Virtual Service Provisioning for Internet of Things Applications in Mobile Edge Computing

Jing Li

February 2022

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

To my parents.

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 80%. This research is partially funded by the Australian Research Council under its Discovery Project Scheme with Grant No. DP200101985.

Jing Li
26 February 2022

Acknowledgments

This thesis is completed with the help and support of many people. I would like to express my gratitude to them.

First and foremost, I would like to express my sincerest gratitude towards my supervisor, Professor Weifa Liang, for his patience and expert guidance during my PhD study. He taught me how to become a qualified researcher with methodical supervision and warm encouragement. He also guided me to handle challenges and difficulties in my research pursuit. I am indebted to him for his unremitting effort in training me in the scientific field.

I would like to thank the other members of my supervisor panel, Dr. Yu Lin and Professor Xiaohua Jia, for their professional guidance and advice. I also would like to thank Professor Song Guo and Professor Wanlei Zhou for their valuable collaboration.

I want to thank the Australian National University for the financial support through awarding me the PhD scholarship to undertake this study. This research is partially funded by the Australian Research Council under its Discovery Project Scheme with Grant No. DP200101985.

I wish to thank the people in the School of Computing at the Australian National University, for their generous help and assistance in various aspects.

I am eternally grateful to my friends, Yu Ma, Meitian Huang, Mike Jia, Zichuan Xu, Wenzheng Xu, Haotian Chang, Junyi Xu, Yuchen Li et al. for their kind help and friendship.

Finally, I want to express my heartfelt gratitude towards my parents, Dexiang Li and Lixia Liu, for their continuous support and encouragement. Without their love and encouragement, it is impossible to finish this thesis.

Publications

Journal

1. **J. Li**, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo. Throughput maximization of delay-aware DNN inference in edge computing by exploring DNN model partitioning and inference parallelism. To appear in *IEEE Transactions on Mobile Computing*, 2021, doi: 10.1109/TMC.2021.3125949.

2. **J. Li**, W. Liang, W. Xu, Z. Xu, X. Jia, W. Zhou, and J. Zhao. Maximizing user service satisfaction for delay-sensitive IoT applications in edge computing. *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 5, pp. 1199 – 1212, 2022.

3. **J. Li**, W. Liang, Z. Xu, X. Jia, and W. Zhou. Service provisioning for multi-source IoT applications in mobile edge computing. *ACM Transactions on Sensor Networks*, vol. 18, no. 2, article. 17, pp. 17:1 – 17:25, 2022.

4. **J. Li**, W. Liang, and Y. Ma. Robust service provisioning with service function chain requirements in mobile edge computing. *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 2138 – 2153, 2021.

5. **J. Li**, W. Liang, M. Huang and X. Jia. Reliability-aware network service provisioning in mobile edge-cloud networks. *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 7, pp. 1545 – 1558, 2020.

6. Y. Ma, W. Liang, **J. Li**, X. Jia, and S. Guo. Mobility-aware and delay-sensitive service provisioning in mobile edge-cloud networks. *IEEE Transactions on Mobile Computing*, vol. 21, no. 1, pp. 196 – 210, 2022.

Conference

1. **J. Li**, W. Liang, Y. Li, Z. Xu, and X. Jia. Delay-aware DNN inference throughput maximization in edge computing via jointly exploring partitioning and parallelism. *Proc. of LCN'21*, IEEE, 2021.

2. **J. Li**, W. Liang, M. Chen, and Z. Xu. Mobility-aware dynamic service placement in D2D-assisted MEC environments. *Proc. of WCNC'21*, IEEE, 2021.

3. **J. Li**, W. Liang, Z. Xu, and W. Zhou. Provisioning virtual services in mobile edge computing for IoT applications with multiple sources. *Proc. of LCN'20*, IEEE, 2020.

4. **J. Li**, W. Liang, W. Xu, Z. Xu, and J. Zhao. Maximizing the quality of user experience of using services in edge computing for delay-sensitive IoT applications. *Proc. of MSWiM'20*, ACM, 2020.

5. **J. Li**, W. Liang, M. Huang, and X. Jia. Providing reliability-aware virtualized network function services for mobile edge computing. *Proc. of ICDCS'19*, IEEE, 2019.

6. Y. Li, W. Liang, and **J. Li**. Profit maximization for service placement and request assignment in edge computing via deep reinforcement learning. *Proc. of MSWiM'21*, ACM, 2021.

7. M. Chen, W. Liang, and **J. Li**. Energy-efficient data collection maximization for UAV-assisted wireless sensor networks. *Proc. of WCNC'21*, IEEE, 2021.

8. S. Lin, W. Liang, and **J. Li**. Reliability-aware service function chain provisioning in mobile edge-cloud networks. *Proc. of ICCCN'20*, IEEE, 2020.

Abstract

The Internet of Things (IoT) paradigm is paving the way for many new emerging technologies, such as smart grid, industry 4.0, connected cars, smart cities, etc. Mobile Edge Computing (MEC) provides promising solutions to reduce service delays for delay-sensitive IoT applications, where cloudlets (edge servers) are deployed in the proximity of IoT devices. With the advent of Network Function Virtualization (NFV) technology, more and more mobile users use virtual network services in MEC networks. Such services usually have specified service requirements such as Service Function Chains (SFCs), where an SFC is a sequence of Virtual Network Functions (VNFs) to form a full-scale network service. Meanwhile, the surging of deep learning brings new vigour and vitality to shape the prospect of IoT, and edge intelligence arises to provision real-time Deep Neural Network (DNN) inference services for users. To accelerate the DNN inference of a request in an MEC network, the DNN inference model can be partitioned into two parts: one is processed on the local IoT device of the request; and the other is processed on a cloudlet in the MEC network. Also, the DNN inference can be further accelerated by allocating multiple threads in a cloudlet in which the request is assigned. Furthermore, the resource demands during the implementation of a request are dynamically evolving. This uncertainty of resource demands at different execution stages of the request does impact the service quality and the profit of network service providers.

In this thesis, we will focus on virtual service provisioning for IoT applications in MEC environments. Considering the unique characteristics of MEC and IoT, it poses several important challenges. That is, (1) how to maximize the accumulative user satisfaction on the use of the IoT services provided by the MEC, (2) how to minimize the operational cost of service provisioning for multi-source IoT applications while meeting their SFC requirements, (3) how to maximize the number of delay-aware DNN service requests admitted, by accelerating each DNN inference through jointly exploring DNN partitioning and inference parallelism, and (4) how to maximize the expected profit collected by the network service provider of an MEC network, under the uncertainty assumption of both computing resource and data rate demands. To this end, in this thesis we will address the challenges. We achieve the following contributions.

Firstly, we consider the user satisfaction problem of using services jointly provided by an MEC network and a remote cloud for delay-sensitive IoT applications, through maximizing the accumulative user satisfaction when different user services have different service delay requirements. A novel metric to measure user satisfaction of using a service is proposed, and efficient approximation and online algorithms for the defined problems under both static and dynamic user service demands are then devised and analyzed.

Secondly, we study service provisioning in an MEC network for multi-source IoT applications with SFC requirements with the aim of minimizing service provisioning cost, where each multi-source IoT application has multiple data streams from different sources to be uploaded to the MEC network for processing and storage, while each data stream must pass through the network functions of the SFC of the IoT application, prior to reaching its destination. A service provisioning framework for such multi-source IoT applications is proposed, through uploading stream data from multiple IoT sources, VNF instance placement and sharing, in-network aggregation of data streams, and workload balancing among cloudlets. Efficient algorithms for service provisioning of multi-source IoT applications in MEC networks, built upon the proposed framework, are also proposed.

Thirdly, we investigate a novel DNN inference throughput maximization problem in an MEC network with the aim to maximize the number of delay-aware DNN service requests admitted, by accelerating each DNN inference through jointly exploring DNN partitioning and inference parallelism. We devise a constant approximation algorithm for the problem under the offline setting, and an online algorithm with a provable competitive ratio for the problem under the online setting, respectively.

Fourthly, we address a robust SFC placement problem with the aim to maximize the expected profit collected by the service provider of an MEC network, under the assumption of both computing resource and data rate demand uncertainties. We start with a special case of the problem where the measurement of the expected demanded resources for each request admission is accurate, under which we propose a near-optimal approximation algorithm for the problem by adopting the Markov approximation technique, which can achieve a provable optimality gap. Then, we extend the proposed approach to the problem of concern, for which we show that the proposed algorithm still is applicable, and the solution delivered has a moderate optimality gap with bounded perturbation errors on the profit measurement.

Finally, we summarize the thesis work and explore several potential research topics that are based on the studies in this thesis.

Contents

List of Figures

Chapter 1

Introduction

In this chapter, we first introduce Mobile Edge Computing (MEC), Internet of Things (IoT), Network Functions Virtualization (NFV), and edge intelligence. We then introduce the system architecture considered in this thesis. We also survey the related research topics on virtual service provisioning for IoT applications in MEC environments, through identifying the key problems and the related challenges in each research topic. We finally state the contributions made in this thesis, and provide the organization of the thesis.

1.1 Mobile Edge Computing

Cloud computing, introduced around 2007, has imbued industry and academia with flexible and high-quality computing platform services, including Hardware as a Service (HaaS), Software as a Service (SaaS), Data as a Service (DaaS), and Infrastructure as a Service (IaaS) [99]. Cloud computing provisions network services to users and empowers end devices with limited resources, through centralizing abundant computing resources and storage resources in the clouds [71]. Cloud computing has pushed the rapid development of many global IT giants. For example, Amazon's business for cloud services has shown tremendous profits, which has made Amazon a leader in the cloud computing business [101].

Nowadays, mobile applications have become an indispensable part of the current generation of people, driven by the technological advancement of mobile devices. The increasing popularity of memory-hungry and computation-intensive mobile terminals has imposed an uttermost effect on cellular and wireless networks, promoting the development and advancement of Mobile Cloud Computing (MCC) technology in mobile environments [48]. In other words, MCC extends cloud computing and focuses on network service provisioning in mobile environments by running the mobile applications on the clouds. The MCC technique has several merits, such as enabling compute-intensive and storage-intensive applications on mobile devices, and prolonging the battery lifetime of mobile devices through task offloading to the clouds. Nevertheless, traditional cloud infrastructures usually suffer from some difficulties to guarantee high Quality of Services (QoS) in mobile environments, in terms of long latency, vulnerable security, and high bandwidth demands. The major concern of the service provisioning in MCC lies in the long propagation delay due to the long distance between the users and the remote clouds, which significantly degrades the service quality and triggers challenges for service provisioning in mobile networks. Therefore, the MCC technique is not suitable for pervasive latency-critical mobile applications, especially those required to initiate intermediate response actions. It is widely agreed that the cloud computing platform is inadequate to optimize the service latency in millisecond-scale for the next-generation networks [2].

To handle the hailed challenges by MCC, Mobile Edge Computing (MEC), borne out of the European Telecommunications Standards Institute (ETSI), is the state-of-the-art paradigm shift after the MCC systems to place edge servers (cloudlets) with moderate capacities at the close locations of mobile users (the edge of the core network) to deliver user-centered network services [41]. The MEC paradigm provides a viable and promising solution to significantly alleviate the latency and jitter experienced by the end-users within their proximate access for both network services and data access [62]. In the context of MEC, the cloudlets have been employed at the network edges to provision seamless cloud services to mobile users without the hassle of traversing the core network [59]. Meanwhile, the core network facilitates the resource sharing among the distributed cloudlets to improve the system performance.

The emerging MEC platform is constructed based on recent techniques, such as Information-Centric Networking (ICN), Software-Defined Networking (SDN), and Network Function Virtualization (NFV). In particular, the MEC is characterized by the white paper from ETSI as follows [78].

- **On-Premises:** The distributed edge servers process the data locally, and send the important information to the remote cloud (or another edge server) for further processing and storage if necessary. In other words, the on-premises MEC platforms can not only run in an isolated manner and retain the data privacy, but also have access to the rest network resources.

- **Proximity:** The edge servers cover the network users in the local vicinity and enable highly-responsive cloud services, improving user experience and collecting information for big data analytics.

- **Lower Latency:** The MEC platforms are established at the proximate locations of the mobile devices to diminish the service latency and alleviate the network congestion.

- **Location Awareness:** The MEC platforms are able to generate location information of mobile devices through leveraging low-level signalling. The location awareness of MEC enhances network management and facilitates handling the mobility of mobile devices in mobile environments.

- **Network Context Information:** Besides location awareness, MEC also has the context awareness to make edge servers to gather network context information from mobile devices, which will be utilized to make efficient offloading decisions and enable context-related service provisioning.

1.2 Internet of Things

The Internet of Things (IoT) paradigm permeates our modern lives in every aspect, and defines a system of billions of interrelated pervasive objects to collect and exchange data, bridging the gap between the physical and cyber worlds. The number of connected devices is foreseen to increase from 700 million to 3.2 billion by 2023 [19]. The IoT architecture is endowed with the power to create a system for interrelated computing and communication among an immense number of real-world objectives (things). At the same time, massive raw data are produced by the IoT devices and analyzed to extract high-level knowledge through adopting data mining approaches.

The geographically distributed IoT devices sense real-time information to observe the physical world, and generate the data streams in large volume and high velocity [16; 93]. Therefore, the data collected from IoT devices need to be analyzed in real time such that the hidden patterns and insights can be extracted in almost no time. Along with the proliferation of the delay-critical and resource-hungry IoT applications, such as online gaming, facial recognition, Virtual Reality (VR), Augmented Reality (AR), and unmanned vehicles, the IoT devices have the natural resource limitation and encounter noticeable challenges to meet the increasing computation demands and the stringent delay requirements for such real-time IoT applications [85]. Conventional IoT devices usually transmit their data to remote clouds for storage and processing [60]. Nevertheless, due to the remoteness of clouds and increasingly congested core networks, it incurs prohibitive long transmission delays and leads to the violation of the real-time data processing requirements of many IoT applications. Also, it consumes much bandwidth resource and energy cost to transmit a large volume of data to the remote cloud [93]. Thus, conventional clouds may not be suitable for delay-sensitive IoT applications [88].

The Mobile Edge Computing (MEC) concept emerges to pull the cloud services from the remote clouds to the network edges to mitigate the experienced network delay. Thus, the IoT devices can make use of the computing resource at the network edges, while the edge servers, rather than the remote clouds, locally collect and analyze the raw data generated by the IoT devices. The MEC paradigm empowers the IoT architecture to improve the efficiency of IoT data processing and remarkably alleviate the workload on the core network.

1.3 Network Function Virtualization

With the advance of mobile communication technology and mobile device fabrications, more and more mobile devices, including mobile phones, tablets, and various sensors, are deployed for business, entertainment, social networking, smart cities, and the Internet of Things (IoT) [50; 51]. Nowadays, mobile users have an increasing demand for diverse and agile network services with high quality [113]. Traditional network service providers provide services to their users with Network Functions (NFs) that are implemented on dedicated middleboxes, such as Network Address

Translator (NAT), firewalls, Intrusion Detection System (IDS) [43]. However, the deployment of various middleboxes for different NFs usually incurs high instantiation expenses, complex management, and weak extensibility [123]. This is because the dedicated middleboxes require manual configuration and management, and the network service providers have to make utilization of a large variety of dedicated middleboxes to offer a series of network services to users. Therefore, the traditional NFs based on dedicated middleboxes face the challenges of fulfilling the ever-growing QoS demands of users, especially in terms of the CAPital EXpenditures (CAPEX) and OPerational EXpenditures (OPEX) [112].

Network Function Virtualization (NFV) is introduced as a promising technology to implement Virtualized Network Functions (VNFs) as software running on Virtual Machines (VMs), through replacing the dedicated hardware applications [82]. Decoupling the NFs from the hardware, the NFV technology enables network service providers to dynamically scale up or scale down their service provisioning to cope with the dynamic user demands [61]. Especially, the benefits of the NFV technique are as follows [6].

- **Independence:** With the NFV technique, the software is decoupled from the hardware, therefore, the software and hardware evolve independently from each other.
- **Flexibility:** The network service provider adopts the NFV technique to adapt to the dynamic user demands with various deployment options through software updating, and avoids unnecessary hardware updating, thereby facilitating the flexible network service provisioning.
- **Scalability:** The NFV technique improves the service quality by flexibly scaling the VNFs with finer granularity to cater to variations of user requests and meet the changing user demands. This is because a single server is able to run several VMs, the capacities of which are easily scaled up and down to improve service efficiency based on the resources.
- **Security:** Security has been a challenge of concern in network service provisioning. A network service provider manages and maintains the network, while granting the users to run the requested services in a virtual space beside a firewall within the network.

The NFV framework was standardized by ETSI in [28], and an example of a high-level NFV framework is illustrated in Figure 1.1. Then the main architectural components of the NFV framework are identified as follows.

- **NFV Infrastructure (NFVI):** This component consists of the virtual resources, virtualization layer, and hardware resources to build a base environment for the execution of the virtual network services.

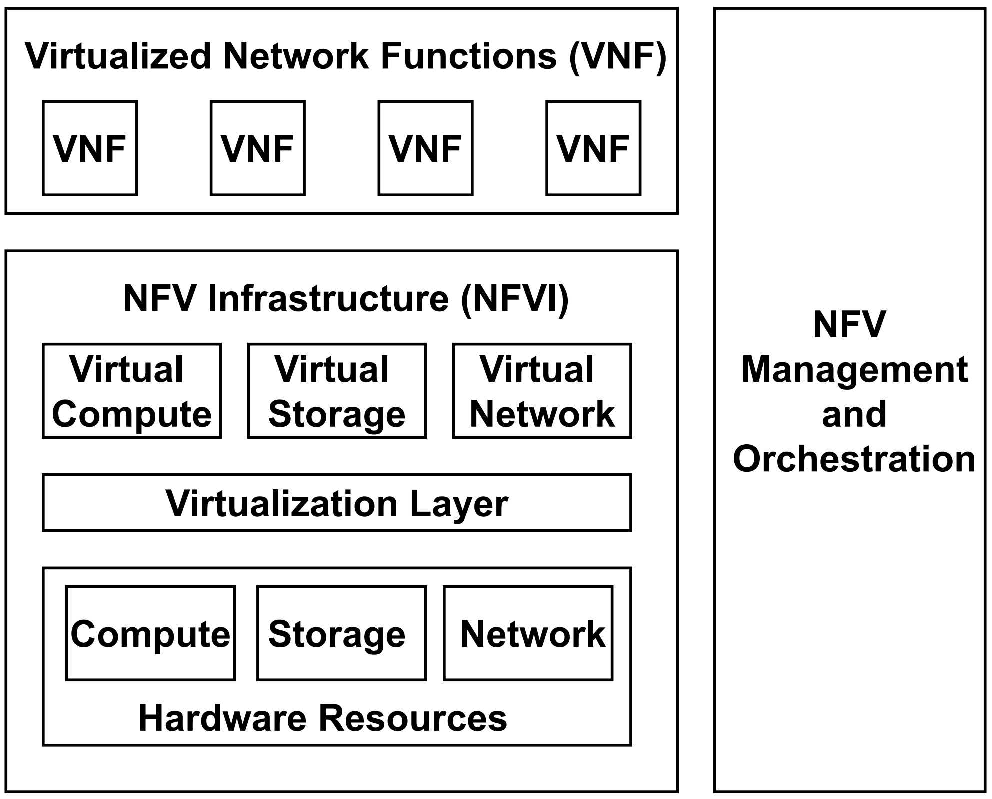


Figure 1.1: An illustrative example of a high-level NFV framework [28].

- **Virtualized Network Functions (VNFs):** The VNFs are the software implementations of NFs running on the VMs, providing the same functionality and external operation interfaces of physical NFs.

- **NFV Management and Orchestration:** This component consists of NFV Orchestrator (NFVO), VNF Manager (VNFM) and Virtualized Infrastructure Manager (VIM). Especially, NFVO is responsible for the lifecycle management of network services, VNFM manages the lifecycle of VNFs, and VIM coordinates the resource allocation for NFVI and manages the interaction between NFVI and VNFs.

In the context of NFV, a full-scale network service is implemented by a sequence of VNFs running on generic servers, known as a Service Function Chain (SFC) [112]. For each mobile user request, a specific SFC is requested to be coordinated and embedded into some physical nodes (cloudlets) in an MEC network. The SFC directs the data traffic flow of the request to pass through the listed VNFs in their specified order through automatic interconnection and resource allocation [29]. Figure 1.2 illustrates an example of an SFC consisting of the VNFs: NAT, firewall and IDS.

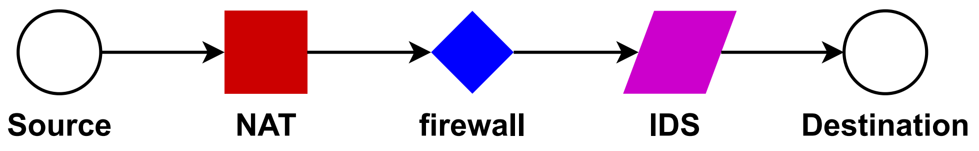


Figure 1.2: An illustrative example of an SFC consisting of NAT, firewall, and IDS, and the data traffic flow traverses the VNFs of the SFC in order.

1.4 Edge Intelligence

Thanks to the rapid development of Artificial Intelligence (AI), nowadays, AI technique has the advantage in capturing insights to make strategic decisions and collect business benefits in various fields, exemplified by computer vision, natural language processing, board games, and robotics [122]. It is widely recognized that the AI-enabled applications will promote the building of smart homes and smart cities to unleash the collective intelligence of the society, thereby preserving citizens' overall quality of life.

The progress of deep learning in the last decade has unlocked the potential to learn deep features of data through leveraging Deep Neural Networks (DNNs) [23]. Meanwhile, it injects new vitality into the computation-intensive deep learning applications by deploying a series of hardware accelerators, such as graphics processing units (GPUs) and tensor processing units (TPUs), due to the high computing demands of DNN training and inference [102]. Benefiting from the breakthroughs of deep learning and the related hardware improvement, the AI technique is becoming an appealing solution to make swift and agile data analysis locally for a large amount of data generated by the IoT devices, rather than sending the data to the remote cloud for analysis. This promotes the marriage of AI and MEC techniques to give rise to edge intelligence. Namely, the MEC technique is moving towards a new paradigm of edge intelligence, through applying AI techniques to process the data generated by IoT devices and applications in many fields, such as facial recognition, traffic monitoring, and speech recognition [122].

Edge intelligence, an emerging paradigm in its early stage currently, interweaves a wide spectrum of technologies and concepts to integrate AI and MEC to achieve intelligent IoT service provisioning, including intelligent task offloading to edge servers, intelligent edge server collaboration, and intelligent data analysis [65]. As suggested by the work from [23], we distinguished edge intelligence into two categories: *AI for edge* and *AI on edge*.

- **AI for edge:** This studies how to efficiently adopt the AI technologies to solve the constrained optimization problems in MEC environments, thereby granting edge with intelligence.

- **AI on edge:** The AI modules and applications are usually compute-intensive, while the edge is endowed with limited computing resource. We then need to design an efficient framework for both model training and inference on edge to fulfil the QoS requirements.

1.5 System Architecture

In this thesis, we consider an MEC network, which consists of a set of Access Points (APs) located on a geographical area (e.g., a metropolitan area). The APs are situated at different locations, such as shopping malls, parks, libraries and restaurants. Co-located with some of the APs, there is a set of cloudlets, each of which possesses limited resource capacities for task offloading and service implementation. Each cloudlet and its co-located AP are connected by a high-speed optical cable, and the delay between them is assumed to be negligible. A mobile user then can offload its task to a cloudlet through connecting to an AP within its transmission coverage. Figure 1.3 illustrates an MEC network that consists of six APs and three cloudlets co-located with APs.

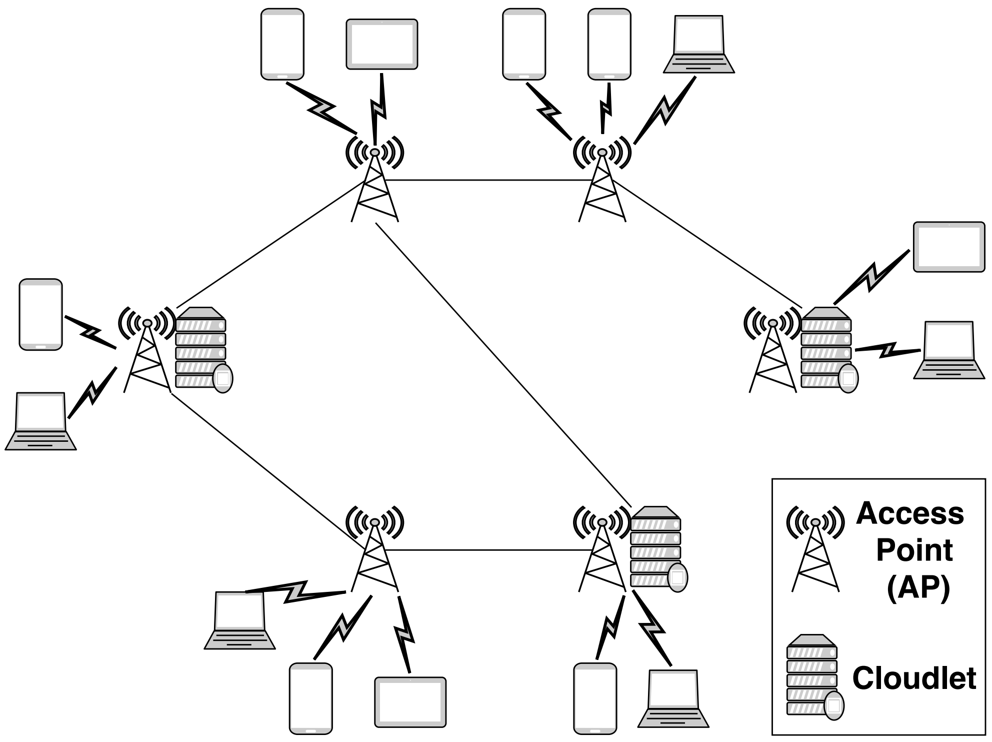


Figure 1.3: An illustrative example of an MEC network consisting of six APs and three cloudlets co-located with APs.

1.6 Research Topics

Fueled by the surging demands of mobile users, Mobile Edge Computing (MEC) is emerging as a promising technology to bring mobile users to the edge of networks to immensely cut down the propagation delays in the evolution to 5G and beyond 5G networks. The rapid development of the Internet of Things (IoT) has promoted the proliferation of numerous mobile applications, including Augmented Reality (AR), Virtual Reality (AR), mobile games, and surveillance, which rely on edge devices (EDs) such as laptops, tablets, and smartphones [51]. This thesis focuses on the following four main research topics:

(1) How to maximize the total user experience of using the services provided by the MEC network, where a user satisfaction is inversely proportional to the extra service delay beyond the user's delay threshold;

(2) How to minimize the total operational cost of implementing multi-source IoT applications in an MEC network, where each multi-source IoT application has multiple data streams from different sources to be uploaded to a location (a cloudlet) in the MEC network for aggregation, processing and storage;

(3) How to maximize the number of Deep Neural Network (DNN) inference requests admitted while meeting their inference delay requirements, through jointly exploring DNN model partitioning and inference parallelism;

(4) How to maximize the expected profit collected by the network service provider of an MEC network through robust Service Function Chain (SFC) placement, under the uncertainty assumption of both computing resource and data rate demands in the implementation of user requests;

These topics are interrelated with each other. Topic (1) designs a novel model to quantify a user satisfaction of using a service provided, and proposes efficient algorithms to provide high efficiency and seamless user experience. Because the operational cost to provide such high-quality network services is a matter of concern, in the sequel, Topic (2) strives to optimize the network performance by minimizing the total operational cost of the service provisioning for multi-source IoT applications in a general case. With edge intelligence arising to provision real-time DNN inference services for users, Topic (3) studies the joint adoption of DNN model partitioning and inference parallelism in maximizing the number of DNN inference requests admitted, while meeting the delay requirement of each request. So far, Topics (1), (2) and (3) assume that the amounts of different resources demanded by each request are given and do not change during the execution of the request. Topic (4) considers the uncertainty assumption of both computing resource and data rates demanded by request executions, and devises an efficient algorithm to maximize the expected profit collected by the network service provider of an MEC network.

1.6.1 Maximizing User Service Satisfaction for Delay-Sensitive IoT Applications

With the emergence of complicated and resource-hungry mobile applications in the Internet of Things (IoT) and smart cities, implementing user tasks on cloudlets of an MEC network becomes an important approach to shorten service response delays, reduce the energy consumption of mobile devices, and improve the user experience of using services ultimately. One fundamental problem of a network service provider is how to maximize the user service satisfaction. We consider an integrated platform consisting of the remote cloud and a set of local cloudlets forming an MEC network for IoT service provisioning. The IoT users offload their tasks to the integrated MEC platform for processing with different service delay requirements. In this thesis, we aim to develop efficient approximation and online algorithms for delay-sensitive service provisioning for IoT applications in the integrated MEC platform. This poses the following challenges.

For a set of offloading task requests, which requests should be assigned to a local cloudlet and which ones should be assigned to the remote cloud for processing, considering the heterogeneity of both computing resource and processing capability of cloudlets and the remote cloud; how to assign different requests to different cloudlets or the remote cloud such that the average user experience of using the services provided by the platform is maximized while keeping the workload among all cloudlets as balanced as possible, where a user satisfaction is inversely proportional to the extra service delay beyond the user's delay threshold; and how to develop a cost model to quantify a user satisfaction of using a service provided by the platform.

Task offloading in MEC networks has been extensively studied in recent years. Most existing work focused on minimizing the energy consumption of mobile devices or the end-to-end delay of a task execution, through partitioning a task into two parts: one part is offloaded to the cloudlets in the MEC for execution and another part is processed by the mobile device itself. Most task offloading concentrated on such a single task offloading.

There are extensive investigations on admitting a set of requests with the aim to minimize the average service delay of offloaded tasks. For example, Gouareb *et al.* [33] considered the problem of minimizing the total service delay of implementation of a service function chain in MECs, by proposing a heuristic. Huang *et al.* [38] considered the delay-sensitive service placement and migration in an MEC network by presenting heuristic algorithms for the problem. Jia *et al.* [42] studied task offloading in an MEC network with the aim to minimize the average delay of all admitted requests, by incorporating queuing delays at both Access Points (APs) and cloudlets. Lyu *et al.* [67] investigated the joint optimization of both task admission decisions and efficient resource allocation to minimize the total energy consumption in an MEC network while the delay requirements of mobile devices are met. They proposed a task admission approach to achieve the asymptotic optimality by pre-admitting resource-restrained mobile devices. Xia [104] considered a set of delay-aware tasks to be offloaded to an MEC network with the aim to minimize the service cost of offloaded tasks. They provided a cost model, and proposed a heuristic for the problem based on the built cost model. Xu *et al.* [109] studied delay-aware service placement in MEC to minimize the operational cost, by assuming that the specified delay of each request cannot be violated, and they developed an approximation algorithm for the problem. Xu *et al.* [108] also considered the delay-aware service offloading problem in MEC with each request having a specific service requirement. They aimed to minimize the service cost through Virtual Network Function (VNF) instance placement, sharing, and migration, by developing online algorithms for request admissions. They, however, did not include the remote cloud as an alternative processing source.

Although the aforementioned investigations on delay-aware task offloading have been extensively studied in the past several years, only a handful of studies take into account service provisioning in MEC platforms for delay-sensitive IoT applications. For example, Arisdakessian *et al.* [8] adopted game theory and designed preference functions for IoT devices and edge nodes based on several metrics. By adopting the preference functions, they developed centralized and distributed algorithms for the assignment of services of IoT devices to edge nodes to minimize the IoT service delay and execution makespan. Alameddine *et al.* [3] studied a joint task offloading for application assignment and resource allocation in an MEC network, by formulating a dynamic task offloading and scheduling problem. They developed a mixed integer linear programming solution and a heuristic solution for the problem. Samanta *et al.* [83] developed a dynamic microservice provisioning scheme for IoT devices in MEC environments, by formulating a novel model that incorporates both the service delay and service price into consideration. Song *et al.* [90] investigated the task assignment for IoT applications in an MEC network while meeting the Quality of Service (QoS) requirement of each task. They developed an efficient heuristic algorithm for the problem. They proposed a heuristic algorithm to achieve efficient network resource utilization for each microservice admission. Xu *et al.* [106] considered the operational cost minimization problem for implementing IoT applications with Service Function Chain (SFC) requirements, by focusing on IoT application placement in an MEC network by developing both randomized and heuristic placement algorithms. Yu *et al.* [115] studied the problem of IoT service provisioning with the objective to meet computing, network bandwidth and QoS requirements of each IoT application. They devised approximation algorithms to deal with the IoT service provisioning under various scenarios, respectively.

Unlike the aforementioned works focusing on either the cost minimization problem or the delay-aware service placement problem in MEC networks, here we consider a set of offloading task requests from IoT devices with different service delay requirements, in which all requests must be served by either cloudlets in an MEC network or a remote cloud. We aim to maximize the accumulative user satisfaction of using the services provided by the integrated platform of the MEC network and the remote cloud. We first devise a novel metric to measure user satisfaction of using a service. We then propose efficient approximation and online algorithms for the defined problem under both static and dynamic user service demands when the bandwidth capacity constraint is negligible. We also present efficient heuristic algorithms for the problem with the bandwidth capacity constraint.

1.6.2 Service Provisioning for Multi-Source IoT Applications

Most IoT applications usually impose Service Function Chain (SFC) enforcement on their data transmission, where each data packet from its source gateway of an IoT device to the destination (a cloudlet) of the IoT application must pass through each Virtual Network Function (VNF) in the SFC in an MEC network. However, little attention has been paid to such a service provisioning of multi-source IoT applications in an MEC network with SFC enforcement. It poses the following challenges to enable efficient service provisioning in an MEC network for multi-source IoT applications with SFC requirements.

First, the provisioning of an IoT service requires the joint consideration of many complicated data processing procedures, such as uploading sensory data streams from multiple sources through different gateways to a single destination, data stream aggregation and routing in the MEC network, and the aggregated data streams are processed by the VNF instances of network functions in the SFC. To minimize computing and bandwidth resource usages, a routing tree rooted at the destination cloudlet and spanning each source (access point) of an IoT application is built, instead of uploading the data stream of each source to the destination separately. The uploaded data streams are aggregated at non-leaf tree nodes, and appropriate numbers of the VNF instances of service functions should also be deployed at the tree nodes for streaming data processing. Building such a routing tree involves a non-trivial interplay between VNF instance placement, aggregation node selections, and routing path selections, which is the first fundamental challenge. The computing and bandwidth resources in an MEC network are usually very precious and costly. The operational cost of service provisioning for multi-source IoT applications depends on not only various resource consumptions but also workload balancing among cloudlets of the MEC network. How to minimize the operational cost of service provisioning while meeting the SFC requirements of IoT applications poses the second challenge. The data streams of each IoT application are allowed to be merged or aggregated at intermediate nodes of the routing tree before reaching their destination. Also, VNFs for such data merging and aggregation usually can be shared among the data streams from different gateways of the IoT application. The last challenge is how to further reduce the operational cost of service provisioning of IoT applications through VNF sharing. In this thesis, we will address the aforementioned challenges.

There are a few investigations of service provisioning of NFV-enabled network services for IoT applications in MEC networks. For example, Song *et al.* [90] considered the QoS-based task allocation in MEC for IoT applications by proposing efficient algorithms. However, they did not incorporate the SFC requirement into consideration. Yu *et al.* [115] studied the problem of IoT service provisioning with the objective to meet computing, bandwidth and QoS requirements of an IoT application. They, however, did not consider the processing of IoT data traffic with SFC enforcement. Mouradian *et al.* [73] proposed an architecture of NFV and SDN-based distributed IoT gateways for large-scale disaster management. They, however, did not focus on operational cost minimization and workload balancing. Xu *et al.* [110] studied the QoS-aware VNF placement of SFCs in MEC for one-source IoT applications. They considered the operational cost minimization problem for the implementation of IoT applications with SFC requirements, and concentrated on IoT application placement in MEC by proposing randomized and heuristic algorithms. Although we also deal with the operational cost minimization problem of service provisioning for multi-source IoT applications, the problem in this study is essentially different from the mentioned work [110] as follows. We here deal with multi-source IoT applications to minimize the operational cost of service provisioning, where data streams from different sources need to be uploaded, aggregated, and processed in the MEC, while the demanded number of NFV instances of each service function in the SFC needs to be placed in cloudlets, and the workload among cloudlets needs to be balanced. We aim to construct a data routing tree for each multi-source IoT application such that the operational cost is minimized, and the work in [110] considered a single SFC chain placement, where there is no data stream aggregation, data routing tree construction, and workload balancing among cloudlets.

Closely related to the problem studied here are the NFV-enabled unicast and multicast problems. There are extensive studies of user unicast and multicast request admissions through resource provisioning and allocations in MEC networks [4; 13; 30; 42; 44; 69; 70; 81; 91; 109; 118]. For example, Jia *et al.* [42] considered assigning user requests to different cloudlets in a Wireless Metropolitan Area Network with the objective of minimizing the maximum delay among requsts. They then developed heuristics for the problem. They [44] also studied workload balancing among cloudlets to reduce the maximum response delay of user requests. Xu *et al.* [109] devised approximation algorithms to offload user requests to different cloudlets under different conditions efficiently. Xia *et al.* [104] considered opportunistic task offloading under link bandwidth, mobile device energy, and cloudlet computing capacity constraints. Ma *et al.* [70] studied dynamically admitting NFV-enabled unicast requests with QoS requirements with the aim to maximize the profit in MEC. They developed an efficient heuristic and an online algorithm for the problem without the QoS requirement. Although they considered the sharing of existing VNF instances among different unicast requests. It can be seen that the problem of NFV-enabled unicast request admissions in [70] is a special case of the NFV-enabled multicast request admissions where the destination set contains only one node. Recently, there have been several studies extending the NFV-enabled unicast routing to NFV-enabled multicast routing in MEC environments. For example, Alhussein *et al.* [4] investigated the embedding of a multicast request with both computing and bandwidth resource requirements to a 5G core substrate network, by enabling multi-path routing between two consecutive VNFs of an SFC. Ceselli *et al.* [13] considered the design optimization such as the VM placement and migration, and user request assignment, by formulating a Mixed Integer Linear Programming (MILP) solution and heuristic algorithms for the problem. Feng *et al.* [30] proposed an algorithm with a performance guarantee for placing VNFs in distributed cloud networks and routing service flows among the placed VNFs under the constraints of SFCs of the requests. Xu *et al.* [107] considered the cost minimization of admitting a single NFV-enabled multicast request with the QoS requirement in MEC, where the implementation of the SFC of each request is consolidated to a single cloudlet. They aim to minimize the admission cost by placing no more than constant numbers of VNF instances of the SFC of the request in different branches of the found pseudo-multicast tree for the request. Ma *et al.* [69] studied admissions of NFV-enabled multicasting requests under both static and dynamic admission scenarios, by proposing approximation and online algorithms for the problems with provable performance guarantees. Soni *et al.* [91] proposed a scalable multicast group management scheme and a workload balancing method for the routing of best-effort traffic and bandwidth-guaranteed traffic. Zhang *et al.* [118] investigated the NFV-enabled multicasting problem in SDNs. They assumed that there are sufficient computing and bandwidth resources to accommodate all multicast requests, for which they provided a 2-approximation algorithm if only one server is deployed for implementing the SFC of each multicast request. Unfortunately, the applicability of their method is very limited and cannot be extended for the general case where multiple servers are employed. Ren *et al.* [81] also considered request admissions of NFV-enabled multicasting under the end-to-end delay constraint by developing both approximation and heuristic algorithms for the problem.

