Efficient Resource Allocation for Throughput Maximization in Next-Generation Networks

Meitian Huang

March 2021

A thesis submitted for the degree of
Doctor of Philosophy at
the Australian National University

To Music,
for the encouragement and smiles she gave me.

Declaration

This thesis is a presentation of the original work except where otherwise stated. I completed this work jointly with my supervisor, Professor Weifa Liang. My contribution to the work is around 85%.

Meitian Huang
March 2021

Acknowledgments

This thesis is the result of three and a half years of work. During this period, I have been accompanied and supported by many people. I would like to express my gratitude to them.

First and foremost I would like to express my sincere gratitude to my supervisor, Professor Weifa Liang, for the continuous support of my Ph.D. study and research with his excellent guidance, patience, and immense knowledge. He taught and guided me to become a qualified computer scientist through expert supervision with his keen insight and valuable experience. I thank him for the systematic guidance and great effort he put into training me in the scientific field.

I would like to thank the other members in my supervisor panel, Professor Brendan McKay and Professor Song Guo for the support to make this thesis possible.

I wish to thank the people in the Research School of Computer Science at the Australian National University, for their generous help and assistance in various aspects. Janette Rawlinson, James Fellows, Elspeth Davies, Trina Merrell, Harriet King, and Christie Liu deserve to be especially appreciated.

I am grateful to my friends, Zichuan Xu, Qiufen Xia, Wenzheng Xu, Xiaojiang Ren, Narjess Afzaly, Mojtaba Rezvani, Yu Ma, Mike Jia, Dongxu Li, Junyi Xu, Haotian Chang, Yang Liu, Jing Li, Yanbo Li, Zhenyue Qin, Jing Xie, Haitao Li, Siyuan Zhang et al. for their great friendship and kind help during my years at the ANU.

I express my special thankfulness to my beloved wife Yuxi Sun, for her love, support, tolerance, and accompany in my life. I want to express the profound gratitude to my beloved parents, Guangde Huang and Wenjing Chai, and my younger brother, Xintian Huang, for their love and continuous support. Without their love and encouragement, this thesis could not have been completed.

Finally, thanks to the people who receive no thanks.

So long, and thanks for all the fish.

Publications

Journal Publications

[1] **Meitian Huang**, Weifa Liang, Xiaojun Shen, Yu Ma, and Haibin Kan. Reliability-Aware Virtualized Network Function Services Provisioning in Mobile Edge Computing. *IEEE Transactions on Mobile Computing*, Vol. 19, No. 11, pp. 2699–2713, 2020.

[2] **Meitian Huang**, Weifa Liang, Yu Ma, and Song Guo. Maximizing Throughput of Delay-Sensitive NFV-Enabled Request Admissions via Virtualized Network Function Placement. To appear in *IEEE Transactions on Cloud Computing*, Vol. XX, Acceptance date: May 2, 2019, DOI: 10.1109/TCC.2019.2915835.

[3] **Meitian Huang**, Weifa Liang, Zichuan Xu, Wenzheng Xu, Song Guo, and Yinlong Xu. Online Unicasting and Multicasting in Software-Defined networks. *Computer Networks*, Vol. 132, pp. 26–39, February 2018.

[4] **Meitian Huang**, Weifa Liang, Zichuan Xu, and Song Guo. Efficient Algorithms for Throughput Maximization in Software-Defined Networks with Consolidated Middleboxes. *IEEE Transactions on Network and Service Management*, Vol. 14, No. 3, pp. 631–645, July 2017.

[5] Yu Ma, Weifa Liang, **Meitian Huang**, Wenzheng Xu, and Song Guo. Virtual Network Function Service Provisioning in MEC via Trading Off the Usages Between Computing and Communication Resources. To appear in *IEEE Transactions on Cloud Computing*, Vol. XX, No. XX, Acceptance date: Dec. 5, 2020, DOI: 10.1109/TCC.2020.3043313.

[6] Jing Li, Weifa Liang, **Meitian Huang**, and Xiaohua Jia. Reliability-Aware Network Service Provisioning In Mobile Edge-Cloud Networks. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, Vol. 31, No. 7, pp. 1545–1558, 2020.

[7] Zichuan Xu, Weifa Liang, **Meitian Huang**, Mike Jia, Song Guo, and Alex Galis. Efficient NFV-Enabled Multicasting in SDNs. *IEEE Transactions on Communications*,

Vol. 67, No. 3, pp. 2052–2070, 2019.

[8] Mike Jia, Weifa Liang, **Meitian Huang**, Zichuan Xu, and Yu Ma. Routing Cost Minimization and Throughput Maximization of NFV-Enabled Unicasting in Software-Defined Networks. *IEEE Transactions on Network and Service Management*, Vol. 15, No. 2, pp. 732–745, June 2018.

[9] Mike Jia, Weifa Liang, Zichuan Xu, **Meitian Huang**, and Yu Ma. QoS-Aware Cloudlet Load Balancing in Wireless Metropolitan Area Networks. *IEEE Transactions on Cloud Computing*, Vol. 8, No. 2, pp. 623–634, 2020.

[10] Zichuan Xu, Weifa Liang, Mike Jia, **Meitian Huang**, and Guoqiang Mao. Task Offloading with Network Function Services in a Mobile Edge-Cloud Network. *IEEE Transactions on Mobile Computing*, Vol. 18, No. 11, pp. 2672–2685, 2019.

Conference Publications

[1] **Meitian Huang**, Weifa Liang, Yu Ma, and Song Guo. Throughput Maximization of Delay-Sensitive Request Admission via Virtualized Network Function Placements and Migrations. *Proceedings of 2018 IEEE International Conference on Communications (ICC)*, May 2018.

[2] **Meitian Huang** and Weifa Liang. Incremental SDN-Enabled Switch Deployment for Hybrid Software-Defined Networks. *Proceedings of 2017 IEEE International Conference on Computer Communications and Networks (ICCCN)*, July 2017.

[3] **Meitian Huang**, Weifa Liang, Zichuan Xu, Mike Jia, and Song Guo. Throughput Maximization in Software-Defined Networks with Consolidated Middleboxes. *Proceedings of 2016 IEEE International Conference on Local Computer Networks (LCN)*, pp. 298–306, November 2016.

[4] **Meitian Huang**, Weifa Liang, Zichuan Xu, Wenzheng Xu, Song Guo, and Yinlong Xu. Dynamic Routing for Network Throughput Maximization in Software-Defined Networks. *Proceedings of 2016 IEEE International Conference on Computer Communications (INFOCOM)*, pp. 2545–2553, April 2016.

[5] Jing Li, Weifa Liang, **Meitian Huang**, and Xiaohua Jia. Providing Reliability-Aware

Virtualized Network Function Services for Mobile Edge Computing. *Proceedings of 39th IEEE Intl Conf on Distributed Computing Systems (ICDCS)*, July, 2019.

[6] Yu Ma, Weifa Liang, **Meitian Huang**, Yang Liu, and Song Guo. Virtual network function provisioning for offloading tasks in MEC by trading off computing and communication resource usages. *Proc of IEEE INFOCOM WKSHP IECCO*, 2019.

[7] Yu Ma, Weifa Liang, **Meitian Huang**, and Song Guo. Profit Maximization of NFV-Enabled Request Admissions in SDNs. *Proceedings of 2018 IEEE Global Communications Conference (GLOBECOM)*, December 2018.

[8] Mike Jia, Weifa Liang, **Meitian Huang**, Zichuan Xu, and Yu Ma. Throughput Maximization of NFV-Enabled Unicasting in Software-Defined Networks. *Proceedings of 2017 IEEE Global Communications Conference (GLOBECOM)*, December 2017.

[9] Zichuan Xu, Weifa Liang, **Meitian Huang**, Mike Jia, Song Guo, and Alex Galis. Approximation and Online Algorithms for NFV-Enabled Multicasting in SDNs. *Proceedings of 2017 International Conference on Distributed Computing Systems (ICDCS)*, pp. 625–634, June 2017.

[10] Mike Jia, Weifa Liang, Zichuan Xu, and **Meitian Huang**. Cloudlet Load Balancing in Wireless Metropolitan Area Networks. *Proceedings of 2016 IEEE International Conference on Computer Communications (INFOCOM)*, pp. 730–738, April 2016.

Abstract

Software-Defined Networking (SDN) and Network Function Virtualization (NFV) have emerged as the foundation of the next-generation network architecture by introducing great flexibility and network automation capabilities, including automatic response to faults and load changes and programmatic provision of network resources and connections. It has been envisioned that the SDN- and NFV-based next-generation network architecture will play a critical role in providing network services to users, where the desired network services, including data transfer and policy enforcement, are fulfilled by allocating network resources using virtualization technologies. However, the disparity between ever-growing user demands and scarce network resources makes resource allocation exceptionally central to the performance of a network service, because only by effectively allocating these scarce resources can a network service provider satisfy users and maximize the gain from running the service.

In this thesis, we study efficient resource allocation for network throughput maximization in next-generation networks, while meeting user resource demands and Quality of Service (QoS) requirements, subject to network resource capacities. This however poses great challenges, namely, (1) how to maximize network throughput, considering that both SDN-enabled switches and links are capacitated, (2) how to maximize the network throughput while taking into account network function and QoS requirements of users, (3) how to dynamically scale and readjust resource allocation for user requests, and (4) how to provision a network service that can satisfy user reliability requirements.

To address these challenges, we provide a thorough study of network throughput maximization problems in the context of the next-generation network architecture, by formulating the problems as optimizations problems and developing novel optimiza-

tion frameworks and algorithms for the problems. Specifically, this thesis makes the following contributions.

Firstly, we consider dynamic user request admissions where user requests arrive one by one and the knowledge of future request arrivals is not given as *a priori*. We develop a novel cost model that accurately captures the usage costs of network resources and propose online algorithms with provable performance guarantees. We are the very first to study the problem and to devise online algorithms with performance guarantees for the problem. In addition, the algorithm design and analysis techniques may be of independent interests and can be applied to other optimization problems in different contexts.

Secondly, we study the problem of realizing user requests with network function requirements, with the objective of maximizing network throughput, while meeting user QoS requirements, subject to resource capacity constraints. For this problem, we develop two algorithms that strive for the trade-off between the accuracy/quality of a solution and the running time of obtaining the solution. To the best of our knowledge, we are the first to formulate a novel optimization problem by jointly taking into account data traffic routing and network function placement, while meeting different user QoS requirements.

Thirdly, we investigate maximization of network throughput by dynamically scaling network resources while minimizing the overall operational cost of a network. We propose a unified framework for two types of resource scaling – vertical scaling and horizontal scaling. Through non-trivial reductions of the problem of concern into several classic problems, we propose an algorithm that has been empirically demonstrated to deliver near-optimal solutions. It is worth noting that unlike existing studies that only dealt with either type of scaling, the proposed framework jointly considers both types of scalings.

Fourthly, we deal with the problem of reliability-aware provisioning of network resources for users, with the aim of maximizing network throughput. We devise an

approximation algorithm with a logarithmic approximation ratio for the general case of this problem. We also develop constant-factor approximation and exact algorithm for two special cases of the problem, respectively. The formulated problem is a generalization of several classic optimization problems. To the best of our knowledge, there is no prior study on similarly formulated problems, and this problem has broad potential applications in other areas as well.

Finally, in addition to extensive theoretical analyses, we also evaluate the performance of proposed algorithms empirically through experimental simulations based on real and synthetic datasets. Experimental results show that the proposed algorithms significantly outperform existing algorithms.

Contents

Acknowledgments **vii**

Publications **ix**

Abstract **xiii**

1 Introduction **1**

1.1 Next-Generation Network Architecture 1

1.1.1 Software-Defined Networking 2

1.1.2 Network Function Virtualization 3

1.1.3 The Synergy between Software-Defined Networking and Network Function Virtualization 5

1.2 Resource Allocation for Network Throughput Maximization 6

1.2.1 Resource Allocation in Software-Defined Networking 6

1.2.2 Resource Allocation for Virtualized Network Functions 7

1.3 Challenges of Resource Allocation in Next-Generation Networks 9

1.4 Research Topics and Aims 12

1.4.1 Network Throughput Maximization of User Requests in Software-Defined Networks 13

1.4.2 Network Throughput Maximization of User Requests with Network Function Requirements 16

1.4.3 Dynamic Adjustment of Resource Allocation via Scalings 19

1.4.4 Reliability-Aware Resource Allocation 22

1.5 Thesis Contributions 24

1.6 Thesis Overview 26

2 Online Unicasting and Multicasting in Software-Defined Networks 27

2.1 Introduction 27

2.2 Preliminaries 30

2.2.1 System Model 30

2.2.2 TCAM and Routing Rule Matching in SDNs 30

2.2.3 User Routing Requests 31

2.2.4 Competitive Ratios of Online Algorithms 32

2.2.5 Problem Definitions 32

2.3 The Usage Costs of Resources of Links and Nodes 33

2.4 An Online Algorithm for Dynamic Unicast Routing 34

2.4.1 Online Algorithm 35

2.4.2 Algorithm Analysis 38

2.5 An Online Algorithm for Multicast Routing 47

2.5.1 Online Algorithm 48

2.5.2 Algorithm Analysis 49

2.6 Performance Evaluation 57

2.6.1 Experimental Environment Settings 57

2.6.2 Performance Evaluation of Different Algorithms 58

2.6.3 Parameter Impacts on Algorithmic Performance 59

2.6.4 Impact of Request Implementation Durations on Algorithm Performance 62

2.7 Summary 64

3 Throughput Maximization in Software-Defined Networks with Consolidated Middleboxes 65

3.1 Introduction 65

3.2 Preliminaries 67

3.2.1 System Model 67

3.2.2 User Requests 68

3.2.3 Problem Definition 69
3.2.4 NP-Hardness 70
3.3 Integer Linear Program 72
3.4 A Heuristic Algorithm 74
3.4.1 A Novel Cost Model of Resource Usages and the Construction of an Auxiliary Graph 74
3.4.2 Algorithm 76
3.4.3 Algorithm Analysis 80
3.5 A Faster Heuristic Algorithm 82
3.5.1 Overview 82
3.5.2 Algorithm 83
3.5.3 Algorithm Analysis 85
3.6 An Online Algorithm 87
3.7 Performance Evaluation 88
3.7.1 Experimental Environment Settings 88
3.7.2 Performance of Different Algorithms within One Time Slot . . . 89
3.7.3 Algorithm Performance within a Finite Time Horizon 93
3.7.4 Impact of Request Durations on the Performance of Different Online Algorithms 94
3.8 Summary 97

4 Virtualized Network Function Placements for Delay-Sensitive Network Function Virtualization-Enabled Requests 99

4.1 Preliminaries 103
4.1.1 System model 103
4.1.2 Virtualized network functions 104
4.1.3 User requests 104
4.1.4 Dynamic admissions of user requests 105
4.1.5 Vertical and horizontal scalings 106

4.1.6 The operational cost 108
4.1.7 Problem definition 109
4.1.8 NP-hardness of the defined problem 110
4.2 Integer Linear Programming 111
4.3 Heuristic Algorithm 114
4.3.1 Overview of the proposed algorithm 114
4.3.2 VNF ordering 115
4.3.3 VNF instance placements and migrations 118
4.3.4 Algorithm 120
4.3.5 Analysis of the proposed algorithm 121
4.4 Performance Evaluation 124
4.4.1 Experimental environment 124
4.4.2 Performance evaluation of different algorithms within a single time slot 125
4.4.3 Performance evaluation of different algorithms within a finite time horizon 129
4.5 Summary 131

5 Reliability-Aware Virtualized Network Function Services Provisioning in Mobile Edge Computing 133
5.1 Introduction 133
5.2 Preliminaries 136
5.2.1 Network model 136
5.2.2 User requests 138
5.2.3 Problem definition 140
5.3 NP-Hardness 140
5.4 Integer Linear Programming 142
5.5 Approximation Algorithm for the Reliability-Aware VNF Instances Provisioning Problem 143

5.5.1 Cost modeling 143
5.5.2 Algorithm description 144
5.5.3 Algorithm analysis 144
5.6 Approximation and Exact Algorithms for Special Cases of the Reliability-Aware VNF Instances Provisioning Problem 151
5.6.1 A constant approximation algorithm for a special reliability-aware VNF instances provisioning problem 151
5.6.2 A dynamic programming algorithm for another special reliability-aware VNF instances provisioning problem 155
5.6.2.1 A dynamic programming algorithm 155
5.7 Performance Evaluation 162
5.7.1 Experimental environment settings 162
5.7.2 Algorithm performance evaluation for the reliability-aware VNF instances provisioning problem 163
5.7.3 Algorithm performance evaluation for special reliability-aware VNF instances provisioning problems 164
5.7.4 Parameter impacts on algorithm performance 167
5.8 Summary 169

6 Conclusion and Future Works 171
6.1 Summary of Contributions 171
6.2 Future Works 173

List of Figures

1.1 A service function chain consisting of three network functions: NAT, Firewall, and IDS 3

1.2 A service provider network G with a set $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ of SDN-enabled switch nodes and a subset $V_S = \{v_1, v_4, v_5, v_6\}$ ($V_S \subseteq V$) of switch nodes attached with servers or data centers. 8

2.1 The construction of $G'(k) = (V'(k), E'(k))$ for an SDN $G = (V, E)$ for online unicasting when request r_k arrives. 35

2.2 The accumulated bandwidth delivered by different algorithms in networks. 58

2.3 The performance of Algorithm 2.1 for online unicasting by varying α and β, when $\sigma_v = \sigma_e = n - 1$. 60

2.4 The performance of Algorithm 2.2 for online multicasting by varying α and β when $\sigma_v = \sigma_e = n - 1$. 61

2.5 The accumulated bandwidth delivered by Algorithm 2.1 for online unicasting and Algorithm 2.2 for online multicasting with thresholds $\sigma_v = \sigma_e = n - 1$ and without the thresholds $\sigma_v = \sigma_e = \infty$ when $\alpha = \beta = 2n$. 62

2.6 The network throughput delivered by Algorithm 2.1 for online unicasting and Algorithm 2.2 for online multicasting by varying the maximum duration of admitted requests in terms of numbers of time slots. 63

3.1 An example of the SDN G constructed from an instance of the GAP with four items and four bins. 71

3.2 An ILP formulation of the network throughput maximization problem 73
3.3 The auxiliary graph construction of G'_i from G for the ith request: (a) the original SDN $G = (V, E)$; and (b) the corresponding auxiliary graph $G'_i = (V'_i, E'_i)$ of G. 75
3.4 Augmenting auxiliary graph G'_i on the left to $G'_{i,v}$ on the right for switch $v \in V_{pm}$ 77
3.5 An example of a bipartite graph G_b. 85
3.6 Performance of different algorithms on the GÉANT within one time slot if the number of request is fixed. 90
3.7 Performance of different algorithms on the GÉANT within one time slot if the number of requests follows a Poisson distribution. 91
3.8 Performance of different algorithms in the GÉANT by varying the number of switches from 100 to 600, while the number of requests is fixed at 160 per time slot. 92
3.9 Performance of different online algorithms in the GÉANT within a time horizon of 200 time slots, where the number of requests arrives at each time slot follows a Poisson distribution with a mean of 30. 93
3.10 The accumulative number of admitted requests and the running time of different online algorithms based on algorithms `ALG-1`, `ALG-2` and `MH` for a time horizon of 200 time slots in ISP networks. 95
3.11 The accumulative number of admitted requests and running time of different online algorithms based on algorithms `ALG-1`, `ALG-2`, and `MH` with different maximum durations of requests when the network is of the GÉANT 96

4.1 An example network G with six nodes and two requests r_1 and r_2 . . . 100
4.2 An example network 103
4.3 Different options for admitting r_2 after request r_1 has been admitted in network G in Figure 4.1 106

4.4 An ILP formulation of the network throughput maximization via VNF instance scaling problem 112
4.5 The four service chains demanded by four requests r_1, r_2, r_3, and r_4 can be represented by the directed graph H 116
4.6 Performance of different algorithms within one time slot, using a real network topology with 40 nodes 126
4.7 Performance of different algorithms within one time slot, using synthetic networks of various sizes 128
4.8 Performance of different algorithms within a finite time horizon, using a real network topology 130

5.1 An MEC consists of five APs and three cloudlets co-located with APs. User devices access the network through their nearby APs and APs are interconnected by optical links of the MEC. 137
5.2 An illustration of the proof of Theorem 5.4. 159
5.3 Performance evaluation of the approximation algorithm `ALG-1` for the reliability-aware VNF instances provisioning problem 163
5.4 Performance of different algorithms for the special case of the reliability-aware VNF instances provisioning problem where each request requires exactly one secondary instance 164
5.5 Performance of different algorithms for the special reliability-aware VNF instances provisioning problem 166
5.6 Performance impact of parameter $n_{\max}$ on algorithm `ALG-1` 168
5.7 Impacts of the admission control policy and the value of α on the performance of algorithm `ALG-1` 168

Chapter 1

Introduction

> The management of society's resources is important because resources are scarce. Scarcity means that society has limited resources and therefore cannot produce all the goods and services people wish to have.
>
> *Principles of Economics*
>
> N. Gregory Mankiw

1.1 Next-Generation Network Architecture

The Internet has accelerated the transition into a digital society, where every device is connected and accessible at any time and from anywhere. This trend is further fueled by the proliferation of Mobile Edge Computing (MEC) and the Internet of Things (IoT). Unfortunately, conventional network architectures cannot meet such ever-growing demands. Software-Defined Networking (SDN) and Network Function Virtualization (NFV) have been emerged and envisioned to revolutionize the networking landscape profoundly. Not only does SDN address the issue that the static architecture of conventional networks is not suited for dynamic computing and storage needs of today's data centers, campuses, and carrier environments [69], but it also promises to simplify network management and enables technology innovation

and evolution dramatically [57, 85, 107, 136]. Meanwhile, NFV has great potential to lead to significant reductions in operating expenses (OPEX) and capital expenses (CAPEX) and facilitate the deployment of new network services with increased agility and faster time-to-value [103, 143].

1.1.1 Software-Defined Networking

In spite of its widespread adoption, the conventional network architecture has been considered ill-suited for the management of large-scale complex networks [57, 85, 107, 136], which demand agile re-policing or re-configurations from time to time. A typical computer network consists of a large number of network devices such as routers and switches [35, 127], and the operator of the network is responsible for configuring policies to respond to a wide range of network events and applications [107]. In a conventional network, the operator must implement desired network policies and accomplish complex tasks using low-level, vendor-specific commands [57]. Automatic reconfiguration and response mechanisms, which are critical to endure the dynamics of faults and adapt to load changes, are virtually non-existent in such a network [85]. As a result, enforcing the required policies in such a dynamic environment is highly challenging, and network service providers are faced with high operating expenses and significant management burdens. One of the key contributing factors of the complexity is that the control plane, which determines how network traffic should be routed in the network, and the data plane, which forwards traffic in accordance with routing decisions made by the control plane, are vertically integrated (i.e., these two planes are bundled together inside networking devices) and vendor-specific [85, 107, 136].

SDN as an emerging networking paradigm promises to change this state of affairs by breaking vertical integration, separating the control logic of a network from the underlying routers and switches, promoting a logically centralized network control, and introducing the ability to program the network. The separation of the control

and data planes allows the network control to become directly programmable via a standardized interface and the underlying infrastructure to become simple packet forwarding devices that can be automated. The separation of concerns introduced between the definition of network policies, their implementation in switching hardware, and the forwarding of traffic, is key to the desired flexibility: by breaking the network control problem into tractable pieces, SDN makes it easier to create and introduce new abstractions in networking, simplifying network management and facilitating network evolution [24, 61, 57, 136]. As a result, SDN has attracted significant attention from both the academia and industry as an essential paradigm for the management of large-scale complex networks. SDN is becoming a key technology for the next-generation network architecture and has been applied to many large-scale networks, including Internet backbone networks and data-center networks, e.g., Google's B4 [56, 68] and Microsoft's SWAN [55].

1.1.2 Network Function Virtualization

Computer networks nowadays rely on a broad spectrum of network functions, including firewall, Intrusion Detection System (IDS), WAN optimizer, and Deep Packet Inspection (DPI), to enhance the performance and security of different network services [44, 119]. Specifically, users require their data traffic to traverse specified sequences of network functions that are referred to as *service function chains*. Figure 1.1 shows a service function chain. In this example, the traffic from the source must traverse Network Address Translation (NAT), firewall, and IDS (in this order) before reaching the destination.

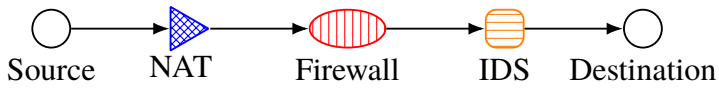


Figure 1.1: A service function chain consisting of three network functions: NAT, Firewall, and IDS

Network functions have been conventionally implemented as dedicated hardware.

Unfortunately, management and deployment of hardware-implemented network functions are complex and costly. For example, a survey conducted by Sherry et al. [119] showed that large networks (10k-100k network nodes) spent over one million dollars on deploying and maintaining hardware-implemented network functions while medium and small networks (1k-10k nodes) spent between $5,000 and $50,000. As demands for more diverse and new short-lived services with high data rates continue to increase, network operators must continuously purchase, install, and maintain extra physical equipment in order to keep up with user demands. These, coupled with requirements for high quality, stability, and stringent protocol adherence, have resulted in prolonged product cycles, very low service agility and heavy dependence on specialized hardware to implement network functions [103, 143].

NFV has drawn significant attention from both industry and academia as a significant paradigm shift in service provisioning. In this new architecture, a network function, such as firewall or NAT in Figure 1.1, can be implemented as an instance of a software component that is offloaded to a network service provider for execution in the provider's network. This allows for the consolidation of many network equipment types onto high volume servers, switches and storage, which could be located in data centers, distributed network nodes, or in close proximity to end users, in order to better tailor for user requirements [103]. A given service can be decomposed into a set of Virtualized Network Function (VNF) instances, each of which can be implemented as a software component running on one or more off-the-shelf physical servers [67, 143]. The VNF instances may then be relocated and instantiated at different locations in a network (e.g., aimed at introduction of a service targeting specific customers in the strategic geographical location) without necessarily requiring the purchase and installation of new hardware. By decoupling network functions from the hardware platform on which network functions are executed, NFV has the potential to lead to significant reductions in operating expenses and capital expenses, and facilitate the deployment of new services with increased agility and faster time-to-value [103, 143].

1.1.3 The Synergy between Software-Defined Networking and Network Function Virtualization

SDN and NFV share many similarities: They both advocate open software and standard network hardware, and they seek to leverage automation and virtualization technologies to achieve their respective goals [67, 85, 103, 107, 143]. They are highly complementary technologies, combining them in the same network will generate higher values. Consequently, it has been envisioned that the centralized control and management applications (such as load balancing, monitoring, and traffic analysis) in SDNs can be partially implemented as VNF instances, and hence benefit from NFV's reliability and elasticity functionalities [103, 143]. Similarly, SDN can accelerate the NFV deployment by providing a flexible and automated way of chaining functions, provisioning and configuration of network connectivity and bandwidth, automation of operations, security and policy control [27].

Due to the prospect of SDN and NFV, network service providers are currently moving towards a next-generation network architecture by adopting these technologies in order to mitigate issues in conventional networks. In a next-generation network, the network service provider is in charge of managing and allocating network resources, e.g., bandwidth and computing resources, and provides network services to the public on a pay-as-you-go basis. In order to acquire resources from the network, customers specify their resource demands and send *user requests* to the network service provider, and the provider then determines if user requests should be admitted and which resources should be allocated for admitted requests. Specifically, to make use of the resources provided by a network service provider, a customer specifies the desired amount of resource and the associated desirable properties of needed resources, e.g., the maximum tolerated end-to-end delay and reliability requirements, to the network service provider. Upon receiving a user request, the network service provider queries the network infrastructure for operational statistics including residual capacities, current workload, etc. Based on the collected information, the network service provider

might reserve resources in corresponding data centers for the request and then grant the customer to access those resources.

1.2 Resource Allocation for Network Throughput Maximization

While both industry [45, 55, 56, 68, 106] and academia [85, 103, 107, 143] are embracing SDN and NFV as the cornerstone of the next-generation network architecture at unprecedented rates, SDN and NFV are still in their infancy and there is a full spectrum of ongoing topics. As more network service providers look at the details of deploying SDN and NFV and realizing the benefits that these key enabling technologies promise, there is an urgent need to develop new methods, tools, and techniques, for successful deployment and broad adoption of these technologies. In this thesis, we study a central topic for the next-generation network architecture – network throughput maximization via effective resource allocation. This is motivated by the fact that while network service providers strive to serve end users, resources that they possess are inherently scarce since they have a finite amount of resources in their systems yet the amounts of resources their customers demand can be unbounded. Therefore, the ability of a network service provider to effectively utilize network resources is crucial to the success of network services.

1.2.1 Resource Allocation in Software-Defined Networking

In order to achieve the economy of scale expected from SDN, network resources should be used effectively and efficiently. Veitch et al. [131] showed that default deployment of some current common use cases often leads to sub-optimal resource allocation and consumption, necessitating efficient algorithms for network resource allocation with various optimization objectives including load balancing, energy consumption reduction, disaster resiliency, etc. A software-defined network consists of

a control plane and a data plane. The data plane consists of network devices, which is referred to as *SDN-enabled switches*, which are interconnected by Internet links. SDN switches forward network packets according to the *forwarding rules* stored in *forwarding table* prepared by the control plane, while the controller implements the control logic and is responsible for translating high-level network requirements into forwarding rules to be installed at switches. Due to their great flexibility, forwarding rules in SDN are more complex and require much more storage space in comparison with the forwarding rules in conventional networks. Therefore, in order to match incoming packets against complex forwarding rules as fast as possible, the forwarding table at each switch is thus generally implemented by Ternary Content Addressable Memory (TCAM) that supports fast, parallel lookups. However, TCAM is 400 times more expensive and 100 times more power consuming per unit than RAM-based storage [85]. Accordingly, the capacity of the forwarding table of an SDN-enabled switch is typically limited to several thousand entries [79]. Such highly restricted capacity of forwarding tables has been recognized as a bottleneck to the scalability of SDN [24, 79, 107], and efficient utilization of forwarding tables to serve a scaling number of users is challenging and vital.

Meanwhile, the amount of traffic that will be carried by SDN is expected to grow exponentially along with the widespread adoption of SDN [24, 61, 57, 136]. To meet rapidly growing demands of users for routing their requests in SDN-enabled networks and Internet backbone networks, there is a need for a large number of forwarding rules stored at each switch node and a large quantity of bandwidth at each link for various request admissions. A methodology that can effectively utilize both types of resources is thus desperately needed.

1.2.2 Resource Allocation for Virtualized Network Functions

Network function virtualization is often offered by network service providers as a public service. To serve end users, network service providers execute VNF instances

for users on resources leased from one or more infrastructure providers [103]. The providers may also need to determine if it is appropriate to chain required network functions to create services for end users. Furthermore, as NFV relies on dynamic creation, modification, and removal of connections between devices in networks and SDN offers these effortlessly, NFV has been integrated into software-defined networks in order to achieve higher performance through the synergy between SDN and NFV, e.g., [45]. In these networks, some SDN-enabled switches are attached with data centers with limited computing and storage resources, and network functions can be executed in these data centers. Such a network service provider network can be represented by a graph where each node represents a switch, each link corresponds to an Internet link between switches, and a subset of switches is connected to servers or data centers to implement VNF instances. Figure 1.2 is an example of a network owned by a network service provider.

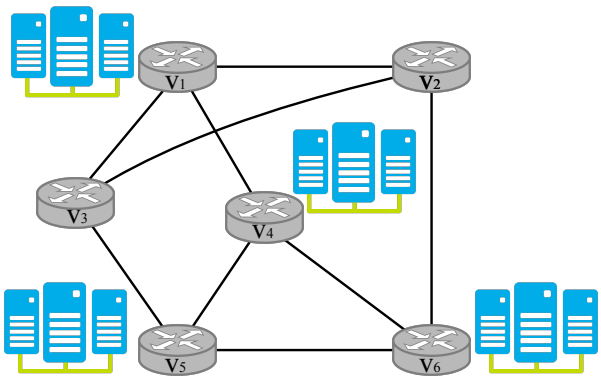


Figure 1.2: A service provider network G with a set $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ of SDN-enabled switch nodes and a subset $V_S = \{v_1, v_4, v_5, v_6\}$ ($V_S \subseteq V$) of switch nodes attached with servers or data centers.

In order to admit a user request from its source to its destinations, it is necessary to allocate (i) space in forwarding tables at switches to set up forwarding rules, (ii) bandwidth resources at links, and (iii) computing resources at data centers. As these network resources have capacity constraints and they are shared among all user requests, resource allocation dramatically impacts the performance of the network.

1.3 Challenges of Resource Allocation in Next-Generation Networks

Network service providers have expanded at a fast pace in recent years, continually striving to meet increasingly higher and rapidly changing user demands [56, 106]. They aim to serve various organizations and users with diverse use-cases by providing heterogeneous resources for them. To this end, being able to allocate and provision network resources efficiently and effectively is of paramount importance, this poses the following challenges.

- One is that user request admissions are subject to capacity constraints of network resources. Specifically, the computing resource of data centers and the bandwidth resource of links in networks, are limited. Combining this with the characteristics of software-defined networks, i.e., significant fluctuations in resource consumptions and availabilities due to all resources in the networks being dynamically allocated, and the limited capacity of TCAM, resource allocation is non-trivial in next-generation networks.

- Another is that effective resource allocation demands a joint consideration of multiple resources, instead of taking into account these resources individually. In order to operate a large-scale public network service, effective management of multiple network heterogeneous resources becomes vital, since unbalanced usages of network resources will result in suboptimal resource utilization. Therefore, there is an urgent need for a scheme of resource allocation and user admission that pays as much attention to one type of network resources, e.g., the forwarding table, as it does to other types of network resources, e.g., the bandwidth of each link. Only through jointly considering multiple network resources and therefore balancing usages of network resources can a network service provider utilize the limited resources in the network to serve an ever-increasing number of users.

- Resource utilization in next-generation networks exhibits high dynamics, thereby requiring *online algorithms* to be proposed. In these networks, user requests typically arrive into a network one by one, and hence the knowledge of future arrivals is not available. Meanwhile, admitted requests might not depart from the system and continuously occupy the allocated resources for an extended period. Allocating limited resources at switches and links while taking into account the dynamic nature of *online user requests*, i.e., requests arrive one by one and no knowledge of future arrivals and departures, is challenging, because an online algorithm for admissions of online user requests is given only a single request at a time and a decision, either acceptance or rejection, needs to be made based on the request, cf. the Secretary Problem [40].

- As network service providers offer their services to the public, they are interested in maximizing their monetary gains by accommodating as many users as possible. As a result of this drive to maximize their profits and the scarcity of network resources, admitting user requests into service providers' networks with the aim of maximizing the number of accommodated users poses significant challenges, that is, how to find a cost-optimal routing path for the data traffic of a request such that the data traffic of the request will pass through the network functions one by one in their orders in the service chain of the request, where each user request usually has both bandwidth and computing resource demands to realize its routing traffic through the network and implementing its network functions, subject to the aforementioned resource capacity constraints.

- Network services based on the next-generation network architecture have many promising applications including smart cities and smart connected vehicles [12, 42, 125, 144]. In the near future, the role of network services in the society will be as fundamental as public utilities, e.g., electricity and water, and applications based on these network services will be operating continuously for several months or years. When an organization embraces the next-generation

network architecture, the organization's IT services will be outsourced to network services, and the organization requires long-term network services from the provider. As the organization evolves, its resource requirements will change over time, making the resource allocations performed when the organization was initially admitted suboptimal. Accordingly, service providers need to dynamically adjust existing resource allocations in order to serve admitted requests better and accommodate more future requests. Such dynamic adjustments of resource allocations are often referred to as *scalings*. Specifically, there are two types of scalings: (i) *horizontal scaling*, i.e., migrating existing VNF instances from their current locations to new locations, and (ii) *vertical scaling*, i.e., instantiating new VNF instances. These scaling options need to be jointly considered and performed without causing interruptions to admitted requests. It thus poses the question of how to perform horizontal and vertical scalings such that no existing requests are impacted, and more newly arriving requests can be admitted.

- Service providers also provide additional services to users, most notably Quality of Service (QoS) guarantees, in addition to providing network services through allocating the required amount of resources. While the additional QoS guarantees generate new sources of income for network service providers, they also put the burden on service providers to be able to satisfy user-specified requirements. Specifically, users may have requirements on the end-to-end delay of network services. Also, users may have requirements on the availability and reliability of the services. These requirements mandate additional resources to be allocated while admitting the user requests. For instance, in order to fulfill a user's reliability requirement, it may be necessary to replicate the same resources for the service at different geographic locations (servers). This thus poses the question of how to meet such QoS requirements by striving for the non-trivial trade-off between the revenue of admitting requests with QoS requirements and the cost of provisioning extra resources to satisfy the requirements.

- Providing users with reliable network services is the top priority of most network service providers, as unreliable services or severe service failures can result in tremendous losses of users, particularly for their mission-critical applications. Nevertheless, the virtualization of network functions imposes reliability concerns on VNFs, which are prone to software issues of virtualization technologies. Thus, how to ensure the reliability of a network service is a fundamental research topic in the provisioning of NFV-enabled services for users.

To tackle the aforementioned challenges, this thesis will devise efficient algorithms for network throughput maximization via effective resource allocation. We distinguish this work from existing ones as follows. (1) this thesis focuses on allocation and provisioning of various network resources for the next-generation network architecture, (2) this thesis studies dynamic admission of users requests, in which requests arrive one by one and future request arrivals and departures are not known, (3) this thesis takes into account various user QoS requirements, including network enforcement policies, end-to-end delays, and service reliability requirements, and (4) this thesis devises efficient algorithms with performance guarantees whereas most existing works proposed heuristic algorithms only. This thesis delivers promising solutions to effective resource allocation and provisioning in the next-generation network architecture, while jointly taking into account various user requirements and resource capacity constraints. The proposed solutions are evaluated both analytically and numerically.

1.4 Research Topics and Aims

To meet ever-growing user service demands and facilitate efficient resource allocation in the next-generation network architecture, this thesis focuses on the following four main research topics:

1. Network throughput maximization of admitting user requests in software-defined networks,

2. Network throughput maximization of admitting user requests with network function and QoS requirements,

3. Dynamic scalings of resource allocation for requests with network function and QoS requirements, and

4. Reliability-aware provisioning of network function services.

These topics are interrelated with each other. Topic (1) studies the admission of user requests in software-defined networks while taking into account the TCAM table capacity at SDN-enabled switches and the bandwidth capacity at links. Topic (2) is a sequel to Topic (1) as it studies how to meet network function requirements of user requests, while also considering the TCAM capacity constraints and link bandwidth capacities. In addition, while Topic (1) and Topic (2) address allocating and provisioning of network resources for user requests upon the arrival of requests, Topic (3) studies how to dynamically adjust existing resource allocations via horizontal and vertical scalings. In contrast to Topics (1) to (3), in which failures of network condiments are considered to be rare, Topic (4) studies resource allocation in next-generation networks while taking into consideration failures of VNF instances and meeting user reliability requirements. In the following, we summarize the specific researches in these topics and the distinctions between existing studies and ours.

1.4.1 Network Throughput Maximization of User Requests in Software-Defined Networks

Early studies on request admissions in SDNs primarily addressed the challenges in traffic engineering and steering for incremental deployment of SDN-enabled devices in existing networks [1, 16, 110]. Most of these studies dealt with data traffic routing, by considering TCAM of limited sizes at each SDN-enabled switch [24, 61, 79]. Specifically, Kanizo et al. [79] studied unicast routing for a group of unicast requests. To overcome the limitation of the forwarding table size, they proposed two approaches

to decompose large SDN forwarding tables into several small ones and to distribute these small tables across the network, while maintaining the overall SDN policy semantics. Huang et al. [58] tackled the limitation of TCAM size by selectively caching forwarding rules in local switches and forwarding packets to the centralized controller if necessary, while Katta et al. [81] proposed CacheFlow to equip each switch with hardware memory implemented by TCAM and secondary software memory in order to create an abstraction of infinite space for forwarding rules. Cohen et al. [24] studied the bounded path-degree maximum-flow problem subject to the TCAM capacity on switches. They proposed an approximate solution with high probability. Meanwhile, Huang et al. [61] studied the multicast problem in SDNs, by devising an approximation algorithm.

These mentioned studies only considered the switch node capacity for a single or a given set of user requests. The most related work due to Huang et al. [60] dealt with admissions of a set of multicast requests with the aim to minimize the total bandwidth consumption on realizing the requests, subject to both node and link capacity constraints. The request admission problem due to Huang et al. [60] thus is substantially different from the one in this thesis, as a different system model was adopted. In this thesis, we assume that each switch has a limited TCAM and no end-to-end delay constraint is imposed on each request. The solution in [60] thus cannot be extended to solve the problem in this thesis. Moreover, Xu et al. [142] considered user query processing among cloudlets in a wireless metropolitan area network such that the network throughput is maximized, while the average access latency among user requests is minimized. Since Xu et al. focused on load-balancing among cloudlets with computing capacity constraints, by assigning different user requests to different cloudlets, neither TCAM nor SDN has ever been considered. Thus, the solution in [142] is not applicable to the problem in this thesis, either.

Unlike the aforementioned studies, we here deal with dynamic admissions of unicast or multicast requests without the knowledge of future arrivals, subject to both

TCAM capacities at switch nodes and bandwidth capacities at links. In particular, the joint consideration of these two different types of network resources when performing online request admission is very challenging, in comparison with existing works that only focused on one type of resource and all requests are given in advance, as some admitted requests may still occupy the resources when a new request arrives, thereby leaving less available resources for later arrived requests. Existing studies failed to address the following crucial aspects in dynamic admissions of requests.

1. Due to dynamic changes in network resources, there desperately needs a cost metric or a set of cost measures to model dynamic consumptions of various network resources and these resources utilization. Building an accurate cost model to capture such dynamics is crucial, which then can be used to guide efficient resource allocation for incoming requests. The key to developing such a cost model is to model the availability and utilization of each resource accurately. An exponential function of a specific resource and its utilization rate is an excellent candidate for such cost modeling, which has been adopted for online request routing in many different types of networks [6, 80, 92, 93, 109].

2. The joint consideration of both the forwarding table capacity at each switch node and the bandwidth capacity at each link makes the cost modeling of resource usages in SDNs more difficult. Meanwhile, the joint consideration of resources at both nodes and links complicates the analysis of the proposed solution, since the analysis of performance (i.e., the competitive ratios) of online algorithms for online unicasting and multicasting in ATM networks, virtual circuit networks [6, 109], and wireless sensor networks [93] are only based on the cost modeling of a single type of resource at either nodes or links.

1.4.2 Network Throughput Maximization of User Requests with Network Function Requirements

While network functions are widely used to guarantee security and performance of routing data packet traffic in contemporary computer networks, the deployment of traditional hardware network functions incurs high capital investment and operational cost [118, 119]. To tackle these issues, recent efforts on new frameworks and architectures for NFV [5, 43, 48, 100, 118], have been demonstrated as promising alternatives to traditional hardware by virtualized network functions in virtual machines. For example, Sekar et al. [118] devised an architecture that focused on software-based implementations of network functions on a shared hardware platform. Qazi et al. [110] developed SIMPLE that enforces high-level routing policies for networks function-specific traffic. Fayazbakhsh et al. [38] proposed FlowTags, because traditional flow rules do not suffice in the presence of dynamic modifications performed by network functions. Martins et al. [100] introduced a virtualization platform to improve network performance by revising existing virtualization technologies to support the deployment of modular, virtual middleboxes on lightweight VMs.

One fundamental problem under the NFV architecture is network throughput maximization of realizing user routing requests with specified service chains while meeting various resource constraints and user QoS requirements. A few recent studies investigated this issue [22, 49], which however neither considered resource constraints such as the forwarding table size constraints on switches, nor took global optimization, thus the solutions delivered are suboptimal, e.g., Charikar et al. [22] assumed that every switch in a network can perform middlebox functions without considering forwarding table sizes. Gushchin et al. [49] assumed that the routing traffic of a request can be split into multiple paths, and proposed a two-stage local optimization (before and after the virtual middleboxes). Zhang et al. [146] presented a routing scheme that reduces TCAM space usage without causing network congestion. However, they did not consider user requests with service chain requirements. Kuo et al.

[88] studied a problem of VM placement and path selection, striving for a tradeoff between link and server usage. This work, however, is different from ours because they assumed that multiple requests of the network function can be satisfied using a single VM that implements the network function.

On the other hand, Li et al. [90] presented the design and implementation of a system that dynamically provisions resources to provide timing guarantees with the objective of maximizing the number of requests admitted to the cloud, while meeting the deadlines of admitted requests. Huang et al. [59] considered a joint optimization problem of middlebox selection and routing with the objective to maximize the throughput of an SDN, and proposed an algorithm based on the Markov approximation technique. Lukovszki et al. [95] considered middlebox placements in a n-node network so that each source-destination pair in a given set has a path of length at most L with one middlebox in it, and each middlebox can be used by at most k pairs. They devised an approximation algorithm for the problem, under the assumption that only one VM (or a network function) is associated with each request, and each server can accommodate no more than k VMs. Clearly, this assumption is over-simplified as the length of a service chain of each request may be far greater than one. Lukovszki and Schmid [96] achieved several important theoretical results under ideal assumptions that each server can accommodate only one VM, and different VMs for different network functions consume the same amount of computing resource.

Some studies focused on deploying service function chains, through provisioning VNF instances with various optimization objectives, which have attracted tremendous attention. Cao et al. [16] studied the problem of policy-aware traffic engineering in SDNs, by assuming that the traffic has to pass a given sequence of network functions. For instance, Sallam et al. [115] investigated the shortest path and maximum flow problems subject to service function chaining constraints by assuming splittable user traffic. Kuo et al. [87] studied the service function chain embedding problem for a multicast request with multiple sources and a single destination. Ren et al. [114] tack-

led the problem of optimally embedding service function trees for a single multicast request and proposed an approximation algorithm for the problem. Xu et al. [137] employed the Lyapunov optimization technique to jointly optimize dynamic service caching and task offloading with the objective of minimizing computation latency experienced by the user and energy consumption of the system. Agarwal et al. [2] formulated an optimization problem with the objective of minimizing the ratio of the actual latency experienced by the user to the user-specified maximum latency. Tomassilli et al. [128] proposed the optimal placement of network service function chains in order to satisfy a set of user demands subject to the ordering constraints specified by service function chains. They devised two logarithmic factor approximation algorithms by transforming the problem to the Set Cover problem [26, 130]. Little attention has been paid to the trade-off between usages of different resources while provisioning VNF instances. Kuo et al. [88] considered the deployment of service function chains with emphasis on the trade-off between link and server usage, while He et al. [51] studied the joint service placement and request scheduling in order to optimally provision edge services while taking into account the demands of both resources that can be shared, e.g., storage, and resources that cannot be shared, e.g., communication and computation. They proposed an algorithm with a constant approximation ratio for a special case of the problem and a heuristic for the general case.

Following the same spirit as [22, 49], we assume that network functions are implemented as software applications running as VMs in servers or data centers. Meanwhile, there are many possibilities for resource sharing, one of which is to use a dedicated VM for each NFV. However, considering a service function chain that is often made up of several functions [119], this approach will clearly not be feasible as physical resources will quickly be consumed, and will be wasteful of resources since most functions are light-weight and can, hence, be processed by a single VM, e.g., by containers within the VM [43]. Therefore, in this thesis, we adopt the idea of

consolidated middleboxes [49], where every flow obtains all its required functional treatment at a single server, because the consolidated middlebox model simplifies traffic routing, helps reduce the number of routing rules in the switches, and removes the topology dependence between different middleboxes. Unlike Charikar et al. [22], we do not allow routing traffic via multiple paths from its source to its destination, because most network functionalities will be applied to the entire packet flow, e.g., encryption and decryption should only be applied to the entire particular traffic. In particular, we consider three types of network resource capacity constraints on both switches and links: memory capacities of SDN-enabled switches, computing capacities of servers that are attached to some switches, and bandwidth capacities of links between switches.

1.4.3 Dynamic Adjustment of Resource Allocation via Scalings

Most existing studies focused on the optimal placement of VNF instances under a specific optimization objective while meeting user-specified resource demands and QoS requirements [116, 138], or on historical VNF instance demand patterns or predictions [14, 82]. However, the aforementioned studies considered the admission of each request separately from others without taking into account potential migrations of VNF instances. As a result, network resources may be overloaded and no future requests can be admitted. Several studies of NFV migrations dealt primarily with the development of migration mechanisms [17, 18, 34, 44, 82, 112, 135]. For example, Eramo et al. [34] provided one of the earliest studies of vertical scaling of VNF instances with the objective of minimizing the sum of energy consumptions of network function instances and the re-configuration cost of network instances. Specifically, with the increase or decrease of data traffic, the computing capacities of VNF instances increase or decrease accordingly. Xia et al. [135] studied the problem of migrating network functions such that the migration cost, defined as the aggregated transferring traffic rate during the migration progress, is minimized. Carpio et al. [17] and

Carpio and Jukan [18] tackled the NFV migration problem, by using replications in place of migrations, because of the adverse effects that migrations have on QoS. They provided an Integer Linear Programming solution to the problem. While Kawashima et al. [82] provided a solution to the dynamic placement of VNFs in a network to follow the traffic variation, they did not take into account the cost of migrating VNFs.

Admitting NFV-enabled requests has been extensively studied in various settings. For example, [72, 70, 97, 138, 139] devised the very first algorithms with performance guarantees for NFV-enabled unicasting and multicasting, respectively. However, the aforementioned studies considered the admission of each request separately from others without taking into account potential migrations of VNF instances. As a result, the network resources may be overloaded and no future requests can be admitted. Several studies of NFV migrations primarily dealt with the development of migration mechanisms [17, 18, 34, 44, 82, 112, 135]. For example, Eramo et al. [34] provided one of the earliest studies of vertical scaling of VNF instances with the objective of minimizing the sum of energy consumptions of network function instances and the re-configuration cost of network instances. Specifically, with the increase or decrease of data traffic, the computing capacities of VNF instances increase or decrease accordingly. Xia et al. [135] studied the problem of migrating network functions such that the migration cost, defined as the aggregated transferring traffic rate during the migration progress, is minimized. Carpio et al. [17] and Carpio and Jukan [18] recently tackled the NFV migration problem, by using replications in place of migrations, because of the adverse effects that migrations have on Quality of Service (QoS). They provided an Integer Linear Programming solution to the problem. While Kawashima et al. [82] provided a solution to the dynamic placement of VNFs in a network to follow the traffic variation, they did not take into account the cost of migrating VNFs. There are recent works in this topic too [39, 75, 94, 133]. For example, Fei et al. [39] proposed a proactive approach that provides extra VNF instances for overloaded network functions in advance, based on the estimation of future flow rates. They first

adopted an efficient online learning method with the objective of minimizing the error in predicting the demands of different network functions. Based on the proposed method, the requested instances with variable processing capacities can be derived. Wang et al. [133] proposed online algorithms with the aims of minimizing the VNF provisioning cost for cases with one service chain and multiple service chains, respectively. Liu et al. [94] studied the optimization of service chains deployment for new users and readjustment of active users service function chains while taking into account resource consumption and operational overhead. Jia et al. [75] investigated the dynamic placement of VNF service chains across geo-distributed data centers in order to serve user requests with the objective of minimizing the operational cost of the network service provider. They proposed an efficient online algorithm for the problem based on the regularization approach. Ma et al. [98, 99] investigated dynamic request admissions with service delay and function chain requirements in a distributed cloud. The objective of the problem considered in [98, 99] is to maximize the profit collected by the service provider. Xu et al. [140] considered user request admissions with NFV service and QoS requirements by placing new VNF instances into clodlets and sharing existing VNF instances among requests [140].

We distinguish our work in this thesis from the aforementioned ones as follows. Existing studies only dealt with either VNF migration [34, 135] or creating multiple VNF instances [17, 39, 75, 94]. We focus on NFV-enabled request admissions while meeting their end-to-end delay requirements, and aim to maximize the network throughput while minimizing the operational cost of the network operator by jointly exploring VNF instance sharing, instantiation, and migration through horizontal and vertical scalings. We propose a unified framework for this and develop an efficient algorithm for the problem.

