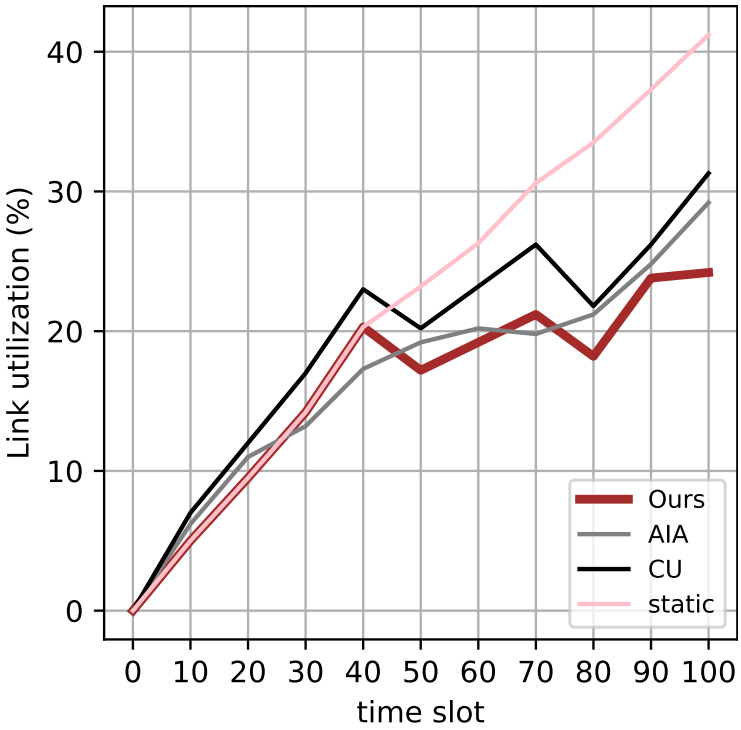

Fig. 3.10 Link utilization over time.

node/link utilization with time because a new SFC request arrives at every time slot and embedding it will occupy the resources of nodes and links. After 40 time slots, all approaches except for the static one fluctuate because the high mobility of UEs changes the placements of SFC requests dynamically. Note that the static remains node utilization unchanged and keeps increasing link utilization. This is because it keeps the placement of SFC requests unchanged but the ingress node of SFC requests is changing, which uses more links to keep the connection between the new ingress node and the origin SFC.

2) *Success rate* and *profit*. Fig. 3.11 and 3.12 show the success rate and profit, respectively, against different mobility intervals of UE. It is clear our approach and AIA can maintain a certain success rate/profit, despite the change of interval. This is because they are resource contention-aware and can adapt to different network environments. On the other hand, CU and static keep the momentum of growth in success rate and profit as the mobility interval increases. This is attributed to the fact that CU and static are unaware of resource contention and therefore their adaptability to the environment is weak. A longer mobility interval implies fewer changes in the network environment, which is beneficial to the performance of these approaches. Take a global view, even though CU and static increase, they are still inferior to our approach. In summary, our proposed approach performs better than the other three

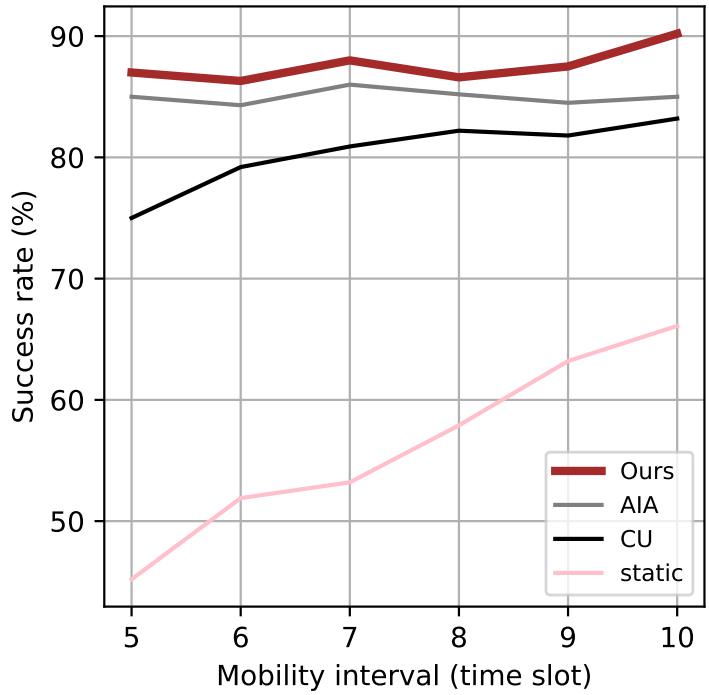

Fig. 3.11 Success rate vs UE mobility.

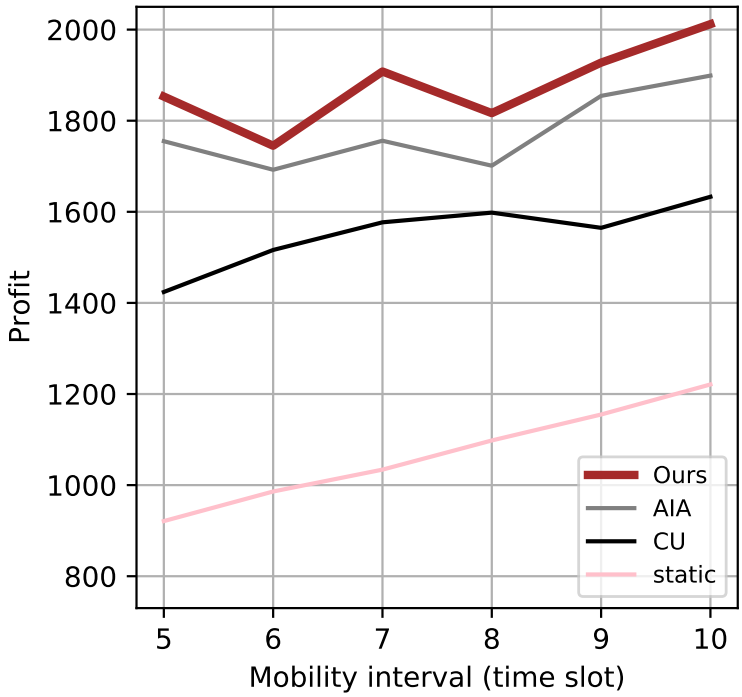

Fig. 3.12 Profit vs UE mobility.

approaches, which strongly validates its superiority and effectiveness in a dynamic network environment.

3.6 Conclusion

This paper formulated the SFC embedding problem within the context of a Network Functions Virtualization (NFV)-based 5G core network by proposing an *Integer Non-Linear Program* (INLP) with stringent constraints. Due to the high difficulty and hardness of the problem, solving the problem is computationally intensive and thus isn't feasible for real-time applications in large-scale networks. Therefore, we have proposed a low-complexity approach that incorporates three heuristic algorithms: CEF-PS, K-CMP, and DAVE. Finally, we employed the well-established solver CPLEX to solve the proposed INLP as a baseline to compare our approach. The numerical results validate its performance in reducing computational complexity while achieving near-optimal solutions for the SFC embedding problem. In the future, we will focus on distributed and decentralized SFC embedding problems. We plan to use a game theory to formulate the problem. Different from the previous work [51, 42, 37], our future work will consider both intra-link and intra-machine resource contention.

Chapter 4

User Traffic Routing for 6G User Experience

This chapter introduces our research at the traffic level for routing 6G traffic to ensure the QoE of users.

4.1 Motivation

The rapid development of the Internet of Things (IoT), fifth-generation (6G) communication and edge computing greatly carry out the prosperity of real-time applications, such as autonomous vehicles, virtual reality, and intelligent traffic. As their names suggest, the operation of real-time applications depends on real-time information, and quickly responds and acts on this information. Outdated information in these applications is valueless and may result in erroneous situations if decision-making is involved. Therefore, it is imperative to keep the data at the destination as fresh as possible for accurate data analysis and correct decision-making.

Fig. 4.1 An example of a user watching video online.