Although there are some similarities between the cost minimization problem for multi-source IoT applications studied here and the NFV-enabled multicasting problem in MEC [69; 81], due to the fact that they both aim to build a routing tree for their solutions, the main difference between them lies in the following. In the multicasting case, a packet from the source (the tree root) will be broadcast to all destinations (leaves) such that a VNF instance of each service function in the SFC is installed along the path from the root to each leaf. In contrast, in the routing tree for a multi-source IoT application, all leaves are data sources. The data streams will converge to the tree root, and the data streams from the children of a non-leaf node in the tree will be aggregated at the node. The number of VNF instances instantiated at a tree node in the path from a leaf to the tree root is determined by the accumulative volume of the data streams at the node, while the accumulative volumes of data streams at different tree nodes are different. Thus, the construction of such a data routing tree for each multi-source IoT application is much more challenging, because the volume of the data stream at each tree node is not fixed, which is determined by the number of children and the data volume of each of the children. Also, the VNF instance placement must meet certain restrictions while keeping the workload among involving cloudlets balanced.

Different from these mentioned studies, here we study the multi-source IoT application service provisioning in an MEC environment. We aim to minimize the operational cost of service provisioning while balancing the workloads among cloudlets in an MEC network. We then propose efficient algorithms to deal with the cost minimization problem for a single multi-source IoT application and a set of multi-source

IoT applications, respectively.

1.6.3 Throughput Maximization of Delay-Aware DNN Inference

Mobile Edge Computing (MEC) delivers a promising solution to delay-aware task offloading services, complementary to traditional cloud computing. With the emerging edge intelligence, it has become a hot topic to provide real-time DNN inference services in MEC environments. For accelerating DNN inference, we consider a widely adopted DNN inference acceleration method – the DNN partitioning method, where the DNN inference model can be partitioned into two parts: one is processed on the local IoT device of the request; and another is processed on a cloudlet in the MEC network. We also adopt the inference parallelism method to leverage multiple threads in a cloudlet for processing the offloaded DNN model part. In this thesis, we jointly adopt DNN partitioning and inference parallelism methods for accelerating DNN inference to meet the inference delay requirements of users. The delay-aware DNN inference service provisioning thus poses the following challenges.

For a DNN inference request with a trained DNN model and a given inference delay requirement, how to determine which cloudlet in an MEC network to accommodate the request? How to partition the DNN model between its local IoT device and its assigned cloudlet such that the inference delay meets its inference delay requirement? How many threads in its assigned cloudlet should be allocated for processing the offloaded DNN part while meeting the inference delay requirement of the request? In this chapter, we will address these challenges, and devise performance-guaranteed approximation and online algorithms for the DNN inference throughput maximization problem in an MEC network under both offline and online request arrival settings, respectively.

The delay-aware task offloading in MEC environments has drawn much attention in recent years. For example, Bozorgchenani *et al.* [12] proposed an evolutionary algorithm to address a task offloading problem in an MEC network, with the aim to minimize not only the task processing delay but also the energy consumption of the processing. Liu *et al.* [64] dealt with the stochastic arrivals of heterogeneous tasks in a three-layer MEC network, and developed an online algorithm to shorten the inference delay of each task processing.

Recently, edge intelligence arises to provision real-time DNN inference services for users and much effort focused on investigating DNN inference accelerations through task offloading in MEC environments. Mohammed *et al.* [72] devised a novel DNN partitioning scheme in an MEC network, and applied the matching theory to distribute the DNN parts to edge servers, with the aim to minimize the total computation time. Tang *et al.* [95] considered the admissions of DNN inference requests of multiple users with the assistance of an edge server to minimize the maximum end-to-end delay among users, by developing an efficient algorithm to achieve the optimal solution. Xu *et al.* [111] investigated the DNN inference offloading in an MEC network, assuming that each requested DNN has been partitioned. They then provided a randomized algorithm and an online algorithm to minimize the total energy consumption and deal with the real-time request admissions, respectively. Zeng *et al.* [116] introduced the cooperation of multiple edge devices with heterogeneous computing capacities for DNN inference, and studied the dynamic DNN workload partitioning and the workload assignment optimization over the edge devices, with the aim to minimize system energy consumption. Kang *et al.* [45] proposed a promising DNN partitioning strategy between a mobile device and a cloud to optimize the experienced inference service delay and the energy consumption, based on the DNN layer granularity. Their strategy, however, only works for chain-topology DNNs. Hu *et al.* [36] studied the DNN partitioning problem in an integrated network consisting of edge servers and a cloud by proposing an optimal DNN partitioning strategy DSL, with the aim to minimize the delay for processing one video frame when the network workload is light. They reduced the problem to a minimum cut problem. Unfortunately, the optimal partitioning claim is suspicious, which will be detailed later.

There are existing approaches for multi-threading DNN inference acceleration, e.g., adopting the frameworks such as TensorFlow [1] and PyTroch [77]. For example, Liu *et al.* [66] designed a customized thread pool for Convolutional Neural Network (CNN) inference through multi-threading on multi-core central processing units (CPUs). Nori *et al.* [76] leveraged a multi-level cache hierarchy to improve the performance of inference parallelism with multi-core CPUs. Furthermore, the resource allocation problem under DNN inference parallelism has also been investigated by researchers [75; 105]. Niu *et al.* [75] proposed a novel framework to execute a DNN model on mobile CPUs and graphics processing units (GPUs) with thread-level parallelism, by adopting a pruning-based model compression technique. Xiang *et al.* [105] included the multi-threading on multi-core CPUs with the assistance of GPUs. They presented a pipeline-based real-time DNN inference framework to deliver an efficient scheduling of CPU and GPU resources. However, their frameworks [75; 105] cannot be applied in this work, since important constraints in the problem such as the inference delay requirements of requests have not been considered.

In comparison with existing work, here we deal with the delay-aware DNN inference service provisioning in an edge computing environment under both offline and online settings of request arrivals. We jointly perform DNN partitioning, allocate offloaded tasks to different cloudlets, and explore inference parallelism through deploying multiple threads. We aim to maximize the number of DNN inference requests admitted while meeting their inference delay requirements, subject to computing capacities on cloudlets. Specifically, we consider the problem under both offline and online request arrival settings: a set of DNN inference requests is given in advance, and a sequence of DNN inference requests arrives one by one without the knowledge of future arrivals, respectively. We then devise a novel constant approximation algorithm for the problem under the offline setting. We also propose an online algorithm with a provable competitive ratio for the problem under the online setting.

1.6.4 Robust Service Provisioning with Service Function Chain Requirements

Service robustness is an important QoS indicator for IoT service provisioning in MEC. Most existing studies considered resource allocation and scheduling in MEC for user request admissions, under the assumption that the amounts of different resources demanded by each request are given *a prior* and do not change during the execution of the request. In practice, the resource demands during the implementation of a request are dynamically evolving, which substantially impacts the service quality and the profit of network service providers. Thus, providing robust services to users against their resource demand uncertainties is a critical issue. In this thesis, we assume that the demanded computing resource and data rate of a request are different at its different execution stages. To provide robust IoT services with SFC requirements in MEC, it poses the following three challenges.

(1) *Optimization of SFC placements*: The requested VNF instances are first instantiated into cloudlets under the cloudlet capacity constraint. Then, VNF instances will be chained in the specified order in the SFC, subject to both the link capacity constraint and the latency requirement of the request. (2) *Uncertain resource demands*: During the request execution period, the request may demand more resources than the ones initially allocated to it, then the scheduling will be infeasible, and an admitted request may never be fully executed. Consequently, the network service provider will earn much less profits due to underestimating the resource demands of each admitted request. (3) *Inaccurate measurements*: As the RSFCP problem depends heavily on real-time measurements of system features in the MEC network (e.g., dynamic resource consumptions of different components such as cloudlets and links), it is difficult to obtain accurate measurements on these features in order to achieve global optimality. The actual profit collected could be perturbed from the initially expected one.

The extensive effort on the Service Function Chain Placement (SFCP) problem and its variants have been conducted in the past several years under both datacenter networks and MEC environments. Particularly, most studies of SFC request admissions were based on the fixed resource allocation mode, i.e., all resource demands are fixed prior to the execution of a request. For example, Beck *et al.* [9] proposed a heuristic to coordinate the composition of SFCs and their embedding into a substrate network with the aim to minimize bandwidth utilization. Jalalitabar *et al.* [40] studied how to efficiently accommodate SFC requests while taking into account the function dependence in SFCs, the computing demand, and the bandwidth demand by SFC requests. They devised a heuristic algorithm for constructing and mapping an SFC by incorporating Dependence Sorting and Independent Grouping. Liu *et al.* [63] dealt with a profit maximization problem by jointly deploying the SFCs of incoming user requests and readjusting the SFC placement of accepted user requests. They developed a column generation based algorithm, considering the resource capacity constraint and the operational overhead constraint. Sun *et al.* [92] devised two heuristic algorithms for the SFC orchestration problem considering the distinct domains provisioned by different network providers. They proposed two SFC partitioning methods, and a bidding mechanism-based sub-SFC mapping solution. Tomassilli *et al.* [96] studied the problem of deploying reliable SFCs over a virtualized network function architecture. They then applied the column generation technique to deal with the decomposition model derived from the Integer Linear Programming (ILP) formulation. Zhu *et al.* [123] took both link bandwidth and node computing resource capacities into consideration. They devised an efficient online heuristic algorithm to improve the resource utilization rate by reducing resource fragmentation in physical networks. Zheng *et al.* [121] aimed to minimize the service delay considering the composition and embedding of hybrid SFCs. They proposed an Eulerian Circuit based approximation algorithm for the problem under a special case where each substrate node is assumed to provide only one unique VNF. They also applied the betweenness centrality technique to devise a heuristic algorithm for the problem under the general case. Zhang *et al.* [119] studied an SFC placement problem with the aim to minimize the total energy consumption in a telecom network. They developed efficient algorithms based on the Markov approximation technique for the problem under offline and online scenarios, respectively.

On the other hand, there are also several studies focusing on request admissions through dynamic resource allocation in MEC, under uncertain resource demands of request implementations. For example, Ali *et al.* [5] proposed a novel metric to measure the robustness of resource allocation in distributed systems against various perturbations applied to system parameters. They also described a procedure to guide efficient resource allocation based on the metric. Chen *et al.* [15] devised a QoS-guaranteed SFC outsourcing algorithm based on the Hidden Markov Model to plan the outsourcing of SFCs, by predicting state sequences with the highest probability. Esposito *et al.* [27] designed an SFC instantiation prototype to deal with the random failures of processes or communication links, based on a fully distributed asynchronous consensus mechanism. Eshraghi *et al.* [117] studied a task offloading problem with unknown processing time for a three-tier cloud computing network with multi-processor access points. They developed an efficient algorithm for the problem by constructing a series of locally tight approximate geometric programming problems, which are iteratively updated. Nguyen *et al.* [74] investigated the deadline-aware SFC orchestration problem under the co-located and geo-distributed schemes, respectively, and developed approximation algorithms for SFC placement and routing with the partial knowledge of traffic demands. Psychas *et al.* [80] considered a job scheduling problem with random resource requirements to maximize the throughput. They proposed two scheduling algorithms for the problem based on a 'Best-Fit' packing method and a 'universal partitioning' method, respectively. Wang *et al.* [100] studied a parallelized SFC placement problem with the aim to ensure efficient data transmission while reducing the end-to-end delay of an SFC. They proposed resource-efficient VNF placement methods to enhance the resiliency of a parallelized SFC via multi-flow backups. Zhang *et al.* [120] investigated a task offloading problem in a 5G small cell network considering the uncertainty of both the resource consumption and the reward of a task. They developed a multi-armed bandit based online learning algorithm, and its regret and violations are proved to be bounded sub-linearly.

We distinguish our work from existing ones as follows. Most existing studies assumed that the resource demands of each SFC request are given in advance [9; 40; 63; 92; 96; 123; 121; 119], however, this assumption may not be realistic as the precise resource demands of a request implementation are not known until its completion [89]. There are a few other studies that considered the uncertainty of the other metrics, such as the execution time, throughput, etc. [5; 15; 27; 117; 74; 80; 100]. The study of [120] includes the uncertainty of the resource consumption in task offloading, however, the efficient SFC placement is not taken into consideration. Here we study the robust SFC problem under the assumption that both computing resource and the data rate demand of each request dynamically change during its implementation. We aim to maximize the expected profit collected by the network service provider through admitting as many requests as possible. To the best of our knowledge, we are the first to study this robust SFC placement problem under the uncertainty of both computing resource and data rate demands of all admitted requests, by devising the very first near-optimal approximation algorithm with a provable performance gap to the problem.

1.7 Thesis Contributions

The main contributions of this thesis are to systematically study the resource allocation and scheduling of virtual service provisioning for IoT applications in MEC environments, through formulating novel system models and optimization frameworks for the aforementioned problems. We aim to improve the user experience of using IoT services, and endow IoT service provisioning with flexibility, pervasiveness, robustness and cost-efficiency. We then propose efficient algorithms to judiciously manage network resources in MEC networks and optimize the system performance with the objective of maximizing the user service satisfaction, minimizing the operational cost, maximizing the number of requests admitted, and maximizing the expected profit, respectively. The contributions of this thesis are summarized as follows.

- For the problem of the service provisioning of delay-sensitive IoT applications in MEC environments, we study how to maximize the total user service satisfaction under both static and dynamic user service demands in Chapter 2, where a user satisfaction is inversely proportional to the extra service delay beyond the user's delay threshold. Specifically, we consider an integrated MEC platform consisting of the remote cloud and a set of local cloudlets, and an IoT user can offload his task with a delay requirement to either the remote cloud or a cloudlet. The contributions involved in this topic include: (1) a novel metric to measure user satisfaction of using an IoT service, (2) approximation and online algorithms with provable performance guarantees for special cases of the defined problems when the bandwidth capacity constraint is negligible, and

(3) efficient heuristic algorithms for the problems with the bandwidth capacity constraint.

- For the problem of service provisioning in an MEC network for multi-source IoT applications with Service Function Chain (SFC) requirements, we aim to minimize the operational cost of such service provisioning in Chapter 3. Each multi-source IoT application has multiple data streams from different sources to be uploaded to a destination in the MEC network for aggregation, processing, and storage purposes, while each data stream must pass through the network functions in the SFC of the IoT application prior to reaching its destination. The contributions involved in this topic include: (1) a service provisioning framework in the MEC network for multi-source IoT applications that consists of uploading the stream data from multiple sources of the IoT application, VNF instance placement and sharing, data stream aggregation and routing, and workload balancing among cloudlets in the MEC network, (2) an efficient algorithm for the cost minimization problem for a single multi-source IoT application, and (3) an efficient algorithm for the cost minimization problem for a set of multi-source IoT applications.

- For the Deep Neuron Network (DNN) inference throughput maximization problem, we consider how to maximize the number of delay-aware DNN inference requests admitted in Chapter 4. To meet the inference delay requirement of a DNN inference request, we first consider adopting the DNN partitioning method to partition the DNN inference model between the local IoT device of the request and the assigned cloudlet in the MEC network. To further accelerate the DNN inference of the request, we then adopt the inference parallelism method by leveraging multiple threads in the assigned cloudlet to process the offloaded DNN model part. Through jointly exploring DNN partitioning and inference parallelism, we consider the problem under both offline and online request arrival settings, and the proposed algorithms handle DNN inference service request admissions with the objective of maximizing the number of requests admitted. The contributions involved in this topic include: (1) a constant approximation algorithm for the problem under the offline setting, and (2) an online algorithm with a provable competitive ratio for the problem under the online setting.

- For the robust SFC placement problem, we investigate how to maximize the expected profit collected by the network service provider of an MEC network in Chapter 5. The dynamic resource demands of IoT services significantly affect the service robustness and the system performance. To provide robust IoT services with SFC requirements, we consider the uncertainty of both computing resource and data rate demands in the implementation of user requests. We then devise a Markov based approximation algorithm for the problem, with the aim of maximizing the expected profit of the network service provider. The contributions involved in this topic include: (1) a Quadratic Integer Programming (QIP) formulation for the problem, and (2) a near-optimal approximation algorithm by adopting the Markov approximation technique, which can achieve a provable optimality gap.

It is also worth noting that the devised algorithms, as well as the proposed algorithm design and analysis techniques, will be of independent interests in many other domains, especially the combinatorial optimization.

1.8 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 addresses the total utility maximization problems under both static and dynamic offloading task request settings, with the aim to maximize the accumulative user satisfaction on the use of the services provided by the MEC. Chapter 3 investigates service provisioning in an MEC network for multi-source IoT applications with SFC requirements, where each multi-source IoT application has multiple data streams from different sources to be uploaded to a location (a cloudlet) in the MEC network for aggregation, processing, and storage, with the aim to minimize the operational cost of such service provisioning. Chapter 4 studies a novel DNN inference throughput maximization problem with the aim to maximize the number of delay-aware DNN service requests, by accelerating each DNN inference through jointly exploring DNN partitioning and inference parallelism. Chapter 5 formulates a novel robust SFC placement problem with the aim to maximize the expected profit collected by the network service provider, under the uncertainty assumption of both computing resource and data rate demands in the implementation of user requests. Chapter 6 summarizes the thesis and proposes future work.

Chapter 2

Maximizing User Service Satisfaction for Delay-Sensitive IoT Applications

In this chapter, we first introduce two novel optimization problems for delay-sensitive Internet of Things (IoT) applications, i.e., the total utility maximization problems under both static and dynamic offloading task request settings, with the aim to maximize the accumulative user satisfaction on the use of the services provided by the Mobile Edge Computing (MEC). We then devise efficient approximation and online algorithms with provable performance guarantees for the problems in a special case where the bandwidth capacity constraint is negligible. We also develop efficient heuristic algorithms for the problems with the bandwidth capacity constraint.

2.1 Introduction

Fuelled by the 5G technology, it is expected that the 5G-supported MEC will be the promising platform for delay-sensitive IoT services for various IoT applications. To explore the potential of MEC to support IoT applications, in this chapter, we deal with offloading task services in MEC for delay-sensitive IoT applications, where IoT devices are resource-constrained, by offloading their tasks to cloudlets or a remote cloud for processing. We here consider an integrated platform consisting of the remote cloud and a set of local cloudlets forming an MEC network for IoT service provisioning. IoT devices or mobile users can offload their tasks to the platform for processing, and different offloading task service requests have different service delay requirements. We aim to devise efficient scheduling algorithms for assigning requests to different cloudlets or the remote cloud while meeting their service delay requirements.

The novelty of the work in this chapter lies in that we consider the user satisfaction of using services provided by an MEC network and a remote cloud for delay-sensitive IoT applications, through maximizing the accumulative user satisfaction when different users offload their tasks with different service delay requirements. A novel metric to measure user satisfaction of using a service is proposed, and efficient approximation and online algorithms for the defined problems under both static and dynamic user service demands are then devised.

The main contributions of this chapter are presented as follows.

- We consider user service satisfaction of using services provided by an MEC network and a remote cloud for delay-sensitive IoT applications, by formulating two novel user service satisfaction problems. We also show that the defined problems are NP-hard.
- We devise approximation and online algorithms with provable performance guarantees for special cases of the defined problems when the bandwidth capacity constraint is negligible. We also develop efficient heuristic algorithms for the problems with the bandwidth capacity constraint.
- We evaluate the performance of the proposed algorithms through experimental simulations. Experimental results demonstrate that the proposed algorithms are promising.

The rest of the chapter is organized as follows. Section 2.2 introduces the system model, notions, notations, problem definitions, and NP-hardness proofs of the defined problems. Section 2.3 devises an approximation algorithm and an efficient heuristic algorithm for the total utility maximization problem without and with the bandwidth capacity constraint, respectively. Section 2.4 deals with dynamic user service request admissions without the knowledge of future arrivals for a given time horizon, and efficient online algorithms for the problem are developed. Section 2.5 evaluates the proposed algorithms empirically, and a summary is given in Section 2.6.

2.2 Preliminaries

In this section, we first introduce the system model. We then give notions, notations, and the modelling of user service satisfaction of using services. We finally quantify the user satisfaction on a provided service in an MEC network, and define the problems precisely.

2.2.1 System model

Consider a heterogeneous MEC network that is represented by an undirected graph $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$, where $\mathcal{AP}$ is the set of Access Points (APs), V is the set of cloudlets, v_0 is the remote cloud, and E is the set of links between APs [56; 55]. Each cloudlet (edge cloud) $v \in V$ is co-located with an AP, and connected through a high-speed optical cable, and the communication delay between them thus is negligible. However, not each AP is co-located with a cloudlet, and the number of cloudlets usually is far smaller than that of APs. Each cloudlet $v \in V$ is associated with a computing capacity $C_v > 0$ and a packet processing rate μ_v. Node v_0 is a remote cloud with unlimited computing and storage resources. Thus, the remote cloud v_0 has the maximum packet processing rate, compared with cloudlets. Each link $e \in E$ has a bandwidth capacity $B(e)$. We further assume that each AP in the MEC network is connected to the remote cloud v_0 through a gateway in the MEC network, and the communication delay is far larger than the communication delay between any pair of APs in the MEC network. Figure 2.1 illustrates a heterogeneous MEC network that consists of five APs, three cloudlets co-located with APs, and a remote cloud.

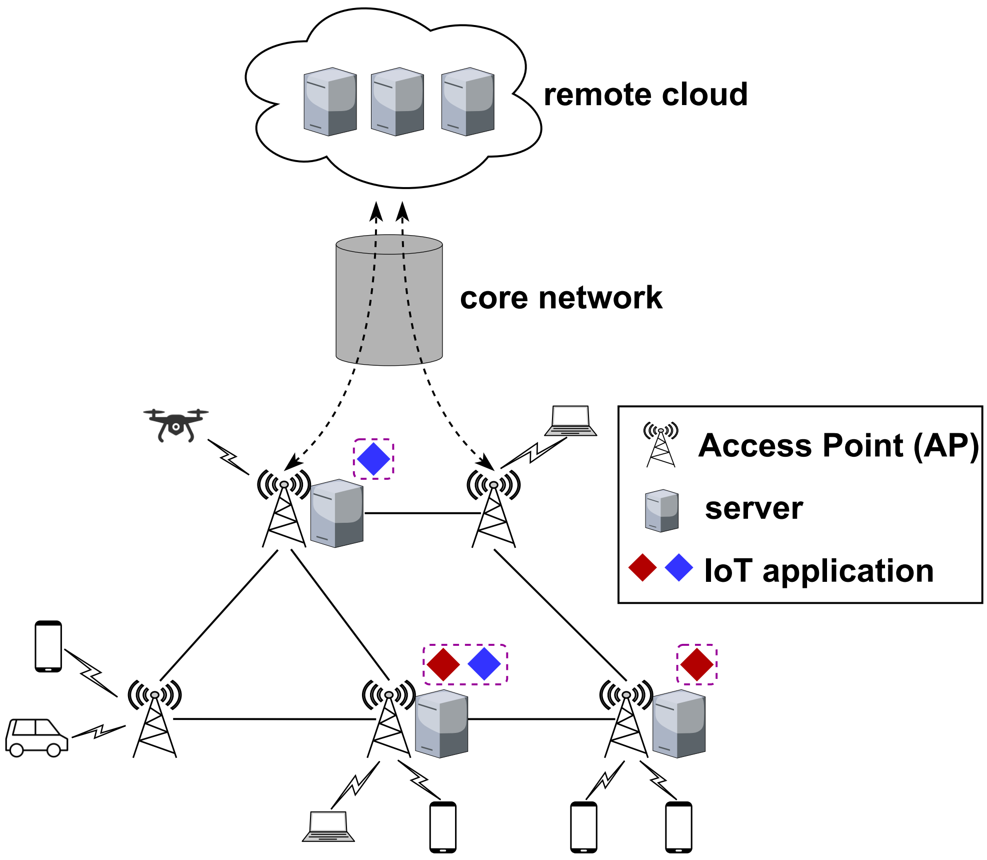


Figure 2.1: An illustrative example of a heterogeneous MEC network that consists of five APs, three cloudlets co-located with APs, and a remote cloud.

We assume that different cloudlets have different computing resource capacities and different processing capabilities. For a given offloading task, the assignment of the task to different cloudlets will result in different computing delays as the workloads and computing capabilities at different cloudlets are different.

We consider a given time horizon that is further divided into T equal time slots. Within each time slot t, let $\mu^t_{v_j}$ represent the processing rate of cloudlet $v_j \in V$, and $C'^t_{v_j}$ the residual computing capacity of v_j at time slot t, where $C'^1_{v_j} = C_{v_j}$ for all

$v_j \in V$, and $\mu^t_{v_0}$ is the processing capability of node v_0, which is the maximum one, i.e., $\mu^t_{v_0} = \max\{\mu^t_{v_j} \mid 0 \leq j \leq |V|\}$. We further assume that the data rate $\gamma^t_{l_i}$ for task offloading of a request r_i from its nearby AP l_i at time slot t is fixed.

Although data uploading from an IoT device to its nearby AP is the bottleneck of some delay-sensitive applications, it becomes insignificant with adopting the 5G technology. Also, given a communication metric (e.g., the link congestion or the Euclidean distance between the two endpoints of each physical link), let $d^t(e)$ be the transmission delay on a link e in the MEC, which is fixed at each time slot t. However, the values of the mentioned parameters may change at different time slots. For the sake of convenience, we will drop index t from these parameters if no confusion arises from the context.

2.2.2 The service delay of an offloading task for service

Consider a set R of requests, each user service request $r_i \in R$ can be expressed by a tuple $r_i = \langle s_i, b_i, l_i, D_i, \beta_i \rangle$, where s_i is the task size (volume), b_i is the demanded bandwidth resource, the user of r_i is under the coverage of AP l_i, D_i is the service delay requirement threshold, and $\beta_i \cdot D_i$ is the maximum service delay the user could tolerate with a constant $\beta_i \geq 1$. Denote by $c(s_i)$ the demanded computing resource to process the offloaded task of request r_i. *The service delay of a request* consists of the uploading delay, the communication delay of routing the data from the data source to the cloudlet (or the remote cloud) for the data processing, and the processing delay of the task at the cloudlet (or the remote cloud), which are as follows.

The uploading delay $d_{upload}(r_i)$ of an offloading task r_i through its located AP l_i is

$$d_{upload}(r_i) = \frac{s_i}{\gamma_{l_i}}, \tag{2.1}$$

where γ_{l_i} is the uplink data rate of AP l_i, which can be calculated by the following Shannon-Hartley formula [98].

$$\gamma_{l_i} = W_{l_i} \log_2(1 + \frac{\kappa_i}{\sigma^2}), \tag{2.2}$$

where W_{l_i} is the total bandwidth of AP l_i divided by the number of users under its coverage, κ_i is the transmission power of IoT device of request r_i, and σ^2 is the noise power.

An offloading task will be served by either a cloudlet or the remote cloud $v_j \in V \cup \{v_0\}$, *the communication delay $d_{comm}(r_i, v_j)$* of offloading task r_i to node v_j is

$$d_{comm}(r_i, v_j) = \begin{cases} \sum_{e \in P_i(v_j)} d(e), & \text{if } v_j \in V \\ d_{comm}(r_i, v_0), & \text{otherwise } (v_j = v_0), \end{cases} \tag{2.3}$$

where $P_i(v_j)$ is a routing path of request r_i between its AP location l_i and the AP location of cloudlet v_j, $d(e)$ is the communication delay on a link e, and $d_{comm}(r_i, v_0)$

is the communication delay of routing the task of r_i from AP l_i to the remote cloud through the gateway.

The processing delay $d_{comp}(r_i, v_j)$ of an offloading task r_i at cloudlet or the remote cloud v_j is

$$d_{comp}(r_i, v_j) = \frac{s_i}{\mu_{v_j}}, \tag{2.4}$$

where s_i is the task size of request r_i, and μ_{v_j} is the processing rate of cloudlet (or the remote cloud).

The service delay $d(r_i, v_j)$ of offloading task r_i to node v_j for service thus is defined as follows.

$$d(r_i, v_j) = d_{upload}(r_i) + d_{comm}(r_i, v_j) + d_{comp}(r_i, v_j). \tag{2.5}$$

Note that we do not include the delay of returning the result to the user as the result usually is no larger than the uploading volume of data, and the delay of returning the result thus is omitted.

2.2.3 User service satisfaction of using a service

In most IoT applications, each service request does have its expected delay threshold and maximum tolerable delay requirement. A task offloading request usually can be represented by a tuple $r_i = \langle s_i, b_i, l_i, D_i, \beta_i \rangle$, where D_i is its delay threshold, if the actual service delay is beyond its threshold D_i, the service may still be acceptable by the user. However, in terms of the service experience, the user of request r_i may not be happy about the service. In other words, the service satisfaction of a user for his requested service can be expressed by a non-increasing function of the service delay he experienced. If a service delay is within the specified threshold, the user satisfies the service with 100%; otherwise, his satisfaction with the service is inversely proportional to the extra service delay beyond the user's delay threshold. Specifically, assume that a user request $r_i \in R$ is assigned to cloudlet or the remote cloud v_j for service, then its service delay is $d(r_i, v_j)$ by Eq. (2.5). If $d(r_i, v_j)$ is no greater than D_i, the user satisfies the service with 100%; otherwise, his satisfaction on the service will dramatically decrease with the increase on the value of $d(r_i, v_j)$, and the maximum tolerant service delay of the user is $\beta_i \cdot D_i$, where $\beta_i \geq 1$ is a constant, representing a certain degree of the delay tolerance of the user. If a service delay is beyond the maximum tolerant service delay of the user, the user satisfaction on the service will become zero. We thus model a user service satisfaction of using a service provided by an MEC network and a remote cloud through *a non-increasing utility function* as follows.

$$u(r_i, v_j) = \begin{cases} (\lambda - \lambda^{\frac{[d(r_i, v_j) - D_i]^+}{\beta_i \cdot D_i}}), & \text{if } d(r_i, v_j) \leq \beta_i \cdot D_i \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

where $[x]^+ = \max\{x, 0\}$, and $\lambda > 1$ is a constant that indicates the delay sensitivity.

It can be seen from Eq. (2.6) that if the service delay is no greater than D_i, $[d(r_i, v_j) - D_i]^+ = 0$, then $\lambda^0 = 1$, and the utility gain of the user is $u(r_i, v_j) = \lambda - 1 > 0$, implying the user is 100% satisified. Otherwise, if the service delay is within the delay range of $(D_i, \beta_i \cdot D_i]$, i.e., $0 < d(r_i, v_j) - D_i \le (\beta_i - 1) \cdot D_i$, then $\frac{[d(r_i,v_j)-D_i]^+}{\beta_i \cdot D_i} = \frac{d(r_i,v_j)-D_i}{\beta_i \cdot D_i} \le \frac{(\beta_i-1)\cdot D_i}{\beta_i \cdot D_i} = \frac{\beta_i-1}{\beta_i} < 1$, and the utility value $u(r_i, v_j) = \lambda - \lambda^{\frac{d(r_i,v_j)-D_i}{\beta_i \cdot D_i}} \le \lambda - \lambda^{\frac{\beta_i-1}{\beta_i}} < \lambda - 1$, i.e., the user satisfaction decreases with the growth of the delay duration and is impacted by both λ and β_i. A larger value of λ means that the utility obtained is more sensitive than that of a smaller λ, and a larger β_i implies that the user of request r_i is more tolerable to his service delay. When the actual service delay $d(r_i, v_j) > \beta_i \cdot D_i$ that is beyond the maximum tolerant service delay of the user of r_i, then $u(r_i, v_j) = 0$ by the utility function definition, and the user satisfaction is 0%. Thus, the value of β_i reflects the service delay tolerance of the user of request r_i at a certain extent.

2.2.4 Problem definitions

In this chapter, we consider the service provisioning in an integrated platform that consists of an MEC network and a remote cloud for delay-sensitive IoT applications, by formulating two novel optimization problems.

Problem 1: Given an MEC network $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$ with a given set R of requests, each request $r_i = \langle s_i, b_i, l_i, D_i, \beta_i \rangle$ in R is expressed by a tuple, where s_i is the size of the offloading task, b_i is the demanded bandwidth resource, l_i is the physical location of the offloading task, D_i is the ideal tolerable delay threshold, and $\beta_i \cdot D_i$ is the maximum tolerable service delay of the request. *The total utility maximization problem* is to maximize the utility sum of all requests in R, i.e., the total user experience of using the services provided by the MEC network, subject to computing capacities on cloudlets and bandwidth capacities on links in G.

As network service providers provide continuing services for their consumers, in the defined problem so far, we have only considered user requests at a given time slot t, where the data rate γ_l^t assigned for each user under the coverage of an AP $l \in \mathcal{AP}$ is fixed at time slot t. However, the value of $\gamma_l^{t'}$ will change at a different time slot $t' \neq t$ which will be determined by the number of users under the coverage of AP l at that time slot. Meanwhile, the transmission delay $d^{t'}(e)$ on a link e in G at time slot t' can also be changed, which is impacted not only by the link length but also the congestion on the link. The processing rate μ_v^t of a node $v \in V \cup \{v_0\}$ may vary at different time slots too. In the following we consider the dynamic user service request admissions within a finite time horizon that consists of T equal time slots.

Problem 2: Given an MEC network $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$ and a finite time horizon that consists of T equal time slots, assume that user service requests arrive one by one without the knowledge of future arrivals, *the online average total utility maximization problem* is to maximize the average sum of accumulative utilities of all admitted requests per time slot within the given time horizon, subject to both computing capacities on cloudlets and bandwidth capacities on links in G.

Theorem 2.1. *The total utility maximization problem in an MEC network $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$ is NP-hard.*

Proof. We show the claim by a reduction from a well-known NP-hard problem – the maximum profit Generalized Assignment Problem (GAP) that is defined as follows [22]. Given n items and m bins, if item i is packed to bin j, it results in a profit $p_{i,j}$ and a size $s_{i,j}$. Usually the size of each item i at different bins is fixed, i.e., $s_{i,j} = s_{i,j'}$ even if $j \neq j'$. Each bin j has a capacity, the problem is to pack as many items as possible to the m bins such that the total profit of the packed items is maximized, subject to bin capacities.

We consider a special case of the total utility maximization problem where the bandwidth resource consumption of each request is negligible, by assuming that there is abundant bandwidth resource on each link in the MEC network. Thus, the routing path of routing the offloaded task of request r_i to cloudlet v is the routing path from AP l_i to cloudlet v with the least communication delay. There are $(|V| + 1)$ bin, where bin $\mathcal{B}_0$ corresponds to the remote cloud with unlimited computing capacity, and each of the other bins corresponds to a cloudlet v with the capacity of C_v. Each item i corresponds to a request r_i. For each bin $\mathcal{B}_j$, each item has the size $s_{i,j} = c(s_i)$ and profit $p_{i,j} = u(r_i, v_j)$. The total utility maximization problem for this special case is to maximize the total utility gain by admitting as many requests as possible, subject to the computing capacities on cloudlets. It can be seen that this special problem is equivalent to the maximum profit GAP. Hence, the total utility maximization problem is NP-hard. $\square$

2.3 Algorithms for the total utility maximization problem

In this section, we deal with the total utility maximization problem. We first consider a special case of the problem where there are abundant bandwidth resources on links, for which we formulate an Integer Linear Programming (ILP) solution for the problem when the problem size is small. Otherwise, we devise an approximation algorithm with a provable approximation ratio for the problem, by reducing the problem to the maximum profit GAP problem. An approximate solution to the latter in turn returns an approximate solution to the former. We also devise an efficient heuristic algorithm for the problem under the bandwidth capacity constraint.

2.3.1 ILP and approximation algorithms for the problem without the bandwidth capacity constraint

We deal with the total utility maximization problem without the bandwidth capacity constraint on links, by assuming that each link has abundant bandwidth resource. We start with the ILP formulation as follows.

$$\text{Maximize} \qquad \sum_{i=1}^{|R|}\sum_{j=0}^{|V|} u(r_i, v_j) \cdot x_{i,j}, \tag{2.7}$$

subject to the following constraints.

$$\begin{aligned} &\text{Eq. (2.1), (2.2), (2.3), (2.4), (2.5), (2.6),} \\ &\quad \forall i, j, \quad 1 \le i \le |R|,\ 0 \le j \le |V| \end{aligned}$$

$$\sum_{j=0}^{|V|} x_{i,j} \le 1, \ \forall i, \quad 1 \le i \le |R| \tag{2.8}$$

$$\sum_{i=1}^{|R|} x_{i,j} \cdot c(s_i) \le C_{v_j}, \ \forall j, \quad 0 \le j \le |V| \tag{2.9}$$

$$x_{i,j} \in \{0,1\}, \ \forall i, j, \quad 1 \le i \le |R|,\ 0 \le j \le |V|, \tag{2.10}$$

where variable $x_{i,j}$ is a binary decision variable, and $x_{i,j} = 1$ implies that offloading task r_i will be served by cloudlet/the remote cloud v_j with $0 \le j \le |V|$. Constraint (2.8) ensures that each request is assigned to at most one node for service. Constraint (2.9) ensures that the accumulative resource demand by all requests assigned to a node is no more than the capacity of the node. Recall that we assume that the remote cloud is node v_0 with unlimited computing resource. Note that for each request r_i in Eq. (2.3), its routing path $P_i(v_j)$ to cloudlet $v_j \in V$ is a shortest path in G between AP l_i and cloudlet v_j, and the weight of each link e in $P_i(v_j)$ is the transmission delay, i.e., d_e, because each link is assumed to have abundant bandwidth resource.