1.4.4 Reliability-Aware Resource Allocation

Several studies on the provisioning of reliable VNF services have been conducted recently. Existing methods to improve reliability of NFV are summarized by Han et al. [50]. For example, Beck et al. [11] conducted one pioneering study related to survivability of VNF instances in the context of the VNF resource allocation, by admitting a set of VNF requests with the aim of reducing the amount of network resource allocated to VNF chains, while computing resilient allocations to protect network services from both link and VNF instance failures. They however did not differentiate reliability requirements of different requests. Casazza et al. [19] cast a problem of assigning virtual machines for network function implementations to servers in order to guarantee the availability of virtual machines via VNF instance replications. They considered a geo-distributed data center network in which there are sufficient resources to accommodate all user requests. Their objective is to maximize the minimum availability among all requests by developing heuristics. Ceselli et al. [20] dealt with the design of edge cloud networks with the aim to determine where to install cloudlets among the potential AP sites by developing efficient heuristics. Ding et al. [30] proposed an approach to enhancing the network resilience while maximizing cost-efficiency of the network through improving the ratio of the reliability to the backup cost of VNF instances. Fan et al. [36, 37] investigated how to map service function chains to a network with high end-to-end reliability requirements, by adopting on-site and off-site VNF instance backups. Their work is one of the first works on this topic, and recently Li et al. [89] proposed onsite and off-site backup mechanisms by developing efficient scheduling algorithms for the problem through adopting the primal and dual update technique. Kang et al. [77] studied the tradeoff between the reliability of a network service, as measured by the probability that the service is correctly executed, and the computational load of servers. Engelmann and Jukan [32] studied the end-to-end service reliability in data center networks (DCNs). They divided large flows into several smaller flows yet provided only one backup flow for reliability guarantee.

Moualla et al. [105] considered the placement of service chains in DCNs, and devised an algorithm for the placement on the special fat-tree topological network structure. Kong et al. [84] proposed a mechanism that employs both backup path protection and VNF instance replication in order to guarantee the availability of a service function chain. The proposed mechanism determines the number of VNF instance replicas required for each VNF in the SFC, and allocates the replicas to physical nodes on the primary and backup paths while taking into account the ordered dependency among VNFs. Herker et al. [52] introduced several VNF backup strategies for service function chains, and analyzed the impact of different data-center architectures on the service provision. They then proposed cost-per-throughput algorithms for a given reliability requirement. Carpio and Jukan [18] investigated how to improve service reliability through the joint consideration of replications and migrations. Qu et al. [111] aimed to minimize the communication bandwidth usage across the network while considering the availability requirements, by developing a heuristic. Aidi et al. [3] recently proposed a framework to efficiently manage survivability of service function chains and the backup VNFs. They aimed to determine both the minimum number and optimal locations of backup VNFs to protect service function chains. The proposed heuristics have been empirically demonstrated to deliver near-optimal solutions.

In this thesis, we consider reliability-aware VNF service provisioning in MEC. We distinguish our work from existing ones as follows. Most existing studies focused on the placement of service function chains in geo-distributed networks that consists of many powerful data centers. We provide the very first study of reliability-aware VNF placement in MEC, where computing resources are pushed to the edge of the access network in the close proximity of users. We assume that the required network function service of a user can be implemented by a single, consolidated VNF instance. Additionally, existing studies mainly focused on the end-to-end requirements of requests specified as the probability of component failures. In this thesis, we differentiate the reliability requirements of different users by providing differentiating numbers of sec-

ondary VNF instances for ensuring their reliability requirements. Meanwhile, most existing studies only provided heuristic solutions without performance guarantees. We here devised efficient algorithms with performance guarantees for the problem of concern. In addition, this thesis formulates a generalized version of the well-known Generalized Assignment Problem (GAP) [25], thus the solutions provided in this thesis may be of independent interest to other domains.

1.5 Thesis Contributions

The main contribution of this thesis is to provide a systematic study on resource allocation and provisioning for SDN- and NFV-based next-generation network architecture with the objective of provisioning elastic, flexible, and pervasive network services, through formulating non-trivial optimization problems, proposing new modeling of network resource usages, and developing novel optimization frameworks and algorithms for the problems. The contributions of this thesis are summarized as follows.

For the online unicasting and multicasting problems in SDNs, we devise online algorithms with performance guarantees in Chapter 2. Specifically, we consider dynamic admissions of a sequence of online unicast or multicast requests that arrive one by one without the knowledge of future arrivals. The proposed algorithms handle request admissions with the objective of maximizing the network throughput, in terms of the acceptance ratio of user requests. The contributions include (1) a novel model for resource usages that accurately captures the usage costs of node and link resources in the admission of a sequence of unicast or multicast requests without the knowledge of future request arrivals, (2) algorithms with provable performance guarantees for online unicasting and multicasting, and (3) the non-trivial proofs of the competitive ratios of the proposed algorithms. To the best of our knowledge, we were the very first to study online unicasting and multicasting in SDNs by taking both node and link constraints into consideration, and devise the very first online algorithms with provable competitive ratios for the problems.

For the problem of realizing user requests with each specifying a sequence of network functions in software-defined networks, we consider how to maximize the network throughput in Chapter 3. We explore the non-trivial trade-off between the accuracy/quality of a solution and the running time of obtaining the solution, by devising two algorithms where the first algorithm has higher time complexity and delivers near-optimal solutions and the second algorithm delivers comparable solutions yet has much shorter running time. The contributions of this topic include: (1) an Integer Linear Programming (ILP) solution that delivers optimal solutions to the problem when the problem size is small, (2) two algorithms for admitting user requests with network function requirement by exploring the non-trivial trade-off between the accuracy/quality of a solution and the running time of obtaining the solution, and (3) an online algorithm for dynamic request admissions.

The problem of maximizing network throughput by dynamically scaling VNF instances while minimizing the accumulative instance scaling cost is considered in Chapter 4. We propose a unified framework for two types of resource scaling – vertical scaling and horizontal scaling, while taking the QoS requirements of user requests into account, which is the very first to the best of our knowledge. The contributions of this work are (1) a non-trivial reduction of the problem of concern into several classic problems, and (2) an algorithm that has been empirically demonstrated to deliver near-optimal solutions.

Approximation and exact algorithms for reliability-aware provisioning of VNF instances are proposed in Chapter 5. A novel optimization problem for placing multiple VNF instances in order to meet the reliability requirement of each request is considered. The contributions of this research topic are (1) an ILP solution for the problem when the problem size is small, (2) an approximation algorithm with a logarithmic approximation ratio, and (3) constant approximation and exact algorithm for two special cases of the problem, respectively.

It is also worth noting that the proposed algorithms and developed algorithm

design and analysis techniques may of independent interests in many other areas, especially in combinatorial optimization.

1.6 Thesis Overview

The remainder of the thesis is organized as follows. Chapter 2 will study network throughput maximization in software-defined networks by admitting as many online unicasting and multicasting requests as possible. Chapter 3 will consider network throughput maximization of user requests with service function chain requirements by implementing each service function chain in a consolidated middlebox. Chapter 4 will investigate dynamic scaling of resource allocation in order to achieve better network throughput, while taking into account the QoS requirements of requests. Chapter 5 will deal with the provisioning of virtualized network function instances subject to reliability requirements of requests. Chapter 6 will summarize the thesis and discuss future work.

Chapter 2

Online Unicasting and Multicasting in Software-Defined Networks

2.1 Introduction

Software-Defined Networking (SDN) has emerged as a key enabling technology of the next-generation network architecture that creates an opportunity to tackle a long-standing problem in traditional networks, by moving the network control logic from the underlying routers and switches to a logically centralized controller and offering the programmability of the network [1, 24, 61, 81]. SDN now is becoming a key technology for the next-generation network architecture, including Internet backbone networks and data center networks such as Google's B4 [68]. However, one fundamental problem in the adoption of the SDN technology by network and cloud service providers is how to enable efficient data traffic routing in the network such that the network throughput is maximized, considering that not only do SDNs usually have both node and link resource capacity constraints but also user requests arrive into SDNs dynamically without the knowledge of future request arrivals.

Low-latency, high-performance matching of forwarding rules in each TCAM plays a vital role in terms of routing efficiency. Therefore, the forwarding table at each switch node typically is implemented by a special yet expensive memory – Ternary Content Addressable Memory (TCAM) that supports fast, parallel lookups [81, 121]. However, the number of entries in each TCAM forwarding table is usually limited to

several thousands [81]. The highly restricted capacity of TCAM has been recognized as the main bottleneck to the scalability of SDN [24, 79, 81, 107]. Efficient utilization of forwarding tables to serve a scaling number of forwarding rules while meeting network resource capacity constraints is an important and challenging research topic. In addition, with ever-growing bandwidth demands by users, it urgently needs efficient routing mechanisms that take into account both the TCAM capacity at each switch node and the bandwidth capacity at each link in the network. Furthermore, the dynamics of online user requests without the knowledge of future request arrivals makes the design of efficient routing protocols for SDNs difficult and challenging.

In this chapter, we study online unicasting and multicasting in SDNs with the aim to maximize the network throughput (or the acceptance rate of requests), where a sequence of unicast or multicast requests arrive one by one without the knowledge of future arrivals. Each incoming request is either accepted or rejected immediately, depending on whether there are sufficient resources to meet its resource demands. Although extensive effort on online unicasting and multicasting in traditional networks has been conducted in the past, most studies considered either the node resource capacity [80, 92, 93] or the link resource capacity [6, 109], none of the studies jointly considered these two types of resource capacities: the forwarding table capacities at switch nodes and the bandwidth capacities at links. We will jointly take into account the resource capacities at both nodes and links, by proposing a novel cost model that unifies the usage costs of these two types of resources simultaneously. It is noticed that there are several recent studies on unicast and multicast routing for SDNs [1, 24, 61, 81]. They, however, only considered the forwarding table capacity without taking into account the link bandwidth constraint. For example, the authors in [1, 24] studied unicast routing by first reducing the node TCAM capacity constraint into the node-degree constraint, followed by finding a node-degree-constrained maximum flow for a given set of unicast routing requests. Such the reduction approach, however, is not applicable in practice, since the TCAM capacity at each node usually is far greater

than the maximum degree of nodes in the network. Huang et al. [61] dealt with the admission of a single multicast request. They reduced the problem of finding a multicast tree for the request to the problem of finding a node-degree-constrained multicast tree that is NP-hard [41]. An approximate solution to the latter returns an approximate solution to the former. In contrast, in this chapter, we consider dynamic admissions of a sequence of online unicast or multicast requests without the knowledge of future request arrivals. The mentioned existing algorithms are not applicable to the problem of concern. Instead, new online algorithms as well as the analysis techniques for online unicasting and multicasting need to be developed.

The main contributions of this chapter are summarized as follows. We study the online unicasting and multicasting problems in SDNs with the aim to maximize the network throughput, by taking both node and link capacities and user bandwidth demands into consideration. We first propose a novel cost model to accurately capture the usage costs of node and link resources in the admission of a sequence of unicast or multicast requests without the knowledge of future request arrivals. We then devise efficient online algorithms with provable competitive ratios for them. We finally evaluate the performance of the proposed algorithms through experimental simulations. The simulation results demonstrate that the proposed algorithms are very promising. To the best of our knowledge, we are the very first to study online unicasting and multicasting in SDNs by taking both node and link constraints into consideration, and devise the very first online algorithms with provable competitive ratios for the problems. In particular, the construction of the auxiliary graph and assignment of their edge weights may be of independent interest and can be applied to other optimization problems in networks.

The remainder of this chapter is organized as follows. Section 2.2 will introduce the system model, notions and notations, and provide problem definitions. Section 2.3 will build a cost model for resource usages. Sections 2.4 and 2.5 will propose online algorithms and analyze their competitive ratios for online unicasting and multicasting

in software-defined networks, respectively. Section 2.6 will evaluate the performance of the proposed algorithms by experimental simulation, and Section 2.7 will summarize the chapter.

2.2 Preliminaries

In this section we first introduce the system model, we then introduce the notions and notations, and we finally define the problem precisely.

2.2.1 System Model

We consider a software-defined network (SDN) $G = (V, E)$, where V is the set of SDN-enabled switch nodes, and E is the set of links that connect the switches. There is an SDN controller co-located with a switch node in G to handle the admission of unicast or multicast requests, by installing forwarding rules to the forwarding tables at switch nodes and allocating bandwidth on the links along the routing paths or trees in G for the requests. Each switch node $v \in V$ is equipped with a TCAM forwarding table for packet forwarding, and the table capacity is L_v rule entries. Each link $e = (u, v) \in E$ has a bandwidth capacity B_e (or $B_{(u,v)}$).

2.2.2 TCAM and Routing Rule Matching in SDNs

The flow table at each SDN switch contains a list of *rules* that determine how packets have to be processed by the switch. Each rule consists of three main components: a *matching pattern*, *an actions field*, and a *priority* [9]. The purpose of the matching pattern is to test if a packet belongs to a flow according to the attributes of the packet such as its source IP address, communication protocol, etc. All packets that are matched by the same matching pattern are considered to belong to the same flow. The actions specified in the actions field of a rule are applied to every packet of the corresponding flow. Some common actions including forwarding, dropping, or rewriting the packets. As a packet may match multiple matching patterns, each rule is also associated with

a priority and only the actions of the matching rule with the highest priority are applied. It is worth noting that a rule may contain other additional components, such as a *counter*, which is used to keep track of the number of packets that have been processed according to this rule.

We adopt the TCAM usage model from [13, 104]. That is, TCAM only stores the matching pattern and priority of every rule, whereas other components of the rule are stored in external memory, e.g., Static Random Access Memory (SRAM) [9]. For each incoming packet, the switch will first undergo a parallel lookup operation to find a matching rule with the highest priority. Having found (the index of) the matching rule for the packet, the action part of the rule is retrieved from SRAM and the specified actions are applied to the packet. Consequently, only one forwarding entry needs to be stored in TCAM for both unicast and multicast requests. Due to the significantly higher cost and smaller capacity of TCAM, we here focus on the capacity constraint of TCAM.

2.2.3 User Routing Requests

We consider two types of user routing requests: unicast and multicast requests that arrive into the system one by one. Let $r_k = (s_k, t_k; b_k)$ be the kth unicast request arrived into the system, where b_k is the bandwidth resource demand of r_k from s_k to t_k. Similarly, let $r_k = (s_k, D_k; b_k)$ be the k-th multicast request, where s_k is the source switch node, D_k is the destination set of switch nodes with $D_k \subseteq V$, and b_k is the amount of demanded bandwidth by the request. Following existing studies [1, 24, 58], we assume that the bandwidth demand of a request can be derived from historical traces of the same or similar request demands. Even the demand of a request varies over time, it can be accurately predicted by making use of its historical traces. For example, we can adopt a popular prediction method such as the auto-regressive moving average method for this purpose. We further assume that the demanded bandwidth b_k by each request r_k is an integer and at least one unit bandwidth, i.e.,

$b_k \geq 1$ for any k.

Given that the forwarding table at each switch node and bandwidth resource at each link are limited, an incoming request will be admitted by the system only if there is a routing path (for the unicast request) or a multicast tree (for the multicast request) that has enough available resources to meet its demands. The system will allocate its demanded resources to each admitted request, and an admitted request will release the resources allocated to it back to the system once it finishes. Otherwise, if the network cannot meet the resource demands of a request, the request will be rejected. A rejected request can be re-submitted to the network, and it can be admitted if there are sufficient available resources in the network to meet its demand.

2.2.4 Competitive Ratios of Online Algorithms

If a sequence of unicast or multicast requests is given in advance, assume that there is an optimal offline algorithm for a maximization problem to admit or reject the requests. Let OPT be the cost of the optimal solution of the problem, and S be the cost of the solution delivered by an online algorithm $\mathcal{A}$ for the requests in the sequence that arrive one by one without the knowledge of future arrivals.The *competitive ratio* of online algorithm $\mathcal{A}$ is ξ (> 1) if $\frac{S}{OPT} \geq \frac{1}{\xi}$ for any instance of the maximization problem. Ideally, the competitive ratio is expected to be a small constant. However, the performance of the SDN can be as bad as $O(n)$ if there is no admission control on requests, since some requests may consume excessive resources in the network, later arrived requests cannot be admitted due to lack of demanded resources, where n is the network size.

2.2.5 Problem Definitions

Given a software-defined network $G = (V, E)$, where each $v \in V$ be a switch node equipped with a TCAM forwarding table of size L_v for packet routing, and let B_e be the bandwidth capacity of link $e \in E$, and a sequence of unicast requests $(s_k, t_k; b_k)$ arrives

into the system one by one without the knowledge of future arrivals, *the network capacity maximization problem for online unicasting* in G is to maximize the network throughput (the accumulated bandwidth of admitted unicast requests), subject to resource capacity constraints on switch nodes and links in the network.

The network capacity maximization problem for online multicasting can be defined similarly. That is, given an SDN $G = (V, E)$, let each $v \in V$ be a switch node equipped with a TCAM of forwarding table of size L_v for packet routing, and let B_e be the bandwidth capacity of each link $e \in E$. Given a sequence of multicast requests $(s_k, D_k; b_k)$ that arrives into the system one by one without the knowledge of future arrivals, the problem is to maximize the network throughput (the accumulated bandwidth of admitted multicast requests), subject to resource constraints on switch nodes and links in the network.

2.3 The Usage Costs of Resources of Links and Nodes

Given a software-defined network $G = (V, E)$, a metric is needed to model the usage costs of resources at its switch nodes and links. One important characteristic of such resource usages is that *the marginal costs* of resource usages inflate with the increase in the workloads of the resources. Compared with a lightly-loaded switch node, a heavily-loaded switch node will spend more time and consume more energy on matching a forwarding rule for an incoming packet, because more rules in such a heavily-loaded switch node need to be considered. For example, in the process of installing a forwarding rule into a switch node with the TCAM capacity of 2,000 entries, existing forwarding rules must be matched to make sure the new forwarding rule does not exist in the TCAM prior to its installation. This process takes five seconds for the first 1,000 entries, while it takes almost two minutes for the next 1,000 entries [81], which directly translates into more energy costs. Thus, when admitting a request, we should make use of lower-cost and under-loaded nodes and links to admit the request, rather than over-loaded ones. Through this observation, we will

make use of exponential functions to model the costs of resource usages, which are the functions of available amounts and the resource workloads. Since usages of overloaded/saturated resources usually incur higher costs by the adopted exponential functions, it can effectively avoid the usage of overloaded/saturated resources. The exponential functions for the usage costs of edge and node resources are given as follows.

Let $L_v(k)$ and $B_e(k)$ be the numbers of available entries in the forwarding table at switch node $v \in V$ and the residual bandwidth on link $e \in E$, respectively, when the kth request r_k arrives. We use an exponential function to model the cost $c_v(k)$ of using the resource at each switch node v by request r_k, which is defined as

$$c_v(k) = L_v(\alpha^{1-\frac{L_v(k)}{L_v}} - 1), \tag{2.1}$$

where $\alpha > 1$ is a constant to be determined later, and $1 - \frac{L_v(k)}{L_v}$ is *the utilization ratio* of the forwarding table of v when request r_k arrives.

The cost $c_e(k)$ of using the bandwidth of link $e \in E$ by request r_k can be similarly defined as

$$c_e(k) = B_e(\beta^{1-\frac{B_e(k)}{B_e}} - 1), \tag{2.2}$$

where $\beta > 1$ is a constant that is similar to α to be determined later, and $1 - \frac{B_e(k)}{B_e}$ is the utilization ratio of the bandwidth of link e when request r_k arrives.

2.4 An Online Algorithm for Dynamic Unicast Routing

In this section, we first describe an online algorithm for the network capacity maximization problem for online unicasting. We then analyze the competitive ratio and time complexity of the proposed algorithm.

2.4.1 Online Algorithm

The basic idea of the online algorithm is to transform the network capacity maximization problem for online unicasting in the original SDN G into the problem of a series of finding a shortest path in edge-weighted, directed auxiliary graphs. The key is how to jointly consider node and edge costs in a unified way in the construction of each auxiliary graph $G'(k)$ for request r_k.

For each unicast request r_k with (s_k, t_k, b_k), to model the usage costs of the resources at nodes and links in the network G by admitting r_k, an edge-weighted, directed graph $G'(k) = (V'(k), E'(k); \omega)$ will be constructed from $G = (V, E)$, and a shortest path in $G'(k)$ from node s'_k to node t'_k will be found. Note that the weight of each edge in $G'(k)$ is *the normalized usage cost* of its corresponding switch node or link, which is an exponential function of the available amount and the workload of the resource at the node or the link. In the following, we first detail the construction of $G'(k)$. We then devise an efficient online algorithm for the network capacity maximization problem for online unicasting.

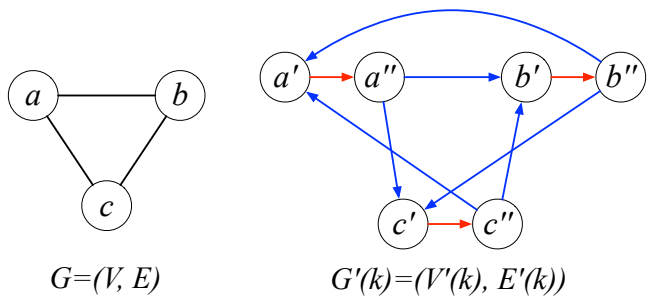


Figure 2.1: The construction of $G'(k) = (V'(k), E'(k))$ for an SDN $G = (V, E)$ for online unicasting when request r_k arrives.

The edge-weighted, directed graph $G'(k) = (V'(k), E'(k); \omega)$ is constructed from $G = (V, E)$ as follows. For each switch node $v \in V$, two nodes v' and v'' are added to $V'(k)$, i.e., $V'(k) = \{v', v'' \mid v \in V\}$, and a directed edge $\langle v', v'' \rangle$ is added to $E'(k)$. For each link $(u, v) \in E$, two directed edges $\langle u'', v' \rangle$ and $\langle v'', u' \rangle$ are added to $E'(k)$, i.e., $E'(k) = \{\langle v', v'' \rangle \mid v \in V\} \cup \{\langle v'', u' \rangle, \langle u'', v' \rangle \mid (u, v) \in E\}$. For brevity, we refer

to the edges in $G'(k)$ derived from switch nodes and links of G as the *node-derived edges* and *link-derived edges*, and denote by $E'_v(k)$ and $E'_e(k)$ the sets of node-derived edges and link-derived edges in $G'(k)$, respectively. Clearly, $E'(k) = E'_e(k) \cup E'_v(k)$ and $E'_e(k) \cap E'_v(k) = \emptyset$. Figure 2.1 illustrates the construction of $G'(k)$.

Depending on its type (node-derived or link-derived edge) and the usage cost of the resource it represents, each edge $e \in E'(k)$ is assigned a weight as follows.

$$\omega_e(k) = \begin{cases} \frac{c_v(k)}{L_v} = \alpha^{1-\frac{L_v(k)}{L_v}} - 1 & \text{if } e = \langle v', v'' \rangle \in E'_v(k), \\ \frac{c_{u,v}(k)}{B_{(u,v)}} = \beta^{1-\frac{B_{(u,v)}(k)}{B_{\langle u,v \rangle}}} - 1 & \text{if } e = \langle u'', v' \rangle \in E'_e(k), \end{cases} \tag{2.3}$$

where the weight of each node-derived edge reflects the forwarding table entry usage on its corresponding switch node, while the weight of each link-derived edge reflects the bandwidth usage on its corresponding link. Notice that the weight of each edge is the normalized usage cost of the resource it represents.

An edge $e' \in E'_e(k)$ derived from a link $e \in E$ is omitted if its residual bandwidth is strictly less than b_k, because e cannot meet the bandwidth demand of request r_k, and thus plays no role in the admission of the kth request. For simplicity, the resulting graph after pruning some edges from it is still denoted as $G'(k)$. Let $P(k)$ be a shortest path in $G'(k)$ from s'_k to t'_k. Following the construction of $G'(k)$, the edges in $P(k)$ are the node-derived and link-derived edges alternatively.

To maximize the network throughput (the accumulated bandwidth of all admitted requests), *an admission control policy* will be adopted. That is, when the length of a shortest path in $G'(k)$ from s'_k to t'_k for each incoming request r_k is greater than a given threshold, the request will be rejected no matter whether there are sufficient resources to admit the request. We here adopt the following admission control policy: a unicast request r_k will be rejected, if (i) the weighted sum of node-derived edges in $P(k)$ is greater than σ_v; or (ii) the weighted sum of the link-derived edges in $P(k)$ is greater than σ_e, where σ_v $(= |V| - 1 = n - 1)$ and σ_e $(= |V| - 1 = n - 1)$ are pre-determined thresholds. In other words, an incoming request r_k is admitted if and only if there

exists a shortest path $P(k)$ in $G'(k)$ from s'_k to t'_k such that

(i) $\sum_{e=\langle v',v''\rangle \in P(k)\cap E'_v(k)} \omega_e(k) \leq \sigma_v$, and

(ii) $\sum_{e=\langle v'',u'\rangle \in P(k)\cap E'_e(k)} \omega_e(k) \leq \sigma_e$.

The detailed algorithm for online unicasting is given in Algorithm 2.1.

Algorithm 2.1 Online routing algorithm for unicast requests

Input: a software define network $G = (V, E)$ and the kth unicast request $r_k = (s_k, t_k; b_k)$;

Output: Maximize the network throughput by admitting or rejecting each arriving unicast request r_k. If admitted, a routing path for r_k will be found.

```
/* Ensure that the switch table of each node v can contain at least one entry */
for each node v ∈ V do
    if the table size at node v is full, i.e., L_v(k) = 0 then
        Remove v and its incident links from G;
    end if
end for
/* Ensure that the residual bandwidth capacity of each link e is at least b_k */
for each link e ∈ E do
    if the residual bandwidth at link e is less than b_k, i.e., B_e(k) < b_k then
        Remove e from G;
    end if
end for
Construct an edge-weighted, directed graph G'(k) = (V'(k), E'(k); ω) from the
resulting subgraph of G and calculate the edge weights according to the resource
utilization when request r_k arrives;
Find a shortest path P(k) in G'(k) from s'_k to t'_k;
if P(k) does not exist then
    Reject unicast request r_k;
else
    if (Σ_{e∈P(k)∩E'_v(k)} ω_e(k) ≤ σ_v) and (Σ_{e∈P(k)∩E'_e(k)} ω_e(k) ≤ σ_e) then
        Admit unicast request r_k with P(k) as its routing path;
        Update the residual resource capacities of links in E and nodes in V;
    else
        Reject unicast request r_k;
    end if
end if
```

2.4.2 Algorithm Analysis

In the following, we analyze the performance of the proposed algorithm. Let $L_{\min}$ be the minimum TCAM size, i.e., $L_{\min} = \min\{L_v \mid v \in V\}$, let $B_{\min}$ be the minimum bandwidth capacity among links, i.e., $B_{\min} = \min\{B_e \mid e \in E\}$, and $b_{\max}$ the maximum bandwidth demand by any unicast request, i.e., $b_{\max} = \{b_{k'} \mid 1 \leq k' \leq k\}$.

We first show the upper bound on the cost of admitted unicast requests as of the arrival of request r_k by Algorithm 2.1 by the following lemma.

Lemma 2.1. *Given an SDN $G = (V, E)$ with forwarding table capacity L_v at each switch node $v \in V$ and link bandwidth capacity B_e at each link $e \in E$, denote by $\mathcal{S}(k)$ the set of unicast requests admitted by the online algorithm, Algorithm 2.1, until the arrival of request r_k. Then, the cost sums of nodes and links in G are*

$$\sum_{v \in V} c_v(k) \leq |\mathcal{S}(k)|(\sigma_v + n - 1) \log \alpha, \tag{2.4}$$

and

$$\sum_{e \in E} c_e(k) \leq \mathbb{B}(k)(\sigma_e + n - 1) \log \beta, \tag{2.5}$$

respectively, where α and β are constants with $2|V| \leq \alpha \leq 2^{L_{\min}}$ and $2|V| \leq \beta \leq 2^{B_{\min}/b_{\max}}$, and $\mathbb{B}(k) = \sum_{k' \in \mathcal{S}(k)} b_{k'}$.

Proof. Consider a unicast request $r_{k'} \in \mathcal{S}(k)$ admitted by the online algorithm. Then, for any switch node $v \in V$, we have

$$\begin{aligned}
& c_v(k'+1) - c_v(k') \\
=& L_v(\alpha^{1-\frac{L_v(k'+1)}{L_v}} - 1) - L_v(\alpha^{1-\frac{L_v(k')}{L_v}} - 1) \\
=& L_v\big(\alpha^{1-\frac{L_v(k'+1)}{L_v}} - \alpha^{1-\frac{L_v(k')}{L_v}}\big) \\
=& L_v\alpha^{1-\frac{L_v(k')}{L_v}}(\alpha^{\frac{L_v(k')-L_v(k'+1)}{L_v}} - 1) \\
\leq& L_v\alpha^{1-\frac{L_v(k')}{L_v}}(\alpha^{\frac{1}{L_v}} - 1)
\end{aligned} \tag{2.6}$$

$$
\begin{aligned}
&= L_v \alpha^{1-\frac{L_v(k')}{L_v}} (2^{\frac{1}{L_v}\log\alpha} - 1) \\
&\leq L_v \alpha^{1-\frac{L_v(k')}{L_v}} (\log\alpha / L_v) && (2.7) \\
&= \alpha^{1-\frac{L_v(k')}{L_v}} \log\alpha, && (2.8)
\end{aligned}
$$

where Inequality (2.6) holds because at most one routing entry is added to the forwarding table of node v, and Inequality (2.7) holds because $2^x - 1 \leq x$ with $0 \leq x \leq 1$, and $(1/L_v)\log\alpha \leq (1/L_v)L_{\min} \leq (1/L_v)L_v = 1$.

Similarly, for any edge $e \in E$, we have

$$
\begin{aligned}
& c_e(k'+1) - c_e(k') \\
&= B_e(\beta^{1-\frac{B_e(k'+1)}{B_e}} - 1) - B_e(\beta^{1-\frac{B_e(k')}{B_e}} - 1) \\
&= B_e \beta^{1-\frac{B_e(k')}{B_e}} (\beta^{\frac{B_e(k')-B_e(k'+1)}{B_e}} - 1) \\
&\leq B_e \beta^{1-\frac{B_e(k')}{B_e}} (\beta^{\frac{b_{k'}}{B_e}} - 1), && (2.9) \\
&= B_e \beta^{1-\frac{B_e(k')}{B_e}} (2^{\frac{b_{k'}}{B_e}\log\beta} - 1) \\
&\leq \beta^{1-\frac{B_e(k')}{B_e}} \cdot b_{k'} \cdot \log\beta, && (2.10)
\end{aligned}
$$

where Inequality (2.9) follows since at most $b_{k'}$ bandwidth units of link e for request $r_{k'}$ are reserved, and Inequality (2.10) follows because $\frac{b_{k'}}{B_e}\log\beta \leq \frac{b_{k'}}{B_e} \cdot \frac{B_{\min}}{b_{\max}} \leq \frac{b_{k'}}{B_e} \cdot \frac{B_e}{b_{k'}} = 1$, and $2^x - 1 \leq x$ with $0 \leq x \leq 1$

We then calculate the cost sum of all nodes or links of G when admitting request $r_{k'}$. Notice that if an edge in $G'(k')$ is not in $P(k')$, its cost does not change after the admission of request $r_{k'}$. The difference in the cost sum of nodes before and after admitting request $r_{k'}$ thus is

$$
\begin{aligned}
& \sum_{v \in V} \big(c_v(k'+1) - c_v(k')\big) \\
= & \sum_{\langle v', v'' \rangle \in P(k') \cap E'_v(k')} (c_v(k'+1) - c_v(k')) \\
\leq & \sum_{\langle v', v'' \rangle \in P(k') \cap E'_v(k')} (\alpha^{1-\frac{L_v(k')}{L_v}} \cdot \log\alpha), \qquad \text{by Inequality (2.8)}
\end{aligned}
$$

$$
\begin{aligned}
&= \log \alpha \sum_{\langle v', v'' \rangle \in P(k') \cap E'_v(k)} \alpha^{1-\frac{L_v(k')}{L_v}} \\
&= \log \alpha \sum_{\langle v', v'' \rangle \in P(k') \cap E'_v(k')} \left((\alpha^{1-\frac{L_v(k')}{L_v}} - 1) + 1 \right) \\
&= \log \alpha \sum_{\langle v', v'' \rangle \in P(k') \cap E'_v(k')} \left(w_{\langle v', v'' \rangle}(k') + 1 \right) \\
&= \log \alpha \left(\sum_{\langle v', v'' \rangle \in P(k') \cap E'_v(k')} \omega_{\langle v', v'' \rangle}(k') + \sum_{\langle v', v'' \rangle \in P(k') \cap E'_v(k')} 1 \right) \\
&\leq (\sigma_v + (n-1)) \log \alpha, \qquad (2.11)
\end{aligned}
$$

where Inequality (2.11) holds because request $r_{k'}$ is admitted only if it meets the admission control policy (i), and any routing path in $G'(k')$ contains no more than $n-1$ node-derived edges.

Similarly, the cost sum of edges by request $r_{k'}$ is

$$
\begin{aligned}
&\sum_{e \in E} \left(c_e(k'+1) - c_e(k') \right) \\
&= \sum_{e' \in P(k') \cap E'_e(k')} (c_e(k'+1) - c_e(k')) \\
&\leq \sum_{e' \in P(k') \cap E'_e(k')} (\beta^{1-\frac{B_e(k')}{B_e}} \cdot b_{k'} \cdot \log \beta), \qquad \text{by Inequality (2.10)} \\
&= b_{k'} \log \beta \sum_{e' \in P(k') \cap E'_e(k)} (\beta^{1-\frac{B_e(k')}{B_e}}) \\
&= b_{k'} \log \beta \sum_{e' \in P(k') \cap E'_e(k')} ((\beta^{1-\frac{B_e(k')}{B_e}} - 1) + 1) \\
&= b_{k'} \log \beta \left(\sum_{e' \in P(k') \cap E'_e(k')} (\beta^{1-\frac{B_e(k')}{B_e}} - 1) + \sum_{e' \in P(k')} 1 \right) \\
&= b_{k'} \log \beta \left(\sum_{e' \in P(k' \cap E'_e(k'))} \omega_{e'}(k') + \sum_{e' \in P(k') \cap E'_e(k')} 1 \right) \\
&\leq b_{k'} \cdot (\sigma_e + n - 1) \log \beta. \qquad (2.12)
\end{aligned}
$$

where Inequality (2.12) holds because request $r_{k'}$ is admitted only if it meets the admission control policy (ii), and follows the fact that any routing path in $G'(k')$

contains no more than $n-1$ link-derived edges.

Notice that $c_v(1) = c_e(1) = 0$ for all $v \in V$ and $e \in E$. Thus, the cost sum of all nodes when request r_k arrives is

$$\begin{aligned}
&\sum_{v\in V} c_v(k) \\
=&\sum_{k'=1}^{k-1}\sum_{v\in V}(c_v(k'+1)-c_v(k')) \\
=&\sum_{k'\in\mathcal{S}(k)}\sum_{v\in V}(c_v(k'+1)-c_v(k')) \\
\leq&\sum_{k'\in\mathcal{S}(k)}((\sigma_v+(n-1))\log\alpha) \qquad (2.13)\\
=&((\sigma_v+(n-1))\log\alpha)\sum_{k'\in\mathcal{S}(k)}1 \\
=&|\mathcal{S}(k)|(\sigma_v+(n-1))\log\alpha,
\end{aligned}$$

where Inequality (2.13) follows from Inequality (2.11).

Likewise, the cost sum of all edges for routing $|\mathcal{S}(k)|$ unicast requests by the online algorithm is

$$\begin{aligned}
&\sum_{e\in E} c_e(k) \\
=&\sum_{k'=1}^{k-1}\sum_{e\in E}(c_e(k'+1)-c_e(k')) \\
=&\sum_{k'\in\mathcal{S}(k)}\sum_{e\in E}(c_e(k'+1)-c_e(k')) \\
\leq&\sum_{k'\in\mathcal{S}(k)}(b_{k'}(\sigma_e+(n-1))\log\beta) \qquad (2.14)\\
=&((\sigma_e+(n-1))\log\beta)\sum_{k'\in\mathcal{S}(k)}b_{k'} \\
\leq&\mathbb{B}(k)(\sigma_e+(n-1))\log\beta,
\end{aligned}$$

where $\mathbb{B}(k) = \sum_{k'\in\mathcal{S}(k)} b_{k'}$, and Inequality (2.14) follows from Inequality (2.12). □

We then provide a lower bound on the length of the routing path to which an optimal offline algorithm routes a request that is rejected by the online algorithm in the following lemma.

Lemma 2.2. *Let $\mathcal{R}(k)$ be the set of unicast requests that are admitted by an optimal offline algorithm yet rejected by the online algorithm, Algorithm 2.1, prior to the arrival of unicast request r_k, and let $P_{opt}(k')$ be the routing path in $G'(k)$ found by the optimal offline algorithm for request $r_{k'} \in \mathcal{R}(k)$. Then, for any request $r_{k'} \in \mathcal{R}(k)$, we have*

$$\sum_{e \in P_{opt}(k')} \omega_e(k') \geq \min\{\sigma_v, \sigma_e\} = |V| - 1 = n - 1, \tag{2.15}$$

where α and β are constants with $2|V| = 2n \leq \alpha \leq 2^{L_{\min}}$ and $2|V| = 2n \leq \beta \leq 2^{B_{\min}/b_{\max}}$.

Proof. Suppose that given an SDN G, there are an optimal offline algorithm and the proposed online algorithm for the problem of concern in G. Assuming that each request is revealed to both algorithms one by one. Each algorithm either admits a request by allocating the demanded resources to the request or rejects the request. The optimal offline algorithm may reject a request that is admitted by the proposed online algorithm or vice versa. Thus, for a given monitoring period, the proposed online algorithm and the optimal offline algorithm may accept different sets of requests, resulting in a difference between the resource availability in the resulting networks by these two algorithms when a request arrives. Denote by $G_{opt}(k')$ and $G(k')$ the networks to which the optimal offline algorithm and the proposed online algorithm are applied prior to the arrival of request $r_{k'}$, respectively.

Consider a unicast request $r_{k'}$ that is admitted by the optimal offline algorithm yet rejected by the proposed online algorithm. Since $r_{k'}$ is admitted by the optimal offline algorithm, it means that the optimal offline algorithm is able to admit $r_{k'}$ into $G_{opt}(k')$ using a path $P_{opt}(k')$ in $G_{opt}(k')$. We can check whether $G'(k')$ contains $P_{opt}(k')$, and there are exactly two cases. Case 1: every node and edge in $P_{opt}(k')$ has its corresponding edges in $G'(k')$, or Case 2: at least one node or one link in $P_{opt}(k')$

does not have a corresponding edge in $G'(k')$, because its available resource cannot meet the demand of $r_{k'}$. In the following, we will argue that in both cases,

$$\sum_{e \in P_{opt}(k')} \omega_e(k') \geq \min\{\sigma_v, \sigma_e\} = |V| - 1 = n - 1,$$

where $\omega_e(k')$ is calculated based on the availability and workload of the resource in $G(k')$.

Case 1: If every node and edge in $P_{opt}(k')$ has a corresponding edge in $G'(k')$ as well, there is at least one path in $G'(k')$ from $s'_{k'}$ to $t'_{k'}$, namely the one corresponds to $P_{opt}(k')$. Consequently, the proposed online algorithm must be able to find a path $P(k')$ such that $P(k')$ can meet the resource demand of request $r_{k'}$ and

$$\sum_{e \in P(k')} \omega_e(k') \leq \sum_{e \in p} \omega_e(k')$$

for any path p in $G'(k')$ from $s'_{k'}$ to $t'_{k'}$ including $P_{opt}(k')$. Since $r_{k'}$ is rejected by the online algorithm, the length of $P(k')$ is no less than the given threshold, i.e.,

$$\sum_{e \in P(k')} \omega_e(k') \geq \min\{\sigma_v, \sigma_e\}.$$

Thus,

$$\min\{\sigma_v, \sigma_e\} \leq \sum_{e \in P(k')} \omega_e(k') \leq \sum_{e \in P_{opt}(k')} \omega_e(k').$$

Case 2: At least one node or one edge in $P_{opt}(k')$ does not have a corresponding edge in $G'(k')$. Recall that a node or a link is not included only if its residual capacity cannot satisfy the resource demands of $r_{k'}$. This case thus can be further divided into two subcases: (a) if there is a node with no available table entry to route the message for the unicast request, then there is a node-derived edge $e' = \langle v', v'' \rangle \in P_{opt}(k')$ in

$G'(k')$ such that $L_v(k') < 1$. Consequently, the length of $P_{opt}(k')$ is greater than σ_v:

$$\begin{aligned}
&\sum_{e \in P_{opt}(k')} \omega_e(k') \\
\geq &\omega_{\langle v', v'' \rangle}(k') \\
= &\alpha^{1-\frac{L_v(k')}{L_v}} - 1 \\
> &\alpha^{1-\frac{1}{L_v}} - 1, \qquad \text{since } L_v(k') < 1 \\
\geq &\alpha^{1-\frac{1}{\log \alpha}} - 1, \qquad \text{since } 2n \leq \alpha \leq 2^{L_{\min}} \leq 2^{L_v} \\
= &\frac{\alpha}{2} - 1 \geq \sigma_v, \qquad \text{by the assumption of that } \alpha \geq 2n.
\end{aligned}$$

(b) If there is an edge in any routing path found by the online algorithm without sufficient bandwidth to route request $r_{k'}$, then there exists an edge $e' = \langle v'', u' \rangle \in P_{opt}(k')$ in $G'(k')$ such that $B_{(v,u)}(k') < b_{k'}$. Therefore, the length of $P_{opt}(k')$ is greater than σ_e:

$$\begin{aligned}
&\sum_{e \in P_{opt}(k')} \omega_e(k') \\
\geq &\omega_{\langle v'', u' \rangle}(k') \\
= &\beta^{1-\frac{B_{(v,u)}(k')}{B_{(v,u)}}} - 1 \\
> &\beta^{1-\frac{b_{k'}}{B_{(u,v)}}} - 1, \qquad \text{since } B_{(v,u)}(k') < b_{k'} \\
\geq &\beta^{1-\frac{1}{\log \beta}} - 1, \qquad \text{since } 2n \leq \beta \leq 2^{B_{\min}/b_{\max}} \leq 2^{B_{(u,v)}/b_{k'}} \\
= &\frac{\beta}{2} - 1 \geq \sigma_e, \qquad \text{by the assumption of that } \beta \geq 2n.
\end{aligned}$$

□

We finally have the following theorem.

Theorem 2.1. *Given an SDN $G = (V, E)$ with both node and link capacities $L_v(\cdot)$ and $B_e(\cdot)$ for all $v \in V$ and $e \in E$, assume that there is a sequence of unicast requests $r_1 = (s_1, t_1; b_1), \ldots, r_k = (s_k, t_k; b_k)$ arriving one by one without the knowledge of future arrivals.*

There is an online algorithm, Algorithm 2.1, with the competitive ratio of $2(\gamma \log \alpha + \log \beta) + 1$ *for the network capacity maximization problem for online unicasting, which takes* $O(|V|^2)$ *time for each request, where* α *and* β *are constants with* $2|V| = 2n \leq \alpha \leq 2^{L_{\min}}$ *and* $2|V| = 2n \leq \beta \leq 2^{B_{\min}/b_{\max}}$*, and* $\gamma = \frac{B_{\min}}{\log \beta}$ *is a value with* $1 < \gamma \leq \frac{B_{\min}}{\log(2n)}$.

Proof. Let $\mathbb{B}_{opt}(k)$ be the total bandwidth of requests admitted by an optimal offline algorithm when request r_k arrives, we then have

$$
\begin{aligned}
&(n-1)(\mathbb{B}_{opt}(k) - \mathbb{B}(k)) \\
\leq &(n-1) \sum_{k' \in \mathcal{R}(k)} b_{k'} \\
= &\sum_{k' \in \mathcal{R}(k)} b_{k'}(n-1) \\
\leq &\sum_{k' \in \mathcal{R}(k)} b_{k'} \Big(\sum_{e \in P_{opt}(k')} \omega_e(k') \Big) \\
\leq &\sum_{k' \in \mathcal{R}(k)} b_{k'} \Big(\sum_{e \in P_{opt}(k)} \omega_e(k) \Big) && (2.16) \\
= &\sum_{k' \in \mathcal{R}(k)} b_{k'} \Big(\sum_{\langle v', v'' \rangle \in P_{opt}(k') \cap E'_v(k')} \omega_{\langle v', v'' \rangle}(k) + \sum_{\langle u'', v' \rangle \in P_{opt}(k') \cap E'_e(k')} \omega_{\langle u'', v' \rangle}(k) \Big) \\
= &\sum_{k' \in \mathcal{R}(k)} b_{k'} \Big(\sum_{\langle v', v'' \rangle \in P_{opt}(k') \cap E'_v(k')} \frac{c_v(k)}{L_v} + \sum_{\langle u'', v' \rangle \in P_{opt}(k') \cap E'_e(k')} \frac{c_{(u,v)}(k)}{B_{(u,v)}} \Big) \\
= &\sum_{k' \in \mathcal{R}(k)} \sum_{\langle v', v'' \rangle \in P_{opt}(k') \cap E'_v(k')} b_{k'} \frac{c_v(k)}{L_v} + \sum_{k' \in \mathcal{R}(k)} \sum_{\langle u'', v' \rangle \in P_{opt}(k') \cap E'_e(k')} b_{k'} \frac{c_{(u,v)}(k)}{B_{(u,v)}} \\
\leq &\sum_{v \in V} c_v(k) \sum_{k' \in \mathcal{R}(k)} \sum_{\langle v', v'' \rangle \in P_{opt}(k') \cap E'_v(k')} \frac{b_{k'}}{L_v} \\
&+ \sum_{(u,v) \in E} c_{(u,v)}(k) \sum_{k' \in \mathcal{R}(k)} \sum_{\langle u'', v' \rangle \in P_{opt}(k') \cap E'_e(k')} \frac{b_{k'}}{B_{(u,v)}} && (2.17) \\
= &\sum_{v \in V} c_v(k) \frac{\sum_{k' \in \mathcal{R}(k)} \sum_{\langle v', v'' \rangle \in P_{opt}(k') \cap E'_v(k')} b_{k'}}{L_v} \\
&+ \sum_{(u,v) \in E} c_{(u,v)}(k) \frac{\sum_{k' \in \mathcal{R}(k)} \sum_{\langle u'', v' \rangle \in P_{opt}(k') \cap E'_e(k')} b_{k'}}{B_{(u,v)}} \\
\leq &\sum_{v \in V} c_v(k) \gamma + \sum_{e \in E} c_{(u,v)}(k) && (2.18) \\
= &\gamma \sum_{v \in V} c_v(k) + \sum_{e \in E} c_{(u,v)}(k) && (2.19)
\end{aligned}
$$

$$
\begin{aligned}
&\leq \gamma |S(k)|(\sigma_v + (n-1)) \log \alpha \\
&\quad + \mathbb{B}(k)(\sigma_e + (n-1)) \log \beta, \qquad \text{by Lemma 2.1} \\
&= 2(n-1)(\gamma |S(k)| \log \alpha + \mathbb{B}(k) \log \beta). \qquad (2.20)
\end{aligned}
$$

Notice that Inequality (2.16) holds because the utilization of each resource does not decrease and consequently the weight of any edge in $G'(k)$ does not decrease with more request admissions, i.e., $\omega_e(k') \leq \omega_e(k)$ for any edge $e \in E'(k)$ and any k' with $1 \leq k' \leq k$. Inequality (2.17) holds since $\sum_{i=1}^{p} \sum_{j=1}^{q} A_i B_j \leq \sum_{i=1}^{p} A_i \sum_{j=1}^{q} B_j$ with $A_i \geq 0$ and $B_j \geq 0$.

The proof of Inequality (2.19) proceeds as follows. For any switch node $v \in V$, each forwarding table entry can be used to admit a request with bandwidth at most $b_{\max}(\leq B_{\min}/\log \beta = \gamma)$, and the forwarding table at each node v has L_v entries. Thus, the accumulated bandwidth of all unicast requests using switch node v as their relay node is no more than $L_v \cdot b_{\max}$. Hence, the accumulated bandwidth of all admitted requests through node v by an optimal offline algorithm is no more than $L_v \cdot b_{\max}$, i.e.,

$$\sum_{k' \in \mathcal{R}(k)} \sum_{\langle v', v'' \rangle \in P_{opt}(k') \cap E'_v(k')} b_{k'} \leq L_v \cdot b_{\max} \leq \gamma \cdot L_v,$$

and

$$\Big(\sum_{k' \in \mathcal{R}(k)} \sum_{\langle v', v'' \rangle \in P_{opt}(k') \cap E'_v(k')} b_{k'} \Big) / L_v \leq \gamma.$$

Meanwhile, all algorithms, including optimal offline algorithms for the problem of concern, the total amount of bandwidth used in any link is no more than its capacity, thus, for every link $e \in E$, the accumulated bandwidth of all admitted requests on it by an optimal offline algorithm is no more than its capacity, i.e.,

$$\sum_{k' \in \mathcal{R}(k)} \sum_{\langle u'', v' \rangle \in P_{opt}(k') \cap E'_e(k')} b_{k'} \leq B_e.$$

Therefore,

$$(\sum_{k' \in \mathcal{R}(k)} \sum_{\langle u'', v' \rangle \in P_{opt}(k') \cap E'_e(k')} b_{k'}) / B_e \leq 1.$$

By Inequality (2.20), we have

$$\begin{aligned} & \frac{\mathbb{B}_{opt}(k) - \mathbb{B}(k)}{\mathbb{B}(k)} \\ \leq & 2(\gamma \frac{|S(k)|}{\mathbb{B}(k)} \log \alpha + \log \beta) \\ \leq & 2(\gamma \log \alpha + \log \beta), \end{aligned} \tag{2.21}$$

where the last step follows because $\mathbb{B}(k) = \sum_{k' \in S(k)} b_{k'}$ and $b_{k'} \geq 1$. From Inequality (2.21), we have

$$\begin{aligned} & \frac{\mathbb{B}_{opt}(k)}{\mathbb{B}(k)} \\ \leq & 2(\gamma \log \alpha + \log \beta) + 1, \\ = & O(\log n), \qquad \text{when } \alpha = \beta = 2|V| = 2n. \end{aligned} \tag{2.22}$$

The auxiliary graph $G'(k) = (V'(k), E'(k))$ contains $|V'(k)|$ $(= 2|V|)$ nodes and $|E'(k)|$ $(= |V| + 2|E|)$ edges, its construction thus takes $O(|V'(k)| + |E'(k)|) = O(|V| + |E|)$ time. In addition, finding a shortest path in $G'(k)$ takes $O(|V'(k)|^2) = O(|V|^2)$ time. Algorithm 2.1 therefore takes $O(|V|^2 + |V| + |E|) = O(|V|^2)$ time. □

2.5 An Online Algorithm for Multicast Routing

In this section, we deal with the network capacity maximization problem for online multicasting. We first propose an efficient online algorithm for the problem. We then analyze the competitive ratio and time complexity of the proposed algorithm.

2.5.1 Online Algorithm

The basic idea of the proposed algorithm is to respond to each incoming multicast request r_k $(= (s_k, D_k; b_k))$ by either admitting or rejecting it, according to an admission control policy. To model the node and link resource usages by admitting multicast request r_k, an auxiliary edge-weighted, directed graph $G'(k) = (V'(k), E'(k); \omega)$ will be constructed, where the weight of each node-derived or link-derived edge of $G'(k)$ reflects the availability and utilization of the resource in that node or link of G.

In order to reduce the resource consumption of admitting a multicast request r_k and admit more future requests, a multicast tree in $G'(k)$ rooted at the source s'_k and spanning all destination nodes in D_k will be found if it exists. If the weighted sums of all node-derived edges and link-derived edges in the multicast tree are less than their corresponding thresholds, then the request will be admitted; otherwise, it will be rejected. In the following, we detail the construction of $G'(k)$, and then devise an online algorithm for the network capacity maximization problem for online multicasting.