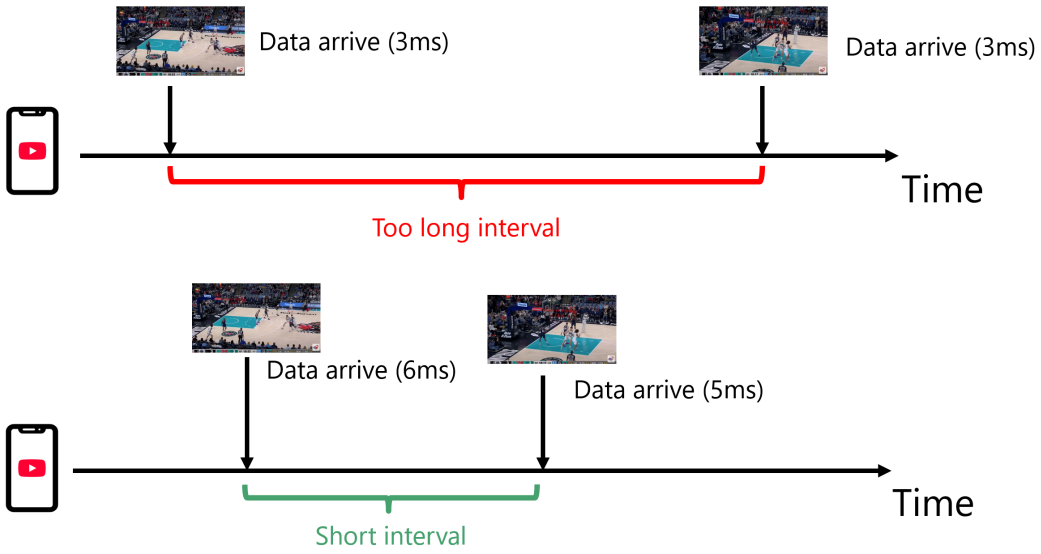

Fig. 4.2 two cases of arrival patterns of two adjacent video data.

Here I give an example to explain why delay is not enough to measure QoE. In Fig. 4.1, a user watching a video where the device is receiving video data continuously. Figure 4.2 shows two cases of arrival patterns of two adjacent video data.

- Case 1: **Low Delay with Long Intervals.** Two video data with smaller delays arrive at the device at a long interval. Despite the low delay, the screen experiences lag the infrequent updates, leading to a poor QoE.
- Case 2: **Higher Delay with Short Intervals.** Two video data with longer delays arrive at the device at a short interval. The screen updates more frequently, ensuring smooth playback and a better QoE than case 1.

Through this example, we can learn a lower delay doesn't result in better QoE. Data arrival interval plays a crucial role in ensuring timely updates on the device. Therefore, we use a new metric, AoI, which captures both delay and data arrival interval, providing a more comprehensive metric to measure QoE.

Recently, the notion of Age of Information (AoI) [34] has been introduced as a novel performance metric to evaluate how fresh the information is when it reaches the destination. For a packet/information received by the destination, the AoI can be simply defined as the *elapsed time between the present and the generation time of the latest useful information at the destination*. Then, the AoI associated with destination d can be given by a function of time t, that is $\Delta_d(t) = t - V_d(t)$, where $V_d(t)$ is the generation time of the latest information that the destination d has received at time t.

For ease of understanding, here is a simple example: a user watching a video where the device is receiving video data continuously, as shown in Fig. 4.1 and data arrival pattern is shown in Fig. 4.7. The first and second data arrive at the destination at time-steps 5 and 8. According to the definition of AoI, the AoI at destination will drop at time steps 5 and 8. Giving an example that the second data arrives at the destination at time-step 15 instead of 8. The AoI will keep increasing until time-step 15, and the AoI at time-step 15 will be a large value. Therefore, a high AoI can represent that data has not been updated for a long time and the regularity of data arrival is poor. Through this example, I explain that delay is not enough to measure QoE as a lower delay doesn't mean better QoE. Data arrival interval is also an important factor that worth to be considered. Therefore, we use a new metric, AoI, which can reflect both delay and data arrival interval. Compared to delay, AoI is a more comprehensive metric to measure QoE.

Using AoI for QoE is of great significance in the B5G/6G area because it provides a more comprehensive understanding of user experience by accounting for both delay and the timeliness of data arrival. Unlike traditional metrics such as delay, AoI captures the freshness of information, which is critical for real-time applications like autonomous driving, remote surgery, and immersive VR/AR experiences. For example, in autonomous driving, optimizing AoI for QoE can enhance passenger safety and can make driving behavior decisions timely and accurately. Autonomous vehicles rely on real-time data from sensors, cameras, LiDAR, and communication systems to navigate, detect obstacles, and respond to changing road conditions. Minimizing AoI ensures that the data being processed is fresh and up-to-date, allowing the vehicle to make accurate and timely decisions.

As a result, many efforts have been paid to proposed approaches to solve this problem with goals of optimizing various performance metrics, such as end-to-end packet delay [71, 7, 59], resource utilization [5, 8], expense [13, 19, 50], energy consumption [5, 59, 39], etc. However, it's important to emphasize that these approaches are not sufficient to optimize AoI and obtain a good AoI performance, because *a good AoI performance is achieved when packets with low delay are delivered regularly* [32]. Existing approaches focus only on minimizing end-to-end packet delay but have never taken the regularity of packet delivery into consideration.

To this end, we are motivated to propose an AoI-aware VNF Placement approach (named VNF_AoI), to automatically and efficiently place VNFs in the system. Then we model the VNF placement problem as a Markov Decision Process (MDP) and solve it by a Deep Reinforcement Learning (DRL)-based algorithm, Deep Deterministic Policy Gradient (DDPG). Our objective is to provide optimal online VNF placement policies to minimize the long-term average AoI at the destination.

Our main contributions of this paper are as follows:

- We formalize the VNF placement problem as a MDP and model it as a mathematical optimization problem aiming at minimizing the average AoI at the destination.
- Since the VNF problem is NP-hard and difficult to find a globally optimal solution, we propose an efficient DRL-based approach to provide an optimal VNF placement policy in the system.
- We conduct extensive simulation experiments to prove the effectiveness of our approach. The results reveal that our proposed approach outperforms the other two baseline approaches in term of acceptance ratio and average AoI at destination d.

4.2 Related Work

Our proposed VNF-AoI is built upon VNF placement and AoI optimization. In this section, we review recent studies on these two topics. Table 4.1 summarizes recent studies on AoI and NFV.

Table 4.1 Summary of recent studies on AoI and NFV.

Literatures	Delay	AoI	NFV	Online	Summary
Dalgkitsis et al. [8]	✓		✓	✓	Minimize the energy consumption while keep low delay of service chain
Chen et al. [7]			✓		Minimize the energy consumption of service chain deployment
Gao et al. [19]	✓		✓	✓	Reduce deployment cost for service chain
Zhang et al. [79]	✓		✓		Reduce delay of VNF in NFV
Houze et al. [49]	✓				Optimize AoI in UAV networks
Chen et al. [5]			✓		Optimize AoI in edge networks
Li et al. [39]		✓			Optimize AoI in wireless netoworks
Ours		✓	✓	✓	Optimize AoI in NFV

4.2.1 VNF Placement

Over the past decade, the VNF placement problem has been widely studied in various scenarios and accordingly solved toward different objectives. For example, Soualah et al. [59] formulate the VNF placement problem as a Binary Integer Linear Programming (BILP)

problem and proposed a heuristic algorithm named aiming at minimizing the total power consumption of VNF instances. Gao et al. [19] studied the VNF placement problem in large could data centers, which is formulated as a Mixed Integer Linear Programing (MILP) problem. Meeting the latency requirement of all SFCs, an efficient heuristic algorithm named as CE-VPS, was designed to address the problem to minimize OpEx and CapEx paid by the network operator. However, heuristics-based algorithms are not flexible to solve real-time VNF placement problems with dynamic changing workloads. Recently, DRL, as an important branch of Machine Learning (ML), is considered as a viable way to combat the limitations in heuristic. In [57], [39] and [50], the VNF placement problem is considered as a part of the MDP-defined VNF Forward Graph Embedding (VNF-GNE) problem. With Deep Q-Network (DQN), Li et al. [39] proposed an adaptive SFC mapping algorithm called ADAP to solve the VNF-GNE problem in 5G Mobile Edge Computing (MEC). Since traditional DRL-based algorithms, e.g., DQN, may encounter slow convergence problems due to the curse of dimensionality, Pham et al. [50] design a Deep Deterministic Policy Gradient (DDPG)-based algorithm to approach VNF-GNE problem. However, the above-mentioned works tend to optimize traditional metrics like packet delay and resource utilization, etc., but instead of AoI.

4.2.2 Age of Information Optimization

The problem of optimizing the AoI has been studied in different network scenarios. Kaul et al. [34] first introduce the concept of AoI and studies AoI in the context of Unmanned Aerial Vehicle (UAV) networks. Ma et al. [49] maintain fresh information at the edge network and propose a channel allocation algorithm to minimize long-term average AoI. Srivastava et al. [60] derive an exactly optimal scheduling policy minimizing the maximum AoI in wireless erasure channels. Besides, many of other works are designed based on an abstracting network model, such as M/M/1, M/G/1, and M/D/1 queueing model, which is not applicable to the VNF placement problem. In summary, existing works either do not optimize AoI in the VNF placement problem, or optimize AoI but not in an NFV-enabled environment. Unlike prior works, in this paper, we study and investigate AoI in an NFV-enabled network.