We then devise an approximation algorithm for the problem by reducing it to the maximum profit GAP, which is a well-known NP-hard problem, and there is an efficient approximation algorithm for it [22].

The reduction is as follows. There are $(|V| + 1)$ bins, where bin $\mathcal{B}_0$ corresponds to the remote cloud with unlimited computing resource, the rest $|V|$ bins correspond to the $|V|$ heterogeneous cloudlets, where $\mathcal{B}_j$ with $1 \le j \le |V|$ represents cloudlet $v_j \in V$ with capacity C_{v_j}. There are $|R|$ requests. Recall that request $r_i \in R$ is located at AP l_i if it is assigned to cloudlet v_j for service with the computing resource consumption $c(s_i)$, then the utility gain is $u(r_i, v_j)$ by Eq. (2.6), which is determined by the experienced service delay $d(r_i, v_j)$. In other words, if we pack request r_i to bin $\mathcal{B}_j$, it generates a profit $u(r_i, v_j)$ with size $c(s_i)$, where $1 \le j \le |V|$; otherwise (if r_i is sent to the remote cloud v_0 for service), its service delay is $d(r_i, v_0)$, and the utility gain is $u(r_i, v_0)$. Note that when the utility obtained by packing a request to a bin is zero, the request will not be admitted. The detailed algorithm is given in `Algorithm` 1.

Algorithm 1 An approximation algorithm for the total utility maximization problem without the bandwidth capacity constraint

```
Require: $|V|$ cloudlets with each $v_j \in V$ having computing capacity $C_{v_j}$, a remote cloud $v_0$ with unlimited computing capacity, i.e., $C_{v_0} = \infty$, a set of requests $R$ with each request $r_i = \langle s_i, b_i, l_i, D_i, \beta_i \rangle$.
Ensure: Admit as many requests as possible from $R$ that maximizes the utility sum of admitted requests.
1: Calculate the shortest path between each cloudlet and each AP, and the weight of each link is the communication delay on the link.
2: Construct an instance of the GAP, where each request $r_i \in R$ has a corresponding item $i$ with size $c(s_i)$ and the profit $u(r_i, v_j)$. Each cloudlet $v_j$ or the remote cloud corresponds to a bin $\mathcal{B}_j$ with bin capacity $cap(\mathcal{B}_j) = C_{v_j}$, where $0 \le j \le |V|$;
3: Find an approximate solution $A$ to the GAP problem with maximizing the utility sum, by invoking the approximation algorithm due to Cohen et al. [22];
4: for any request $r \in A$ with utility zero do
5:     $A \leftarrow A \setminus \{r\}$; /* remove request $r$ from the solution */;
6: end for;
7: return the solution $A$ as the solution of the total utility maximization problem without the bandwidth capacity constraint.
```

2.3.2 Heuristic algorithm for the problem with the bandwidth capacity constraint

We now consider the problem under the bandwidth capacity constraint by developing a greedy algorithm that proceeds iteratively.

At each iteration, for a request $r_i \in R$ to be offloaded, we first identify the set of cloudlets $V' \subseteq V$ and the set of links $E' \subseteq E$ with sufficient residual computing resource and bandwidth resource to accommodate request r_i, respectively. We then find a routing path $P_i(v_j)$ in the induced subgraph $G' = (\mathcal{AP} \cup V' \cup \{v_0\}, E')$ of graph $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$ with the least communication delay from location l_i of request r_i to cloudlet $v_j \in V'$, through links in E'. Then, the utility gain of assigning request r_i to cloudlet v_j through the routing path $P_i(v_j)$ can be obtained. Among all nodes in $V' \cup \{v_0\}$, we then identify a node $\hat{v}_i$ with the maximum utility gain for request r_i. Because the remote cloud v_0 has the unlimited computing resource, the remote cloud can be identified as the offloading node of the request too. However, if the maximum utility gain of assigning request r_i to the node $\hat{v}_i$ is zero, the request r_i will be rejected. Then, among all requests to be offloaded, we identify a request $r_{i'}$ with the maximum utility gain for its admission. If request $r_{i'}$ is assigned to a cloudlet, the residual computing resource on the cloudlet and residual bandwidth resource on the links in the routing path are then updated accordingly. This procedure continues until all requests are either admitted or rejected. The detailed algorithm is given in `Algorithm` 2.

Algorithm 2 A heuristic algorithm for the total utility maximization problem with the bandwidth capacity constraint

Require: $|V|$ cloudlets with each $v_j \in V$ having computing capacity C_{v_j}, a remote cloud v_0 with unlimited computing capacity, i.e., $C_{v_0} = \infty$, each link $e \in E$ connecting cloudlets has a bandwidth capacity, and a set of requests R with each request $r_i = \langle s_i, b_i, l_i, D_i, \beta_i \rangle$.

Ensure: Admit as many requests as possible from R that maximizes the utility sum of admitted requests.

```
1:  $\mathbb{R} \leftarrow R$; /* the requests to be offloaded */
2:  $A \leftarrow \emptyset$; /* the solution */
3:  while $\mathbb{R} \neq \emptyset$ do
4:      for each request $r_i \in \mathbb{R}$ do
5:          Identify the set of cloudlets $V' \subseteq V$ and the set of links $E' \subseteq E$ with sufficient residual resource for $r_i$;
6:          Find the routing path $P_i(v_j)$ from AP $l_i$ to each cloudlet $v_j \in V'$ with the least communication delay, through links in $E'$;
7:          Calculate the utility gain $u(r_i, v_j)$ if $r_i$ is assigned to each cloudlet $v_j \in V'$ through the routing path $P_i(v_j)$;
8:          Calculate the utility gain $u(r_i, v_0)$ if $r_i$ is assigned to the remote cloud $v_0$.
9:          Find node $\hat{v}_i \in V' \cup \{v_0\}$ with the maximum utility gain $u(r_i, \hat{v}_i)$ for request $r_i$;
10:         if $u(r_i, \hat{v}_i) = 0$ then
11:             $r_i$ is rejected;
12:             $\mathbb{R} \leftarrow \mathbb{R} \setminus \{r_i\}$;
13:         end if;
14:     end for;
15:     Find request $r_{i'} \in \mathbb{R}$ with the maximum utility gain, and admit request $r_{i'}$ by assigning request $r_{i'}$ to node $\hat{v}_{i'}$;
16:     $A \leftarrow A \cup \{r_{i'}\}$; $\mathbb{R} \leftarrow \mathbb{R} \setminus \{r_{i'}\}$;
17:     if $\hat{v}_{i'}$ is a cloudlet then
18:         Update the residual resource on cloudlet $\hat{v}_{i'}$ and the links on the routing path $P_{i'}(\hat{v}_{i'})$;
19:     end if;
20: end while;
21: return the solution $A$ as the solution of the total utility maximization problem with the bandwidth capacity constraint;
```

2.3.3 Algorithm analysis

In the following, we first analyze the approximation ratio and time complexity of the approximation algorithm, `Algorithm` 1. We then analyze the time complexity of the heuristic algorithm, `Algorithm` 2.

Lemma 1. Given an MEC network $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$ and a set R of user requests, the upper bound on the optimal solution of the total utility maximization problem in G is $(\lambda - 1) \cdot |R|$.

Proof. The claim of that the optimal solution is upper bounded by $(\lambda - 1) \cdot |R|$ is shown as follows. If a request $r_i \in R$ can be served within its specified delay threshold, i.e., $d(r_i, v_j) \leq D_i$, the utility obtained by this service is $(\lambda - 1)$; if $D_i < d(r_i, v_j) \leq \beta_i \cdot D_i$, its utility is $\lambda - \lambda^{\frac{d(r_i,v_j)-D_i}{\beta_i \cdot D_i}} < \lambda - 1$; otherwise, its utility is 0. □

Theorem 2.2. *Given an MEC network $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$ and a set R of offloading task requests, there is an approximation algorithm,* `Algorithm` *1, for the total utility maximization problem without the bandwidth capacity constraint, which delivers an approximate solution with a $\frac{1}{2+\epsilon}$ approximation ratio. The time complexity of the approximation algorithm is $O(|V| \cdot |\mathcal{AP}|^2 + (|V| + 1) \cdot |R| \cdot \log \frac{1}{\epsilon} + \frac{|V|+1}{\epsilon^4})$, where ϵ is a constant with $0 < \epsilon \leq 1$.*

Proof. The approximation ratio of the proposed algorithm, `Algorithm` 1 can be obtained directly by adopting the analysis of the approximation algorithm due to Cohen *et al.* [22]. The solution delivered by the algorithm is no less than $\frac{1}{2+\epsilon}$ times the optimal one, where ϵ is a constant with $0 < \epsilon \leq 1$.

The running time of `Algorithm` 1 is analyzed as follows. Finding the shortest paths between each cloudlet and each AP takes $O(|V| \cdot |\mathcal{AP}|^2)$ time, while the approximation algorithm due to Cohen *et al.* [22] takes $O((|V| + 1) \cdot |R| \cdot \log \frac{1}{\epsilon} + \frac{|V|+1}{\epsilon^4})$ time. Thus, the time complexity of `Algorithm` 1 is $O(|V| \cdot |\mathcal{AP}|^2 + (|V| + 1) \cdot |R| \cdot \log \frac{1}{\epsilon} + \frac{|V|+1}{\epsilon^4})$. □

Theorem 2.3. *Given an MEC network $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$ and a set R of offloading task requests, there is an algorithm,* `Algorithm` *2, for the total utility maximization problem with the bandwidth capacity constraint, which delivers a feasible solution, and its time complexity is $O(|R|^2 \cdot |\mathcal{AP}|^2)$.*

Proof. It can be seen that the solution delivered by `Algorithm` 2 is feasible because no specified constraint is violated. The time complexity of `Algorithm` 2 is analyzed as follows.

There are at most $|R|$ iterations. Within each iteration, a request with the maximum utility gain from the remaining requests is admitted. The time of calculating the utility gain of admitting a request r_i is dominated by the time of finding the shortest path from the location of request r_i to each cloudlet with sufficient computing resource through links with sufficient bandwidth resource for r_i, which takes $O(|\mathcal{AP}|^2)$ time. Thus, the time complexity of the proposed algorithm is $O(|R|^2 \cdot |\mathcal{AP}|^2)$. □

2.4 Online algorithms for the online average total utility maximization problem

In this section, we study dynamic user service request admissions, where user service requests arrive one by one without the knowledge of future arrivals. All arrived requests will be considered at the beginning of the next time slot. We start with a special case of the problem where the bandwidth capacity constraint is not considered, for which we devise an online algorithm with a provable competitive ratio. We then develop an efficient online algorithm for the problem with the bandwidth capacity constraint.

Notice that once an admitted request finishes its service, its occupied resources will be released back to the system in the end of the time slot it leaves. Thus, the available capacity of each cloudlet or link at each time slot is its residual capacity at that time slot, and those occupied resources are not available for new request admissions at the next time slot.

2.4.1 Online algorithm for the problem without the bandwidth capacity constraint

Denote by $C_v(i)$ the residual computing resource at cloudlet $v \in V$ before considering request r_i, and $C_v(1) = C_v$ initially. If request r_i is assigned to cloudlet v for service, $C_v(i+1) = C_v(i) - c(s_i)$, where $c(s_i)$ is the demanded computing resource of request r_i. Otherwise, request r_i is assigned to the remote cloud for service, and nothing has to be done because the remote cloud has unlimited resource. To capture the computing resource usage on cloudlets, a computing resource usage cost model is introduced as follows.

$$w_v(i) = C_v(\alpha^{1-\frac{C_v(i)}{C_v}} - 1), \tag{2.11}$$

where $\alpha > 1$ is a turning parameter reflecting the sensitivity of the workload at each cloudlet v, and $1 - \frac{C_v(i)}{C_v}$ is the utilization ratio of cloudlet v.

The normalized computing resource cost of assigning offloading request r_i to cloudlet v thus is

$$\psi_v(i) = \frac{w_v(i)}{C_v} = \alpha^{1-\frac{C_v(i)}{C_v}} - 1 \tag{2.12}$$

Upon the arrival of request r_i, we first identify the set $V' \subseteq V$ of cloudlets with sufficient residual computing resource to accommodate request r_i. Then we find the shortest routing path from the located AP l_i of request r_i to each cloudlet $v \in V'$ and calculate the utility gain if request r_i is assigned to cloudlet v through the shortest routing path. Among all cloudlets in V', we identify the set of cloudlets $\mathcal{Q}_i \subseteq V'$ with the positive utility gains for request r_i, and request r_i is assigned to the cloudlet in $\mathcal{Q}_i$ with the minimum normalized computing resource cost by Eq. (2.12). If no such cloudlet exists, request r_i can then be assigned to the remote cloud with unlimited computing resource. However, if the utility gain brought by assigning request r_i to the assigned node is 0 (i.e., $d(r_i, v_j) > \beta_i \cdot D_i$), the request can be rejected.

We now assume that request r_i is assigned to node $v' \in V \cup \{v_0\}$ with the utility gain u_i. If request r_i is assigned to the remote cloud (i.e., $v' = v_0$) with a positive utility gain, it indicates that the request will be admissible. Although r_i is admissible, its admission needs further examined to avoid its admission will consume too much resource, by adopting an admission control policy. That is, request r_i will be rejected if both the following conditions are met. (i) The normalized computing resource cost of cloudlet $v' \in V$ that will accommodate request r_i is greater than $|V| \cdot u_i$, i.e., $\psi_{v'}(i) > |V| \cdot u_i$; and (ii) assigning request r_i to the remote cloud will result in the zero utility gain (i.e., exceeding the maximum tolerable service delay). Note that if condition (i) is met while condition (ii) is violated (i.e., assigning request r_i to the remote cloud will result in a positive utility gain), request r_i is admitted and assigned to the remote cloud.

The detailed online algorithm with a provable competitive ratio is given in `Algorithm` 3.

2.4.2 Online algorithm for the problem with the bandwidth capacity constraint

We then deal with the online average total utility maximization problem with the bandwidth capacity constraint by devising an efficient online algorithm as follows.

Recall that for the problem without the bandwidth capacity constraint, we introduce a computing resource cost model to capture the dynamic consumptions of computing resources on cloudlets. Similarly, we here introduce the bandwidth resource cost model to capture the dynamic bandwidth resource consumptions of links as follows.

$$w_e(i) = B_e(\delta^{1-\frac{B_e(i)}{B_e}} - 1), \tag{2.13}$$

where $\delta > 1$ is a turning parameter reflecting the sensitivity of the workload at each link e, B_e is the bandwidth capacity of link $e \in E$, $B_e(i)$ is the residual bandwidth resource on link $e \in E$ before considering request r_i, and $1 - \frac{B_e(i)}{B_e}$ is the utilization ratio of link e.

The normalized bandwidth cost of link $e \in E$ for request r_i thus is

$$\psi_e(i) = \frac{w_e(i)}{B_e} = \delta^{1-\frac{B_e(i)}{B_e}} - 1. \tag{2.14}$$

The normalized bandwidth cost on a routing path $P_i(v)$ of request r_i then is $\sum_{e \in P_i(v)} \psi_e(i)$.

The *total normalized cost* of assigning request r_i to cloudlet v through the routing path $P_i(v)$ consists of the normalized computing resource cost $\psi_v(i)$ on v and the

Algorithm 3 Online algorithm for the online average total utility maximization problem without the bandwidth capacity constraint

```
Require: $|V|$ cloudlets with each $v_j \in V$ having computing capacity $C_{v_j}$, a remote
    cloud $v_0$ with unlimited computing capacity, i.e., $C_{v_0} = \infty$, a set of requests $R$
    with each request $r_i = \langle s_i, b_i, l_i, D_i, \beta_i \rangle$ arrived one by one, there is no knowledge
    of future request arrivals.
Ensure: Maximize the average total utility gain of admitted requests per time slot
    within the time horizon.
 1: $A \leftarrow \emptyset$; /* the solution */
 2: while request $r_i$ arrives do
 3:     Identify the set of cloudlets $V' \subseteq V$ with sufficient residual computing re-
    source for $r_i$, and find the routing path $P_i(v_j)$ from AP $l_i$ to each cloudlet $v_j \in V'$
    with the smallest communication delay;
 4:     $\mathcal{Q}_i \leftarrow \emptyset$; /* the set of candidate cloudlets for $r_i$ */
 5:     for each cloudlet $v_j \in V'$ do
 6:         Calculate the utility gain of assigning request $r_i$ to cloudlet $v_j$;
 7:         if its utility gain is positive then
 8:             $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup \{v_j\}$;
 9:         end if;
10:     end for;
11:     if $\mathcal{Q}_i = \emptyset$ then
12:         if assigning $r_i$ to remote cloud makes positive utility then
13:             Admit $r_i$ by assigning $r_i$ to remote cloud;
14:         else
15:             Reject $r_i$;
16:         end if;
17:     else
18:         Identify the cloudlet $v' \in \mathcal{Q}_i$ with the minimum normalized cost by
    Eq. (2.12). And calculate the utility gain $u_i$ if request $r_i$ is assigned to cloudlet $v'$;
19:         if $\psi_{v'}(i) > |V| \cdot u_i$ then
20:             if assigning $r_i$ to remote cloud makes positive utility then
21:                 Admit $r_i$ by assigning $r_i$ to remote cloud;
22:             else
23:                 Reject $r_i$;
24:             end if;
25:         else
26:             Admit $r_i$ by assigning $r_i$ to cloudlet $v'$;
27:         end if;
28:     end if;
29:     if $r_i$ is admitted then
30:         $A \leftarrow A \cup \{r_i\}$;
31:     end if
32: end while;
33: return a feasible solution $A$ to the online average total utility maximization prob-
    lem without the bandwidth capacity constraint;
```

normalized bandwidth cost on $P_i(v)$, i.e.,

$$\phi(P_i(v)) = \psi_v(i) + \sum_{e \in P_i(v)} \psi_e(i). \tag{2.15}$$

Similar to `Algorithm` 3, the proposed online algorithm proceeds as follows. Upon the arrival of request r_i, we first identify the set of cloudlets $V' \subseteq V$ and the set of links $E' \subseteq E$ with sufficient computing and bandwidth resources to accommodate request r_i. We then assign each cloudlet $v \in V'$ and each link $e \in E'$ with a normalized computing resource cost $\psi_v(i)$ by Eq. (2.12) and a normalized bandwidth resource cost $\psi_e(i)$ by Eq. (2.14), respectively. We thirdly find a routing path $P_i(v_j)$ in the induced subgraph $G' = (\mathcal{AP} \cup V' \cup \{v_0\}, E')$ of graph $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$ with the least communication delay on the path from location l_i of request r_i to each cloudlet $v_j \in V'$. Hereafter, among all cloudlets in V', we finally identify the set of cloudlets $\mathcal{Q}_i \subseteq V'$ with the positive utility gains for request r_i through the associated routing paths. Then, among all cloudlets in $\mathcal{Q}_i$, we assign request r_i to cloudlet $v' \in \mathcal{Q}_i$ through its routing path with the minimum total normalized cost by Eq. (2.15). If no such a cloudlet exists, the request can be assigned to the remote cloud v_0. However, if the utility gain brought by such an assignment is 0, the request will be rejected.

Assuming that request r_i is assigned to node $v' \in V \cup \{v_0\}$ with the utility gain u_i. If request r_i is assigned to the remote cloud (i.e., $v' = v_0$) with a positive utility gain, it will be admitted. Although r_i is admissible with the utility gain u_i when it is assigned to cloudlet $v' \in V$, its admission needs further to be examined to avoid consuming too much resource through an admission control policy. That is, request r_i will still be rejected if both the following conditions are met: (i) The normalized computing resource cost of cloudlet v' or the normalized bandwidth resource cost of the routing path $P_i(v')$ is greater than $|V| \cdot u_i$, i.e., $\psi_{v'}(i) > |V| \cdot u_i$ or $\sum_{e \in P_i(v')} \psi_e(i) > |V| \cdot u_i$; and (ii) the utility gain is zero if the request is assigned to the remote cloud. Note that if condition (i) is met while condition (ii) is violated (i.e., assigning request r_i to the remote cloud will result in a positive utility gain), request r_i is admitted and assigned to the remote cloud.

The detailed algorithm is given in `Algorithm` 4.

2.4.3 Algorithm analysis

The rest is to analyze the competitive ratio and time complexity of `Algorithm` 3. The time complexity of `Algorithm` 4 is also analyzed.

Let R be the set of requests arrived for the given time horizon. Denote by $\mathcal{Z}(i) \subseteq R$ the set of requests admitted by `Algorithm` 3 prior to the arrival of request r_i, and u_{max} and u_{min} the maximum and minimum utility gains of admitting any request, respectively. Following Eq. (2.6), for a request r_i, $u_{max} = \lambda - 1$ when $d(r_i, v_j) \leq D_i$, while $u_{min} = \min_{r_i \in R}\{\lambda - \lambda^{\frac{\beta_i - 1}{\beta_i}}\}$ when $d(r_i, v_j) = \beta_i \cdot D_i$, and both u_{max} and u_{min} are constants.

Algorithm 4 A heuristic algorithm for the online average total utility maximization problem with the bandwidth capacity constraint

Require: $|V|$ cloudlets with each $v_j \in V$ having computing capacity C_{v_j}, a remote cloud v_0 with unlimited computing capacity, i.e., $C_{v_0} = \infty$, each link $e \in E$ connecting cloudlets has a bandwidth capacity, and a set of requests R with each request $r_i = \langle s_i, b_i, l_i, D_i, \beta_i \rangle$ arrived one by one with no future knowledge.

Ensure: Maximize the average total utility gain of admitted requests per time slot within the time horizon.

```
1:  $A \leftarrow \emptyset$; /* the solution */
2:  while request $r_i$ arrives do
3:      Identify the set of cloudlets $V' \subseteq V$ and the set of links $E' \subseteq E$ with sufficient residual resource for $r_i$, and find the routing path $P_i(v_j)$ from AP $l_i$ to each cloudlet $v_j \in V'$ with the least communication delay, through links in $E'$;
4:      $\mathcal{Q}_i \leftarrow \emptyset$; /* the set of candidate cloudlets for $r_i$ */
5:      for each cloudlet $v_j \in V'$ do
6:          if the utility gain $u(r_i, v_j)$ of assigning $r_i$ to $v_j$ via $P_i(v_j)$ is positive then
7:              $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup \{v_j\}$;
8:          end if;
9:      end for;
10:     if $\mathcal{Q}_i = \emptyset$ then
11:         if assigning $r_i$ to remote cloud makes positive utility then
12:             Admit $r_i$ by assigning $r_i$ to remote cloud;
13:         else
14:             Reject $r_i$;
15:         end if;
16:     else
17:         Identify the cloudlet $v' \in \mathcal{Q}_i$ and its routing path $P_i(v')$ with the minimum total normalized cost by Eq. (2.15), and calculate the utility gain $u_i$ by assigning $r_i$ to $v'$ through $P_i(v')$;
18:         if $\psi_{v'}(i) > |V| \cdot u_i$ or $\sum_{e \in P_i(v')} \psi_e(i) > |V| \cdot u_i$ then
19:             if assigning $r_i$ to remote cloud makes positive utility gain then
20:                 Admit $r_i$ by assigning $r_i$ to remote cloud;
21:             else
22:                 Reject $r_i$;
23:             end if;
24:         else
25:             Admit $r_i$ by assigning $r_i$ to cloudlet $v'$ through $P_i(v')$;
26:         end if;
27:     end if;
28:     if $r_i$ is admitted then
29:         $A \leftarrow A \cup \{r_i\}$;
30:     end if
31: end while;
32: return a feasible solution $A$ to the online average total utility maximization problem with the bandwidth capacity constraint;
```

Lemma 2. Given an MEC network $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$ and a finite time horizon that consists of T time slots, let R be the set of requests arriving one by one within the given time horizon, denote by $\mathcal{Z}(i)$ the set of requests admitted by `Algorithm` 3 prior to the arrival of request r_i. Then, the sum of usage cost of all cloudlets is

$$\sum_{v \in V} w_v(i) \leq 2 \cdot |V| \cdot \log_2 \alpha \cdot \sum_{r_{i'} \in \mathcal{Z}(i)} (c(s_{i'}) \cdot u_{i'}), \tag{2.16}$$

where α is a constant with $2|V| \cdot u_{max} + 2 \leq \alpha \leq 2^{\frac{C_{min}}{c_{max}}}$, $u_{max} = \lambda - 1$, $C_{min} = min\{C_v \mid v \in V\}$, and $c_{max} = \max\{c(s_i) \mid r_i \in R\}$.

Proof. If request $r_{i'} \in R$ is rejected, or admitted by being assigned to the remote cloud, the usage cost of all cloudlets do not change. Otherwise (request $r_{i'}$ is admitted and assigned to cloudlet v'), we have $C_{v'}(i'+1) = C_{v'}(i') - c(s_{i'})$. Then,

$$\begin{aligned}
& w_{v'}(i'+1) - w_{v'}(i') \\
=& C_{v'} \cdot (\alpha^{1-\frac{C_{v'}(i'+1)}{C_{v'}}} - 1) - C_{v'} \cdot (\alpha^{1-\frac{C_{v'}(i')}{C_{v'}}} - 1) \\
=& C_{v'} \cdot \alpha^{1-\frac{C_{v'}(i')}{C_{v'}}} \cdot (\alpha^{\frac{C_{v'}(i')-C_{v'}(i'+1)}{C_{v'}}} - 1) \\
\leq& C_{v'} \cdot \alpha^{1-\frac{C_{v'}(i')}{C_{v'}}} \cdot (\alpha^{\frac{c(s_{i'})}{C_{v'}}} - 1) \\
=& C_{v'} \cdot \alpha^{1-\frac{C_{v'}(i')}{C_{v'}}} \cdot (2^{\frac{c(s_{i'})}{C_{v'}} \cdot \log_2 \alpha} - 1) \\
\leq& C_{v'} \cdot \alpha^{1-\frac{C_{v'}(i')}{C_{v'}}} \cdot \frac{c(s_{i'})}{C_{v'}} \cdot \log_2 \alpha. & (2.17) \\
=& c(s_{i'}) \cdot \alpha^{1-\frac{C_{v'}(i')}{C_{v'}}} \cdot \log_2 \alpha, & (2.18)
\end{aligned}$$

where Ineq. (2.17) holds because $2^x - 1 \leq x$ with $0 \leq x < 1$.

If request $r_{i'}$ is not assigned to cloudlet v, the usage cost of cloudlet v does not change. Then, the difference in the sums of the usage costs of all cloudlets before and after admitting request $r_{i'}$ is

$$\begin{aligned}
& \sum_{v \in V} (w_v(i'+1) - w_v(i')) = w_{v'}(i'+1) - w_{v'}(i') \\
\leq& c(s_{i'}) \cdot \alpha^{1-\frac{C_{v'}(i')}{C_{v'}}} \cdot \log_2 \alpha, \quad \text{by Eq. (2.18)} \\
=& \log_2 \alpha \cdot c(s_{i'}) \cdot ((\alpha^{1-\frac{C_{v'}(i')}{C_{v'}}} - 1) + 1) \\
=& \log_2 \alpha \cdot c(s_{i'}) \cdot (\psi_{v'}(i') + 1), \quad \text{by Eq. (2.12)} \\
\leq& \log_2 \alpha \cdot c(s_{i'}) \cdot (|V| \cdot u_{i'} + 1) & (2.19) \\
\leq& 2 \cdot \log_2 \alpha \cdot |V| \cdot c(s_{i'}) \cdot u_{i'}, & (2.20)
\end{aligned}$$

where Ineq. (2.19) holds because request $r_{i'}$ is admitted by the admission control policy.

The sum of the usage costs of all cloudlets prior to the arrival of request r_i is

$$\begin{aligned}\sum_{v\in V} w_v(i) &= \sum_{i'=1}^{i-1}\sum_{v\in V}(w_v(i'+1)-w_v(i')) \\ &= \sum_{r_{i'}\in\mathcal{Z}(i)}\sum_{v\in V}(w_v(i'+1)-w_v(i')) \\ &\le \sum_{r_{i'}\in\mathcal{Z}(i)}(2\cdot\log_2\alpha\cdot|V|\cdot c(s_{i'})\cdot u_{i'}), \text{ by Eq. (2.20)} \\ &=2\cdot|V|\cdot\log_2\alpha\cdot\sum_{r_{i'}\in\mathcal{Z}(i)}(c(s_{i'})\cdot u_{i'}). \end{aligned} \quad (2.21)$$

□

Denote by $\mathcal{D}(i)$ the set of requests admitted by the optimal solution but rejected by `Algorithm` 3 prior to the arrival of request r_i, and denote by $\mathcal{H}(i)$ the set of requests admitted by both the optimal solution and `Algorithm` 3 prior to the arrival of request r_i. It can be seen that set $\mathcal{D}(i)\cup\mathcal{H}(i)$ is the set of admitted requests by the optimal solution. Then, for each request $r_i\in\mathcal{H}(i)$, we have

$$u_i^* \le \frac{u_{max}}{u_{min}}\cdot u_i, \quad (2.22)$$

where u_i^* and u_i are the utility gains of admitting request r_i by the optimal solution and `Algorithm` 3, respectively, while u_{max} and u_{min} are the maximum and minimum utilities by admitting any request, which are constants.

Lemma 3. Given an MEC network $G=(\mathcal{AP}\cup V\cup\{v_0\},E)$ and a finite time horizon that consists of T time slots, let R be the set of requests arriving one by one over the time horizon, denote by $\mathcal{D}(i)$ the set of requests admitted by the optimal solution but rejected by `Algorithm` 3 prior to the arrival of request r_i. Denote by $v_{i'}^*$ the node in the optimal solution to which request $r_{i'}\in\mathcal{D}(i)$ is assigned. We have $v_{i'}^*\in V$, $\forall r_{i'}\in\mathcal{D}(i)$, i.e., the requests in the set $\mathcal{D}(i)$ are assigned to cloudlets instead of the remote cloud in the optimal solution.

Proof. We prove the lemma by contradiction. We assume that there is a request $r_{i'}\in\mathcal{D}(i)$ that is assigned to the remote cloud in the optimal solution. It can be seen that if a positive utility gain can be obtained when $r_{i'}$ is assigned to the remote cloud, then $r_{i'}$ can be assigned to the remote cloud by `Algorithm` 3. However, request $r_{i'}$ is rejected by `Algorithm` 3. This results in a contradiction. □

Lemma 4. Given an MEC network $G=(\mathcal{AP}\cup V\cup\{v_0\},E)$ and a finite time horizon that consists of T time slots, let R be the set of requests arriving one by one over the time horizon, denote by $\mathcal{D}(i)$ the set of requests admitted by the optimal solution but rejected by `Algorithm` 3 prior to the arrival of request r_i. Denote by $v_{i'}^*$ the node in the optimal solution to which request $r_{i'}\in\mathcal{D}(i)$ is assigned, and denote by $u_{i'}^*$ the utility for request $r_{i'} \in \mathcal{D}(i)$ in the optimal solution. Then, for each request $r_{i'} \in \mathcal{D}(i)$,

$$\psi_{v^*_{i'}}(i') > |V| \cdot \frac{u_{min}}{u_{max}} \cdot u^*_{i'}, \tag{2.23}$$

when $2|V| \cdot u_{max} + 2 \leq \alpha \leq 2^{\frac{C_{min}}{c_{max}}}$.

Proof. We show the claim by distinguishing two cases when request $r_{i'}$ is rejected: Case 1. there is no sufficient computing resource on cloudlet $v^*_{i'}$ to admit request $r_{i'}$ by `Algorithm` 3; and Case 2: there is sufficient computing resource on cloudlet $v^*_{i'}$ to admit request $r_{i'}$ in `Algorithm` 3. But the selected cloudlet $v_{i'}$ to admit request r_i violates the admission control policy in `Algorithm` 3.

Case 1. As $v^*_{i'}$ is the cloudlet with no enough computing resource to accommodate request $r_{i'}$ prior to the arrival of $r_{i'}$ in `Algorithm` 3, i.e., $C_{v^*_{i'}}(i') < c(s_{i'})$, then

$$\begin{aligned}
\psi_{v^*_{i'}}(i') &= \alpha^{1-\frac{C_{v^*_{i'}}(i')}{C_{v^*_{i'}}}} - 1 > \alpha^{1-\frac{c(s_{i'})}{C_{v^*_{i'}}}} - 1 \\
&\geq \alpha^{1-\frac{1}{\log_2 \alpha}} - 1, \quad \text{since } \alpha \leq 2^{\frac{C_{min}}{c_{max}}} \leq 2^{\frac{C_{v^*_{i'}}}{c(s_{i'})}} \\
&= \frac{\alpha}{2} - 1 \geq |V| \cdot u^*_{i'}, \quad \text{since } \alpha \geq 2|V| \cdot u_{max} + 2 \\
&\geq |V| \cdot \frac{u_{min}}{u_{max}} \cdot u^*_{i'}.
\end{aligned} \tag{2.24}$$

Case 2. Because we assign request r_i to the cloudlet $v_{i'}$ with the minimum normalized computing resource cost by Eq. (2.12) in `Algorithm` 3 (cloudlet $v_{i'}$ could be cloudlet $v^*_{i'}$), we have

$$\psi_{v^*_{i'}}(i') \geq \psi_{v_{i'}}(i'). \tag{2.25}$$

Since the admission control policy is violated if request $r_{i'}$ is assigned to cloudlet $v_{i'}$ in `Algorithm` 3, according to the condition (i) of the admission control policy, then

$$\psi_{v_{i'}}(i') > |V| \cdot u_{i'} \geq |V| \cdot \frac{u_{min}}{u_{max}} \cdot u^*_{i'}. \tag{2.26}$$

Thus, we have

$$\psi_{v^*_{i'}}(i') > |V| \cdot \frac{u_{min}}{u_{max}} \cdot u^*_{i'}. \tag{2.27}$$

□

Lemma 5. Given an MEC network $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$ and a finite time horizon that consists of T time slots, let R be the set of requests arriving one by one over the time horizon, denote by $\mathcal{D}(i)$ the set of requests admitted by the optimal solution but rejected by `Algorithm` 3 prior to the arrival of request r_i. Denote by $\mathbb{P}_{opt}(i)$ and $\mathbb{P}(i)$ the total utilities of admitted requests by an optimal solution and the solution delivered by `Algorithm` 3 prior to the arrival of request r_i, respectively. We have

$$\mathbb{P}_{opt}(i) \le \frac{u_{max}}{u_{min}} \cdot \mathbb{P}(i) + \sum_{r_{i'} \in \mathcal{D}(i)} u^*_{i'}. \tag{2.28}$$

Proof. Recall that $\mathcal{D}(i) \cup \mathcal{H}(i)$ is the set of admitted requests by the optimal solution. We have

$$\begin{aligned}\mathbb{P}_{opt}(i) &= \sum_{r_{i'} \in \mathcal{H}(i)} u^*_{i'} + \sum_{r_{i'} \in \mathcal{D}(i)} u^*_{i'} \\ &\le \frac{u_{max}}{u_{min}} \cdot \sum_{r_{i'} \in \mathcal{H}(i)} u_{i'} + \sum_{r_{i'} \in \mathcal{D}(i)} u^*_{i'}, \quad \text{by Eq. (2.22)} \\ &\le \frac{u_{max}}{u_{min}} \cdot \mathbb{P}(i) + \sum_{r_{i'} \in \mathcal{D}(i)} u^*_{i'}. \end{aligned} \tag{2.29}$$

Ineq. (2.29) holds as $\mathcal{H}(i)$ is the subset of admitted requests by `Algorithm` 3. □

Theorem 2.4. *Given an MEC network $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$ and a finite time horizon that consists of T time slots, let R be the set of requests arriving one by one over the time horizon, there is an online algorithm with a competitive ratio of $O(\log |V|)$,* `Algorithm` *3, for the online average total utility maximization problem without the bandwidth capacity constraint, which takes $O(|\mathcal{AP}|^2)$ time to admit each request when $\alpha = 2|V| \cdot u_{max} + 2$, where u_{max} is the maximum utility gain of a request.*

Proof. Recall that $\mathcal{D}(i)$ is the set of requests admitted by the optimal solution but rejected by `Algorithm` 3 prior to the arrival of request r_i. Let $\mathcal{D}(i, v)$ be the set of requests in $\mathcal{D}(i)$ which are assigned to cloudlet v by the optimal solution, i.e., $\mathcal{D}(i, v) = \{r_{i'} \mid r_{i'} \in \mathcal{D}(i), v^*_{i'} = v\}$. It can be seen that $\mathcal{D}(i) = \cup_{v \in V} \mathcal{D}(i, v)$. We have

$$\begin{aligned} &|V| \cdot (\mathbb{P}_{opt}(i) - \frac{u_{max}}{u_{min}} \cdot \mathbb{P}(i)) \le |V| \cdot \sum_{r_{i'} \in \mathcal{D}(i)} u^*_{i'}, \text{ by Lemma 5} \\ &= \sum_{r_{i'} \in \mathcal{D}(i)} |V| \cdot u^*_{i'} < \frac{u_{max}}{u_{min}} \cdot \sum_{r_{i'} \in \mathcal{D}(i)} \psi_{v^*_{i'}}(i'), \quad \text{by Lemma 4} \\ &\le \frac{u_{max}}{u_{min}} \cdot \sum_{r_{i'} \in \mathcal{D}(i)} \psi_{v^*_{i'}}(i) && (2.30) \\ &= \frac{u_{max}}{u_{min}} \cdot \sum_{r_{i'} \in \mathcal{D}(i)} \frac{w_{v^*_{i'}}(i)}{C_{v^*_{i'}}} = \frac{u_{max}}{u_{min}} \cdot \sum_{v \in V} \sum_{r_{i'} \in \mathcal{D}(i,v)} \frac{w_v(i)}{C_v} && (2.31) \\ &\le \frac{u_{max}}{u_{min}} \cdot \sum_{v \in V} w_v(i) \sum_{r_{i'} \in \mathcal{D}(i,v)} \frac{1}{C_v} && (2.32) \\ &\le \frac{u_{max}}{u_{min}} \cdot \sum_{v \in V} w_v(i) && (2.33) \\ &\le 2 \cdot |V| \cdot \frac{u_{max}}{u_{min}} \cdot \log_2 \alpha \cdot \sum_{r_{i'} \in \mathcal{Z}(i)} (c(s_{i'}) \cdot u_{i'}), \quad \text{by Lemma 2,} \end{aligned}$$

where Ineq. (2.30) holds because the utilization of the computing resource does not decrease. Eq. (2.31) holds because $\mathcal{D}(i) = \cup_{v \in V} \mathcal{D}(i, v)$. Ineq. (2.32) holds by $\sum_{i=1}^{p} \sum_{j=1}^{q} \mathcal{A}_i \mathcal{B}_j \leq \sum_{i=1}^{p} \mathcal{A}_i \sum_{j=1}^{q} \mathcal{B}_j$. Ineq. (2.33) holds because the computing resource consumption of each cloudlet is within its capacity, by assuming that the demanded computing resource of each request $r_{i'}$ is no less than 1, i.e., $c(s_{i'}) \geq 1$, and each cloudlet v can accommodate at most $\lfloor C(v) \rfloor$ requests. We thus have

$$\begin{aligned}
\mathbb{P}_{opt}(i) &< \frac{u_{max}}{u_{min}} \cdot \mathbb{P}(i) + 2 \cdot \frac{u_{max}}{u_{min}} \cdot \log_2 \alpha \cdot \sum_{r_{i'} \in \mathcal{Z}(i)} (c(s_{i'}) \cdot u_{i'}) \\
&\leq \frac{u_{max}}{u_{min}} \cdot \mathbb{P}(i) + 2 \cdot \frac{u_{max}}{u_{min}} \cdot \log_2 \alpha \cdot c_{max} \cdot \sum_{r_{i'} \in \mathcal{Z}(i)} \cdot u_{i'}.
\end{aligned}$$

Then,

$$\begin{aligned}
\frac{\mathbb{P}_{opt}(i)}{\mathbb{P}(i)} &< \frac{\frac{u_{max}}{u_{min}} \cdot \mathbb{P}(i) + 2 \cdot \frac{u_{max}}{u_{min}} \cdot \log_2 \alpha \cdot c_{max} \cdot \sum_{r_{i'} \in \mathcal{Z}(i)} u_{i'}}{\mathbb{P}(i)} \\
&= \frac{\frac{u_{max}}{u_{min}} \cdot \sum_{r_{i'} \in \mathcal{Z}(i)} u_{i'} + 2 \cdot \frac{u_{max}}{u_{min}} \cdot \log_2 \alpha \cdot c_{max} \cdot \sum_{r_{i'} \in \mathcal{Z}(i)} u_{i'}}{\sum_{r_{i'} \in \mathcal{Z}(i)} u_{i'}} \\
&= \frac{u_{max}}{u_{min}} + 2 \cdot \frac{u_{max}}{u_{min}} \cdot \log_2 \alpha \cdot c_{max} \\
&= O(\log |V|), \quad \text{when } \alpha = 2|V| \cdot u_{max} + 2.
\end{aligned}$$

The time complexity of `Algorithm` 3 is dominated by the time of finding the shortest paths (delay) from AP l_i to cloudlets, which takes $O(|\mathcal{AP}|^2)$ time. □

Theorem 2.5. *Given an MEC network $G = (\mathcal{AP} \cup V \cup \{v_0\}, E)$ and a finite time horizon that consists of T time slots, let R be the set of requests arriving one by one over the time horizon, there is an online algorithm,* `Algorithm` *4, for the online average total utility maximization problem with the bandwidth capacity constraint.*

Proof. It can be seen that the solution delivered by `Algorithm` 4 is feasible because all constraints imposed on the problem are met.