Given an SDN $G = (V, E)$ with node and link capacities $L_v(\cdot)$ and $B_e(\cdot)$, respectively, an auxiliary edge-weighted, directed graph $G'(k) = (V'(k), E'(k); w)$ for each multicast request r_k is constructed, and the construction of $G'(k)$ is identical to the construction of $G'(k)$ for the kth unicast request as shown in Figure 2.1. That is, for each switch node $v \in V$, two nodes v' and v'' are added to $V'(k)$, and there is a directed edge $\langle v', v'' \rangle$ added to $E'(k)$. For each edge $(u, v) \in E$, two directed edges $\langle v'', u' \rangle$ and $\langle u'', v' \rangle$ are added to $E'(k)$, i.e., $V'(k) = \{v', v'' \mid v \in V\}$ and $E'(k) = \{\langle v', v'' \rangle \mid v \in V\} \cup \{\langle v'', u' \rangle, \langle u'', v' \rangle \mid (u, v) \in E\}$.

Now, given a multicast request r_k with $(s_k, D_k; b_k)$, the problem is transformed to finding a multicast tree $T(k)$ in $G'(k)$ rooted at s'_k and spanning all nodes in $D'_k = \{u' \mid u \in D_k\}$ such that the weighted sum of the edges in $T(k)$ is minimized, which is a classic NP-hard *directed Steiner tree problem*. As it is very unlikely to find an exact solution for it in polynomial time, an approximate solution suffices, by applying

the approximation algorithm in [21]. The approximation ratio of the approximation algorithm is $|D_k|^{\epsilon}$, and its running time is a polynomial function of n and $\frac{1}{\epsilon}$, where ϵ is a constant with $0 < \epsilon \leq 1$. The choice of ϵ will determine the accuracy of the solution obtained and the running time of the algorithm. For the sake of simplicity, in the rest of this chapter, we abuse the notation of $T(k)$ and denote it as the corresponding multicast tree in G rooted at s_k and spanning all terminal nodes in D_k too.

To prevent admitting some multicast requests that will degrade the performance of the proposed online algorithm significantly, an admission control policy will be adopted. That is, a multicast request r_k is admitted only if it meets

(i) $\sum_{e \in T(k) \cap E'_v(k)} \omega_e(k) \leq \sigma_v$, and

(ii) $\sum_{e \in T(k) \cap E'_e(k)} \omega_e(k) \leq \sigma_e$,

where $\sigma_v = \sigma_e = |V| - 1 = n - 1$.

The detailed online algorithm for the network capacity maximization problem for online multicasting is given in Algorithm 2.2.

2.5.2 Algorithm Analysis

In the following, we analyze the competitive ratio and time complexity of Algorithm 2.2. Recall that $L_{\min}$ is the minimum TCAM size among switch nodes, i.e., $L_{\min} = \min\{L_v \mid v \in V\}$, $B_{\min}$ is the minimum bandwidth capacity among links, i.e., $B_{\min} = \min\{B_e \mid e \in E\}$, $b_{\max}$ is the maximum bandwidth demand by any multicast request, K is the maximum number of terminal nodes in any multicast request, i.e., $K = \max\{|D_{k'}| \mid 1 \leq k' \leq k\}$, and ϵ is a fixed value with $0 < \epsilon \leq 1$. We start with the following lemma.

Lemma 2.3. *Given an SDN $G = (V, E)$ with node capacity L_v for each switch node $v \in V$ and link bandwidth capacity B_e for each link $e \in E$, denote by $\mathcal{S}(k)$ the set of multicast requests admitted by the online algorithm, Algorithm 2.2, until the arrival of multicast request*

Algorithm 2.2 Online routing algorithm for multicast requests

Input: An SDN $G = (V, E)$ and an incoming multicast request r_k with $(s_k, D_k; b_k)$ and $D_k \subseteq V$;

Output: Maximize the network throughput by admitting or rejecting each arriving multicast request r_k. If admitted, a routing multicast tree for r_k will be found.

```
/* Ensure that the switch table of each node v has at least |D_k| available entries */;
for each node v ∈ V do
    if the available table size at node v is zero then
        Remove v and its incident links from G;
    end if
end for
/* Ensure that the residual bandwidth capacity of each link e is at least b_k */
for each link e ∈ E do
    if the residual bandwidth at link e is less than b_k, i.e., B_e(k) < b_k then
        Remove e from G;
    end if
end for
Construct an edge-weighted, directed graph G'(k) = (V'(k), E'(k); ω) from the
resulting subgraph of G and calculate the edge weights according to the resource
utilization when request r_k arrives;
Find an approximate multicast tree T(k) in G'(k) rooted at s'_k and spanning all
nodes in D'_k, by applying the algorithm due to Charikar et al. [21];
if T(k) does not exist then
    Reject multicast request r_k;
else
    if (Σ_{e∈T(k)∩E'_v(k)} ω_e(k) ≤ σ_v) and (Σ_{e∈T(k)∩E'_e(k)} ω_e(k) ≤ σ_e) then
        Admit multicast request r_k with T(k);
        Update the residual resource capacities of links in E and nodes in V;
    else
        Reject request r_k;
    end if
end if
```

r_k. Then, the cost sums of nodes and of links of G when multicast request r_k arrives are

$$\sum_{v \in V} c_v(k) \leq |\mathcal{S}(k)|(\sigma_v + n - 1) \log \alpha, \tag{2.23}$$

and

$$\sum_{e \in E} c_e(k) \leq \mathbb{B}(k)(\sigma_e + n - 1) \log \beta, \tag{2.24}$$

respectively, where α and β are constants with $2|V| \leq \alpha \leq 2^{L_{\min}}$ and $2|V| \leq \beta \leq 2^{B_{\min}/b_{\max}}$, and $\mathbb{B}(k) = \sum_{k' \in \mathcal{S}(k)} b_{k'}$.

Proof. Consider an admitted multicast request $r_{k'} \in \mathcal{S}(k)$ by the online algorithm. If the edge derived from a switch node $v \in V$ is not in $T(k')$, then $c_v(k'+1) - c_v(k') = 0$. The cost sum of all nodes in G for admitting multicast request $r_{k'}$ is

$$\begin{aligned}
&\sum_{v \in V} (c_v(k'+1) - c_v(k')) \\
=& \sum_{v \in V \cap T(k')} (c_v(k'+1) - c_v(k')) \\
=& \sum_{v \in T(k')} \left(L_v(\alpha^{1-\frac{L_v(k'+1)}{L_v}} - 1) - L_v(\alpha^{1-\frac{L_v(k')}{L_v}} - 1)\right) \\
=& \sum_{v \in T(k')} \left(L_v \alpha^{1-\frac{L_v(k')}{L_v}} \left(\alpha^{\frac{L_v(k') - L_v(k'+1)}{L_v}} - 1\right)\right) \\
\leq& \sum_{v \in T(k')} \left(L_v \alpha^{1-\frac{L_v(k')}{L_v}} \left(\alpha^{\frac{1}{L_v}} - 1\right)\right) && (2.25) \\
=& \sum_{v \in T(k')} \left(L_v \alpha^{1-\frac{L_v(k')}{L_v}} \left(2^{\log \alpha \cdot \frac{1}{L_v}} - 1\right)\right) \\
\leq& \sum_{v \in T(k')} L_v \alpha^{1-\frac{L_v(k')}{L_v}} \cdot \log \alpha \cdot \frac{1}{L_v} && (2.26) \\
\leq& (\sigma_v + (n-1)) \log \alpha, && (2.27)
\end{aligned}$$

where Inequality (2.25) holds, because only one forwarding table entry in the switch table at node v is required to admit multicast request $r_{k'}$ [13, 104]. Inequality (2.26) follows, as $2^x - 1 \leq x$ with $0 \leq x \leq 1$, and $\alpha \leq 2^{L_{\min}} \leq 2^{L_v/d_v}$, and Inequality (2.27) holds from the fact that request $r_{k'}$ is admitted only if the admission control policy (i)

is met, and a multicast tree cannot have more than $n-1$ node-derived edges.

Notice that $c_v(1) = 0$ for all $v \in V$. The cost sum of nodes by routing all requests in $\mathcal{S}(k)$ is

$$\begin{aligned}
&\sum_{v \in V} c_v(k) \\
=&\sum_{k'=1}^{k-1} \sum_{v \in V} (c_v(k'+1) - c_v(k')) \\
=&\sum_{k' \in \mathcal{S}(k)} \sum_{v \in V} (c_v(k'+1) - c_v(k')) \\
\leq&\sum_{k' \in \mathcal{S}(k)} \big(\sigma_v + (n-1)\big) \log \alpha, \qquad \text{by Inequality (2.27)} \\
=&|\mathcal{S}(k)|(\sigma_v + (n-1)) \log \alpha.
\end{aligned}$$

Thus, Inequality (2.23) holds.

Inequality (2.24) can be similarly proven by adopting the technique for Inequality (2.12), omitted. □

We then show the lower bound on the cost sum of node-derived and link-derived edges of the multicast tree $T(k')$ for a multicast request $r_{k'}$, which is admitted by an optimal offline algorithm but rejected by the online algorithm, as follows.

Lemma 2.4. *Let $\mathcal{R}(k)$ be the set of multicast requests admitted by an optimal offline algorithm yet rejected by the online algorithm, Algorithm 2.2, prior to the arrival of multicast request r_k. Let $T_{opt}(k')$ be the multicast tree in $G'(k)$ found by the optimal offline algorithm for request $r_{k'} \in \mathcal{R}(k)$. Then, for each multicast request $r_{k'} \in \mathcal{R}(k)$, we have $\sum_{e \in T_{opt}(k')} \omega_e(k') \geq \frac{\min\{\sigma_v, \sigma_e\}}{K^\epsilon}$, where α and β are constants with $2|V| = 2n \leq \alpha \leq 2^{L_{\min}}$ and $2|V| = 2n \leq \beta \leq 2^{B_{\min}/b_{\max}}$.*

Proof. Suppose that given an SDN G and there are an optimal offline algorithm and the proposed online algorithm for the network capacity maximization problem for online multicasting. Assuming that each request is revealed to both algorithms one by one. Each algorithm can either admit the request by allocating the demanded

resources to it or reject it. However, the optimal offline algorithm may reject a request that is admitted by the proposed online algorithm, or vice versa. Thus, for a given monitoring period, the proposed online algorithm and the optimal offline algorithm may accept different sets of requests, resulting in a difference between the resource availability in the resulting networks by these two algorithms when a request arrives. Denote by $G_{opt}(k')$ and $G(k')$ the networks to which the optimal offline algorithm and the proposed online algorithm are applied prior to the arrival of request $r_{k'}$, respectively.

Consider a multicast request $r_{k'}$ that is admitted by the optimal offline algorithm yet rejected by the proposed online algorithm. Since $r_{k'}$ is admitted by the optimal offline algorithm, it means that the optimal offline algorithm is able to admit $r_{k'}$ into $G_{opt}(k')$ using a tree $T_{opt}(k')$ in $G_{opt}(k')$. We check whether $G'(k')$ derived from $G(k')$ contains $T_{opt}(k')$, and there are exactly two cases. Case 1: every node and edge in $T_{opt}(k')$ has a corresponding edge in $G'(k')$, or Case 2: at least one node or one link in $T_{opt}(k')$ does not have a corresponding edge in $G'(k')$, because its available resource capacity cannot meet the demand of $r_{k'}$. In the following, we will show that for both cases,

$$\sum_{e \in T_{opt}(k')} \omega_e(k') \geq \frac{\min\{\sigma_v, \sigma_e\}}{K^\epsilon},$$

where $\omega_e(k')$ is calculated based on the availability and workload of the resource in $G(k')$.

Case 1. If every node and edge in $T_{opt}(k')$ has a corresponding edge in $G'(k')$, there is at least one tree in $G'(k')$ that roots at $s'_{k'}$ and spans the nodes in $D_{k'}$, namely the one corresponds to $T_{opt}(k')$. Consequently, the proposed online algorithm must be able to find a tree $T(k')$ in $G'(k')$ such that $T(k')$ can meet the resource demand of request $r_{k'}$ and

$$\sum_{e \in T} \omega_e(k') \geq \frac{\sum_{e \in T(k')} \omega_e(k')}{|D_{k'}|^\epsilon} \geq \frac{\sum_{e \in T(k')} \omega_e(k')}{K^\epsilon}$$

for any tree T in $G'(k')$ that roots at $s'_{k'}$ and spans the nodes in $D_{k'}$, including $T_{opt}(k')$.

Since $r_{k'}$ is rejected by the online algorithm, the weighted sum of edges in $T(k')$ is no less than the given threshold, i.e.,

$$\frac{\sum_{e \in T(k')} \omega_e(k')}{K^\epsilon} \geq \frac{\min\{\sigma_v, \sigma_e\}}{K^\epsilon}.$$

Thus,

$$\sum_{e \in T_{opt}(k')} \omega_e(k') \geq \frac{\sum_{e \in T(k')} \omega_e(k')}{K^\epsilon} \geq \frac{\min\{\sigma_v, \sigma_e\}}{K^\epsilon}.$$

Case 2. At least one node or one edge in $T_{opt}(k')$ does not have a corresponding edge in $G'(k')$. Recall that a node or a link is not included only if its residual capacity cannot satisfy the resource demands of $r_{k'}$. This case thus can be further divided into two subcases: (a) if there is not any multicast tree with sufficient available table entries to route the message of the request, then there must have a node-derived edge $e' = \langle v'', w_{vu} \rangle \in T_{opt}(k') \cap E'_v(k')$ derived from node $v \in V$ such that $L_v(k') < d_v \leq K$. Consequently, the weighted sum of edges in $T_{opt}(k')$ is greater than σ_v / K^ϵ as follows.

$$\begin{aligned}
&\sum_{e \in T_{opt}(k')} \omega_e(k') \\
\geq &\omega_{\langle v', v'' \rangle}(k') \\
= &\alpha^{1 - \frac{L_v(k')}{L_v}} - 1 \\
> &\alpha^{1 - \frac{d_v}{L_v}} - 1, \qquad \text{since } L_v(k') < d_v \\
\geq &\alpha^{1 - \frac{1}{\log \alpha}} - 1, \qquad \text{since } 2n \leq \alpha \leq 2^{L_{\min}} \leq 2^{L_v / d_v} \\
= &\frac{\alpha}{2} - 1 \\
\geq &\sigma_v \\
\geq &\frac{\sigma_v}{K^\epsilon}.
\end{aligned}$$

(b) If there is not any multicast tree with sufficient bandwidth on one of its links to admit multicast request k', then the weighted sum of edges in $T_{opt}(k')$ is greater than σ_e / K^ϵ, which can be proved using the similar method as the one for online unicasting

in the proof body of Lemma 2.2, and thus omitted. □

We finally have the following theorem.

Theorem 2.2. *Given an SDN $G = (V, E)$ with both node and link capacities $L_v(\cdot)$ and $B_e(\cdot)$ for all $v \in V$ and $e \in E$, assume that there is a sequence of multicast requests $r_k = (s_k, D_k; b_k)$ arriving one by one without the knowledge of future arrivals with $D_k \subseteq V$. There is an online algorithm, Algorithm 2.2, with the competitive ratio of $2K^{\epsilon}(\gamma \log \alpha + \log \beta) + 1$ for the network capacity maximization problem for online multicasting, which takes $O((|V| + |E|)^{1/\epsilon} K^{2/\epsilon})$ time, where α and β are constants with $2|V| = 2n \leq \alpha \leq 2^{L_{\min}}$ and $2|V| = 2n \leq \beta \leq 2^{B_{\min}/b_{\max}}$, $\gamma = B_{\min}/\log \beta$, and ϵ is a constant with $0 < \epsilon \leq 1$.*

Proof. Let $\mathbb{B}_{opt}(k)$ be the total bandwidth of all admitted multicast requests by an optimal offline algorithm. Combining Lemmas 2.3 and 2.4, we have

$$
\begin{aligned}
&\frac{(n-1)}{K^{\epsilon}}(\mathbb{B}_{opt}(k) - \mathbb{B}(k)) \\
\leq &\frac{(n-1)}{K^{\epsilon}} \sum_{k' \in \mathcal{R}(k)} b_{k'} \\
= &\sum_{k' \in \mathcal{R}(k)} b_{k'} \frac{(n-1)}{K^{\epsilon}} \\
\leq &\sum_{k' \in \mathcal{R}(k)} b_{k'} \big(\sum_{e \in T_{opt}(k')} \omega_e(k') \big), \qquad \text{by Lemma 2.4} \\
= &\sum_{k' \in \mathcal{R}(k)} b_{k'} \big(\sum_{\langle v'', w_{vu} \rangle \in T_{opt}(k') \cap E'_v(k')} \omega_{\langle v'', w_{vu} \rangle}(k') \\
&+ \sum_{\langle w_{vu}, u' \rangle \in T_{opt}(k') \cap E'_e(k')} \omega_{\langle w_{vu}, u' \rangle}(k') \big) \\
\leq &\sum_{k' \in \mathcal{R}(k)} b_{k'} \big(\sum_{\langle v'', w_{vu} \rangle \in T_{opt}(k') \cap E'_v(k')} \omega_{\langle v'', w_{vu} \rangle}(k) \\
&+ \sum_{\langle w_{vu}, u' \rangle \in T_{opt}(k') \cap E'_e(k')} \omega_{\langle w_{vu}, u' \rangle}(k) \big) \qquad (2.28) \\
= &\sum_{k' \in \mathcal{R}(k)} b_{k'} \big(\sum_{\langle v'', w_{vu} \rangle \in T_{opt}(k') \cap E'_v(k')} \frac{c_v(k)}{L_v} \\
&+ \sum_{\langle w_{vu}, u' \rangle \in T_{opt}(k') \cap E'_e(k')} \frac{c_{(v,u)}(k)}{B_{(v,u)}} \big)
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{k' \in \mathcal{R}(k)} \sum_{\langle v'', w_{vu} \rangle \in T_{opt}(k') \cap E'_v(k')} b_{k'} \frac{c_v(k)}{L_v} \\
&\quad + \sum_{k' \in \mathcal{R}(k)} \sum_{\langle w_{vu}, u' \rangle \in T_{opt}(k') \cap E'_e(k')} b_{k'} \frac{c_{(v,u)}(k)}{B_{(v,u)}} \\
&\leq \sum_{v \in V} c_v(k) \sum_{k' \in \mathcal{R}(k)} \sum_{\langle v'', w_{vu} \rangle \in T_{opt}(k') \cap E'_v(k')} \frac{b_{k'}}{L_v} \\
&\quad + \sum_{(v,u) \in E} c_{(v,u)}(k) \sum_{k' \in \mathcal{R}(k)} \sum_{\langle w_{vu}, u' \rangle \in T_{opt}(k') \cap E'_e(k')} \frac{b_{k'}}{B_{(v,u)}} \\
&= \sum_{v \in V} c_v(k) \frac{\sum_{k' \in \mathcal{R}(k)} \sum_{\langle v'', w_{vu} \rangle \in T_{opt}(k') \cap E'_v(k')} b_{k'}}{L_v} \\
&\quad + \sum_{(v,u) \in E} c_{(v,u)}(k) \frac{\sum_{k' \in \mathcal{R}(k)} \sum_{\langle w_{vu}, u' \rangle \in T_{opt}(k') \cap E'_e(k')} b_{k'}}{B_{(v,u)}} \\
&\leq \sum_{v \in V} c_v(k) \gamma + \sum_{e \in E} c_e(k) \\
&\leq \gamma \sum_{v \in V} c_v(k) + \sum_{e \in E} c_e(k) \\
&\leq \gamma |S(k)| (\sigma_v + (n-1)) \log \alpha \\
&\quad + \mathbb{B}(k) (\sigma_e + (n-1)) \log \beta \\
&= 2(n-1)(\gamma |S(k)| \log \alpha + \mathbb{B}(k) \log \beta)
\end{aligned} \tag{2.29}
$$

where Inequality (2.28) follows because the resource utilization is always nondecreasing.

From Inequality (2.29), we have

$$
\frac{\mathbb{B}_{opt}(k) - \mathbb{B}(k)}{\mathbb{B}(k)} \leq 2K^{\epsilon} \left(\frac{|\mathcal{S}(k)|}{\mathbb{B}(k)} \gamma \log \alpha + \log \beta \right). \tag{2.30}
$$

Hence,

$$
\begin{aligned}
&\frac{\mathbb{B}_{opt}(k)}{\mathbb{B}(k)} \\
&\leq 2K^{\epsilon} \left(\frac{|\mathcal{S}(k)|}{\mathbb{B}(k)} \gamma \log \alpha + \log \beta \right) + 1, \\
&\leq 2K^{\epsilon} (\gamma \log \alpha + \log \beta) + 1, \qquad \text{since } \frac{|\mathcal{S}(k)|}{\mathbb{B}(k)} \leq 1,
\end{aligned}
$$

$$\leq 2K^{\epsilon}(\gamma \log(2n)) + 1, \qquad \text{if } \alpha = \beta = 2|V| = 2n,$$
$$= O(K^{\epsilon} \log n).$$

The rest is to analyze the time complexity of Algorithm 2.2. Recall that given an SDN $G = (V, E)$, we first construct an auxiliary graph $G'(k) = (V'(k), E'(k))$, which contains $|V'(k)|$ $(= |V| + |E|)$ nodes and $|E'(k)|$ $(= |V| + 4|E|)$ edges. Thus, the construction of $G'(k)$ takes $O(|V| + |E|)$ time. It then takes $O(|V'(k)|^{1/\epsilon}|D_k|^{2/\epsilon})$ time to find an approximate Steiner tree in $G'(k)$ for a multicast request r_k, by employing the algorithm in [21]. As a result, Algorithm 2.2 takes $O((|V| + |E|)^{1/\epsilon}K^{2/\epsilon})$ time. □

2.6 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms through experimental simulations. We also investigate the impact of important parameters on the performance of the proposed algorithms.

2.6.1 Experimental Environment Settings

We consider networks with 50, 100, 150, 200, and 250 nodes, respectively. For each network size, 30 network instances are generated, using the tool GT-ITM [15]. The size of TCAM L_v of each switch node $v \in V$ varies from 500 to 5,000, and the bandwidth capacity B_e of each link $e \in E$ varies from 1,000 Mbps to 10,000 Mbps [79, 85, 107]. The bandwidth demand b_k of each unicast or multicast request r_k is randomly drawn between 1 Mbps and 50 Mbps. The number of destinations in a multicast request is randomly chosen between 1% and 15% of the network size. The value in each figure is the mean of the results out of 30 network instances with 30 different sequences of 50,000 unicast requests or 20,000 multicast requests.

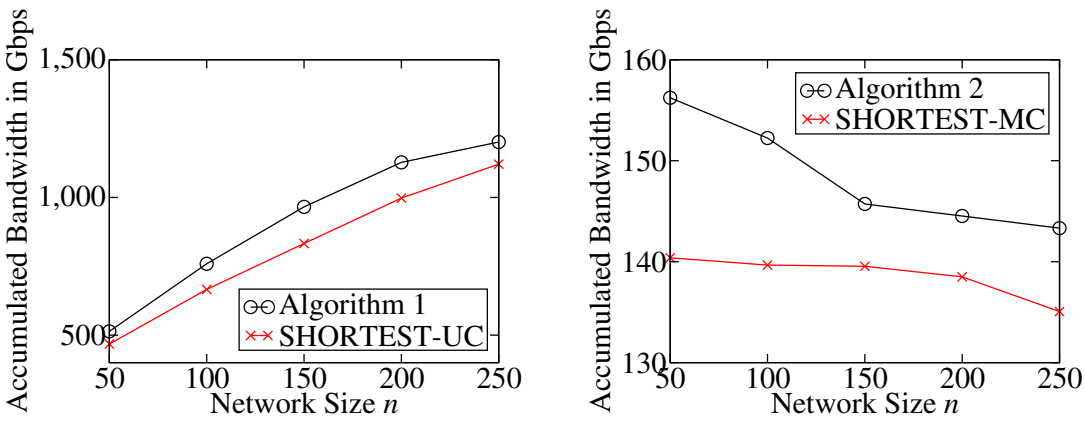
To speed up the running time of the proposed algorithm, Algorithm 2.2, we employ a simpler approach based on the single-source shortest path algorithm to find an approximate, minimum Steiner tree, instead of the time-consuming approximation

algorithm in [21] that delivers a better solution.

Since the proposed algorithms in this chapter are the very first ones that jointly consider node and edge capacities in SDNs, comparing the performance of the proposed algorithms with those only considered either one type of resource capacity may be unfair. To evaluate the performance of the proposed algorithms against the benchmarks, we here propose two heuristics `SHORTEST-UC` and `SHORTEST-MC` for online unicasting and multicasting respectively. Specifically, for each request r_k, the heuristic algorithms first remove the links and nodes from the network G that do not have enough residual resources to support the admission of the request, and then assign each link the same weight. `SHORTEST-UC` finds a shortest path with the minimum number of links from the source to the destination of request r_k, while `SHORTEST-MC` finds a single-source shortest path tree spanning all destination nodes rooted at the source and spanning all destinations of a multicast request r_k.

2.6.2 Performance Evaluation of Different Algorithms

We first evaluate the proposed two online algorithms Algorithm 2.1 and Algorithm 2.2 against algorithms `SHORTEST-UC` and `SHORTEST-MC`, by varying network size n from 50 to 250 while keeping other parameters fixed, i.e., $\alpha = \beta = 2n$ and $\sigma_e = \sigma_v = n - 1$.



(a) Accumulated bandwidth for online unicasting

(b) Accumulated bandwidth for online multicasting

Figure 2.2: The accumulated bandwidth delivered by different algorithms in networks.

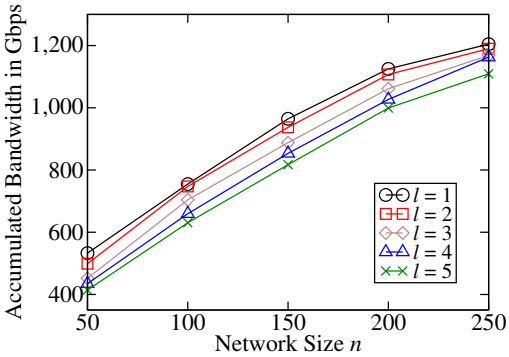
Figure 2.2 plots the performance curves of different algorithms, from which it can be seen both Algorithm 2.1 and Algorithm 2.2 outperform `SHORTEST-UC` and `SHORTEST-MC`. Specifically, Algorithm 2.1 outperforms algorithm `SHORTEST-UC`. As shown in Figure 2.2 (a), Algorithm 2.1 delivers 10% more accumulated bandwidth than that of `SHORTEST-UC`. Meanwhile, Algorithm 2.2 delivers more accumulated bandwidth and admits more requests than that of `SHORTEST-MC`, too. In particular, As shown in Figure 2.2 (b), the accumulated bandwidth delivered by Algorithm 2.2 is 10% more than that by `SHORTEST-MC` when $n = 50$. However, Algorithm 2.2 still delivers 9% more accumulated bandwidth compared with `SHORTEST-MC` when the network size is 250. The reason is that with the growth of the network size n, the number of destinations in each multicast request increases, requiring more node and link resources for the request realization.

2.6.3 Parameter Impacts on Algorithmic Performance

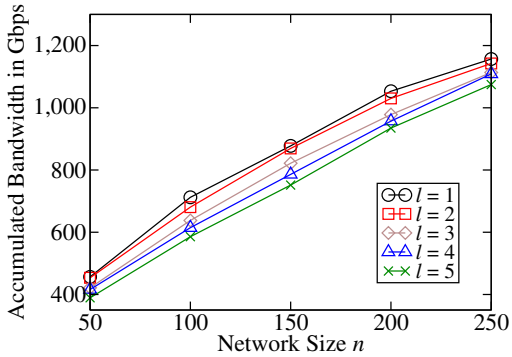
We first investigate the impact of parameters α and β on the performance of the proposed algorithms, by varying them from $2^1 n$ to $2^5 n$ while keeping $\sigma_v = \sigma_e = n - 1$. Figure 2.3 and 2.4 plot the performance curves of Algorithm 2.1 and Algorithm 2.2, by varying either α or β while fixing the other.

It can be seen from Figures 2.3 (a) to 2.3 (c) that when α is fixed, the larger the value of β, the less the accumulated bandwidth delivered by Algorithm 2.1 and vice versa. For instance, when $\alpha = 2^1 n$ and $n = 50$, Algorithm 2.1 with $\beta = 2^1 n$ delivers 15% more the accumulated bandwidth than itself with $\beta = 2^5 n$. It also can be seen that the performance gap of Algorithm 2.1 under different α and β is flat with the increase of network size n. Similarly, Figure 2.4 plots the performance curves of Algorithm 2.2, by varying the values of exactly one of α and β each time, from which it can be seen when α is fixed, the larger the value of β, the less the accumulated bandwidth delivered by Algorithm 2.2 and vice versa.

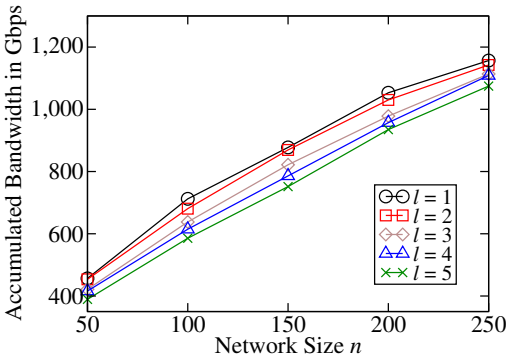
We then study the impact of the admission thresholds σ_v and σ_e on the perfor-



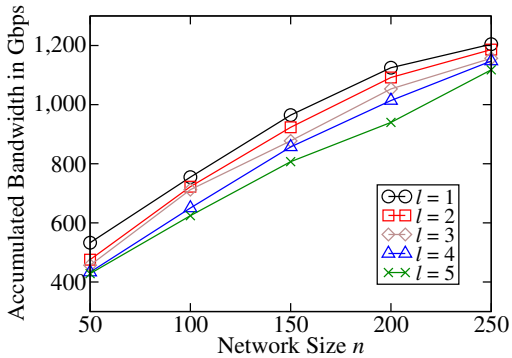
(a) The performance of Algorithm 2.1 by varying β $(= 2^l n)$ when $\alpha = 2^1 n$



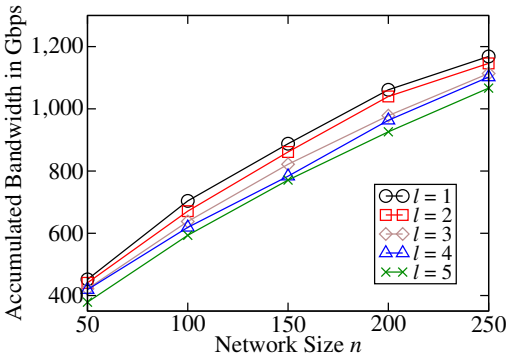
(b) The performance of Algorithm 2.1 by varying β $(= 2^l n)$ when $\alpha = 2^3 n$



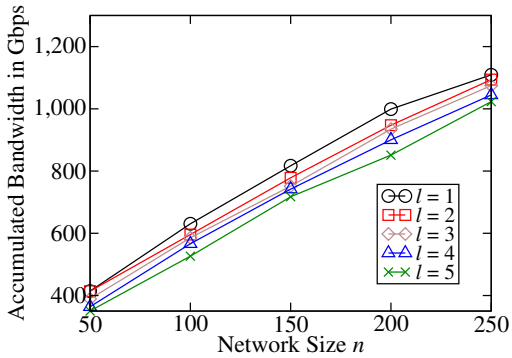
(c) The performance of Algorithm 2.1 by varying β $(= 2^l n)$ when $\alpha = 2^5 n$



(d) The performance of Algorithm 2.1 by varying α $(= 2^l n)$ when $\beta = 2^1 n$

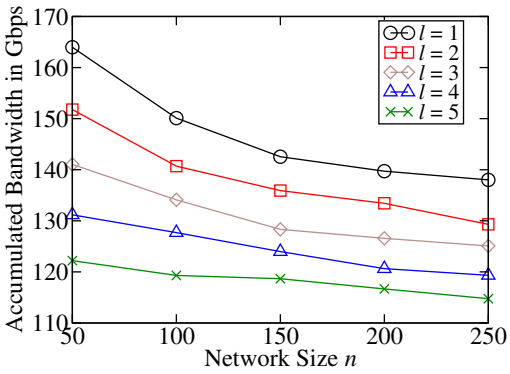


(e) The performance of Algorithm 2.1 by varying α $(= 2^l n)$ when $\beta = 2^3 n$

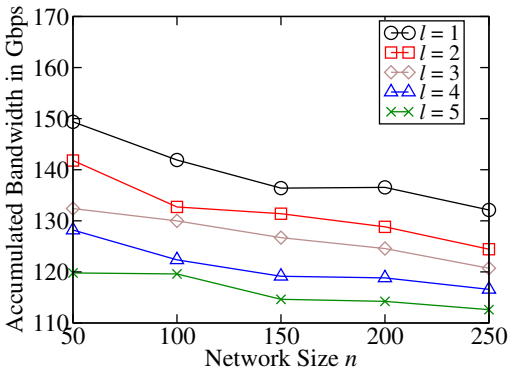


(f) The performance of Algorithm 2.1 by varying α $(= 2^l n)$ when $\beta = 2^5 n$

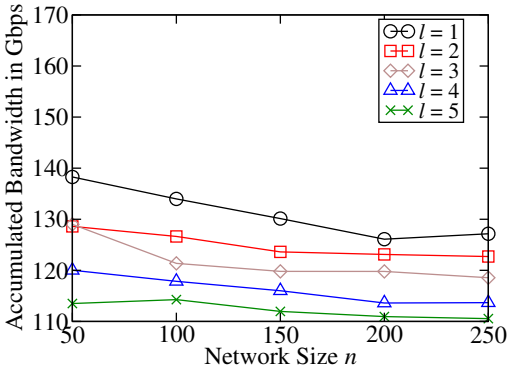
Figure 2.3: The performance of Algorithm 2.1 for online unicasting by varying α and β, when $\sigma_v = \sigma_e = n - 1$.



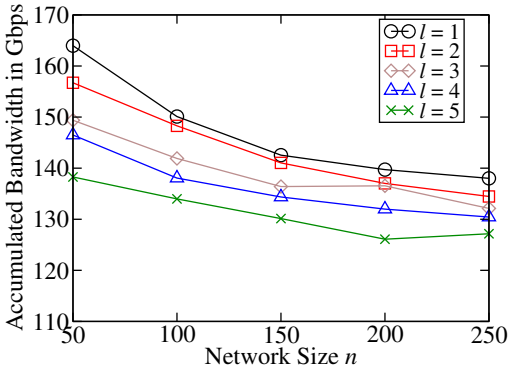
(a) The performance of Algorithm 2.2 by varying β $(= 2^l n)$ when $\alpha = 2^1 n$



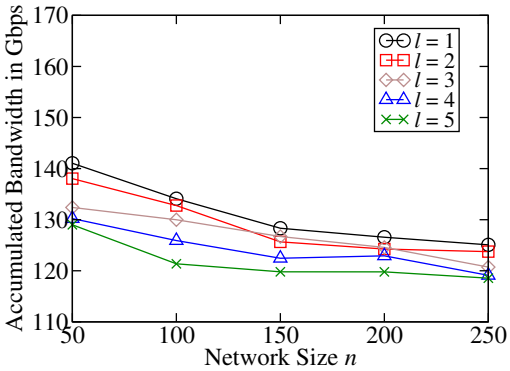
(b) The performance of Algorithm 2.2 by varying β $(= 2^l n)$ when $\alpha = 2^3 n$



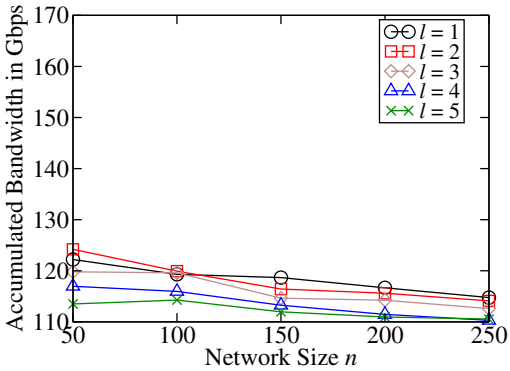
(c) The performance of Algorithm 2.2 by varying β $(= 2^l n)$ when $\alpha = 2^5 n$



(d) The performance of Algorithm 2.2 by varying α $(= 2^l n)$ when $\beta = 2^1 n$



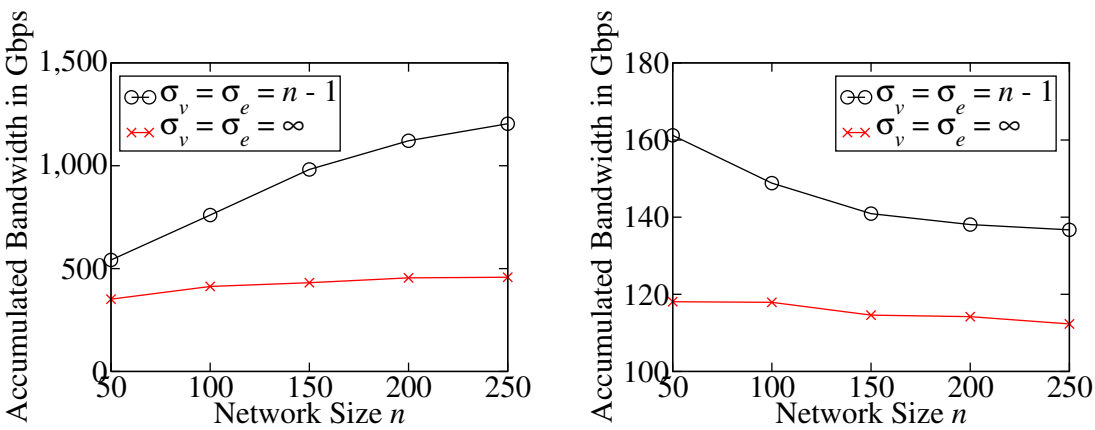
(e) The performance of Algorithm 2.2 by varying α $(= 2^l n)$ when $\beta = 2^3 n$



(f) The performance of Algorithm 2.2 by varying α $(= 2^l n)$ when $\beta = 2^5 n$

Figure 2.4: The performance of Algorithm 2.2 for online multicasting by varying α and β when $\sigma_v = \sigma_e = n - 1$.

mance of algorithms 2.1 and 2.2. Figure 2.5 plots the performance curves of Algorithm 2.1 and Algorithm 2.2 with and without the admission control thresholds, from which it can be seen that both of them with admission control significantly outperform themselves without the admission control. Specifically, for online unicasting, the performance gap of Algorithm 2.1 with and without the thresholds becomes larger and larger with the increase in network size n, as shown in Figure 2.5 (a). For example, the ratio of the accumulated bandwidths delivered by Algorithm 2.1 with and without the admission control grows from 1.25 to 2.5 when $n = 25, 250$ respectively. For the online multicasting, the performance gap of Algorithm 2.2 with and without the admission control is relatively stable, as shown in Figure 2.5 (b). The difference in the accumulated bandwidth only drops from approximately 30 Gbps to approximately 24 Gbps for a monitoring period when the network size increases from 100 to 250.

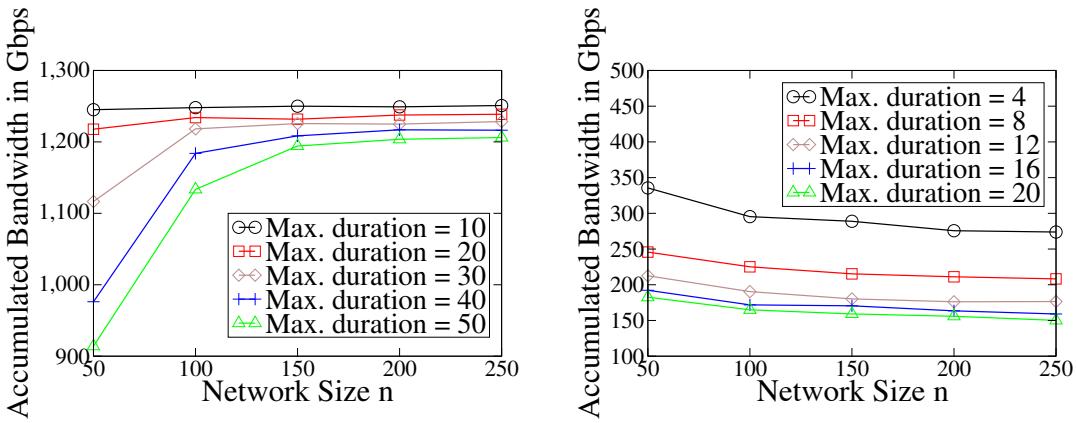


(a) Accumulated bandwidth by Algorithm 2.1 (b) Accumulated bandwidth by Algorithm 2.2

Figure 2.5: The accumulated bandwidth delivered by Algorithm 2.1 for online unicasting and Algorithm 2.2 for online multicasting with thresholds $\sigma_v = \sigma_e = n - 1$ and without the thresholds $\sigma_v = \sigma_e = \infty$ when $\alpha = \beta = 2n$.

2.6.4 Impact of Request Implementation Durations on Algorithm Performance

We now investigate the impact of the durations (numbers of time slots) of admitted requests on the performance of the proposed online algorithms Algorithm 2.1 and



(a) Accumulated bandwidth by Algorithm 2.1 (b) Accumulated bandwidth by Algorithm 2.2

Figure 2.6: The network throughput delivered by Algorithm 2.1 for online unicasting and Algorithm 2.2 for online multicasting by varying the maximum duration of admitted requests in terms of numbers of time slots.

Algorithm 2.2, by varying the maximum duration of admitted requests, where each admitted request will release the resources allocated to it when it departs from the network. From Figure 2.6 (a)–(b), we can see that the longer the maximum duration is, the less the number of requests the proposed online algorithms admit. The reason behind is that longer resource holding durations by the admitted requests result in fewer resources for later request admissions, which limits the ability of the network to admit more requests. Specifically, Figure 2.6 (a) indicates that the impact of the maximum duration of admitted unicast requests is significant on the performance of the algorithm when the network size is small, because a small network has fewer resources, it cannot accommodate more requests if the currently admitted requests do not depart from it. On the other hand, from Figure 2.6 (b), we can see that the gap between the performance of the online multicast algorithm is roughly the same for all network sizes when the maximum duration of admitted requests changes, due to the fact that multicast requests require much more resources and readily consume all the resources in the network even if some admitted requests depart from the network shortly.

2.7 Summary

In this chapter, we studied online unicasting and multicasting in SDNs with resource capacity constraints on switch nodes and links, and user request bandwidth demands. We first proposed a novel cost model to model the usage costs of different resources. We then devised novel online algorithms with provable competitive ratios for online unicasting and multicasting. We finally evaluated the performance of the proposed algorithms through experimental simulation. Simulation results indicate that the proposed algorithms are very promising.

Chapter 3

Throughput Maximization in Software-Defined Networks with Consolidated Middleboxes

3.1 Introduction

Computer networks nowadays rely on various middleboxes, including firewall, Intrusion Detection Systems (IDSs), WAN optimizers, and Deep Packet Inspections (DPIs), to enhance the performance and security of different network services [44, 54, 119]. Unfortunately, the management and deployment of these hardware middleboxes are complex and costly [119]. For example, statistics indicated that large networks (10k-100k nodes) spent over a million dollars on deploying and maintaining hardware middleboxes while medium and small networks (1k-10k nodes) spent between $5,000 and $50,000 in the last five years [119]. With the advancement of the Network Function Virtualization (NFV), middleboxes can be implemented in Virtual Machines (VMs) that run in Physical Machines (PMs) [43, 110, 119]. The NFVs can be relocated and instantiated at servers located at different locations in a network without needs of purchasing and installing expensive middleboxes. By decoupling network functions from the hardware platform on which network functions are executed, NFV has the great potential to lead to significant reductions in operating expenses (OPEX) and capital expenses (CAPEX) of network service providers and facilitate the deployment

of new services with increased agility and faster time-to-value [103]. We refer to the software implementation of middleboxes as the *consolidated middleboxes*. Along with the technique of Software-Defined Networking (SDN), consolidated middleboxes offer a promising alternative way to provide cheap and simplified management of middleboxes [49, 118].

In this chapter, we deal with realizing user requests with each specifying a sequence of middleboxes in SDNs with the aim to maximize the network throughput. This problem poses great challenges. One challenge is that different types of resources in SDNs have different capacities. For instance, the forwarding table of an SDN-enabled switch usually is made by Ternary Content-Addressable Memory (TCAM) to facilitate fast, parallel lookups of forwarding rules. TCAM however is expensive and energy-hungry, its capacity thus is restricted to a few thousand table entries [85]. Meanwhile, the computing resource of the PM attached to an SDN-enabled switch is limited too. Another challenge is that all resources in an SDN are dynamically allocated, causing significant fluctuations in their consumptions and availabilities. The time-varying nature of resource demands and consumptions complicates the cost modeling of resource usages. In addition, each user request requires its traffic to traverse a specified sequence of middleboxes that is referred to the *service chain* of the request. In this chapter, we will address the aforementioned challenges.

In spite of several studies of consolidated middleboxes [22, 49, 110], none of the studies has taken the forwarding table size into consideration. Almost all existing solutions adopt a strategy that decomposes the routing path finding and the service chain execution into two separate subtasks [49], the solutions thus are suboptimal. To the best of our knowledge, we are the first to formulate a novel routing optimization problem with consolidated middleboxes in SDNs by jointly taking into account both routing path finding and consolidated middlebox placement while meeting different user QoSs, by providing efficient heuristic solutions.

The main contributions of this chapter are as follows. We consider the network

throughput maximization problem of realizing user requests with service chains in SDNs, subject to various network resource capacity constraints. We first formulate an Integer Linear Program (ILP) solution to the problem when the problem size is small. We then devise a heuristic by providing a novel cost model to model resource consumptions. We also propose a faster heuristic to quickly respond to user requests, by exploring non-trivial tradeoffs between the accuracy (quality) of a solution and the running time of obtaining the solution. Furthermore, we consider dynamic admissions of user requests where user requests arrive one by one without the knowledge of future arrivals, by showing how to extend the proposed algorithms for dynamic admissions of requests. We finally evaluate the performance of the proposed algorithms through simulations, based on real and synthetic network topologies. Experimental results demonstrate that the proposed algorithms are very promising, compared to a baseline algorithm and the ILP, which delivers optimal solutions.

The rest of the chapter is organized as follows. Section 3.2 will introduce the system model and notations, and define the problem. Section 3.3 will formulate an ILP solution to the problem. Section 3.4 and Section 3.5 will present two heuristic algorithms that strive for non-trivial tradeoffs between the accuracy of a solution and the running time of obtaining the solution. Section 3.6 will devise an online algorithm for dynamic request admissions. Section 3.7 will evaluate the performance of the proposed algorithms through simulations, and Section 3.8 will summarize the chapter.

3.2 Preliminaries

3.2.1 System Model

We consider a software-defined network represented by a directed graph $G = (V, E)$, where V is the node set and E is the edge set. Each node $v \in V$ represents an SDN-enabled switch, while each directed edge $\langle u, v \rangle \in E$ represents a link from switch

u to switch v. Each switch $v \in V$ is equipped with a Ternary Content-Addressable Memory (TCAM) forwarding table that can accommodate at most L_v forwarding rules. A subset of switches in V connected to physical machines (PMs) to implement middleboxes as virtual machines. As such a switch and its attached PM usually are connected by a high-speed optical link, the latency between them is negligible. In the rest of this chapter, the switches and their attached PMs will be used interchangeably. Denote by V_{pm} ($\subseteq V$) the set of switches that have attached PMs. Without loss of generality, we assume that each PM attached to a switch $v \in V_{pm}$ has limited computing resource capacity, denoted by C_v. If switch $v \in V \setminus V_{pm}$, then $C_v = 0$. Similarly, each link $e \in E$ has a bandwidth capacity B_e. We assume that there is a logically centralized SDN controller for network G that collects and processes user requests, by installing forwarding rules into the forwarding tables in switches, assigning the middleboxes for the requests to PMs, and allocating bandwidth on links.

3.2.2 User Requests

We assume that time is slotted into *equal time slots*. User requests are scheduled by the centralized SDN controller in the beginning of each time slot. Let $S(t)$ be the set of arrived user requests in time slot t. Each *user request* has a certain amount of bandwidth demand to route its traffic in G from a source switch to a destination switch that passes through a sequence of middleboxes, and the request also has the end-to-end delay requirement. Let $r_i \in S(t)$ be a user request, represented by a quintuple $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle$, where $s_i, t_i \in V$ are, respectively, its source and destination switches, b_i is its bandwidth demand, SC_i is its service chain, and $d_i \in \mathbb{R}^+$ is its end-to-end delay constraint. Admission of request r_i therefore involves routing the traffic from the source switch s_i to the destination switch t_i via a routing path $P_i = \langle s_i, \ldots, t_i \rangle$ subject to the specified constraints.

Following the same assumption as in [49, 100, 110, 118], we assume that services in SC_i are run in a single VM and different VMs serving different requests can be

consolidated to a single Physical machine (PM). Specifically, when the traffic of request r_i arrives at the PM hosting the VM for its service chain SC_i, the traffic will be directed to the VM and the services in SC_i are applied in the specified order. Performing the services in SC_i for r_i thus will consume the computing resource of a PM. Denote by $C(i,j)$ the amount of computing resource needed by SC_i in a PM attached to switch $v_j \in V_{pm}$. Notice that some services in SC_i may alter the volume of the traffic of request r_i. For instance, the volume of traffic increases if encryption is applied to the traffic, while the volume of traffic decreases if compression is applied to the traffic. We here define $\lambda_i \in \mathbb{R}^+$ as the ratio between the volumes of the traffic of request r_i before and after processing at a PM. Since request r_i requires an amount b_i of bandwidth to route its traffic before processing, it needs an amount $\lambda_i \cdot b_i$ of bandwidth to route the processed traffic. The value of λ_i for each request r_i is given and can be derived from historical traces of similar requests [23]. In addition, each request r_i has a tolerant end-to-end delay requirement d_i. Suppose that request r_i is admitted with a routing path P_i from its source s_i to its destination t_i, and its service chain SC_i is implemented on a PM-attached switch $v \in V_{pm}$ on P_i. Let $d(P_i)$ and $d(i,v)$ be the network delay experienced by r_i via path P_i and the processing delay of r_i at PM v, respectively. The network delay $d(P_i)$ is proportional to the number of switches on P_i, and the average processing delay $d(i,v)$ depends on the complexity of the service chain SC_i which usually is given as *a priori*. Then, the end-to-end delay D_i of r_i via path P_i is the sum of the network delay of P_i and the processing delay of SC_i, i.e., $d(P_i) + d(i,v)$. It has to be guaranteed that $d(P_i) + d(i,v) \leq d_i$ for every admitted request r_i.

3.2.3 Problem Definition

Given an SDN $G = (V, E)$, a subset of switches V_{pm} ($\subseteq V$) with each attaching a PM with computing capacity C_v, the forwarding table capacity L_v for each switch $v \in V$, the bandwidth capacity B_e for each link $e \in E$, and a set of user requests $S(t)$ at time slot t, the *network throughput maximization problem* in G is to admit as many user

requests in $S(t)$ as possible such that the number of requests admitted is maximized while the end-to-end delay d_i, bandwidth demand b_i, and computing demand $C(i,j)$ of the service chain SC_i of each admitted request $r_i \in S(t)$ is met, subject to resource capacity constraints in G.

Given an SDN $G = (V, E)$, a subset of switches V_{pm} ($\subseteq V$) with each attaching a PM with computing capacity C_v, the forwarding table capacity L_v for each switch $v \in V$, the bandwidth capacity B_e for each link $e \in E$, and a given time horizon T that consists of T equal time slots, assume that the set of requests arrived at time slot t is $S(t)$ and the duration of each request $r_i = \langle s_i, t_i, b_i, SC_i, d_i, \tau_i \rangle \in S(t)$ in the system is τ_i time slots with $1 \leq \tau_i \leq \tau_{\max}$, the *online network throughput maximization problem* in G is to admit as many user requests as possible during time horizon T such that the number of requests admitted is maximized while the end-to-end delay d_i, bandwidth demand b_i, and computing demand $C(i,j)$ of the service chain SC_i of each admitted request $r_i \in S(t)$ is met, subject to resource capacity constraints in G.