4.3 System Model and problem formulation

4.3.1 System Model

We consider a system, where status updates are randomly generated by multiple sources, and then delivered to a common destination node over an NFV-enabled network with different

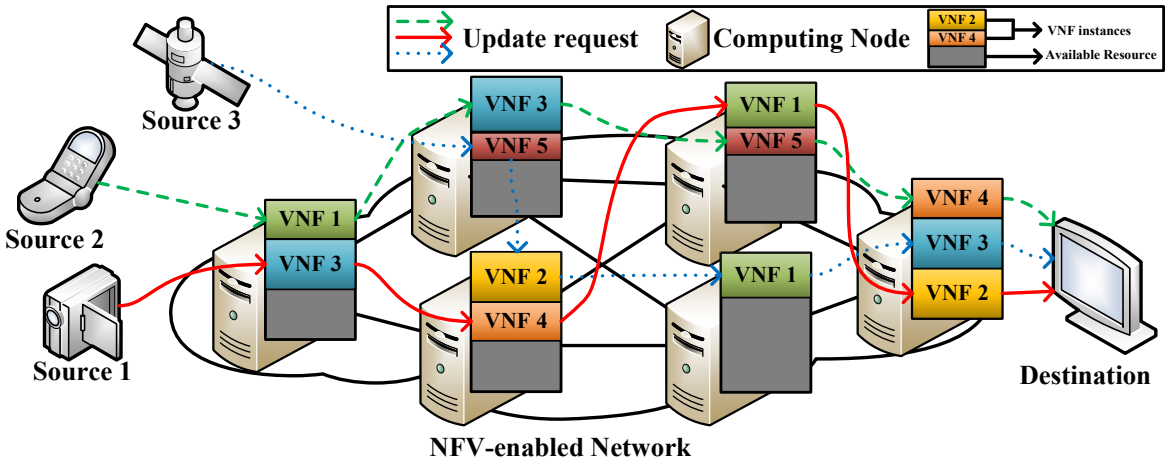

Fig. 4.3 A NFV-enabled multiple sources updating system. Three SFCs (labeled in three different colors) are embedded in the network.

delays, as shown in Fig. 4.3. For such system, we model it as an undirected graph $\boldsymbol{G} = (\boldsymbol{N}, \boldsymbol{E})$. The $\boldsymbol{N}$ represents the set of all physical nodes in the system including three types of nodes, the set of source nodes $\boldsymbol{S}$, the set of the computing nodes in the NFV-enabled network $\boldsymbol{M}$, and a destination node d, i.e., $\boldsymbol{N} = \boldsymbol{S} \cup \boldsymbol{M} \cup \{d\}$. The $\boldsymbol{E}$ represents the set of all physical links between the nodes of $\boldsymbol{N}$. For each node $m \in \boldsymbol{M}$ and each link $e \in \boldsymbol{E}$, they provide resource capacities for VNFs and to transmit update packets, respectively. So we use C_m^{core} and C_m^{mem} to denote the number of CPU cores and the memory capacity of each NFV node $m \in \boldsymbol{M}$, respectively. Similarly, we use C_e^{bw} to represent the bandwidth capacity of link $e \in E$. Assume the system supports a set of VNFs $\boldsymbol{F}$. Moreover, we assume that other types of resources are totally sufficient in each node and link.

The system time is divided into a set of discrete time steps $\boldsymbol{T} = (1, 2, 3, .., t, ...)$, where $t \in \boldsymbol{T}$ represents t-th time-step. In the system, we assume every update is generated at the beginning of time-steps and the lifetime of each update spans multiple time-steps. Let $\boldsymbol{U}_t$ denote the set of update requests preparing to enter and already active in the network at time-step t. We define each update $u \in \boldsymbol{U}_t$ as $u = (s, f_1, f_2, ..., f_n, d)$, where $s \in \boldsymbol{S}$ is the ingress node of update u, d is the egress node of the update u, and $f_1, f_2, ..., f_n$ is a SFC including a set of VNFs that update u requires to sequentially steered over.

Notice that each VNF $f \in u$ should be placed on a NFV node $m \in \boldsymbol{M}$ and require a certain amount of resources on NFV node m to support update u.

Let C_f^{core} and C_f^{mem} denote the number of CPU cores and memory capacity required by VNF f of update u, respectively. Similarly, let C_u^{bw} denote the bandwidth consumption of an update u. Therefore, the availability ratio of CPU core, memory of NFV node m and the

bandwidth of link e, respectively, at time-step t, are denoted by

$$\omega_{m,t}^{core} = 1 - \frac{\sum_{u \in \boldsymbol{U}_t} \sum_{f \in u} x_{m,t}^{f} C_{f}^{core}}{C_{m}^{core}}, \tag{4.1}$$

$$\omega_{m,t}^{mem} = 1 - \frac{\sum_{u \in \boldsymbol{U}_t} \sum_{f \in u} x_{m,t}^{f} C_{f}^{mem}}{C_{m}^{mem}}, \tag{4.2}$$

$$\omega_{e,t}^{bw} = 1 - \frac{\sum_{u \in \boldsymbol{U}_t} y_{e,t}^{u} C_{u}^{bw}}{C_{e}^{bw}}, \tag{4.3}$$

where $x_{m,t}^{f}$ and $y_{e,t}^{u}$ are two binary variables to indicate whether VNF f is deployed in node m and whether update u passes through physical link e at time-step t, respectively.

4.3.2 Age of Information

Prior to introducing problem formulation, we characterize the AoI of the destination d in the context of our system model. The concept of AoI is first defined as the elapsed time since the latest received status update packet at a destination node [34]. We follow the definition in [34] and formulate the AoI at the destination d as

$$\Delta_d(t) = t - V_d(t), \forall t \in \boldsymbol{T}, \tag{4.4}$$

where $V_d(t)$ is the generation time of the latest update that the destination node d has received at time-step t. Over the time-step interval [0, t), let $\Delta_{d,t}$ be the average AoI of destination d, defined as

$$\Delta_{d,t} = \frac{1}{t} \int_0^t \Delta_d(x) dx. \tag{4.5}$$

For ease of understanding, we take an example of the evolution of AoI at the destination d, as shown in Fig. 4.4. Without loss of generality, our observation begins at time-step 0 when the AoI is $\Delta_d(0) = 0$. In the time-step interval [0, 8), the destination d does not receive any new packets from sources, so the value of $\Delta_d(t)$ will grow linearly with t, which means the current update is getting older. Upon the destination d sees an update which is timestamped 3, at time-step 5, the AoI of the destination d is reset to $\Delta_d(5) = 5 - 3 = 2$. Then the AoI continues to grow until the next update arrives. Thus, the AoI function $\Delta_d(t)$ is shown in the pattern of saw tooth waveform and $\Delta_{d,t}$ is the normalized area under the saw tooth waveform.

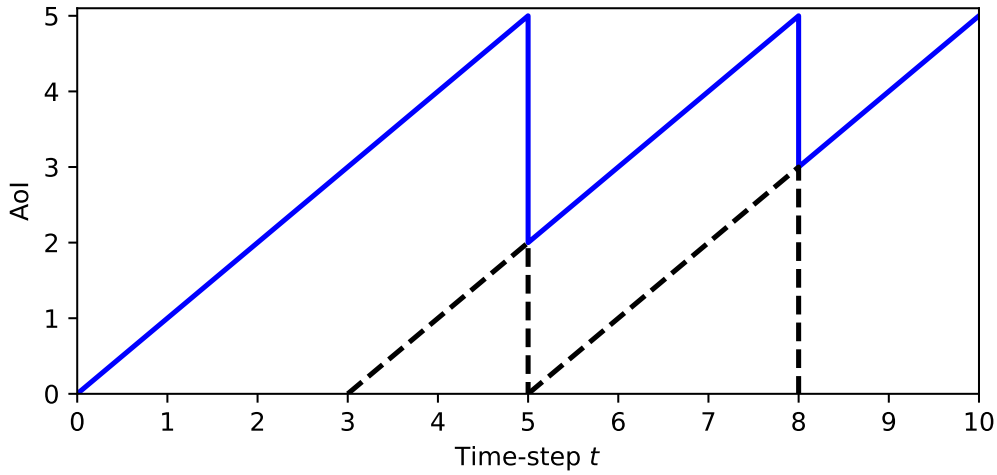

Fig. 4.4 An example of the evolution of AoI at the destination d, $\Delta_d(t)$.