The time complexity of `Algorithm` 4 for a request admission is dominated by the time of finding the routing paths with the least communication delay from AP l_i to cloudlets with sufficient resource for request r_i, which takes $O(|\mathcal{AP}|^2)$ time. □

2.5 Performance Evaluation

In this section, we conduct the performance evaluation of the proposed algorithms. We also investigate the impact of important parameters on the performance of the proposed algorithms.

2.5.1 Environment setting

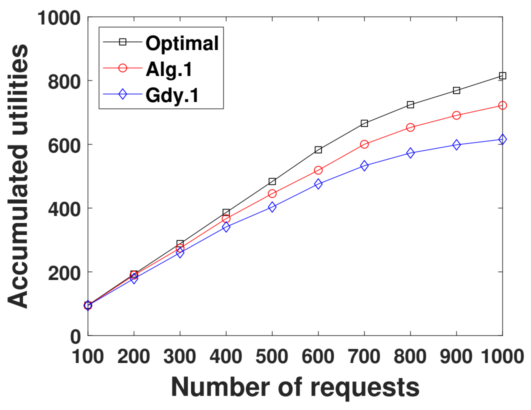
We consider a heterogeneous MEC network consisting of 200 APs, 10% of which are co-located with cloudlets [109]. The topologies of MEC networks are generated by GT-ITM [34]. For each AP, the bandwidth at each time slot is drawn from 20 MHz to 40 MHz randomly [97], the signal-to-noise ratio (i.e., $\frac{\kappa_i}{\sigma^2}$) of an AP is set as 30 dB [103]. For each cloudlet, the capacity varies from $3,000$ MHz to $7,000$ MHz [108] and its processing rate varies from 0.5 MB to 2 MB per ms [70]. For each request, its task size is randomly drawn from 1 MB to 5 MB [90], the demanded computing resource is randomly drawn from 20 MHz to 300 MHz [109] and the demanded bandwidth resource ranges from 5 Mbps to 50 Mbps [115]. The delay requirement threshold of a request is randomly drawn from 10 ms to 50 ms [70], and β_i ranges from 1 to 3. The bandwidth capacity of each link varies from 200 Mbps to $2,000$ Mbps [70], and the transmission delay of a link at each time slot is chosen from 2 ms to 5 ms randomly [108], while the transmission delay from an AP to the remote cloud through the gateway varies from 80 ms to 100 ms. We further assume the processing rate of the remote cloud is 20 MB per ms. λ is set as 2 and ϵ is set as 0.5. The turning parameters α and δ are set as $2|V| \cdot u_{max} + 2$, where $|V|$ is the number of cloudlets and $u_{max} = \lambda - 1$. We assume that there are 100 time slots and 1000 requests arrive at each time slot one by one. The duration of each request is randomly drawn from 1 to 3 time slots [70]. The result in each figure is the mean of the results by applying an algorithm on 20 MEC network instances of the same size, where the running time of an algorithm is the actual amount of time of finding a solution, based on a desktop with a 3.60 GHz Intel 8-Core i7-7700 CPU and 16 GB RAM. Unless specified, the above parameters will be adopted in the default setting.

To evaluate `Algorithm` 1 (referred to as `Alg.1`) for the total utility maximization problem without the bandwidth capacity constraint, we propose two comparison benchmarks. One is the ILP solution (2.7) (referred to as `Optimal`) which is the optimal solution to the problem; another is a greedy algorithm (referred to as `Gdy.1`), which picks requests in R randomly, and assigns the picked request to the cloudlet (or the remote cloud) with the maximum utility gain, this procedure continues until all requests are assigned. To study `Algorithm` 2 (referred to as `Alg.2`) for the total utility maximization problem with the bandwidth capacity constraint, we also give a comparison algorithm for it, which is a greedy algorithm (referred to as `Gdy.2`) that requests are picked randomly. For each picked request, it first finds a routing path with the least communication delay from the location of the request to each cloudlet with sufficient computing resource, through the links with sufficient bandwidth resource for the request. It then assigns the request to the cloudlet (or the remote cloud) with the maximum utility gain. This procedure continues until all requests are assigned.

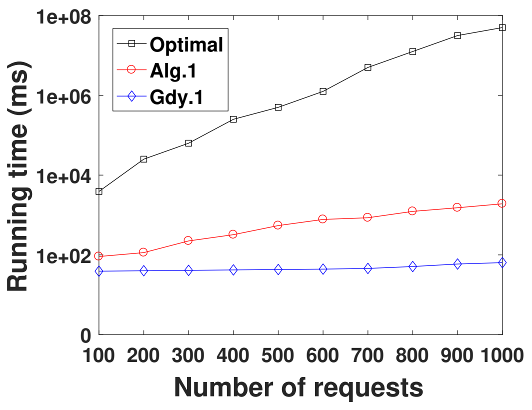
To investigate the performance of `Algorithm` 3 (referred to as `Alg.3`) for the online average total utility maximization problem without the bandwidth capacity constraint, a comparison online algorithm (referred to as `Gdy.3`) is proposed, which is the online version of `Gdy.1`. To evaluate `Algorithm` 4 (referred to as `Alg.4`) for the online average total utility maximization problem with the bandwidth capacity constraint, a greedy algorithm (referred to as `Gdy.4`), which is the online version of `Gdy.2`, is also proposed for the performance evaluation purpose.

2.5.2 Performance evaluation of the proposed algorithms for the total utility maximization problem

We first studied the performance of `Alg.1` against algorithms `Optimal` and `Gdy.1`, by varying the number of requests from 100 to 1,000. We then evaluated the performance of `Alg.2` against algorithm `Gdy.2`, by varying the number of requests from 100 to 1,000. Fig. 2.2 and Fig. 2.3 depict the accumulated utilities and running times of different algorithms for the total utility maximization problem without and with the bandwidth capacity constraint. It can be seen from Fig. 2.2(a) that when the number of requests reaches 1,000, the performance achieved by algorithm `Gdy.1` is 88.5% of that by `Alg.1` while the performance achieved by `Alg.1` is 85.2% of that by algorithm `Optimal`. Meanwhile, it can be seen from Fig. 2.3(a) that `Alg.2` outperforms algorithm `Gdy.2` on the performance improvement by at least 10.8% with 1,000 requests. The rationale behind is that both `Alg.1` and `Alg.2` better utilize the network resource by provisioning satisfied services to more users, compared with the greedy algorithms, and they take much less running time in comparison with the ILP solution that takes a much longer running time.



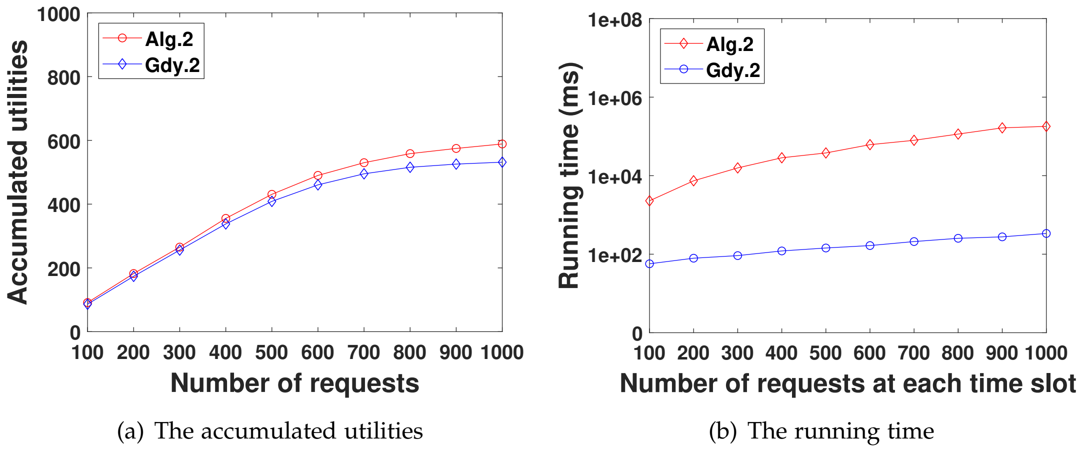
(a) The accumulated utilities



(b) The running time

Figure 2.2: Performance of different algorithms for the total utility maximization problem without the bandwidth capacity constraint.

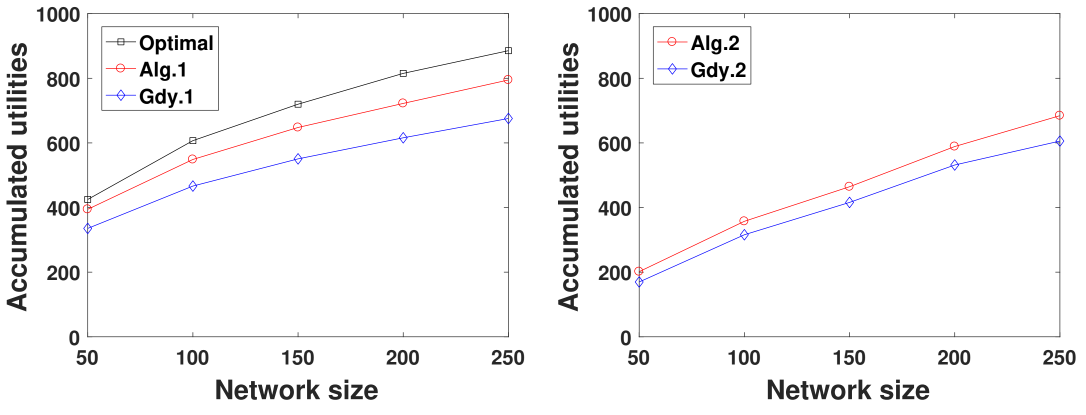
We then studied the impact of network size on the proposed algorithms with 1,000 requests, by varying the number of APs from 50 to 250. Recall that 10% of APs are co-located with cloudlets. Fig. 2.4(a) and Fig. 2.4(b) demonstrate the accumulated utilities by different algorithms for the total utility maximization problem without and with the bandwidth capacity constraint. We can see from Fig. 2.4(a) that when the network size is 250, the performance achieved by algorithm `Gdy.1` is 76.3% of that by `Alg.1` while the performance achieved by `Alg.1` is 84.8% of that by `Optimal`. The



(a) The accumulated utilities (b) The running time

Figure 2.3: Performance of different algorithms for the total utility maximization problem with the bandwidth capacity constraint.

similar performance behaviors can be observed in Fig. 2.4(b) too. This is because both `Alg.1` and `Alg.2` facilitate the efficient cooperation between the remote cloud and local cloudlets to maximize the accumulated user satisfaction when the network size is large.



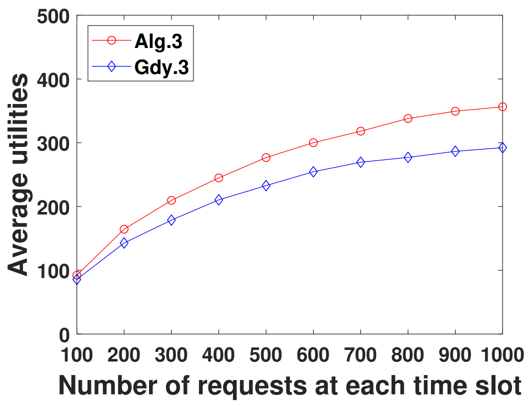
(a) Performance of different algorithms for the total utility maximization problem without the bandwidth capacity constraint

(b) Performance of different algorithms for the total utility maximization problem with the bandwidth capacity constraint

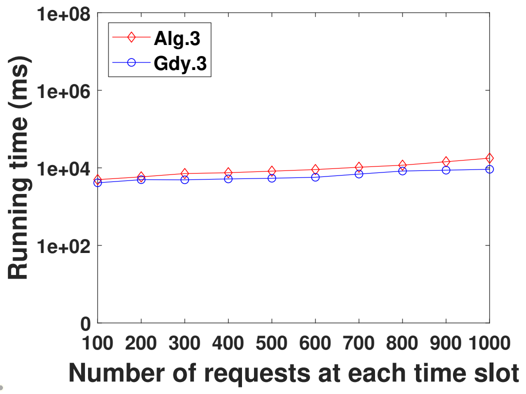
Figure 2.4: The impact of network size on the performance of the proposed algorithms

2.5.3 Performance evaluation of the proposed algorithms for the online average total utility maximization problem

We first studied the performance of `Alg.3` and `Alg.4` against algorithms `Gdy.3` and `Gdy.4`, respectively, by varying the number of requests arriving at each time slot from 100 to 1,000, over the time horizon (100 time slots). Fig. 2.5 and Fig. 2.6 plot the average utilities and running times of different algorithms for the online average total utility maximization problem without and with the bandwidth capacity constraint. With 1,000 requests arriving at each time slot, in Fig. 2.5(a) algorithm `Alg.3` outperforms `Gdy.3` by 22.1%, while in Fig. 2.3(a) algorithm `Alg.4` outperforms `Gdy.4` by 16.4%. This can be justified by that either `Alg.3` or `Alg.4` establishes an efficient admission control policy to admit requests with larger utility gain but less resource consumption, without any knowledge of future request arrivals.

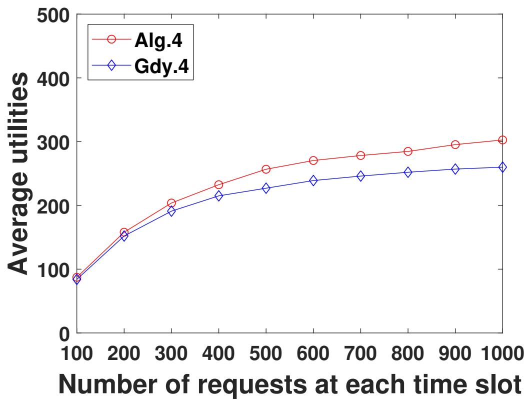


(a) The average utilities

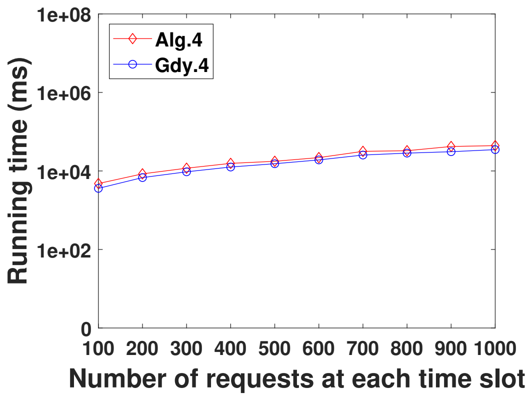


(b) The running time

Figure 2.5: Performance of different algorithms for the online average total utility maximization problem without the bandwidth capacity constraint.



(a) The average utilities

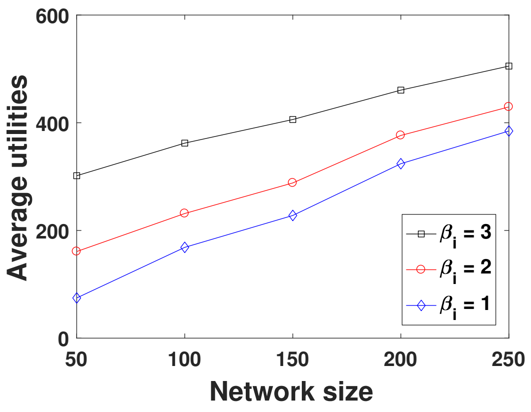


(b) The running time

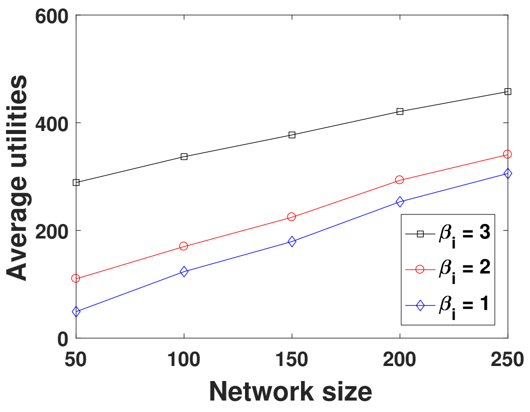
Figure 2.6: Performance of different algorithms for the online average total utility maximization problem with the bandwidth capacity constraint.

The rest is to investigate the impact of important parameters on the performance of the proposed algorithms, including parameter β_i, parameter α and parameter δ. We also study the performance of the online algorithms with and without adopting the admission control policy. Recall that parameter β_i reflects the service delay tolerance of user u_i, while parameter α and parameter δ reflect the sensitivity of the workload at each cloudlet and link, respectively. For the sake of convenience, in the rest experimental simulations, it is assumed that the time horizon consists of 100 time slots and 1,000 requests arrive at each time one by one.

We first evaluated the impact of parameter β_i on the performance of the proposed algorithms, by varying the network size from 50 to 250. Fig. 2.7 illustrates the impact of parameter β_i on the proposed algorithms `Alg.3` and `Alg.4` when $\beta_i = 1$, 2, and 3 respectively. It can be seen from Fig. 2.7(a) that when the network size is 50, the performance of `Alg.3` with $\beta_i = 1$ is 24.8% of itself with $\beta_i = 3$. And when the network size is 250, the performance of `Alg.3` with $\beta_i = 1$ is 76.2% of itself with $\beta_i = 3$. The similar performance behavior can be found in Fig. 2.7(b). The rationale behind is that a larger β_i leads to a larger tolerable service delay, and more requests can be admitted. In addition, when the network size is small (i.e., the available computing resource is very limited), the mobile users have to utilize the remote cloud better to process their requests, resulting in longer service delays. Thus, a larger β_i is important in admitting requests when the network size is small.



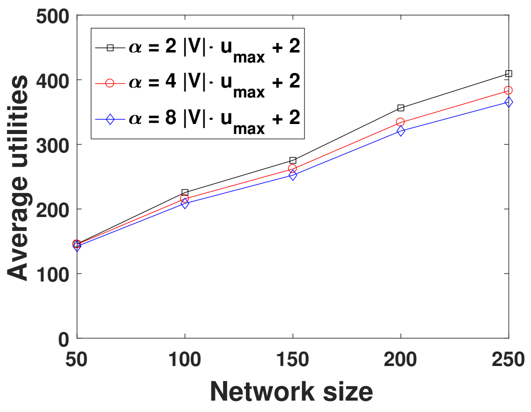
(a) Performance of `Alg.3` for the online average total utility maximization problem without the bandwidth capacity constraint



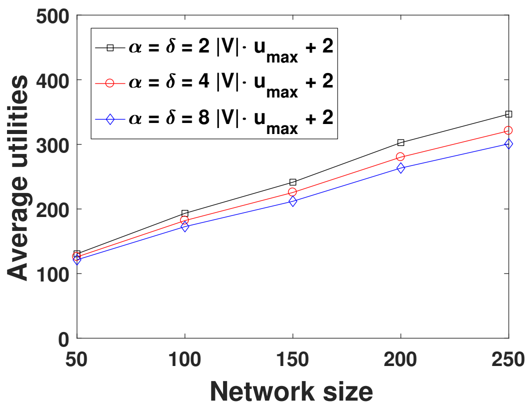
(b) Performance of `Alg.4` for the online average total utility maximization problem with the bandwidth capacity constraint

Figure 2.7: The impact of β_i on the performance of the proposed algorithms.

We then studied the impact of parameter α on the performance of the algorithm `Alg.3`, and the impacts of parameter α and δ on the performance of the algorithm `Alg.4`, by varying the network size from 50 to 250. Fig.2.8(a) demonstrates the performance of `Alg.3` with parameter $\alpha = 2|V| \cdot u_{max} + 2$, $4|V| \cdot u_{max} + 2$, and $8|V| \cdot u_{max} + 2$, respectively, where $|V|$ is the number of cloudlets, and $u_{max} = \lambda - 1$ is the maximum possible utility gain for a request. While Fig.2.8(b) shows the performance of `Alg.4` with parameter $\alpha = \delta = 2|V| \cdot u_{max} + 2$, $4|V| \cdot u_{max} + 2$, and $8|V| \cdot u_{max} + 2$, respectively. As depicted by Fig.2.8(a), when the network size is 250, the performance of `Alg.3` with $\alpha = 8|V| \cdot u_{max} + 2$ is 89.3% of itself with $\alpha = 2|V| \cdot u_{max} + 2$. The similar performance behavior can be found in Fig. 2.8(b). The justification is that with a larger α or δ, the normalized cost of computing resource or bandwidth resource becomes higher by Eq. (2.12) and Eq. (2.14), and it intends to be conservative and reject requests.



(a) Performance of `Alg.3` for the online average total utility maximization problem without the bandwidth capacity constraint



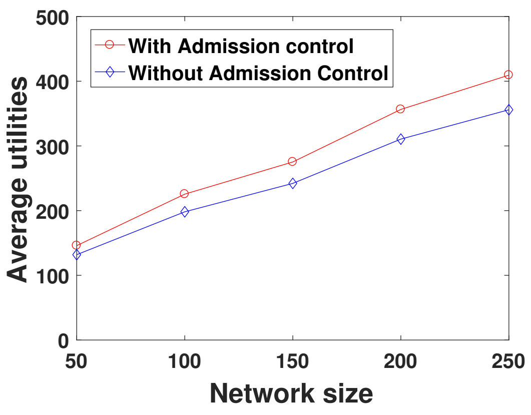
(b) Performance of `Alg.4` for the online average total utility maximization problem with the bandwidth capacity constraint

Figure 2.8: The impacts of the parameter α and δ on the performance of the proposed algorithms.

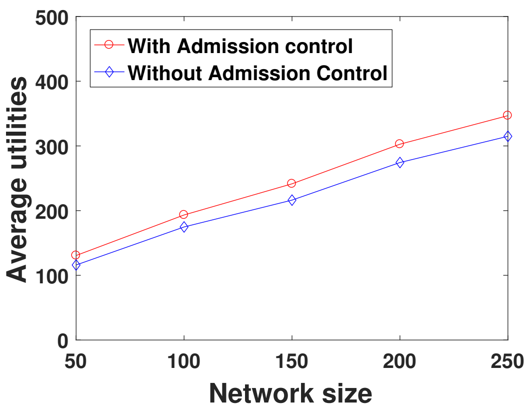
We finally investigated the impact of the admission control policy, by varying the network size from 50 to 250. Fig. 2.9(a) and Fig. 2.9(b) plot the performance of `Alg.3` and `Alg.4` with and without the admission control policy. It can be seen from Fig. 2.9(a) that when the network size is 250, the performance of `Alg.3` without the admission control policy is 86.9% of itself with the admission control policy. The similar performance behavior can be found in Fig. 2.9(b). This can be justified by that with a reasonable admission control policy, the requests with larger utility gains but less computing resource consumption will be admitted. Thus, the admission control policy is important to deal with the dynamic request admissions.

2.6 Summary

In this chapter, we studied the user satisfaction on the use of services for delay-sensitive IoT applications in an edge computing environment, by offloading user service requests to either the remote cloud or local cloudlets in an MEC network. We first formulated two novel optimization problems and showed their NP-hardness. We then proposed efficient approximation and heuristic algorithms for the admissions of a set of requests. We also developed online algorithms for the admissions of dynamic requests without the knowledge of future arrivals. We finally evaluated the performance of the proposed algorithms through experimental simulations. Experimental results demonstrate that the proposed algorithms are promising.



(a) Performance of `Alg.3` for the online average total utility maximization problem without the bandwidth capacity constraint



(b) Performance of `Alg.4` for the online average total utility maximization problem with the bandwidth capacity constraint

Figure 2.9: The impacts of the admission control policy on the performance of the proposed algorithms.

Chapter 3

Service Provisioning for Multi-Source IoT Applications

In this chapter, we first formulate two novel optimization problems: the cost minimization problem for a single multi-source Internet of Things (IoT) application, and the cost minimization problem for a set of multi-source IoT applications, respectively. We then propose a service provisioning framework in the Mobile Edge Computing (MEC) network for multi-source IoT applications. We also devise an efficient algorithm for the cost minimization problem built upon the proposed service provisioning framework, and further extend the solution for the cost minimization problem for a set of multi-source IoT applications.

3.1 Introduction

With the fast development of 5G, MEC promises to greatly reduce data processing delays of IoT services, by deploying computing resource (e.g., cloudlets) in the proximity of IoT devices [2]. Thus, IoT devices can directly forward their streaming sensory data to virtual services instantiated in Virtual Machines (VMs) in cloudlets of the MEC network. A typical IoT application usually collects sensory data from multiple sources (e.g., different IoT devices) located at different geographical locations, aggregates the streaming data at some intermediate nodes (cloudlets), and ultimately routes the aggregated data stream to a destination (a cloudlet) for further processing and storage. To ensure the security and privacy of data stream routing in the MEC network, various network service functions such as firewalls, intrusion detection systems, proxies, and load balancers may also be deployed. For example, IoT applications usually need a sequence of network service functions for data aggregation and filtering, to guarantee the real-time in-network processing (summation, averaging, maximum or minimum) of IoT data streams. Such a sequence of network service functions is referred to as a *Service Function Chain* (SFC). Conventional network functions are implemented by dedicated hardware - middleboxes, making their deployment and maintenance very expensive and not agile. Network Function Virtualization (NFV) [20; 21; 35] as a new technology promises to provide inexpensive and flexible network services, and implements the Virtual Network Functions (VNFs) as software in VMs or containers in cloudlets. NFV makes virtual service provisioning becomes affordable, and easy to adopt and maintain.

The novelties of the work in this chapter lie in formulating two novel cost minimization problems for multi-source IoT applications in an MEC network, where each multi-source IoT application has multiple data streams from different sources to be uploaded to the MEC network for processing and storage, while each data stream must pass through the network functions in the SFC of the IoT application prior to reaching its destination. A service provisioning framework for such multi-source IoT applications is proposed through VNF instance placement and sharing, workload balancing among cloudlets, and in-network aggregation of data streams. Efficient algorithms for service provisioning of multi-source IoT applications, built upon the proposed framework, are also proposed. To the best of our knowledge, this is the first work on service provisioning for multi-source IoT applications in MEC networks with multiple critical constraints.

The main contributions of this chapter are given as follows. We consider the service provisioning in an MEC network for multi-source IoT applications with SFC requirements. We first formulate two optimization problems: the cost minimization problem for a single multi-source IoT application and the cost minimization problem for a set of multi-source IoT applications, respectively, and show that both problems are NP-hard. We then propose a novel service provisioning framework in an MEC network for a single multi-source IoT application, which consists of multiple data streams uploading from different subnetworks through wireless access points (gateway nodes), data stream in-network aggregation and routing, VNF instance placement and sharing, and workload balancing among cloudlets in the MEC network. Due to the NP-hardness of the two defined problems, we thirdly devise efficient heuristic algorithms for them. We finally evaluate the performance of the proposed algorithms through experimental simulations. Experimental results demonstrate that the proposed algorithms are promising, and outperform their comparison counterparts.

The rest of the chapter is organized as follows. Section 3.2 introduces notions, notations, and the problem definitions. Section 3.3 shows that the defined problems are NP-hard. Section 3.4 proposes a novel service provisioning framework for multi-source IoT applications with SFC requirements. Section 3.5 devises an algorithm for the cost minimization problem for a single multi-source IoT application. Section 3.6 extends the proposed algorithm to the service provisioning of a set of multi-source IoT applications through network slicing. Section 3.7 evaluates the proposed algorithms empirically, and a summary is given in Section 3.8.

3.2 Preliminaries

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

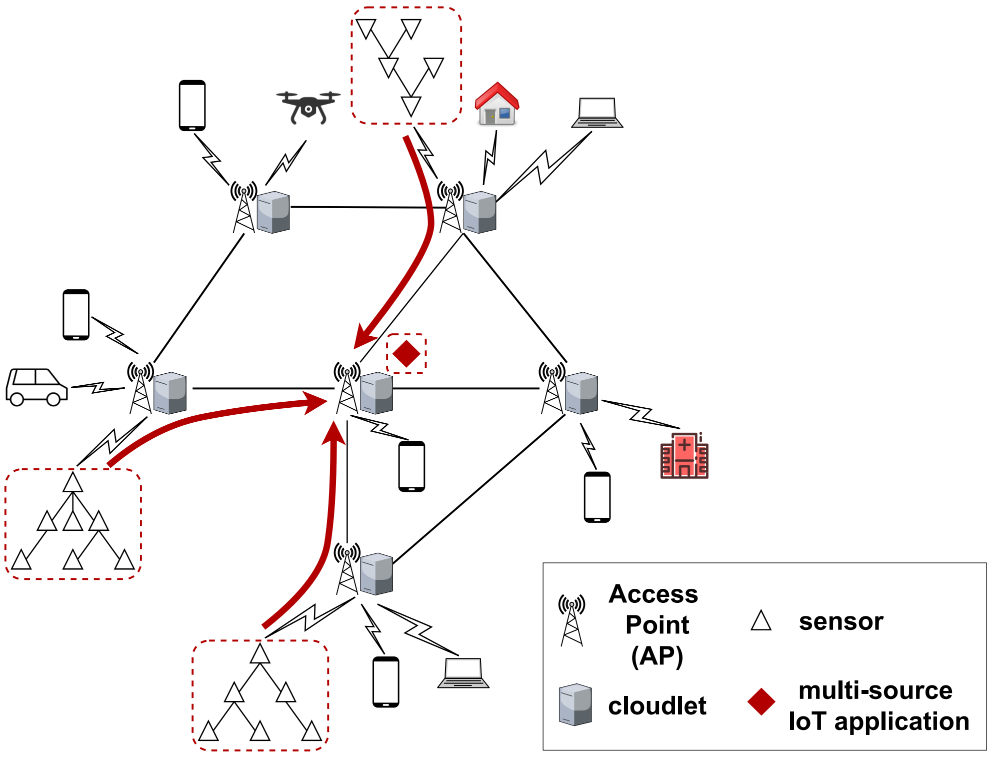


Figure 3.1: An illustrative example of an MEC network that consists of 6 APs, and there are 6 cloudlets co-located with the APs. There is a destination cloudlet (service provisioning) for a multi-source IoT application, where the multi-source IoT application contains 3 data stream sources with each from a different sensor subnetwork of the IoT application.

3.2.1 System Model

An MEC network is represented as an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of links between nodes. Each node $v \in V$ is an Access Point (AP). Associated with each AP, there is a co-located cloudlet $v \in V$ (edge cloud) with computing capacity $C_v > 0$. The AP and its co-located cloudlet are connected through a high-speed optical cable, and the communication delay between each AP and its co-located cloudlet thus is negligible. Assume that the MEC network supports a set of services in which each service is associated with a set of VNFs. Denote by $\mathcal{F} = \langle f_1, f_2, \ldots, f_{|\mathcal{F}|} \rangle$ the set of VNFs in the system, where the implementation of VNF $f_j \in \mathcal{F}$ consumes the amount c_{f_j} of computing resource of a cloudlet. One implementation of f_j is termed as one of VNF instances of the function and each such a VNF instance has a data processing capacity μ_{f_j}, where $1 \leq j \leq |\mathcal{F}|$. Figure 3.1 is an illustrative example of an MEC network.

3.2.2 Problem definitions

Given an MEC network $G = (V, E)$, we consider a multi-source IoT application, which provides continuous surveillance to monitor different geographical areas (e.g., public parks) in a metropolitan region, and different types of IoT sensory devices are deployed in the monitoring areas. The data streams generated from these IoT devices can then be aggregated within their subnetworks and uploaded to the MEC through their gateways (APs). We further assume that there is a destination node (cloudlet) in the MEC for the aggregated data storage of each IoT application, where all updated streaming data from the gateways (APs) of the IoT application will be stored for user ad hoc queries. When the stream data from each gateway is routed to the destination node, the data stream may be merged or aggregated with the data streams from the other gateways of the application. However, each data stream from a gateway to the destination node must pass through the specified VNF instances in the SFC to ensure the security and privacy of the data stream and network performance. Thus, the VNF instances of each function in the SFC must be instantiated in the cloudlets of their routing paths and some of the VNF instances can be shared among the data streams from different gateways of the IoT application. For the sake of convenience, in the rest of our discussion, we assume that there are sufficient computing and bandwidth resources in the MEC to accommodate the VNF instances and data stream routing for the IoT application. In summary, we aim to build a VNF network slicing for each multi-source IoT application such that the operational cost of implementing the application is minimized, in terms of the computing and communication resource consumptions, and the workload balancing among cloudlets involved. We refer to this IoT-driven optimization problem as *the cost minimization problem for a single multi-source IoT application in MEC*. An illustrative example of such an IoT application is given in Figure 3.1.

Definition 1. Given an MEC $G = (V, E)$, and a multi-source IoT application with a Service Function Chain (SFC) requirement $\langle f_1, f_2, \dots f_L \rangle$, in which there is a set $S = \{s_1, s_2, \dots, s_K\} \subset V$ of sources and one destination $d \in V$ for the application. Each source s_i has a streaming data rate ρ_{s_i} with $1 \le i \le K$, *the cost minimization problem for a single multi-source IoT application* is to find a data routing tree T in G rooted at $d \in V$ and spanning all nodes in S, and place the demanded number of VNF instances of each f_j in the SFC with $1 \le j \le L$ at some nodes in T that meets the following conditions: (1) the operational cost of implementing the IoT application is minimized; (2) the data stream in the tree path from each s_i to the destination d must pass through the VNF instances of functions in the SFC in order; (3) there is a defined *data aggregation function* $g(v)$ at each non-leaf node v in T, and the value of $g(v)$ is determined by the values of its l children: $g(v_1), g(v_2), \dots, g(v_l)$, where $v_1, \dots, v_l$ are the children of node v. Particularly, $g(v) = \rho_v$ for each leaf node (source) v in T. For example, the value of function $g(\cdot)$ can be the sum of the data rates of all data streams of its children, or the average of the data rates of its children, depending on which types of IoT applications; and (4) the workload at each cloudlet (in terms of the total amount of computing resource consumed for hosting VNF instances of different functions) is as balanced as possible. The optimization objective is to minimize the operational cost of T, where the operational cost of implementing a multi-source IoT application consists of the computing cost, communication cost, and workload balancing cost that are defined as follows.