3.2.4 NP-Hardness

We show that the network throughput maximization problem is NP-hard by the following lemma.

Lemma 3.1. *The network throughput maximization problem in a software-defined network $G = (V, E)$ is NP-hard.*

Proof. We show that the network throughput maximization problem in $G = (V, E)$ is NP-hard, by a polynomial reduction from the *generalized assignment problem* (GAP) which is a well-known NP-hard problem [25]. Given an instance of the GAP in the form of a set of bins $\mathcal{B} = \{b_1, \ldots, b_n\}$, a set of items $\mathcal{I} = \{i_1, \ldots, i_m\}$, bin capacities $cap : \mathcal{B} \mapsto \mathbb{R}^+$ and $size : \mathcal{B} \times \mathcal{I} \mapsto \mathbb{R}^+$. For each item i_j with $1 \leq j \leq m$ and bin b_k with $1 \leq k \leq n$, we are given a size $size(j,k)$ and a profit $profit(j,k)$. The problem is to pack a subset $U \subseteq \mathcal{I}$ of items to the bins in $\mathcal{B}$ such that the total profit by these items is maximized. The GAP problem is a well-studied problem.

We first construct an SDN $G = (V, E)$, through adding a stand-alone switch i for each item i in $\mathcal{I}$, a PM-attached switch b for each bin b in $\mathcal{B}$, a virtual sink v_0 that is serving as the common destination for all requests, a link from each stand-alone switch to each PM-attached switch, and a link from each PM-attached switch to the virtual sink v_0. That is, $V = \mathcal{I} \cup \mathcal{B} \cup \{v_0\}$ and $E = \{\langle i, b\rangle \mid i \in \mathcal{I}, b \in \mathcal{B}\} \cup \{\langle b, v_0\rangle \mid b \in \mathcal{B}\}$. Figure 3.1 shows an example of the constructed SDN $G = (V, E)$.

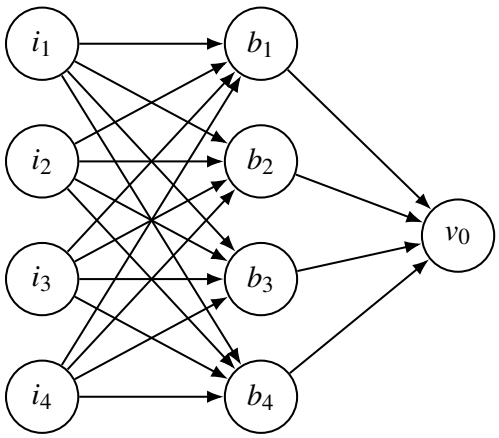


Figure 3.1: An example of the SDN G constructed from an instance of the GAP with four items and four bins.

The forwarding table size at each node in V and the bandwidth resource capacity of each link in E are set to infinity. Moreover, $V_{pm} = \mathcal{B}$ and the computing capacity of each node m in V_{pm} is set to $cap(m)$, the capacity of bin m.

We then generate a set of requests S: For each item $n \in \mathcal{I}$, we add to S a request $r_i = \langle s_i, t_i, b_i, SC_i, d_i\rangle$, where s_i is set to the switch $n \in V$, t_i is set to the virtual sink t, $b_i = 0$, the computing resource demand $C(n, m)$ to process its service chain at $m \in V_{pm}$ is $size(n, m)$, and $d_i = \infty$. Therefore, routing the set of requests S into network G is an instance of the network throughput maximization problem. We finish by noting that the network throughput maximization problem has a solution of admitting K requests if and only if the GAP with identical profits has a solution of profit K. $\square$

3.3 Integer Linear Program

In this section, we formulate the network throughput maximization problem as an Integer Linear Program (ILP), where x_i is a decision variable with value 1 if request r_i is admitted and 0 otherwise. z_i^v is a decision variable with value 1 if and only if the traffic of r_i is processed by the PM attached to switch $v \in V_{pm}$. For brevity, denote by $\delta^+(v)$ and $\delta^-(v)$ the sets of leaving and entering edges of a switch $v \in V$, respectively. In addition, to distinguish between traffic before and after being processed at a PM, we introduce two decision variables $w_i^{pre}(e)$ and $w_i^{post}(e)$ with value 1 if and only if link e carries the unprocessed and processed traffic, respectively. The detailed description is given in Figure 3.2.

Constraint (3.2) ensures that if and only if a request $r_i \in S(t)$ is admitted, it will be processed in exactly one PM. The volume of the traffic may change after the processing at v, while the volume is conserved at other non-terminal switches except the switch $v \in V_{pm}$ where it is processed.

Constraints (3.3) and (3.4) capture *traffic changing* at PM-attached switches that process traffic of user requests and *traffic conservation* at non-terminal switches. Specifically, if request r_i is processed at $v \in V_{pm}$, then (i) exactly one incoming edge of v carries the unprocessed traffic and none of the outgoing edges of v carries the unprocessed traffic; and (ii) exactly one of the outgoing edges of v carries the processed traffic, and none of the incoming edges of v carries the processed traffic. Otherwise, if the traffic of r_i is not processed by the PM attached to switch $v \in V_{pm}$ but goes through v, either (i) exactly one incoming edge and one outgoing edge of v carry the unprocessed traffic, or (ii) exactly one incoming edge and one outgoing edge of v carry the processed traffic.

Constraints (3.5) and (3.8) ensure that no unprocessed traffic enters any source switch s_i and no processed traffic leaves the terminal switch t_i.

Constraints (3.6) and (3.7) handle the cases where the traffic of a request v_i is processed at the source switch s_i or the terminal switch t_i.

$$\textit{maximize} \quad \sum_{i=1}^{|S(t)|} x_i, \tag{3.1}$$

subject to

$$\sum_{v \in V} z_i^v = x_i, \qquad i = 1, \ldots, |S(t)| \tag{3.2}$$

$$\sum_{e \in \delta^-(v)} w_i^{pre}(e) - \sum_{e \in \delta^+(v)} w_i^{pre}(e) = z_i^v, \qquad \forall v \in V \setminus \{s_i\}, \;\; i = 1, \ldots, |S(t)| \tag{3.3}$$

$$\sum_{e \in \delta^+(v)} w_i^{post}(e) - \sum_{e \in \delta^-(v)} w_i^{post}(e) = z_i^v, \qquad \forall v \in V \setminus \{t_i\}, \;\; i = 1, \ldots, |S(t)| \tag{3.4}$$

$$\sum_{e \in \delta^-(s_i)} w_i^{pre}(e) = 0, \qquad i = 1, \ldots, |S(t)| \tag{3.5}$$

$$\sum_{e \in \delta^+(s_i)} w_i^{pre}(e) = x_i - z_i^{t_i}, \qquad i = 1, \ldots, |S(t)| \tag{3.6}$$

$$\sum_{e \in \delta^-(t_i)} w_i^{post}(e) = x_i - z_i^{t_i}, \qquad i = 1, \ldots, |S(t)| \tag{3.7}$$

$$\sum_{e \in \delta^+(t_i)} w_i^{post}(e) = 0, \qquad i = 1, \ldots, |S(t)| \tag{3.8}$$

$$\sum_{e \in E} \left(w_i^{pre}(e) + w_i^{post}(e)\right) + \sum_{v \in V} z_i^v \cdot D_p(i, v) \leq d_i, \qquad i = 1, \ldots, |S(t)| \tag{3.9}$$

$$\sum_{i=1}^{|S(t)|} \left(b_i \cdot w_i^{pre}(e) + \lambda_i \cdot b_i \cdot w_i^{post}(e)\right) \leq B_e, \qquad \forall e \in E \tag{3.10}$$

$$\sum_{i=1}^{|S(t)|} \sum_{e \in \delta^+(v)} \left(w_i^{pre}(e) + w_i^{post}(e)\right) \leq L_v, \qquad \forall v \in V \tag{3.11}$$

$$\sum_{i=1}^{|S(t)|} z_i^v \leq C_v, \qquad \forall v \in V \tag{3.12}$$

$$w_i^{pre}(e), w_i^{post}(e) \in \{0, 1\}, \qquad \forall e \in E, \;\; i = 1, \ldots, |S(t)| \tag{3.13}$$

$$x_i \in \{0, 1\}, \qquad i = 1, \ldots, |S(t)| \tag{3.14}$$

$$z_i^v \in \{0, 1\}, \qquad \forall v \in V_{pm}, \;\; i = 1, \ldots, |S(t)| \tag{3.15}$$

$$z_i^v = 0, \qquad \forall v \in V \setminus V_{pm}, \;\; i = 1, \ldots, |S(t)|. \tag{3.16}$$

Figure 3.2: An ILP formulation of the network throughput maximization problem

Constraint (3.9) enforces that the end-to-end delay requirement, which is the sum of the network delay $D_n(P_i)$ and the processing delay $D_p(i, v)$, of every admitted request is met, where the network delay $D_n(P_i)$ is calculated by $\sum_{e \in E} \left(w_i^{pre}(e) + w_i^{post}(e)\right)$ and the processing delay $D_p(i, v)$ is $\sum_{v \in V} z_i^v \cdot D_p(i, v)$. Since z_i^v is 1 only for node v that implements the consolidated middleboxes for request r_i, only the processing delay at the node v is incurred.

Constraint (3.10) enforces the bandwidth capacity constraint for each link $e \in E$. Constraint (3.11) imposes the forwarding table capacity constraint for each switch $v \in V$, and Constraint (3.12) models the computing capacity constraint of PMs attached to each switch $v \in V_{pm}$.

Constraint (3.13), (3.14), and (3.15) restrict the range of decision variables to 0 and 1 inclusively. Constraint (3.16) indicates that if there is no PM at a switch $v \in V \setminus V_{pm}$, then it cannot process any request.

Since the ILP solution is time-consuming, it is only applicable when the problem size is small. The rest of this chapter will develop efficient, scalable solutions to the problem.

3.4 A Heuristic Algorithm

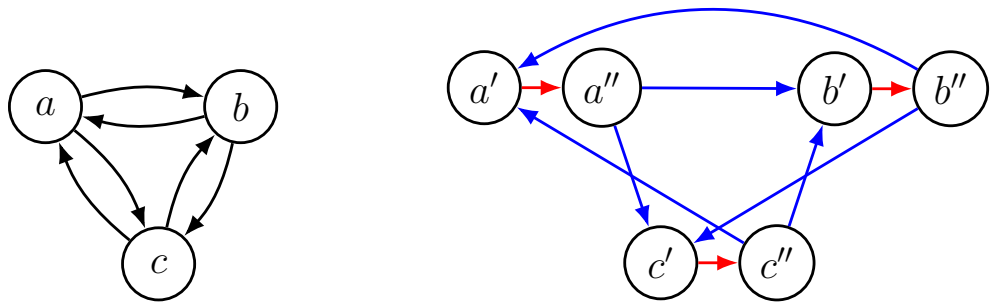
In this section, we focus on devising an efficient heuristic for the problem. We first propose a cost model to capture the dynamic resource usages in G, and then devise the algorithm through a reduction that reduces the problem into shortest path findings in a series of auxiliary graphs derived from G.

3.4.1 A Novel Cost Model of Resource Usages and the Construction of an Auxiliary Graph

Given an SDN G, it contains different types of resources such as computing resources at servers, TCAM sizes at switches, and bandwidth resources at links. Designing an efficient algorithm for the network throughput maximization problem needs to utilize

these resources judiciously, through the guidance of an efficient cost metric that can accurately capture the usages and utilizations of different resources. In the following, we first propose a cost model of resource usages. We then reduce the problem of concern in G into another problem of finding shortest paths in a series of auxiliary graphs G'_i that are derived by implementing the service chain at different servers in G.

Given an SDN $G = (V, E)$ and a request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle$, the auxiliary graph $G'_i = (V'_i, E'_i; \omega_i)$ for request i is constructed as follows. For each switch v in V, two vertices v' and v'' are added to V'_i, and a directed edge $\langle v', v'' \rangle$ is added to E'_i. For each link $\langle u, v \rangle$ in E, an edge $\langle u'', v' \rangle$ is added to E'_i, i.e., $V'_i = \{v', v'' \mid v \in V\}$ and $E'_i = \{\langle v', v'' \rangle \mid v \in V\} \cup \{\langle u'', v' \rangle \mid \langle u, v \rangle \in E\}$. Intuitively, each edge $\langle v', v'' \rangle$ in G'_i represents switch node v and an edge $\langle u'', v' \rangle$ represents link $\langle u, v \rangle$ in G. An example of such an auxiliary graph is given in Figure 3.3.



(a) $G = (V, E)$ (b) $G'_i = (V'_i, E'_i)$

Figure 3.3: The auxiliary graph construction of G'_i from G for the ith request: (a) the original SDN $G = (V, E)$; and (b) the corresponding auxiliary graph $G'_i = (V'_i, E'_i)$ of G.

The cost model of resource usages in G is proposed as follows. For a given type of resource, the marginal cost of its usage dramatically inflates with the increase of its utilization ratio, since the larger proportion of the resource is occupied, the higher risk the resource capacity will be violated. We therefore use an exponential function to model the cost of resource usage.

Denote by $RL_{v,i}$ the residual capacity of the forwarding table at $v \in V$ and $RB_{e,i}$

the residual bandwidth of link $e = \langle v, u \rangle \in E$ when request r_i arrives. Then, the weights of their corresponding edge $e' \in E'_i$ in G'_i are

$$\omega_i(e') = \begin{cases} \alpha^{1-\frac{RL_{v,i}}{L_v}} & \text{if } e' = \langle v', v'' \rangle \in E'_i, \\ \beta^{1-\frac{RB_{\langle v,u \rangle,i}}{B_{\langle v,u \rangle}}} & \text{if } e' = \langle v'', u' \rangle \in E'_i, \end{cases} \tag{3.17}$$

where α and β are constants with $\alpha, \beta > 1$. The larger the values of α and β, the more the resources with high utilizations will be discouraged to use, since their marginal costs will increase with the increase of their utilization ratios.

Notice that the usage cost of computing resource in PMs has not been incorporated into the auxiliary graph G'_i, because admitting a request r_i via a PM-attached switch $v \in V_{pm}$ does not necessarily consume the computing resource of the PM. Only if the service chain SC_i of r_i is realized in it will the computing resource of its attached PM be consumed.

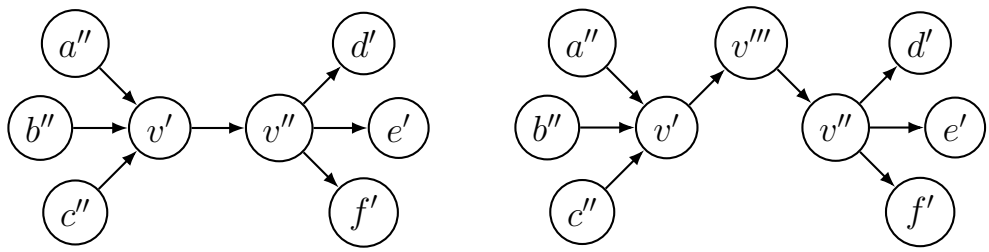
3.4.2 Algorithm

The basic idea behind the proposed algorithm is to reduce the problem in G into finding the shortest paths in a series of graphs G'_i with $1 \leq i \leq |S|$. In the following we first consider a single request admission. We then extend the solution to the admissions of a set of requests.

The detailed algorithm is described as follows. We first consider admitting a request $r_i \in S(t)$ where $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle$. We find a shortest path in $G'_i = (V'_i, E'_i)$ from s_i to t_i such that its corresponding routing path in G meets both its bandwidth demand b_i and its end-to-end delay d_i and there is a switch $v \in V_{pm}$ attached a PM in the path with sufficient computing resource to process its service chain SC_i. Specifically, we first remove the edges without adequate resources from G, and then construct $G'_i = (V'_i, E'_i)$ from the resulting graph G.

To include computing resource in PMs for the admission of request r_i, we then augment G'_i for each PM-attached switch $v \in V_{pm}$ and denote by $G'_{i,v} = (V'_{i,v}, E'_{i,v})$

the graph obtained by augmenting G'_i for a PM-attached switch $v \in V_{pm}$. The only difference between $G'_{i,v}$ and G'_i is that the directed edge $\langle v', v'' \rangle$ is removed, and a new node v''' and edges $\langle v', v''' \rangle$ and $\langle v''', v'' \rangle$ are added to $V'_{i,v}$ and $E'_{i,v}$, respectively, as shown in Figure 3.4 (b).



(a) Part of auxiliary graph $G'_i = (V'_i, E'_i)$

(b) Part of augmented auxiliary graph $G'_{i,v} = (V'_{i,v}, E'_{i,v})$

Figure 3.4: Augmenting auxiliary graph G'_i on the left to $G'_{i,v}$ on the right for switch $v \in V_{pm}$

Moreover, the weight of edge $\langle v''', v'' \rangle$ is identical to the weight of $\langle v', v'' \rangle$ in G'_i while the weight of $\langle v', v''' \rangle$ is $\gamma^{1-\frac{RC_v}{C_v}}$, where $\gamma > 1$ is a tuning parameter which usually is a constant, RC_v is the residual computing capacity, and C_v is the capacity of v. Therefore, if $v \in V_{pm}$ is considered to process service chain SC_i of request r_i, routing the traffic of r_i is to find a path $P_i(v)$ in $G'_{i,v}$ that is the concatenation of a shortest path in $G'_{i,v}$ from s_i to v and a shortest path in $G'_{i,v}$ from v to t_i. Let $l(P_i(v))$ and $d(P_i(v))$ be the length and delay of $P_i(v)$, i.e., $l(P_i(v)) = \sum_{e \in P_i(v)} \omega_i(e)$ and $d(P_i(v)) = \sum_{e \in P_i(v)} d(e) + d(i, v)$, where $d(i, v)$ is the processing duration of SC_i of r_i at the PM attached to switch v.

The problem of admitting request r_i in G is then reduced to the problem of finding a shortest path $P_i(v)$ from one of the augmented auxiliary graphs $G'_{i,v}$ derived from node v with the minimum length $\min\{l(P_i(v)) \mid v \in V_{pm}\}$ that meets the end-to-end delay d_i. The detailed description of the algorithm is given in `Procedure` 3.1.

We say that the derived "routing path" P_i for request r_i at step 6 in `Procedure` 3.1 is a pseudo-routing path or a walk, i.e., the nodes and links on P_i may appear multiple times, it can even contain cycles. This is unavoidable for a certain type of network

Procedure 3.1 Admitting a single request r_i

Input: an SDN $G = (V, E)$ and the current considering request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle$
Output: find a routing path $P_i = \langle s_i, \ldots, v \in V_{pm}, \ldots, t_i \rangle$ that satisfies b_i, SC_i, and d_i if it exists.

Construct the auxiliary graph $G'_i = (V'_i, E'_i; \omega_i)$ for G;
$P_i^{sel} \leftarrow \infty$; /* a path in an augmented auxiliary graph with the minimum sum of edge weights */
$l_{\min} \leftarrow \infty$; /* the minimum length of routing paths */
for each PM-attached switch $v \in V_{pm}$ **do**
  Construct $G'_{i,v}$ by augmenting G'_i;
  Let $P_i(v)$ be the concatenation of a shortest path in $G'_{i,v}$ from s'_i to v''' and a shortest path in $G'_{i,v}$ from v''' to t'_i;
  if $(d(P_i(v)) + d(i, v) \leq d_i)$ & $(l(P_i(v)) \leq l_{\min})$ **then**
    $P_i^{sel} \leftarrow P_i(v)$;
    $l_{\min} \leftarrow l(P_i(v))$;
    $v_{\min} \leftarrow v$; /* which PM will be used */
  end if
end for
The corresponding *pseudo-routing path (walk)* P_i in G is then derived from P_i^{sel} via PM $v_{\min}$ if it exists.

topologies. In the following, we show the existence of a simple shortest path P_i in G for request r_i if G meets certain conditions.

Lemma 3.2. *Given a directed weighted graph $G = (V, E)$, a specific node v, and a request r with source s and destination t, there is a simple shortest path in G from s to t that passes through node v if any path in another graph H from nodes v_0 to v does not contain any articulation points, where v_0 is a virtual node and edges $\langle v_0, s \rangle$ and $\langle t, v_0 \rangle$ are two virtual edges with weights of zeros, and they are added to graph G, i.e., $H = (V \cup \{v_0\}, E \cup \{\langle v_0, s \rangle, \langle t, v_0 \rangle\})$ is then obtained.*

Proof. It is known that v_0 is only connected with nodes s and t in H, if there is an articulation point u in any path from v_0 to v, this implies that any path between s (or t) and node v must pass through u, thus, if a path in G from s to t must contain u, then it appears in the path at least twice. □

Lemma 3.2 provides a necessary condition of the existence of a simple path in G from s to t that passes through v. That is, such a simple path exists if any path in H

from v_0 to v does not contain articulation points. If for any request r and a specified node v, the condition in Lemma 3.2 holds, a simple shortest path in G from s to t via v can be found as follows.

We start with *the minimum-cost two edge-disjoint path problem*: Given two nodes s and t in $G = (V, E)$, the problem is to find two edge-disjoint paths between s and t such that the sum of weighted edges in these two paths is minimum. There is an efficient algorithm for this problem due to Suurballe [123], and an improved algorithm later is proposed by Suurballe and Tarjan [124].

To find two edge-disjoint paths in graph G from s to t such that the cost sum of the two paths is minimum, Suurballe's algorithm proceeds as follows. It first finds a shortest path in G from s to t. It then reverses the direction of the edges in the shortest path, and finds a shortest path in the resulting graph from s to t. As a result, two edge-disjoint paths between s and t are then found through the exclusive union of the two found paths, and the cost sum of the two paths is the minimum one [123]. Clearly, this algorithm can be modified to find two node-disjoint paths between a pair of nodes so that the cost sum of the two paths is minimum, by adopting the node splitting technique due to Suurballe [123].

We now consider the simple shortest path problem in G between a pair of nodes s and t that passes through a specified node v. We reduce this problem in G to the problem of finding two node-disjoint paths in another graph H' such that the cost sum of the two paths is minimum. We first construct a directed auxiliary graph $H' = (V_{H'}, E_{H'})$ where $V_{H'} = \{v', v'' \mid v \in V\} \cup \{v_0\}$ and $E_{H'} = \{\langle u', v'' \rangle, \langle v', u'' \rangle \mid (u, v) \in E\} \cup \{\langle v_0, s \rangle, \langle v_0, t \rangle\}$, by adding a virtual node v_0 and virtual edges into H' and assigned both newly added virtual edges $\langle v_0, s \rangle$ and $\langle v_0, t \rangle$ with weights of zeros. We then find two edge-disjoint paths in H' between v_0 and v' such that the weighted sum of the paths is minimum. We finally have a simple path in G from s to t via v that is derived from the found two node-disjoint paths, by removing the virtual node v_0 and its incident two edges. The resulting path between s and t is a simple path via v and

the sum of its weighted edges is the minimum one.

Having considered a single request admission, in the following we deal with the admissions of a set of requests $S(t)$ at time slot t, by admitting the requests one by one until no more requests can be admitted. A non-admitted request in $S(t)$ is admitted immediately if it has the minimum implementation cost at that moment. Specifically, given a to-be-admitted request $r_i \in S(t)$, `Procedure` 3.1 is employed to find a routing path for r_i *without committing the admission* which means that the SDN controller does not allocate resources to meet the demands by this request. A found path P_i^{sel} with the minimum cost among all remaining requests in $S(t)$ will be admitted and its demanded resources will be allocated to it, the residual resource availabilities in G are updated accordingly. Meanwhile, if `Procedure` 3.1 fails to find a path P_i^{sel} for r_i, request r_i will be rejected at time slot t. This procedure repeats until every request in $S(t)$ is either rejected or admitted. The detailed description is given by Algorithm 3.1.

Algorithm 3.1 A heuristic for admitting a set of requests $S(t)$

Input: an SDN $G = (V, E)$ and a set of requests $S(t)$
Output: Determine which request $r_i \in S(t)$ to be admitted and its routing path P_i^{sel}

1: $S' \leftarrow S(t)$; /* the set of requests to be admitted */
2: **while** $S' \neq \emptyset$ **do**
3: **for each** request $r_i \in S'$ **do**
4: Find a routing path P_i^{sel} for request r_i, by invoking `Procedure` 3.1;
5: **if** path P_i^{sel} does not exist **then**
6: $S' \leftarrow S' \setminus \{r_i\}$; /* remove r_i from S' */
7: **end if**
8: **end for**
9: Let r_{i_0} be the request with $l(P_{i_0}^{sel}) = \min_{r_i \in S(t)}\{l(P_i^{sel})\}$;
10: Admit request r_{i_0} using the routing path $P_{i_0}^{sel}$, and update the resource availabilities of G by deducting the resources for accommodating $P_{i_0}^{sel}$;
11: $S' \leftarrow S' \setminus \{r_{i_0}\}$.
12: **end while**

3.4.3 Algorithm Analysis

In the following, we first show that the solution delivered by Algorithm 3.1 is feasible, and then analyze its time complexity.

Lemma 3.3. *Given the augmented auxiliary graph $G'_{i,v} = (V'_{i,v}, E'_{i,v})$ derived from $G = (V, E)$ and a switch $v \in V_{pm}$ for request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle \in S(t)$, the concatenation of a shortest path from s'_i to v''' and another shortest path in $G'_{i,v}$ from v''' to t'_i will result in a valid pseudo-routing path in G from s_i to t_i with PM-attached switch v in the path. Alternatively, a simple shortest path from s'_i to t'_i through v''' delivered by applying Suurballe's algorithm is also a feasible solution if G meets the condition in Lemma 3.2.*

Proof. Let $P_i(v)$ be the concatenation of a shortest path from s'_i to v''' and a shortest path from v''' to t'_i in $G'_{i,v}$. For simplicity, we use a *link-derived edge* to represent an edge $\langle u'', v' \rangle$ in $E'_{i,v}$ that is derived from edge $\langle u, v \rangle$ in E, and a *switch-derived edge* to denote an edge $\langle v', v'' \rangle$ in $E'_{i,v}$ that is derived from switch $v \in V$. We claim that (i) path $P_i(v)$ consists of link-derived and switch-derived edges alternatively; and (ii) path $P_i(v)$ can satisfy the requirements of request r_i, i.e., the bandwidth demand b_i, the forwarding table demand, the computing resource demand for its service chain SC_i, and the end-to-end requirement d_i. Claim (i) is obvious because there is only an outgoing edge for each switch v', i.e., $\langle v', v'' \rangle$. Claim (ii) holds because the augmented auxiliary graph $G'_{i,v}$ is the result of removing the edges and switches in G'_i that cannot meet resource requirements of request r_i, and path $P_i(v)$ is feasible only when its end-to-end delay is no greater than d_i.

The feasibility of the simple shortest path via a data center if it does exist can be proven similarly, omitted. □

Theorem 3.1. *Given an SDN $G = (V, E)$ with a set V of switches and a set E of links, a subset $V_{pm} \subseteq V$ of switches with attached PMs, a set of user requests $S(t)$ at time slot t, there is an algorithm, Algorithm 3.1, for the network throughput maximization problem, which delivers a feasible solution in $O(|S(t)|^2|V|^4)$ time.*

Proof. The solution delivered by Algorithm 3.1 is feasible because each auxiliary graph is constructed from the subgraph of G that only includes the resources with sufficient residual capacities for the request. Consequently, the routing path in G derived from the found path in G'_i is feasible.

The time complexity of Algorithm 3.1 is analyzed as follows. In `Procedure` 3.1, the construction and augmentation of the auxiliary graph G'_i take $O(|V| + |E|)$ time, while finding a shortest path in each of the $|V_{pm}|$ augmented auxiliary graphs $G'_{i,v}$ takes $O(|V|^3)$ time. `Procedure` 3.1 thus takes $O(|V|^3 + |V| + |E|) = O(|V|^3)$ time. For each request $r_i \in S(t)$, `Procedure` 3.1 is invoked at most $|V_{pm}|$ times. The number of requests is $O(|S(t)|)$. If we make use of Suurballe's algorithm to find a simple shortest path from s'_i to t'_i in $G'_{i,v}$ through v''', it takes $O(|E'_i| + |V'_i| \log |V'_i|) = O(|E| + |V| \log |V|)$ time as the construction of the auxiliary graph H' and finding shortest paths in H take no more than that amount of time. The time complexity of Algorithm 3.1 thus is $O(|S(t)|^2 |V_{pm}||V|^3) = O(|S(t)|^2 |V|^4)$. The theorem holds. □

3.5 A Faster Heuristic Algorithm

Although Algorithm 3.1 delivers a near optimal solution empirically, which can be seen in later experimental evaluations, its running time is quite high and may fail to respond to user requests on time, considering user requests arrive one by one without the knowledge of future request arrivals. In this section we devise a faster heuristic that strives for the non-trivial trade-off between the accuracy of a solution and the running time of obtaining the solution.

3.5.1 Overview

A key ingredient of this faster heuristic is that a candidate solution to admit a subset of $S(t)$ of requests based on the residual resource capacities of G in the beginning of time slot t, and there is no updating to these residual capacities when all requests in $S(t)$ are considered. Thus, a candidate solution is identified first. It then further refines the candidate solution iteratively until no resource capacity violation occurs.

3.5.2 Algorithm

We first find a set of candidate routing paths $\mathcal{P}_i$ in G for each request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle \in S(t)$, without considering the resource capacity constraints of G, where a shortest path from s_i to t_i is treated as a candidate path of r_i as long as it has one PM-attached switch in V_{pm} that satisfies b_i, SC_i, and d_i. As the service chain SC_i of r_i must be served at one of $|V_{pm}|$ PM-attached switches, we can find at most $|V_{pm}|$ candidate shortest paths for each r_i. Notice that we find candidate routing paths for requests in $S(t)$ on the augmented auxiliary graphs based on the resource availability of G as of the beginning of time slot t, through finding a shortest path from s'_i to v''' and a shortest path from v''' to t'_i in $G'_{i,v}$ for each request $r_i \in S(t)$ and $v \in V_{pm}$. Let $P_i(v_j) = \langle s_i, \ldots, v_j, \ldots, t_i \rangle$ be a found path in G_{i,v_j} for request r_i, whereas v_j ($\in V_{pm}$) is a switch that fulfills the service chain SC_i and $P_i(v_j)$ meets the resource and end-to-end delay constraints of r_i. Denote by $\mathcal{P}_i$ be the set of candidate paths for request r_i, we then have,

$$\mathcal{P}_i = \{P_i(v_j) \mid v_j \in V_{pm}\}. \tag{3.18}$$

Having the set of candidate paths $\mathcal{P}_i$ for each request r_i, we then pick only one candidate path $P_i(v_j)$ in $\mathcal{P}_i$ for request r_i in a way such that the cost sum of the selected paths for all requests is minimized, while ensuring that the computing capacity of each PM is not violated. In essence, selecting a path $P_i(v_j) \in \mathcal{P}_i$ to route request $r_i \in S(t)$ is equivalent to selecting a PM attached to a switch $v \in V_{pm}$ to implement SC_i for r_i. As different PMs may have different computing capacities, this means that the processing SC_i of r_i at different PMs will incur different computing resource demands. We thus reduce this problem to the GAP, which is defined as follows. Given a set of items $\mathcal{I}$ and a set of bins $\mathcal{B}$, where each bin $b \in \mathcal{B}$ has a capacity $cap(b)$, each item $i \in \mathcal{I}$ has a size $size(i, b)$, and a profit $profit(i, b)$ if item i is placed in bin b, the problem is to place a subset of items U ($\subseteq \mathcal{I}$) in bins $\mathcal{B}$ such that the sum of the profits of items in U is maximized and the sum of sizes of items placed in every bin is no more than the capacity of the bin.

We now treat each PM-attached switch $v_j \in V_{pm}$ as a bin and each request r_i in $S(t)$ as an item, whereas the capacity of each bin v_j is its residual computing capacity, i.e., $cap(v_j) = LC_{v_j}$, the size of an item r_i in a bin v_j is the computing demand of the service chain SC_i in the PM attached to v_j, i.e., $size(r_i, v_j) = C(i, j)$, and the profit of placing an item r_i in a bin v_j is the reciprocal of the length of the candidate path that fulfills r_i on v_j, i.e., $profit(i, j) = \frac{1}{l(P_i(v_j))}$.

Having reduced the network throughput maximization problem to the GAP, we now solve the GAP and each solution to the GAP yields a solution to the original problem. Specifically, we use the algorithm proposed by Cohen et al. [25] that guarantees a $(2 - \epsilon)$-approximation ratio, where ϵ is a constant with $0 < \epsilon \leq 1$, to solve the GAP. Denote by U a solution found by this algorithm as a placement of a subset of items in bins. U yields a potential admission of requests in $S(t)$: for every request r_i treated as an item, if it is placed in a bin representing $v_j \in V_{pm}$, then it is admitted with the routing path $P_i(v_j)$; otherwise, r_i is rejected.

Due to the construction of the GAP, admitting requests in S based on the solution U to the GAP ensures that the sum of computing demands of requests of which the service chains are fulfilled in the same PM will not exceed the computing capacity of the PM. However, the bandwidth and forwarding table capacities may be violated, as routing paths may have overlapping resources. Now, for each request allocated to a bin, its computing demand can be met without violating the computing capacity of the bin. Some requests however may violate the bandwidth and forwarding table size capacities of some links and nodes while routing their traffic. We thus perform adjustments to eliminate such potential resource violations by selectively rejecting some requests. Let $P_i^{sel} = P_i(v_j) = \langle s_i, \ldots, v_j, \ldots, t_i \rangle$ be the path to route the traffic of request r_i according to U, where $v_j \in V_{pm}$. The basic idea behind the adjustment here is to carefully find such a path with resource capacity violations iteratively and remove its request from admission. This procedure continues until there is no violation of resource capacity. To this end, a bipartite graph $G_b = (U_b, V_b, E_b)$ is constructed,

where U_b is the set of selected routing paths for all potentially admitted requests, V_b is the set of edges in $\cup_{r_i \in S(t)} E(P_i^{sel})$ which each corresponds to a resource in G. There is an edge between a node $P_i^{sel} \in U_b$ and a node $e \in V_b$ if e is in P_i^{sel}. The weight of edge $(P_i^{sel}, e) \in E_b$ is the ratio of the demand of r_i on that resource to the sum of those of all requests on that resource, which represents the contribution of r_i to the resource capacity violation of e. An example of such a bipartite graph G_b is shown in Figure 3.5.

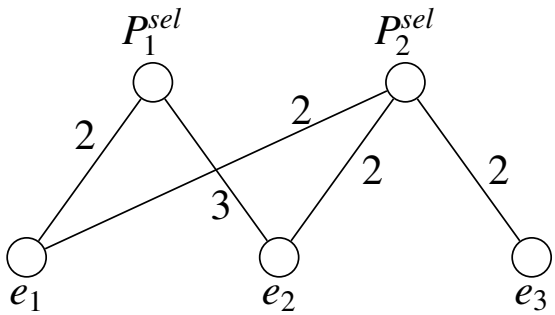


Figure 3.5: An example of a bipartite graph G_b.

To eliminate resource capacity violations, we iteratively remove one node P_i^{sel} and its incident edges in G_b with the maximum weighted sum of the incident edges, and update G_b by removing nodes in V_b that their resource overloadings are avoided due to the removal of node P_i^{sel}. For example, in Figure 3.5, both P_1^{sel} and P_2^{sel} violate the computing capacity constraints of e_1, e_2, and e_3. Since P_2^{sel} results in more violations of resource capacity constraints than P_1^{sel} does, it will be removed first. This procedure continues until no edge is left in E_b, a feasible solution will be obtained ultimately. The detailed description is given in Algorithm 3.2.

3.5.3 Algorithm Analysis

In the following, we show that the solution delivered by Algorithm 3.2 is a feasible solution. We then analyze the time complexity of the proposed algorithm.

Theorem 3.2. *Given an SDN $G = (V, E)$ with a set V of switches and a set E of links, a subset $V_{pm} \subseteq V$ of switches with each attaching with a PM, a set of user requests $S(t)$,*

Algorithm 3.2 A faster heuristic for routing a set of requests $S(t)$ into a G

Input: an SDN $G = (V, E)$ and a set of user requests $S(t)$
Output: Routing decisions for each request $r_i \in S(t)$

```
Build an auxiliary graph G' = (V', E') for G;
Initialize 𝒫, the set of candidate routing paths in G for all requests in S(t), to ∅;
for each user request r_i ∈ S(t) do
   𝒫_i ← ∅; /* the set of candidate paths for request r_i */
   for each PM-attached switch v_j ∈ V_pm do
      Find a path P_i(v_j) for r_i via node v_j, by invoking Procedure 3.1;
      if P_i(v_j) exists then
         𝒫_i ← 𝒫_i ∪ {P_i(v_j)};
      end if
   end for
   if 𝒫_i is empty then
      Reject request r_i;
   else
      𝒫 ← 𝒫 ∪ {𝒫_i};
   end if
end for
Construct an instance of the GAP by representing each request as an item and
each node in V_pm as a bin;
Solve the GAP instance by invoking the algorithm in [25];
Construct a bipartite graph G_b = (U_b, V_b, E_b) that reflects potential capacity viola-
tions;
while there are edges in E_b do
   Update G_b by the removal of such a node in U_b that has the maximum weighted
   sum of its incident edges and its incident edges from E_b.
end while
```

there is an algorithm for the network throughput maximization problem, Algorithm 3.2, which delivers a feasible solution in $O(|S(t)||V|^3 + |V| \cdot \frac{|S(t)|^3}{\epsilon})$ *time, where* ϵ *is a given constant with* $0 < \epsilon \leq 1$.

Proof. Recall that Algorithm 3.2 consists of three phases: (i) find a set of candidate routing paths for each request; (ii) select only one routing path for each request to meet computing capacities of nodes in V_{pm}; and (iii) eliminate the requests that violate bandwidth or forwarding table capacities. The feasibility of the solution delivered by Algorithm 3.2 immediately follows from Phase (ii).

The rest is to analyze the time complexity of Algorithm 3.2. Phase (i) takes

$O(|S(t)||V|^3)$ time, because $O(|V_{pm}|) = O(|V|)$ shortest paths are found for each request $r_i \in S(t)$ in augmented auxiliary graphs and each shortest path takes $O(|V|^2)$ time. The running time of Phase (ii) is dominated by the time required to solve the GAP, which is $O(|V| \cdot \frac{|S(t)|^3}{\epsilon})$ [25]. Phase (iii) takes $O(|S(t)|(|V| + |E|))$ time, there are $O(|S(t)|(|V| + |E|))$ edges in the bipartite graph G_b, following the construction of the bipartite graph. In the worst scenario, each request violates the resource capacities on all switches and links. The theorem thus holds. □

3.6 An Online Algorithm

In this section, we study the online network throughput maximization problem, by considering dynamic admissions of user requests within a finite time horizon T. We will make use of the proposed algorithms in the previous section to solve this problem. We assume that the system evolves over time. The time is partitioned into equal time slots, and the user request admission scheduling proceeds in the beginning of each time slot. Some implementing requests may also leave the system, and the resources occupied by them will be released back to the system in the end of the current time slot. The released resources will be available in the beginning of the next time slot. The detailed online algorithm for dynamic request admissions is given in Algorithm 3.3.

Algorithm 3.3 Online algorithm within a finite time horizon T

Input: an SDN $G = (V, E)$ and time horizon T
Output: determine which request $r_i \in S(t)$ to be admitted and its routing path P_i^{sel} at each time slot t with $1 \leq t \leq T$.

```
for t ← 1 to T do
    Release all resources occupied by the requests that left in the end of time slot
    (t − 1), and recalculate the residual resources in G;
    Let S(t) be the set of arrived requests in the beginning of time slot t;
    if S(t) ≠ ∅ then
        Find a subset S′(t) ⊆ S(t) of requests that are admissible at time slot t,
        by invoking either Algorithm 3.1 or Algorithm 3.2 based on the available
        resources in G.
    end if
end for
```

Theorem 3.3. *Given an SDN $G = (V, E)$ with a set V of switches and a set E of links, a subset $V_{pm} \subseteq V$ of switches with each attaching with a PM, and a finite time horizon T, there is an algorithm for the online network throughput maximization problem, Algorithm 3.3, which delivers a feasible solution in $O(\sum_{t=1}^{T}(|S(t)||V|^3 + |V| \cdot \frac{|S(t)|^3}{\epsilon}))$ time if Algorithm 3.2 is used as its subroutine, where ϵ is a given constant with $0 < \epsilon \leq 1$.*

Proof. The time complexity of Algorithm 3.3 per time slot is the identical to the one for Algorithm 3.2, omitted. □

3.7 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms through experimental simulations, using real and synthetic SDNs. We start with the experimental environments, we then evaluate the performance of the proposed heuristic algorithms for the network throughput maximization problem. We then evaluate the performance of the online algorithms for the online network throughput maximization problem. We finally investigate the impact of parameters on the performance of the proposed algorithms.

3.7.1 Experimental Environment Settings

We adopt commonly used, real network topologies including GÉANT [100] and several ISP networks from [122] in the simulations, where GÉANT [100] is a European network consisting of 40 nodes and 122 links. The size of the forwarding table of each switch is set from 1,000 to 8,000 randomly [85]. The bandwidth of each Internet link varies from 1,000 Mbps to 10,000 Mbps [83]. There are nine PMs for the GÉANT topology as set in [49] and the number of PMs in ISP networks are provided by [110]. The computing capacity of each PM is from 4,000 to 8,000 MHz [53]. The delay of a link is between 2 milliseconds (*ms*) and 5 *ms* [83, 85]. We consider five types of middleboxes: Firewall, Proxy, NAT, IDS, and Load Balancing, and their computing demands

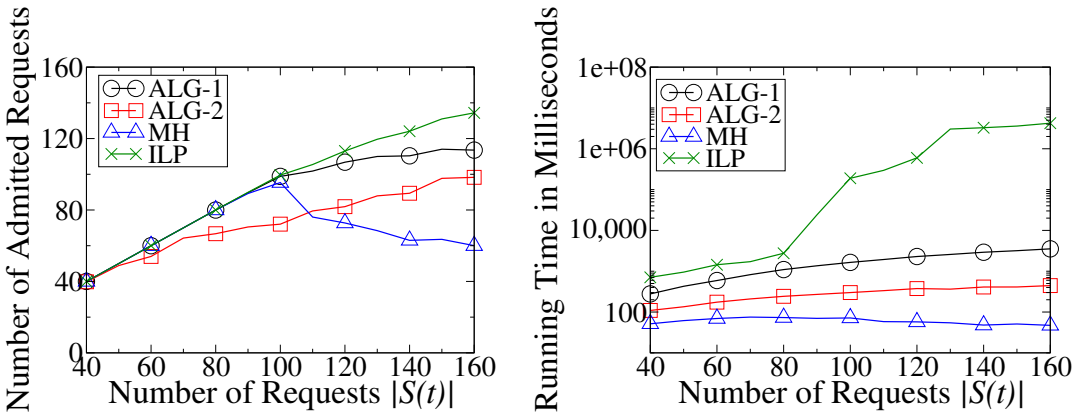
are adopted from [49, 100]. The running time is obtained based on a machine with a 3.40GHz Intel i7 Quad-core CPU and 16 GiB RAM. The default accuracy parameter ϵ in solving GAP is set to 0.1. Unless otherwise specified, these parameters will be adopted in the default setting. Each request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle \in S(t)$ is generated as follows. Given a network $G = (V, E)$, two nodes in V are randomly drawn as the source s_i and the destination t_i of request r_i. The bandwidth demand b_i is randomly drawn from 10 to 120 Mbps [4] and the end-to-end delay d_i is set from 40 ms to 400 ms randomly [102].

We evaluate Algorithm 3.1 and Algorithm 3.2 against a baseline heuristic which is described as follows. Sort all requests in $S(t)$ in increasing order of their computing resource demands, and then, for each request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle$ in $S(t)$, find a shortest path in G from s_i to a PM-attached switch v ($\in V_{pm}$) with the minimum number of hops from s_i and a shortest path from v to t_i. We refer to this minimum-hop-based baseline as algorithm `MH`, and algorithm ILP, Algorithm 3.1 and Algorithm 3.2 as `ILP`, `ALG-1` and `ALG-2`, respectively. Each value in figures is the mean of the results of 30 trials.

3.7.2 Performance of Different Algorithms within One Time Slot

In the following, we investigate the performance of the proposed algorithms `ILP`, `ALG-1`, `ALG-2`, and `MH` in the GÉANT topology within a single time slot.

Figure 3.6 (a) shows the number of requests admitted by different algorithms, when the number of requests arrived at a time slot is in the range from 40 to 160. It can be seen that both algorithms `ALG-1` and `MH` can admit as many requests as `ILP` does if there are less than 100 requests. Otherwise, only algorithm `ALG-1` can achieve a comparable throughput as `ILP`. This means that the network throughput of algorithm `ALG-2` is inferior to algorithm `ALG-1`, and the gap between their performance enlarges from nearly zero at $|S(t)| = 40$ to 21 at $|S(t)| = 160$. The reason is that algorithm `ALG-2` will reject more requests with the increase on the number of requests, as the



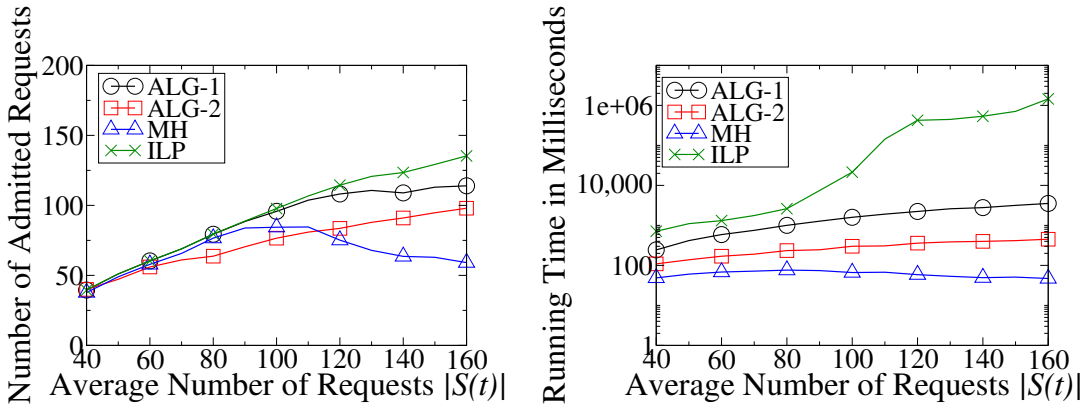
(a) The number of requests admitted if the number of requests arrived is fixed

(b) The running times (in milliseconds) on a logarithmic scale if the number of requests arrived is fixed

Figure 3.6: Performance of different algorithms on the GÉANT within one time slot if the number of request is fixed.

likelihood of routing paths that algorithm `ALG-2` finds for different requests being overlapping and resource violation soars. Meanwhile, it can be seen that algorithm `MH` outperforms algorithm `ALG-2` only when the number of requests is small. Specifically, when there are 160 requests, the number of requests admitted by algorithm `MH` is only 60% of that by algorithm `ALG-2` but runs much faster. The reason behind is that algorithm `MH` does not guarantee that the routing path of request r_i from its source s_i to its destination t_i has the minimum weight, since it finds shortest paths from s_i to a PM-attached switch and from that PM-attached switch to t_i separately. Figure 3.6 (b) illustrates the amounts of time spent by different algorithms, from which it can be seen that the running time of algorithm `ILP` is several orders of magnitude of the other mentioned algorithms, while algorithm `MH` is the fastest one, and `ALG-2` is faster than `ALG-1` significantly. In addition, the running time of algorithm `ILP` starts rising when there are more than 80 requests. The reason is that when the number of requests is small and their resource demands are relatively small compared to the network capacity, many feasible solutions that achieves the best performance exist, yet as the resource demands become increasingly considerable relative to the network capacity, fewer optimal solutions exist, and thus `ILP` spends a significant amount of time on

searching for such an optimal solution. Figure 3.6 also demonstrates that algorithm `ILP` suffers poor scalability and cannot finish within a reasonable amount of time when the problem size is large.



(a) The number of requests admitted if the number of requests arrived follows a Poisson distribution

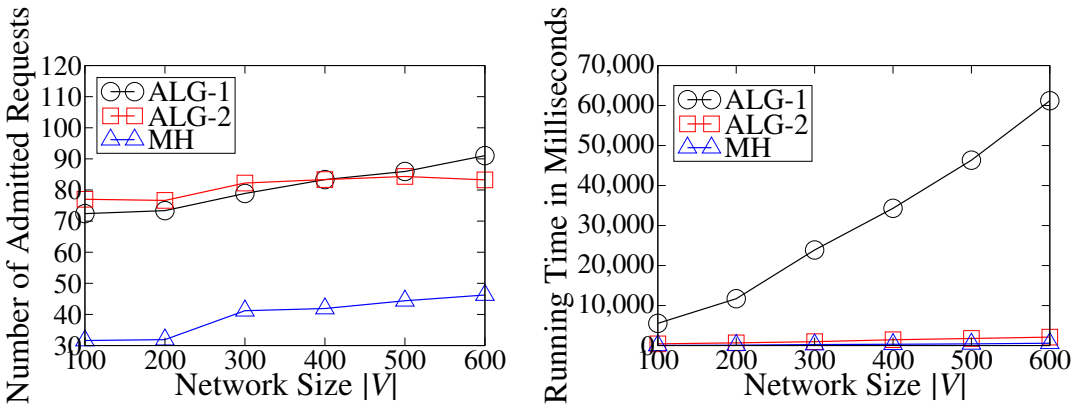
(b) The running times (in milliseconds) on a logarithmic scale if the number of requests arrived follows a Poisson distribution

Figure 3.7: Performance of different algorithms on the GÉANT within one time slot if the number of requests follows a Poisson distribution.

We now evaluate the performance of different algorithms if the number of requests arrived follows a Poisson distribution with the mean between 40 and 160. The results of the four mentioned algorithms are summarized in Figure 3.7 (a)-(b), from which it can be seen that the similar behavior patterns are present in Figure 3.6 (a)-(b). For instance, it can be seen that the number of requests admitted by algorithm `ALG-1` is identical to that by algorithm `ILP` when the number of requests is no more than 70. Meanwhile, the number of admitted requests by algorithm `ALG-2` is on a par with that of algorithm `ALG-1`, and it outperforms algorithm `MH` by 40% when there are more than 120 requests in the set. Similarly, algorithm `ILP` is the slowest one while algorithm `MH` is the fastest one. In particular, when the mean of the number of requests is set at 140, the running time of algorithm `ILP` is more than 1,000 times of that of algorithm `ALG-1`, whilst algorithm `ALG-1` is more than six times slower than algorithm `ALG-2`. Although the running time of algorithm `MH` is the fastest, the number of requests admitted by it is the smallest one.

We finally evaluate the performance of different algorithms by varying the network size. As publicly available topologies such as [83, 122] have limited sizes, we adopt the widely used Barabási-Albert model [10] to generate networks of different sizes. Namely, we vary the number of switches in an SDN from 100 to 600 while fixing the number of requests at 160. The results are depicted in Figure 3.8.

It can be seen that from Figure 3.8 (a) that algorithms `ALG-1` and `ALG-2` achieve the similar throughput, while algorithm `MH` admits only no more than half the requests admitted by either of the two heuristics. Figure 3.8 (b) reveals that algorithm `ALG-2` runs much faster than algorithm `ALG-1`. In contrast to high admission ratios delivered by algorithms `ALG-1` and `ALG-2`, algorithm `MH` admits less than one half as many as requests as the other two algorithms, neutralizing its advantage of having the lowest running time among all three algorithms indicated in Figure 3.8 (b). Figure 3.8 (b) also reveals that algorithm `ALG-2` runs much faster than algorithm `ALG-1`, e.g., algorithm `ALG-1` spends 61,208 *ms* on admitting 160 requests to a network with 600 switches, while algorithm `ALG-2` takes only 2,106 *ms*.