4.3.3 Problem Formulation

VNF Placement Constraint. Each update requests $u \in \boldsymbol{U}_t$ consists of a set of VNFs $f_1, f_2, ..., f_n$ that needs to be placed over the NFV-enabled network. The following constraint in Eq. (4.6) ensures each VNF $f \in u$ is only placed on an NFV node m at time-step t.

$$\sum_{m \in \boldsymbol{M}} \sum_{f \in u} x_{m,t}^{f} = 1, \forall u \in \boldsymbol{U}_t \tag{4.6}$$

Node and Link Resource Capacities Constraints. The following two constraints in Eq. (4.7) and (4.8) illustrate that the aggregated CPU core and memory resource demand of all VNFs placed on NFV node m should not exceed the resource capacity of NFV node m at any time-step. Similarly, the third constraint, i.e., Eq. (4.9) illustrates that the aggregated bandwidth demand of all updates passing through link e should not exceed the bandwidth capacity of link e at any time step.

$$\sum_{u \in \boldsymbol{U}_t} \sum_{f \in u} x_{m,t}^{f} C_f^{core} \leq C_m^{core}, \forall m \in \boldsymbol{M}, t \tag{4.7}$$

$$\sum_{u \in \boldsymbol{U}_t} \sum_{f \in u} x_{m,t}^{f} C_f^{mem} \leq C_m^{mem}, \forall m \in \boldsymbol{M}, t \tag{4.8}$$

$$\sum_{u \in \boldsymbol{U}_t} y_{e,t}^{u} C_u^{bw} \leq C_e^{bw}, \forall e \in \boldsymbol{E}, t \tag{4.9}$$

End-to-End Delay Constraint. The following constraint in Eq. (4.10) ensures the QoS guarantee of each update request $u \in \boldsymbol{U}_t$. Here, we mainly focus on a SLA requirement of transmission delay. Let L_e and L_f be the transmission delay of link e and the processing

delay of VNF f, respectively. For each update request u, its end-to-end delay through the system should not exceed its maximum tolerated delay ϕ_r.

$$\sum_{f \in u} \sum_{m \in \boldsymbol{M}} x_{m,t}^{f} L_f + \sum_{e \in \boldsymbol{E}} y_{e,t}^{u} L_e \leq \phi_u, \forall u \in \boldsymbol{U}_t \tag{4.10}$$

In this paper, our objective is to minimize the long-term average AoI at the destination, $\Delta_{d,}$, under the constraints of Eq. (4.6)-(4.10), i.e.,

$$\begin{aligned} \textbf{min} \quad & \lim_{t \to +\infty} \Delta_{d,t} \\ \textbf{s.t.} \quad & \text{Eq. (4.6), (4.7), (4.8), (4.9), (4.10).} \end{aligned} \tag{4.11}$$

4.4 Algorithm Design

In this section, we introduce the design of our proposed VNF-AoI approach. First, we formulate the problem as Markov Decision Process (MDP) model. Then, we describe the design of DDPG.

4.4.1 Setup

We propose an intelligent approach, VNF-AoI, to optimize the VNF placement with dynamic network input. In this section, we start by following a standard DRL-based framework setup to describe our design of VNF-AoI, where a learning agent interacts with an environment at discrete time steps and consequently provides an optimal VNF placement based on the experience it has learned. The interaction process is a decision-making process that can be formulated as a MDP model. In particular, the MDP model is illustrated as follows. At each time-step $t \in \boldsymbol{T}$, the agent observes a state o_t from the environment E and identifies an action a_t based on a policy $\pi(o_t)$. The action a_t, then, is performed in the environment E and the agent consequently obtains a new state o_{t+1} with the corresponding reward r_t, which is used to evaluate the quality of the action a_t. The essential parameters used for the interaction process in MDP are described by a three-tuple $(\mathscr{O}, \mathscr{A}, \mathscr{R})$, referring to the state space, action space and reward, respectively.

State Space $\mathscr{O}$. The state space can be described by $\mathscr{O} = \{o_t, \forall t\}$, where each $o_t \in \mathscr{O}$ includes the information of all network resources of all NFV nodes and links and incoming VNF profile. To reduce the dimension of the state space and computational complexity, we normalize the state of each NFV node m and incoming VNF f at time-step t into a small

range (from 0 to 1), denoted by,

$$\omega_{m,t} = \frac{1}{2}\omega_{m,t}^{core} + \frac{1}{2}\omega_{m,t}^{mem}, \tag{4.12}$$

$$\omega_{f,t} = \frac{1}{2}\frac{C_f^{core}}{C_{f,max}^{core}} + \frac{1}{2}\frac{C_f^{mem}}{C_{f,max}^{mem}}, \tag{4.13}$$

where $C_{f,max}^{core} = max(C_f^{core}), \forall f \in \boldsymbol{F}$ and $C_{f,max}^{mem} = max(C_f^{mem}), \forall f \in \boldsymbol{F}$. Next, the o_t can be formulated as

$$o_t = \{\omega_{m,t}, \omega_{e,t}^{bw}, \omega_{f,t}, \forall m \in \boldsymbol{M}, e \in \boldsymbol{E}, \}. \tag{4.14}$$

Action Space $\mathscr{A}$. The action space shall include all possible placements for each incoming VNF f at time-step t. Thus the size of $\mathscr{A}$ is equal to the number of NFV nodes, and each action $a_t \in \mathscr{A}$ at time-step t can be defined as $a_t = m$, where $m \in \boldsymbol{M}$ represents the NFV node where the incoming VNF f should be placed.

Reward $\mathscr{R}$. The reward is used to reflect the influence of action a_t conducting in the state o_t, denoted by a function $r(s_t, a_t)$. Recall our objective is to minimize the long-term average AoI of the destination d. Thus, if action a_t can get good performance in the reduction of Eq. (4.11), the reward r_t should be associated with a higher value. On the contrary, we set r_t to a small value. Based on the above discussion, we define the reward of action a_t as follows,

$$r_t = r(s_t, a_t) = \begin{cases} -\Delta_{d,t+1}, & \text{if action } a_t \text{ is feasible} \\ -\infty, & \text{otherwise.} \end{cases} \tag{4.15}$$

4.4.2 Deep Reinforcement Learning Algorithm

Overview. The constraints of the problem and Eq. (4.14) indicate that the state space $\mathscr{S}$ is high-dimension. The traditional DRL algorithm, DQN, has been proved in poor performance while coping with complex problems having high-dimension state space. Thus, we apply the Deep Deterministic Policy Gradient (DDPG) algorithm to solve the VNF placement problem. DDPG, as a novel DRL approach that combines the advantages of policy gradient and DQN, is to establish an optimal strategy policy π in providing VNF placement decision at runtime, i.e., $\pi : \mathscr{S} \rightarrow \mathscr{A}$. In DDPG, the agent is based on an Actor-Critic framework containing two Deep Neural Networks (DNN)-based networks, an actor and a critic. For an input environment state s_t, the actor makes an action decision according to a policy π and the critic uses a Q function to value each pair of state actions.

Actor-Critic Network. The actor π trains for a state-action policy function $\pi(o_t|\theta^\pi)$ with parameters θ^π. The function $\pi(o_t|\theta^\pi)$ is a deterministic policy, which receives as input

Algorithm 4 DDPG Algorithm

Initialize critic network $Q(o_t, a_t|\theta^Q)$ and actor network $\pi(o_t|\theta^\pi)$ with parameters θ^Q and θ^π, respectively
Initialize target networks $\bar{Q}(o_t, a_t|\theta^{\bar{Q}})$ and $\bar{\pi}(o_t|\theta^{\bar{\pi}})$ with parameters $\theta^{\bar{Q}} \leftarrow \theta^Q$ and $\theta^{\bar{\pi}} \leftarrow \theta^\pi$, respectively
Initialize replay buffer $\mathscr{D}$
for episode from 1 to the number of episodes **do**
    Initialize a random process $\mathscr{N}$ for action exploration
    Receive initial observation state s_t
    for cycle from 1 to the number of cycles **do**
        Choose action a_t based on the current policy π, i.e.,

$$a_t = \pi(o_t|\theta^\pi) + \mathscr{N}_t$$

        Execute the action a_t, observe a new state o_{t+1} and calculate Eq. (4.15) to get the corresponding reward

$$r_t = r(s_t, a_t)$$

        Store the transition $(o_t, a_t, r_t, o_{t+1}) \rightarrow \mathscr{D}$
        Sample a minibatch of N transitions $(o_i, a_i, r_i, o_{i+1}) \leftarrow \mathscr{D}$
        Set $y_i = r_i + \gamma\bar{\pi}(s_{t+1}|\theta^{\bar{\pi}})|\theta^{\bar{Q}}$
        Update the critic by minimizing the loss:

$$L(\theta^Q) = \frac{1}{N}\sum_i^N (y_i - Q(o_i, a_i|\theta^Q))^2$$

        Update the actor policy using the sampled policy gradient:

$$J(\theta^\pi) = \mathbb{E}[Q(o_t, \pi(o_t|\theta^\pi)|\theta^Q)]$$

        Update target networks:

$$\theta^{\bar{Q}} = \alpha\theta^Q + (1-\alpha)\theta^{\bar{Q}}, \theta^{\bar{\pi}} = \alpha\theta^\pi + (1-\alpha)\theta^{\bar{\pi}}$$

    end for
end for

of the current state o_t and outputs an exact action a_t for the agent, denoted by $a_t = \pi(o_t)$. The critic Q trains for an action-value Q function $Q(o_t, a_t|\theta^Q)$, with parameters θ^Q. The function $Q(o_t, a_t|\theta^Q)$ reflects the expected return if following a deterministic policy π and taking an action a_t in state s_t, denoted by

$$Q(o_t, a_t|\theta^Q) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma Q(s_{t+1}, \pi(s_{t+1}))], \tag{4.16}$$

where $\gamma \in [0, 1)$ is a factor discounting future rewards.