The computing cost: For each node v in the data routing tree T with the accumulative data steam $g(v)$, the computing cost at node v for accommodating the VNF instances of the multi-source IoT application is

$$C_{comp}(v) = \sum_{j=1}^{L} I(v, f_j) \cdot \lceil \frac{g(v)}{\mu_{f_j}} \rceil \cdot c_{f_j} \cdot c_{comp}, \tag{3.1}$$

where $I(v, f_j)$ is an indication function, which is 1 if there is any VNF instance of f_j in cloudlet v; otherwise 0, and the range of j is $1 \leq j \leq L$, and $v \in V$. $\lceil \frac{g(v)}{\mu_{f_j}} \rceil$ is the number of VNF instances of f_j required to process the accumulative data stream at node v, and c_{f_j} is the amount of computing resource of an instance of VNF f_j. The value of c_{comp} is the price of a unit computing resource consumption at any cloudlet $v \in V$. If node v is a leaf node in T and assuming that node v is source s_i, then $g(v) = \rho_{s_i}$; otherwise, assuming that v has l children $v_1, \ldots, v_l$, then $g(v) = g(g(v_1), \ldots, g(v_l))$, where $g(v)$ is an aggregation function at node v, e.g., $g(v)$ can be a summation function with $g(v) = \sum_{i=1}^{l} g(v_i)$.

The communication cost: The communication cost of transmitting a volume $\rho(u, v)$ of data on an edge (u, v) from node u to node v in G is

$$C_{comm}(u, v) = \rho(u, v) \cdot b_{comm}(u, v), \tag{3.2}$$

where $\rho(u, v)$ is the accumulative volume of data streams transmitted from node u to node m, and $b_{comm}(u, v)$ is the price of unit data transfer along edge (u, v).

The workload balancing cost: As the workload W_v of each cloudlet $v \in V$ is the actual computing resource consumption of cloudlet v, it is well-known that a heavily loaded cloudlet usually has a much longer processing delay, compared to a lightly loaded cloudlet. Therefore, a penalty cost of this delay should be taken into account, which can be expressed by *the workload utility* of the cloudlet as follows.

Denote by γ_v the workload utility of cloudlet v for an IoT application which is defined as

$$\gamma_v = \frac{W_v}{C_v} = \frac{\sum_{j=1}^{L} I(v, f_j) \cdot \lceil \frac{g(v)}{\mu_{f_j}} \rceil \cdot c_{f_j}}{C_v}, \tag{3.3}$$

where $W_v = \sum_{j=1}^{L} I(v, f_j) \cdot \lceil \frac{g(v)}{\mu_{f_j}} \rceil \cdot c_{f_j}$ is the computing resource consumption on cloudlet v, and C_v is the computing capacity of cloudlet v.

The workload balancing cost at each cloudlet v thus is defined as follows.

$$cost_{load}(v) = \beta_{load} \cdot (2^{\gamma_v} - 1), \tag{3.4}$$

where β_{load} is a given coefficient of the workload balancing cost with $\beta_{load} > 0$, and $0 \leq cost_{load}(v) \leq \beta_{load}$ since $0 \leq \gamma_v \leq 1$. Note that the workload balancing cost is modeled as an exponential function of the workload utility, as a workload balancing cost increases with the increase on the workload utility, while the increasing rate of the workload balancing cost is supposed to become faster with the increase of the workload utility.

The optimization objective of the cost minimization problem for a single multi-source IoT application thus is to find a data routing tree T in G such that its operational cost $c(T)$ is minimized, where

$$c(T) = \sum_{v \in V(T)} C_{comp}(v) + \sum_{(u,v) \in E(T)} C_{comm}(u,v) + \sum_{v \in V(T)} cost_{load}(v),$$

where $V(T)$ and $E(T)$ are the set of nodes and edges in the data routing tree T, respectively. Notice that the solution of the problem consists of the construction of a data routing tree T, the placement of VNF instances of each VNF f_j in the SFC at some cloudlet nodes in T, and the workload balancing among cloudlets involved.

Remark 1. We may define another weighted version of the objective function as follows:

$$c'(T) = \alpha \cdot \sum_{v \in V(T)} C_{comp}(v) + (1 - \alpha) \cdot \sum_{(u,v) \in E(T)} C_{comm}(u,v),$$

where $\sum_{v \in V(T)} C_{comp}(v)$ and $\sum_{(u,v) \in E(T)} C_{comm}(u,v)$ are the computing cost and the communication cost, respectively, while α is a constant with $0 \leq \alpha \leq 1$.

Through adopting such a weighted version of the objective function, we can configure the value of α and investigate the impact of different network resource types.

So far, we have formulated a cost minimization problem for a single multi-source IoT application in an MEC network, through performing NFV-enabled network slicing to accommodate the IoT service. However, the MEC network as a public service platform is expected to accommodate different IoT applications for different users while minimizing the accumulative operational cost of implementing the IoT applications. We thus define another cost minimization problem for a set of multi-source IoT applications in the MEC network as follows.

Definition 2. Given an MEC network $G = (V, E)$ and a set $\mathcal{I}$ of multi-source IoT applications with each having an SFC requirement, assume that the VNF instances of different IoT applications cannot be shared with each other due to security and privacy concerns. *The cost minimization problem for a set of multi-source IoT applications* is to implement all IoT applications in $\mathcal{I}$ by finding a data routing tree in G for each application in $\mathcal{I}$ such that the total operational cost of implementing all IoT applications in $\mathcal{I}$ is minimized, assuming that there are sufficient resources in G to accommodate all the IoT applications, i.e., the optimization objective is to minimize

$$\sum_{r_i \in \mathcal{I}} c(T_{r_i}), \tag{3.5}$$

where $r_i \in \mathcal{I}$ is an IoT application, T_{r_i} is the data routing tree in G for $r_i \in \mathcal{I}$, and $c(T_{r_i})$ is the operational cost of T_{r_i} in Eq. (3.5) with $1 \leq i \leq |\mathcal{I}|$. In other words, the problem is to build $|\mathcal{I}|$ VNF-enabled network slicing in the MEC for all IoT applications in $\mathcal{I}$. In case the MEC network G does not have sufficient resources to accommodate all IoT applications, the problem then is to admit as many IoT applications as possible while minimizing the accumulative operational cost of admitted IoT applications.

3.3 NP-hardness of the defined problems

In this section, we show that the defined two problems are NP-hard.

Theorem 3.1. *The cost minimization problem for a single multi-source IoT application in an MEC network $G = (V, E)$ is NP-hard.*

Proof. We consider a very special case of the problem of concern, where the workload balancing at each cloudlet will not be considered, and the data stream rates of all sources of an IoT application are identical, i.e., $\rho = \rho_1 = \rho_2 = \ldots = \rho_K$, the aggregate function $g(\cdot)$ at each node is the average of the sum of data stream rates of its children, i.e., $g(v) = \frac{\sum_{i=1}^{l} g(v_i)}{l} = \rho$, where node v has l children $v_1, v_2, \ldots, v_l$, and all VNF instances of each service function in the SFC are instantiated at the destination node d of the IoT application. This special cost minimization problem for a single multi-source IoT application then is equivalent to the Steiner tree problem in G that finds a Steiner tree rooted at the destination node d and spanning all source nodes in $S \subset V$ such that the weighted sum of edges in the tree is minimized. It is well known that the Steiner tree problem is NP-hard [14], the cost minimization problem for a single multi-source IoT application thus is NP-hard, too. □

Theorem 3.2. *The cost minimization problem for a set of multi-source IoT applications in an MEC network $G = (V, E)$ is NP-hard.*

Proof. When $|\mathcal{I}| = 1$, the problem becomes the cost minimization problem for a single multi-source IoT application in MEC, which is NP-hard by Theorem 3.1. Thus, the cost minimization problem for a set of multi-source IoT applications is NP-hard, too. □

3.4 A novel framework for IoT-driven service provisioning

In this section, we provide a generic service provisioning framework for multi-source IoT applications in an MEC network. We consider an IoT application that consists of multiple sources, where each source is a base station (an AP) of a sensor subnetwork. The streaming sensory data from the sensors in the subnetwork are collected at the source (the base station or the gateway) and will be uploaded to the MEC network for further processing and storage. We assume that there is a destination node d in the MEC for the aggregated data storage. To ensure the security and privacy of transferring data stream, each data stream from its source to the destination must pass through a specified SFC as the requirement of the IoT application. For example, in a metropolitan region, there are many public parks. Assuming that a sensor network is deployed for each park, the monitored data stream from each sensor network will be uploaded to the MEC through its nearby AP. The collected data will be finally stored at the destination node for storage and processing.

A naive approach for sensory data collection from sources to the destination node is to build a routing path in the MEC network from each source (an AP) to the destination node, and place the demanded number of VNF instances of each service function in the SFC along the cloudlets in the routing path to meet the data rate of the source. This IoT-driven service in an MEC network is expected for a long run, which implies that the owner of the IoT application will pay its IoT service at the cost proportional to the resource occupied by the service. However, running IoT services through this naive method is not economical, because the data streams from different sources of an IoT application can be aggregated or merged through exploring temporal-spatial data correlations. A much less accumulative volume of data streams of these sources can be routed to the destination node, thereby reducing the communication cost. On the other hand, the certain number of VNF instances of each service function in the SFC for the data stream of each source must be instantiated in the nodes in the routing path. The number of VNF instances of each service function in the SFC for the aggregated data stream could be significantly reduced if the data stream aggregation at each intermediate node is performed, implying that less computing resource will be consumed.

To provide a cost-effective service in MEC for a multi-source IoT application, a data routing tree rooted at the destination node and spanning all source nodes in the MEC can be built to reduce both computing and communication costs through data aggregation and VNF instance instantiations at nodes in the routing paths. Each non-leaf node v in the tree except the destination has a parent node and a set of children nodes. There is an aggregation function $g(v)$ at v that aggregates the data streams from its children, e.g., a simple aggregation function is the sum of the data streams of its children. Also, less numbers of VNF instances of a VNF in the SFC can be instantiated at node v for its aggregated data processing. However, the data stream along the unique routing path in the tree from each source to the destination must pass through its demanded number of VNF instances in the SFC. Thus, both communication and computing resources for the IoT application can be shared among the data streams from different sources of the application through the data routing tree. Figure 3.2 is an illustrative example of a data routing tree for an IoT application.

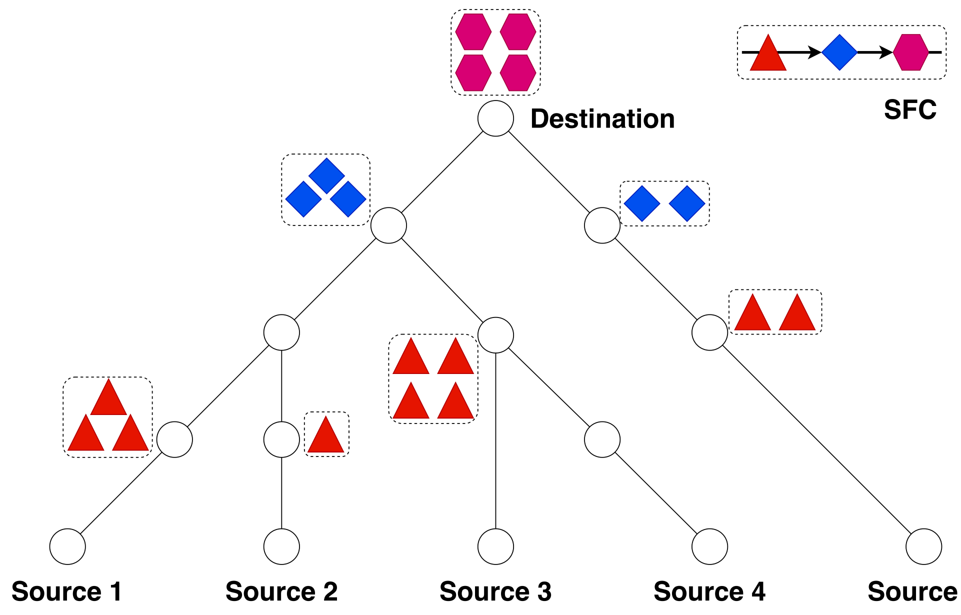


Figure 3.2: An illustrative example of a data routing tree for a multi-source IoT application with three service functions in its SFC. There are 5 sources $s_1, s_2, \ldots, s_5$, and the data stream from each source to the destination node (the tree root) must pass through the VNF instances of each service function in the SFC.

3.5 Algorithm for cost minimization problem for a single multi-source IoT application

Since the cost minimization problem for a single multi-source IoT application in MEC is NP-hard, in this section we first develop an efficient algorithm for it, based on the proposed service framework. We then analyze the properties of the solution delivered and the time complexity of the proposed algorithm.

3.5.1 Overview of the proposed algorithm

The proposed algorithm proceeds greedily. We assume that the destination (a cloudlet) of each multi-source IoT application is given in advance. Let S be the set of all sources of the IoT application. For each source node $s \in S$, a data routing tree $T(s)$ is constructed by setting the destination node d as the tree root, adding node $s \in S$ as the first source node to the tree, followed by adding the other source nodes to $T(s)$ greedily, one by one. Each time, a source node with the minimum cost increment compared with the previous cost of $T(s)$ will be chosen to add to the tree until all source nodes are added to the tree. Thus, a data routing tree $T(s_0)$ with the minimum operational cost is chosen from the $|S|$ trees, which will be the solution to the problem. Specifically, we first choose a source node $s \in S$ as the very first node to add to the data routing tree $T(s)$. To this end, we find a shortest path P_1 in G

from s to the destination d, and place the VNF instances of each function $f \in SFC$ to the nodes along path P_1 while balancing the workload of the nodes in the path (the detailed VNF instance placement will be discussed later), where the number of VNF instances of each service function in the SFC is determined by the data rate of source s. Let $S' = S \setminus \{s\}$ be the set of the rest source nodes that have not been added to $T(s)$ yet. The rest source nodes in S' then are added to tree $T(s)$ iteratively, one by one. We claim that the data stream along the tree path from each source node to the destination must pass through its demanded number of VNF instances of each service function only once, i.e., no VNF instances of a service function is wrongly placed, or placed in multiple nodes in the path.

Let $T(s,k)$ be a partial data routing tree of $T(s)$ by adding the first k source nodes into the tree with $1 \leq k \leq K$. When $k = 1$, the partial data routing tree $T(s,1)$ is a shortest path P_1 in G from s to d, where each edge $(v_i, v_{i+1}) \in P_1$ with $v_1 = s$, $v_{|P_1|+1} = d$, and v_{i+1} is the parent of v_i in the tree. The accumulative data stream on edge (v_i, v_{i+1}) in P_1 is $g(v_i)$, where $g(v)$ is an aggregate function at node v. Therefore, the communication cost on edge (v_i, v_{i+1}) is $b_{comm}(v_i, v_{i+1}) \cdot g(v_i)$, where $b_{comm}(v_i, v_{i+1})$ is the communication cost of one unit data transfer on edge (v_i, v_{i+1}). The number of VNF instances of each service function f_j will be deployed to some of the nodes along path P_1, i.e., the placement order of these VNF instances in P_1 from s to d is $f_1, f_2, \ldots, f_L$. We will deal with the VNF instance placement later, by incorporating the workload balancing among the cloudlets in P_1. It can be seen that the claim holds for this initial tree construction.

We assume that $T(s, k-1)$ has been constructed, and the first $(k-1)$ sources are added to the tree. We now construct $T(s,k)$ by adding a new source node s' to the tree.

Let $P_k(u, s')$ be a shortest path in $G \setminus T(s, k-1)$ between a tree node u and a source node s' that has not been contained in $T(s, k-1)$ yet. Let $P_k^T(u,d)$ be the unique tree path in $T(s, k-1)$ between node u and destination d. The VNF instances of functions of the subchain $\langle f_j, f_{j+1}, \ldots, f_L \rangle$ of the SFC are deployed in $P_k^T(u,d)$, i.e., no VNF instances of function $f_{j'}$ with $1 \leq j' \leq j-1$ are deployed in any node in $P_k^T(u,d)$. For the VNF instances of $f_{j'}$ placed at the node v in path $P_k^T(u,d)$ with $j \leq j' \leq L$, the accumulate data dream $g(v)$ at node v then is recomputed, by incorporating the data stream $\rho_{s'}$ from source s' through branch $P_k(u, s')$. Therefore the number of VNF instances of $f_{j'}$ at node v in path $P_k^T(u,d)$ is $\lceil \frac{g(v)}{\mu_{f_{j'}}} \rceil$.

For the data stream along a routing path from s' to d that consists of two segments $P_k^T(u,d)$ and $P_k(s', u)$, although the data stream along the tree path $P_k^T(u,d)$ has been processed by the VNF instances in the subchain $\langle f_j, \ldots, f_L \rangle$ in $P_k^T(u,d)$, the data stream along path $P_k(s', u)$ from node s' to node u has not been processed by the VNF instances of each service function in the subchain $\langle f_1, f_2, \ldots, f_{j-1} \rangle$ of the SFC. We thus need to place the number of VNF instances for each of them in the nodes in $P_k(s', u)$ in their specified order, where the rate of data stream on each edge in $P_k(s', u)$ is $\rho_{s'}$. In the following, we show how to place the VNF instances of each function in $\langle f_1, f_2, \ldots, f_{j-1} \rangle$ to cloudlets (nodes) in $P_k(s', u)$ to process the data stream while balancing the workload among the cloudlets, by developing a dynamic program solution for this workload-aware VNF instance placement problem.

3.5.2 Workload-aware VNF instance placement in a routing path

We here consider the workload-aware VNF instance placement on the nodes in a routing path. We assume that there is a data stream from source s' with data rate $\rho_{s'}$, the number of VNF instances of each function $f_{j'}$ in the SFC needs to be placed to a node in path $P_k(u, s')$ in order, where $1 \leq j' \leq j-1$. We aim to place the VNF instances to minimize the sum of the workload balancing costs among the nodes by formulating the following workload-aware VNF instance placement problem.

Let $u_0, u_1, u_2, \ldots, u_p$ be the node sequence in path $P_k(s', u)$ starting from u with $u_0 = u$, $u_p = s'$ and $p \geq 1$, where the computing resource consumption and computing capacity of each cloudlet node $u_i \in V$ are W_{u_i} and C_{u_i}, respectively. *The workload utility* of node u_i is $\gamma_{u_i} = \frac{W_{u_i}}{C_{u_i}}$ by Eq. (3.3).

There is a subchain $\langle f_{j-1}, f_{j-2}, \ldots, f_1 \rangle$ of the SFC for the data stream from source s' with data rate $\rho_{s'}$. The number of VNF instances of $f_{j'}$ to be placed along path $P_k(u, s')$ is $\lceil \frac{\rho_{s'}}{\mu_{f_{j'}}} \rceil$, where $\mu_{f_{j'}}$ is the data processing capacity of an VNF instance of function $f_{j'}$ in any cloudlet and $1 \leq j' \leq j-1$. The amount of computing resource consumed by the VNF instances of $f_{j'}$ is

$$w_{f_{j'}} = \lceil \frac{\rho_{s'}}{\mu_{f_{j'}}} \rceil \cdot c_{f_{j'}}. \tag{3.6}$$

Let $y_1, y_2, \ldots, y_q$ represent the function sequence $\langle f_{j-1}, f_{j-2}, \ldots, f_1 \rangle$ with $y_1 = f_{j-1}$ and $y_q = f_1$. Each $y_{j'}$ has a workload $w_{y_{j'}}$ by Eq. (3.6) with $1 \leq j' \leq q$. *The workload-aware VNF instance placement problem* is to place the VNF instances of functions in the subchain $\langle y_1, y_2, \ldots, y_q \rangle$ to the nodes in path $u_1, u_2, \ldots, u_p$ such that the sum of the workload balancing cost is minimized, assuming that there is sufficient computing resource at each cloudlet u_i to accommodate all VNF instances of function $y_{j'}$ in the subchain with $1 \leq i \leq p$ and $1 \leq j' \leq q$.

We devise a dynamic programming algorithm for the workload-aware VNF instance placement problem as follows.

Let $U_i = u_1, u_2, \ldots, u_i$ be the subsequence of $u_1, \ldots, u_p$ and $Y_{j'} = y_1, y_2, \ldots, y_{j'}$ the subsequence of $y_1, \ldots, y_q$ with $1 \leq i \leq p$ and $1 \leq j' \leq q$.

Denote by $B(i, j')$ the optimal cost of workload balancing by placing the VNF

Algorithm 5 Algorithm for the workload-aware VNF instance placement problem.

Require: A node sequence $u_1, u_2, \ldots, u_p$, a function sequence $y_1, y_2, \ldots y_q$, and the date rate $\rho_{s'}$ of a source s', assuming that the current workload W_{u_i} and capacity C_{u_i} of each cloudlet node u_i is given.

Ensure: the number of VNF instances of each function y_j is placed to a node in the node sequence such that the function order does not change and the sum of the workload balancing cost among the nodes is minimized.

```
1:  for $i \leftarrow 1$ to $p$ do
2:      for $j' \leftarrow 1$ to $q$ do
3:          if $i = 1$ then
4:              $B(i,j') \leftarrow \beta_{load} \cdot (2^{\gamma_{u_1} + \frac{\sum_{l=1}^{j'} w y_l}{C_{u_1}}} - 1)$;
5:          else
6:              if $j' = 1$ then
7:                  $B(i,j') \leftarrow \beta_{load} \cdot (\sum_{i'=1}^{i} (2^{\gamma_{u_{i'}}} - 1) + \min\{2^{\gamma_{u_{i'}} + \frac{w y_1}{C_{u_{i'}}}} - 2^{\gamma_{u_{i'}}} | 1 \le i' \le i\})$;
8:              else
9:                  $B(i,j') \leftarrow \min\{B(i-1,j') + \beta_{load} \cdot (2^{\gamma_{u_i}} - 1),\ B(i-1,j'-1) + \beta_{load} \cdot (2^{\gamma_{u_i} + \frac{w y_{j'}}{C_{u_i}}} - 1)\}$;
10:             end if;
11:         end if;
12:     end for;
13: end for;
14: return the solution $B(p,q)$.
```

instances of service functions in $Y_{j'}$ to nodes in U_i. Then,

$$B(i,j') = \begin{cases} \beta_{load} \cdot (2^{\gamma_{u_1} + \frac{\sum_{l=1}^{j'} w y_l}{C_{u_1}}} - 1) & i = 1, \\ \beta_{load} \cdot (\sum_{i'=1}^{i} (2^{\gamma_{u_{i'}}} - 1) + \\ \quad \min\{2^{\gamma_{u_{i'}} + \frac{w y_1}{C_{u_{i'}}}} - 2^{\gamma_{u_{i'}}} | 1 \le i' \le i\}) & j' = 1, \\ \min\{B(i-1,j') + \beta_{load} \cdot (2^{\gamma_{u_i}} - 1), \\ \quad B(i-1,j'-1) + \beta_{load} \cdot (2^{\gamma_{u_i} + \frac{w y_{j'}}{C_{u_i}}} - 1)\} & i > 1 \ \&\ j' > 1, \end{cases}$$

where γ_{u_i} is the workload utility of node u_i, $B(1,j') = \beta_{load} \cdot (2^{\gamma_{u_1} + \frac{\sum_{l=1}^{j'} w y_l}{C_{u_1}}} - 1)$ implies that there is only one cloudlet u_1 and the VNF instances of the chain will be hosted by the cloudlet, while $B(i,1) = \beta_{load} \cdot (\sum_{i'=1}^{i} (2^{\gamma_{u_{i'}}} - 1) + \min\{2^{\gamma_{u_{i'}} + \frac{w y_1}{C_{u_{i'}}}} - 2^{\gamma_{u_{i'}}} | 1 \le i' \le i\})$ indicates that the VNF instances of y_1 is added to such a cloudlet that results in the minimum increase of the workload balancing cost among the cloudlets. $B(i,j') = B(i-1,j') + \beta_{load} \cdot (2^{\gamma_{u_i}} - 1)$ implies that the VNF instances of the subchain

$Y_{j'}$ are placed to the first $(i-1)$ nodes in the node sequence U_i. $B(i,j') = B(i-1,j'-1) + \beta_{load} \cdot (2^{\gamma_{u_i} + \frac{w_{y_{j'}}}{C_{u_i}}} - 1)$ implies that the VNF instances of the subchain $Y_{j'-1}$ are placed to the nodes in the node sequence U_{i-1}, and the VNF instances of $y_{j'}$ are placed on node u_i.

The value of the solution to the workload-aware VNF instance placement problem thus is $B(p,q)$. The time complexity of the proposed algorithm is $\mathcal{O}(p \cdot q) = \mathcal{O}(|V| \cdot L)$ as $p \leq |V|$ and $q \leq L$, where L is the length of the SFC. The detailed algorithm is given in `Algorithm` 5.

It can be seen that the data stream from source s' to the destination node d passes through the demanded VNF instances of each function f_j in the SFC with $1 \leq j \leq L$, while the data streams of the first $(k-1)$ sources in tree $T(s,k)$ pass through their demanded numbers of VNF instances of each service function in the SFC, following the proposed tree construction.

The cost gain $\Delta(T(s,k-1),u,s')$, by connecting source s' to tree $T(s,k-1)$ through the tree node u, is the sum of the increased computing cost, communication cost and workload balancing cost by adding source s' to tree $T(s,k-1)$.

The detailed algorithm for the construction of a potential partial routing tree, by connecting source s' to tree $T(s,k-1)$ through the tree node u, is given in `Algorithm` 6.

3.5.3 Algorithm for cost minimization problem for a single multi-source IoT application

In the following, we propose an efficient algorithm for the cost minimization problem for a single multi-source IoT application, by making use of `Algorithm` 6 as its subroutine. Specifically, we first construct $|S|$ data routing trees, where S is the set of sources of the IoT application. Each data routing tree $T(s)$ is constructed by setting the destination node d as the tree root and source $s \in S$ as the first node added to the tree. Then each of the rest source nodes in $S \setminus \{s\}$ is added to $T(s)$ iteratively. In each iteration, the source node with the minimum cost increment compared with the previous cost of $T(s)$ will be chosen to add to the tree until all source nodes are added, by invoking `Algorithm` 6. Having constructed $|S|$ data routing trees, the data routing tree $T(s_0)$ with the minimum operational cost is chosen as the solution to the problem. The detailed algorithm is presented in `Algorithm` 7.

3.5.4 Analysis of the proposed algorithm

The rest is to address some important properties of the built data routing tree and the time complexity of the proposed algorithm. Recall that $f_1, f_2, \ldots f_L$ are the service functions in the SFC of the IoT application.

Algorithm 6 A potential routing tree construction by connecting source s' to tree $T(s, k-1)$ through the tree node u.

```
Require: An MEC network $G = (V, E)$ and a multi-source IoT application with an $SFC = \langle f_1, f_2, \ldots, f_L \rangle$, a destination $d \in V$, $K$ sources $s_1, s_2 \ldots, s_K$, and each source $s_i$ has a data stream rate $\rho_{s_i}$ with $1 \leq i \leq K$.
Ensure: A potential partial routing tree rooted at $d$ and spanning $k$ source nodes with the first joining in source $s$, and assuming that the first $(k-1)$ sources are added already and the partial routing tree is $T(s, k-1)$, a routing tree is constructed with the minimum cost where source $s'$ is joining to the tree through a path $P_k(s', u)$ and $u$ is the tree node.
1: if $k = 1$ then
2:     Find a shortest path $P_1$ in $G$ from $s'$ to $d$;
3:     Place the demanded number of VNF instances of $f_j$ to the nodes along path $P_1$ with $1 \leq j \leq L$ by invoking Algorithm 5;
4: else
5:     Update the number of VNF instances of each function $f_j$ at each node $v$ in the tree path $P_k^T(u, d)$ from $u$ to $d$, considering to add an extra $\rho_{s'}$ data stream to each edge along path $P_k^T(u, d)$;
6:     /* assuming $f_{j-1}$ is the first service function in the subchain from $f_1$ to $f_L$ whose VNF instances do not appear in a node in $P_k^T(u, d)$; */
7:     Place the demanded number of VNF instances of each function $f_{j'}$ with $1 \leq j' \leq j-1$ to the nodes of a shortest path $P_k(s', u)$ in $G \setminus T(s, k-1)$ from $s'$ to $u'$ with the data rate $\rho_{s'}$ by invoking Algorithm 5, where $(u', u)$ is the tree edge with endpoints $u'$ and $u$, and $u'$ is a child of $u$ in the tree, by balancing the workload among the nodes in the path;
8:     $\Delta(T(s, k-1), u, s') \leftarrow c(T(s, k-1) \cup P_k(s', u)) - c(T(s, k-1))$; /* $T(s, k-1) \cup P_k(s', u)$ is the partial routing tree by connecting source $s'$ to tree $T(s, k-1)$ through the tree node $u$ */
9: end if;
10: return the constructed partial routing tree $T(s, k-1) \cup P_k(s', u)$ and the increased cost $\Delta(T(s, k-1), u, s')$.
```

Algorithm 7 Heuristic algorithm for the cost minimization problem for a single multi-source IoT application in MEC.

Require: An MEC network $G = (V, E)$ and an IoT application with an $SFC = \langle f_1, f_2, \ldots, f_L \rangle$ and K sources $s_1, s_2 \ldots, s_K$ and a destination $d \in V$, each source s_i has a data rate ρ_{s_i} with $1 \leq i \leq K$.

Ensure: A data routing tree rooted at d and spanning all source nodes, and different numbers of VNF instances of the SFC are placed in the nodes of the tree such that the total operational cost outing of the IoT application is minimized.

```
1:  $cost \leftarrow \infty$; /* the operational cost of the solution */
2:  $T \leftarrow \emptyset$; /* the data routing tree for the IoT application */
3:  **for** $s \in S$ **do**
4:      /* The construction of a routing tree $T(s)$ with the minimum cost*/
5:      Find a shortest path $P_1$ in $G$ from $s$ to $d$;
6:      Construct the partial routing tree $T(s, 1)$ through placing the VNF instances
        of service function in the SFC along the nodes in $P_1$ with the data rate $\rho_s$, by
        calling Algorithm 5;
7:      $k \leftarrow 1$;
8:      $S' \leftarrow S \setminus \{s\}$;
9:      **while** $S' \neq \emptyset$ **do**
10:         $k \leftarrow k + 1$
11:         **for** each $s' \in S'$ **do**
12:             **for** each $u \in T(s)$ **do**
13:                 Construct the partial routing tree $T(s, k-1) \cup P_k(s', u)$ by connect-
        ing source $s'$ to tree $T(s, k-1)$ through the tree node $u$ along the path $P_k(s', u)$,
        by calling Algorithm 6;
14:                 Compute the total cost gain $\Delta(T(s, k-1), u, s')$ if path $P_k(s', u)$ is
        added to $T(s, k-1)$;
15:             **end for**;
16:         **end for**;
17:         $u_0, s'_0 \leftarrow$ min $\arg_{u \in T(s), s' \in S'} \{\Delta(T(s, k-1), u, s')\}$;
18:         $T(s, k) \leftarrow T(s, k-1) \cup P_k(s'_0, u_0)$; /* adding a source $s'_0$ with the minimum
        cost gain to the data routing tree */
19:         $S' \leftarrow S' \setminus \{s'_0\}$;
20:     **end while**
21:     $T(s) \leftarrow T(s, K)$;
22:     **if** $c(T(s)) < cost$ **then**
23:         $cost \leftarrow c(T(s))$;
24:         $s_0 \leftarrow s$;
25:     **end if**;
26: **end for**;
27: **return** the solution $T(s_0)$ with the total operational cost $c(T(s_0))$.
```

Lemma 6. Let $e_1, e_2, \ldots, e_q$ be the edges on the routing path in the data routing tree T from a source node s to the destination d, and let $v_1, v_2, \ldots, v_{q+1}$ are the corresponding nodes in the path with $v_1 = s$ and $v_{q+1} = d$. We have (i) The VNF instances of each function $f_j \in SFC$ are installed in the nodes in its specified order with $1 \leq j \leq L$, i.e., if the VNF instances of f_j are deployed in node v_i, then the VNF instances of $f_{j'}$ cannot be deployed to any node $v_{i'}$ with $j' > j$ and $i' < i$. We term this as the VNF deployment order by the SFC; and (ii) The VNF instances of any service function $f \in SFC$ cannot appear in multiple nodes in any routing path in the tree from a source to the destination, i.e., it is prohibited that VNF instances of function f are deployed into two different nodes v_i and v_j in the routing path with $i \neq j$. Otherwise, the data stream data will pass through the VNF instances of f twice from its source to the destination.

Proof. Property (i) holds because the data routing tree constructed by `Algorithm` 7 follows the function dependency of the SFC.

By Property (i) of tree T, if some VNF instances of a function f_j are instantiated at node v_i, then any of its other VNF instances cannot be deployed to other nodes in the subtree rooted at v_i; otherwise, Property (ii) is violated, i.e., the data stream from a descendent leaf node of v_i will pass through the VNF instances of the service function twice at two different cloudlets (at that node and at v_i), and this is prohibited by the SFC requirement of the IoT application. □

Lemma 7. Let $T(s, k)$ be a constructed partial data routing tree that contains the first $(k-1)$ source nodes by `Algorithm` 6, where source s is the first one added to the tree, for all k with $1 \leq k \leq K$. Then, the data stream along the tree path from each of the sources to the destination will pass through the demanded number of VNF instances of each service function in the SFC, and the solution delivered is feasible.

Proof. The claim can be shown by induction. Following `Algorithm` 6, when $k = 1$, the tree is a shortest path from s to d and the number of VNF instances of each service function in the SFC are placed to the nodes in the path. Then the claim clearly holds. We assume that the first $(k-1)$ sources have been added to the data routing tree and the claim holds for the data stream from each of the sources. Now, we show the claim still holds when adding the kth source into the tree. Following the algorithm, the number of VNF instances of each service function in the SFC is increased or instantiated accordingly to meet the data rate of the data stream from the kth source. The claim still holds. □

Theorem 3.3. *Given an MEC network $G = (V, E)$ and an IoT application that consists of K sources $s_1, \ldots, s_K$ and a destination $d \in V$, where each source gateway $s_i \in V$ has a data rate ρ_{s_i} with $1 \leq i \leq K$, and there is an SFC requirement $\langle f_1, \ldots, f_L \rangle$ for the IoT application and a defined data aggregation function $g(\cdot)$, the cost minimization problem for a single multi-source IoT application in G is to find a data routing tree T in G for implementing the IoT application such that the total operational cost is minimized. There is an efficient algorithm,* `Algorithm` *7, which delivers a feasible solution for the problem. The algorithm takes $\mathcal{O}(|V|^2 \cdot K^3 \cdot L)$ time.*

Proof. We first show that the solution delivered is feasible by Lemma 6 and Lemma 7. That is, the data stream from each source $s \in S$ to the destination d must pass through the demanded number of VNF instances of each function in the SFC prior to the destination. Let a data routing tree $T(s)$ rooted at d be chosen as the problem's solution, and the construction of $T(s)$ proceeds iteratively. Within each iteration, a source node joins in the tree. Let $T(s,k)$ be the partial data routing tree of $T(s)$ that contains the first k source nodes. Clearly, for tree $T(s,1)$, the data stream ρ_s is routed along a shortest path (i.e., $T(s,1)$) in G from s to d, and the demanded number of VNF instances of each service function in the SFC is instantiated in the nodes along the path. Assume that all data streams from the sources in the partial data routing tree $T(s,k-1)$ meet the condition: their demanded numbers of VNF instances of each service function in the SFC are placed in nodes of $T(s,k-1)$ already. When we consider the kth source s' joining in $T(s,k-1)$ to form tree $T(s,k)$, assume that node u is the tree node at which a branch from s' to u is attached to $T(s,k-1)$. It can be seen that the data stream with data rate $\rho_{s'}$ along the path $P_k(s',u) \cup P_k^T(u,d)$ from s' to d must pass through the demanded VNF instances of each service function in the SFC, following the construction of $T(s,k)$ by `Algorithm` 6. The solution delivered by `Algorithm` 7 thus is feasible.

We then analyze the time complexity of the proposed algorithm, `Algorithm` 7. There are K potential data routing trees to be constructed. To construct a data routing tree that contains a source s initially, another source node with the minimum cost increment will be chosen to add to the tree, and this procedure continues until all source nodes are added to the tree. To calculate the cost increment of adding a source s' to the partial data routing tree, we first find the shortest paths between source s' and each node in the partial data routing tree and that takes $\mathcal{O}(|V|^2)$ time, and the VNF instance placements along the tree paths by invoking the dynamic programming algorithm - `Algorithm` 5 that takes $\mathcal{O}(|V|^2 \cdot L)$ time. Thus, the running time of `Algorithm` 7 is $\mathcal{O}(|V|^2 \cdot K^3 \cdot L)$. □

Remarks: The proposed algorithm assumed that a single cloudlet can host all VNF instances of any IoT application. This assumption is reasonable in practice as the resource demands by a single IoT application should be far less than the amount of resource a cloudlet can provide. However, it is not uncommon that the resources in MEC are very limited, the proposed algorithm can be easily modified for this resource restriction case. That is, if the residual resource of a cloudlet is less than the resource demands of an IoT application, that cloudlet will not involve resource allocation for the IoT application by hosting any of the VNF instances of the IoT application (i.e., it cannot accommodate any VNF instance in the data routing tree for the IoT application). The cloudlet, however, is still part of the data routing tree for the IoT application to maintain the network connectivity, i.e., there may be routing paths passing through the cloudlet.

3.6 Algorithm for cost minimization problem for a set of multi-source IoT applications

In this section, we consider the service provisioning for a set of multi-source IoT applications in an MEC network, where each IoT application can share the VNF instances of data streams from its different sources, but it is prohibited to share VNF instances among different IoT applications due to their security and privacy concerns. In other words, we aim to perform VNF-enabled network slicing in the MEC to provide services for different IoT applications such that the total operational cost is minimized.

We implement the multi-source IoT applications in $\mathcal{I}$ one by one iteratively. Each time, one application with the minimum operational cost is chosen from the remaining applications by invoking `Algorithm` 7. This procedure continues until all IoT applications in $\mathcal{I}$ are implemented. The detailed algorithm is given as `Algorithm` 8.