(a) The number of requests admitted by different algorithms

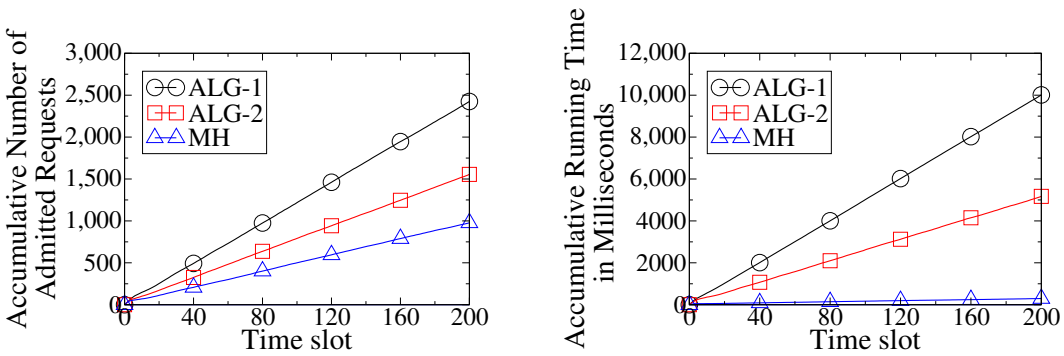
(b) Running time of different algorithms in milliseconds

Figure 3.8: Performance of different algorithms in the GÉANT by varying the number of switches from 100 to 600, while the number of requests is fixed at 160 per time slot.

3.7.3 Algorithm Performance within a Finite Time Horizon

We now consider a time horizon that consists of 200 time slots, under which we evaluate the performance of the online versions of the proposed algorithms, assuming that the number of requests at each time slot follows a Poisson distribution with a mean of 30, and each admitted request spans from 1 to 10 time slots randomly.

The results are summarized in Figure 3.9. It can be seen from Figure 3.9 (a) that algorithm `MH` has the lowest network throughput among the mentioned algorithms. On the other hand, algorithms `ALG-1` and `ALG-2` utilize resources more efficiently, and hence admit much more requests than that of algorithm `MH` by 150% and 50%, respectively. It can also be seen from Figure 3.9 (b) that the running time of algorithm `MH` is negligible compared with those of algorithms `ALG-1` and `ALG-2`. It must be noticed that this running time comes at the cost of admitting much fewer requests. Although the running time of algorithm `ALG-2` is less than that of algorithm `ALG-1`, the gap between them becomes smaller. Specifically, algorithm `ALG-2` is only half of the running time of algorithm `ALG-1` while it has only 10% of the running time of algorithm `ILP`. The main reason is that the additional time incurred from constructing an instance of the GAP cannot be ignored when the number of requests is relatively small, which will be offset when the number of requests is greater.



(a) Accumulative number of requests admitted (b) Accumulative running time of algorithms

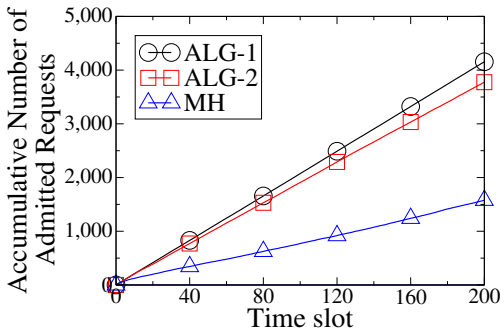
Figure 3.9: Performance of different online algorithms in the GÉANT within a time horizon of 200 time slots, where the number of requests arrives at each time slot follows a Poisson distribution with a mean of 30.

The rest is to evaluate algorithms ALG-1, ALG-2, and MH in three network topologies from [122]: AS-4755 is a network with 121 switches and 296 links, AS-1755 with 172 switches and 762 links, and AS-3967 with 212 switches and 886 links.

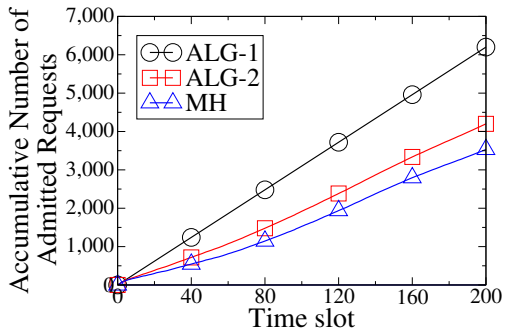
The results are illustrated in Figure 3.10, from which it can be seen that in terms of the performance, algorithm ALG-1 is the best while algorithm MH is the worst. The performance gap between algorithm ALG-1 and algorithm ALG-2 is small compared to that in the GÉANT, since the size of these three networks is larger than that of the GÉANT, and routing paths delivered by algorithm ALG-2 for different requests at each time slot are less likely to overlap. In AS-4755, the difference on requests admitted by algorithms ALG-1 and ALG-2 is less than 500, while algorithm MH admits no more than 40% the number of requests by the other two algorithms. On the other hand, the performance of algorithm MH in both AS-1755 and AS-3967 improves due to the larger resource capacity of the network. The accumulative running time of these three algorithms when admitting requests into different networks are shown in Figure 3.10 (d)-(f). The results in both Figure 3.10 (d) and (e) are similar: the running time of algorithm MH is slightly smaller than that of algorithm ALG-2, and algorithm ALG-1 is much slower than both algorithms ALG-1 and MH. However, we notice from Figure 3.10 (f) that algorithm ALG-2 is faster than algorithm MH, since algorithm ALG-2 only needs to find shortest paths in a graph once.

3.7.4 Impact of Request Durations on the Performance of Different Online Algorithms

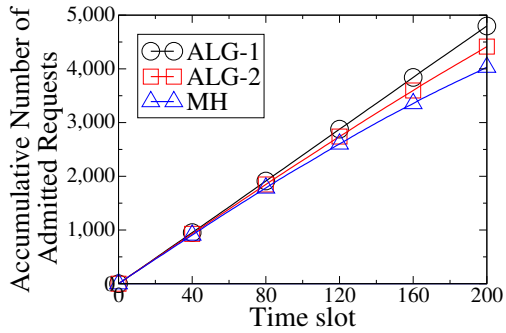
We finally evaluate the impact of the maximum duration of requests on the performance of different online algorithms based on algorithms ALG-1, ALG-2, and MH, by varying the maximum duration from 5 time slots to 25 time slots. The results are presented in Figure 3.11. From Figure 3.11 (a)-(b) we can see that the longer the maximum duration, the fewer requests admitted by all mentioned algorithms, because longer durations result in less available resources in the network. Figure 3.11 (a)-(c)



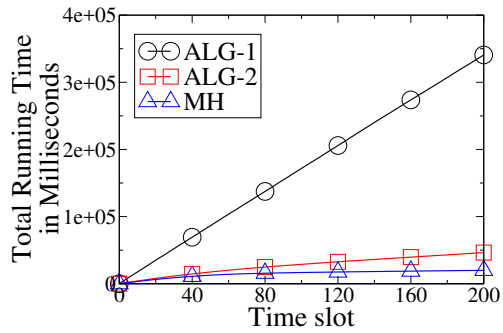
(a) The accumulative number of requests admitted in the AS-4755 network



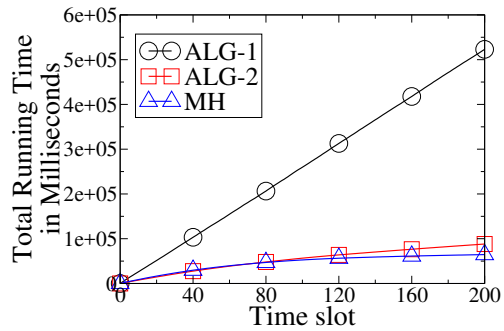
(b) The accumulative number of requests admitted in the AS-1755 network



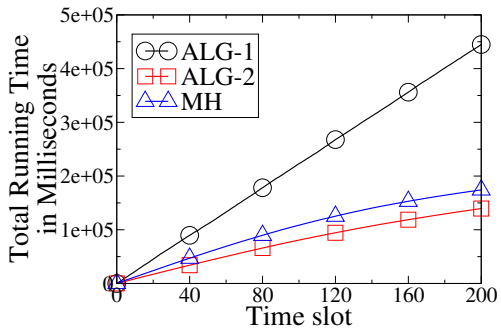
(c) The accumulative number of requests admitted in the AS-3967 network



(d) The accumulative running time of different algorithms in the AS-4755 network

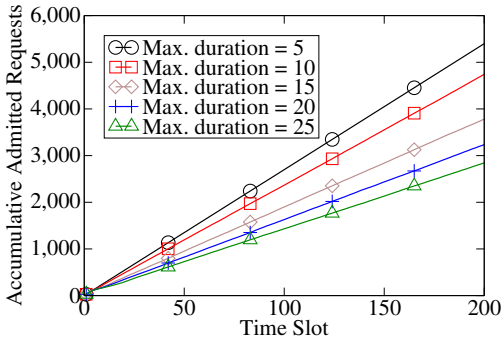


(e) The accumulative running time of different algorithms in the AS-1755 network

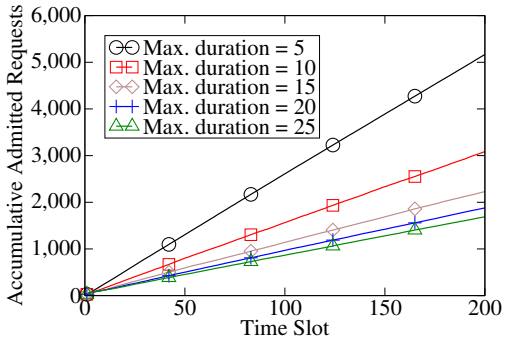


(f) The accumulative running time of different algorithms in the AS-3967 network

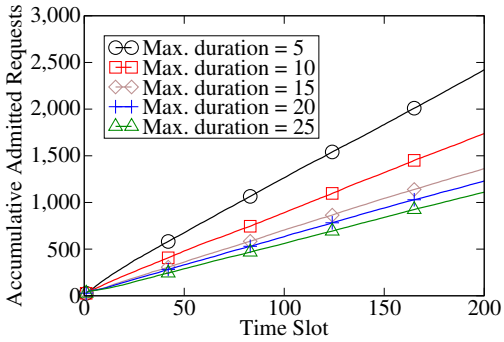
Figure 3.10: The accumulative number of admitted requests and the running time of different online algorithms based on algorithms `ALG-1`, `ALG-2` and `MH` for a time horizon of 200 time slots in ISP networks.



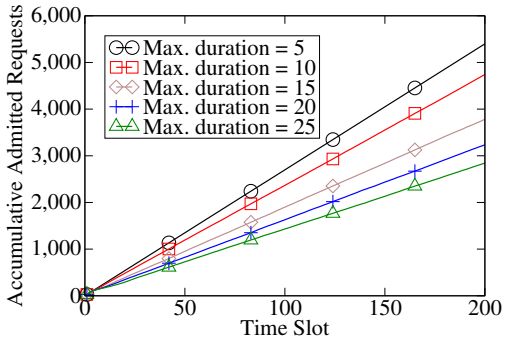
(a) The number of requests admitted by `ALG-1`



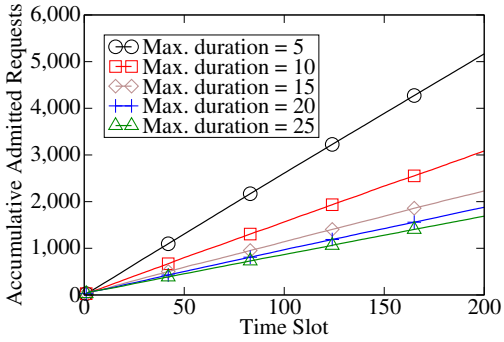
(b) The number of requests admitted by `ALG-2`



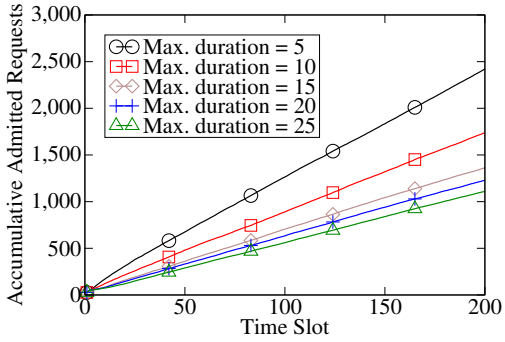
(c) The number of requests admitted by `MH`



(d) The number of requests admitted by `ALG-1`



(e) The number of requests admitted by `ALG-2`



(f) The number of requests admitted by `MH`

Figure 3.11: The accumulative number of admitted requests and running time of different online algorithms based on algorithms `ALG-1`, `ALG-2`, and `MH` with different maximum durations of requests when the network is of the GÉANT

show that when the maximum duration of requests is five time slots, algorithm `ALG-2` admits as many requests as algorithm `ALG-1`, yet algorithm `MH` only admits half the number of requests as the two heuristics. We also see that an increase in the request durations has a greater impact on algorithms `ALG-2` and `MH` than that on algorithm `ALG-1`, as algorithm `ALG-1` is more exhaustive. In addition, it can also be seen from Figure 3.11 (d)-(f) that longer durations are associated with less running time, because if other parameters keep unchanged and request durations become longer, more resources will be fully occupied and accordingly, the auxiliary graphs will have fewer vertices and edges, shortening operations on the auxiliary graphs and reducing the running time of all algorithms.

3.8 Summary

In this chapter, we studied the admissions of user requests with each having a sequence of network functions in an SDN so that the network throughput can be maximized, subject to the constraints of forwarding table capacity, network bandwidth capacity, and computing resource capacity at PMs. We first formulated an ILP solution when the problem size is small. We then devised two heuristic algorithms that strive for a fine tradeoff between the solution accuracy and the running time to obtain the solutions. We also investigated the dynamic admissions of requests within a finite time horizon by extending the proposed algorithms to solve dynamic request admissions. We finally evaluated the performance of the proposed algorithms through simulations, using real and synthetic network topologies. Experimental results demonstrated that both proposed algorithms admit more requests than a baseline algorithm, and the quality of solutions delivered is on a par with that of optimal solutions yet the proposed algorithms are significantly faster.

Chapter 4

Virtualized Network Function Placements for Delay-Sensitive Network Function Virtualization-Enabled Requests

Network Function Virtualization (NFV) has attracted significant attention from both industry and academia as an important paradigm shift on network service provisioning. Under this new NFV architecture, a *service chain* that consists of various network functions such as Firewall, Intrusion Detection System (IDS), WAN optimizer, and Deep Packet Inspection (DPI) can be decomposed into a set of Virtualized Network Functions (VNFs) that are implemented as software components in off-the-shelf physical servers [103], and these VNF implementations are often referred to as *VNF instances*. Each VNF instance can be relocated or instantiated at different locations (servers) in a network without necessarily purchasing and installing new hardware. By decoupling network functions from their traditional dedicated hardware implementation, NFV has great potentials to significantly reduce the operating expenses (OPEX) and capital expenses (CAPEX) of network service providers. NFV also facilitates the deployment of new network function services with agility and faster time-to-value [103].

Most existing studies focused on the optimal placement of VNF instances under a specific optimization objective while meeting user-specified resource demands and Quality of Service (QoS) requirements [116, 138, 139], or on historical VNF instance demand patterns or predictions [14, 82], yet little attention has been paid on maximizing network throughput via VNF instance vertical and horizontal scalings at the same time. In this chapter, we focus on maximizing the network throughput by admitting as many as NFV-enabled user requests through jointly exploring VNF instance horizontal and vertical scalings, where *horizontal scaling* is to migrate existing VNF instances from their current locations (servers) to other locations (servers) to meet the delay requirements of both currently executing and newly admitted requests, and *vertical scaling* is to instantiate new VNF instances for newly admitted requests if horizontal scaling is infeasible. Figure 4.1 is an illustrative example of vertical and horizontal scalings. where network G consists of six nodes $v_1, v_2, \ldots, v_6$. Given two requests r_1

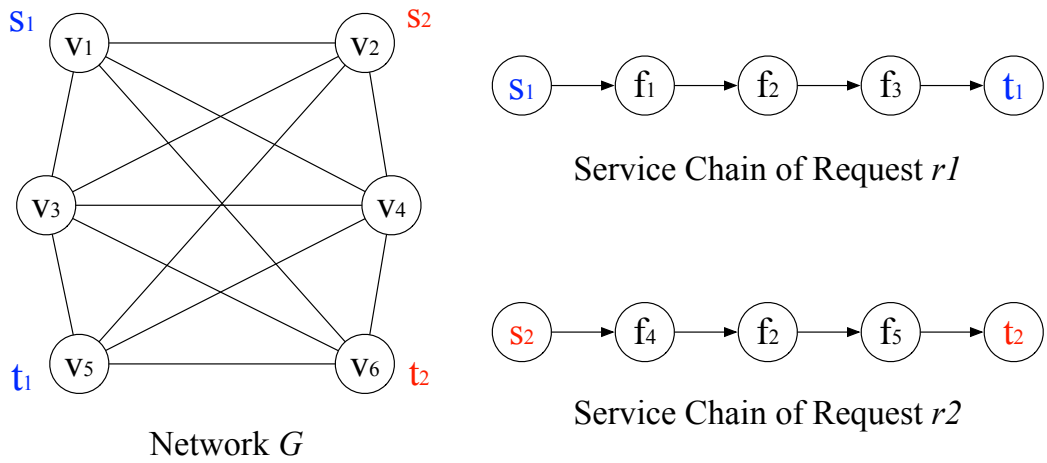


Figure 4.1: An example network G with six nodes and two requests r_1 and r_2

and r_2 with each having its end-to-end delay requirement, request r_1 requires data traffic from its source s_1 (co-located with v_1) to its destination t_1 (co-located with v_5) to traverse network functions f_1, f_2, and f_3, while request r_2 requires data traffic from its source s_2 (co-located with v_2) to its destination t_2 (co-located with v_6) to traverse network functions f_4, f_2, and f_5. Suppose that each node in G is co-located with a

server (or a server cluster) that can accommodate VNF instances. Assume that request r_1 is admitted first, a VNF instance of network function f_2 is instantiated in v_3 for the request, because v_3 is close to the source and destination of request r_1. As a result, when admitting request r_2 that also requires network function f_2, the network service operator can either (a) let requests r_1 and r_2 share the existing VNF instance of f_2 in node v_3, or (b) migrate the existing VNF instance from node v_3 to another location closer to the source and destination of request r_2, or (c) instantiate a new VNF instance of f_2 in node v_4. Apparently, (a) is feasible only if the data traffic routing of request r_2 through v_3 can satisfy its end-to-end delay requirement and the VNF instance of f_2 in v_3 can handle the data traffic of both requests r_1 and r_2 simultaneously; (b) is applicable only if the end-to-end delay requirement of request r_1 will not be violated after migrating the VNF instance serving r_1 from v_3 to v_4; and (c) should be applied if the first two options are infeasible, because it creates an additional VNF instance and may incur a higher cost.

In this chapter, we focus on network throughput maximization by dynamically admitting as many as NFV-enabled requests with end-to-end delay requirements into a network while minimizing the operational cost of the network service provider through joint considerations of VNF instance horizontal and vertical scalings. This problem poses two major challenges. One is to how to admit a new request to meet its service chain, and for each VNF in its service chain, there are three options: sharing existing VNF instances, vertical scaling, and horizontal scaling, which option should be adopted? When different user requests demand the same type of network functions, these requests can be admitted by sharing the same VNF instance in the same server if that instance is able to handle the data traffic of all the requests. Despite lower the operational cost by sharing existing VNF instances among requests, sometimes vertical and horizontal scalings are necessary to avoid the violation of end-to-end delay requirements of some of the requests. In vertical scaling, each user request can be satisfied by instantiating a new instance for each of its network functions individually.

However, vertical scaling may incur a high operational cost due to higher resource consumption than needed. On the other hand, in horizontal scaling, user requests can be admitted by migrating existing VNF instances from their current locations to new locations, incurring a one-time scaling cost. Another challenge is how to perform dynamic admissions of incoming requests in a non-disruptive way. When performing either horizontal or vertical scaling, we must ensure that both resource demands and end-to-end delay requirements of all currently executing requests are not violated, yet different admission strategies of admitting incoming requests heavily impact on the currently executing requests. Allowing arriving requests to share the same VNF instance with currently executing requirements will not affect the end-to-end delay experienced by executing requests, provided that the total data traffic of requests sharing the same VNF instance does not exceed the processing capacity of the VNF instance. In comparison, horizontal scaling has a significant impact on currently executing requests and arriving requests, as the VNF instances used by executing requests will be migrated from their current locations to new locations and data traffic of executing requests will be routed via different paths. On the other end of the spectrum, vertical scaling has minimum influence on currently executing requests, because arriving requests are admitted by instantiating new VNF instances in vertical scaling.

In this chapter, we will address the aforementioned challenges. To the best of our knowledge, we are the first to provide a unified framework for dynamic NFV-enabled request admissions by jointly performing both vertical and horizontal scalings to maximize the number of requests admitted while minimizing the total operational cost. We also devise an efficient algorithm for the problem.

The rest of the section is organized as follows. Section 4.1 will introduce the system model, notions and notations, and the problem definition. Section 4.2 will formulate an Integer Linear Programming solution for the problem. Section 4.3 will propose an efficient algorithm for the problem. Section 4.4 will conduct a performance evaluation

of the proposed algorithm, and Section 4.5 will conclude the section.

4.1 Preliminaries

In this section, we first introduce the system model, notations and notions used in this chapter. We then define the problem formally.

4.1.1 System model

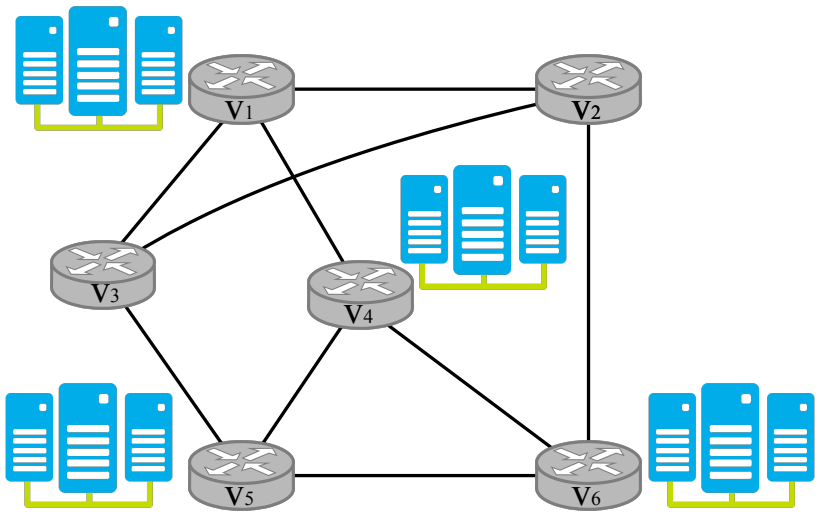


Figure 4.2: An example network

The service provider network is represented by an undirected graph $G = (V, E)$, where V is the set of switch nodes and E is the set of links that interconnect the nodes in V. A subset of switch nodes V_S ($\subseteq V$) is attached with servers that can implement network functions as VNF instances. Each server attached to switch $v \in V_S$ has computing capacity cap_v, and the connection between the server and its attached switch is by a high-speed optical cable, their communication bandwidth and communication delay usually are negligible. Each link $e = (u, v) \in E$ between two switch nodes has a transmission delay D_e ($= D_{u,v}$). We assume that switches in G are connected via high-capacity optical links so that the bandwidth capacity of the link is not a bottleneck. Figure 4.2 demonstrates a service provider network G with a set of switch

nodes $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and a subset of switch nodes $V_S = \{v_1, v_4, v_5, v_6\}$ that are attached with servers.

4.1.2 Virtualized network functions

The operator of network G offers L types of VNFs to serve different users. Let $\mathcal{F} = \{f_1, f_2, \ldots, f_L\}$ be the set of VNFs and let c_i be the computing resource requirement of VNF $f_i \in \mathcal{F}$. An instance of network function f_i can be instantiated in a switch node $v \in V_S$ if the server attached to v has sufficient computing resource to accommodate the instance, i.e., its residual computing resource is no less than the computing resource demand c_i by a VNF instance of f_i. For a given network function f_i, there may exist multiple its VNF instances. We thus distinguish different instances of the same network function by denoting the jth instance of f_i as I_{i_j}. A VNF instance may be used to serve multiple user requests at the same time, yet *the maximum data packet processing rate* m_i that each instance of a network function f_i can process is given in advance. We assume that each VNF instance can serve at least one request. If existing VNF instances of network function f_i are inadequate to handle request demands, its additional VNF instances can be instantiated, and each newly instantiated one will incur a setup cost of C_i^{setup} with $1 \leq i \leq L$. Denote by $C_{ser}(i)$ the service cost of a VNF instance of network function f_i per time slot.

4.1.3 User requests

Each *NFV-enabled user request* r_k consists of a traffic routing request from its source to its destination, a service chain, and the end-to-end delay requirement, where each data packet traffic must pass through each network function in the service chain, while the end-to-end delay of its routing path is no greater than its specified one. Formally speaking, each request r_k is defined as a quintuple $(s_k, t_k, b_k, d_k, SC_k)$, where s_k and t_k are the source and destination nodes, respectively, b_k is the request packet rate, d_k is the end-to-end delay requirement, and SC_k is the service chain of the request.

Specifically, SC_k is a sequence of ℓ_k (≥ 1) network functions that request r_k requires, i.e., $SC_k = \langle f_{k_1}, f_{k_2}, \ldots, f_{k_{\ell_k}} \rangle$, where $f_{k_j} \in \mathcal{F}$ is the jth VNF in the service chain of request r_k. In other words, each data packet traffic of request r_k must pass through the VNF instances in SC_k in their specified order. Assuming that the data traffic of r_k is routed via a routing path $P_k = (v_1 = s_k, v_2, \ldots, v_n = t_k)$ in G between s_k and t_k, the end-to-end delay $D(P_k)$ on the routing path experienced by r_k is

$$D(P_k) = \sum_{i=1}^{n-1} D_{v_i, v_{i+1}}, \tag{4.1}$$

where $D_{v_i, v_{i+1}}$ is the transmission delay on link $(v_i, v_{i+1}) \in E$. We focus on network transmission delays experienced by requests because they are the dominating factor.

In order to admit an NFV-enabled request r_k, a routing path P_k in G between source s_k and destination t_k needs to be identified, such that (i) path P_k must traverse VNF instances of the specified sequence of network functions in $SC_k = \langle f_{k_1}, f_{k_2}, \ldots, f_{k_{\ell k}} \rangle$; and (ii) the end-to-end transmission delay $D(P_k)$ according to Equation (4.1) is no greater than its delay requirement d_k.

4.1.4 Dynamic admissions of user requests

We assume that time is equally slotted into *time slots*, and these time slots are numbered as $1, 2, \ldots, T$, where T is the length of a monitoring period. When a request r_k arrives, it will be either admitted or rejected in the beginning of the next time slot of its arrival.

Denote by $\mathcal{R}(t)$ the set of newly arrived requests in the beginning of time slot $t \in \{1, 2, \ldots, T\}$. Due to limited network resources, the network service provider of G may only accommodate a subset of requests in $\mathcal{R}(t)$ by allocating demanded network resources to them. Once a request is admitted, its data traffic will occupy or share the allocated resources it demanded for a certain amount of time. When its data traffic finishes, all resources occupied by its data traffic will be released back to the network for the admissions of other requests. Accordingly, let $\mathcal{A}(t)$ be the set

of admitted requests that have not departed from the system in the beginning of time slot t with $\mathcal{A}(0) = \emptyset$. In other words, $\mathcal{A}(t)$ is the set of admitted requests that still occupy the system resources at least for time slot t. Similarly, for each network function $f_i \in \mathcal{F}$, denote by $\mathcal{I}_i(t)$ the set of VNF instances of f_i in the end of time slot $t-1$ with $\mathcal{I}_i(0) = \emptyset$.

As requests are dynamically admitted or departed, the amounts of residual resources in the beginning of different time slots are different. Let $RE_v(t)$ be the residual computing capacity of the server attached to $v \in V_S$ in the beginning of time slot t with $RE_v(0) = cap_v$.

4.1.5 Vertical and horizontal scalings

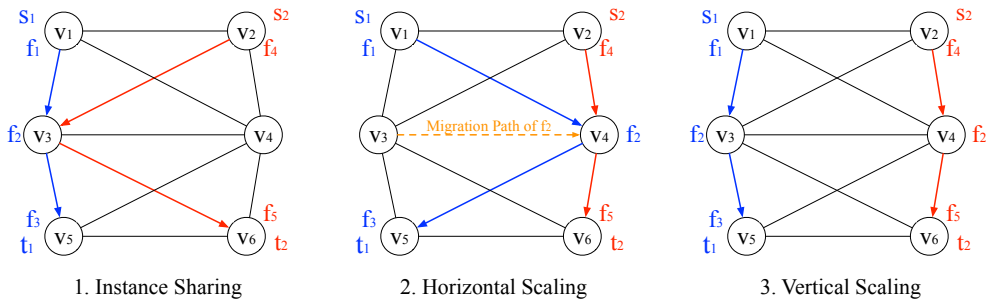


Figure 4.3: Different options for admitting r_2 after request r_1 has been admitted in network G in Figure 4.1

The deployment of VNF instances of a specific network function typically is based on information available at the time of its deployment. For instance, the initial deployment of VNF instances can be based on historical traffic data or previous requests that requested which types of network functions, the request data packet rates, their QoS requirements, and so on. When new requests arrive, either vertical scaling or horizontal scaling should be performed. However, it is worth noting that resource and delay requirements of existing user requests must not be violated, regardless of what scaling should be performed for a particular network function.

In the beginning of each time slot t, assume that some of VNF instances have been instantiated for request admissions in previous time slots. Some of the VNF

instances can be shared to admit newly arrived requests in $\mathcal{R}(t)$, provided that (i) the resource requirements of existing admitted requests in $\mathcal{A}(t)$ are not violated; and (ii) the computing capacities of existing VNF instances are not violated. If either of the two conditions is not met, additional VNF instances need to be instantiated or existing VNF instances need to be migrated to other locations to meet the requirements of both currently executing requests and newly admitted requests.

Horizontal scaling of VNF instances is applicable if existing VNF instances can be migrated to new locations such that the resource requirements of both newly admitted requests in $\mathcal{R}(t)$ and currently executing requests in $\mathcal{A}(t)$ can be satisfied simultaneously.

Denote by $\mathcal{A}_{i,j}(t)$ ($\subseteq \mathcal{A}(t)$) the set of requests processed by the VNF instance I_{i_j} in the beginning of time slot t. As the location of a VNF instance I_{i_j} may change at different time slots, denote by $l_{i,j}(t)$ ($\in V_S$) the location of instance I_{i_j} during time slot t. The VNF instance I_{i_j} may be migrated from $l_{i,j}(t-1)$ in the beginning of time slot t. *The cost of migrating an existing VNF instance* I_{i_j} from its current location $v_a^{i,j}$ to its destination location $v_b^{i,j}$ through a given path $P^{i,j} = (v_a^{i,j} = v_0^{i,j}, v_1^{i,j}, \ldots, v_n^{i,j} = v_b^{i,j})$ is

$$C_{i,j}^{mig}(P^{i,j}) = \sum_{r_k \in \mathcal{A}_{i,j}(t)} b_k \cdot |P^{i,j}|, \tag{4.2}$$

where b_k is the packet rate (the bandwidth demand) of request r_k and $|P^{i,j}|$ is the number of links on path $P^{i,j}$, while $P^{i,j}$ usually is the shortest path in G from node v_i to node v_j with $1 \leq i, j \leq |V|$.

Notice that if VNF instance I_{i_j} is migrated from location $l_{i,j}(t)$ to location $l_{i,j}(t+1)$, the end-to-end transmission delays experienced by requests in $\mathcal{A}_{i,j}(t)$ may change because these requests need to be served by the VNF instance I_{i_j} in location $l_{i,j}(t+1)$, instead of location $l_{i,j}(t)$. Consequently, the traffic of these executing requests needs to be re-routed through different paths. To avoid any service disruption to currently executing requests in $\mathcal{A}_{i,j}(t)$, the migration of every VNF instance I_{i_j} should be performed only if none of the delay requirement of all admitted requests in it will be

violated. However, sometimes due to the network characteristics or user requests, horizontal scaling cannot satisfy newly arrived requests that demand network function f_i, because it will violate the delay requirements of admitted requests. As a result, new instances of f_i must be instantiated at the expense of *the setup cost* of C_i^{setup}.

Take the two requests r_1 and r_2 in Figure 4.1 for instance. Figure 4.3 demonstrates different strategies for the admission of request r_2 after r_1 has been admitted: (1) instance sharing, i.e., r_1 and r_2 share the instance of f_2 in node v_3; (2) horizontal scaling, i.e., migrating the instance of f_2 from v_3 to v_4 in order to serve both requests r_1 and r_2; and (3) vertical scaling, i.e., instantiating a new instance of f_2 in node v_4 to serve request r_2 only.

4.1.6 The operational cost

The network service provider of G is interested in minimizing the network operational cost to maximize its profit. Recall that $\mathcal{I}_i(t)$ is the set of VNF instances of network function $f_i \in \mathcal{F}$ at the end of time slot t. Accordingly, $\mathcal{I}_i(t-1)$ is the set of VNF instances in network G at the beginning of time slot t. The network operational cost at each time slot t consists of the following two costs.

Service cost Each VNF instance I_{i_j} in $\mathcal{I}_i(t)$ will have a service cost of $C_{ser}(i)$. The total service cost $SC(t)$ in time slot t thus is the sum of service costs of all VNF instances, i.e.,

$$SC(t) = \sum_{f_i \in \mathcal{F}} \sum_{I_{i_j} \in \mathcal{I}_i(t)} C_{ser}(i). \tag{4.3}$$

Scaling cost Each VNF instance $I_{i_j} \in \mathcal{I}_i(t)$ is also associated with a scaling cost, which is defined as follows.

$$C_{scal}(I_{i_j}, t) = \begin{cases} 0, & \text{if } I_{i_j} \in \mathcal{I}_i(t-1) \text{ and } l_{i,j}(t) = l_{i,j}(t-1) \\ C_{i,j}^{mig}(P^{i,j}), & \text{if } I_{i_j} \in \mathcal{I}_i(t-1) \text{ and } l_{i,j}(t) \neq l_{i,j}(t-1) \\ C_i^{setup}, & \text{if } I_{i_j} \notin \mathcal{I}_i(t-1) \end{cases} \tag{4.4}$$

where $l_{i,j}(t)$ is the location of VNF instance I_{i_j} at the end of time slot t, Case 1 in Equation (4.4) corresponds to the case in which no scaling is applied to instance I_{i_j}, i.e., instance I_{i_j} already exists in previous time slot $t-1$ and the location of I_{i_j} does not change; Case 2 in Equation (4.4) corresponds to the case in which horizontal scaling is applied to instance I_{i_j}, i.e., instance I_{i_j} already exists in previous time slot $t-1$ yet its location changes; and Case 3 in Equation (4.4) corresponds to the case in which vertical scaling is applied to instance I_{i_j}, i.e., instance I_{i_j} does not exist in previous time slot $t-1$.

The total scaling cost of all VNF instances in network G for time slot t is

$$C_{scal}(t) = \sum_{f_i \in \mathcal{F}} \sum_{I_{i_j} \in \mathcal{I}_i(t)} C_{scal}(I_{i_j}, t). \tag{4.5}$$

The total operational cost of network G in time slot t is

$$TC(t) = SC(t) + C_{scal}(t). \tag{4.6}$$

Table 4.1 lists the notations used in this chapter.

4.1.7 Problem definition

Given the defined service provider network $G = (V, E)$, a time slot $t \in \{1, 2, \ldots, T\}$, in which there are a set of executing requests $\mathcal{A}(t)$ and a set of VNF instances $\mathcal{I}_i(t-1)$ of network function $f_i \in \mathcal{F}$ at the end of time slot $t-1$, and a set of newly arrived NFV-

Table 4.1: Table of Symbols

Notations	Descriptions
$G = (V, E)$	the service provider network
V_S $(\subseteq (V))$	the subset of switch nodes with servers attached
cap_v	the computing capacity of server attached to $v \in V_S$
D_e	the transmission delay of link $e \in E$
$\mathcal{F}$ $(= \{f_1, f_2, \ldots, f_L\})$	the set of network functions
c_i	the computing resource demand of VNF $f_i \in \mathcal{F}$
$\mathcal{I}_i(t)$	the set of VNF instances of f_i in the end of time slot $t-1$
I_{i_j} $(\in \mathcal{I}_i(t))$	the jth instance of f_i
m_i	the maximum data packet processing rate that each instance of a network function f_i
C_i^{setup}	the setup cost of a new instance of f_i
$C_{ser}(i)$	the service cost of a VNF instance network function f_i per time slot
$C_{scal}(I_{i_j}, t)$	the scaling cost of all NFV instances in I_{i_j} in time slot t
r_k $(= (s_k, t_k, b_k, d_k, SC_k))$	a user request with resource s_k, destination t_k, request packet rate b_k, end-to-end delay requirement d_k, and service chain SC_k
$\mathcal{R}(t)$	the set of newly arrived requests in the beginning of time slot t
$\mathcal{A}(t)$	the set of admitted requests that have not departed from the system in the beginning of time slot t
$l_{i,j}(t)$ $(\in V_S)$	the location of instance I_{i_j} during time slot t
$C_{i,j}^{mig}(P^{i,j})$	the cost of migrating an existing VNF instance I_{i_j} via a given path $P^{i,j}$

enabled requests $\mathcal{R}(t)$ with end-to-end delay requirements, the *network throughput maximization problem via VNF instance scalings* in G is to admit as many as requests in $\mathcal{R}(t)$ by determining the set $\mathcal{I}_i(t)$ of VNF instances and their locations for every network function $f_i \in \mathcal{F}$, and the assignment of each admitted request to one or multiple VNF instances, while minimizing the total operational cost of the network service provider, subject to computing capacities on servers in G.

4.1.8 NP-hardness of the defined problem

Lemma 4.1. *The network throughput maximization problem via VNF instance scalings in G is NP-hard.*

Proof. We show NP-hardness of the problem by a reduction from a special case of the Generalized Assignment Problem (GAP). Since this special case of the GAP is NP-hard [25], the network throughput maximization problem via VNF instance scalings is NP-hard as well.

Given a GAP instance consisting of a set $\mathcal{B}$ of bins, a set $\mathcal{I}$ of items, bin capacities $cap : \mathcal{B} \mapsto \mathbb{R}^+$, and the size of placing item $i \in \mathcal{I}$ in bin $b \in \mathcal{B}$: $size(i, b)$, the GAP is to assign a maximum subset $U \subseteq \mathcal{I}$ of items to the bins in $\mathcal{B}$ such that the capacity of each bin is not violated.

We can generate an instance of the network throughput maximization problem via VNF instance scalings as follows. We first construct network $G = (V, E)$. We create a node i for each item i in $\mathcal{I}$, and a node b for each bin b in $\mathcal{B}$. We also add a virtual node s that serves as the destination of all requests. In order to construct the edge set, we create a link between each node i and each node b, and a link from b to node s. That is, $V = \mathcal{I} \cup \mathcal{B} \cup \{s\}$ and $E = \{(i, b) \mid i \in \mathcal{I}, b \in \mathcal{B}\} \cup \{(b, s) \mid b \in \mathcal{B}\}$.

We then generate a set of requests $\mathcal{R}(t)$: For each item $i \in \mathcal{I}$, we add to $R(t)$ a request $r_k = \langle s_k, t_k, b_k, d_i, SC_k \rangle$, where s_k is set to the node $i \in V$, t_k is set to node s, the traffic rate b_k is $size(i, m)$, and $d_i = \infty$. Therefore, routing the set of requests $\mathcal{R}(t)$ into network G is an instance of the network throughput maximization via VNF instance scalings problem. □

4.2 Integer Linear Programming

In this section, we formulate the network throughput maximization problem via VNF instance scalings as an Integer Linear Program (ILP). For brevity, denote by $N(v)$ the set of neighbors of node v in G. The ILP includes the following decision variables.

$H_{i_j}^{u,v}$ is a decision variable of value 1 if VNF instance I_{i_j} is migrated from node u to node v and value 0 otherwise. $M_{i_j}^{u,v}(r, s)$ is a decision variable that is 1 if the migration of VNF instance I_{i_j} from node u to node v is routed through edge (r, s), and 0 otherwise.

V_j^v is a decision variable that has value 1 if an additional instance of VNF f_j is instantiated in node v.

$Y_{i,k}^v$ is a variable that has value 1 if the k-th network function required by request r_i is satisfied by an instance in switch node v and value 0 otherwise.

$$\textit{maximize} \quad \sum_{1 \le i \le |\mathcal{A}(t)|} X_i, \tag{4.7}$$

subject to

$$C^H_{total} = \sum_{1 \le i \le |\mathcal{A}(t) \cup \mathcal{R}(t)|} \sum_{1 \le j \le L} \sum_{u,v \in V_S, u \ne v} \sum_{r,s \in V} M^{u,v}_{i,j}(r,s) \tag{4.8}$$

$$C^V_{total} = \sum_{1 \le j \le L} \sum_{v \in V_S} C^{setup}_j \cdot V^v_j \tag{4.9}$$

$$C^H_{total} + C^V_{total} + \sum_{f_i \in \mathcal{F}} \sum_{I_{i_j} \in \mathcal{I}_i(t)} C_{ser}(i) \le B \tag{4.10}$$

$$Y^v_{i,k} \le V^v_j + \sum_u H^{u,v}_{i_j} \qquad \forall 1 \le i \le |\mathcal{A}(t) \cup \mathcal{R}(t)|, \ 1 \le j \le L, \ 1 \le k \le l_i, \ v \in V \tag{4.11}$$

$$\sum_{u \in N(v)} (W^k_i(u,v) - W^k_i(v,u)) = Y^v_{i,k}, \qquad \forall 1 \le i \le |\mathcal{A}(t) \cup \mathcal{R}(t)|, \ 1 \le k \le l_i, \ SC_{i_k} = f_j, \ v \in V \tag{4.12}$$

$$\sum_{u \in N(v)} (W^{k+1}_i(v,u) - W^{k+1}_i(u,v)) = Y^v_{i,k}, \qquad \forall 1 \le i \le |\mathcal{A}(t) \cup \mathcal{R}(t)|, \ 1 \le k \le l_i, \ SC_{i_k} = f_j, \ v \in V \tag{4.13}$$

$$\sum_{u \in N(s_i)} W^0_i(u, s_i) = 0, \qquad \forall 1 \le i \le |\mathcal{A}(t) \cup \mathcal{R}(t)| \tag{4.14}$$

$$\sum_{u \in N(t_i)} W^{l_i}_i(t_i, u) = 0, \qquad \forall 1 \le i \le |\mathcal{A}(t) \cup \mathcal{R}(t)|, \ v \in V \tag{4.15}$$

$$\sum_{u \in N(s_i)} W^0_i(s_i, u) = X_i - Y^{s_i}_{i,1}, \qquad \forall 1 \le i \le |\mathcal{A}(t) \cup \mathcal{R}(t)| \tag{4.16}$$

$$\sum_{u \in N(t_i)} W^{l_i}_i(u, t_i) = X_i - Y^{t_i}_{i,k}, \qquad \forall 1 \le i \le |\mathcal{A}(t) \cup \mathcal{R}(t)| \tag{4.17}$$

$$\sum_{u \in N(s_i)} W^{l_i}_i(s_i, u) = Y^{s_i}_{i,K-1}, \qquad \forall 1 \le i \le |\mathcal{A}(t) \cup \mathcal{R}(t)| \tag{4.18}$$

$$\sum_{u,v \in V} \sum_{0 \le j \le l_i} (W^j_i(u,v) \cdot D_{(u,v)}) \le d_i, \qquad \forall 1 \le i \le |\mathcal{A}(t) \cup \mathcal{R}(t)| \tag{4.19}$$

$$\sum_{r_i \in \mathcal{R}(t) \cup \mathcal{A}(t)} c_i \cdot Y^v_{i,k} \le cap_v(t), \qquad \forall v \in V \tag{4.20}$$

$$\sum_{r_i \in \mathcal{R}(t) \cup \mathcal{A}(t)} b_i \cdot Y^v_{i,k} \le m_i, \qquad \forall v \in V \tag{4.21}$$

$$\sum_{w \in N(v)} M^{u,v}_{i_j}(w,v) - \sum_{w \in N(v)} M^{u,v}_{i_j} j(v,w) = 0, \qquad \forall I_{i_j} \in \mathcal{I}_i(t), 1 \le j \le L \tag{4.22}$$

$$\sum_{w \in N(u)} M^{u,v}_{i_j} j(w,v) = H^{u,v}_{i_j}, \qquad \forall I_{i_j} \in \mathcal{I}_i(t), 1 \le j \le L \tag{4.23}$$

$$\sum_{w \in N(v)} M^{u,v}_{i_j} j(u,w) = H^{u,v}_{i_j}, \qquad \forall I_{i_j} \in \mathcal{I}_i(t), 1 \le j \le L \tag{4.24}$$

$$X_i = 1 \qquad \forall r_i \in \mathcal{A}(t) \tag{4.25}$$

$$X_i \in \{0,1\} \qquad \forall r_i \in \mathcal{R}(t) \tag{4.26}$$

$$Y^v_{i,k} \in \{0,1\} \qquad \forall 1 \le i \le |\mathcal{A}(t) \cup \mathcal{R}(t)|, \ 1 \le k \le L, \ v \in V \tag{4.27}$$

$$V^v_j \in \{0,1\} \qquad \forall 1 \le j \le L, \ v \in V_S \tag{4.28}$$

$$V^v_j = 0 \qquad \forall 1 \le j \le L, \ v \in V \setminus V_S \tag{4.29}$$

$$H^{u,v}_{i_j} = H^{v,u}_{i_j} = H^{u,u}_{i_j} = 0 \qquad \forall 1 \le j \le L, \ v \in V \setminus V_S, \ u \in V \tag{4.30}$$

$$H^{u,v}_{i_j} \in \{0,1\} \qquad \forall 1 \le j \le L, \ v,u \in V_S, \ v \ne u \tag{4.31}$$

$$W^j_i(u,v) \in \{0,1\}, \qquad \forall e \in E, \ i = 1, \ldots, |\mathcal{R}(t)|, j \tag{4.32}$$

$$M^{u,v}_{i_j}(r,s) \in \{0,1\}, \qquad \forall u,v,r,s \in V \tag{4.33}$$

Figure 4.4: An ILP formulation of the network throughput maximization via VNF instance scaling problem

$W_i^k(u,v)$ is a decision variable that is 1 if edge (u,v) carries the traffic of request r_i after k-th network function in the service chain is applied and before $(k+1)$-th network function is applied; otherwise, the value is 0.

X_i is 1 if request r_i is admitted or 0 otherwise.

Constraint (4.8) calculates the total cost of horizontal scaling according to Equation (4.2). Meanwhile, Constraint (4.9) calculates the total cost of vertical scaling. These calculations reflect with those defined in Equation (4.4).

In order to capture the objective of the problem, i.e., maximizing the network throughput while minimizing the total operational cost of the network operator, we here introduce the concept of a *budget* B on the total operational cost. Specifically, Constraint (4.10) requires that the total operational cost of G, which is calculated according to Equation (4.6), is no greater than a given budget B. The value of B can be initially set to a large value, and a tighter bound of B can be obtained later by binary search iteratively.

Constraint (4.11) enforces that switch v can be used to implement network function f_j required by a request r_i only if a new instance is instantiated in v or an existing instance is migrated to v.

Constraints (4.12) and (4.13) capture traffic changing at switch nodes that accommodate VNF instances that process traffic of requests and traffic conservation at non-destination switches. Specifically, if request r_i is processed at $v \in V_S$, then (i) exactly one incoming edge of v carries the unprocessed traffic and none of the outgoing edges of v carries the unprocessed traffic; and (ii) exactly one of the outgoing edges of v carries the processed traffic, and none of the incoming edges of v carries the processed traffic. Otherwise, if the traffic of r_i is not processed at switch $v \in V_S$ but goes through v, either (i) exactly one incoming edge and one outgoing edge of v carries the unprocessed traffic, or (ii) exactly one incoming edge and one outgoing edge of v carries the processed traffic.

Constraints (4.14) and (4.15) ensure that if the traffic of a given request r_i any

source switch s_i and no traffic leaves the terminal switch t_i. Constraints (4.16) and (4.17) handle the cases where the traffic of a request v_i is processed at the source switch s_i or the terminal switch t_i. Constraint (4.18) handles the cases where VNF instances for request r_i are instantiated in its source node s_i.

Constraint (4.19) enforces the end-to-end delay requirement of each admitted request.

Constraints (4.20) and (4.21) model the computing capacity constraint of each switch $v \in V_{pm}$ and the processing capacity constraint of each VNF instance, respectively.

Constraints (4.22) to (4.24) deal with the migration paths. They ensure that the migration of VNF instance I_{i_j} follows a valid path.

Constraints (4.25) to (4.33) restrict the ranges of decision variables to 0 and 1, if applicable.

The formulated ILP serves for two purposes. One is that it can find an optimal solution to the problem when its size is small; another is that it serves as a benchmark for the performance evaluation of proposed algorithms.

4.3 Heuristic Algorithm

Since the problem is NP-hard and the proposed ILP solution takes prohibitive time when the problem size is large, it is only applicable when the problem size is small. In this section, we devise an efficient, scalable algorithm for the problem. We start by introducing the basic idea behind the proposed algorithm. We then detail the two stages of the proposed algorithm and the description of the algorithm itself. We finally analyze the time complexity of the proposed algorithm.

4.3.1 Overview of the proposed algorithm

The basic idea of the proposed algorithm is described as follows. In order to place VNF instances of different network functions to servers while meeting the end-to-end delay

requirements of NFV-enabled requests, we are not just considering each request alone, we instead take all the VNF instances needed by all requests into consideration. To this end, we use a directed graph to model the relationships between the VNF instances of different network functions among the service chains of all being considered requests. We then prioritize the VNF instance scalings of different network functions. That is, which VNF instance should be considered first. We order the network functions in $\mathcal{F}$, by incorporating VNF dependency in service chains of requests in $\mathcal{R}(t) \cup \mathcal{A}(t)$. We finally perform VNF instance scalings by reducing this subproblem to a series of GAP instances, while each GAP instance will determine which type of scaling should be performed. The algorithm thus consists of two stages: (i) *prioritize the order of network functions in* $\mathcal{F}$; and (ii) *perform VNF instance vertical or horizontal scaling for each ordered network function in* $\mathcal{F}$ one by one. In the following, we deal with these two stages in detail.

4.3.2 VNF ordering

Each request r_k in $\mathcal{R}(t) \cup \mathcal{A}(t)$ has a service chain SC_k. The network functions in the service chain must be applied to each data packet traffic of the request in their specified order. Furthermore, the appearance order of a specific network function in the service chains of different requests is different, which makes prioritizing scaling orders of VNF instances of different network functions become difficult. For example, take four requests in Figure 4.5 for consideration. If we consider scaling network function f_4 for request r_3 first, we may not fully exploit the fact that requests r_1 and r_2 require network function f_6 to be executed before network function f_4, and thus the VNF instances of these two network functions should be placed in reasonably close proximity to each other to meet the delay requirement of requests r_1 and r_2 and reduce their resource consumptions. As a result, considering VNF instance scaling for f_4 first will be not as effective as considering f_4 after f_6.

We thus utilize the constraints imposed by the service chains of requests to order

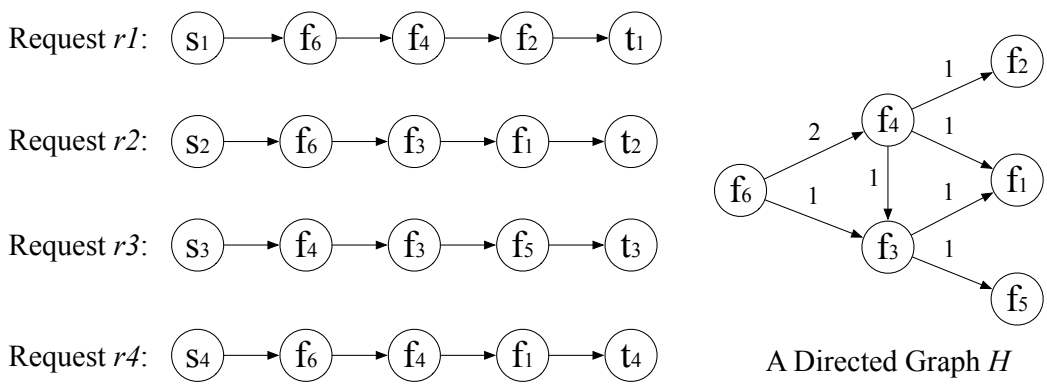


Figure 4.5: The four service chains demanded by four requests r_1, r_2, r_3, and r_4 can be represented by the directed graph H

network functions in $\mathcal{F}$ such that if a network function f_i appears before another network function f_j in the ordering, then the number of requests that require f_j before f_i is minimized. The detailed ordering of network functions is described as follows.