To accelerate training process, both the critic Q and the actor π are established with a replica: 1) actor target $\bar{\pi}$ with parameter $\theta^{\bar{\pi}}$ and 2) critic target $\bar{Q}$ with parameter $\theta^{\bar{Q}}$.

Training. DDPG trains the agent by iteratively updating the parameters of those networks $\pi, \bar{\pi}, Q$ and $\bar{Q}$ in episodes. There are multiple sequential time-steps in each episode. At each time-step t, the agent first calculates an action a_t by adding the output of the policy $\pi(s_t)$ with a noise. After executing a_t, the agent obtains the next state s_{t+1} from the environment and immediate feedback r_t by Eq. (4.15). The transition comprises (s_t, a_t, r_t, s_{t+1}) which will be collected and stored in the experience pool $\mathscr{D}$. Then, the agent samples a mini-batch experiences $\mathscr{M}$ from $\mathscr{D}$ and updates the parameters of the critic networks in the direction of $\nabla_{\theta^Q} J(\theta^Q)$ by minimizing the loss $L(\theta^Q)$ using collected experiences from experience pool $\mathscr{D}$, where

$$L(\theta^Q) = \frac{1}{|\mathscr{M}|} \sum_i (y_i - Q(o_i, a_i|\theta^Q))^2 \tag{4.17}$$

and

$$y_i = r_i + \gamma \bar{Q}(o_{t+1}, \bar{\pi}(s_{t+1}|\theta^{\bar{\pi}})|\theta^{\bar{Q}}). \tag{4.18}$$

The agent updates the parameters of the actor based on policy gradient. The parameters of actor's network, π, is updated in the direction of $\nabla_{\theta^\pi} J(\theta^\pi)$, where ∇_{θ^π} denotes the derivative of θ^π and $J(\theta^\pi)$ denotes the policy objective function:

$$J(\theta^\pi) = \mathbb{E}[Q(o_t, \pi(o_t|\theta^\pi)|\theta^Q)] \tag{4.19}$$

After updating π and Q by using Eq. (4.19) and (4.17), the agent finally applies a soft update to the target network $\bar{\pi}$ and $\bar{Q}$, i.e.,

$$\theta^{\bar{Q}} = \tau\theta^Q + (1-\tau)\theta^{\bar{Q}}, \theta^{\bar{\pi}} = \tau\theta^\pi + (1-\tau)\theta^{\bar{\pi}}. \tag{4.20}$$

where $\tau \ll 1$ controls the updating amplitude.

Table 4.2 Parameter for training DRL agent

Hyper-parameter	Value
Episodes	3000
Number of Time Steps, $\|\boldsymbol{T}\|$	1000
Actor/Critic Network Learning Rate	0.001
Discount Factor, γ	0.9
Size of Mini-batch, $\|\mathscr{M}\|$	36
Capacity of Experience Pool, $\|\mathscr{D}\|$	10000
Soft Replacement Tau, τ	0.01

4.5 Evaluation

In this section, we conduct extensive simulation experiments to evaluate the effectiveness of our proposed approach in several realistic network topologies.

4.5.1 Simulation Setup

Simulation Platform. We construct a Python-based simulation environment and implement our proposed VNF-AoI approach using the PyTorch machine learning framework. All simulations are conducted on the Google Colab platform[1]. The hyper-parameters selected for this simulation are shown in Table 4.2.

Network Topology. Our experiments are conducted on several network topologies of different scales from the Internet Topology Zoo[2] [36], such as (a) Gridnet (9 nodes and 20 links), (b) Ans (18 nodes and 25 links), (c) BtEurope (24 nodes and 37 links) and (d) AttMpls (25 nodes and 56 links). Each computing node m has a number of CPU cores, ranging from 10 to 15. The memory capacity of each computing node m ranges from 6 GB to 8 GB.

Status Update. As aforementioned in Sec. 4.3.1, the definition of a status update u contains a source s, a SFC $f_1, f_2, ..., f_n$, and a destination d. For each update u, the source s is generated by randomly picking a node $s \in \boldsymbol{S}$, and the destination d is set to a given node $m' \in \boldsymbol{M}$ by default. We generate a SFC by randomly picking VNFs from ten VNF types and each type of VNF has its own usage of CPU core ranging from 1 to 5, and usage of bandwidth ranging from 0.5 to 1 GB.

Baseline Algorithms. We design two baseline algorithms for comparison. The details are as follows. **First-Fit Algorithm (FFA)**: for each incoming VNF, FFA first filters out all available NFV nodes that satisfy resource capacities constraints in Eq. (4.7) and (4.8) as a set

[1]Google Colab: https://colab.research.google.com/

[2]Internet Topology Zoo: http://www.topology-zoo.org/

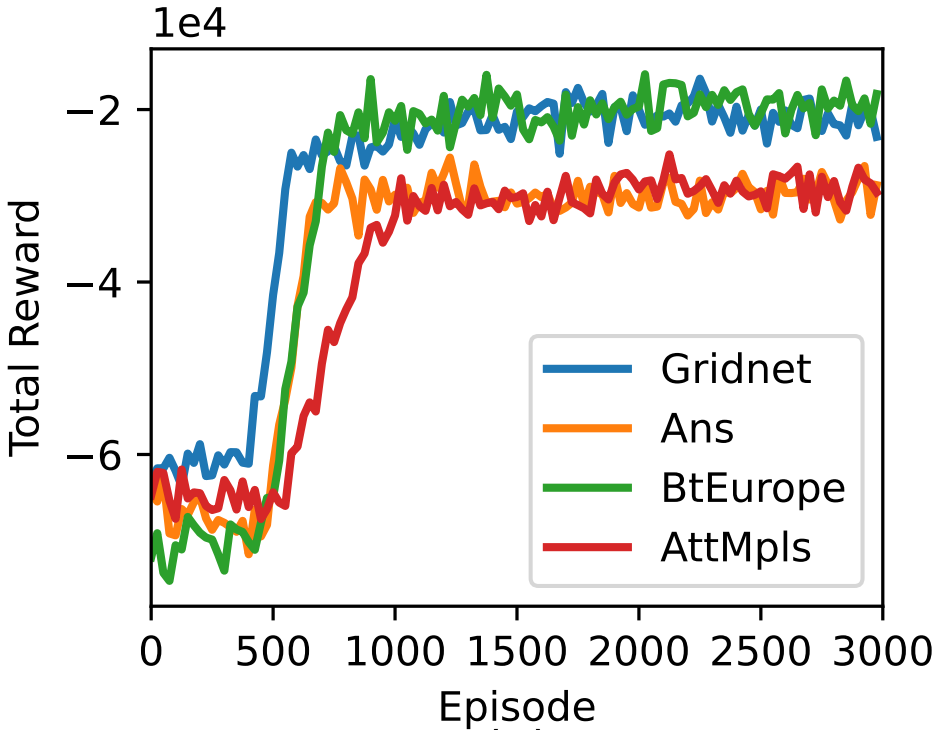

Fig. 4.5 The total reward for our proposed VNF_AoI under different network topologies.

$M' \subseteq M$. The first node in M' is always chosen by FFA to place VNF. **Random Placement Algorithm (RPA)**: by the same way as FFA, RPA gets a candidate node set $M' \subseteq M$. Then, RFA randomly chooses one from M' to allocate the VNF on it.

4.5.2 Performance Evaluation

Training Efficiency. To show the training efficiency, we train VNF_AoI in different network topologies with a different number of nodes and links. Fig. 4.5 plots the average total reward when training VNF_AoI under different network topologies. At the beginning of training, our VNF_AoI shows poor performance on total reward. This happens due to the penalty caused by a low update request acceptance rate. Then, with the increase in the number of episodes, the reward increases until it reaches a relatively stable level. As a result, the reward converges at about 700 episodes for Gridnet, and at about 1250 episodes for AttMpls. We can also observe that VNF_AoI takes more episodes to converge for more complicated network topology, e.g., 300 episodes for Gridnet and 750 episodes for AttMpls.