Algorithm 8 An algorithm for the service provisioning of a set of multi-source IoT applications.

```
Require: An MEC network $G = (V, E)$ and a set $\mathcal{I}$ of IoT applications with an
    $SFC = \langle f_{i,1}, f_{i,2}, \ldots, f_{i,L_i} \rangle$ of each application $r_i \in \mathcal{I}$ and $K_i$ sources $s_{i,1}, s_{i,2}, \ldots, s_{i,K_i}$
    and a destination $d_i \in V$, each source $s_{i,j}$ has a data rate $\rho_{i,j}$ with $1 \leq j \leq K_i$.
Ensure: A solution of implementing all IoT applications such that the total cost is
    minimized, where the solution of each IoT application $r_i$ is a data routing tree
    with $K_i$ sources.
1: $total\_cost \leftarrow 0$; /* the value of the solution */
2: $\mathcal{T} \leftarrow \emptyset$ /* the set of data routing trees for the IoT applications in $\mathcal{I}$ */
3: $\mathcal{U} \leftarrow \mathcal{I}$; /* the set of IoT applications that have not been implemented in the
   MEC yet */
4: while $\mathcal{U} \neq \emptyset$ do
5:     for each IoT application $r_i \in \mathcal{U}$ do
6:         Find a data routing tree $T_{r_i}$ for it, and compute the total operational cost
   $c(T_{r_i})$ of tree $T_{r_i}$, by calling Algorithm 7;
7:     end for;
8:     Let $r_{i_0} = \text{argmin}_{r_i \in \mathcal{U}}\{c(T_{r_i})\}$ /* $T_{r_{i_0}}$ is a data routing tree with the minimum
   cost among the applications in $\mathcal{U}$ */ ;
9:     $\mathcal{T} \leftarrow \mathcal{T} \cup \{T_{r_{i_0}}\}$;
10:    Update all computing resource in cloudlets for implementing $r_{i_0}$, and calcu-
   late the residual computing resource of cloudlets in the MEC;
11:    $\mathcal{U} \leftarrow \mathcal{U} \setminus \{r_{i_0}\}$;
12:    $total\_cost \leftarrow total\_cost + c(T_{r_{i_0}})$;
13: end while; return The solution $\mathcal{T}$.
```

Theorem 3.4. *Given an MEC network $G = (V, E)$, each cloudlet $v \in V$ has computing capacity C_v, a set $\mathcal{I}$ of multi-source IoT applications with each application $r_i \in \mathcal{I}$ having*

K_i sources and an SFC_i with length of L_i, and each source $s_{i,j}$ of r_i has a data rate $\rho_{i,j}$ with $1 \leq j \leq K_i$. There is an efficient algorithm for the cost minimization problem for a set of multi-source IoT applications, `Algorithm` *8, which takes $\mathcal{O}(|V|^2 \cdot K_{max}^3 \cdot L_{max} \cdot |\mathcal{I}|^2)$ time, where $K_{max} = \max\{K_i \mid r_i \in \mathcal{I}\}$ and $L_{max} = \max\{L_i \mid r_i \in \mathcal{I}\}$ are the maximum number of sources and the maximum length of all SFCs among IoT applications in $\mathcal{I}$, respectively.*

Proof. Because `Algorithm` 8 invokes `Algorithm` 7 iteratively, the feasibility of the solution delivered by `Algorithm` 8 for the cost minimization problem for a set of multi-source IoT applications can be shown by the feasibility of the solution delivered by `Algorithm` 7 for the cost minimization problem, which has been proved in Theorem 3.3, omitted.

The time complexity of `Algorithm` 8 is $\mathcal{O}(|V|^2 \cdot K_{max}^3 \cdot L_{max} \cdot |\mathcal{I}|^2)$, as an application with the minimum operational cost from the remaining applications is chosen at each iteration by invoking `Algorithm` 7, while the time complexity of `Algorithm` 7 is $\mathcal{O}(|V|^2 \cdot K_{max}^3 \cdot L_{max})$ by Theorem 3.3, and the number of the invoking of `Algorithm` 7 is $\mathcal{O}(\mathcal{I}^2)$. □

3.7 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms for the service provisioning for IoT applications in an MEC network. We also investigate the impact of important parameters - the number L of service function in a service chain, the number K of sources, and the workload balancing coefficient β_{load}, on the performance of the proposed algorithms.

3.7.1 Environment settings

We consider an MEC network that consists of 100 APs, with each having a co-located cloudlet. We generate the topologies of MEC networks using GT-ITM [34]. The capacity of each cloudlet is set from $30,000$ to $60,000$ MHz [81]. We assume that there are 10 types of VNFs, and the computing resource consumption of each VNF instance ranges from 20 to 100 MHz [69]. We further assume that the processing rate of each VNF instance varies from 30 to 80 packet per millisecond, where each data packet is of size 64 KB [69]. The number of different types of SFCs is set as 20, and the length of each varies from 3 to 7 [70]. For each multi-source IoT application request, the number of sources is set from 4 to 8, and the data rate of each source ranges from 2 to 10 packet per millisecond [70]. The communication cost of each link is randomly drawn from \$0.05 to \$0.4 per data packet [69]. The computing resource consumption cost is set at \$0.1 per MHz [107]. The coefficient of the workload balancing cost (i.e., β_{load}) is set at $2,000$. The aggregation function of each multi-source IoT application is randomly chosen from the set $\{summation, averaging, maximum, minimum\}$. The value in each figure is the mean of the results out of 20 MEC network instances of the same size. The actual running time of each algorithm is based on a desktop with a 3.60 GHz Intel 8-Core i7-7700 CPU and 16 GB RAM. Unless otherwise specified, these parameters will be adopted in the default setting.

We evaluate the proposed algorithm `Algorithm` 7 (referred to as `Alg.7`) against two baseline algorithms: `Heu.1_S` and `Heu.2_S`, for the cost minimization problem for a single multi-source IoT application. In algorithm `Heu.1_S`, we first find the shortest path between each source and the destination node, and the VNF instances of each service function in the SFC will be instantiated along the routing path. In this heuristic, neither computing resource (VNF instances) nor bandwidth resource is shared among the data streams of different sources for each IoT application. On the contrary, algorithm `Heu.2_S` consolidates all the VNFs in the SFC of each request into a single cloudlet, and the VNFs are shared among the data streams from all sources. Especially, algorithm `Heu.2_S` first finds a shortest path P_1 from a randomly selected source s to the destination node of the request, and the cloudlet v with the least workload utility by Eq. (3.3) on the shortest path is chosen to accommodate the VNFs. Based on path P_1, the data routing tree T for the request then is constructed by including the shortest path from each remaining source $s' \in S \setminus \{s\}$ to the chosen cloudlet v, one by one, and the data streams are then aggregated at each non-leaf node of the data routing tree T.

We then evaluate the proposed algorithm `Algorithm` 8 (referred to as `Alg.8`) against algorithms `Heu.1_M` and `Heu.2_M`, for the cost minimization problem for a set of multi-source IoT applications. Algorithms `Heu.1_M` and `Heu.2_M` are built upon algorithms `Heu.1_S` and `Heu.2_S`, respectively. Especially, algorithm `Heu.1_M` processes IoT applications one by one by algorithm `Heu.1_S`, and picks them for consideration randomly. Algorithm `Heu.2_M` is similarly developed, omitted.

3.7.2 Performance evaluation of different algorithms

We first studied the performance of algorithm `Alg.7` against algorithms `Heu.1_S` and `Heu.2_S`, respectively, by varying the network size from 50 to 250. Fig. 3.3(a) and Fig. 3.3(b) plot the total cost and running time of different algorithms. It can be seen from Fig. 3.3(a) that when the network size is 250, the total cost achieved by `Alg.7` is 41.6% of that by algorithm `Heu.1_S`, and 75.6% of that by algorithm `Heu.2_S`. This is because `Alg.7` establishes an efficient data routing tree that jointly considers VNF instance sharing and communication path sharing among different data streams of each IoT application, and the workload balancing among cloudlets.

We then investigated the performance of `Alg.8` against algorithms `Heu.1_M` and `Heu.2_M`, by varying the network size from 50 to 250, with $1,000$ IoT applications. Fig. 3.4(a) and Fig. 3.4(b) depict the total cost and running time of different algorithms. It can be seen from Fig. 3.4(a) that when the network size is 250, the total cost achieved by algorithm `Alg.8` is 45.3% of that by algorithm `Heu.1_M`, and 80.3% of that by algorithm `Heu.2_M`. The rationale behind this is that with the increase on the network size, algorithm `Alg.8` outperforms algorithms `Heu.1_M` and `Heu.2_M` not only in resource sharing but also in workload balancing among cloudlets.

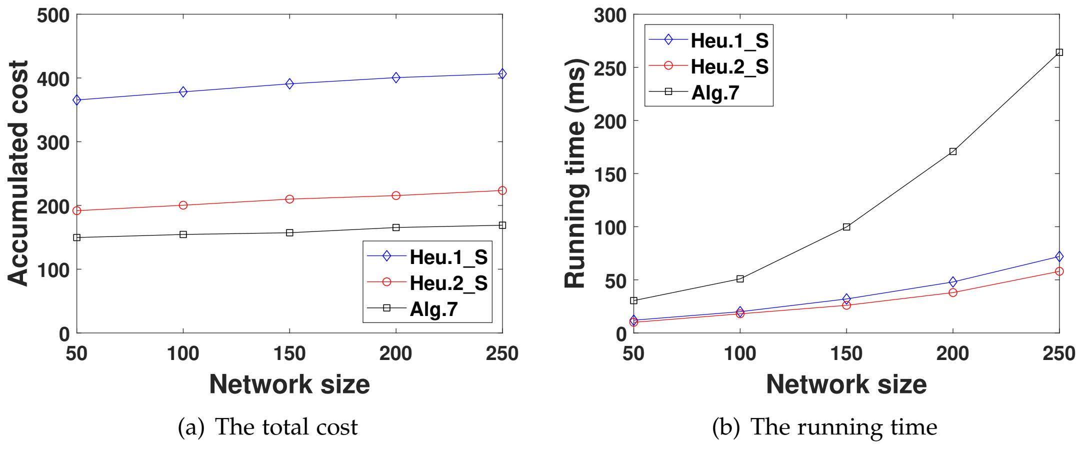


Figure 3.3: Performance of different algorithms for the cost minimization problem for a single multi-source IoT application, by varying the network size from 50 to 250.

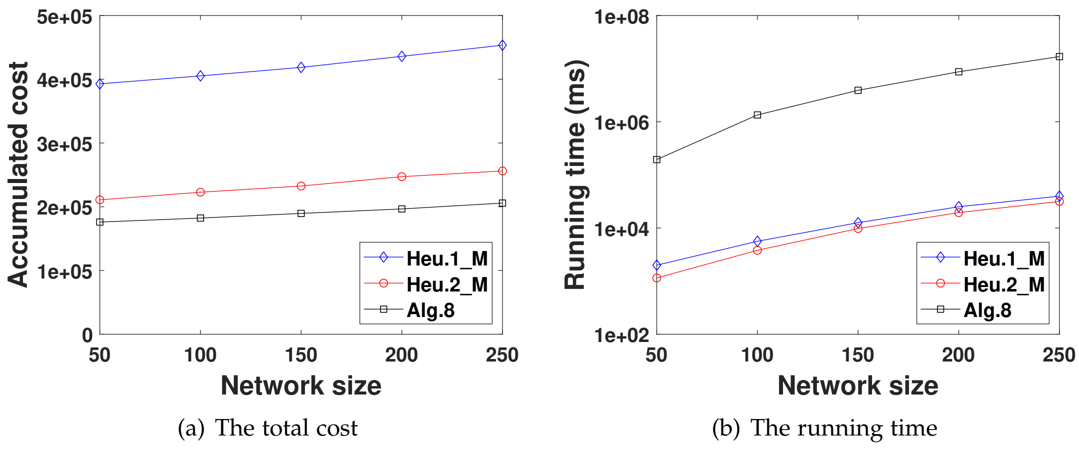
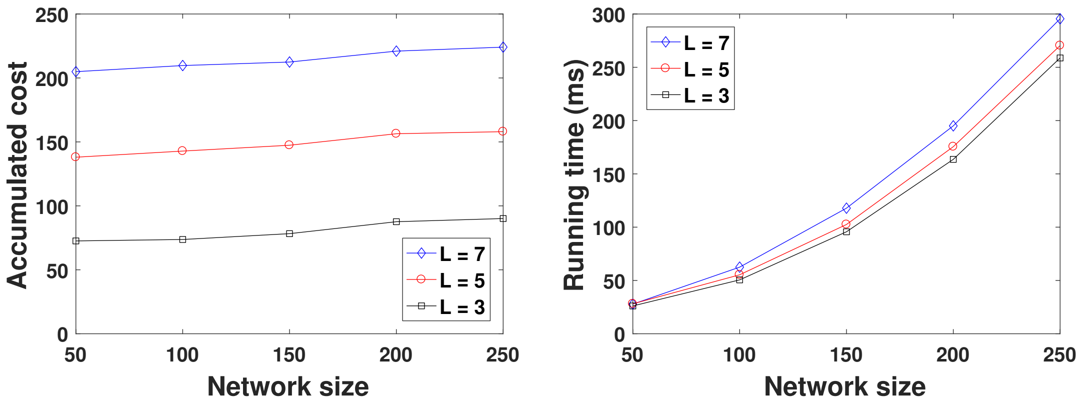


Figure 3.4: Performance of different algorithms for the cost minimization problem for a set of multi-source IoT applications, by varying the network size from 50 to 250.

3.7.3 Impact of important parameters on the performance of the proposed algorithms

We finally investigated the impact of important parameters on the performance of the proposed algorithms for the defined problems, including the length L of SFCs, the number K of sources, and the coefficient β_{load} of the workload balancing cost.

We first studied the impact of the length L of SFCs on the proposed algorithms. Fig. 3.5(a) and Fig. 3.5(b) demonstrates the total cost and running time of the solution delivered by algorithm `Alg.7` for different network sizes when $L = 3$, 5, and 7, respectively. Fig. 3.6(a) and Fig. 3.6(b) depict the total cost and running time of the solution delivered by algorithm `Alg.8` with 1,000 IoT applications for different network sizes when $L = 3$, 5 and 7, respectively. It can be seen from Fig.3.5(a) that with the network size of 250, the total cost of the solution delivered by algorithm `Alg.7` when $L = 3$ is 40.2% of itself when $L = 7$. This can be justified that with a small value of L, the IoT applications consumes less computing resource, thus, both the computing cost and the workload balancing cost decrease. The similar performance behaviors can be observed in Fig. 3.6(a), too.

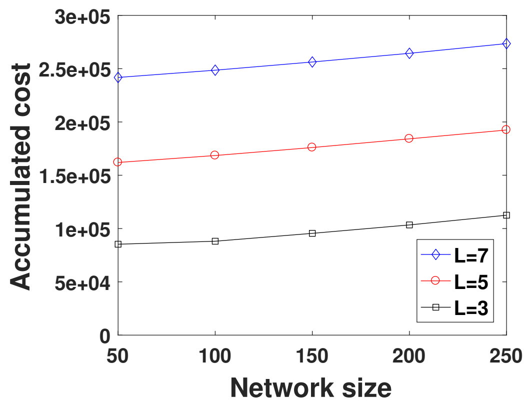


(a) The total cost (b) The running time

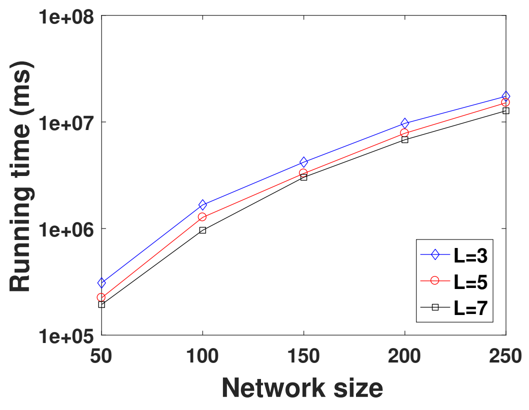
Figure 3.5: The impact of length L of the SFC on the performance of `Alg.7`.

We then investigated the impact of the number K of sources on the proposed algorithms. Fig. 3.7(a) and Fig. 3.7(b) plot the total cost and running time of the solution delivered by algorithm `Alg.7` for different network sizes when $K = 4$, 6, and 8 respectively. Fig. 3.8(a) and Fig. 3.8(b) show the total cost and running time of the solution delivered by algorithm `Alg.7` with 1,000 IoT applications for different network sizes when $K = 4$, 6, and 8. It can be seen from Fig.3.7(a) that with the network size of 250, the total cost of the solution delivered by algorithm `Alg.7` when $K = 4$ is 48.7% of itself when $K = 8$. This can be justified that with a small value of K, the IoT applications consumes not only less computing resource, but also less data rates. Thus, the communication cost, the computing cost, and the workload balancing cost decrease. The similar performance behaviors can be observed in Fig. 3.8(a), too.

We finally evaluated the impact of the coefficient β_{load} of the workload balancing

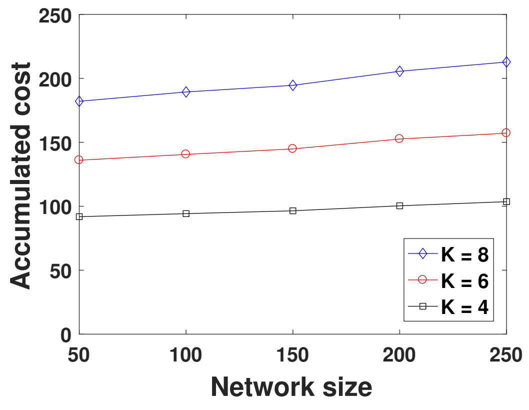


(a) The total cost

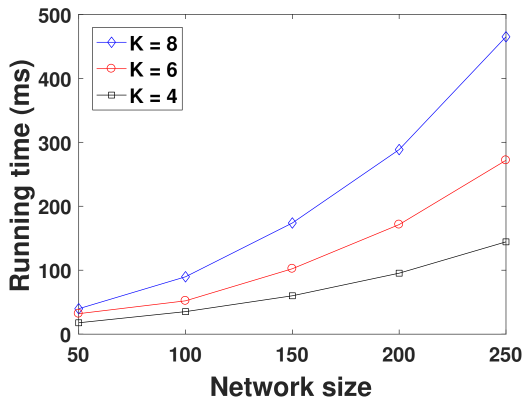


(b) The running time

Figure 3.6: The impact of length L of the SFC on the performance of `Alg.8`.

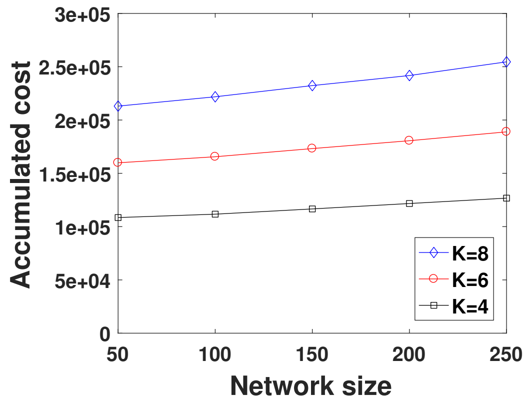


(a) The total cost

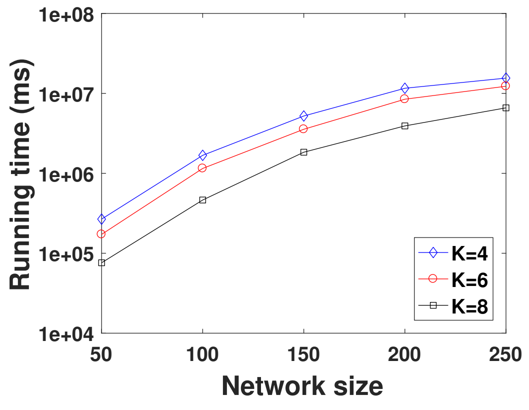


(b) The running time

Figure 3.7: The impact of the number K of sources on the performance of `Alg.7`.



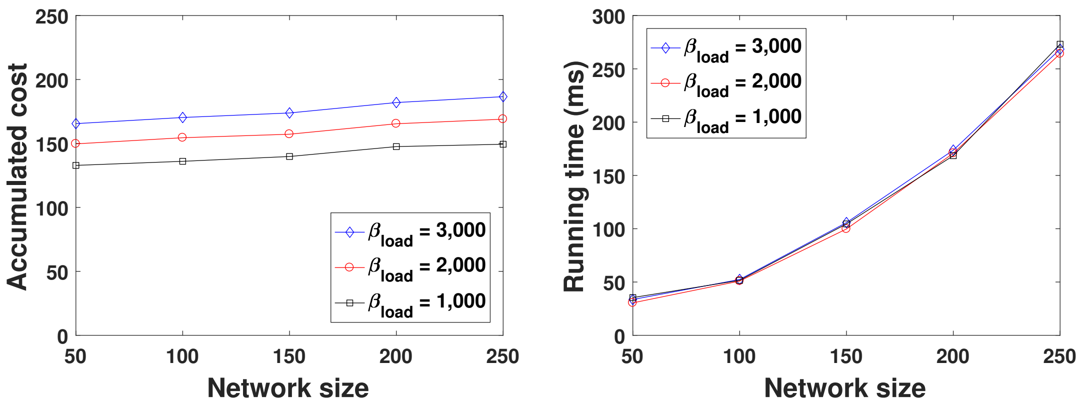
(a) The total cost



(b) The running time

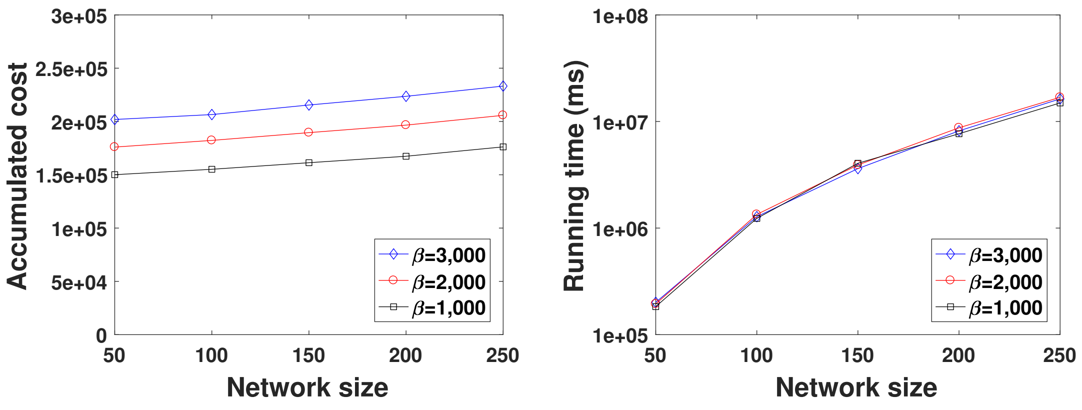
Figure 3.8: The impact of the number K of sources on the performance of `Alg.8`.

cost on the proposed algorithms. Fig. 3.9(a) and Fig. 3.9(b) show the total cost and running time of algorithm `Alg.7` for different network sizes when $\beta_{load} = 1,000$, 2,000, and 3,000, respectively. Fig. 3.10(a) and Fig. 3.10(b) depict the total cost and running time of algorithm `Alg.7` with $1,000$ IoT applications for different network sizes when $\beta_{load} = 1,000$, 2,000, and 3,000, respectively. From Fig. 3.9(a), the total cost of algorithm `Alg.7` when $\beta_{load} = 1,000$ is 80.4% of itself when $\beta_{load} = 3,000$. This is because more cost is caused to guarantee the workload balancing among the cloudlets in the MEC network, with the increase on the value of β_{load}. It can also be seen from Fig. 3.9(a) that the performance gap of algorithm `Alg.7` from $\beta_{load} = 2,000$ to $\beta_{load} = 3,000$ is 90.1% of itself from $\beta_{load} = 1,000$ to $\beta_{load} = 2,000$. The rationale behind this is that a lager β_{load} will lead to a more balanced workload among the cloudlets, thus, the total workload utility (W_v/C_v) of each cloudlet is reduced. The similar performance behaviors can be observed in Fig. 3.10(a), too.



(a) The total cost (b) The running time

Figure 3.9: The impact of β_{load} on the performance of `Alg.7`.



(a) The total cost (b) The running time

Figure 3.10: The impact of β_{load} on the performance of `Alg.8`.

3.8 Summary

In this chapter, we studied the service provisioning in an MEC network for multi-source IoT applications with SFC requirements. We formulated two novel cost minimization problems for multi-source IoT applications: the cost minimization problem for a single multi-source IoT application and the cost minimization problem for a set of multi-source IoT applications, respectively, and showed that both the problems are NP-hard. We then proposed a novel service provisioning framework for multi-source IoT applications, which includes uploading stream data from multiple sources to the MEC network, in-network data stream aggregation and routing, VNF instance placement and sharing, and workload balancing in the MEC network. We thirdly devised efficient algorithms for the defined problems based upon the built framework. We finally evaluated the performance of the proposed algorithms through experimental simulations. Experimental results demonstrate that the proposed algorithms are promising.

Chapter 4

Throughput Maximization of Delay-Aware DNN Inference

In this chapter, we first introduce a novel Deep Neural Network (DNN) inference throughput maximization problem with the aim to maximize the number of delay-aware DNN service requests admitted, by accelerating each DNN inference through jointly exploring DNN partitioning and multi-thread execution parallelism. We then devise an approximation algorithm and an online algorithm for the problem under the offline and online settings, respectively.

4.1 Introduction

Prized by the practitioners in both academia and industry, deep learning has ignited a booming of intelligent IoT devices, which pushes edge intelligence to the horizon [122]. However, due to portable size of most mobile devices, they are resource-constrained and have limited processing capabilities to support compute-intensive deep learning applications. Thus, traditional solutions to such applications are to resort to the powerful clouds for processing [72]. Since it results in a long response delay by offloading a Deep Neural Network (DNN) inference task from an IoT device to a remote cloud, this drives new opportunities to push DNN inferences to the edge of core networks by providing real-time DNN inference services [45]. Mobile Edge Computing (MEC) has become a promising paradigm to provision computing and storage resources in local cloudlets with well-trained DNN models for EDs, by offloading the whole or part of DNNs from EDs via Access Points (APs) to the MEC network for accelerating inference processing [111]. With the advance of intelligent IoT devices, it is urgent to utilize both computing and storage resources of cloudlets in an MEC network for DNN inference services in a real-time manner [116].

To alleviate the DNN inference delay, it is desirable that a DNN model can be partitioned into two parts: one is executed in its local IoT device and another is executed in a cloudlet of the MEC network [52; 53]. This task offloading method is referred to as the *DNN partitioning* method, which has been widely adopted for DNN model inferences as the amounts of output data in some intermediate layers of a DNN model are significantly smaller than that of its raw input data layer [36], thereby reducing the delay of data uploading from its IoT device to the MEC network. The potential of cloudlets on accelerating DNN inference can be further unleashed by exploring inference parallelism, i.e., leveraging multiple threads in a cloudlet to shorten the inference time further. A network provider can deploy multi-threading for DNN inference under existing frameworks (e.g. TensorFlow [1] and PyTorch [77]), or via a customized thread pool [66]. The delay-aware DNN inference service provisioning thus poses the following challenges.

For a DNN inference request with a trained DNN model and a given inference delay requirement, how to determine which cloudlet in an MEC network to accommodate the request? How to partition the DNN model between its local IoT device and its assigned cloudlet such that the inference delay meets its inference delay requirement? How many threads in its assigned cloudlet should be allocated for processing the offloaded DNN part while meeting the inference delay requirement of the request? In this chapter, we will address these challenges, and devise performance-guaranteed approximation and online algorithms for the DNN inference throughput maximization problem in an MEC network under both offline and online request arrival settings, respectively.

The novelty of the work in this chapter lies in jointly exploring DNN model partitioning and inference parallelism to accelerate the DNN inference process. The admission of a set of delay-aware DNN inference requests, rather than a single delay-aware DNN inference request, is considered for the first time in MEC. Also, to speed up the inference process in MEC, the multi-thread concept is adopted in DNN inferences, and performance-guaranteed approximation and online algorithms for the defined problem under both offline and online request arrival settings are proposed and analyzed.

The main contributions of this chapter are given as follows. We first study a DNN inference throughput maximization problem in an edge computing environment under an offline setting of request admissions, with the aim to maximize the number of requests admitted, subject to computing capacities on cloudlets in the MEC network. We approach the problem by jointly partitioning the DNN model, allocating the offloaded part of the DNN to a cloudlet, and exploring inference parallelism by utilizing multiple threads in the cloudlet to meet its inference delay requirement. We then show the NP-hardness and devise an approximation algorithm for this offline setting. We also deal with the problem under the online setting of request arrivals, where a sequence of delay-aware DNN inference requests arrives one by one without the knowledge of future arrivals, for which we develop an online algorithm with a provable competitive ratio. We finally conduct experimental simulations to evaluate the performance of the proposed algorithms. Experimental results demonstrate that the proposed algorithms are promising.

The rest of the chapter is organized as follows. Section 4.2 introduces the system model, notions and notations, and defines the problems formally. The NP-hardness proof of the defined problem is given here as well. Section 4.3 devises an approximation algorithm for the delay-aware DNN inference problem when all requests are given in advance. Section 4.4 proposes an online algorithm for the dynamic delay-aware DNN inference problem. Section 4.5 evaluates the proposed algorithms empirically, and a summary is given in Section 4.6.

4.2 Preliminaries

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

4.2.1 System model

Given a geographical area (e.g., a metropolitan area), a set N of Access Points (APs) (or base stations) is deployed in the area. Associated with each AP, there is a co-located cloudlet (edge server) n with computing capacity C_n, which is interconnected with the AP via an optical cable and the transmission delay between them is neglected [58; 57]. For the sake of convenience, denote by N the set of cloudlets too. Since the wireless transmission range of each IoT device is fixed, the number of cloudlets the device can reach is limited, too [94]. Denote by $\mathbb{N}(r_i) \subset N$ the set of cloudlets located in the data transmission range of the IoT device of request r_i, i.e., request r_i can be served by any cloudlet $n \in \mathbb{N}(r_i)$ if the cloudlet has sufficient computing resource for processing the offloaded part of the DNN model of request r_i. Fig. 4.1 is an illustrative example of an MEC network, in which the IoT device of a request can only reach a subset of cloudlets in the MEC network, i.e., any cloudlet that is within its transmission range.

In this chapter, we focus on DNN inference. Compared with DNN training, the DNN inference requires much less CPU and GPU resources. Therefore, instead of making use of costly hardware accelerators (e.g., GPUs) for DNN inference, CPUs are more favored to cope with real-time DNN inference services at the network edge with cost-efficiency [76], due to their high availability and efficient scalability [32]. For simplicity, we will focus on CPU-based cloudlets, and each CPU core corresponds to a *thread*, since applying hyper-threading technique (generating two threads per CPU core) may decrease the performance due to additional context switching [66]. We assume that each cloudlet $n \in N$ has a computing capacity C_n, which is measured by the number of threads (CPU cores) in it. Note that the results of this chapter can be easily extended to deal with cloudlets equipped with GPU resource as well, by incorporating the threads of GPU cores and leveraging the parallelism of GPUs on inference acceleration.

4.2.2 User requests and the DNN model

There is a set R of delay-aware DNN inference requests from different user IoT devices. Each request $r_i \in R$ has an inference delay requirement D_i [68], and its requested DNN can be modeled as a directed acyclic graph (DAG) $G_i = (V_i, E_i)$, where V_i is the set of inference layers and E_i is the set of layer dependencies. V_i contains $|V_i|$ inference layers: $v_{i,0}, v_{i,1}, v_{i,2}, \ldots, v_{i,|V_i-1|}$, where $v_{i,0}$ is a virtual input layer, and each of the rest inference layers can be further partitioned into sub-layers,

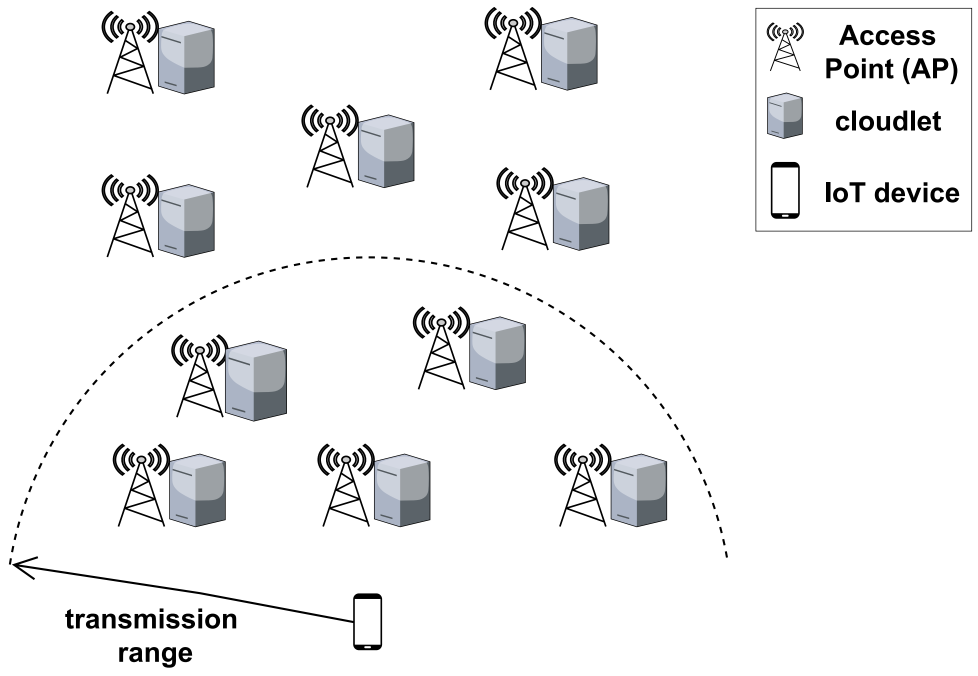


Figure 4.1: An example of a geographical area with 10 APs, each of APs is co-located with a cloudlet. There is an IoT device which can offload its task to any cloudlet within its transmission range if the cloudlet has sufficient computing resource to process its offloaded task.

which implies that the matrix composed by its input and output can be processed in parallel if multiple threads are allocated to the matrix multiplication computation. Each edge $(v_{i,j}, v_{i,l}) \in E_i$ represents the computation dependency of layer $v_{i,l}$ on layer $v_{i,j}$, i.e., layer $v_{i,j}$ needs to be computed first and its output is the input of layer $v_{i,l}$ with $v_{i,l} \in \mathcal{N}^+(v_{i,j})$, where $\mathcal{N}^+(v_{i,j}) = \{v_{i,l} \mid (v_{i,j}, v_{i,l}) \in E_i\}$ is the outgoing set from layer $v_{i,j}$ in G_i, Similarly, the incoming set $\mathcal{N}^-(v_{i,l})$ of $v_{i,l}$ with $\mathcal{N}^-(v_{i,l}) = \{v_{i,j} \mid (v_{i,j}, v_{i,l}) \in E_i\}$, can be defined as well. We assume that the DNN model of each request has been trained, and stored in both cloudlets and its IoT device. An example of a DNN is given in Fig. 4.2.

Denote by $f(v_{i,j})$ the required number of floating-point operations of layer $v_{i,j}$, which is the workload to execute layer $v_{i,j}$. Note that $f(v_{i,0}) = 0$, as $v_{i,0}$ is a virtual input layer.

To utilize the processing capabilities of the local IoT device and the assigned cloudlet to accelerate DNN inference process, the layer set V_i of request r_i can be partitioned into two disjoint subsets V_i^{loc} and V_i^{mec}, i.e., $V_i = V_i^{loc} \cup V_i^{mec}$ and $V_i^{loc} \cap V_i^{mec} = \emptyset$, where layer nodes in V_i^{loc} and V_i^{mec} will be executed at the local IoT device and a cloudlet, respectively. Let $V_i^{tran} \subset V_i^{loc}$ be the layer set that the output of each layer $v_{i,j} \in V_i^{tran}$ will be sent to the cloudlet for processing. Note that $v_{i,0}$ is a virtual input layer which is in the DNN part on its IoT device, i.e., $v_{i,0} \in V_i^{loc}$. In case where

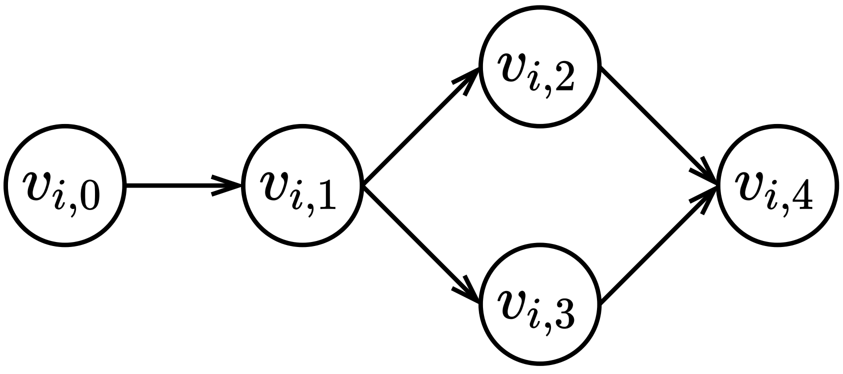


Figure 4.2: An example of a DNN that consists of 4 layers.

$V_i^{mec} = V_i \setminus \{v_{i,0}\}$, and $V_i^{loc} = V_i^{tran} = \{v_{i,0}\}$, the raw input of request r_i will be transmitted to the cloudlet for DNN processing. We assume that if layer $v_{i,j} \in V_i^{mec}$ is executed in a cloudlet, then any layer $v_{i,l} \in \mathcal{N}^+(v_{i,j})$ must be executed in the cloudlet with $(v_{i,j}, v_{i,l}) \in E_i$, as the DNN is partitioned into two parts only.

4.2.3 Inference delay model

Because the result of a DNN inference usually is small, the downloading time of the inference result is negligible [111]. The inference delay of a DNN inference request usually consists of (1) the processing delay on its local IoT device; (2) the data transmission delay from the local IoT device to its assigned cloudlet; and (3) the processing delay of the part of the DNN model in the cloudlet, which are defined as follows.