If network function f_i appears immediately before f_j in the service chain of request r_k, then there exists an ordering constraint between f_i and f_j. Let $f_i \prec_{r_k} f_j$ represent the ordering constraint between f_i and f_j in request r_k, e.g., $f_6 \prec_{r_1} f_4$ and $f_4 \prec_{r_3} f_3$ for Figure 4.5. Then, given a set of requests in $\mathcal{R}$, ordering constraints inducted by these requests form a set $\mathcal{O} = \{f_{k_j} \prec_{r_k} f_{k_{j+1}} \mid r_k \in \mathcal{R}, 1 \leq j \leq l_k - 1\}$. For instance, given the service chains of request r_1, r_2, and r_3 in Figure 4.5, the set of ordering constraints is $\{f_6 \prec_{r_1} f_4,\ f_4 \prec_{r_1} f_2,\ f_6 \prec_{r_2} f_3,\ f_3 \prec_{r_2} f_1,\ f_4 \prec_{r_3} f_3,\ f_3 \prec_{r_3} f_5\}$. Having the set of ordering constraints $\mathcal{O}$, we can identify an ordering of network functions. For instance, given three service chains in Figure 4.5 with ordering constraints $\mathcal{O} = \{f_6 \prec_{r_1} f_4,\ f_4 \prec_{r_1} f_2,\ f_6 \prec_{r_2} f_3,\ f_3 \prec_{r_2} f_1,\ f_4 \prec_{r_3} f_3,\ f_3 \prec_{r_3} f_5\}$, one ordering of the network functions in $\mathcal{F}$ is $\langle f_6, f_4, f_3, f_2, f_1, f_5 \rangle$. Notice that there exist many other orderings that satisfy the ordering constraints including $\langle f_6, f_4, f_3, f_2, f_5, f_1 \rangle$ and $\langle f_6, f_4, f_3, f_1, f_2, f_5 \rangle$.

Given a set of arrived requests $\mathcal{R}(t)$ and a set of admitted requests $\mathcal{A}(t)$, and a set of network functions $\mathcal{F}$, a weighted directed graph $H = (N, A; \omega)$ is constructed, where each network function $f_i \in \mathcal{F}$ is represented by a node in N, i.e., $N = \mathcal{F}$.

For each request $r_i \in \mathcal{R}(t) \cup \mathcal{A}(t)$ with service chain $SC_i = \langle f_{i_1}, f_{i_2}, \ldots, f_{i_{\ell_i}} \rangle$ and $1 \leq j \leq \ell_i - 1$, if set A does not contain arc $\langle I_{i_j}, f_{i_{j+1}} \rangle$, one arc $\langle I_{i_j}, f_{i_{j+1}} \rangle$ is added to A and the weight on it is set to 1; otherwise, its weight is incremented by one. As a result, graph $H = (N, A; \omega)$ is a directed graph that may be or may not be a Directed Acyclic Graph (DAG) [26], because it is very likely that for some network functions f_i and f_j, network function f_i appears before network function f_j in the service chain of one request, yet f_i appears after f_j in the service chain of another request, resulting in a directed cycle in H. If such a cycle exists, an ordering of network functions in $\mathcal{F}$ is not achievable. Instead, all directed cycles in H must be removed to make the resulting graph a DAG. It thus is desirable to eliminate all directed cycles in H by removing a minimum weight set of arcs from H to minimize the number of requests whose network function order in their service chains are violated. However, identifying such a minimum weight set of removal arcs from H is NP-hard [29], we have the following lemma.

Lemma 4.2. *Given a set of admitted requests $\mathcal{A}(t)$, a set of arrived requests $\mathcal{R}(t)$, and a set of network functions $\mathcal{F}$, the auxiliary weighted, directed graph $H = (N, A; \omega)$ constructed may contain directed cycles. Eliminating all directed cycles by removing a minimum weight directed set of arcs from A is NP-hard.*

Due to NP-hardness of the problem of concern, we instead remove a set E' ($E' \subset A$) of arcs from H to make the resulting graph become a DAG, and E' can be obtained by applying an approximation algorithm due to Demetrescu *et al.* [29], and the approximate solution E' is bounded by the number of arcs of a longest simple cycle of the directed graph [29].

Denote by $H' = (N, A'; \omega)$ the DAG after the removal of arcs in E' from H. We then perform topological sorting on H'. As a result, each network function (corresponding a node in H) is sorted and all network functions in $\mathcal{F}$ have their sorted order.

The detailed procedure of ordering network functions in $\mathcal{F}$ is given in `Procedure` 4.1.

Procedure 4.1 Ordering network functions in $\mathcal{F}$

Input: a set of network functions $\mathcal{F}$, a set of arrived requests $\mathcal{R}(t)$, and a set of admitted requests $\mathcal{A}(t)$

Output: each network function in $\mathcal{F}$ is ordered at time slot t with $0 < t \leq T$.

1: Construct a directed weighted graph $H = (N, A; \omega)$ with $N = \mathcal{F}$, using the service chains of requests in $\mathcal{R}(t) \cup \mathcal{A}(t)$ and $A = \emptyset$;
2: **for** each request $r_i \in \mathcal{R}(t) \cup \mathcal{A}(t)$ with service chain $SC_i = \langle f_{i_1}, f_{i_2}, \ldots, f_{i_{\ell_i}} \rangle$ **do**
3: **for** $k \leftarrow 1$ to $\ell_i - 1$ **do**
4: **if** A contains arc $\langle f_{i_k}, f_{i_{k+1}} \rangle$ **then**
5: $\omega(\langle f_{i_k}, f_{i_{k+1}} \rangle) \leftarrow \omega(\langle f_{i_k}, f_{i_{k+1}} \rangle) + 1$; /* increment the weight of an existing arc */
6: **else**
7: $A \leftarrow A \cup \{\langle f_{i_k}, f_{i_{k+1}} \rangle\}$; /* add a new arc to A and set its weight to 1 */
8: $\omega(\langle f_{i_k}, f_{i_{k+1}} \rangle) \leftarrow 1$;
9: **end if**
10: **end for**
11: **end for**
12: Find an approximate, minimum weight set E' ($\subset A$), by applying the approximation algorithm due to Demetrescu *et al.* [29] on graph H;
13: A DAG $H' = (N, A'; \omega')$ is obtained, by removing all arcs in E' from H, where $A' = A \setminus E'$;
14: **return** a topological ordering of nodes (or network functions in $\mathcal{F}$) in N;

4.3.3 VNF instance placements and migrations

We then deal with the second stage of the proposed algorithm. Having ordered network functions in $\mathcal{F}$, we now determine which scaling among the two scaling techniques should be applied to the VNF instances of each network function, and we examine the network functions one by one by its topological sorting order.

Let f_i be the network function that is currently being examined. We determine which scaling should be applied to a VNF instance of f_i, by constructing an instance of the Generalized Assignment Problem (GAP) that is defined as follows.

Given a set of items $\mathcal{I}$ and a set of bins $\mathcal{B}$, where each bin $b \in \mathcal{B}$ has a capacity $cap(b)$, each item $i \in \mathcal{I}$ has a size of $size(i, b)$ and a profit $profit(i, b)$ if item i is placed in bin b, the problem is to assign a subset of items U ($\subseteq \mathcal{I}$) to bins $\mathcal{B}$ such that the total profit of the items in U is maximized, while the sum of sizes of placed items in each bin is no more than the capacity of the bin.

We construct an instance of the GAP for the currently considering network function f_i as follows. Each existing VNF instance I_{i_j} of f_i is represented by a bin with

capacity $cap = m_i$. Each bin representing I_{i_j} corresponds to sharing the existing instance I_{i_j} of f_i. For each node $u \in V_S$ with sufficient residual computing capacity to accommodate an instance of f_i, a bin $\mathcal{V}_u^i$ is added to a collection $\mathcal{B}$ of bins, representing instantiating a new VNF instance at the node u. The capacity of each of the bins is $cap(\mathcal{V}_u^i) = m_i$.

For each existing VNF instance I_{i_j} at node u ($\in V_S$) and $v \in V_S \setminus \{u\}$, $\mathcal{H}_{u,v,j}^i$ is added to the set of bins $\mathcal{B}$. Note that migrating an existing VNF instance I_{i_j} of f_i may violate the end-to-end network transmission requirements of currently executing requests. Thus, bin $\mathcal{H}_{u,v,j}^i$ is added only if migrating VNF instance I_{i_j} from node u to node v does not violate the end-to-end network transmission requirements of executing requests in $\mathcal{A}(t)$ that use I_{i_j}. The capacity of bin $\mathcal{H}_{u,v,j}^i$ is $cap(\mathcal{H}_{u,v,j}^i) = m_i$.

Each request r_k in $\mathcal{R}(t) \cup \mathcal{A}(t)$ with SC_k containing f_i is represented as an item r_k. The size $size(r_k, b)$ is the traffic rate b_k of request r_k for all bins $b \in \mathcal{B}$. Notice that GAP is a maximization problem, whereas the objective of the network throughput maximization problem via VNF instance scalings is to admit as many as requests while minimizing the total operational cost. The profit $profit(r_k, b)$ of putting item r_k into bin $b \in \mathcal{B}$ thus is $profit(r_k, b)$, which is defined as follows.

If bin b is of the form I_{i_j}, which represents sharing the existing VNF instance I_{i_j}, then $profit(r_k, I_{i_j})$ is set to 1, because sharing an existing VNF instance incurs little additional costs and it should be highly encouraged. If bin b is of the form $\mathcal{V}_u^i$, which represents instantiating a new VNF instance at u, then $profit(r_k, \mathcal{V}_u^i)$ is the reciprocal of the setup cost of VNF instances for network function f_i, C_i^{setup}, i.e., $profit(r_k, \mathcal{V}_u^i) = 1/C_i^{setup}$. If bin b is of the form $\mathcal{H}_{u,v,j}^i$, which represents migrating the existing VNF instance from u to v, then $profit(r_k, \mathcal{H}_{u,v,j}^i)$ is the reciprocal of the migration cost as given by Equation (4.2).

The rationale behind the construction of a GAP instance for each network function is that it aims to maximize the number of requests admitted while minimizing the total operational cost of the network operator. The profit setting reflects the costs

associated with different scaling options defined in Equation (4.5). Thus, the solution to the GAP instance, in terms of assignments of a subset of items to bins, corresponds to a solution in which as many as requests are admitted and the total operational cost of their admissions is minimized.

Given each network function $f_i \in \mathcal{F}$, its corresponding instance of the GAP can be solved, using an approximation algorithm with a $\frac{1}{2+\epsilon}$-approximation ratio due to Cohen *et al.* [25], where ϵ is a constant with $0 < \epsilon \leq 1$. Let S be the solution delivered by the approximation algorithm, which consists of pairs of admitted requests and bins. If request r_k is assigned to a bin representing sharing an existing VNF instance I_{i_j}, then the request should share the existing VNF instance I_{i_j}; (ii) if the request is placed in a bin $\mathcal{V}_u^i$, which represents a VNF instance instantiation of f_i at node u, then a new VNF instance of f_i should be instantiated at node u; (iii) otherwise, if it is placed in a bin $\mathcal{H}_{u,v,j}^i$, which represents horizontal scaling from u to v, then the VNF instance I_{i_j} in u should be migrated to v. If request r_i will not be assigned to any bin, then the request should be rejected.

4.3.4 Algorithm

The aforementioned process handles the instance placements and migrations of one network function f_i at each time. By repetitively applying the process for every network function in its topological sorting order, we can consider VNF instance scaling of all network functions in $\mathcal{F}$. Note that resource allocations for vertical (VNF instance instantiation) and horizontal scalings (VNF instance migration) are not actually performed on G until all network functions have been examined. Instead, in each iteration of the process, we work on a copy of the original network.

After iteratively examining all network functions, we then calculate the end-to-end network transmission delays of all involved requests. If the end-to-end delay requirement of a request $r_k \in \mathcal{R}(t)$ is not met or the algorithm fails to identify a VNF instance for one or more network functions in its service chain SC_k, request r_k will be

rejected, and its related scalings will not be performed unless those scalings are also demanded by the other requests.

Due to the construction of the GAP instance for each network function f_i, request r_k may be assigned to a bin that represents either an instance sharing, a vertical scaling, or a horizontal scaling. If r_k is assigned to a bin that represents either vertical scaling or horizontal scaling and r_k is the only request that is assigned to the bin, then the scaling corresponds to the bin will not be performed, because r_k should be rejected and no scaling should be applied for the sake of it. Note that the horizontal scaling of existing VNF instances should not violate the end-to-end delay requirements of all admitted requests in $\mathcal{A}(t)$.

To calculate the end-to-end delay of every request r_k in $\mathcal{A}(t)$ experienced at each VNF instance of its service chain, and if this delay is greater than its delay requirement d_k, the violation occurs due to the VNF instance migrations of network functions demanded by request r_k. To resolve this potential violation on the end-to-end delay requirement of request r_k, for each VNF instance that is used by r_k needs to be migrated from its current location u to a new location v, the VNF instance is not migrated to v, and a new VNF instance of the network function is instantiated in v instead.

The detailed algorithm is given in Algorithm 4.1.

4.3.5 Analysis of the proposed algorithm

In the following, we first show that the solution delivered by Algorithm 4.1 is feasible. We then analyze its time complexity.

Theorem 4.1. *Given a network $G = (V, E)$ with a set V of switches and a set E of links, a subset $V_S \subseteq V$ of switches with attached servers to implement VNF instances, a set of network functions $\mathcal{F}$ provided by the network, a set of admitted requests $\mathcal{A}(t)$ that are still being executed in G in the beginning of time slot t, and a set of newly arrived requests $\mathcal{R}(t)$, there is an algorithm, Algorithm 4.1, for the network throughput maximization problem via VNF instance scalings, which delivers a feasible solution in $O(\ell_{\max}(|\mathcal{R}(t)| + |\mathcal{A}(t)|) +$*

Algorithm 4.1 An efficient heuristic for admitting a set of requests $\mathcal{R}(t)$ into G at time slot t

Input: a service provider network $G = (V, E)$, a set of network functions $\mathcal{F}$, a set of user requests $\mathcal{R}(t)$, and a set of admitted requests $\mathcal{A}(t)$ that are still executing in the network

Output: If each request in $\mathcal{R}(t)$ should be admitted and the VNF instances used for each admitted request

1: $\mathcal{R}'(t) \leftarrow \mathcal{R}(t)$; /* the set of requests that have not been marked as non-admitted */
2: **for** each request r_k in $\mathcal{R}'(t)$ **do**
3: Associate r_k with an attribute $r_k.scalings$, which is the set of scalings to be performed for request r_k, and initialize the attribute $r_k.scalings$ to $\emptyset$;
4: **end for**
5: Order network functions in $\mathcal{F}$ by invoking `Procedure` 4.1;
6: **for** each network function f_i in the sorted order **do**
7: Let $\mathcal{R}_i(t)$ be the subset of requests in $\mathcal{R}'(t) \cup \mathcal{A}(t)$ that require network function f_i;
8: Construct an instance of the GAP with the set of items representing requests $\mathcal{R}_i(t)$ and the set of bins representing different scaling options;
9: Let S be an approximate solution to the GAP instance, by invoking the algorithm due to Cohen *et al.* [25];
10: **for** each request r_k in $\mathcal{R}_i(t)$ **do**
11: **if** request r_k is not assigned to a bin in the solution obtained in Step 9 **then**
12: $\mathcal{R}'(t) \leftarrow \mathcal{R}'(t) \setminus \{r_k\}$; /* Mark request r_k as a non-admitted request */
13: **else**
14: Add to $r_k.scalings$ the scaling corresponding to the bin to which request r_k is assigned in the obtained solution;
15: **end if**
16: **end for**
17: **end for**
18: **for each** request $r_k \in \mathcal{R}'(t)$ **do**
19: **if** the end-to-end requirement of r_k calculated according to Equation (4.1) is not met **then**
20: $\mathcal{R}'(t) \leftarrow \mathcal{R}'(t) \setminus \{r_k\}$;
21: **end if**
22: **end for**
23: **return** the set of scalings $r_k.scalings$ to be performed for every request r_k in $\mathcal{R}'(t)$;

$L^3 + L|V|^2 \cdot \frac{|\mathcal{R}(t) \cup \mathcal{A}(t)|^3}{\epsilon}$) *time, where* L *is the number of network functions provided by* G*, i.e.,* $L = |\mathcal{F}|$*,* $\ell_{\max}$ *is the maximum length of service chains of all requests, i.e.,* $\ell_{\max} = \max\{$*the length* ℓ_k *of the service chain of* $r_k \mid r_k \in \mathcal{A}(t) \cup \mathcal{R}(t)\}$*, and* ϵ *is a given constant with* $0 < \epsilon \leq 1$*.*

Proof. The solution delivered by Algorithm 4.1 is feasible because each instance of the GAP is constructed from the subgraph of G that only includes the resources with sufficient residual capacities for request admissions. For instance, a bin that represents vertical scaling at node u is added only if u has sufficient residual computing capacity to accommodate an additional VNF instance. Furthermore, the construction of each GAP instance ensures that the capacity constraints of servers and VNF instances are not violated. Consequently, the scaling and admission decisions based on solving each the GAP instance are feasible.

We now analyze the running time of Algorithm 4.1 as follows. Recall that Algorithm 4.1 consists of two stages: (i) finding an ordering of network functions in $\mathcal{F}$ by constructing an auxiliary directed graph and eliminating all directed cycles in the auxiliary graph; and (ii) constructing an instance of the GAP and finding a solution for each network function in its topological order. in the resulting auxiliary graph.

Stage (i) takes $O(\ell_{\max}(|\mathcal{R}(t)| + |\mathcal{A}(t)|) + L^3)$ time, because the procedure examines the service chain of every request to construct the auxiliary graph $H = (N, A; \omega)$, which results in $O(\ell_{\max}(|\mathcal{R}(t)| + |\mathcal{A}(t)|))$ running time, and removing cycles in H using the algorithm due to Demetrescu *et al.* [29] takes $O(|N| \cdot |A|) = O(L^3)$ time. The running time of Stage (ii) is dominated by the time required to solve a GAP instance for each of L network functions in $\mathcal{F}$. Given $O(|\mathcal{R}(t)| + |\mathcal{A}(t)|)$ items and $O(|V_S|^2) = O(|V|^2)$ bins, the algorithm due to Cohen *et al.* [25] takes $O(|V|^2 \cdot \frac{|\mathcal{R}(t) \cup \mathcal{A}(t)|^3}{\epsilon})$ time for each of L network function. The theorem thus holds. □

4.4 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithm through experimental simulations using both real and synthetic network topologies. We start with the experimental environments, and then evaluate the performance of the proposed algorithm.

4.4.1 Experimental environment

We adopt both real network topologies [83] and synthetic network topologies [10] in the simulations. The real network topologies are adopted from the Internet Zoo Project by Knight *et al.* [83], which is an ongoing initiative to collect data network topologies of different countries. We also adopt the widely used Barabási-Albert model [10], which can generate sythentic networks that follow the well-known network characteristic - scale-free, i.e., the degree distribution in a network typically follows a power law. Both sets of topologies are widely used in evaluating algorithm performance.

The parameter settings are consistent with previous studies, including the number of servers [62], the computing capacity of each server and link delays [135], and types of network functions and their computing resource demands [34, 62]. In the case of a single time slot, we assume that some VNF instances are randomly placed in the network already. Each request $r_i \in \mathcal{R}(t)$ is generated randomly by choosing two nodes in V as source s_i and destination t_i, and assigning its traffic rate b_i and end-to-end delay requirement d_i as per [62]. The generation of the service chain SC_i of each request r_i is consistent with the one in [34, 135]. The default accuracy parameter ϵ in solving the GAP is set at 0.1. The running time is obtained based on a machine with a 3.40GHz Intel i7 Quad-core CPU and 16 GB RAM.

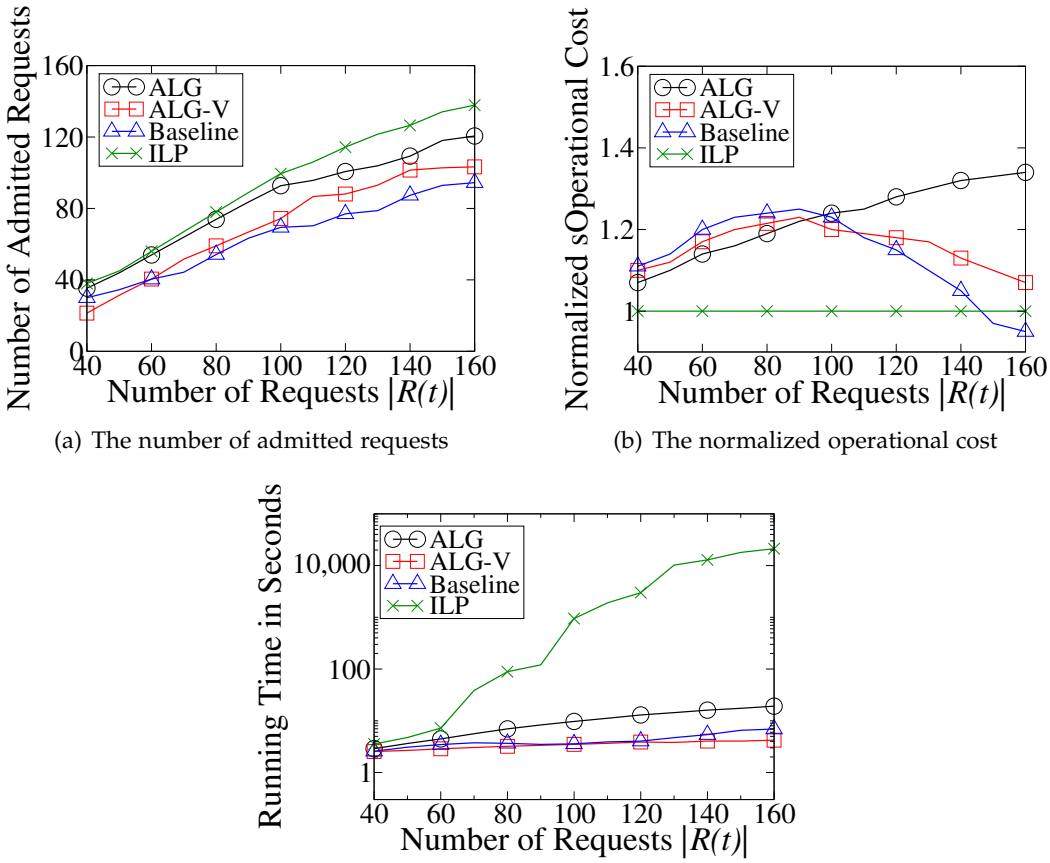
We evaluate the performance of the proposed algorithm against a baseline algorithm that is inspired by the devised algorithms in [33, 101]. The baseline algorithm is described as follows. For every arriving request $r_i \in \mathcal{R}(t)$, the greedy algorithm constructs a multi-stage graph, in which each stage corresponds to a network function

in the service chain SC_i of request r_i, and then the baseline algorithm finds a shortest path in the multi-stage graph from s_i to t_i. Additionally, we evaluate the impact of horizontal scaling by running a variant of Algorithm 4.1 without horizontal scaling, i.e., the algorithm does not migrate any existing VNF instances. We refer to the ILP solution, Algorithm 4.1, Algorithm 4.1 with vertical scaling and without horizontal scaling, and the greedy algorithm as `ILP`, `ALG`, `ALG-V`, and `Baseline` respectively. Each value in figures is the average of the results of 30 trials.

4.4.2 Performance evaluation of different algorithms within a single time slot

We first study the performance of different algorithms at a single time slot.

Figure 4.6 (a) shows the number of requests admitted by different algorithms in a real network, by varying the number of requests. Due to the small problem size, we are able to obtain optimal solutions using `ILP`. It can be seen that algorithm `ALG` achieves near-optimal throughput and performs the best among different algorithms. Specifically, when the number of requests is small, all algorithms perform well by admitting from 70% to 90% of all requests. However, with the increase on the number of requests, only the proposed algorithm `ALG` can achieve high network throughput. Despite the narrow performance gap between algorithms `ALG` and `Baseline` when the number of requests is 40, algorithm `ALG` admits 20% more requests when there are 160 requests. This demonstrates the superiority of the proposed algorithm. Meanwhile, it can be seen from Figure 4.6 (a) that algorithm `ALG-V` outperforms algorithm `Baseline` except when the number of requests is very small. The reason is that there is abundant network resource in the system if the number of reuqests is small and the demanded resource by the requests is small too. Therefore, the impact of VNF instance sharing and scaling is not as significant as when there is a number of requests, and `Baseline` can easily find a path in the constructed multi-stage graph that satisfies the delay requirement.



(a) The number of admitted requests

(b) The normalized operational cost

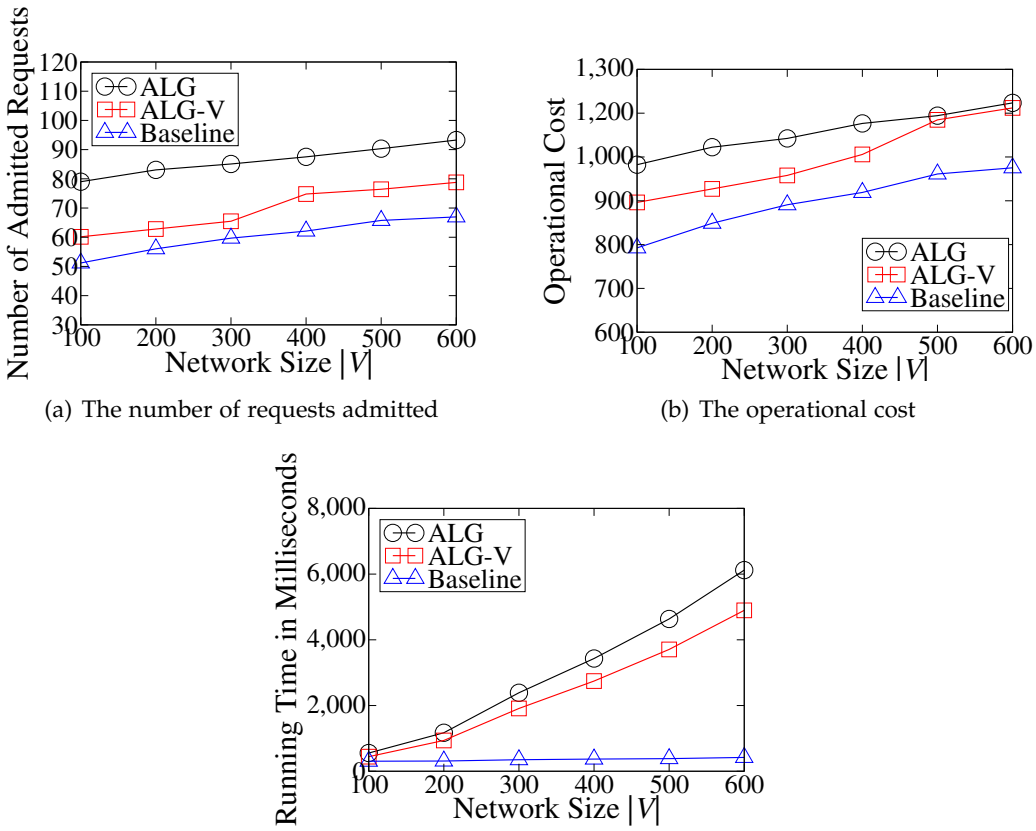
(c) The running time

Figure 4.6: Performance of different algorithms within one time slot, using a real network topology with 40 nodes

Figure 4.6 (b) shows the operational costs of different algorithms for a single time slot. The costs have been normalized with respect to the optimal cost found by the `ILP` for ease of discussion. When the number of requests is less than 100, we notice two trends. First, the operational costs of all algorithms except the `ILP` increase with more requests due to more request admissions. Second, among algorithms `ALG`, `ALG-V`, and `Baseline`, `ALG` has the lowest operational cost, because it constructs instances of the GAP that accurately capture the costs of different options to admit user requests, thereby network resources being efficiently utilized. Moreover, when the number of requests is greater than 100, only the normalized operational cost of algorithm `ALG` increases whereas those of algorithms `ALG-V` and `Baseline` decrease. This is due to different numbers of requests admitted by different algorithms. Although the solution delivered by algorithm `ALG` has a higher operational cost, the algorithm achieves a much higher network throughput than the other two comparison algorithms.

Figure 4.6 (c) illustrates the running times of different algorithms. Meanwhile, algorithm `ALG-V` is an order of magnitude faster than `ALG`, because there are only $O(|V|)$ bins that represent vertical scaling in the GAP instances by algorithm `ALG-V`, whereas algorithm `ALG` has additional $O(|V|^2)$ bins to represent horizontal scaling of VNF instances. More importantly, Figure 4.6 demonstrates that the running time of `ILP` is prohibitively high, thus, the algorithm suffers from poor scalability and is not applicable to the problem with large scale. This necessitates an effective, scalable method that can find near-optimal solutions in a much shorter amount of time, for which algorithm `ALG` is an excellent candidate.

Figure 4.7 (a) plots the curves of numbers of requests admitted by different algorithms at a single time slot, by varying the network size from 100 to 600 while fixing the number of requests at 100. Notice that due to the large network sizes, `ILP` does not finish in a reasonable amount of time, thus the optimal results are not included in these figures. Similar to Figure 4.6 (a), it can be observed that the proposed algorithm `ALG` achieves the highest throughput among all algorithms, and algorithm



(a) The number of requests admitted

(b) The operational cost

(c) The running time

Figure 4.7: Performance of different algorithms within one time slot, using synthetic networks of various sizes

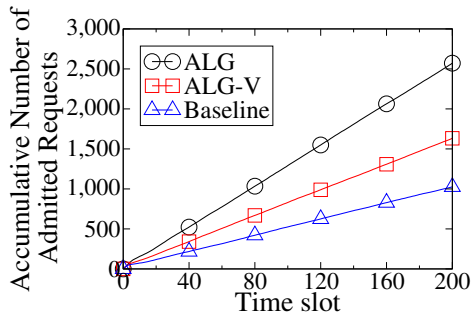
`ALG-V` outperforms algorithm `Baseline` in all network sizes. Figure 4.7 (b) plots the operational costs of different algorithms, from which we notice that the operational cost of an algorithm is highly correlated to the number of requests admitted by it. Take algorithm `ALG` for instance. It has the highest network throughput and the highest operational cost as well among the three algorithms. The rationale behind is that the more requests admitted, the larger the resource consumption needed to meet the resource demands of admitted requests. In addition, Figure 4.7 (c) shows the running times of different algorithms. Both algorithms `ALG` and `ALG-V` have large running times. This can be explained by noticing that during the construction of a GAP instance, algorithm `ALG` adds a bin that represents horizontal scaling of an instance only if the scaling does not violate the resource requirements. When the network size is

large, the possible violation of resource requirements increases, thereby reducing the size of the GAP instance constructed by algorithm `ALG`. Thus, the difference in the sizes of GAP instances constructed by algorithms `ALG` and `ALG-V` is small and these algorithms run in similar amounts of time.

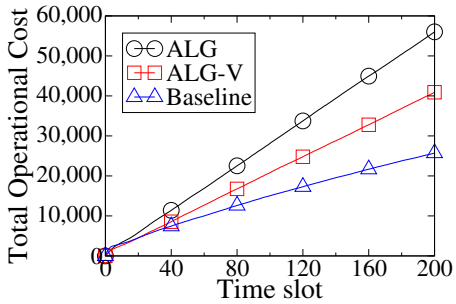
4.4.3 Performance evaluation of different algorithms within a finite time horizon

We then evaluate the performance of the proposed algorithm within a time horizon consisting of 200 time slots. The number of requests at each time slot follows a Poisson distribution with a mean of 30, and each admitted request spans 1 to 10 time slots randomly. In the beginning of each time slot, some executing requests will depart from the network and the allocated resources for them will be released back to the network, before handling newly arrived requests.

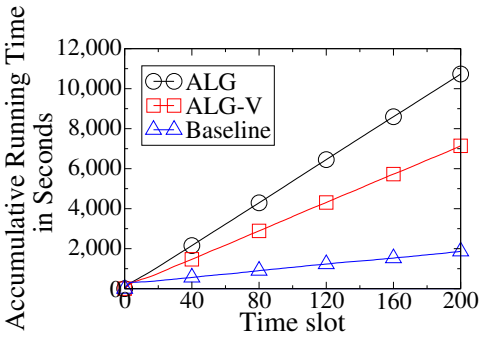
The results are summarized in Figure 4.8. It can be seen from Figure 4.8 (a) that algorithm `Baseline` has the lowest network throughput among all three aforementioned algorithms. On the contrary, algorithms `ALG` and `ALG-V` can admit more requests through effectively scaling VNF instances to meet the resource requirements of requests. The total operational cost of each algorithm in the end of each time slot is shown in Figure 4.8 (b), from which we can see the similar patterns as shown in Figure 4.7 (b). That is, due to the larger number of requests admitted by `ALG`, the operational cost of algorithm `ALG` is higher than the other two algorithms. The differences in operational costs of different algorithms do not mean algorithm `ALG` is inferior, because the network throughput is the main optimization objective, and algorithm `ALG` indeed has the highest network throughput. Meanwhile, Figure 4.8 (c) shows that the running time of algorithm `Baseline` is much smaller in comparison with the ones of algorithms `ALG` and `ALG-V`. It must be reiterated that this running time comes at the expense of admitting much fewer requests. Specifically, although the running time of algorithm `Baseline` is approximately one-fifth of that of algorithm`ALG`, algorithm



(a) The number of requests admitted for a period



(b) The total operational cost



(c) The total running time

Figure 4.8: Performance of different algorithms within a finite time horizon, using a real network topology

`Baseline` only admits half the number of requests as algorithm `ALG` does in the end of the time horizon. The performance difference demonstrates the effectiveness of the proposed algorithm in maximizing the throughput of a network.

4.5 Summary

In this chapter, we studied the network throughput maximization problem, by admitting as many as NFV-enabled requests while meeting their QoS requirements, through jointly considering vertical scaling by instantiating new VNF instances and horizontal scaling by migrating existing VNF instances to new locations. We first formulated the problem as an ILP. We then proposed an efficient heuristic for the problem. We finally evaluated the performance of the proposed algorithm through conducting experiments. Experimental results demonstrated that the proposed algorithm outperforms a baseline algorithm, and the solution quality of the proposed algorithm is on a par with that of the optimal solution delivered by the ILP.

Chapter 5

Reliability-Aware Virtualized Network Function Services Provisioning in Mobile Edge Computing

5.1 Introduction

Mobile devices, including smartphones and tablets, have experienced exponential growth in recent years. This trend is coupled with the evolution of new mobile applications with stringent requirements such as real-time and interactive applications. However, executing computation-intensive applications on mobile devices of portable sizes is heavily constrained by their limited computing, storage, and battery capacities. Mobile edge computing (MEC) has emerged, which brings computation and storage resources, which are referred to as *cloudlets* [117], to the edge of mobile networks [28]. These cloudlets are co-located with access points (APs) in the networks and they are accessible by mobile users via wireless access. A key advantage of deployment of cloudlets is that the close physical proximity between cloudlets and users introduces shorter communication delays, thereby improving the user experience. It thus has been envisioned that MEC has many promising applications including in smart cities and smart connected vehicles [12, 42, 125, 144].

Many network service providers recently have started migrating their infrastructure to take advantage of network functions virtualization (NFV) [67, 103]. Due to the continuous advances in computing hardware, it is possible to replace resource demanding user applications, such as voice recognition, image processing, and other supporting tasks for smart cities, with software components that provide the same capability on top of commodity servers. Each network function, such as a software-based video transcoder for a smart city system, runs in a virtual machine, referred to as a virtualized network function (VNF) instance, hosted in cloudlets. NFV enables users to offload their tasks to nearby cloudlets via wireless APs for processing.

While implementing user applications as VNF instances promises significant flexibility and ease of the management of networks, it also brings more concerns on the reliability of running VNF instances. Traditional carrier-grade systems have been engineered to offer nearly 99.999% (five nines) reliability and designed to be highly fault-tolerant [36]. Achieving this level of reliability is extremely difficult for NFV in an MEC environment due to various reasons including the cloudlets hosting VNF instances are more prone to errors and failures compared to the dedicated hardware appliances, and software for implementing VNF instances may contain bugs [36] and is prone to failures. As a result, in order to provide reliable VNF services to users while meeting their requirement of the service and mitigate the risk of failures, additional VNF instances should be created at other peer cloudlets in the MEC.

In this chapter, we consider reliability-aware VNF services provisioning in an MEC. We assume that each user request needs a VNF service with a specified reliability requirement. In order to meet the reliability requirement of the user, multiple VNF instances need to be placed at different cloudlets. We distinguish between the single primary VNF instance and one or multiple secondary VNF instances for each offloaded user request [50]: the former is an active VNF instance while the latter are idle ones until the primary one fails. Moreover, as the usage of network resources follows pay-as-you-go, there are multiple users competing for limited network resources,

yet the resources at each cloudlet have capacity constraints. Hence, it needs to be determined which requests should be admitted, subject to the capacity constraints on cloudlets.

The novelty of the work in this chapter lies in the provisioning of reliability-aware VNF instances for network function services in MEC, by formulating a novel reliability-aware VNF instances provisioning problem. Efficient approximation and exact algorithms are proposed for allocating network resources to accommodate primary and secondary VNF instances in different cloudlets to meet individual user reliability requirements. Furthermore, the formulated optimization problem may be of independent interests as the Generalized Assignment Problem (GAP) is a special case of the problem. To the best of our knowledge, there is no prior study on similarly formulated problems, and this problem, which generalizes the GAP problem, has wide potential applications in practice, particularly in the fault-tolerance domain.

The main contributions of this chapter are described as follows. We study the provisioning of reliability-aware VNF instances in an MEC to meet individual user reliability requirements. We first formulate a novel optimization problem for provisioning multiple VNF instances at different cloudlets in MEC to meet the reliability requirement of each request. We then show the NP-hardness of the problem and formulate an integer linear programming (ILP) solution to the problem when its size is small, otherwise, we develop an approximation algorithm with an approximation ratio of $O(\log K)$, where K is the number of cloudlets in the MEC. Moreover, we also consider two special cases of the problem. Specifically, we devise a constant approximation algorithm for a special case of the problem where each request needs only one primary and one secondary VNF instances. We also develop a dynamic programming algorithm that delivers an exact solution to another special case of the problem where the VNF instances of different types of virtual network functions demand the same amount of computing resources. We finally evaluate the performance of the proposed algorithms through experimental simulations. Experimental results demonstrate that

the proposed algorithms are promising and the empirical results outperform their analytical counterparts as the theoretical estimations usually are very conservative.

The rest of the chapter is organized as follows. Section 5.2 introduces notions, notations, and the problem definitions. Section 5.3 shows the NP-hardness of the problem. Section 5.4 provides an ILP formulation of the problem, while Section 5.5 devises an algorithm with a logarithmic approximation ratio for the problem. Moreover, Section 5.6 deals with two special cases of the problem and proposes a constant approximation algorithm and an exact solution for the two special cases, respectively. Section 5.7 evaluates the proposed algorithms empirically, and Section 5.8 concludes the chapter.

5.2 Preliminaries

In this section, we first introduce the system model, notions and notations. We then define the problem precisely.

5.2.1 Network model

We consider an MEC environment for a metropolitan region. Given a wireless metropolitan area network (WMAN) $G = (V, E)$ consisting of Access Points (APs) and K cloudlets, where each cloudlet k is co-located with an AP and they are interconnected by a high-speed optical cable, yet not all APs have co-located cloudlets, i.e., the number of cloudlets in the network is far less than the number of APs. Each cloudlet k has a (residual) computing capacity C_k. E is the set of links in which a link $e \in E$ between two APs if they are interconnected by an optical cable. We assume that there are $|\mathcal{F}|$ different types of network function services offered by the network service provider. Denote by $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$ the set of network functions. The computing resource demand of each VNF instance of a network function $f_i \in \mathcal{F}$ is c_i. Mobile users can access the network ubiquitously through their nearby APs. Figure 5.1 is an illustrative example of such a network that consists of five APs and three cloudlets.

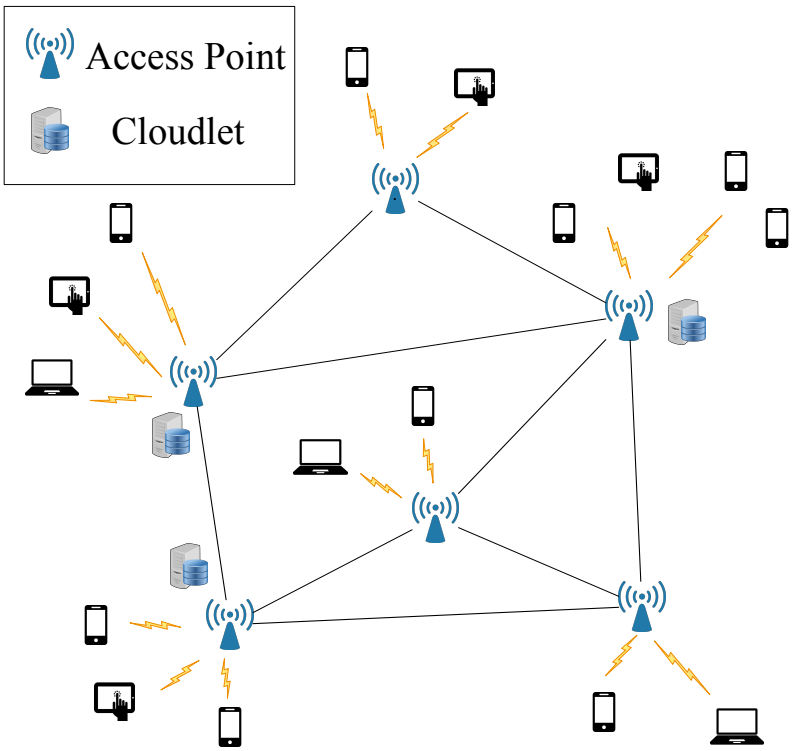


Figure 5.1: An MEC consists of five APs and three cloudlets co-located with APs. User devices access the network through their nearby APs and APs are interconnected by optical links of the MEC.

5.2.2 User requests

Denote by $\mathcal{R}$ the set of user requests. Each user request $r_j \in \mathcal{R}$ is defined by a quadruple (f_j, R_j, c_j, p_j), where f_j ($\in \mathcal{F}$) is the requested type of VNF, R_j is the reliability requirement of request r_j with $0 < R_j \leq 1$, c_j is the amount of computing resource demanded for implementing the VNF instance of f_j in a cloudlet, and p_j ($\in \mathbb{R}^+$) is the revenue collected by the service provider if request r_j is admitted. In contrast to some previous studies, we here assume that a user request can be served by a single VNF instance [74, 140], or a consolidated VNF instance [63]. This assumption is in alignment with real world deployments of VNF instances, such as Google's enterprise software-defined networks (SDN) [8], in which a single general-purpose computer implements multiple network functions, including network address translation (NAT), firewall, Intrusion Detection System (IDS), Dynamic Host Configuration Protocol (DHCP) daemon.

We assume that each request r_j needs $(n_j + 1)$ VNF instances of its network function f_j, to meet its reliable service requirement, where n_j is calculated by the reliability requirement of the request. Without loss of generality, we assume that $n_j + 1 \leq K$, i.e., the reliability provided by the system through duplicating the VNF instances to different cloudlets is no more than the number of cloudlets K in the system. Among the $(n_j + 1)$ VNF instances for request r_j, one of the VNF instances will serve as *its primary VNF instance*, and the remaining n_i VNF instances will serve as *its secondary VNF instances*. Specifically, the value of $n_j \geq 0$ is determined by the reliability requirement R_j of request r_j, which is a minimum non-negative integer to meet the following inequality.

$$\begin{aligned}
&\Pr(\text{at least one of the } (n_j+1) \text{ VNF instances is available}) \\
&= 1 - \Pr(f_j)^{n_j+1} \cdot \prod_{l=0}^{n_j} \Pr(K_{jl}) && (5.1) \\
&= 1 - \Pr(f_j)^{n_j+1} \cdot \Pr(K)^{n_j+1} && (5.2)
\end{aligned}$$

$$\geq R_j, \tag{5.3}$$

where $\Pr(f_i)$ is the failure probability of an instance of network function f_i and $\Pr(K_{j_l})$ is the failure probability of the lth instance of request r_j, and step (5.1) in Equation (5.3) follows because, in order to be functionally identical to the primary instance, the secondary VNF instances are often running on the same software version and using the same configuration as their primary VNF instance, thereby having the same failure probabilities as the failure probability of the primary VNF instance. Step (5.2) in Equation (5.3) follows we assume that cloudlets in the system have the same failure probabilities. This assumption is in alignment with the common industry practice where network operators keep their equipments to a limited range so that they can standardize the process and minimize their OPEX. We also assume that failures of VNF instance are independent with each other and links (optical cables) in G are reliable.

As most network functions are stateful, the primary and secondary VNF instances of a network function need to synchronize with each other so that there is minimum interruptions when the primary VNF instance fails. It is worth noting that the internal state of a VNF instance often contain sensitive information, e.g., encryption keys. If synchronization of the internal states of primary and secondary VNF instances is via the same links as user traffic, sensitive information may be exposed. Instead, the internal state of the primary VNF instance should be periodically distributed to secondary VNF instances over a dedicated out-of-band, i.e., separate from user traffic, network. The commonly used Control Plane Network (CPN) [47, 120] is an excellent candidate for synchronization between the primary and secondary VNF instances of a network function.

5.2.3 Problem definition

Given an MEC $G = (V, E)$, a set $\mathcal{R}$ of requests, associated with each request $r_j \in \mathcal{R}$, there is a payment p_j, a network function $f_j \in \mathcal{F}$, and a reliability requirement represented by the number n_j ($0 \leq n_j < K$) of secondary VNF instances with each of them being placed at a different cloudlet, *the reliability-aware VNF instances provisioning problem* is to admit as many requests in $\mathcal{R}$ as possible such that the total revenue collected by the service provider is maximized, subject to computing capacity constraint on each cloudlet and the specified reliability guarantee of each admitted request.

A request r_j is *admitted* if its $(n_j + 1)$ VNF instances are placed at $(n_j + 1)$ different cloudlets, and each cloudlet has sufficient computing resource to meet the computing resource demand c_j of its network function f_j, assuming that $\max_{1 \leq j \leq |\mathcal{R}|}\{n_j\} \leq K - 1$. There are a set $\mathcal{R}$ of requests, each request $r_j \in \mathcal{R}$ needs $n_j + 1$ replicas of its VNF f_j, and its admission will result in a revenue of p_j where each instance of f_j demands the amount c_{j_i} of computing resource from each of the $n_j + 1$ cloudlets. The problem is to admit as many requests in $\mathcal{R}$ as possible to maximize the revenue, subject to the cloudlet computing capacity constraint.

Notice that in this chapter we do not explicitly consider user mobility. However, we assume the users are allowed to move in the MEC, and they can issue their requests for services through the APs in which they are located.

Table 5.1 lists the frequently used notations used in this chapter.

5.3 NP-Hardness

Theorem 5.1. *The reliability-aware VNF instances provisioning problem in an MEC $G(V, E)$ is NP-complete.*

Proof. We show the claim through a reduction from the well-known NP-complete knapsack problem that is described as follows. Given n items, each item a_i has a weight w_i and a profit p_i with $1 \leq i \leq n$. There is a bin with capacity W. If an item

Table 5.1: Table of Symbols

Notations	Descriptions
$G = (V, E)$	a WMAN
$k \in K$	a cloudlet co-located with an AP
C_k	the residual computing capacity of cloudlet k
$\mathcal{F}$ $(= f_1, f_2, \ldots, f_{\lvert\mathcal{F}\rvert})$	the set of network functions offered in G
f_i	a network functions offered in G
c_i	the computing resource demand of network function f_i
$\mathcal{R}$	the set of user requests
$r_j = (f_j, R_j, c_j, p_j) \in \mathcal{R}$	a user request, where f_j $(\in \mathcal{F})$ is the requested type of VNF, R_j is the reliability requirement with $0 < R_j \leq 1$, c_j is the demanded amount of computing resource, and p_j $(\in \mathbb{R}^+)$ is the revenue
n_j	the number of secondary VNF instances for request r_j to meet its reliable service requirement

a_j can be packed into the bin, it brings a profit of p_j. *The knapsack problem* is to pack as many items as possible such that the total profit of packed items is maximized, subject to the bin capacity W.

Given an instance of the knapsack problem, we construct an instance of the reliability-aware VNF instances provisioning problem as follows. We assume that there are two cloudlets ($K = 2$) with each having W computing capacity. There are n requests, which correspond the n items in the knapsack problem. We further assume that each request r_j has only one secondary VNF instance, i.e., $n_j = 1$. As each request implementation has a primary VNF instance. Thus, there are two VNF instances for each request allocated to two cloudlets. The computing demand of each of the two VNF instances of r_j is w_j. If request r_j is admitted, the revenue brought by its implementation is p_j. Then, the reliability-aware VNF instances provisioning problem is to admit as many requests as possible such that the total revenue collected from admitted requests is maximized, subject the computing capacity constraints on the two cloudlets.

It can be seen that a solution to the latter in turn returns a solution to the former,

while the reduction from an instance of the knapsack problem to an instance of the reliability-aware VNF instances provisioning problem takes polynomial time. Also, verifying the solution of the latter can be easily done in polynomial time. As the knapsack problem is NP-complete, the reliability-aware VNF instances provisioning problem is NP-complete, too. □

5.4 Integer Linear Programming

In this section, we formulate an ILP solution for the problem. We assume that x_j is a boolean variable, where $x_j = 1$ implies that request r_j is admitted; otherwise, it will not be admitted. Furthermore, if r_j is admitted, there are exactly $n_j + 1$ cloudlets with each having the amount c_j of computing resource to meet the computing resource demand by its primary and secondary VNF instances. The optimization objective thus is

$$\textit{maximize} \qquad \sum_{j=1}^{n} p_j x_j,$$

subject to the following constraints.

$$\sum_{j=1}^{|\mathcal{R}|} c_j \cdot y_{j,k} \leq C_k, \quad \forall k, 1 \leq k \leq K \tag{5.4}$$

$$(n_j + 1) \cdot x_j = \sum_{k=1}^{K} y_{j,k}, \quad \forall r_j \in \mathcal{R} \tag{5.5}$$

$$x_j \in \{0, 1\}, \quad \forall r_j \in \mathcal{R} \tag{5.6}$$

$$y_{j,k} \in \{0, 1\}, \quad \forall r_j \in \mathcal{R}, \ \forall k, 1 \leq k \leq K, \tag{5.7}$$

where $y_{j,k}$ is a boolean variable, $y_{j,k} = 1$ if a VNF instance of request r_j is allocated to cloudlet k; $y_{j,k} = 0$ otherwise. Constraint (5.4) says that the total computing demand by the VNF instances of all different requests r_j in any cloudlet k is no greater than its residual computing resource capacity. Constraint (5.5) ensures that the number of secondary VNF instances at different cloudlets for each admitted request r_j is exactly

$n_j + 1$. Constraints (5.6) and (5.7) limit the ranges of boolean variables x_j and $y_{j,k}$ to either 0 or 1.

5.5 Approximation Algorithm for the Reliability-Aware VNF Instances Provisioning Problem

In this section, we propose an approximation algorithm for the reliability-aware VNF instances provisioning problem. We start with cost modeling, and then provide an overview of the proposed algorithm and details of the algorithm. We finally analyze the approximation ratio and time complexity of the proposed algorithm.