Acceptance Ratio. We compare the service acceptance ratio of all three algorithms for different network topologies. Fig. 4.6 shows the results of the acceptance ratio in 1000 time steps. Note that VNF_AoI has been trained in 3000 episodes while each episode contains

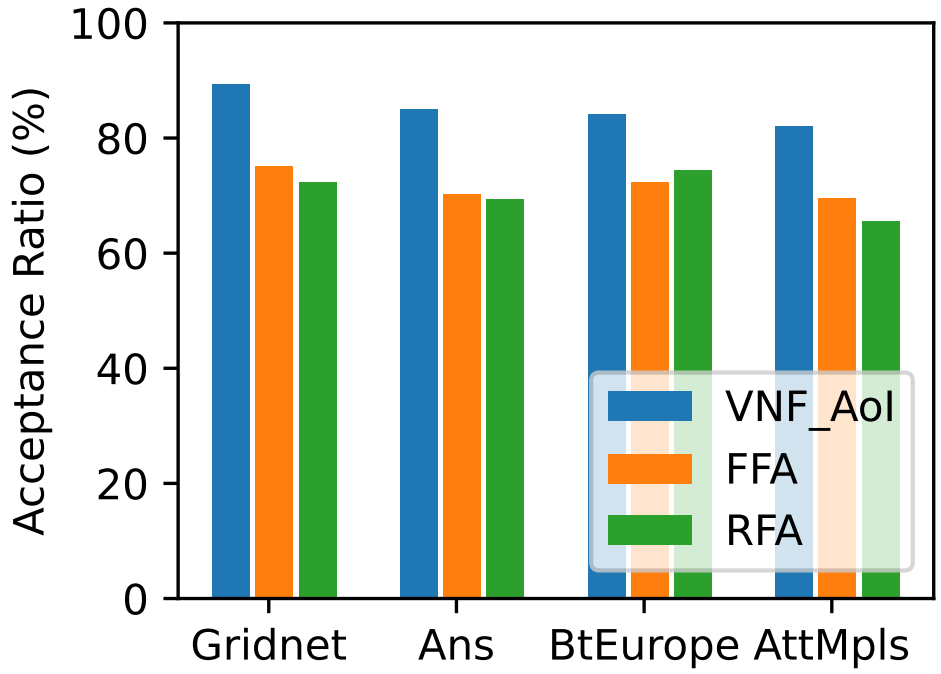

Fig. 4.6 The average SFC acceptance ratio of different algorithm.

Table 4.3 Average AoI of different algorithms

	VNF_AoI	**FFA**	**RFA**
Gridnet	24.3	27 (10%)	27 (10%)
Ans	32.5	39.7 (18.1%)	40.9 (20.5%)
BtEurope	24.4	31.2 (21.7%)	32.1 (23.9%)
AttMples	29.4	36.3 (19%)	37.3 (21.1%)

1000 time steps. As a result, our VNF_AoI is always above the baseline, which averagely outperforms FFA and RPA by 12.8% and 14.7%, respectively. It indicates that our VNF_AoI could make VNF placement decisions more intelligently and reasonably, thereby deploying more update requests.

Long-term Average AoI.Finally, we explore the AoI evolution of VNF_AoI as compared with other baseline algorithms in 1000 time steps. As shown in Fig. 4.7, at most time-steps, our VNF_AoI can achieve lower AoI compared with FFA and RPA. Calculated by Eq. (4.5), we can get the average AoI of three algorithms in Table 4.3, where improved percentage is shown in brackets. In summary, our VNF_AoI can averagely outperform other baseline algorithms by 20.3%.

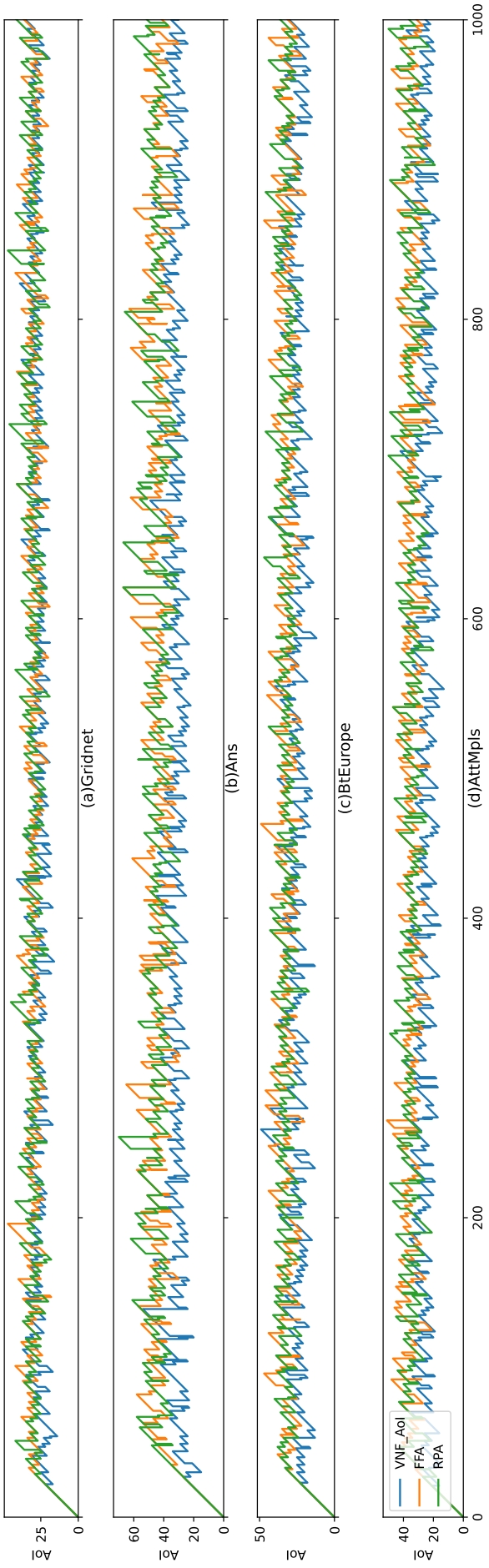

Fig. 4.7 The AoI evolution of three algorithms under four different network topologies.

4.6 Conclusion

In this research, we focus on the VNF placement problem aiming at optimizing the average AoI of the destination in an NFV-enabled network. According to our survey, most of the existing works solve VNF placement with the goal of optimizing traditional metrics, like end-to-end packet delay, resource utilization, expense, energy consumption, etc., rather than AoI. Therefore, an AoI-aware VNF Placement approach, called VNF_AoI, has been proposed to provide optimal online VNF placement policies to minimize the long-term average AoI at the destination. Finally, we conduct extensive simulations on four network topologies to validate the effectiveness of our proposed VNF_AoI. Related results clearly demonstrate that our VNF-AoI can significantly reduce average AoI by about 20.3%.

Chapter 5

Conclusions and Future Works

5.1 Conclusions

In this dissertation, the author proposed three research to scale the NFV network at three levels: server level, service chain level, and traffic level.

At the hardware level, the author presents HyScaler, a dynamic and hybrid VNF scaling system that enables the construction of elastic SFCs. HyScaler leverages the benefits of VS, HSC, and HSS to effectively scale VNF instances based on the changing network traffic demands. The key features of HyScaler include: A monitoring module that detects overloaded VNFIs and triggers the scaling process, A scaling module that makes local scaling decisions (VS and HSC) or sends scaling requests to a global orchestrator, and A global orchestrator that runs the proposed GVPC algorithm to make optimal scaling decisions, including scaling VNFIs across multiple servers. The author implemented a prototype of HyScaler on the OpenNetVM platform and conducted extensive experiments to validate its scalability and the effectiveness of the GVPC algorithm. The results show that HyScaler can improve the performance of VNFIs by about 1.02 times compared to the original NFV platform.

At the VNF level, the author formulated the SFC embedding problem within the context of a Network Functions Virtualization (NFV)-based 5G core network. Due to the high complexity of the problem, they proposed a low-complexity approach that incorporates three heuristic algorithms: Cost Efficiency Factor Priority Sorting, K-Cost Minimizing Path, and Delay-Aware VNF Embedding. The numerical results validate the performance of the proposed approach in reducing computational complexity while achieving near-optimal solutions for the SFC embedding problem.

At the traffic level, the author proposed a DRL-based approach called VNF-AoI to address the VNF placement problem in an NFV-enabled multiple-source updating system. The goal is to minimize the long-term average AoI at the destination, which is a crucial

performance metric for real-time applications that rely on fresh data. The author formulated the VNF placement problem as an MDP and solve it using the DDPG algorithm, a DRL-based technique. This approach allows the system to learn an optimal VNF placement policy through interaction with the environment, overcoming the limitations of heuristic-based algorithms that struggle with dynamic workloads. The author conducted extensive simulations to validate the effectiveness of their proposed VNF-AoI approach, demonstrating that it outperforms other baseline algorithms in terms of acceptance ratio and average AoI at the destination. The results show that VNF-AoI can significantly reduce the average AoI by about 20.3% compared to other methods.

5.2 Future Works

This dissertation constructs an efficient network function scaling and placement in NFV for beyond 5G and future 6G areas. While significant progress has been made, several areas for future research require further exploration:

- **Scale-in VNF**: In the first research, a system was designed to scale out a VNF when it becomes overloaded or "hot." However, the challenge of scaling in a VNF when its load decreases or "cools down" remains unaddressed. Scaling in is crucial for optimizing resource consumption and cost efficiency. Future work about scaling-in should be explored.

- **Diverse QoS requirement**: In the second research, I optimize the QoS on terms of delay and throughput. However, with the advent of B5G and 6G networks, the spectrum of QoS requirements is expected to expand significantly, incorporating new dimensions such as reliability, energy efficiency, and ultra-low latency. Exploring new solutions to address these diverse QoS remains a critical challenge.