The processing delay on the local IoT device: Denote by $\eta_{loc}(r_i)$ the processing rate of the local IoT device of request r_i, which is measured in the number of floating-point operations per second. With $f(v_{i,j})$ the required number of floating-point operations of layer $v_{i,j}$, the processing delay $d_{i,j}^{loc}$ of layer $v_{i,j}$ on the local IoT device of request r_i then is

$$d_{i,j}^{loc} = \frac{f(v_{i,j})}{\eta_{loc}(r_i)}. \tag{4.1}$$

Since V_i^{loc} is the set of layers processed on the local IoT device, the total processing delay of the set of layers V_i^{loc} on the local IoT device of request r_i is

$$d_{loc}(V_i^{loc}) = \sum_{v_{i,j} \in V_i^{loc}} d_{i,j}^{loc}. \tag{4.2}$$

The transmission delay from the local IoT device to the assigned cloudlet: Denote by $O_{i,j}$ the output data size of layer $v_{i,j}$ in the DNN G_i of request r_i, where $O_{i,0}$ is the output of the virtual input layer $v_{i,0}$, i.e., the data size of the raw input of the DNN G_i, and the volume of $O_{i,j}$ can be obtained from the requested DNN model [111]. The data transmission rate $\lambda_{i,n}$ from the IoT device of user request r_i to cloudlet $n \in \mathbb{N}(r_i)$ is computed through the Shannon-Hartley theorem [31], i.e.,

$$\lambda_{i,n} = B_n \log_2(1 + \frac{P_i}{dist_{i,n}^{\beta} \cdot \sigma^2}), \tag{4.3}$$

where B_n is the bandwidth of the AP co-located with cloudlet n, P_i is the transmission power of the IoT device of r_i, $dist_{i,n}$ is the distance between the IoT device of r_i and the AP co-located with cloudlet n [18], σ^2 is the noise power, and β is a path loss factor with $\beta = 2$ or 4 for short or long distance, respectively [18].

The transmission delay of transmitting the output of layer $v_{i,j}$ of request r_i to cloudlet $n \in \mathbb{N}(r_i)$ is

$$d_{i,j}^{tran}(n) = \frac{O_{i,j}}{\lambda_{i,n}}. \tag{4.4}$$

As V_i^{tran} is the set of layers whose outputs will be uploaded and transferred to cloudlet $n \in \mathbb{N}(r_i)$. Then the transmission delay of the output data by V_i^{tran} of request r_i is

$$d_{tran}(V_i^{tran}, n) = \sum_{v_{i,j} \in V_i^{tran}} d_{i,j}^{tran}(n). \tag{4.5}$$

The processing delay on a cloudlet: The inference processing delay of the offloaded layers in set V_i^{mec} in a cloudlet is assumed to be proportional to the number of allocated threads (e.g., 1 thread vs 2 threads) to this offloaded part. Furthermore, we assume that at most K threads are allocated to each offloaded DNN part in a cloudlet. And all threads in any cloudlet n have the same processing rate $\eta_{mec}(n)$, and is measured by the number of floating-point operations per second. Let $k_{i,n}$ be the number of allocated threads in cloudlet $n \in \mathbb{N}(r_i)$ to process the offloaded part of request r_i with $1 \leq k_{i,n} \leq K$. Then, the accumulative processing rate of cloudlet n for request r_i is $k_{i,n} \cdot \eta_{mec}(n)$.

Recall that $f(v_{i,j})$ is the required number of floating-point operations of layer $v_{i,j}$, the processing delay of layer $v_{i,j}$ assigned to cloudlet n with $k_{i,n}$ allocated threads is

$$d_{i,j}^{mec}(n, k_{i,n}) = \frac{f(v_{i,j})}{k_{i,n} \cdot \eta_{mec}(n)}. \tag{4.6}$$

The processing delay of the offloaded layer set V_i^{mec} of request r_i in cloudlet n with $k_{i,n}$ allocated threads thus is

$$d_{mec}(V_i^{mec}, n, k_{i,n}) = \sum_{v_{i,j} \in V_i^{mec}} d_{i,j}^{mec}(n, k_{i,n}). \tag{4.7}$$

The end-to-end delay: The end-to-end delay of offloading the layer set V_i^{mec} to cloudlet $n \in \mathbb{N}(r_i)$ for request r_i is

$$d_{d2d}(V_i^{loc}, V_i^{mec}, n, k_{i,n}) = d_{loc}(V_i^{loc}) + d_{tran}(V_i^{tran}, n) + d_{mec}(V_i^{mec}, n, k_{i,n}). \quad (4.8)$$

To admit request r_i by meeting its inference delay requirement D_i, we must have

$$d_{d2d}(V_i^{loc}, V_i^{mec}, n, k_{i,n}) \leq D_i. \quad (4.9)$$

4.2.4 Problem definition

Definition 1. Given a set N of cloudlets co-located with APs in a geographical area, and a set of delay-aware DNN requests R issued from IoT devices, each DNN inference request $r_i \in R$ issued from an IoT device has a DNN inference model G_i with the inference delay requirement D_i, assuming that the model has been stored in both its local IoT device and cloudlets. To accelerate the DNN inference, each DNN model is partitioned into two parts: one part is executed in its local IoT device, and another part is offloaded to a cloudlet within the transmission range of the IoT device, and is allocated with up to K (≥ 1) threads in the cloudlet for processing, the *DNN inference throughput maximization problem* in such an MEC environment is to maximize as many DNN inference requests admitted as possible while meeting their inference delay requirements, subject to computing capacity on each cloudlet in N.

Definition 2. Given a set N of cloudlets co-located with APs in a geographical area, and a sequence of delay-aware DNN inference requests arrives one by one without the knowledge of future arrivals, each DNN inference request r_i issued from an IoT device has a DNN inference model G_i with the inference delay requirement D_i, assuming that each DNN inference model is stored in both its local IoT device and cloudlets. To accelerate the DNN inference, each DNN model is partitioned into two parts: one part is executed in its local IoT device, and another part is offloaded to a cloudlet within the transmission range of the IoT device, and is allocated with up to K threads in the cloudlet for inference process, the *dynamic DNN inference throughput maximization problem* in an MEC environment is to maximize as many DNN inference requests admitted as possible without the knowledge of future arrivals, while meeting their inference delay requirements, subject to computing capacity on each cloudlet in N.

Theorem 4.1. *The DNN inference throughput maximization problem is NP-hard.*

Proof. The NP-hardness of the problem is shown through a reduction from an NP-hard problem - the maximum profit generalized assignment problem (GAP) [22], which is defined as follows. Given a set of items and a set of bins, each bin b has a capacity $cap(b)$, and a profit $p(a,b)$ can be collected by packing item a to bin b with size $size(a,b)$. The maximum profit GAP is to maximize the total profit by packing as many items as possible to the bins, subject to the capacities of bins.

We consider a special case of the DNN inference throughput maximization problem, where the transmission delay is neglectable, and the entire DNN model of each request will be offloaded to a cloudlet for inference process. We also assume that each cloudlet allocates one thread for each request, i.e., $K = 1$. We then can calculate the inference delay if request r_i is offloaded to cloudlet $n \in \mathbb{N}(r_i)$. Each bin b_n corresponds to a cloudlet n with the capacity of C_n and each item r_i corresponds to a request r_i. If the inference delay requirement of request r_i can be met by offloading its DNN model to cloudlet n, the profit of packing item r_i to bin b_n is 1, and 0 otherwise. The delay-aware DNN inference problem under this special case is to maximize the number of admitted requests, subject to computing capacities on cloudlets. It can be seen that this special problem is equivalent to the maximum profit GAP. Thus, the delay-aware DNN inference problem is NP-hard, due to the NP-hardness of the maximum profit GAP. $\square$

4.3 Approximation algorithm for DNN inference throughput maximization problem

In this section, we deal with the DNN inference throughput maximization problem by devising an approximate solution for it. Specifically, we first consider the offloading part of the DNN model of request r_i to cloudlet $n \in N$ with $k_{i,n}$ allocated threads, $1 \leq k_{i,n} \leq K$. If such an assignment is feasible (i.e., meeting its inference delay requirement), cloudlet n will become a candidate for the assignment of r_i. We then determine the minimum number $k_{i,n}^{min}$ of threads needed to meet the inference delay requirement. We finally reduce the problem to a maximum profit generalized assignment problem (GAP). Thus, an approximate solution to the latter in turn returns an approximate solution to the former. We also analyze the correctness and time complexity of the proposed approximation algorithm.

4.3.1 Partitioning the DNN model and offloading part of it to cloudlet n with $k_{i,n}$ allocated threads

We assume that part of the DNN model of request r_i will be offloaded to cloudlet n with $k_{i,n}$ allocated threads for processing, $1 \leq k_{i,n} \leq K$. To this end, we reduce this partitioning problem into the maximum flow and minimum cut problem in an auxiliary graph $G'_{i,n,k_{i,n}}$. We then show that the minimum cut in $G'_{i,n,k_{i,n}}$ corresponds to an optimal partition of the DNN, and the value of the minimum cut is the minimum inference delay by offloading part of the DNN model to cloudlet n with $k_{i,n}$ allocated threads. If this minimum inference delay meets the inference delay requirement D_i, the partition of the DNN or part of DNN offloading is feasible.

In the following, we construct the auxiliary graph $G'_{i,n,k_{i,n}}$ for the DNN model partition of request r_i to cloudlet n with $k_{i,n}$ allocated threads, where $n \in \mathbb{N}(r_i)$ and $1 \leq k_{i,n} \leq K$. Recall that $d_{i,j}^{tran}(n)$ represents the transmission delay of the output of layer $v_{i,j}$ to cloudlet n, and $d_{i,j}^{mec}(n, k_{i,n})$ represents the processing delay of layer $v_{i,j}$

on cloudlet n with $k_{i,n}$ threads, respectively. For the sake of convenience, in the rest of discussions, we substitute the notations of $d^{tran}_{i,j}(n)$ and $d^{mec}_{i,j}(n, k_{i,n})$, with $d^{tran}_{i,j}$ and $d^{mec}_{i,j}$, respectively.

Given a DNN model $G_i = (V_i, E_i)$, the construction of the auxiliary flow network $G'_{i,n,k_{i,n}} = (V'_i, E'_i)$ with edge capacity $w(\cdot)$ is given as follows, where $V'_i = \{v_{i,j}, v'_{i,j} \mid v_{i,j} \in V_i\} \cup \{s, t\}$ and $E'_i = \{(s, v_i), (v_{i,j}, v'_{i,j}) \mid v_{i,j} \in V_i\} \cup \{(v_{i,j}, t) \mid v_{i,j} \in V_i \setminus \{v_{i,0}\}\} \cup \{(v'_{i,j}, v_{i,l}), (v_{i,l}, v_{i,j}) \mid (v_{i,j}, v_{i,l}) \in E_i\}$. Specifically, we add two virtual nodes s and t as the source and sink nodes, respectively. We add two nodes $v_{i,j}$ and $v'_{i,j}$ for each node $v_{i,j} \in V_i$. We also add edges $(s, v_{i,j})$ and $(v_{i,j}, v'_{i,j})$ for each node $v_{i,j} \in V_i$, together with the edges $(v_{i,j}, t)$ for each node $v_{i,j} \in V_i \setminus \{v_{i,0}\}$. We also add edges $(v'_{i,j}, v_{i,l})$ and $(v_{i,l}, v_{i,j})$ for each layer dependency $(v_{i,j}, v_{i,l}) \in E_i$ in G_i.

The edge capacity assignment of $G'_{i,n,k_{i,n}}$, as well as the assignment justification, is given as follows.

The capacity $w(s, v_{i,0})$ of edge $(s, v_{i,0}) \in E'_i$ is infinite, which implies that the raw input data of request r_i is at its local IoT device initially, i.e., $w(s, v_{i,0}) = \infty$.

For each edge $(s, v_{i,j}) \in E'_i$ with $v_{i,j} \in V_i \setminus \{v_{i,0}\}$, its capacity is the processing delay $d^{mec}_{i,j}$ of layer $v_{i,j}$ on cloudlet n with $k_{i,n}$ allocated threads, i.e., $w(s, v_{i,j}) = d^{mec}_{i,j}$.

For each edge $(v_{i,j}, t) \in E'_i$ with $v_{i,j} \in V_i \setminus \{v_{i,0}\}$, its capacity is the processing delay $d^{loc}_{i,j}$ of layer $v_{i,j}$ on the local IoT device of request r_i, i.e., $w(v_i, t) = d^{loc}_{i,j}$. Note that $v_{i,0}$ is a virtual input layer on the local IoT device, therefore, edge $(v_{i,0}, t)$ is not included in the auxiliary graph.

For each edge $(v_{i,j}, v'_{i,j}) \in E'_i$ with $v_{i,j} \in V_i$, its capacity is the transmission delay of the output of layer $v_{i,j}$ to cloudlet n, i.e., $w(v_{i,j}, v'_{i,j}) = d^{tran}_{i,j}$.

For each edge $(v'_{i,j}, v_{i,l}) \in E'_i$ with $(v_{i,j}, v_{i,l}) \in E_i$, its capacity is set as infinity, i.e., $w(v'_{i,j}, v_{i,l}) = \infty$. This is because the transmission delay $d^{tran}_{i,j}$ of the output of $v_{i,j}$ has been assigned to the edge $(v_{i,j}, v'_{i,j})$. If the output of $v_{i,j}$ is transmitted to the cloudlet to process all its successor layers $\mathcal{N}^+(v_{i,j})$, $d^{tran}_{i,j}$ is counted only once by including edge $(v_{i,j}, v'_{i,j})$ in a minimum cut of the auxiliary graph,

For each edge $(v_{i,l}, v_{i,j}) \in E'_i$ with $(v_{i,j}, v_{i,l}) \in E_i$, its capacity is set as infinity, i.e., $w(v_{i,l}, v_{i,j}) = \infty$. This is due to the following reason. Given a potential cut M (set of edges) on the auxiliary graph, V'_i is partitioned into two sets V_s and V_t, i.e., $V_s \cup V_t = V'_i$ and $V_s \cap V_t = \emptyset$, where source $s \in V_s$ and sink $t \in V_t$. For each layer dependency $(v_{i,j}, v_{i,l}) \in E_i$, we have that if $v_{i,j} \in V_t$, then $v_{i,l} \in V_t$, which implies that if a layer $v_{i,j}$ is executed on a cloudlet, its successor layer $v_{i,l}$ has to be executed on the cloudlet too. Similarly, we have that if $v_{i,l} \in V_s$, then $v_{i,j} \in V_s$, for each $(v_{i,j}, v_{i,l}) \in E_i$. The claims will be shown in Lemma 9.

One example of the construction of the auxiliary flow network is illustrated in Fig. 4.3. A potential cut in it is $M = \{(v_{i,1}, t), (v_{i,1}, v'_{i,1}), (s, v_{i,2}), (s, v_{i,3}), (s, v_{i,4})\}$, and the set V_i is partitioned into two disjoint sets V_s and V_t, where $V_s = \{s, v_{i,0}, v'_{i,0}, v_{i,1}\}$ and $V_t = \{t, v'_{i,1}, v_{i,2}, v'_{i,2}, v_{i,3}, v'_{i,3}, v_{i,4}, v'_{i,4}\}$. This implies that $v_{i,1}$ is executed in the local IoT device, i.e., $V^{loc}_i = V_s \cap V_i = \{v_{i,0}, v_{i,1}\}$, where $v_{i,0}$ is the virtual input layer. While $v_{i,2}, v_{i,3}$, and $v_{i,4}$ are executed in cloudlet n with $k_{i,n}$ allocated threads, i.e.,

$V_i^{mec} = V_t \cap V_i = \{v_{i,2}, v_{i,3}, v_{i,4}\}$.

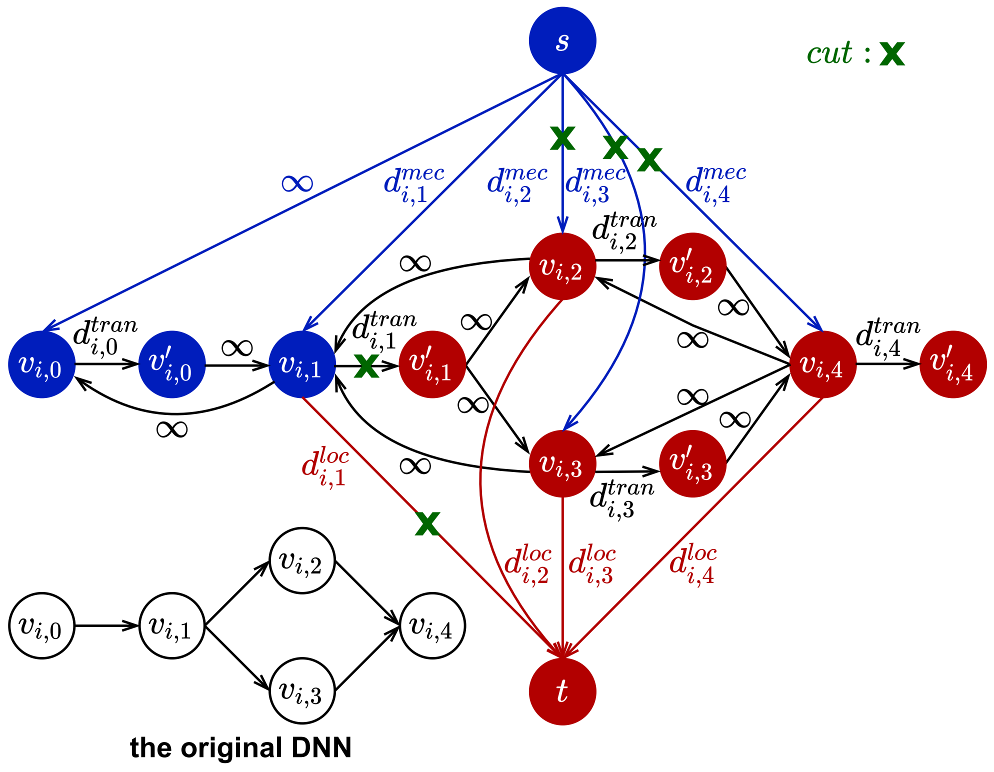


Figure 4.3: An illustrative example of the auxiliary flow network $G'_{i,n,k_{i,n}}$ driven from a DNN (consisting of 4 layers) of request r_i, by assigning part of the DNN to cloudlet n with $k_{i,n}$ allocated threads for the DNN inference process.

It can be seen that the minimum cut M^* in the auxiliary graph $G'_{i,n,k_{i,n}}$ corresponds to a two-partitioning $\{V_i^{loc}, V_i^{mec}\}$ of the DNN G_i that minimizes the inference delay of request r_i when r_i is assigned to cloudlet n with $k_{i,n}$ threads, where $V_i^{loc} = V_s \cap V_i$ and $V_i^{mec} = V_t \cap V_i$. In other words, $\sum_{e \in M^*} w(e)$ is the entire inference delay of the DNN by the partitioning of $\{V_i^{loc}, V_i^{mec}\}$. This claim will be shown rigorously later.

4.3.2 Approximation algorithm

Having partitioned the DNN model of each request into two connected components, the DNN inference throughput maximization problem can be solved through a reduction to a maximum profit GAP as follows.

If the DNN part of request r_i can be offloaded to cloudlet $n \in \mathbb{N}(r_i)$ with $k_{i,n}$ allocated threads to meet its delay requirement D_i, then a minimum number $k_{i,n}^{min}$ of allocated threads in cloudlet n for request r_i with $1 \le k_{i,n}^{min} \le k_{i,n}$ needs to be identified while still meeting the delay requirement. This can be achieved by binary search on the value range $[1, K]$ of $k_{i,n}^{min}$ by constructing no more than $\lceil \log K \rceil$ auxiliary graphs.

Denote by $p(i, n)$ the throughput gain if request r_i is assigned to cloudlet n, where $p(i, n) = 0$ indicates that either (i) cloudlet n is not within the transmission range of the IoT device of request r_i, i.e., $n \notin \mathbb{N}(r_i)$; or (ii) request r_i is not admissible by assigning it to cloudlet n with up to K allocated threads, i.e., its inference delay requirement cannot be met with any DNN partitioning strategy, even if request r_i is assigned to cloudlet n by allocating it with the maximum number of threads K; and $p(i, n) = 1$ otherwise. Note that in case $p(i, n) = 0$, the value of $k_{i,n}^{min}$ is set as ∞.

The problem reduction proceeds as follows. For each cloudlet $n \in \mathcal{N}$, there is a corresponding bin b_n with capacity of C_n. While for each request r_i, there is a corresponding item i with $1 \leq i \leq |R|$. If item i can be assigned to bin b_n with $k_{i,n}^{min}$ allocated threads while meeting its delay requirement, it has a profit of $p(i, n)$ $(= 1)$ with a size of $k_{i,n}^{min}$; otherwise, its profit is $p(i, n) = 0$ with a size of ∞, the problem then is to pack as many items as possible to the $|N|$ bins such that the accumulative profit is maximized, subject to the capacity on each bin. This is the maximum profit GAP, and there is an approximation algorithm for it [22].

The detailed algorithm for the delay-aware DNN inference problem is then given in `Algorithm` 9.

4.3.3 Algorithm analysis

In the following, we analyze the properties of the constructed auxiliary graph $G'_{i,n,k_{i,n}}$, and show the correctness of the proposed approximation algorithm. We also analyze the approximation ratio and time complexity of the approximation algorithm.

Lemma 8. Given a DNN $G_i = (V_i, E_i)$, its layer set V_i is partitioned into two disjoint subsets V_i^{loc} and V_i^{mec}, where V_i^{loc} and V_i^{mec} are executed in the local IoT device of request r_i and a cloudlet in edge computing, respectively. For each layer dependency $(v_{i,j}, v_{i,l}) \in E_i$ in the DNN G_i, we have (i) if $v_{i,j} \in V_i^{mec}$, then $v_{i,l} \in V_i^{mec}$; (ii) if $v_{i,l} \in V_i^{loc}$, then $v_{i,j} \in V_i^{loc}$; and (iii) $v_{i,0} \in V_i^{loc}$.

This is because the DNN model G_i can only be partitioned into two parts.

Lemma 9. Given a DNN inference model $G_i = (V_i, E_i)$ and its auxiliary flow network $G'_{i,n,k_{i,n}} = (V'_i, E'_i)$, let V_s and V_t be the sets of nodes in $G'_{i,n,k_{i,n}}$ partitioned by a potential cut M, where V_s contains source node s and V_t contains destination node t, respectively. For each layer dependency $(v_{i,j}, v_{i,l}) \in E_i$ in G_i, we have (i) if $v_{i,j} \in V_t$, then $v_{i,l} \in V_t$; (ii) if $v_{i,l} \in V_s$, then $v_{i,j} \in V_s$; and (iii) $v_{i,0} \in V_s$.

Proof. In the construction of $G'_{i,n,k_{i,n}}$, for each layer dependency $(v_{i,j}, v_{i,l}) \in E_i$ in G_i, an edge $(v_{i,l}, v_{i,j})$ with capacity of infinity is added to $G'_{i,n,k_{i,n}}$. This implies that edge $(v_{i,l}, v_{i,j})$ in G'_i cannot be included in the cut M, and $v_{i,j}$ is always reachable from $v_{i,l}$. We now show claim (i) by contradiction, i.e., we assume that $v_{i,l} \in V_s$. However, $v_{i,j} \in V_t$ and $v_{i,l}$ can reach $v_{i,j}$ through edge $(v_{i,l}, v_{i,j})$. This contradicts the definition

Algorithm 9 An approximation algorithm for the delay-aware DNN inference problem

Require: A set N of cloudlets co-located with APs in a monitoring area and a set of requests R.

Ensure: Maximize the network throughput by admitting as many DNN inference requests as possible.

```
1: for each request $r_i \in R$ do
2:     for each cloudlet $n \notin \mathbb{N}(r_i)$ do
3:         $k^{min}_{i,n} \leftarrow \infty$; $p(i,n) \leftarrow 0$;
4:     end for;
5:     for each cloudlet $n \in \mathbb{N}(r_i)$ do
6:         Construct auxiliary graph $G'_{i,n,K}$;
7:         Calculate the inference delay $d_{d2d}(r_i, n, K)$ by finding a minimum cut in $G'_{i,n,K}$;
8:         if $d_{d2d}(r_i, n, K) > D_i$ then
9:             $k^{min}_{i,n} \leftarrow \infty$; $p(i,n) \leftarrow 0$; /* the delay requirement of request $r_i$ cannot be met when assigned to cloudlet $n$ with $K$ threads.*/
10:        else
11:            $k_l \leftarrow 1$; $k_r \leftarrow K$; /* Use binary search to find a minimum number $k^{min}_{i,n}$ of threads to meet the delay requirement $D_i$ of request $r_i$ when it is assigned to cloudlet $n$/*
12:            while $k_l \leq k_r$ do
13:                $k_m \leftarrow \lfloor (k_l + k_r)/2 \rfloor$;
14:                Construct auxiliary graph $G'_{i,n,k_m}$;
15:                Calculate the inference delay $d_{d2d}(r_i, n, k_m)$ by finding a minimum cut in $G'_{i,n,k_m}$;
16:                if $d_{d2d}(r_i, n, k_m) \leq D_i$ then
17:                    $k_r \leftarrow k_m - 1$;
18:                else
19:                    $k_l \leftarrow k_m + 1$;
20:                end if;
21:            end while;
22:            $k^{min}_{i,n} \leftarrow k_l$; $p(i,n) \leftarrow 1$;
23:        end if;
24:    end for;
25: end for;
26: Construct a maximum profit GAP instance, where each cloudlet $n$ corresponds to a bin $b_n$ with a capacity $C_n$, each request $r_i$ corresponds to an item $i$ with a size $k^{min}_{i,n}$ and a profit $p(i,n)$ if assigned to bin $b_n$;
27: An approximate solution $\mathbb{A}$ is obtained, by invoking the approximation algorithm for the maximum profit GAP in [22];
28: return the solution $\mathbb{A}$ to the delay-aware DNN inference problem.
```

of a cut. The proof of claim (ii) is similar to that of claim (i), omitted. Claim (iii) always holds, as the capacity of edge $(s, v_{i,0})$ is infinity and node s can reach $v_{i,0}$. $\square$

Lemma 10. Given a DNN model $G_i = (V_i, E_i)$ and its auxiliary flow network $G'_{i,n,k_{i,n}} = (V'_i, E'_i)$, let V_s and V_t be the sets of nodes in $G'_{i,n,k_{i,n}}$ partitioned by a minimum cut, where V_s contains s and V_t contains t, respectively. Let M^* be the set of edges in the minimum cut of $G'_{i,n,k_{i,n}}$. In the auxiliary graph $G'_{i,n,k_{i,n}}$, (i) if $v_{i,j} \in V_t \cap V_i$ with $v_{i,j} \in V_i \setminus \{v_0\}$, then edge $(s, v_{i,j}) \in M^*$ and edge $(v_{i,j}, t) \notin M^*$; (ii) if $v_{i,j} \in V_s \cap V_i$ with $v_{i,j} \in V_i \setminus \{v_0\}$, then edge $(v_{i,j}, t) \in M^*$ and edge $(s, v_{i,j}) \notin M^*$; and (iii) edge $(s, v_{i,0}) \notin M^*$.

Proof. (i) Since edge $(s, v_{i,j})$ directly connects source s to $v_{i,j}$, edge $(s, v_{i,j})$ must be included in M^* by the definition of the minimum cut. Assuming that $(v_{i,j}, t) \in M^*$, as $v_{i,j} \in V_t$, it can be seen that the removal of $(v_{i,j}, t)$ from M^* results in a new cut M' with a smaller value, i.e., $M^* = M' \cup \{(v_{i,j}, t)\}$ and $\sum_{e \in M'} w(e) < \sum_{e \in M^*} w(e)$. This contradicts the assumption that M^* is a minimum cut. Thus, $(v_{i,j}, t) \notin M^*$. (ii) The proof is similar to (i), omitted. (iii) This is due to the fact that the capacity of edge $(s, v_{i,0})$ is infinite. $\square$

Lemma 11. Given a DNN model $G_i = (V_i, E_i)$ and its auxiliary flow network $G'_{i,n,k_{i,n}} = (V'_i, E'_i)$. Each potential DNN partitioning $\{V_i^{loc}, V_i^{mec}\}$ in G_i corresponds to a potential cut M in $G'_{i,n,k_{i,n}}$.

Proof. Similar to Lemma 10, we construct such a cut M on $G'_{i,n,k_{i,n}}$ by any DNN partitioning $\{V_i^{loc}, V_i^{mec}\}$, where $M = \{(s, v_{i,j}) \mid v_{i,j} \in V_i^{mec}\} \cup \{(v_{i,j}, t) \mid v_{i,j} \in V_i^{loc} \setminus \{v_{i,0}\}\} \cup \{(v_{i,j}, v'_{i,j}) \mid v_{i,j} \in V_i^{tran}\}$. Then we have $V_s = \{s\} \cup \{v_{i,j} \mid v_{i,j} \in V_i^{tran}\} \cup \{v_{i,j}, v'_{i,j} \mid v_{i,j} \in V_i^{loc} \setminus V_i^{tran}\}$, and $V_t = \{t\} \cup \{v'_{i,j} \mid v_{i,j} \in V_i^{tran}\} \cup \{v_{i,j}, v'_{i,j} \mid v_{i,j} \in V_i^{mec}\}$.

We now show that M is a feasible cut in $G_{i,n,k_{i,n}}$, i.e., we show that for the given cut M, (i) any node in V_s is reachable from s; and (ii) any node in V_t is not reachable from s.

(i) Source s can reach any node $v_{i,j} \in V_i^{loc}$, since edge $(s, v_{i,j})$ with $v_{i,j} \in V_i^{loc}$ is not added to cut M. Source s can also reach any node $v'_{i,j}$ with $v_{i,j} \in V_i^{loc} \setminus V_i^{tran}$ through edges $(s, v_{i,j})$ and $(v_{i,j}, v'_{i,j})$, because edge $(v_{i,j}, v'_{i,j})$ with $v_{i,j} \in V_i^{loc} \setminus V_i^{tran}$ is not added to cut M.

(ii) We have $\{v'_{i,j} \mid v_{i,j} \in V_i^{tran}\} \subset V_t$, because for each $v_{i,j} \in V_i^{tran}$, node $v'_{i,j}$ has only one incoming edge $(v_{i,j}, v'_{i,j})$, however, $(v_{i,j}, v'_{i,j}) \in M$, then $v'_{i,j}$ is not reachable from s. We now show that $\{v_{i,j} \mid v_{i,j} \in V_i^{mec}\} \subset V_t$ by a contradiction. Assume that there exists a node $v_{i,j} \in V_i^{mec}$, which is reachable from source s. Because $\{(s, v_{i,j}) \mid v_{i,j} \in V_i^{mec}\} \subset M$, source s can reach $v_{i,j}$ only if source s first reaches a node $v_{i,a} \in V_{loc}$ with edge $(s, v_{i,a})$. Also, the path from any node $v_{i,a} \in V_{loc}$ to any node $v_{i,j} \in V_{mec}$ in $G'_{i,n,k_{i,n}}$ must pass through an edge in $\{(v_{i,b}, v'_{i,b}) \mid v_{i,b} \in V_i^{tran}\}$ by the construction of the auxiliary graph. However, with $\{(v_{i,b}, v'_{i,b}) \mid v_{i,b} \in V_i^{tran}\} \subset M$, source s cannot reach any node $v_{i,j} \in V_i^{mec}$ though any node $v_{i,a} \in V_{loc}$, which results in a contradiction. We have $\{v'_{i,j} \mid v_{i,j} \in V_i^{mec}\} \subset V_t$, because each $v'_{i,j}$ has only one incoming edge $(v_{i,j}, v'_{i,j})$ with $v_{i,j} \in V_i^{mec}$, but s cannot reach any node $v_{i,j} \in V_i^{mec}$ as mentioned. To reach sink t, source s must first reach any node in V_i^{mec}, due to $\{(v_{i,j}, t) \mid v_{i,j} \in V_i^{loc} \setminus \{v_{i,0}\}\} \subset M$. However, source s cannot reach any node $v_{i,j} \in V_i^{mec}$ as mentioned. Therefore, source s cannot reach sink t and $t \in V_t$. □

Lemma 12. Given a DNN $G_i = (V_i, E_i)$ and its auxiliary flow network $G'_{i,n,k_{i,n}} = (V'_i, E'_i)$, let V_s and V_t be the node sets of $G'_{i,n,k_{i,n}}$ partitioned by a minimum cut M^*, where V_s contains s and V_t contains t. The minimum cut M^* in $G'_{i,n,k_{i,n}}$ corresponds to a feasible DNN partitioning $\{V_i^{loc}, V_i^{mec}\}$ of G_i, where $V_i^{loc} = V_s \cap V_i$ and $V_i^{mec} = V_t \cap V_i$.

Proof. Let $V_i^{loc} = V_s \cap V_i$ and $V_i^{mec} = V_t \cap V_i$, such a DNN partitioning $\{V_i^{loc}, V_i^{mec}\}$ is feasible, because the minimum cut in $G'_{i,n,k_{i,n}}$ and a feasible DNN partitioning in G_i have the same patterns by Lemma 8 and 9. Especially, we have $\{(s, v_{i,j}) \mid v_{i,j} \in V_i^{mec}\} \cup \{(v_{i,j}, t) \mid v_{i,j} \in V_i^{loc} \setminus \{v_{i,0}\}\} \subset M^*$, by Lemma 10. We can also obtain the set of layers $V_i^{tran} \subset V_i^{loc}$, the successor layers of which are in V_i^{mec}. We then show $\{(v_{i,j}, v'_{i,j}) \mid v_{i,j} \in V_i^{tran}\} \subset M^*$ by a contradiction. Assume that it exists a node $v_{i,j} \in V_{tran}$ with $(v_{i,j}, v'_{i,j}) \notin M^*$, and there is a layer dependency $(v_{i,j}, v_{i,l})$ with $v_{i,j} \in V_{tran} \subset V_i^{loc}$ and $v_{i,l} \in V_{mec}$. As $V_i^{loc} = V_s \cap V_i$ and $V_i^{mec} = V_t \cap V_i$, we have $v_{i,j} \in V_s$ and $v_{i,l} \in V_t$. However, the capacity of edge $(v'_{i,j}, v_{i,l})$ is infinite, and $v_{i,j}$ can reach $v_{i,l}$ though edges $(v_{i,j}, v'_{i,j})$ and $(v'_{i,j}, v_{i,l})$, which contradicts the definition of a cut. Thus we have the minimum cut $M^* = \{(s, v_{i,j}) \mid v_{i,j} \in V_i^{mec}\} \cup \{(v_{i,j}, t) \mid v_{i,j} \in V_i^{loc} \setminus \{v_{i,0}\}\} \cup \{(v_{i,j}, v'_{i,j}) \mid v_{i,j} \in V_i^{tran}\}$, because M^* is already a cut of $G'_{i,n,k_{i,n}}$, which has been shown in Lemma 11. □

Lemma 13. Given a DNN model $G_i = (V_i, E_i)$ and its auxiliary flow network $G'_{i,n,k_{i,n}} = (V'_i, E'_i)$, the minimum cut M^* of $G'_{i,n,k_{i,n}}$ corresponds to a feasible DNN partitioning with the minimum inference delay.

Lemma 13 can be derived from Lemma 11 and Lemma 12.

Theorem 4.2. *Given a set N of cloudlets co-located with APs in a monitoring area, and a set R of delay-aware DNN inference requests with inference delay requirements, there is a $\frac{1}{2+\epsilon}$−approximation algorithm,* `Algorithm` *9, for the DNN inference throughput maximization problem, which takes $O(|R| \cdot |N| \cdot \lceil \log K \rceil \cdot (|V|_{max} + |E|_{max}) \cdot |V|^2_{max} + |N| \cdot |R| \cdot \log \frac{1}{\epsilon} + \frac{|N|}{\epsilon^4})$ time, where ϵ is a constant with $0 < \epsilon \leq 1$, K is the maximum number of threads allocated to a request in any cloudlet, $|V|_{max}$ and $|E|_{max}$ are the maximum numbers of layers and edges in a DNN of any request, respectively.*

Proof. Given the assigned cloudlet n and the number of allocated threads, a minimum cut in the constructed auxiliary graph results in an optimal DNN partitioning to minimize the inference delay of a request r_i by Lemma 13, and a minimum number of allocated threads for request r_i to meet its inference delay requirement is then found if request r_i is assigned to cloudlet n. The approximation ratio of `Algorithm` 9 is $\frac{1}{2+\epsilon}$, derived from the approximation algorithm in [22] directly.

The time complexity of `Algorithm` 9 is analyzed as follows. There are at most $|R| \cdot |N| \cdot \lceil \log K \rceil$ auxiliary graphs for $|R|$ requests to be constructed, while it takes $O((|V|_{max} + |E|_{max}) \cdot |V|^2_{max})$ time to find a minimum cut in each auxiliary graph by the algorithm in [11]. Also, it takes $O(|N| \cdot |R| \cdot \log \frac{1}{\epsilon} + \frac{|N|}{\epsilon^4})$ time for the maximum GAP, by the approximation algorithm in [22]. Thus, the time complexity of `Algorithm` 9 is $O(|R| \cdot |N| \cdot \lceil \log K \rceil \cdot (|V|_{max} + |E|_{max}) \cdot |V|^2_{max} + |N| \cdot |R| \cdot \log \frac{1}{\epsilon} + \frac{|N|}{\epsilon^4})$. □

4.4 Online algorithm for dynamic DNN inference throughput maximization problem

In this section, we deal with dynamic admissions of delay-aware DNN inference requests, assuming that a sequence of requests arrives one by one without the knowledge of future request arrivals. We propose an online algorithm with a provable competitive ratio for the dynamic DNN inference throughput maximization problem.

4.4.1 Online algorithm

In the proposed online algorithm, we adopt the similar strategy as we did for `Algorithm` 9. That is, for each incoming request r_i, we first identify the minimum number $k^{min}_{i,n}$ of allocated threads on each cloudlet $n \in \mathbb{N}(r_i)$ to meet its inference delay requirement, where the value of $k^{min}_{i,n}$ is obtained by binary search on the value range $[1, K]$, and we construct no more than $\lceil \log K \rceil$ auxiliary graphs to find $k^{min}_{i,n}$.