5.5.1 Cost modeling

The approximation algorithm examines the requests in $\mathcal{R}$ one by one. When request r_j is considered, the resource availability of cloudlets affects whether r_j can be admitted. We thus denote by $C_k(j)$ the amount of residual computing resource at cloudlet k with $C_k(0) = C_k$.

The key to the approximation algorithm is that we use an exponential function to model the cost $w_k(j)$ of instantiating a VNF instance of request r_j with computing resource demand c_j at cloudlet k, which is defined as follows.

$$w_k(j) = \frac{C_k}{c_j}(\alpha^{1-\frac{C_k(j)}{C_k}} - 1), \tag{5.8}$$

where $\alpha > 1$ is a constant to reflect the sensitivity of the workload at each cloudlet, which will be determined later, and $1 - \frac{C_k(j)}{C_k}$ is *the utilization ratio* of cloudlet k when request r_j is considered. The rationale is that the high utilization a resource has, the higher the risk associated with the resource utilization, thus the usage of resources with high utilization should be discouraged. The proposed exponential function will guide the resource allocations.

5.5.2 Algorithm description

We first sort all requests by the ratio of the payment of each request to the total amount of its computing resource demand to meet its VNF instance reliability. In other words, we rank request r_j ahead of request r_i if $\frac{p_j}{(n_j+1)c_j} \geq \frac{p_i}{(n_i+1)c_i}$ for any two requests $r_i \in \mathcal{R}$ and $r_j \in \mathcal{R}$. For the sake of convenience, we assume that the sorted request sequence is $r_1, r_2, \ldots, r_{|\mathcal{R}|}$. We then examine requests in the sequence one by one to determine whether a request will be admitted immediately.

For each request r_j, we calculate *its normalized cost* if one of its VNF instances is instantiated at cloudlet k as follows.

$$\psi_k(j) = \frac{w_k(j)}{C_k} = \frac{\alpha^{1-\frac{C_k(j)}{C_k}} - 1}{c_j}. \tag{5.9}$$

For each request r_j, we identify top-$(n_j + 1)$ cloudlets with the lowest normalized costs, and denote by $\mathcal{K}_j$ the set of cloudlets. We here introduce *an admission control policy for request admissions*. That is, if the sum of normalized costs of the identified top-$(n_j + 1)$ cloudlets for request r_j is greater than K, i.e.,

$$\frac{\sum_{k \in \mathcal{K}_j} \psi_k(j)}{p_j} > K, \tag{5.10}$$

then request r_j will be rejected. Otherwise, it will be admitted, and its $(n_j + 1)$ VNF instances will be placed in the top-$(n_j + 1)$ cloudlets. The algorithm detail for the reliability-aware VNF instances provisioning problem is given in `Algorithm` 5.1.

5.5.3 Algorithm analysis

In the following, we first provide an upper bound on the sum of costs of all cloudlets in G after a subset of requests in $\mathcal{R}$ is admitted. We then show a lower bound on the sum of costs of cloudlets that an optimal solution uses to admit a request rejected by the proposed algorithm, and we finally analyze the approximation ratio of the proposed approximation algorithm.

Algorithm 5.1 Approximation algorithm for the reliability-aware VNF instances provisioning problem

Input: A set of K cloudlets with each having a residual computing capacity C, a set of requests $\mathcal{R}$ with each request $r_j = (f_j, R_j, c_j, p_j)$

Output: Admit a subset of requests in $\mathcal{R}$ that maximizes the sum of revenues of admitted requests while meeting the reliability requirement of each admitted request.

$\mathcal{A} \leftarrow \emptyset$ /* the set of admitted requests */;
Sort requests in $\mathcal{R}$ by the ratio of the payment of each request to the total amount of its computing resource demand to meet its VNF instance reliability;
for each request r_j in the sorted order **do**
  Calculate the number n_j of secondary VNF instances of its network function by Inequality (5.3);
  Assign each cloudlet k a cost by Equation (5.9) if r_j is admitted;
  Identify top-$(n_j + 1)$ cloudlets $C_{j_1}, C_{j_2}, \ldots, C_{j_{n_j}}, C_{j_{(n_j+1)}}$, with smallest costs, by a linear time selection algorithm;
  if there is a cloudlet j_k among the $n_j + 1$ identified cloudlets such that $C_{j_k} < c_j$ **or** the sum of costs of the $(n_j + 1)$ identified cloudlets is greater than $K \cdot p_j$ (by Inequality (5.10)) **then**
    Reject request r_j;
  else
    Allocate a VNF instance of request r_j to each of the top-$(n_j + 1)$ cloudlets;
    Update the residual computing capacity of each of these cloudlets as $C_{j_k} \leftarrow C_{j_k} - c_j$ for all k with $1 \leq k \leq n_j + 1$;
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{r_j\}$;
  end if
end for
return Set $\mathcal{A}$ of admitted requests and their primary and secondary VNF instance placements in cloudlets.

Lemma 5.1. *Given an MEC $G(V, E)$ and a set of requests $\mathcal{R}$, denote by $\mathcal{A}$ the set of requests admitted by* `Algorithm` *5.1, then, the sum of costs of all K cloudlets is*

$$\sum_{k=1}^{K} w_k(j) \leq 2K \cdot \log \alpha \cdot \sum_{r_{j'} \in \mathcal{A}} p_{j'},$$

where α is a constant with $2K \cdot Q_{\max} + 2 \leq \alpha \leq 2^{C_{\min}/c_{\max}}$, $C_{\min}$ $(= \min\{C_k \mid 1 \leq k \leq K\})$ is the minimum cloudlet computing capacity, $\mathcal{R}$ is the set of requests, and $Q_{\max}$ $(= \max\{p_{j'} \cdot c_{j'} \mid r_{j'} \in \mathcal{R}\})$ is the maximum product of the profit and computing resource demand among requests.

Proof. Consider a request $r_{j'} \in \mathcal{A}$ admitted by the approximation algorithm. Then, for any cloudlet k with $1 \leq k \leq K$, we have

$$\begin{aligned}
& w_k(j'+1) - w_k(j') \\
=& \frac{C_k}{c_{j'+1}} \cdot (\alpha^{1-\frac{C_k(j'+1)}{C_k}} - 1) - \frac{C_k}{c_{j'}} \cdot (\alpha^{1-\frac{C_k(j')}{C_k}} - 1) \\
\leq& \frac{C_k}{c_{j'}} \cdot (\alpha^{1-\frac{C_k(j'+1)}{C_k}} - 1) - \frac{C_k}{c_{j'}} \cdot (\alpha^{1-\frac{C_k(j')}{C_k}} - 1) && (5.11) \\
=& \frac{C_k}{c_{j'}} \cdot \left(\alpha^{1-\frac{C_k(j'+1)}{C_k}} - \alpha^{1-\frac{C_k(j')}{C_k}}\right) \\
=& \frac{C_k}{c_{j'}} \cdot \alpha^{1-\frac{C_k(j')}{C_k}} (\alpha^{\frac{C_k(j')-C_k(j'+1)}{C_k}} - 1) \\
\leq& \frac{C_k}{c_{j'}} \cdot \alpha^{1-\frac{C_k(j')}{C_k}} (\alpha^{\frac{c_{j'}}{C_k}} - 1) && (5.12) \\
=& \frac{C_k}{c_{j'}} \cdot \alpha^{1-\frac{C_k(j')}{C_k}} (2^{\frac{c_{j'}}{C_k} \log \alpha} - 1) \\
\leq& \frac{C_k}{c_{j'}} \cdot \alpha^{1-\frac{C_k(j')}{C_k}} (c_{j'} \cdot \log \alpha / C_k) && (5.13) \\
=& \alpha^{1-\frac{C_k(j')}{C_k}} \cdot \log \alpha. && (5.14)
\end{aligned}$$

where Inequality (5.11) follows because requests are sorted, Inequality (5.12) holds because at most $c_{j'}$ amount of computing resource is consumed at cloudlet k, and Inequality (5.13) holds because $2^x - 1 \leq x$ with $0 \leq x \leq 1$.

Recall that $\mathcal{K}_{j'}$ is the set of cloudlets in which the proposed approximation algorithm places a VNF instance of $r_{j'}$. We then calculate the sum of costs of cloudlets in G when admitting request $r_{j'}$. Notice that if none of the VNF instances of $r_{j'}$ is created at a cloudlet k, the cost of the cloudlet does not change after the admission of request $r_{j'}$. The difference in the cost sum of all cloudlets before and after admitting request $r_{j'}$ thus is

$$
\begin{aligned}
&\sum_{k=1}^{K}\big(w_k(j'+1)-w_k(j')\big)=\sum_{k\in\mathcal{K}_{j'}}\big(w_k(j'+1)-w_k(j')\big)\\
&\leq\sum_{k\in\mathcal{K}_{j'}}\big(\alpha^{1-\frac{C_k(j')}{C_k}}\cdot\log\alpha\big),\quad\text{by Inequality (5.14)}\\
&=\log\alpha\sum_{k\in\mathcal{K}_{j'}}\Big(\frac{w_k(j')}{C_k}+1\Big)\\
&=\log\alpha\left(\sum_{k\in\mathcal{K}_{j'}}\frac{w_k(j')}{C_k}+\sum_{k\in\mathcal{K}_{j'}}1\right)\\
&\leq\log\alpha\cdot((p_{j'}\cdot K)+K) \qquad (5.15)\\
&=\log\alpha\cdot K\cdot(p_{j'}+1)\\
&\leq\log\alpha\cdot K\cdot(p_{j'}+p_{j'}) \qquad (5.16)\\
&=2\log\alpha\cdot K\cdot p_{j'}, \qquad (5.17)
\end{aligned}
$$

where Ineq. (5.15) follows from the fact that $r_{j'}$ is admitted and Ineq. (5.10), Ineq. (5.16) follows because $p_{j'}\geq 1$.

The cost sum of all cloudlets after having examined last request $r_{|\mathcal{R}|}$ thus is

$$
\begin{aligned}
\sum_{k=1}^{K}w_k(|\mathcal{R}|+1)&=\sum_{j'=1}^{|\mathcal{R}|}\sum_{k=1}^{K}(w_k(j'+1)-w_k(j'))\\
&=\sum_{r_{j'}\in\mathcal{A}}\sum_{k=1}^{K}(w_k(j'+1)-w_k(j'))\\
&\leq\sum_{r_{j'}\in\mathcal{A}}(2K\cdot p_{j'}\cdot\log\alpha) \qquad (5.18)
\end{aligned}
$$

$$= 2K \cdot \log \alpha \cdot \sum_{r_{j'} \in \mathcal{A}} p_{j'},$$

where Inequality (5.18) follows from Inequality (5.17). □

Let $\mathcal{D}$ ($\subseteq \mathcal{R}$) be the set of requests that are admitted by an optimal algorithm but rejected by the proposed approximation algorithm. We now prove a lower bound on the sum of normalized costs of all cloudlets used to admit any request in $\mathcal{D}$.

Lemma 5.2. *Let $\mathcal{D}$ be the set of requests that are admitted by an optimal algorithm yet rejected by the approximation algorithm, and let $\mathcal{K}_{j'}^{opt}$ be the set of $(n_{j'} + 1)$ cloudlets to which the optimal algorithm places VNF instances for request $r_{j'} \in \mathcal{D}$. Then, for any request $r_{j'} \in \mathcal{D}$, we have*

$$\frac{\sum_{k \in \mathcal{K}_{j'}^{opt}} \psi_k(j')}{p_{j'}} \geq K, \tag{5.19}$$

where α is a constant with $2K \cdot Q_{\max} + 2 \leq \alpha \leq 2^{C_{\min}/c_{\max}}$, $C_{\min}$ $(= \min\{C_k \mid 1 \leq k \leq K\})$ is the minimum cloudlet computing capacity, and $Q_{\max}$ $(= \max\{p_{j'} \cdot c_{j'} \mid r_{j'} \in \mathcal{R}\})$ is the maximum product of the profit and computing resource demand among requests.

Proof. Consider a request $r_{j'}$ that is admitted by the optimal algorithm yet rejected by the proposed approximation algorithm. Since $r_{j'}$ is admitted by the optimal algorithm, it means that the optimal algorithm is able to admit $r_{j'}$ using a set $\mathcal{K}_{j'}^{opt}$ of cloudlets. There are exactly two cases. Case 1: every cloudlet in $\mathcal{K}_{j'}^{opt}$ has sufficient resources to admit $r_{j'}$; and Case 2: at least one cloudlet in $\mathcal{K}_{j'}^{opt}$ does not have sufficient resources to meet the resource demand of $r_{j'}$.

In the following we show that Inequality (5.19) holds in both of the two cases.

Case 1: If every cloudlet in $\mathcal{K}_{j'}^{opt}$ has sufficient resources to admit $r_{j'}$, the proposed approximation algorithm must be able to find a set $\mathcal{K}_{j'}$ of cloudlets such that $\mathcal{K}_{j'}$ can meet the resource demand of request $r_{j'}$ and for any set including $\mathcal{K}_{j'}^{opt}$ of $n_{j'} + 1$ cloudlets. That is, $\sum_{k \in \mathcal{K}_{j'}} \psi_k(j') \leq \sum_{k \in \mathcal{K}_{j'}^{opt}} \psi_k(j')$. Since $r_{j'}$ is rejected by the proposed algorithm, the cost sum of cloudlets in $K_{j'}^{opt}$ when admitting request $r_{j'}$ is no less than the given threshold in the admission control policy (5.10), i.e., $K \cdot p_{j'} \leq \sum_{k \in \mathcal{K}_{j'}} \psi_k(j') \leq$

$\sum_{k \in \mathcal{K}_{j'}^{opt}} \psi_k(j')$.

Case 2: At least one cloudlet in $\mathcal{K}_{j'}^{opt}$ does not have sufficient available resource to meet the demand of request $r_{j'}$. Thus, there must exist at least one cloudlet k' such that its residual computing capacity $C_{k'}(j')$ is less than the computing demand $c_{j'}$. Consequently, the sum of the normalized costs of cloudlets in $K_{j'}^{opt}$ is greater than $Kp_{j'}$:

$$\begin{aligned}
&\sum_{k \in K_{j'}^{opt}} \psi_k(j') \geq \psi_{k'}(j') = \frac{\alpha^{1-\frac{C_{k'}(j')}{C_{k'}}} - 1}{c_{j'}} \\
&> \frac{\alpha^{1-\frac{c_{j'}}{C_{k'}}} - 1}{c_{j'}}, \quad \text{since } C_k(j') < c_{j'} \\
&\geq \frac{\alpha^{1-\frac{1}{\log \alpha}} - 1}{c_{j'}}, \quad \text{since } \alpha \leq 2^{C_{\min}/c_{\max}} \leq 2^{C_{k'}/c_{j'}} \\
&= \frac{\frac{\alpha}{2} - 1}{c_{j'}} \geq Kp_{j'}, \quad \text{since } \alpha \geq 2K \cdot Q_{\max} + 2 \geq c_{j'} \cdot p_{j'} + 2.
\end{aligned}$$

□

Theorem 5.2. *Given an MEC $G(V, E)$, K cloudlets in G, and a set $\mathcal{R}$ of requests, there is an approximation algorithm,* `Algorithm` *5.1, with an approximation ratio of $O(\log K)$ for the reliability-aware VNF instances provisioning problem, which takes $O(|\mathcal{R}| \cdot \log |\mathcal{R}| + |\mathcal{R}| \cdot K)$ time.*

Proof. Let $\mathbb{P}_{opt}$ and $\mathbb{P}$ be the total revenues of admitted requests by an optimal algorithm and the proposed approximation algorithm for requests in $\mathcal{R}$, respectively, we then have

$$\begin{aligned}
&K(\mathbb{P}_{opt} - \mathbb{P}) \leq K \sum_{r_{j'} \in \mathcal{D}} p_{j'} = \sum_{r_{j'} \in \mathcal{D}} K \cdot p_{j'} \\
&\leq \sum_{r_{j'} \in \mathcal{D}} \Big(\sum_{k \in \mathcal{K}_{j'}^{opt}} \frac{w_k(j')}{C_k} \Big), \quad \text{by Lemma 5.2}
\end{aligned}$$

$$\leq \sum_{r_{j'} \in \mathcal{D}} \Big(\sum_{k \in \mathcal{K}_{j'}^{opt}} \frac{w_k(j)}{C_k} \Big) \tag{5.20}$$

$$\leq \sum_{k=1}^{K} w_k(j) \sum_{r_{j'} \in \mathcal{D}} \Big(\sum_{k' \in \mathcal{K}_{j'}^{opt}} \frac{1}{C_{k'}} \Big) \tag{5.21}$$

$$\leq \sum_{k=1}^{K} w_k(j) \cdot 1 \tag{5.22}$$

$$= \sum_{k=1}^{K} w_k(j) \leq 2K \log \alpha \cdot \sum_{j' \in \mathcal{A}} p_{j'} \quad \text{by Lemma 5.1.} \tag{5.23}$$

Notice that Inequality (5.20) holds because the utilization of each resource does not decrease and consequently the cost of any cloudlet k with $1 \leq k \leq K$ does not decrease with more request admissions. Inequality (5.21) holds since $\sum_{i=1}^{p} \sum_{j=1}^{q} A_i B_j \leq \sum_{i=1}^{p} A_i \sum_{j=1}^{q} B_j$ for all $A_i \geq 0$ and $B_j \geq 0$. The proof of Inequality (5.22) proceeds as follows. All algorithms, including an optimal algorithm for the problem of concern, the total amount of allocated computing resources in any cloudlet is no more than the capacity of the cloudlet.

By Inequality (5.23), we have

$$\begin{aligned}
\frac{\mathbb{P}_{opt}}{\mathbb{P}} &= \frac{\mathbb{P}_{opt} - \mathbb{P}}{\mathbb{P}} + 1 \\
&\leq \frac{2 \log \alpha \cdot \sum_{r_{j'} \in \mathcal{A}} p_{j'}}{\sum_{r_{j'} \in \mathcal{A}} p_{j'}} + 1 \leq 2 \log \alpha + 1 \\
&= O(\log(K \cdot Q_{\max})), \quad \text{when } \alpha = 2K \cdot Q_{\max} + 2. \\
&= O(\log K + \log Q_{\max}) \\
&= O(\log K), \quad \text{as } Q_{\max} \text{ usually is a constant,}
\end{aligned}$$

where $Q_{\max}$ is the maximum product of the profit and computing resource demand among requests, i.e., $Q_{\max} = \max\{p_{j'} \cdot c_{j'} \mid r_{j'} \in \mathcal{R}\}$.

The running time of `Algorithm` 5.1 is dominated by the time required to sort all requests by their ratio, which takes $O(|\mathcal{R} \cdot |\log|\mathcal{R}|)$ time. Then, there are $|\mathcal{R}|$ iterations, and within each iteration j, it identifies the top-$(n_j + 1)$ cloudlets with the

smallest weights using a linear time selection algorithm. The algorithm thus takes $O(|\mathcal{R}| \cdot \log |\mathcal{R}| + |\mathcal{R}| \cdot K)$ time. □

5.6 Approximation and Exact Algorithms for Special Cases of the Reliability-Aware VNF Instances Provisioning Problem

In this section, we study two special cases of the reliability-aware VNF instances provisioning problem, and propose a constant approximation algorithm and an exact algorithm for the cases, respectively.

5.6.1 A constant approximation algorithm for a special reliability-aware VNF instances provisioning problem

We start dealing with a special case of the reliability-aware VNF instances provisioning problem where each request has only one primary and secondary VNF instances, i.e., $n_j = 1$ for all requests $r_j \in \mathcal{R}$. Even for this special case, it is still NP-hard, for which we develop a constant approximation algorithm by a non-trivial reduction to the well-known Generalized Assignment Problem (GAP) [25], and a solution to the latter in turn returns a feasible solution to the former.

The Generalized Assignment Problem (GAP) is defined as follows. Given a set $\mathcal{B}$ of K bins with each bin $B_k \in \mathcal{B}$ having identical capacity of C, a set $\mathcal{I}$ of N items, and for each pair of item $I_j \in \mathcal{I}$ and bin $B_k \in \mathcal{B}$, a profit $p(I_j, B_k)$ is obtained if item I_j is placed to bin B_k with size $s(I_j, B_k)$, the GAP is to pack as many as items in set $\mathcal{I}$ into the bins in $\mathcal{B}$ such that the total profit is maximized, subject to the capacity constraint on each bin.

Given an instance of this special reliability-aware VNF instances provisioning problem, we construct an instance of the GAP as follows. We first assume that the K cloudlets are indexed into $1, 2, \ldots, K$ and K is even. Otherwise, we assume that there

are $K+1$ cloudlets, and one cloudlet is a *virtual one* that implies that it does not exist. We pair the K cloudlets into $K/2$ pairs, let $B'_1, B'_2, \ldots, B'_{K/2}$ be the $K/2$ *pair-bins* with each having $2C$ computing capacity as the instance of the GAP problem. Each request $r_j \in \mathcal{R}$ has a corresponding item I_j and with resource demand $2c_j$ for its primary and secondary VNF instances.

If the primary and secondary VNF instances of request $r_j \in \mathcal{R}$ will be placed into one pair of cloudlets, corresponding to one bin B'_k with size $2c_j$, then the capacity $2C$ of bin B'_k is no less than the total resource demand $2c_j$ by these two VNF instances, and this placement will result in a profit (revenue) $p(I_j, B'_k)$ that is defined as

$$p(I_j, B'_k) = p_j,$$

where $1 \leq j \leq |\mathcal{R}|$ and $1 \leq k \leq K/2$.

The size of placing item I_j into B'_k is $s(I_j, B'_k) = 2c_j$, corresponding to the resource demand $2c_j$ of VNF instances of request r_j.

The constructed GAP instance can be solved by invoking the approximation algorithm due to Cohen *et al.* [25] with an approximation ratio of $\frac{1}{2+\epsilon}$, where ϵ is a constant with $0 < \epsilon \leq 1$.

The solution delivered by the approximation algorithm will be in the form of assigning a set $\mathcal{I}' \subseteq \mathcal{I}$ of requests. Specifically, each item $I_j \in \mathcal{I}'$ that corresponds to the primary and secondary VNF instances of request r_j are assigned to cloudlets B_{2k} and B_{2k+1} if its corresponding item I_j is assigned to bin B'_k with $1 \leq k \leq K/2$.

We then extend the solution from the even K to the odd K, for which we create *a virtual cloudlet* with the same capacity as other cloudlets. Thus, we now have $K' = K+1$ bins. They are corresponding $K'/2$ pair-bins $B'_1, B'_2, \ldots, B'_{\frac{K+1}{2}}$ with each having the computing capacity of $2C$. Assume that the virtual cloudlet is paired with the cloudlet indexed K. We then apply the approximation algorithm for the GAP problem due to Cohen *et al.* [25].

Let $\mathcal{I}'$ be the approximate solution and $\mathcal{I}'_1.\mathcal{I}'_2, \ldots, \mathcal{I}'_{K'/2}$ the sets of admitted re-

quests in bins $B'_1, B'_2, \ldots, B'_{K'/2}$, respectively. Let $P'_1, P'_2, \ldots, P'_{K'/2}$ be the profit sum of requests admitted in their corresponding pair-bins.

Denote by $\sum_{i=1}^{K'/2} P'_i$ the total revenue collected by admitting the requests in $\mathcal{I}'$. The average profit among the $K'/2$ pair-bins thus is $\frac{\sum_{i=1}^{K'} P'_i}{K'/2}$. Let $P'_{\min} = \min_{1 \le i \le K'/2}\{P'_i\}$. If $P'_{K'/2} = P'_{\min}$, we discard the requests in $B'_{K'/2}$ from the solution, i.e., $\mathcal{I}' = \mathcal{I}' \setminus \mathcal{I}'_{K'/2}$; otherwise, let $P'_{i_0} = P'_{\min}$ and $i_0 \neq K'/2$, we swap the admitted requests between pair-bin B'_{i_0} and pair-bin $B'_{K'/2}$, i.e., $\mathcal{I}'_{i_0} = \mathcal{I}'_{K'/2}$ and $\mathcal{I}'_{K'/2} = \mathcal{I}'_{i_0}$. Then, the solution is $\mathcal{I}' = \mathcal{I}' \setminus \mathcal{I}'_{K'/2}$. The detailed algorithm is given in `Algorithm` 5.2.

Theorem 5.3. *Given an MEC $G(V, E)$ that contains K cloudlets with each having identical residual computing capacity C, and a set of requests $\mathcal{R}$ with each request $r_j \in \mathcal{R}$ having exactly one secondary VNF instance, there is an approximation algorithm with an approximation ratio of $\frac{1}{6+\epsilon'}$,* `Algorithm` *5.2, for this special reliability-aware VNF instances provisioning problem. The proposed algorithm takes $O(\frac{|\mathcal{R}| \cdot K}{\epsilon'} + \frac{K}{\epsilon'^4})$ time, where ϵ' is a constant with $0 < \epsilon' \le 3$.*

Proof. We first show that the solution delivered by `Algorithm` 5.2 is feasible. Since a pair-bin B'_k that corresponds to cloudlets $2k$ and $2k+1$ has the computing capacity of $2C$, and the size $s(I_j, B'_k) = 2c_j$ of placing item I_j (admitting request r_j by instantiating its primary and secondary VNF instances into B_{2k} and B_{2k+1}) is equal to their resource demand $2c_j$, it follows that the allocation of VNF instances by solving the GAP problem, i.e., the capacity constraint on each cloudlet will not be violated.

We then analyze the approximation ratio of `Algorithm` 5.2, by distinguishing it into two cases, depending on whether the value of K is even or not. If K is even, then the solution is an approximate solution, which is $\frac{1}{2+\epsilon}$ times the optimal by the approximation algorithm in [25]; otherwise (K is odd), we create a virtual cloudlet indexed as K' $(= K+1)$. Let $\mathcal{I}'$ be the approximate solution delivered by the approximation algorithm. As this solution is built upon the assumption that there are K' cloudlets, we in fact have only K cloudlets in the network. Following `Algorithm` 5.2, let A be the total profit of the approximate solution, the minimum profit among the $K'/2$ pairs of cloudlets thus is no greater than $\frac{A}{K'/2}$. As we will remove all admitted requests in the

Algorithm 5.2 An approximation algorithm for the special reliability-aware VNF instances provisioning problem with $n_j = 1$ for every request $r_j \in \mathcal{R}$

Input: A set of K cloudlets with each having a residual computing capacity C, a set of requests $\mathcal{R}$ with each request $r_j = (f_j, R_j, c_j, p_j)$

Output: Admit a subset $\mathcal{I}'$ of requests in $\mathcal{R}$ that maximizes the sum of revenues of admitted requests while meeting the reliability of each admitted request.

1: **if** K is even **then**
2: Construct an instance of the GAP, where each request $r_j \in \mathcal{R}$ has a corresponding item I_j and each pair of cloudlets has a corresponding bin B'_k with bin capacity $cap(B'_k) = 2C$ with $1 \leq k \leq K/2$;
3: Find an approximate solution $\mathcal{I}'$ to the GAP problem using the approximation algorithm due to Cohen *et al.* [25];
4: **else**
5: Construct an instance of the GAP, where each request $r_j \in \mathcal{R}$ has a corresponding item I_j and each pair of cloudlets has a corresponding bin B'_k with bin capacity $cap(B'_k) = 2C$ with $1 \leq k \leq \frac{K+1}{2}$, where a virtual cloudlet (bin) is added to the system, and let $K' = K + 1$, which is even;
6: Find an approximate solution $\mathcal{I}'$ to the GAP problem using the approximation algorithm due to Cohen *et al.* [25];
7: $P'_{\min} \leftarrow \infty$;
8: **for** $i \leftarrow 1$ to $K'/2$ **do**
9: Calculate P'_i from $\mathcal{I}'_i$;
10: **if** $P'_i < P'_{\min}$ **then**
11: $P'_{\min} \leftarrow P'_i$; $i_0 \leftarrow i$; $\mathcal{I}'_{temp} \leftarrow \mathcal{I}'_{i_0}$;
12: **end if**;
13: **end for**;
14: **if** $i_0 = K'/2$ **then**
15: $\mathcal{I}' \leftarrow \mathcal{I}' \setminus \mathcal{I}'_{K'/2}$;
16: **else**
17: $\mathcal{I}'_{i_0} \leftarrow \mathcal{I}'_{K'/2}$; $\mathcal{I}'_{K'/2} \leftarrow \mathcal{I}'_{temp}$; $\mathcal{I}' \leftarrow \mathcal{I}' \setminus \mathcal{I}'_{K'/2}$;
18: **end if**;
19: **end if**;
20: **for** each item $I_j \in \mathcal{I}'_k$ assigned to bin B'_k with $1 \leq k \leq \lceil K/2 \rceil$ **do**
21: **if** $k \leq \lfloor K/2 \rfloor$ **then**
22: Instantiate the primary and secondary VNF instances of f_j for request r_j in cloudlets $2k - 1$ and $2k$;
23: **end if**;
24: **end for**;
25: **return** the set of admitted requests and their primary and secondary VNF instance placements in cloudlets.

minimum profit pair-bin from the solution, the resulting profit by the solution thus is at least $A - \frac{A}{K'/2} = A(1 - \frac{2}{K+1}) \geq A/3$ due to the fact that $K \geq 2$. Since $A \geq \frac{OPT}{2+\epsilon}$, $\frac{A}{3} \geq \frac{OPT}{6+\epsilon'}$, where OPT is the optimal solution to the problem, and $\epsilon' = 3\epsilon$ is a constant with $0 < \epsilon' \leq 3$.

We finally analyze the running time of `Algorithm` 5.2. The construction of the GAP instance takes $O(|\mathcal{R}| \cdot K)$ time, while invoking the approximation algorithm due to Cohen *et al.* [25] takes $O(\frac{|\mathcal{R}| \cdot K}{\epsilon'} + \frac{K}{\epsilon'^4})$ time. The solution delivered by the proposed algorithm, `Algorithm` 5.2, thus is no less than $\frac{1}{6+\epsilon'}$ times the optimal one, where ϵ' is a constant with $0 < \epsilon' \leq 3$. □

5.6.2 A dynamic programming algorithm for another special reliability-aware VNF instances provisioning problem

We then study another special case of the reliability-aware VNF instances provisioning problem where different VNF instances of different network functions have the same amounts of computing resource demands, i.e., $c_i = c_j$ for all $f_i \in \mathcal{F}$ and $f_j \in \mathcal{F}$. For the sake of convenience, we assume that the computing resource demand by each network function is one computing unit. We propose an exact algorithm for the problem through a reduction to a profit maximization problem (defined later), and the solution to the latter in turn returns a solution to the former. The reduction is as follows.

5.6.2.1 A dynamic programming algorithm

Denote by n the number of requests $r_1, r_2, \ldots, r_n$ in $\mathcal{R}$, which correspond n jobs $J_1, J_2, \ldots, J_n$. There are K cloudlets in G, and each can be treated as a bin B_i with computing capacity C_i, $1 \leq i \leq K$. The reliability requirement of request r_j is implemented by placing $(n_j + 1)$ VNF instances to $n_j + 1$ cloudlets with each being allocated a computing unit at each cloudlet. The implementation of job J_j will take U_j bins and consume one computing unit in each of the chosen U_j bins. The profit (revenue) obtained by implementing job J_j is p_j. *The profit maximization problem* then is to find a subset $\mathcal{A}$ of $\mathcal{R}$ ($\mathcal{A} \subseteq \mathcal{R}$) such as the sum of profits of admitted requests in $\mathcal{A}$ is maximized, subject to the computing capacity on each cloudlet in the network.

The defined profit maximization problem can be solved, using dynamic program-

ming as follows. Without loss of generality, we assume

$$C_1 \geq C_2 \geq \ldots \geq C_K, \tag{5.24}$$

and

$$U_1 \geq U_2 \geq \ldots \geq U_n. \tag{5.25}$$

Because each job needs at most one computing unit from any bin, we assume $n \geq C_1$, otherwise excessive capacity larger than n will be useless.

Let $W_i = |\{j \mid C_j \geq i\}|$ be the number of bins that have at least i computing units, for each and every i with $1 \leq i \leq n$. Notice that some W_i may be zero. We then have

$$W_1 \geq W_2 \geq \ldots, \geq W_n. \tag{5.26}$$

Definition: If a job J is selected which needs U computing units, then we assign one computing unit starting from the bin with the largest remaining capacity in an non-increasing order of remaining capacities until U computing units have been assigned. We refer to this scheduling method as *the canonical scheduling*.

Theorem 5.4. *All jobs $J_1, J_2, \ldots, J_n$ can be selected if and only if the following condition is satisfied.*

$$\sum_{j=1}^{i} U_j \leq \sum_{j=1}^{i} W_j \quad \textit{for all } i \textit{ with } 1 \leq i \leq n. \tag{5.27}$$

Proof. We prove Claim (5.27) by induction on the number of jobs n.

We start with induction basis. When $n = 1$, there is only one job that requires U_1 computing units. Therefore, if J_1 can be selected, we must have $U_1 \leq W_1$. On the other hand, if $U_1 \leq W_1$, then J_1 can be selected because its demanded U_1 computing units can be satisfied. We then assume that Claim (5.27) holds for all n with $1 \leq n \leq h$. We finally show that Claim (5.27) also holds when $n = h + 1$ as follows.

On one hand, suppose that all $h + 1$ jobs can be selected. In this case, since the first h jobs can be selected by the induction assumption, we have $\sum_{j=1}^{i} U_j \leq \sum_{j=1}^{i} W_j$

for all i with $1 \leq i \leq h$. Now, because the first $h+1$ jobs can also be selected, we then must have $\sum_{j=1}^{h+1} U_j \leq \sum_{j=1}^{h+1} W_j$ where $\sum_{j=1}^{h+1} U_j$ is the total number of computing units required by the $h+1$ jobs and $\sum_{j=1}^{h+1} W_j$ is the total number of available computing units provided by all bins, because any bin with a capacity larger than $h+1$ can only contribute at most $h+1$ computing units.

On the other hand, suppose Claim (5.27) holds, i.e., $\sum_{j=1}^{i} U_j \leq \sum_{j=1}^{i} W_j$ for all i with $1 \leq i \leq h+1$. We show that all $h+1$ jobs can be selected by canonically scheduling J_1 and distinguishing two cases: Case A and Case B, to deal with the remaining h jobs.

Case A: if $U_1 = W_1$, then we have $\sum_{j=2}^{i} U_j \leq \sum_{j=2}^{i} W_j$ for all i with $2 \leq i \leq h+1$. Because there are h remaining jobs, by the induction assumption, all the remaining h jobs can be selected.

Case B: if $U_1 < W_1$, then not all W_1 bins are used, and there are $W_1 - U_1$ bins not used. We relabel the h remaining jobs by labeling job J_2 as J'_1, J_3 as J'_2, ..., and J_{h+1} as J'_h. Accordingly, we have $U'_1 = U_2$, $U'_2 = U_3$, ..., $U'_h = U_{h+1}$. Denote by W'_i the number of bins that have at least i remaining computing units for all i with $1 \leq i \leq n$. We further distinguish Case B into two subcases: (i) $W_2 \leq U_1$; and (ii) $W_2 > U_1$, respectively.

Subcase (i): if $W_2 \leq U_1$, then

$$\begin{aligned} W'_1 &= W_2 + W_1 - U_1, \\ W'_2 &= W_3, \\ &\vdots \\ W'_h &= W_{h+1}. \end{aligned} \tag{5.28}$$

We thus have $U'_1 = U_2 \leq W_2 + W_1 - U_1 = W'_1$, because $U_1 + U_2 \leq W_1 + W_2$. In general, for all i with $2 \leq i \leq h$, we have

$$\sum_{j=1}^{i} U'_j = \sum_{j=2}^{i+1} U_j = \sum_{j=1}^{i+1} U_j - U_1$$

$$\leq \sum_{j=1}^{i+1} W_j - U_1, \text{ by Claim (5.27)}$$
$$= \sum_{j=3}^{i+1} W_j + (W_2 + W_1 - U_1), \text{ by Equation (5.28)}$$
$$= \sum_{j=2}^{i} W'_j + W'_1 = \sum_{j=1}^{i} W'_j \tag{5.29}$$

By the induction assumption, all h remaining jobs can be selected by the bins.

Subcase (ii): Assume that there is a p such that $W_2 > U_1$, $W_3 > U_1$, ..., $W_p > U_1$ but $W_{p+1} \leq U_1$. We then have

$$W'_1 = W_2 + (W_1 - W_2) = W_1,$$
$$W'_2 = W_3 + (W_2 - W_3) = W_2,$$
$$\vdots$$
$$W'_{p-1} = W_p + (W_{p-1} - W_p) = W_{p-1}, \tag{5.30}$$
$$W'_p = W_{p+1} + (W_p - U_1), \tag{5.31}$$
$$W'_{p+1} = W_{p+2},$$
$$\vdots$$
$$W'_h = W_{h+1}. \tag{5.32}$$

Note that if $p = h + 1$, then from Inequality (5.25) we have $W_j > U_j$ for all j with $1 \leq j \leq h + 1$. Then, all jobs can be selected. We thus assume that $p \leq h$, and this situation is illustrated in Figure 5.2. We now have $W'_1 = W_1 \geq W_2 > U_1 \geq U_2 = U'_1$.

For $i = 2, \ldots, p - 1$, we have

$$\sum_{j=1}^{i} U'_j = \sum_{j=2}^{i+1} U_j \leq \sum_{j=2}^{i+1} U_1, \quad \text{from inequality (5.25)}$$
$$\leq \sum_{j=1}^{i} W_j, \quad \text{from the assumption of subcase (ii)}$$
$$\leq \sum_{j=1}^{i} W'_j. \tag{5.33}$$

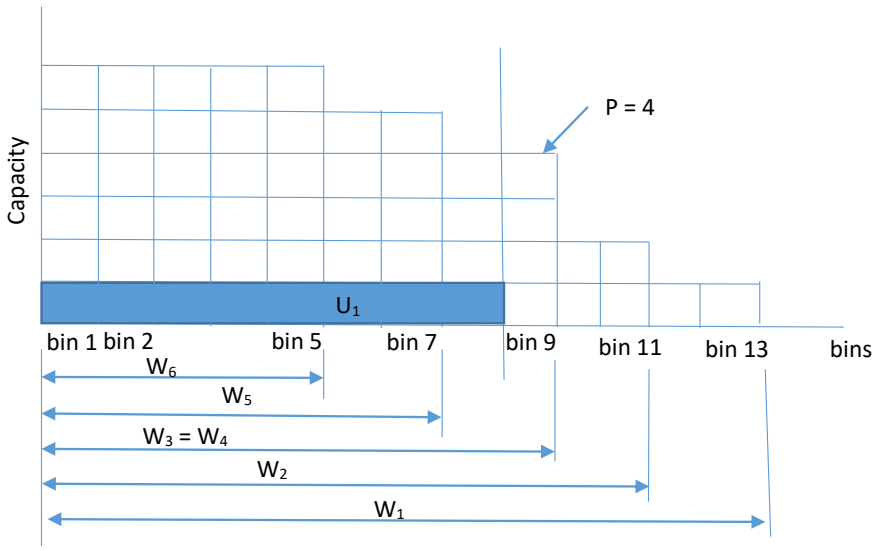


Figure 5.2: An illustration of the proof of Theorem 5.4.

For $i = p, p+1, \ldots, h$, we have

$$\begin{aligned}
\sum_{j=1}^{i} W'_j &= \sum_{j=1}^{p-1} W'_j + W'_p + \sum_{j=p+1}^{i} W'_j, \\
&= \sum_{j=1}^{p-1} W_j + W_{p+1} + (W_p - U_1) + \sum_{j=p+1}^{i} W'_j, \text{ by Equation (5.31).} \\
&= \sum_{j=1}^{p+1} W_j - U_1 + \sum_{j=p+2}^{i+1} W_j, \text{ by Equation (5.32).} \\
&= \sum_{j=1}^{i+1} W_j - U_1 \geq \sum_{j=1}^{i+1} U_j - U_1 = \sum_{j=2}^{i+1} U_j = \sum_{j=1}^{i} U'_j. \qquad (5.34)
\end{aligned}$$

Therefore, by the induction assumption, all h remaining jobs can be selected by the bins. The theorem thus follows. □

We thus can derive the following corollary from Theorem 5.4.

Corollary 5.4.1. *Given any subset of jobs $\{J_{j_1}, J_{j_2}, \ldots, J_{j_p}\}$ of a set of jobs $\{J_1, J_2, \ldots, J_n\}$ with $U_{j_1} \geq U_{j_2} \geq \ldots \geq U_{j_p}$, $j_l \in \{1, 2, \ldots, n\}$ and $1 \leq l \leq p$, each job J_{j_l} demands U_{j_l} ($1 \leq U_{j_l} \leq K$) bins with one computing unit per bin. Assume that there are K bins with each bin B_k having C_k computing units and $C_1 \geq C_2 \geq \ldots \geq C_K$, all the jobs in the subset is*

admissible by the K bins if and only if the following condition is satisfied.

$$\sum_{i=1}^{h} U_{j_i} \leq \sum_{i=1}^{h} W_i, \text{ for all } h \text{ with } 1 \leq h \leq p. \tag{5.35}$$

Recall the admission of a job J_j leads to a profit of p_j. Let $U = \sum_{j=1}^{n} U_j$. Define $L_i = \sum_{j=1}^{i} W_j$ for all i with $1 \leq i \leq n$. Denote by $P(i,h,Y)$ the maximum profit from selecting i jobs from the first h jobs with $i \leq h \leq n$, whose total number of computing units needed is Y. Clearly $0 \leq Y \leq \sum_{j=1}^{n} U_j = U$. If there is no solution, then $P(i,h,Y) = 0$. The initiation of $P(0,h,Y)$ is 0 for any $1 \leq h \leq n$ and $0 \leq Y \leq U$. The recurrence is defined as follows.

$$P(i,h,Y) = \max \begin{cases} p_h + P(i-1,h-1,Z), \\ \qquad \text{if } Y = Z + U_h \text{ and } Y \leq L_h, \\ P(i,h-1,Y), \text{ otherwise.} \end{cases} \tag{5.36}$$

where p_h is the revenue collected if request r_h is admitted and $U_h = n_h + 1$ is the number of VNF instances placed for request r_h, and Z is the total number of computing units required by the first $h-1$ jobs admitted prior to the admission of job h, i.e., $Z = \sum_{i=1}^{h-1} U_{j_i}$. Note that from Corollary 5.4.1, $Y \leq L_i$ is to guarantee that the solution is valid.

Denote by $V(i,h,Y) = \{j_1, j_2, \ldots, j_i\}$ the set of indices of the selected i jobs that achieves $P(i,h,Y)$ – the maximum profit for this sub-problem. Then, $V(i,h,Y)$ is recursively defined as follows.

$$V(i,h,Y) = \begin{cases} V(i,h-1,Y) \text{ if } P(i,h,Y) = P(i,h-1,Y), \\ V(i-1,h-1,Z) \cup \{h\}. \end{cases} \tag{5.37}$$

The solution to the problem is $\max\{P(i,h,Y) \mid 1 \leq i \leq h \leq n, 0 \leq Y \leq U\}$. Specifically, given a set $\mathcal{R}$ of requests, a set of K cloudlets with each cloudlet k accommodating C_k VNF instances, Recurrence (5.36) can be used to derive the maximum

revenue by admitting a subset of requests in $\mathcal{R}$. An exact algorithm for the reliability-aware VNF instances provisioning problem then follows, and the detailed algorithm is given in Algorithm 5.3.

Algorithm 5.3 An exact algorithm for the special reliability-aware VNF instances provisioning problem

Input: K cloudlets with each cloudlet j accommodating C_j VNF instances with the same computing resource demand by different network function instances, a set of requests $\mathcal{R}$ with each request $r_j = (f_j, R_i, c_j, p_j) \in \mathcal{R}$

Output: Admit a subset of requests in $\mathcal{R}$ such that the sum of revenues of admitted requests is maximized while the reliability requirement of each admitted request is met.

1: **for each** request $r_j \in \mathcal{R}$ **do**
2: Calculate the number n_j of secondary VNF instances by Inequality (5.3);
3: **end for**
4: Identify a subset $\mathcal{A}$ of requests in $\mathcal{R}$ to maximize the revenue collected by solving Recurrence (5.36) for all i, h, and C, and let a subset $\mathcal{A}$ is identified such that the revenue is maximized;
5: Let $r'_1, r'_2, \ldots, r'_{|\mathcal{A}|}$ be the sequence of the ordered requests in the solution obtained in $\mathcal{A}$;
6: **for** $j \leftarrow 1$ to $|\mathcal{A}|$ **do**
7: Perform $n_j + 1$ VNF instance allocations to $n_j + 1$ cloudlets, where n_j VNF instances serve as secondary VNFs and one VNF instance serves as the primary VNF instance for each admitted request $r'_j \in \mathcal{A}$.
8: **end for**

The rest is to analyze the time complexity of `Algorithm` 5.3 by the following theorem.

Theorem 5.5. *Given an MEC $G(V, E)$ and a set of requests $\mathcal{R}$, assume that each cloudlet k of the K cloudlets in G has a computing capacity C_k with $1 \leq k \leq K$, there is an exact algorithm,* `Algorithm` *5.3, for the reliability-aware VNF instances provisioning problem with different VNF instances have demanded the same amount of computing resource, which delivers an exact solution within $O(K \cdot |\mathcal{R}|^3)$ time, where K is the number of cloudlets in G.*

Proof. The running time of `Algorithm` 5.3 is dominated by solving the recurrence in the dynamic programming and subsequently VNF instances allocations for each admitted request, both take $O(K \cdot |\mathcal{R}|^3)$ time since $0 \leq i \leq |\mathcal{R}|$, $1 \leq h \leq |\mathcal{R}|$. The time complexity for solving Recurrence (5.36) is as follows. Consider how to select i jobs from the first h jobs with $i \leq h$, such that the total required computing unit is Y. Obviously, $0 \leq W \leq \sum_{j=1}^{n} U_j \leq Kn$, as each request has at most K VNF instances

with each placed at one of the K cloudlets. For each fixed i, h, and W, we find a solution that has the maximum total profit. Since $1 \leq i \leq h \leq |\mathcal{R}|$, there are $O(K|\mathcal{R}|^3)$ sub-problems need to solve. □

5.7 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms for the reliability-aware VNF instances provisioning problem. We also investigate the impact of parameters on the performance of the proposed algorithms.

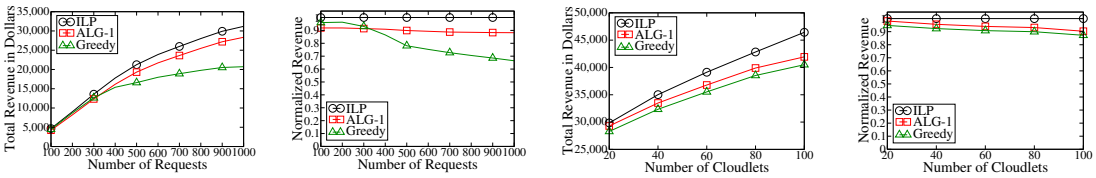
5.7.1 Experimental environment settings

We assume that a MEC $G = (V, E)$ consists of 100 APs, in which the number of cloudlets K is 10% of the network size, and the cloudlets are randomly co-located with some of the APs. Each network topology is generated using the widely adopted approach due to Barabási and Albert [10]. The computing capacity C_k of each cloudlet is drawn in a range from 2,000 to 4,000 MHz [53]. The network offers 20 different types of network functions, i.e., $|\mathcal{F}| = 20$, where the computing resource demand c_i of a VNF instance of network function $f_i \in \mathcal{F}$ is randomly drawn between 40MHz and 400MHz [4]. The number n_j of secondary VNF instances of the network function for request r_j is randomly drawn between 0 and 4, because in the extreme case where the failure rate of each cloudlet is 10%, allocating four secondary VNF instances and one primary VNF instance to a request is able to guarantee a carrier-grade reliability of 99.999% (five nines). The running time obtained of each mentioned algorithm is based on a desktop with a 4GHz quad-core Intel i7 CPU and 16 GB RAM.

We evaluate the proposed algorithms `Algorithm` 5.1, `Algorithm` 5.2, `Algorithm` 5.3, and the ILP solution, which are referred to as `ALG-1`, `ALG-2`, `ALG-3`, and `ILP`, respectively. Each value in figures is the mean of the results of 30 trials.

5.7.2 Algorithm performance evaluation for the reliability-aware VNF instances provisioning problem

We first evaluate the proposed approximation algorithm `Algorithm` 5.1 for the reliability-aware VNF instances provisioning problem against a baseline algorithm `Greedy` that adopts a linear cost model. Given a request $r_j = (f_j, R_j, c_j, p_j)$ that needs $(n_j + 1)$ VNF instances, algorithm `Greedy` places its VNF instances into top-$(n_j + 1)$ cloudlets with the largest residual computing capacity. It takes $O(K)$ time to identify the top-$(n_j + 1)$ cloudlets if a linear-time selection algorithm is applied. Thus, the running of algorithm `Greedy` is $O(K \cdot |\mathcal{R}|)$, where K is the number of cloudlets and $\mathcal{R}$ is the set of requests.



(a) Total revenues by different algorithms with different numbers of requests

(b) Normalized revenues by different algorithms with different numbers of requests

(c) Total revenues by different algorithms with different number of cloudlets

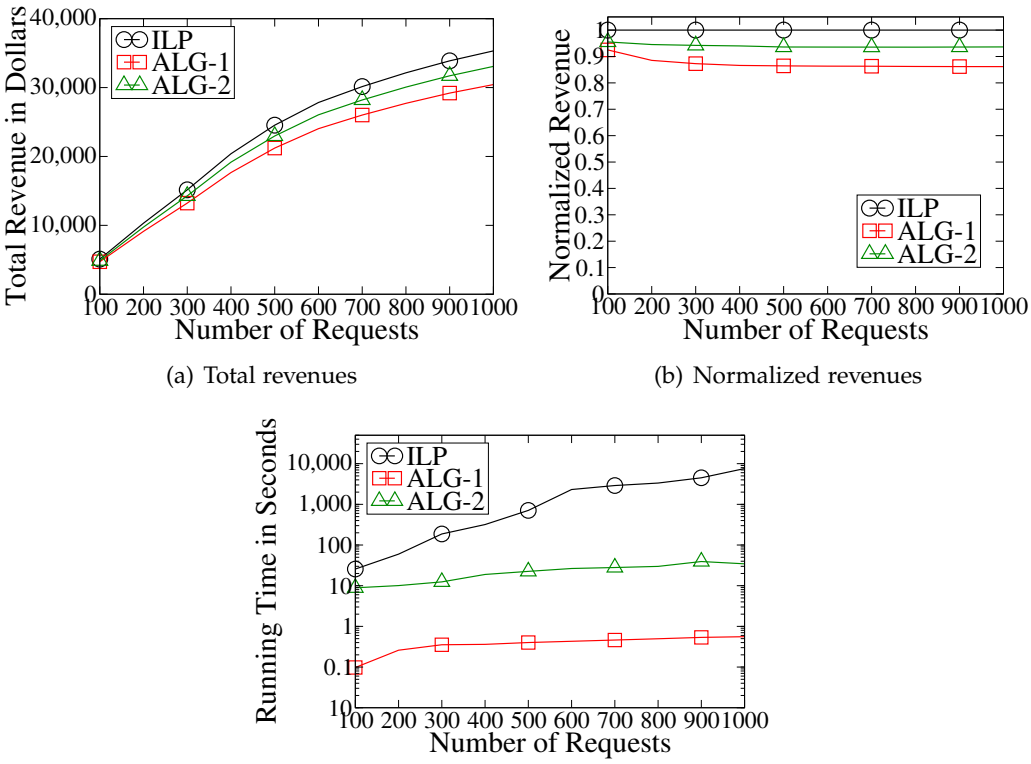
(d) Normalized revenues by different algorithms with different number of cloudlets

Figure 5.3: Performance evaluation of the approximation algorithm `ALG-1` for the reliability-aware VNF instances provisioning problem

Figure 5.3 (a)-(b) shows the curves of the total revenues and the normalized revenues delivered by algorithms `ILP`, `ALG-1` and `Greedy`, respectively, where *the normalized revenue* refers to the average revenue of admitting a request. It can be observed from these two figures that when the number of requests is very small, algorithm `Greedy` achieves a slightly better performance than algorithm `ALG-1`. However, with more and more request arrivals over time, the total revenue delivered by algorithm `ALG-1` increases steadily while algorithm `Greedy` grows at a much slower rate. When the number of requests reaches 1,000, the total revenue of algorithm `Greedy` is approximately 70% of that of algorithm `ALG-1`. The reason behind is that algorithm `ALG-1` is more conservative: it rejects those requests with high costs by the admis-

sion control policy to alleviate overloading of network resources, thereby achieving better performance. We also evaluate the approximation ratio of algorithm `ALG-1` empirically through comparing the total revenue delivered by algorithm `ALG-1` and the optimal one by the optimal algorithm `ILP`. It can be seen from Figure 5.3 (c)-(d) that the empirical approximation ratio of algorithm `ALG-1` is at most 1.17 in all cases, compared with the analytical approximation ratio of 20 according to Theorem 5.2 when $K = 100$. This demonstrates that the empirical performance of algorithm `ALG-1` is significantly better than its analytical counterpart.