- **Handling Multiple User Devices**: In the third research, a strategy was developed to route traffic to a single device while minimizing the Age of Information (AoI). However, real-world networks involve multiple devices connecting simultaneously. In such scenarios, the challenge lies in routing traffic for multiple devices while ensuring their AoI remains optimized. Future research direction will be developing efficient scheduling mechanisms to minimize the overall AoI for all devices.

References

[1] (2020). *White Paper: Myth-busting DPDK in 2020*. The Linux Foundation. https://nextgeninfra.io/dpdk-myth-busting-2020/#js-home-hero.

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

[3] Barbette, T., Soldani, C., and Mathy, L. (2015). Fast userspace packet processing. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 5–16.

[4] Chen, M., Sun, Y., Hu, H., Tang, L., and Fan, B. (2020a). Energy-saving and resource-efficient algorithm for virtual network function placement with network scaling. *IEEE Transactions on Green Communications and Networking*, 5(1):29–40.

[5] Chen, M., Sun, Y., Hu, H., Tang, L., and Fan, B. (2020b). Energy-saving and resource-efficient algorithm for virtual network function placement with network scaling. *IEEE Transactions on Green Communications and Networking*, 5(1):29–40.

[6] Chen, W., Wang, Z., Zhang, H., Yin, X., and Shi, X. (2021). Cost-efficient dynamic service function chain embedding in edge clouds. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 310–318. IEEE.

[7] Chen, Y. and Wu, J. (2020). Latency-efficient vnf deployment and path routing for reliable service chain. *IEEE Transactions on Network Science and Engineering*, 8(1):651–661.

[8] Dalgkitsis, A., Mekikis, P.-V., Antonopoulos, A., Kormentzas, G., and Verikoukis, C. (2020). Dynamic resource aware vnf placement with deep reinforcement learning for 5g networks. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE.

[9] Dobrescu, M., Argyraki, K., and Ratnasamy, S. (2012). Toward predictable performance in software Packet-Processing platforms. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 141–154.

[10] Dong, J., Ota, K., and Dong, M. (2022). Why vr games sickness? an empirical study of capturing and analyzing vr games head movement dataset. *IEEE MultiMedia*, 29(2):74–82.

[11] Duan, J., Wu, C., Le, F., Liu, A. X., and Peng, Y. (2017). Dynamic scaling of virtualized, distributed service chains: A case study of ims. *IEEE Journal on Selected Areas in Communications*, 35(11):2501–2511.

[12] Dwaraki, A. and Wolf, T. (2016). Adaptive service-chain routing for virtual network functions in software-defined networks. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, pages 32–37.

[13] Dwiardhika, D. and Tachibana, T. (2018). Cost efficient vnf placement with optimization problem for security-aware virtual networks. In *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pages 1–3. IEEE.

[14] ETSI, G. (2014). Network functions virtualization (nfv); architectural framework. *ETsI Gs NFV*, 2:V1.

[15] Fan, Q., Pan, P., Li, X., Wang, S., Li, J., and Wen, J. (2022a). Drl-d: Revenue-aware online service function chain deployment via deep reinforcement learning. *IEEE Transactions on Network and Service Management*, 19(4):4531–4545.

[16] Fan, X., Zhao, G., Tu, H., Xu, H., and Huang, H. (2022b). Mascot: Mobility-aware service function chain routing in mobile edge computing. In *2022 19th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 461–469. IEEE.

[17] Fei, X., Liu, F., Xu, H., and Jin, H. (2018). Adaptive vnf scaling and flow routing with proactive demand prediction. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 486–494. IEEE.

[18] Gao, T., Li, X., Wu, Y., Zou, W., Huang, S., Tornatore, M., and Mukherjee, B. (2020a). Cost-efficient vnf placement and scheduling in public cloud networks. *IEEE Transactions on Communications*, 68(8):4946–4959.

[19] Gao, T., Li, X., Wu, Y., Zou, W., Huang, S., Tornatore, M., and Mukherjee, B. (2020b). Cost-efficient vnf placement and scheduling in public cloud networks. *IEEE Transactions on Communications*, 68(8):4946–4959.

[20] Ghaznavi, M., Shahriar, N., Kamali, S., Ahmed, R., and Boutaba, R. (2017). Distributed service function chaining. *IEEE Journal on Selected Areas in Communications*, 35(11):2479–2489.

[21] Haeffner, W., Napper, J., Stiemerling, M., Lopez, D., and Uttaro, J. (2019). Service function chaining use cases in mobile networks. *Internet Engineering Task Force*.

[22] Han, Z., Yang, Y., Wang, W., Zhou, L., Nguyen, T. N., and Su, C. (2022). Age efficient optimization in uav-aided vec network: a game theory viewpoint. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):25287–25296.

[23] Hantouti, H., Benamar, N., and Taleb, T. (2020). Service function chaining in 5g & beyond networks: Challenges and open research issues. *IEEE Network*, 34(4):320–327.

[24] Harutyunyan, D., Shahriar, N., Boutaba, R., and Riggio, R. (2022). Latency and mobility–aware service function chain placement in 5g networks. *IEEE Transactions on Mobile Computing*, 21(5):1697–1709.

[25] Hu, G., Li, Q., Ai, S., Chen, T., Duan, J., and Wu, Y. (2020). A proactive auto-scaling scheme with latency guarantees for multi-tenant nfv cloud. *Computer Networks*, 181:107552.

[26] Hwang, J., Ramakrishnan, K. K., and Wood, T. (2015). Netvm: High performance and flexible networking using virtualization on commodity platforms. *IEEE Transactions on Network and Service Management*, 12(1):34–47.

[27] IBM (2023). Ibm ilog cplex optimizer. website. https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer.

[28] Jia, Y., Wu, C., Li, Z., Le, F., and Liu, A. (2018). Online scaling of nfv service chains across geo-distributed datacenters. *IEEE/ACM Transactions on Networking*, 26(2):699–710.

[29] Jin, P., Fei, X., Zhang, Q., Liu, F., and Li, B. (2020). Latency-aware vnf chain deployment with efficient resource reuse at network edge. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 267–276. IEEE.

[30] Jonsson, P. (2024). Ericsson mobility report. *Nov*.

[31] Kadota, I., Sinha, A., Uysal-Biyikoglu, E., Singh, R., and Modiano, E. (2018a). Scheduling policies for minimizing age of information in broadcast wireless networks. *IEEE/ACM Transactions on Networking*, 26(6):2637–2650.

[32] Kadota, I., Sinha, A., Uysal-Biyikoglu, E., Singh, R., and Modiano, E. (2018b). Scheduling policies for minimizing age of information in broadcast wireless networks. *IEEE/ACM Transactions on Networking*, 26(6):2637–2650.

[33] Katti, R. and Prince, S. (2019). A survey on role of photonic technologies in 5g communication systems. *Photonic Network Communications*, 38:185–205.

[34] Kaul, S., Gruteser, M., Rai, V., and Kenney, J. (2011). Minimizing age of information in vehicular networks. In *2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 350–358. IEEE.

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

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

[37] Le, S., Wu, Y., Guo, Y., and Del Vecchio, C. (2021). Game theoretic approach for a service function chain routing in nfv with coupled constraints. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(12):3557–3561.

[38] Li, D., Ota, K., Zhong, Y., Dong, M., Tang, Y., and Qiu, J. (2019a). Towards high-efficient transaction commitment in a virtualized and sustainable rdbms. *IEEE Transactions on Sustainable Computing*, 6(3):507–521.

[39] Li, G., Feng, B., Zhou, H., Zhang, Y., Sood, K., and Yu, S. (2020). Adaptive service function chaining mappings in 5g using deep q-learning. *Computer Communications*, 152:305–315.

[40] Li, H., Ota, K., and Dong, M. (2018). Virtual network recognition and optimization in sdn-enabled cloud environment. *IEEE Transactions on Cloud Computing*, 9(2):834–843.

[41] Li, H., Ota, K., and Dong, M. (2019b). Ls-sdv: Virtual network management in large-scale software-defined iot. *IEEE Journal on Selected Areas in Communications*, 37(8):1783–1793.

[42] Li, J., Shi, W., Ye, Q., Zhang, N., Zhuang, W., and Shen, X. (2021). Multiservice function chain embedding with delay guarantee: A game-theoretical approach. *IEEE Internet of Things Journal*, 8(14):11219–11232.

[43] Lin, R., Yu, S., Luo, S., Zhang, X., Wang, J., and Zukerman, M. (2021). Column generation based service function chaining embedding in multi-domain networks. *IEEE Transactions on Cloud Computing*, 11(1):185–199.

[44] Liu, H., Long, S., Li, Z., Fu, Y., Zuo, Y., and Zhang, X. (2022a). Revenue maximizing online service function chain deployment in multi-tier computing network. *IEEE Transactions on Parallel and Distributed Systems*, 34(3):781–796.

[45] Liu, L., Xu, H., Niu, Z., Li, J., Zhang, W., Wang, P., Li, J., Xue, J. C., and Wang, C. (2022b). Scaleflux: Efficient stateful scaling in nfv. *IEEE Transactions on Parallel and Distributed Systems*.