5.7.3 Algorithm performance evaluation for special reliability-aware VNF instances provisioning problems



(a) Total revenues

(b) Normalized revenues

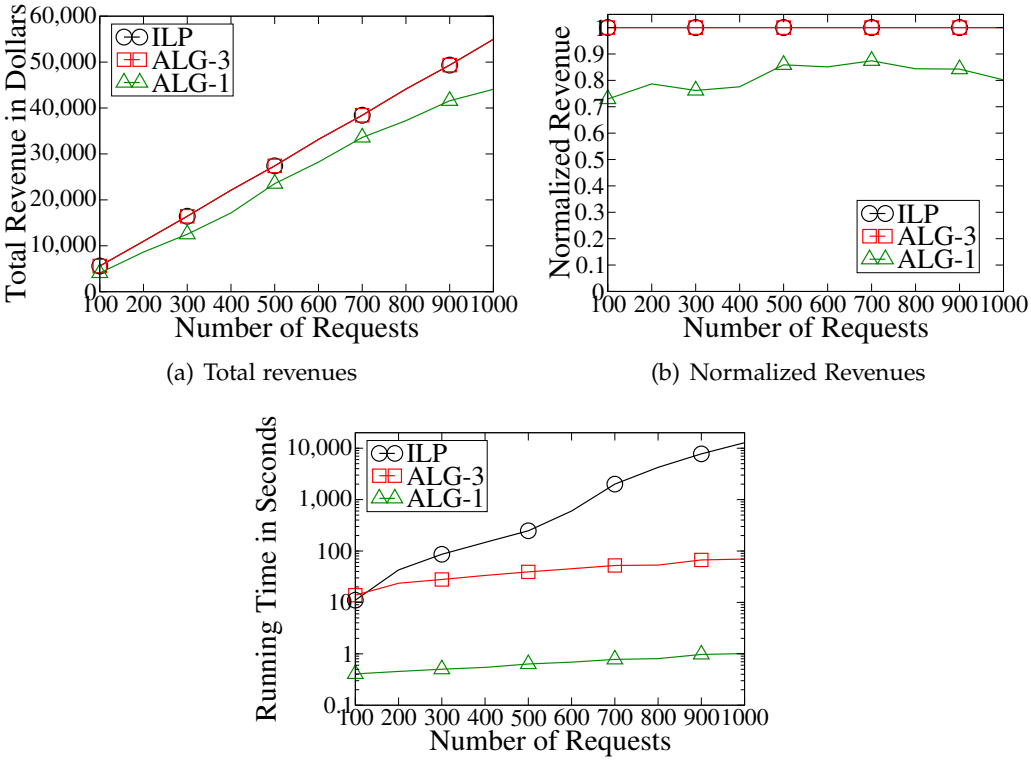
(c) Running time on a logarithmic scale

Figure 5.4: Performance of different algorithms for the special case of the reliability-aware VNF instances provisioning problem where each request requires exactly one secondary instance

What follows is to study the performance of the proposed approximation algorithm `ALG-2` against algorithms `ALG-1` and `ILP` for the special case of the problem where every request needs only one secondary VNF instance. It can be observed from Figure 5.4 (a)-(b) that the total revenue delivered by algorithm `ALG-2` is nearly close to the exact one delivered by algorithm `ILP`, while the solution delivered by algorithm `ALG-1` is the worst. Specifically, when the number of requests is 1,000, the total revenue by algorithm `ALG-2` is around 93% of the optimal solution, which is an 8% improvement over the one delivered by algorithm `ALG-1`. It also can be seen from Figure 5.4 (c) that among the three comparison algorithms, algorithm `ALG-1` runs fastest, while algorithm `ILP` is the slowest. This demonstrates a non-trivial tradeoff between the quality of a solution and the running time to deliver the solution.

We thirdly evaluate the performance of `ALG-3` against `ILP` and `ALG-1` for the special reliability-aware VNF instances provisioning problem where the computing demand by each network function instance is identical, by varying the number of requests from 100 to 1,000.

Figure 5.5 (a)-(c) show the total revenue, normalized revenue, running time of the mentioned algorithms, respectively. It can be seen from Figure 5.5 (a)-(b) that both algorithms `ALG-3` and ILP deliver exact solutions to the problem, while the heuristic algorithm `ALG-1` delivers a very good solution. The total revenue delivered by algorithm `ALG-1` is no less than 80% of that by algorithms `ALG-3` and `ILP`. Despite the optimal solutions delivered by both algorithms `ALG-3` and `ILP`, it can be seen from Figure 5.5 (c) that algorithm `ILP` is very time-consuming and exhibits poor scalability while algorithm `ALG-3` runs significantly faster. It is observed that when there are 100 requests, the running time of algorithm `ALG-3` is larger than that of algorithm `ILP`, due to the overhead of constructing data structures for solving the recurrence in dynamic programming. However, when the number of requests reaches 1,000, algorithm `ALG-3` takes less than one minute while algorithm `ILP` takes more than two hours to find an optimal solution. Meanwhile, Figure 5.5 (c) also indicates



(a) Total revenues

(b) Normalized Revenues

(c) Running time on a logarithmic scale

Figure 5.5: Performance of different algorithms for the special reliability-aware VNF instances provisioning problem

that the running time of algorithm `ALG-1` is only a small fraction of algorithm `ALG-3`, not to mention algorithm `ILP`.

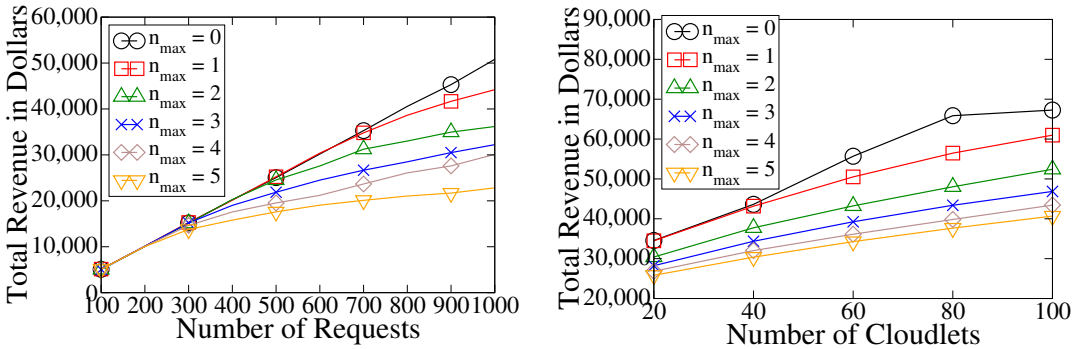
5.7.4 Parameter impacts on algorithm performance

We finally study the impact of important parameters on the performance of algorithms `ALG-1`.

In all experiments so far we assumed that the maximum number $n_{\max}$ of secondary VNF instances of each request is set at 4. We now investigate the impact of $n_{\max}$ on the performance of algorithm `ALG-1`, by varying $n_{\max}$ while fixing the number of cloudlets at 20. Figure 5.6 (a) shows the total revenue delivered by it when the number of requests grows from 100 to 1,000. Notice that when the number of requests is less than 300, the total revenues of different algorithms with different $n_{\max}$ are nearly identical, meaning the network has a relatively abundant amount of resources to accommodate different requests. However, with more requests, the larger the value of $n_{\max}$, the smaller the total revenue delivered by algorithm `ALG-1`. The reason behind this is that the amount of available resources in the network and the payments of requests do not change, yet each request demands more resources when $n_{\max}$ increases.

We also evaluate the total revenues delivered by algorithm `ALG-1` by varying the number of cloudlets K while fixing the number of requests at 1,000. The results are shown in Figure 5.6 (b). When the number K of cloudlets increases, the amount of available resources increases accordingly. As a result, the larger K is, the higher the total revenue delivered by algorithm `ALG-1` is. Meanwhile, for the same K, the larger the value of $n_{\max}$, the lower the total revenue delivered by `ALG-1`, because the revenue of a request does not increase yet the resource demand of the request increases.

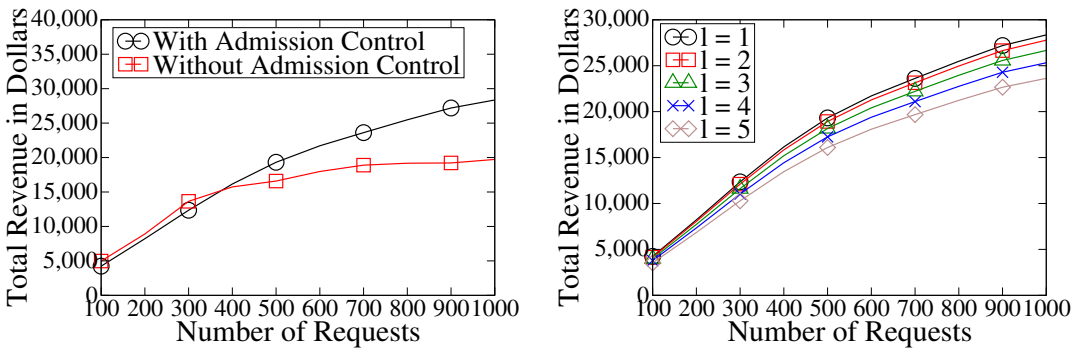
The rest is to investigate the impacts of both the admission control policy and the parameter α in Equation (5.9) on the performance of algorithm `ALG-1`. Figure 5.7 (a) shows the performance of algorithm `ALG-1` with and without adopting the admission control policy. It can be seen that algorithm `ALG-1` achieves a higher revenue if it does



(a) Total revenues by varying $n_{\max}$ for different numbers of requests

(b) Total revenues by varying $n_{\max}$ for different numbers of cloudlets

Figure 5.6: Performance impact of parameter $n_{\max}$ on algorithm `ALG-1`



(a) Total revenues by algorithm `ALG-1` with and without the admission control policy

(b) Total revenues by algorithm `ALG-1` with different α $(= 2^l \cdot K)$

Figure 5.7: Impacts of the admission control policy and the value of α on the performance of algorithm `ALG-1`

not adopt the admission control policy for the first four hundred requests. However, with more and more request arrivals, it achieves a higher revenue in the long term if the admission control policy is adopted. The rationale is that without any admission control policy, requests that consume excessive resources will be admitted if there are sufficient resources for them. Consequently, such resource allocations will heavily impact the admissions of future requests. As a result, the total revenue by algorithm `ALG-1` without the admission control policy is only two-thirds of that by itself with the admission control policy. Figure 5.7 (b) plots the performance curves of algorithm `ALG-1` by varying the value of α in Equation (5.9) from 2^1K to 2^5K, where K is the

number of cloudlets in the network. It can be seen from Figure 5.7 (b) that the larger the value of α, the less the total revenue delivered by `ALG-1` and vice versa. This is due to the fact that the larger the value of α, the higher the cost of using an overloaded resource will be, leading to more conservative resource usage.

5.8 Summary

In this chapter, we studied reliability-aware VNF instances provisioning in MEC, by casting a novel optimization problem. We first showed that the problem is NP-hard and formulated an integer linear program solution for it. We then proposed a logarithmic-approximation algorithm for the problem. Particularly for a special case of the problem where each request needs only one secondary VNF instance, we developed a constant approximation algorithm. Moreover, we proposed an exact algorithm for another special case of the problem where resource consumptions of different VNF instances are identical. We finally evaluated the performance of the proposed algorithms through experimental simulations. Experimental results demonstrated that the proposed algorithms are promising, and the empirical results delivered by the proposed algorithms outperform their analytical counterparts as theoretical estimation usually are very conservative.

Chapter 6

Conclusion and Future Works

> We can only see a short distance ahead, but we can see plenty there that needs to be done.
>
> *Computing Machinery and Intelligence*
>
> Alan Turing

This chapter summarizes the contributions we made in this thesis, followed by a discussion of potential research topics derived from this work.

6.1 Summary of Contributions

Efficient resource allocation for throughput maximization in the SDN- and NFV-based next-generation networks has been studied in this thesis. Novel concepts, models, and optimization techniques were proposed in order to enable efficient resource allocation and achieve high network throughput. We devised online algorithms with performance guarantees for the problem of dynamic admissions of a sequence of user requests arriving one by one without the knowledge of future arrivals. We proposed two algorithms for the problem of realizing user requests with specified network functions service chains. To tackle the problem of maximizing network throughput by dynamically scaling network resources while minimizing the operational cost of a network service provider, we presented the very first unified optimization framework for vertical scaling and horizontal scaling, while taking the Quality of Service (QoS)

requirements of user requests into account. We also developed approximation and exact algorithms for reliability-aware provisioning of VNF instances with the aim of maximizing the network throughput while satisfying the reliability requirement of each request and the capacity constraints on network resources. The proposed algorithms and associated algorithm design and analysis techniques may be of independent interests in many other areas, particularly in the combinatorial optimization domain.

The main contributions of this thesis are summarized as follows.

- We addressed the online request admission problem in software-defined networks with an objective of maximizing the network throughput, for which we proposed a novel admission cost model to accurately capture various resource consumptions. We also devised an online algorithm for admitting user requests, subject to TCAM capacities at SDN-enabled switches and bandwidth capacities at links in the network.

- We studied a novel resource allocation problem in the presence of user-specified network function requirements that aims to maximize the network throughput, subject to resource capacity and users QoS constraints. We explored the non-trivial trade-off between the accuracy/quality of a solution and the running time of the algorithm obtaining the solution by devising two algorithms, where the first algorithm has higher time complexity and delivers near-optimal solutions and the second algorithm delivers comparable solutions yet has much shorter running time.

- We formulated a novel optimization problem of maximizing the network throughput by dynamically scaling VNF instances while minimizing the operational cost of the network service provider, for which we first proposed a unified framework that jointly takes into account both vertical scaling and horizontal scaling, subject to the QoS requirements of user requests. We then performed non-trivial reductions to transform the problem to classic optimization problems, based

on which we designed an algorithm that has been empirically demonstrated to deliver a near-optimal solution.

- We investigated reliability-aware provisioning of VNF instances with the aim to maximize the network throughput, subject to user QoS requirements and resource capacities, for which we first formulated a novel optimization problem that generalizes various optimization problems. We then devised a logarithmic-factor approximation algorithm for the problem. For two special cases of the problem, we provided a constant approximation and an exact algorithm, respectively.

- We conducted extensive experiments by simulations, using both real and synthetic datasets to evaluate all proposed algorithms and investigate the impact of constraint parameters on their performance. Experimental results showed that the proposed algorithms outperform existing ones significantly in aspects of maximizing network throughput, minimizing operational costs, and meeting QoS requirements.

6.2 Future Works

There are several potential research topics that can be further explored based on the work in this thesis.

Firstly, we will investigate how emerging technologies for offloading interactive, computation-intensive tasks, e.g., Google's Project Stream [46], Valve's Steam Link [129], and Nvidia's GameStream [108], impact resource allocation. The concept of computation offloading, where a computation-intensive task such as face recognition is offloaded to a remote, powerful server for processing, instead of being executed locally, has been introduced for more than a decade [91], and cloud computing is one of the many forms of computation offloading [86]. Offloading delay-sensitive interactive applications, such as massively multiplayer online games, which technolo-

gies such as Google's Project Stream envision to solve, is still a novel concept. Such technologies often promise to enable an interactive computer game to be offloaded to remote servers or clusters of servers with abundant computing capacity to process computation-intensive tasks including updating the game state and rendering graphics of the game, whereas clients simply display the rendered video stream, cf. "dumb terminals" in the mainframe era [113]. This computing paradigm will have a profound influence on resource allocation as end-to-end delays play a critical role, particularly for virtual reality (VR) and augmented reality (AR) applications. John Carmack recommended a "motion-to-photons latency" of no more than 20 milliseconds [76] for VR systems, yet the network latency between Seattle and San Francisco alone is already 20 milliseconds [7]. This demands a resource allocation scheme that prioritizes both network throughput and stringent end-to-end delay requirements of these applications.

Secondly, we will study improving resource allocations through exploiting historical resource allocation traces. In this thesis, we formulated several resource allocation problems mathematically as optimization problems and solved them online with *instantaneous* user request and network information. Since the problems are NP-complete, the optimal solutions are very difficult to obtain in real time. With the increase in user QoS and end-to-end delay requirements, conventional methods are facing great challenges in designing more sophisticated resource allocation schemes to further improve system performance with scarce network resources. Inexpensive cloud storage makes it very easy to save the information as data on historical requests that were previously ignored and discarded. Using the similarities among user requests, the solutions of resource allocation in historical user requests can be exploited to improve the resource allocation of the current user request. More specifically, the solutions for resource allocation in historical user requests can be searched offline and stored in advance. When a user request arrives, it is not necessary to use conventional optimization methods to solve the resource allocation problem online. Instead,

we only need to compare the current user request with historical user requests and find the most similar one. Then, we use the solution of the most similar historical user requests to allocate network resources for the current user request. The offline characteristic makes it possible to use advanced cloud computing techniques to find optimal or near-optimal solutions of resource allocation for historical user requests, which can improve the performance of resource allocation accordingly.

Thirdly, we will research how to handle user mobility. Mobile devices, including mobile phones and tablets, are the dominating mean of computing for the majority of the population. These devices rely on Wi-Fi, Bluetooth, and cellar networks for Internet access, and these networks are characterized by intermittent network connectivity and volatile user locations. As a result of the intrinsic trait of high user mobility, provisioning seamless, uninterrupted services to delay-sensitive applications is vital. There are two techniques to cope with user mobility. One is to passively migrate services between servers, following user mobility trajectories. Specifically, when a user moves out from one area, the most suitable target area to accommodate the migrated service will be identified, and then a migration of service will be performed. Another is to proactively replicate services for users at some strategic locations to where users are likely to move, given their mobility profiles.

Fourthly, we will look at the optimization objectives from the perspective of users. This thesis focused on several optimization problems with objectives aligning with the interests of network service providers, e.g., the throughput and operational cost of a network. It is worth noting that the ecosystem consists of both network service providers and their customers, and the interests of customers do not necessarily overlap with those of network service providers. A plausible extension of this thesis is thus to take into account several optimization objectives that better aligns with the interests of users.

Fifthly, we will evaluate the performance increase provided by "almost shortest paths" [31]. Our proposed algorithms focus on finding shortest paths for one or a

group of user requests at a time. One possible extension to this is to identify the "almost shortest paths" [31]. There are two potential benefits: (i) finding shortest paths for individual users can introduce artificial network congestions and bottlenecks, limiting network capacity in accepting more users, and (ii) the lower computation complexity of identifying almost shortest paths is more versatile to suit situations where real-time responses are highly desirable. We will conduct both numerical and empirical analyses of the performance benefits brought by this extension.

Bibliography

[1] S. Agarwal, M. Kodialam, and T. V. Lakshman. "Traffic engineering in software defined networks". In: *2013 Proceedings IEEE INFOCOM*. 2013, pp. 2211–2219. DOI: 10.1109/INFCOM.2013.6567024.

[2] S. Agarwal, F. Malandrino, C. Chiasserini, and S. De. "Joint VNF Placement and CPU Allocation in 5G". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pp. 1943–1951. DOI: 10.1109/INFOCOM.2018.8485943.

[3] S. Aidi, M. F. Zhani, and Y. Elkhatib. "On Improving Service Chains Survivability Through Efficient Backup Provisioning". In: *2018 14th International Conference on Network and Service Management (CNSM)*. 2018, pp. 108–115.

[4] Amazon Web Services Inc. *AWS | Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting*. http://aws.amazon.com/ec2/. 2018.

[5] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat. "xOMB: Extensible Open MiddleBoxes with commodity servers". In: *2012 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. 2012, pp. 49–60.

[6] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. "On-line Routing of Virtual Circuits with Applications to Load Balancing and Machine Scheduling". In: *J. ACM* 44.3 (May 1997), pp. 486–504. DOI: 10.1145/258128.258201.

[7] AT&T. *U.S. Network Latency*. http://ipnetwork.bgtmo.ip.att.net/pws/network_delay.html. 2018.

[8] J. Bailey and S. Stuart. "FAUCET: Deploying SDN in the Enterprise". In: *ACM Queue* 14 Issue 5 (2016), pp. 54–68.

[9] S. Banerjee and K. Kannan. "Tag-In-Tag: Efficient flow table management in SDN switches". In: *10th International Conference on Network and Service Management (CNSM) and Workshop*. 2014, pp. 109–117. DOI: 10.1109/CNSM.2014.7014147.

[10] A.-L. Barabási and R. Albert. "Emergence of Scaling in Random Networks". In: *Science* 286.5439 (1999), pp. 509–512. DOI: 10.1126/science.286.5439.509. eprint: http://science.sciencemag.org/content/286/5439/509.full.pdf.

[11] M. T. Beck, J. F. Botero, and K. Samelin. "Resilient allocation of service Function chains". In: *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2016, pp. 128–133. DOI: 10.1109/NFV-SDN.2016.7919487.

[12] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. "Fog Computing and Its Role in the Internet of Things". In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. MCC '12. Helsinki, Finland: ACM, 2012, pp. 13–16. DOI: 10.1145/2342509.2342513.

[13] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. "Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN". In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. Hong Kong, China: ACM, 2013, pp. 99–110. DOI: 10.1145/2486001.2486011.

[14] C. Bu, X. Wang, M. Huang, and K. Li. "SDNFV-Based Dynamic Network Function Deployment: Model and Mechanism". In: *IEEE Communications Letters* 22.1 (2018), pp. 93–96. DOI: 10.1109/LCOMM.2017.2654443.

[15] K. Calvert and E. Zegura. *GT internetwork topology models (GT-ITM)*. 1996.

[16] Z. Cao, M. Kodialam, and T. V. Lakshman. "Traffic Steering in Software Defined Networks: Planning and Online Routing". In: DCC '14 (2014), pp. 65–70. DOI: 10.1145/2627566.2627574.

[17] F. Carpio, W. Bziuk, and A. Jukan. "Replication of Virtual Network Functions: Optimizing Link Utilization and Resource Costs". In: *CoRR* abs/1702.07151 (2017). arXiv: 1702.07151.

[18] F. Carpio and A. Jukan. "Balancing the Migration of Virtual Network Functions with Replications in Data Centers". In: *CoRR* abs/1705.05573 (2017). arXiv: 1705.05573.

[19] M. Casazza, P. Fouilhoux, M. Bouet, and S. Secci. "Securing virtual network function placement with high availability guarantees". In: *2017 IFIP Networking Conference (IFIP Networking) and Workshops*. 2017, pp. 1–9. DOI: 10.23919/IFIPNetworking.2017.8264850.

[20] A. Ceselli, M. Premoli, and S. Secci. "Mobile Edge Cloud Network Design Optimization". In: *IEEE/ACM Transactions on Networking* 25.3 (2017), pp. 1818–1831. DOI: 10.1109/TNET.2017.2652850.

[21] M. Charikar, C. Chekuri, T. yat Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. "Approximation Algorithms for Directed Steiner Problems". In: *Journal of Algorithms* 33.1 (1999), pp. 73 –91. DOI: https://doi.org/10.1006/jagm.1999.1042.

[22] M. Charikar, Y. Naamad, J. Rexford, and X. K. Zou. "Multi-commodity flow with in-network processing". In: *arXiv preprint arXiv:1802.09118* (2018).

[23] C.-H. Chi, J. Deng, and Y.-H. Lim. "Compression Proxy Server: Design and Implementation." In: *USENIX Symposium on Internet Technologies and Systems*. 1999.

[24] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz. "On the effect of forwarding table size on SDN network utilization". In: *IEEE INFOCOM 2014 - IEEE*

Conference on Computer Communications. 2014, pp. 1734–1742. DOI: 10.1109/INFOCOM.2014.6848111.

[25] R. Cohen, L. Katzir, and D. Raz. "An efficient approximation for the Generalized Assignment Problem". In: *Information Processing Letters* 100.4 (2006), pp. 162 –166. DOI: https://doi.org/10.1016/j.ipl.2006.06.003.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009.

[27] C Cui et al. "Network functions virtualisation: Network operator perspectives on industry progress. White Paper No. 3, Issue 1". In: *SDN and OpenFlow World Congress, Dusseldorf-Germany*. 2014.

[28] A. V. Dastjerdi and R. Buyya. "Fog Computing: Helping the Internet of Things Realize Its Potential". In: *Computer* 49.8 (2016), pp. 112–116. DOI: 10.1109/MC.2016.245.

[29] C. Demetrescu and I. Finocchi. "Combinatorial algorithms for feedback problems in directed graphs". In: *Information Processing Letters* 86.3 (2003), pp. 129 –136. DOI: https://doi.org/10.1016/S0020-0190(02)00491-X.

[30] W. Ding, H. Yu, and S. Luo. "Enhancing the reliability of services in NFV with the cost-efficient redundancy scheme". In: *2017 IEEE International Conference on Communications (ICC)*. 2017, pp. 1–6. DOI: 10.1109/ICC.2017.7996840.

[31] D. Dor, S. Halperin, and U. Zwick. "All-Pairs Almost Shortest Paths". In: *SIAM Journal on Computing* 29.5 (2000), pp. 1740–1759. DOI: 10.1137/S0097539797327908. eprint: https://doi.org/10.1137/S0097539797327908.

[32] A. Engelmann and A. Jukan. "A Reliability Study of Parallelized VNF Chaining". In: *2018 IEEE International Conference on Communications (ICC)*. 2018, pp. 1–6. DOI: 10.1109/ICC.2018.8422595.

[33] V. Eramo, M. Ammar, and F. G. Lavacca. "Migration Energy Aware Reconfigurations of Virtual Network Function Instances in NFV Architectures". In: *IEEE Access* 5 (2017), pp. 4927–4938. DOI: 10.1109/ACCESS.2017.2685437.

[34] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca. "An Approach for Service Function Chain Routing and Virtual Function Network Instance Migration in Network Function Virtualization Architectures". In: *IEEE/ACM Transactions on Networking* 25.4 (2017), pp. 2008–2025. DOI: 10.1109/TNET.2017.2668470.

[35] K. R. Fall and W. R. Stevens. *TCP/IP illustrated, volume 1: The protocols*. Addison-Wesley, 2011.

[36] J. Fan, C. Guan, Y. Zhao, and C. Qiao. "Availability-aware mapping of service function chains". In: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. 2017, pp. 1–9. DOI: 10.1109/INFOCOM.2017.8057153.

[37] J. Fan, M. Jiang, and C. Qiao. "Carrier-grade availability-aware mapping of Service Function Chains with on-site backups". In: *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. 2017, pp. 1–10. DOI: 10.1109/IWQoS.2017.7969152.

[38] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. "Enforcing Network-wide Policies in the Presence of Dynamic Middlebox Actions Using Flowtags". In: *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. NSDI'14. Seattle, WA: USENIX Association, 2014, pp. 533–546.

[39] X. Fei, F. Liu, H. Xu, and H. Jin. "Adaptive VNF Scaling and Flow Routing with Proactive Demand Prediction". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pp. 486–494. DOI: 10.1109/INFOCOM.2018.8486320.

[40] T. S. Ferguson. "Who Solved the Secretary Problem?" In: *Statist. Sci.* 4.3 (Aug. 1989), pp. 282–289. DOI: 10.1214/ss/1177012493.

[41] M. Furer and B. Raghavachari. "Approximating the Minimum-Degree Steiner Tree to within One of Optimal". In: *Journal of Algorithms* 17.3 (1994), pp. 409–423. DOI: https://doi.org/10.1006/jagm.1994.1042.

[42] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere. "Edge-centric Computing: Vision and Challenges". In: *SIGCOMM Comput. Commun. Rev.* 45.5 (Sept. 2015), pp. 37–42. DOI: 10.1145/2831347.2831354.

[43] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar. "Stratos: A Network-Aware Orchestration Layer for Middleboxes in the Cloud". In: *CoRR* abs/1305.0209 (2013). arXiv: 1305.0209.

[44] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. "OpenNF: Enabling Innovation in Network Function Control". In: *Proceedings of the 2014 ACM Conference on SIGCOMM*. SIGCOMM '14. Chicago, Illinois, USA: ACM, 2014, pp. 163–174. DOI: 10.1145/2619239.2626313.

[45] Google. *Partnering toward the next generation of mobile networks*. https://blog.google/topics/internet-access/partnering-toward-next-generation-mobile-networks/. 2017.

[46] Google Inc. *Project Stream - Google*. https://projectstream.google.com/. 2018.

[47] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat. "Evolve or Die: High-Availability Design Principles Drawn from Google's Network Infrastructure". In: 2016.

[48] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett. "SDX: A Software Defined Internet Exchange". In: *Proceedings of the 2014 ACM Conference on SIGCOMM*.

SIGCOMM '14. Chicago, Illinois, USA: ACM, 2014, pp. 551–562. DOI: 10.1145/2619239.2626300.

[49] A. Gushchin, A. Walid, and A. Tang. "Scalable Routing in SDN-enabled Networks with Consolidated Middleboxes". In: *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*. HotMiddlebox '15. London, United Kingdom: ACM, 2015, pp. 55–60. DOI: 10.1145/2785989.2785999.

[50] B. Han, V. Gopalakrishnan, G. Kathirvel, and A. Shaikh. "On the Resiliency of Virtual Network Functions". In: *IEEE Communications Magazine* 55.7 (2017), pp. 152–157. DOI: 10.1109/MCOM.2017.1601201.

[51] T. He, H. Khamfroush, S. Wang, T. L. Porta, and S. Stein. "It's Hard to Share: Joint Service Placement and Request Scheduling in Edge Clouds with Sharable and Non-Sharable Resources". In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 2018, pp. 365–375. DOI: 10.1109/ICDCS.2018.00044.

[52] S. Herker, X. An, W. Kiess, S. Beker, and A. Kirstaedter. "Data-Center Architecture Impacts on Virtualized Network Functions Service Chain Embedding with High Availability Requirements". In: *2015 IEEE Globecom Workshops (GC Wkshps)*. 2015, pp. 1–7. DOI: 10.1109/GLOCOMW.2015.7414158.

[53] Hewlett-Packard Development Company. *L.P. Servers for enterprise bladeSystem, rack & tower and hyperscale*. http://www8.hp.com/us/en/products/servers/. 2015.

[54] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. "Is It Still Possible to Extend TCP?" In: *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*. IMC '11. Berlin, Germany: ACM, 2011, pp. 181–194. DOI: 10.1145/2068816.2068834.

[55] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. "Achieving high utilization with software-driven WAN". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 15–26.

[56] C.-Y. Hong, S. Mandal, M. Al-Fares, M. Zhu, R. Alimi, K. N. B., C. Bhagat, S. Jain, J. Kaimal, S. Liang, K. Mendelev, S. Padgett, F. Rabe, S. Ray, M. Tewari, M. Tierney, M. Zahn, J. Zolla, J. Ong, and A. Vahdat. "B4 and After: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-defined WAN". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '18. Budapest, Hungary: ACM, 2018, pp. 74–87. DOI: 10.1145/3230543.3230545.

[57] F. Hu, Q. Hao, and K. Bao. "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation". In: *IEEE Communications Surveys Tutorials* 16.4 (2014), pp. 2181–2206. DOI: 10.1109/COMST.2014.2326417.

[58] H. Huang, S. Guo, P. Li, W. Liang, and A. Y. Zomaya. "Cost Minimization for Rule Caching in Software Defined Networking". In: *IEEE Transactions on Parallel and Distributed Systems* 27.4 (2016), pp. 1007–1016. DOI: 10.1109/TPDS.2015.2431684.

[59] H. Huang, S. Guo, J. Wu, and J. Li. "Joint middlebox selection and routing for software-defined networking." In: *ICC*. 2016, pp. 1–6.

[60] L.-H. Huang, H.-C. Hsu, S.-H. Shen, D.-N. Yang, and W.-T. Chen. "Multicast traffic engineering for software-defined networks". In: *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*. IEEE. 2016, pp. 1–9.

[61] L.-H. Huang, H.-J. Hung, C.-C. Lin, and D.-N. Yang. "Scalable steiner tree for multicast communications in software-defined networking". In: *arXiv preprint arXiv:1404.3454* (2014).

[62] M. Huang, W. Liang, Z. Xu, and S. Guo. "Efficient Algorithms for Throughput Maximization in Software-Defined Networks With Consolidated Middleboxes". In: *IEEE Transactions on Network and Service Management* 14.3 (2017), pp. 631–645. DOI: 10.1109/TNSM.2017.2725240.

[63] M. Huang, W. Liang, Z. Xu, and S. Guo. "Efficient Algorithms for Throughput Maximization in Software-Defined Networks With Consolidated Middleboxes". In: *IEEE Transactions on Network and Service Management* 14.3 (2017), pp. 631–645. DOI: 10.1109/TNSM.2017.2725240.

[64] M. Huang, W. Liang, Z. Xu, and S. Guo. "Efficient Algorithms for Throughput Maximization in Software-Defined Networks With Consolidated Middleboxes". In: *IEEE Transactions on Network and Service Management* 14.3 (2017), pp. 631–645. DOI: 10.1109/TNSM.2017.2725240.

[65] M. Huang, W. Liang, Z. Xu, M. Jia, and S. Guo. "Throughput Maximization in Software-Defined Networks with Consolidated Middleboxes". In: *2016 IEEE 41st Conference on Local Computer Networks (LCN)*. 2016, pp. 298–306. DOI: 10.1109/LCN.2016.58.

[66] M. Huang, W. Liang, Z. Xu, W. Xu, S. Guo, and Y. Xu. "Online unicasting and multicasting in software-defined networks". In: *Computer Networks* 132 (2018), pp. 26 –39. DOI: https://doi.org/10.1016/j.comnet.2017.12.011.

[67] R. Jain and S. Paul. "Network virtualization and software defined networking for cloud computing: a survey". In: *IEEE Communications Magazine* 51.11 (2013), pp. 24–31. DOI: 10.1109/MCOM.2013.6658648.

[68] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. "B4: Experience with a globally-deployed software defined WAN". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 3–14.

[69] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, and W. Kellerer. "Interfaces, attributes, and use cases: A compass for SDN". In: *IEEE Communications Magazine* 52.6 (2014), pp. 210–217.

[70] M. Jia, W. Liang, M. Huang, Z. Xu, and Y. Ma. "Routing Cost Minimization and Throughput Maximization of NFV-Enabled Unicasting in Software-Defined Networks". In: *IEEE Transactions on Network and Service Management* 15.2 (2018), pp. 732–745. DOI: 10.1109/TNSM.2018.2810817.

[71] M. Jia, W. Liang, M. Huang, Z. Xu, and Y. Ma. "Routing Cost Minimization and Throughput Maximization of NFV-Enabled Unicasting in Software-Defined Networks". In: *IEEE Transactions on Network and Service Management* 15.2 (2018), pp. 732–745. DOI: 10.1109/TNSM.2018.2810817.

[72] M. Jia, W. Liang, M. Huang, Z. Xu, and Y. Ma. "Throughput Maximization of NFV-Enabled Unicasting in Software-Defined Networks". In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. 2017, pp. 1–6. DOI: 10.1109/GLOCOM.2017.8254756.

[73] M. Jia, W. Liang, Z. Xu, M. Huang, and Y. Ma. "QoS-Aware Cloudlet Load Balancing in Wireless Metropolitan Area Networks". In: *IEEE Transactions on Cloud Computing* (2018), pp. 1–1. DOI: 10.1109/TCC.2017.2786738.

[74] M. Jia, W. Liang, and Z. Xu. "QoS-Aware Task Offloading in Distributed Cloudlets with Virtual Network Function Services". In: *Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*. MSWiM '17. Miami, Florida, USA: ACM, 2017, pp. 109–116. DOI: 10.1145/3127540.3127561.

[75] Y. Jia, C. Wu, Z. Li, F. Le, and A. Liu. "Online Scaling of NFV Service Chains Across Geo-Distributed Datacenters". In: *IEEE/ACM Transactions on Networking* 26.2 (2018), pp. 699–710. DOI: 10.1109/TNET.2018.2800400.

[76] John Carmack. *John Carmack's Latency Mitigation Strategies*. https://www.twentymilliseconds.com/post/latency-mitigation-strategies/. 2013.

[77] J. Kang, O. Simeone, and J. Kang. "On the Trade-Off Between Computational Load and Reliability for Network Function Virtualization". In: *IEEE Communications Letters* 21.8 (2017), pp. 1767–1770. DOI: 10.1109/LCOMM.2017.2698040.

[78] Y. Kanizo, O. Rottenstreich, I. Segall, and J. Yallouz. "Optimizing Virtual Backup Allocation for Middleboxes". In: *IEEE/ACM Transactions on Networking* 25.5 (2017), pp. 2759–2772. DOI: 10.1109/TNET.2017.2703080.

[79] Y. Kanizo, D. Hay, and I. Keslassy. "Palette: Distributing tables in software-defined networks". In: *INFOCOM, 2013 Proceedings IEEE*. IEEE. 2013, pp. 545–549.

[80] K. Kar, M. Kodialam, T. V. Lakshman, and L. Tassiulas. "Routing for network capacity maximization in energy-constrained ad-hoc networks". In: *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*. Vol. 1. 2003, 673–681 vol.1. DOI: 10.1109/INFCOM.2003.1208717.

[81] N. Katta, O. Alipourfard, J. Rexford, and D. Walker. "Infinite CacheFlow in Software-defined Networks". In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. HotSDN '14. Chicago, Illinois, USA: ACM, 2014, pp. 175–180. DOI: 10.1145/2620728.2620734.

[82] K. Kawashima, T. Otoshi, Y. Ohsita, and M. Murata. "Dynamic placement of virtual network functions based on model predictive control". In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. 2016, pp. 1037–1042. DOI: 10.1109/NOMS.2016.7502957.

[83] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. "The internet topology zoo". In: *IEEE Journal on Selected Areas in Communications* 29.9 (2011), pp. 1765–1775.

[84] J. Kong, I. Kim, X. Wang, Q. Zhang, H. C. Cankaya, W. Xie, T. Ikeuchi, and J. P. Jue. "Guaranteed-Availability Network Function Virtualization with Network Protection and VNF Replication". In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. 2017, pp. 1–6. DOI: 10.1109/GLOCOM.2017.8254730.

[85] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. "Software-defined networking: A comprehensive survey". In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76.

[86] K. Kumar and Y. Lu. "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?" In: *Computer* 43.4 (2010), pp. 51–56. DOI: 10.1109/MC.2010.98.

[87] J. Kuo, S. Shen, M. Yang, D. Yang, M. Tsai, and W. Chen. "Service Overlay Forest Embedding for Software-Defined Cloud Networks". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 720–730. DOI: 10.1109/ICDCS.2017.62.

[88] T. Kuo, B. Liou, K. C. Lin, and M. Tsai. "Deploying Chains of Virtual Network Functions: On the Relation Between Link and Server Usage". In: *IEEE/ACM Transactions on Networking* 26.4 (2018), pp. 1562–1576. DOI: 10.1109/TNET.2018.2842798.

[89] J. Li, W. Liang, M. Huang, and X. Jia. "Providing reliability-aware virtualized network function services for mobile edge computing." In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 2019, pp. 1–10.

[90] Y. Li, L. T. X. Phan, and B. T. Loo. "Network functions virtualization with soft real-time guarantees". In: *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*. IEEE. 2016, pp. 1–9.

[91] Z. Li, C. Wang, and R. Xu. "Computation Offloading to Save Energy on Handheld Devices: A Partition Scheme". In: *Proceedings of the 2001 International*

Conference on Compilers, Architecture, and Synthesis for Embedded Systems. CASES '01. Atlanta, Georgia, USA: ACM, 2001, pp. 238–246. DOI: 10.1145/502217.502257.

[92] W. Liang and Y. Liu. "Online Data Gathering for Maximizing Network Lifetime in Sensor Networks". In: *IEEE Transactions on Mobile Computing* 6.1 (2007), pp. 2–11. DOI: 10.1109/TMC.2007.250667.

[93] W. Liang and X. Guo. "Online Multicasting for Network Capacity Maximization in Energy-Constrained Ad Hoc Networks". In: *IEEE Transactions on Mobile Computing* 5.9 (2006), pp. 1215–1227. DOI: 10.1109/TMC.2006.133.

[94] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu. "On Dynamic Service Function Chain Deployment and Readjustment". In: *IEEE Transactions on Network and Service Management* 14.3 (2017), pp. 543–553. DOI: 10.1109/TNSM.2017.2711610.

[95] T. Lukovszki, M. Rost, and S. Schmid. "It's a Match!: Near-Optimal and Incremental Middlebox Deployment". In: *SIGCOMM Comput. Commun. Rev.* 46.1 (Jan. 2016), pp. 30–36. DOI: 10.1145/2875951.2875956.

[96] T. Lukovszki and S. Schmid. "Online Admission Control and Embedding of Service Chains". In: *Post-Proceedings of the 22Nd International Colloquium on Structural Information and Communication Complexity - Volume 9439*. SIROCCO 2015. Montserrat, Spain: Springer-Verlag, 2015, pp. 104–118. DOI: 10.1007/978-3-319-25258-2_8.

[97] Y. Ma, W. Liang, and J. Wu. "Online NFV-enabled multicasting in mobile edge cloud networks." In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 2019, pp. 1–10.

[98] Y. Ma, W. Liang, and Z. Xu. "Online Revenue Maximization in NFV-Enabled SDNs". In: *2018 IEEE International Conference on Communications (ICC)*. 2018, pp. 1–7. DOI: 10.1109/ICC.2018.8422333.

[99] Y. Ma, W. Liang, Z. Xu, and S. Guo. "Profit Maximization for Admitting Requests with Network Function Services in Distributed Clouds". In: *IEEE Transactions on Parallel and Distributed Systems* 30.5 (2019), pp. 1143–1157. DOI: 10.1109/TPDS.2018.2874257.

[100] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. "ClickOS and the Art of Network Function Virtualization". In: *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, 2014, pp. 459–473.

[101] M. Mechtri, C. Ghribi, and D. Zeghlache. "A Scalable Algorithm for the Placement of Service Function Chains". In: *IEEE Transactions on Network and Service Management* 13.3 (2016), pp. 533–546. DOI: 10.1109/TNSM.2016.2598068.

[102] Microsoft. *Plan network requirements for Skype for business.* https://technet.microsoft.com/en-us/library/gg425841.aspx. 2015.

[103] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba. "Network Function Virtualization: State-of-the-Art and Research Challenges". In: *IEEE Communications Surveys Tutorials* 18.1 (2016), pp. 236–262. DOI: 10.1109/COMST.2015.2477041.

[104] T. Mishra and S. Sahni. "PETCAM – A Power Efficient TCAM Architecture for Forwarding Tables". In: *IEEE Transactions on Computers* 61.1 (2012), pp. 3–17. DOI: 10.1109/TC.2011.84.

[105] G. Moualla, T. Turletti, and D. Saucez. "An Availability-aware SFC placement Algorithm for Fat-Tree Data Centers". In: *2018 IEEE International Conference on Cloud Networking (CloudNet)*. 2018, pp. 1–6.

[106] NTT Communications. *NTT communications and jba's sdn project wins ibc 2013 innovation award.* https://www.ntt.com/aboutus_e/news/data/20130918.html. 2013.

[107] B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti. "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks". In: *IEEE Communications Surveys Tutorials* 16.3 (2014), pp. 1617–1634. DOI: 10.1109/SURV.2014.012214.00180.

[108] NVIDIA Corporation. *NVIDIA GameStream | Play PC Games on NVIDIA SHIELD*. https://www.nvidia.com/en-us/shield/games/gamestream/. 2018.

[109] S. Plotkin. "Competitive routing of virtual circuits in ATM networks". In: *IEEE Journal on Selected Areas in Communications* 13.6 (1995), pp. 1128–1136. DOI: 10.1109/49.400667.

[110] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. "SIMPLE-fying Middlebox Policy Enforcement Using SDN". In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. Hong Kong, China: ACM, 2013, pp. 27–38. DOI: 10.1145/2486001.2486022.

[111] L. Qu, C. Assi, K. Shaban, and M. J. Khabbaz. "A Reliability-Aware Network Service Chain Provisioning With Delay Guarantees in NFV-Enabled Enterprise Datacenter Networks". In: *IEEE Transactions on Network and Service Management* 14.3 (2017), pp. 554–568. DOI: 10.1109/TNSM.2017.2723090.

[112] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield. "Split/Merge: System Support for Elastic Execution in Virtual Middleboxes". In: *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. nsdi'13. Lombard, IL: USENIX Association, 2013, pp. 227–240.

[113] E. S. Raymond. *The Art of UNIX Programming*. Pearson Education, 2003.

[114] B. Ren, D. Guo, G. Tang, X. Lin, and Y. Qin. "Optimal Service Function Tree Embedding for NFV Enabled Multicast". In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 2018, pp. 132–142. DOI: 10.1109/ICDCS.2018.00023.

[115] G. Sallam, G. R. Gupta, B. Li, and B. Ji. "Shortest Path and Maximum Flow Problems Under Service Function Chaining Constraints". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pp. 2132–2140. DOI: 10.1109/INFOCOM.2018.8485996.

[116] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye. "Provably efficient algorithms for joint placement and allocation of virtual network functions". In: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. 2017, pp. 1–9. DOI: 10.1109/INFOCOM.2017.8057036.

[117] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. "The Case for VM-Based Cloudlets in Mobile Computing". In: *IEEE Pervasive Computing* 8.4 (2009), pp. 14–23. DOI: 10.1109/MPRV.2009.82.

[118] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. "Design and Implementation of a Consolidated Middlebox Architecture". In: *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012, pp. 323–336.

[119] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. "Making middleboxes someone else's problem: network processing as a cloud service". In: *ACM SIGCOMM Computer Communication Review* 42.4 (2012), pp. 13–24.

[120] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat. "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network". In: *Sigcomm '15*. 2015.

[121] E. Spitznagel, D. Taylor, and J. Turner. "Packet classification using extended TCAMs". In: *11th IEEE International Conference on Network Protocols, 2003. Proceedings.* 2003, pp. 120–131. DOI: 10.1109/ICNP.2003.1249762.

[122] N. Spring, R. Mahajan, and D. Wetherall. "Measuring ISP topologies with Rocketfuel". In: *ACM SIGCOMM Computer Communication Review* 32.4 (2002), pp. 133–145.

[123] J. W. Suurballe. "Disjoint paths in a network". In: *Networks* 4.2 (1974), pp. 125–145. DOI: 10.1002/net.3230040204. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230040204.

[124] J. W. Suurballe and R. E. Tarjan. "A quick method for finding shortest pairs of disjoint paths". In: *Networks* 14.2 (1984), pp. 325–336. DOI: 10.1002/net.3230140209. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230140209.

[125] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck. "Mobile Edge Computing Potential in Making Cities Smarter". In: *IEEE Communications Magazine* 55.3 (2017), pp. 38–43. DOI: 10.1109/MCOM.2017.1600249CM.

[126] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck. "Mobile Edge Computing Potential in Making Cities Smarter". In: *IEEE Communications Magazine* 55.3 (2017), pp. 38–43. DOI: 10.1109/MCOM.2017.1600249CM.

[127] A. S. Tanenbaum and D. J. Wetherall. *Computer Networks*. 5th. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010.

[128] A. Tomassilli, F. Giroire, N. Huin, and S. Pérennes. "Provably Efficient Algorithms for Placement of Service Function Chains with Ordering Constraints". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pp. 774–782. DOI: 10.1109/INFOCOM.2018.8486275.

[129] Valve Corporation. *Steam Link*. https://store.steampowered.com/steamlink/about. 2018.

[130] V. V. Vazirani. *Approximation Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2001.

[131] P. Veitch, M. J. McGrath, and V. Bayon. "An instrumentation and analytics framework for optimal and robust NFV deployment". In: *IEEE Communications Magazine* 53.2 (2015), pp. 126–133.

[132] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser. "Service Entity Placement for Social Virtual Reality Applications in Edge Computing". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pp. 468–476. DOI: 10.1109/INFOCOM.2018.8486411.

[133] X. Wang, C. Wu, F. Le, A. Liu, Z. Li, and F. Lau. "Online VNF Scaling in Datacenters". In: *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. 2016, pp. 140–147. DOI: 10.1109/CLOUD.2016.0028.

[134] Z. Wang, W. Liang, M. Huang, and Y. Ma. "Delay-Energy Joint Optimization for Task Offloading in Mobile Edge Computing". In: *CoRR* abs/1804.10416 (2018). arXiv: 1804.10416.

[135] J. Xia, Z. Cai, and M. Xu. "Optimized Virtual Network Functions Migration for NFV". In: *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. 2016, pp. 340–346. DOI: 10.1109/ICPADS.2016.0053.

[136] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie. "A Survey on Software-Defined Networking". In: *IEEE Communications Surveys Tutorials* 17.1 (2015), pp. 27–51. DOI: 10.1109/COMST.2014.2330903.

[137] J. Xu, L. Chen, and P. Zhou. "Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pp. 207–215. DOI: 10.1109/INFOCOM.2018.8485977.

[138] Z. Xu, W. Liang, M. Huang, M. Jia, S. Guo, and A. Galis. "Approximation and Online Algorithms for NFV-Enabled Multicasting in SDNs". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 625–634. DOI: 10.1109/ICDCS.2017.43.

[139] Z. Xu, W. Liang, M. Huang, M. Jia, S. Guo, and A. Galis. "Efficient NFV-Enabled Multicasting in SDNs". In: *IEEE Transactions on Communications* 67.3 (2019), pp. 2052–2070. DOI: 10.1109/TCOMM.2018.2881438.

[140] Z. Xu, W. Liang, M. Jia, M. Huang, and G. Mao. "Task Offloading with Network Function Requirements in a Mobile Edge-Cloud Network". In: *IEEE Transactions on Mobile Computing* (2018), pp. 1–1. DOI: 10.1109/TMC.2018.2877623.

[141] Z. Xu, W. Liang, M. Jia, M. Huang, and G. Mao. "Task Offloading with Network Function Requirements in a Mobile Edge-Cloud Network". In: *IEEE Transactions on Mobile Computing* (2018), pp. 1–1. DOI: 10.1109/TMC.2018.2877623.

[142] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo. "Efficient Algorithms for Capacitated Cloudlet Placements". In: *IEEE Transactions on Parallel and Distributed Systems* 27.10 (2016), pp. 2866–2880. DOI: 10.1109/TPDS.2015.2510638.

[143] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang. "A comprehensive survey of Network Function Virtualization". In: *Computer Networks* 133 (2018), pp. 212–262. DOI: https://doi.org/10.1016/j.comnet.2018.01.021.

[144] S. Yi, C. Li, and Q. Li. "A Survey of Fog Computing: Concepts, Applications and Issues". In: *Proceedings of the 2015 Workshop on Mobile Big Data*. Mobidata '15. Hangzhou, China: ACM, 2015, pp. 37–42. DOI: 10.1145/2757384.2757397.

[145] Q. Zhang, Y. Xiao, F. Liu, J. C. S. Lui, J. Guo, and T. Wang. "Joint Optimization of Chain Placement and Request Scheduling for Network Function Virtualization". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 731–741. DOI: 10.1109/ICDCS.2017.232.

[146] S. Q. Zhang, Q. Zhang, A. Tizghadam, B. Park, H. Bannazadeh, R. Boutaba, and A. Leon-Garcia. "Sector: TCAM Space Aware Routing on SDN". In: *2016 28th International Teletraffic Congress (ITC 28)*. Vol. 01. 2016, pp. 216–224. DOI: 10.1109/ITC-28.2016.138.