[46] Luo, J., Li, J., Jiao, L., and Cai, J. (2020). On the effective parallelization and near-optimal deployment of service function chains. *IEEE Transactions on Parallel and Distributed Systems*, 32(5):1238–1255.

[47] Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R., and Huici, F. (2014). Clickos and the art of network function virtualization. In *11th USENIX symposium on networked systems design and implementation (NSDI 14)*, pages 459–473.

[48] Palkar, S., Lan, C., Han, S., Jang, K., Panda, A., Ratnasamy, S., Rizzo, L., and Shenker, S. (2015). E2: A framework for nfv applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 121–136.

[49] Papathanail, G., Pentelas, A., and Papadimitriou, P. (2021). Towards fine-grained resource allocation in nfv infrastructures. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE.

[50] Pham, Q. T. A., Bradai, A., Singh, K. D., and Hadjadj-Aoul, Y. (2019). Multi-domain non-cooperative vnf-fg embedding: A deep reinforcement learning approach. In *INFOCOM 2019-IEEE International Conference on Computer Communications*, pages 1–6. IEEE.

[51] Qin, X., Ma, T., Tang, Z., Zhang, X., Zhou, H., and Zhao, L. (2023). Service-aware resource orchestration in ultra-dense leo satellite-terrestrial integrated 6g: A service function chain approach. *IEEE Transactions on Wireless Communications*, pages 1–1.

[52] Rahman, S., Ahmed, T., Huynh, M., Tornatore, M., and Mukherjee, B. (2018). Auto-scaling vnfs using machine learning to improve qos and reduce cost. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE.

[53] Rizzo, L. (2012). netmap: a novel framework for fast packet i/o. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 101–112.

[54] Sairam, R., Bhunia, S. S., Thangavelu, V., and Gurusamy, M. (2019). Netra: Enhancing iot security using nfv-based edge traffic analysis. *IEEE Sensors Journal*, 19(12):4660–4671.

[55] Sedaghat, M., Hernandez-Rodriguez, F., and Elmroth, E. (2013). A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, pages 1–10.

[56] Series, M. (2015). Imt vision–framework and overall objectives of the future development of imt for 2020 and beyond. *Recommendation ITU*, 2083.

[57] Solozabal, R., Ceberio, J., Sanchoyerto, A., Zabala, L., Blanco, B., and Liberal, F. (2019). Virtual network function placement optimization with deep reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 38(2):292–303.

[58] Soualah, O., Mechtri, M., Ghribi, C., and Zeghlache, D. (2017). Energy efficient algorithm for vnf placement and chaining. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 579–588. IEEE.

[59] Soualah, O., Mechtri, M., Ghribi, C., and Zeghlache, D. (2018). A green vnf-fg embedding algorithm. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 141–149. IEEE.

[60] Srivastava, A., Sinha, A., and Jagannathan, K. (2019). On minimizing the maximum age-of-information for wireless erasure channels. In *2019 International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*, pages 1–6. IEEE.

[61] Sun, C., Bi, J., Zheng, Z., Yu, H., and Hu, H. (2017). Nfp: Enabling network function parallelism in nfv. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 43–56.

[62] Tang, H., Zhou, D., and Chen, D. (2018). 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.

[63] Tao, X., Ota, K., Dong, M., Qi, H., and Li, K. (2021). Congestion-aware scheduling for software-defined sag networks. *IEEE Transactions on Network Science and Engineering*, 8(4):2861–2871.

[64] Thiruvasagam, P. K., Kotagi, V. J., and Murthy, C. S. R. (2022). A reliability-aware, delay guaranteed, and resource efficient placement of service function chains in softwarized 5g networks. *IEEE Transactions on Cloud Computing*, 10(3):1515–1531.

[65] Toosi, A. N., Son, J., Chi, Q., and Buyya, R. (2019). Elasticsfc: Auto-scaling techniques for elastic service function chaining in network functions virtualization-based clouds. *Journal of Systems and Software*, 152:108–119.

[66] Tu, W., Wei, Y.-H., Antichi, G., and Pfaff, B. (2021). Revisiting the open vswitch dataplane ten years later. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 245–257.

[67] Wang, G., Zhou, S., Zhang, S., Niu, Z., and Shen, X. (2020a). Sfc-based service provisioning for reconfigurable space-air-ground integrated networks. *IEEE Journal on Selected Areas in Communications*, 38(7):1478–1489.

[68] Wang, Q., Kang, M., Wu, G., Ren, Y., and Su, C. (2020b). A practical secret key generation scheme based on wireless channel characteristics for 5g networks. *IEICE TRANSACTIONS on Information and Systems*, 103(2):230–238.

[69] Woo, S., Sherry, J., Han, S., Moon, S., Ratnasamy, S., and Shenker, S. (2018). Elastic scaling of stateful network functions. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 299–312.

[70] Yang, L., Jia, J., Lin, H., and Cao, J. (2023). Reliable dynamic service chain scheduling in 5g networks. *IEEE Transactions on Mobile Computing*, 22(8):4898–4911.

[71] Yang, S., Li, F., Yahyapour, R., and Fu, X. (2019). Delay-sensitive and availability-aware virtual network function scheduling for nfv. *IEEE Transactions on Services Computing*.

[72] Yu, H., Chen, Z., Sun, G., Du, X., and Guizani, M. (2020). 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.

[73] Yu, H., Yang, J., and Fung, C. (2018a). Elastic network service chain with fine-grained vertical scaling. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE.

[74] Yu, H., Yang, J., Fung, C., Boutaba, R., and Zhuang, Y. (2018b). 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. IEEE.

[75] Yue, Y., Cheng, B., Liu, X., Wang, M., Li, B., and Chen, J. (2021). Resource optimization and delay guarantee virtual network function placement for mapping sfc requests in cloud networks. *IEEE Transactions on Network and Service Management*, 18(2):1508–1523.

[76] Zeng, C., Liu, F., Chen, S., Jiang, W., and Li, M. (2018). Demystifying the performance interference of co-located virtual network functions. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 765–773. IEEE.

[77] Zhai, D., Meng, X., Yu, Z., Hu, H., and Han, X. (2021). A fine-grained and dynamic scaling method for service function chains. *Knowledge-Based Systems*, 228:107289.

[78] Zhang, C., Dong, M., and Ota, K. (2020). Fine-grained management in 5g: Dql based intelligent resource allocation for network function virtualization in c-ran. *IEEE Transactions on Cognitive Communications and Networking*, 6(2):428–435.

[79] Zhang, Q., Liu, F., and Zeng, C. (2019a). Adaptive interference-aware vnf placement for service-customized 5g network slices. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 2449–2457.

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

[81] Zhang, W., Liu, G., Zhang, W., Shah, N., Lopreiato, P., Todeschi, G., Ramakrishnan, K., and Wood, T. (2016). Opennetvm: A platform for high performance network service chains. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, pages 26–31.

[82] Zhang, X., Wu, C., Li, Z., and Lau, F. C. (2017a). Proactive vnf provisioning with multi-timescale cloud resources: Fusing online learning and online optimization. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE.

[83] Zhang, Y., Anwer, B., Gopalakrishnan, V., Han, B., Reich, J., Shaikh, A., and Zhang, Z.-L. (2017b). Parabox: Exploiting parallelism for virtual network functions in service chaining. In *Proceedings of the Symposium on SDN Research*, pages 143–149.

[84] Zhang, Y., Zhang, Z.-L., and Han, B. (2019b). Hybridsfc: Accelerating service function chains with parallelism. In *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–7. IEEE.

[85] Zhou, W., Yang, Y., Xu, M., and Chen, H. (2019). Accommodating dynamic traffic immediately: A vnf placement approach. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE.

[86] Zhu, S., Ota, K., and Dong, M. (2022). Energy-efficient artificial intelligence of things with intelligent edge. *IEEE Internet of Things Journal*, 9(10):7525–7532.

Publications

Journals

1. Zhenke Chen, He Li, Kaoru Ota, Mianxiong Dong, “Profit-Maximizing Service Function Chain Embedding in NFV-based 5G Core Networks,” IEEE Transactions on Network Science and Engineering (TNSE), vol. 11, no. 6, pp. 6105-6117, Nov.-Dec. 2024.

2. Zhenke Chen, He Li, Kaoru Ota, Mianxiong Dong, "HyScaler: A Dynamic, Hybrid VNF Scaling System for Building Elastic Service Function Chains across Multiple Servers," IEEE Transactions on Network and Service Management (TNSM), vol. 20, no. 4, pp. 4803-4814, December 2023.

Proceeding of International Conference

1. Zhenke Chen, He Li, Kaoru Ota, Mianxiong Dong, "Deep Reinforcement Learning for AoI Aware VNF Placement in Multiple Source Systems," IEEE Global Communications Conference (GLOBECOM 2022), Rio de Janeiro, Brazil, December 4-8, 2022.