Let $C_n(i)$ be the residual computing capacity (the available number of threads) in cloudlet $n \in \mathbb{N}(r_i)$ when request r_i arrives, and $C_n(1) = C_n$ initially. If request r_i is admitted, $k^{min}_{i,n}$ threads in cloudlet n are allocated to process its offloaded DNN layers to meet its inference delay requirement D_i, then $C_n(i) = C_n(i) - k^{min}_{i,n}$. We model the *usage cost* $w_n(i)$ of cloudlet n by an exponential function when considering request r_i admission as follows.

$$w_n(i) = C_n(\alpha^{1-\frac{C_n(i)}{C_n}} - 1), \tag{4.10}$$

where $(1 - \frac{C_n(i)}{C_n})$ is the computing resource usage ratio of cloudlet n, and $\alpha > 1$ is a tuning parameter that reflects the sensitivity of the computing resource usage ratio of any cloudlet.

The *normalized usage cost* $\psi_n(i)$ of cloudlet n then is defined as

$$\psi_n(i) = \frac{w_n(i)}{C_n} = \alpha^{1-\frac{C_n(i)}{C_n}} - 1. \tag{4.11}$$

When request r_i arrives, we partition its DNN model G_i and calculate the resulting inference delay if $k_{i,n}$ threads in cloudlet $n \in \mathbb{N}(r_i)$ are allocated for request r_i, with $1 \leq k_{i,n} \leq K$, by finding the minimum cut in the constructed auxiliary graph. We then identify a minimum number $k^{min}_{i,n}$ of threads on each cloudlet $n \in \mathbb{N}(r_i)$

to meet the inference delay requirement of request r_i, by binary search on the value range $[1, K]$ of $k_{i,n}^{min}$, which can be achieved by constructing no more than $\lceil \log K \rceil$ auxiliary graphs. Let $Q_i \subset \mathbb{N}(r_i)$ be the set of candidate cloudlets for request r_i, where each cloudlet $n \in Q_i$ has sufficient computing resource (the number of threads $k_{i,n}^{min}$) to meet the inference delay requirement of request r_i, i.e., $C_n(i) \geq k_{i,n}^{min}$.

Request r_i will be rejected if Q_i is empty, i.e., the delay requirement of request r_i cannot be met by assigning r_i to any cloudlet in the MEC network within its transmission range. Otherwise, a candidate cloudlet $n' \in Q_i$ with the minimum normalized usage cost is identified by Eq. (4.11). However, request r_i can still be rejected if its demanded computing resource is too costly. Therefore, an admission control policy to determine the admission of request r_i is proposed as follows: request r_i will be rejected if the normalized usage cost of cloudlet n' is greater than $|N|$, i.e., $\psi_{n'}(i) > |N|$, where N is the set of cloudlets.

The details of the proposed online algorithm for the dynamic DNN inference throughput maximization problem are given in `Algorithm` 10.

4.4.2 Algorithm analysis

In the following, we analyze the competitive ratio and time complexity of the proposed online algorithm, `Algorithm` 10.

Lemma 14. Given an MEC network consisting of a set N of cloudlets, a sequence of delay-aware DNN inference requests arrives one by one without the knowledge of future arrivals. Assume that at most K threads on a cloudlet can be allocated to accelerate the DNN inference of each request. Let $\mathcal{A}(i)$ be the set of requests admitted by `Algorithm` 10 prior to the arrival of request r_i. When request r_i arrives, the sum of the usage costs of all cloudlets in N is

$$\sum_{n \in N} w_n(i) \leq 2 \cdot K \cdot |N| \cdot |\mathcal{A}(i)| \cdot \log_2 \alpha. \tag{4.12}$$

Proof. In case request $r_{i'}$ is rejected, the usage cost of each cloudlet does not change. Otherwise, assuming that request $r_{i'}$ is admitted by assigning r_i' to cloudlet $n' \in \mathbb{N}(r_i)$, i.e., $C_{n'}(i'+1) - C_{n'}(i') = k_{i',n'}^{min} \leq K$, since as most K threads can be allocated to a request on any cloudlet. Then,

$$\begin{aligned}
w_{n'}(i'+1) - w_{n'}(i') &= C_{n'} \cdot (\alpha^{1-\frac{C_{n'}(i'+1)}{C_{n'}}} - 1) - C_{n'} \cdot (\alpha^{1-\frac{C_{n'}(i')}{C_{n'}}} - 1) \\
=& C_{n'} \cdot \alpha^{1-\frac{C_{n'}(i')}{C_{n'}}} \cdot (\alpha^{\frac{C_{n'}(i')-C_{n'}(i'+1)}{C_{n'}}} - 1) \\
\leq& C_{n'} \cdot \alpha^{1-\frac{C_{n'}(i')}{C_{n'}}} \cdot (\alpha^{\frac{K}{C_{n'}}} - 1) = C_{n'} \cdot \alpha^{1-\frac{C_{n'}(i')}{C_{n'}}} \cdot (2^{\frac{K}{C_{n'}} \cdot \log_2 \alpha} - 1) \\
\leq& C_{n'} \cdot \alpha^{1-\frac{C_{n'}(i')}{C_{n'}}} \cdot \frac{K}{C_{n'}} \cdot \log_2 \alpha. && (4.13) \\
=& K \cdot \alpha^{1-\frac{C_{n'}(i')}{C_{n'}}} \cdot \log_2 \alpha, && (4.14)
\end{aligned}$$

Algorithm 10 Online algorithm for the dynamic DNN inference throughput maximization problem

Require: An MEC network consisting of a set N of cloudlets, and a sequence of requests arriving one by one without the knowledge of future arrivals.

Ensure: Maximize the number of admitted requests.

```
1: $\mathbb{A} \leftarrow \emptyset$; /* the solution */
2: while a request $r_i$ arrives do
3:     $Q_i \leftarrow \mathbb{N}(r_i)$; /* the candidate cloudlet set for $r_i$ */
4:     for each cloudlet $n \in Q_i$ do
5:         Construct auxiliary graph $G'_{i,n,K}$, and calculate the inference delay $d_{d2d}(r_i, n, K)$ by finding a minimum cut in $G'_{i,n,K}$;
6:         if $d_{d2d}(r_i, n, K) > D_i$ then
7:             $k^{min}_{i,n} \leftarrow \infty$;
8:         else
9:             $k_l \leftarrow 1$; $k_r \leftarrow K$;
10:            while $k_l \leq k_r$ do
11:                $k_m \leftarrow \lfloor (k_l + k_r)/2 \rfloor$; Construct auxiliary graph $G'_{i,n,k_m}$, and calculate the inference delay $d_{d2d}(r_i, n, k_m)$ by finding a minimum cut in $G'_{i,n,k_m}$;
12:                if $d_{d2d}(r_i, n, k_m) \leq D_i$ then
13:                    $k_r \leftarrow k_m - 1$;
14:                else
15:                    $k_l \leftarrow k_m + 1$;
16:                end if;
17:            end while;
18:            $k^{min}_{i,n} \leftarrow k_l$;
19:        end if
20:        if $k^{min}_{i,n} > C_n(i)$ then
21:            $Q_i \leftarrow Q_i \setminus \{n\}$;
22:        end if
23:    end for
24:    if $Q_i = \emptyset$ then
25:        Reject request $r_i$;
26:    else
27:        Identify the cloudlet $n' \in Q_i$ with the minimum normalized usage cost $\psi_{n'}(i)$, by Eq. (4.11);
28:        if $\psi_{n'}(i) > |N|$ then
29:            Reject request $r_i$;
30:        else
31:            Admit request $r_i$ by assigning the DNN part of $r_i$ to cloudlet $n'$, and $\mathbb{A} \leftarrow \mathbb{A} \cup \{r_i\}$;
32:        end if;
33:    end if;
34: end while;
35: return Solution $\mathbb{A}$ to the dynamic DNN inference throughput maximization problem.
```

where Ineq (4.13) holds since $2^x - 1 \leq x$ with $0 \leq x \leq 1$.

If request $r_{i'}$ is admitted by assigning it to cloudlet n', only the usage cost of cloudlet n' changes while the usage cost of any other cloudlet remains the same. Thus, after admitting request $r_{i'}$, we have

$$\begin{aligned}
&\sum_{n \in N} (w_n(i'+1) - w_n(i')) = w_{n'}(i'+1) - w_{n'}(i') \\
\leq& K \cdot \alpha^{1-\frac{C_{n'}(i')}{C_{n'}}} \cdot \log_2 \alpha, \quad \text{by Eq. (4.14)} \\
=& K \cdot \log_2 \alpha \cdot ((\alpha^{1-\frac{C_{n'}(i')}{C_{n'}}} - 1) + 1) \\
=& K \cdot \log_2 \alpha \cdot (\psi_{n'}(i') + 1), \quad \text{by Eq. (4.11)} \\
\leq& K \cdot \log_2 \alpha \cdot (|N| + 1) && (4.15) \\
\leq& 2 \cdot K \cdot |N| \cdot \log_2 \alpha, && (4.16)
\end{aligned}$$

where Ineq.(4.15) holds as the admission control policy is met.

We then calculate the sum of the usage costs of all cloudlets prior to the arrival of request r_i as follows.

$$\begin{aligned}
\sum_{n \in N} w_n(i) =& \sum_{i'=1}^{i-1} \sum_{n \in N} (w_n(i'+1) - w_n(i')) \\
=& \sum_{r_{i'} \in \mathcal{A}(i)} \sum_{n \in N} (w_n(i'+1) - w_n(i')) \\
\leq& \sum_{r_{i'} \in \mathcal{A}(i)} (2 \cdot K \cdot |N| \cdot \log_2 \alpha), \quad \text{by Eq. (4.16)} \\
=& 2 \cdot K \cdot |N| \cdot |\mathcal{A}(i)| \cdot \log_2 \alpha. && (4.17)
\end{aligned}$$

Hence, the lemma follows. □

Lemma 15. Given an MEC network consisting of a set N of cloudlets, a sequence of delay-aware DNN inference requests arrives one by one without the knowledge of future arrivals. Assume that at most K threads can be allocated on any cloudlet to accelerate the DNN inference of a request. Let $\mathcal{B}(i)$ be the set of requests admitted by the optimal solution but rejected by `Algorithm` 10 prior to the arrival of request r_i. Each request $r_{i'} \in \mathcal{B}(i)$ is assumed to be assigned to cloudlet $n^*_{i'}$ in the optimal solution. Then, for each request $r_{i'} \in \mathcal{B}(i)$, we have

$$\psi_{n^*_{i'}}(i') > |N|, \tag{4.18}$$

when $2|N| + 2 \leq \alpha \leq 2^{\frac{C_{min}}{K}}$, and $C_{min} = \min_{n \in N}\{C_n\}$ is the minimum computing capacity of a cloudlet.

Proof. Since request $r_{i'}$ is admitted by the optimal solution, it can be seen that there is at least one cloudlet in the transmission range of the local IoT device of request $r_{i'}$, i.e., $\mathbb{N}(r_{i'}) \neq \emptyset$. We distinguish the rejection of request $r_{i'}$ by `Algorithm` 10 into two cases: Case 1. Cloudlet $n^*_{i'}$ does not have sufficient computing resource for request $r_{i'}$ when $r_{i'}$ arrives. Case 2. Cloudlet $n^*_{i'}$ has sufficient computing resource for request $r_{i'}$ when $r_{i'}$ arrives, and request r_i is allocated to cloudlet $n_{i'}$ ($n_{i'}$ could be $n^*_{i'}$), by `Algorithm` 10. However, the admission control policy is violated, and request $r_{i'}$ is rejected.

Case 1. Cloudlet $n^*_{i'}$ does not have sufficient computing resource for request $r_{i'}$ when $r_{i'}$ arrives, i.e., $C_{n^*_{i'}}(i') < k^{min}_{i',n^*_{i'}} \leq K$. We then have

$$\begin{aligned}
\psi_{n^*_{i'}}(i') &= \alpha^{1-\frac{C_{n^*_{i'}}(i')}{C_{n^*_{i'}}}} - 1 > \alpha^{1-\frac{K}{C_{n^*_{i'}}}} - 1 \\
&\geq \alpha^{1-\frac{1}{\log_2 \alpha}} - 1, \quad \text{when } \alpha \leq 2^{\frac{C_{min}}{K}} \leq 2^{\frac{C_{n^*_{i'}}}{K}} \\
&= \frac{\alpha}{2} - 1 \geq |N|, \quad \text{when } \alpha \geq 2|N| + 2.
\end{aligned}$$

Case 2. `Algorithm` 10 assigns request $r_{i'}$ to cloudlet $n_{i'} \in \mathbb{N}(r_{i'})$ with the minimum normalized usage cost by Eq. (4.11), we have

$$\psi_{n^*_{i'}}(i') \geq \psi_{n_{i'}}(i'). \tag{4.19}$$

However, the admission control policy is violated if request $r_{i'}$ is assigned to cloudlet $n_{i'}$, i.e., $\psi_{n_{i'}}(i') > |N|$, we have

$$\psi_{n^*_{i'}}(i') > |N|. \tag{4.20}$$

Hence, the lemma follows. □

Theorem 4.3. *Given a set N of cloudlets co-located with $|N|$ APs, each cloudlet $n \in N$ has computing capacity c_v in terms of the number of threads, a sequence of delay-aware DNN inference requests arrives one by one without the knowledge of future arrivals. Assuming at most K threads in a cloudlet can be allocated to accelerate the DNN inference of each request, there is an online algorithm for the dynamic DNN inference throughput maximization problem,* `Algorithm` *10, with a competitive ratio of $O(\log |N|)$. The algorithm takes $O(|N| \cdot \lceil \log K \rceil \cdot (|V_{max}| + |E_{max}|) \cdot |V_{max}|^2)$ time for the admission of each request when $\alpha = 2|N| + 2$, where N is the set of cloudlets, $|V_{max}|$ and $|E_{max}|$ are the maximum numbers of layers and edges in a DNN among the DNN models of all requests, respectively.*

Proof. Let $\mathcal{P}_{opt}(i)$ be the set of admitted requests by an optimal solution prior to the arrival of request r_i. $\mathcal{A}(i)$ is the set of request admitted by `Algorithm` 10 prior to the arrival of request r_i, while $\mathcal{B}(i)$ is the set of requests admitted by the optimal solution but rejected by `Algorithm` 10 prior to the arrival of request r_i. Denote by $\mathcal{B}(i, n)$ the set of requests in $\mathcal{B}(i)$ which are assigned to cloudlet n by an optimal solution, i.e.,

$\mathcal{B}(i,n) = \{r_{i'} \mid r_{i'} \in \mathcal{B}(i), n^*_{i'} = n\}$. It can be seen that $\mathcal{B}(i) = \cup_{n\in N}\mathcal{B}(i,n)$. Then,

$$
\begin{aligned}
|N| \cdot (|\mathcal{P}_{opt}(i)| - |\mathcal{A}(i)|) &\le |N| \cdot |\mathcal{B}(i)| = \sum_{r_{i'}\in\mathcal{B}(i)} |N| \\
&< \sum_{r_{i'}\in\mathcal{B}(i)} \psi_{n^*_{i'}}(i'), \quad \text{by Lemma 15} \\
&\le \sum_{r_{i'}\in\mathcal{B}(i)} \psi_{n^*_{i'}}(i) \qquad (4.21) \\
&= \sum_{r_{i'}\in\mathcal{B}(i)} \frac{w_{n^*_{i'}}(i)}{C_{n^*_{i'}}} = \sum_{r_{i'}\in\mathcal{B}(i,n)} \sum_{n\in\mathcal{N}} \frac{w_n(i)}{C_n} \\
&\le \sum_{n\in N} w_n(i) \sum_{r_{i'}\in\mathcal{B}(i,n)} \frac{1}{C_n} \qquad (4.22) \\
&\le \sum_{n\in N} w_n(i) \qquad (4.23) \\
&\le 2 \cdot K \cdot |N| \cdot |\mathcal{A}(i)| \cdot \log_2 \alpha, \quad \text{by Lemma 14,} \qquad (4.24)
\end{aligned}
$$

where Ineq. (4.21) holds since the computing resource consumption of any cloudlet does not decrease. Ineq. (4.22) holds as $\sum_{i=1}^{p}\sum_{j=1}^{q} A_iB_j \le \sum_{i=1}^{p} A_i \sum_{j=1}^{q} B_j$, with $A_i \ge 0$ and $B_j \ge 0$. Ineq. (4.23) holds because each admitted request consumes at least one thread, and the computing capacity of each cloudlet is not violated. Then we have

$$|\mathcal{P}_{opt}(i)| - |\mathcal{A}(i)| < 2 \cdot K \cdot \log_2 \alpha \cdot |\mathcal{A}(i)|. \qquad (4.25)$$

Thus, we have

$$
\begin{aligned}
\frac{|\mathcal{P}_{opt}(i)|}{|\mathcal{A}(i)|} &= \frac{|\mathcal{P}_{opt}(i)| - |\mathcal{A}(i)|}{|\mathcal{A}(i)|} + 1 \\
&< \frac{2 \cdot K \cdot \log_2 \alpha \cdot |\mathcal{A}(i)|}{|\mathcal{A}(i)|} + 1 \\
&= 2 \cdot K \cdot \log_2 \alpha + 1 = O(\log |N|), \quad \text{when } \alpha = 2|N| + 2.
\end{aligned}
$$

We then analyze the time complexity of `Algorithm` 10 as follows. Upon the arrival of request r_i, at most $|N| \cdot \lceil \log K \rceil$ auxiliary graphs are constructed, and it takes $O((|V|_{max} + |E|_{max}) \cdot |V|^2_{max})$ time to find a minimum cut in each auxiliary graph by the algorithm in [11]. Finding the cloudlet with the minimum normalized usage cost takes $O(|N|)$ time. Thus, `Algorithm` 10 takes $O(|N| \cdot \lceil \log K \rceil \cdot (|V_{max}| + |E_{max}|) \cdot |V_{max}|^2)$ time for the admission of each request. □

4.5 Performance Evaluation

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

4.5.1 Experimental Settings

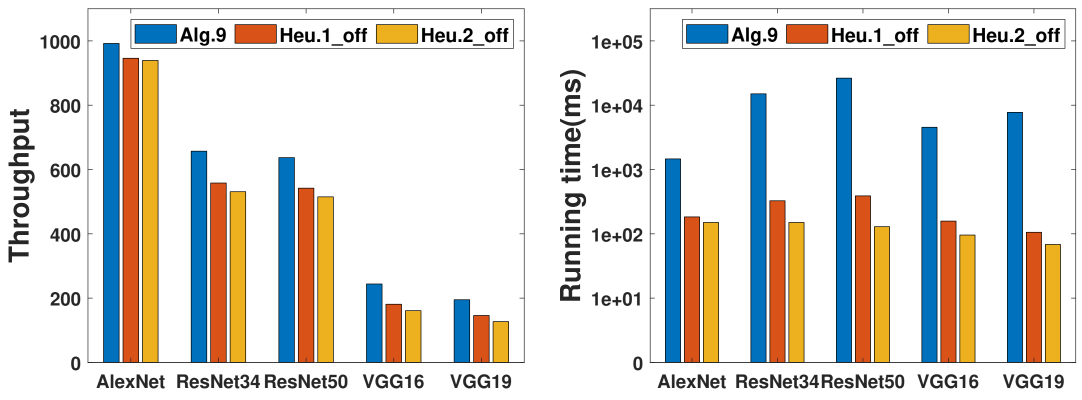
The geographical area is a 1 $km \times 1$ km square area [72] and there are 100 cloudlets deployed on a regular 10×10 grid MEC network, each of which is co-located with an AP. The bandwidth of each AP varies from 2 MHz to 10 MHz [36], and each AP can provide services to a user within 100 m [94]. There are delay-aware 1,000 DNN inference requests, each of which is issued from an IoT device, and the IoT devices are randomly scattered over the defined geographical area. For DNN inferences, we here consider the well-known DNNs: {AlexNet, ResNet34, ResNet50, VGG16, VGG19}. Each request contains an image for inference, which is extracted from the videos in the self-driving dataset BDD100K [114]. The transmission power of each IoT device is randomly drawn between 0.1 Watt and 0.5 Watt [18; 94], noise power σ^2 is set as 1×10^{-10} Watt, and the path loss factor β is set as 4 [18]. The inference delay requirement of a request varies from 0.1 s to 0.3 s. Each cloudlet is assumed to be equipped with 32, 48, or 64 CPU cores [7]. Recall that we assume that each CPU core corresponds to a thread, as applying hyper-threading technique (generate two threads per CPU core) may affect the performance due to additional context switching [66]. We further assume that at most 10 threads (CPU cores) can be allocated for a request in any cloudlet, i.e., $K = 10$. All CPU cores in a cloudlet are assumed to share the same clock speed that varies from 2.5 GHz to 3 GHz in different cloudlets [7]. And the clock speed of each IoT device ranges from 0.5 GHz to 1 GHz [18]. Each cloudlet or IoT device is assumed to conduct 4 floating-point operations per cycle [72]. The parameter ϵ in `Algorithm` 9 is set as 0.5 by Theorem 4.2. For `Algorithm` 10, the parameter α in Eq. (4.11) is set as $2|N| + 2$ by Theorem 4.3. The value in each figure is the mean of the results out of 15 different runs of MEC network instances of the same size. The running time of each algorithm is obtained, based on a desktop with a 3.60 GHz Intel 8-Core i7-7700 CPU and 16 GB RAM. Unless otherwise specified, the above parameters are adopted by default.

To evaluate the performance of `Algorithm` 9 (referred to as `Alg.9`) for the DNN inference throughput maximization problem, we here introduce two heuristic algorithms as benchmarks: algorithms `Heu.1_off` and `Heu.2_off`. Algorithm `Heu.1_off` is based on an existing DNN partitioning strategy *Neurosurgeon* [45], which, however, only works for chain-topology DNNs. Therefore, we preprocess each DAG-topology DNN by a topological sorting approach as did in [36], and then adopt Neurosurgeon to partition the DNN between the local IoT device and a cloudlet. For each request r_i, we can find the minimum number $k_{i,n}^{min}$ of needed threads among all cloudlets to meet its inference delay requirement. Algorithm `Heu.1_off` then admits a request r_i with the minimum number $k_{i,n}^{min}$ of threads among all requests, iteratively, until no more request can be admitted. Algorithm `Heu.2_off` offloads the entire DNN model of a request to its nearest cloudlet with the number of needed threads to meet its inference delay requirement. Similarly, algorithm `Heu.2_off` admits a request with the minimum number of needed threads among all requests, iteratively, until no more requests can be admitted.

To investigate the performance of `Algorithm` 10 (referred to as `Alg.10`) for the dynamic DNN inference throughput maximization problem, two heuristic algorithms `Heu.1_on` and `Heu.2_on` are proposed, which are the online versions of algorithms `Heu.1_off` and `Heu.2_off`, respectively, i.e., algorithm `Heu.1_on` greedily admits each incoming request by offloading the request to the cloudlet with the minimum number of threads needed among all cloudlets, by the modified strategy - Neurosurgeon. In contrast, algorithm `Heu.2_on` greedily admits each incoming request by offloading its entire DNN model to the nearest cloudlet of the request that has sufficient numbers of threads. Either algorithm `Heu.1_on` or algorithm `Heu.2_on` rejects an incoming request if no cloudlet has the number of threads needed.

4.5.2 Performance evaluation of the proposed algorithm for the DNN inference throughput maximization problem

We first evaluated the performance of algorithm `Alg.9` against the algorithms `Heu.1_off` and `Heu.2_off` with 1,000 requests, over the considered DNNs, respectively. Fig. 4.4 demonstrates the throughput and running time delivered by different algorithms. It can be seen from Fig. 4.4(a) that algorithm `Alg.9` admits more requests than algorithms `Heu.1_off` and `Heu.2_off` in all cases. For example, for ResNet34, algorithm `Alg.9` outperforms algorithms `Heu.1_off` and `Heu.2_off` by 17.7% and 23.7%, respectively. This is because algorithm `Alg.9` establishes an efficient DNN partitioning strategy for each inference request, and makes resource-efficient decisions of request admissions, compared with the benchmark algorithms. Also, the performance of algorithm `Alg.9` over VGG19 is 19.7% of itself over AlexNet. This is because the inference over VGG19 requires the largest number of floating-point operations (about 19.6 G), while the inference over AlexNet requires the smallest number of floating-point operations (about 0.7 G).

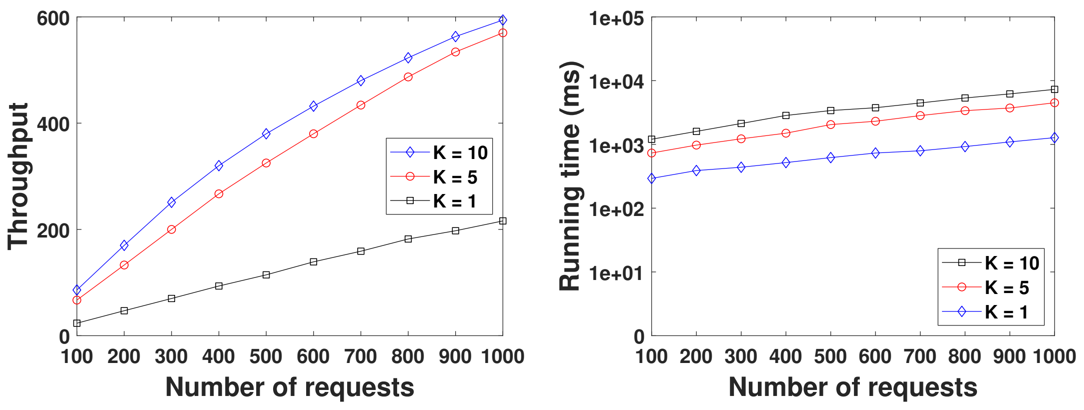


(a) The throughput (b) The running time

Figure 4.4: Performance of different algorithms for the DNN inference throughput maximization problem over different DNNs.

We then studied the impact of parameter K on the performance of algorithm `Alg.9`, by varying the number of requests from 100 to 1,000, where the requested

DNN of each request is randomly drawn from the predefined DNN set. Fig. 4.5 illustrates the throughput and running time of algorithm `Alg.9` when $K = 1$, 5, and 10, respectively. From Fig. 4.5(a), we can see that when the number of requests is $1,000$, the performance of algorithm `Alg.9` when $K = 1$ is 36.4% of itself when $K = 10$. This is due to the fact that a larger value of K implies that more threads can be allocated to accelerate the DNN inference of a request to meet its inference delay requirement.



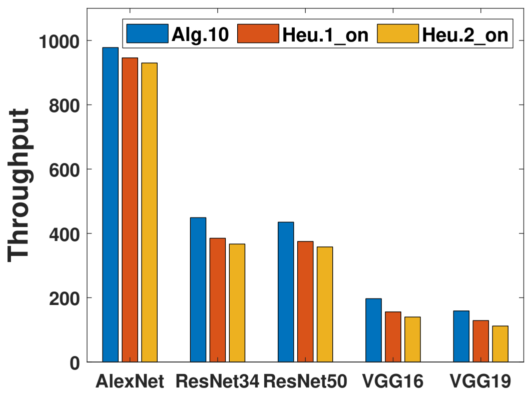
(a) The throughput (b) The running time

Figure 4.5: Impact of parameter K on the performance of algorithm `Alg.9`.

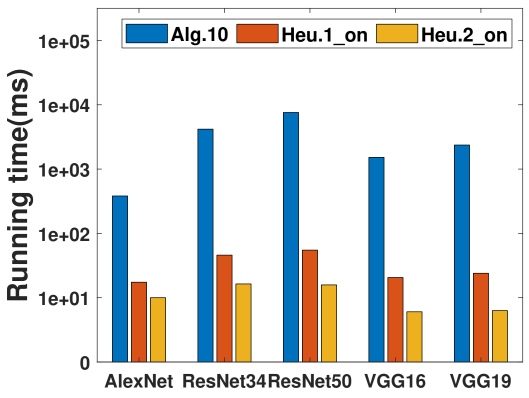
4.5.3 Performance evaluation of the proposed algorithms for the dynamic DNN inference throughput maximization problem

We now investigated the performance of algorithm `Alg.10` against algorithms `Heu.1_on` and `Heu.2_on` for the dynamic DNN inference throughput maximization problem, over the considered DNNs, respectively. Fig. 4.6 plots the throughput and running time of different algorithms. From Fig. 4.6(a), we can see that algorithm `Alg.10` outperforms algorithms `Heu.1_on` and `Heu.2_on` respectively. For example, for ResNet34, algorithm `Alg.9` outperforms algorithms `Heu.1_off` and `Heu.2_off` by 18.7% and 21.1%, respectively. The reason is that algorithm `Alg.10` assigns an incoming request to a cloudlet with the minimum normalized usage cost by Eq. (4.11), and minimizes the number of threads needed to meet its delay requirement by an efficient DNN partitioning strategy. A well-designed admission control policy is also adopted by algorithm `Alg.10` to avoid resource overspending. In Fig. 4.6(a), algorithm `Alg.10` over AlexNet has the best performance, among the predefined DNN set, because the inference over AlexNet requires the smallest number of floating-point operations, which is consistent with the performance behaviors in Fig. 4.4(a).

We then studied the impact of parameter K on the performance of algorithm `Alg.10`, by varying the number of requests from 100 to $1,000$, where the requested DNN of each request is randomly drawn from the predefined DNN set. Fig. 4.7



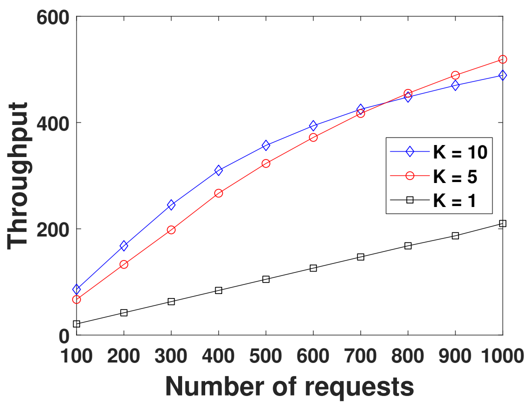
(a) The throughput



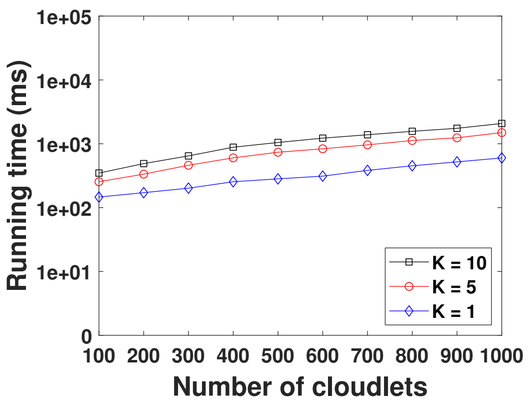
(b) The running time

Figure 4.6: Performance of different algorithms for the dynamic DNN inference throughput maximization problem over different DNNs.

depicts the performance curves of algorithm `Alg.10` when $K = 1$, 5, and 10, respectively. It can be seen from Fig 4.7(a) that when the number of requests is 100, the throughput achieved by algorithm `Alg.10` with $K = 1$ is 23.2% of itself with $K = 10$. This is because more requests with stringent delay requirements can be admitted with a large value of K in the case of abundant network resource (with a small number of incoming requests). However, with over 800 requests, algorithm `Alg.10` with $K = 5$ delivers the best performance, compared with itself with $K = 1$ or 10. The rationale behind this is that when the resource is limited (with a large number of incoming requests), setting K with a reasonable value helps maximize the throughput by reserving the resource for future request admissions.



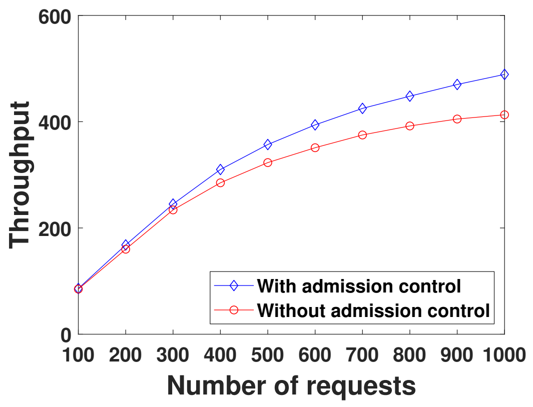
(a) The throughput



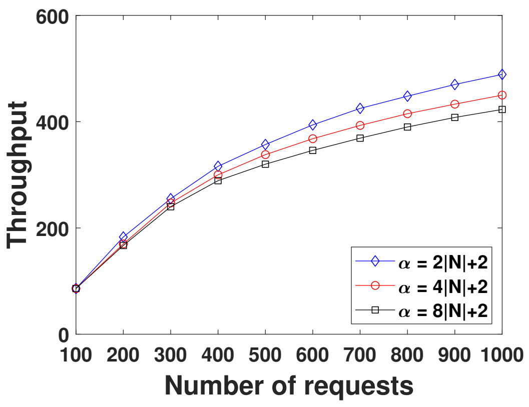
(b) The running time

Figure 4.7: Impact of parameter K on the performance of algorithm `Alg.10`.

We finally evaluated the impact of the admission control policy and parameter α on the performance of algorithm `Alg.10`, where $\alpha > 1$ is a tuning parameter that reflects the sensitivity of the computing resource usage ratio of any cloudlet. Fig 4.8 shows the performance behaviors of algorithm `Alg.10` with and without the admission control policy. Algorithm `Alg.10` with the admission control policy admitted 18.4% more requests than itself without the admission control policy in the case of $1,000$ cloudlets. This can be justified by that an efficient admission control policy intends to admit requests with a small number of threads needed, therefore, the achieved throughput is maximized. Fig 4.8(b) plots the throughput curves of algorithm `Alg.10` with parameter $\alpha = 2|N| + 2$, $4|N| + 2$, and $8|N| + 2$, respectively, where $|N|$ is the number of cloudlets. It can be seen from Fig 4.8(b) that when the number of requests reaches $1,000$, the performance of algorithm `Alg.10` with $\alpha = 8|N| + 2$ policy admitted 86.5% of itself with $\alpha = 2|N| + 2$. This is because the normalized usage cost increases with the increase on the value of parameter α by Eq. (4.11), and algorithm `Alg.10` intends to reject the incoming requests by the admission control policy.



(a) The throughput



(b) The throughput

Figure 4.8: The impacts of the admission control policy and parameter α on the performance of algorithm `Alg.10`.

4.6 Summary

In this chapter, we investigated the DNN inference service provisioning with inference delay requirements in an MEC environment. We studied a DNN inference throughput maximization problem with the aim to maximize the number of requests admitted, subject to computing capacities on cloudlets in the MEC network. To meet the inference delay requirement of each request, we jointly explored the DNN model partitioning and multi-thread parallelism in the cloudlet for inference acceleration. We then showed the NP-hardness of the problem and devised an approximation algorithm with a provable approximation ratio for the problem. In addition, we also considered the dynamic DNN inference throughput maximization problem where a sequence of dealy-aware DNN inference requests arrives one by one without the knowledge of future arrivals, for which we developed an online algorithm with a provable competitive ratio. We finally evaluated the performance of the proposed algorithms by experimental simulations. Experimental results demonstrated that the proposed algorithms are promising.

Chapter 5

Robust Service Provisioning with Service Function Chain Requirements

In this chapter, we first introduce a novel Robust Service Function Chain Placement (RSFCP) problem under the uncertainty assumption of both computing resource and data rates demanded by request executions, through the placement of Service Function Chains (SFCs). We first formulate the RSFCP problem as a Quadratic Integer Programming (QIP). We then develop a near-optimal approximation algorithm for it, by adopting the Markov approximation technique. We also analyze the proposed approximation algorithm with the optimality gap, and the bounds on the convergence time and perturbation caused by resource demand uncertainties.

5.1 Introduction

Service robustness is crucial to limit the performance degradation of network services with unpredictable changes, due to dynamic resource demands [5]. Most previous studies considered the service function chain placement (SFCP) problem under the assumption of accurate amounts of resources demanded for its VNF instance placements of each request [9; 40; 63; 92; 96; 123; 121; 119]. In reality, there is an uncertainty of resource demands during various stages of a request implementation [89]. In addition, there exists potential blockage during the transmission of 5G mmWave signals [39], and the resource demands are time-varying to guarantee high-quality task offloading [120].

The novelty of this chapter lies in the formulation of a novel Robust Service Function Chain Placement (RSFCP) problem in MEC environments, under the assumption of both demanded resource uncertainty and measurement inaccuracy, and a near-optimal approximation algorithm with a good performance guarantee for the problem is devised.

The main contributions of this chapter are as follows. We first formulate a novel RSFCP problem with the aim to maximize the expected profit collected by the network service provider of an MEC network, under the uncertainty assumption of both computing resource and data rate demands in the implementation of a user request. We show that the problem is NP-hard and provide a Quadratic Integer Programming (QIP) formulation for it. We then develop an efficient approximation algorithm for it. Specifically, we start with a special case of the problem where the measurement of the expected demanded resources for each request admission is accurate, under which we propose a near-optimal approximation algorithm by adopting the Markov approximation technique, which can achieve a provable optimality gap. We then extend the proposed approach to the problem of concern, for which we show that the proposed algorithm is still applicable, and the solution delivered has a moderate optimality gap with bounded perturbation errors on the profit measurement. We finally evaluate the performance of the proposed algorithm through experimental simulations. Experimental results demonstrate that the proposed algorithm is promising, and outperforms the baseline algorithms.

The remainder of this chapter is organized as follows. Section 5.2 introduces the system model and the problem definition. Section 5.3 formulates a QIP solution to the problem and shows the NP-hardness of the problem. Sections 5.4 proposes a near-optimal approximation algorithm for the problem. Section 5.5 evaluates the performance of the proposed algorithm by experimental simulations, and a summary is given in Section 5.6.

5.2 Preliminaries

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

5.2.1 System model

We consider an MEC network that consists of Access Points (APs), cloudlets, and links connecting the APs. Each cloudlet in which VNFs are deployed, is co-located with an AP, while an AP may not necessarily be co-located with any cloudlet. The MEC network can be represented by an undirected graph $G = (V, E)$, where V is the set of network nodes and E is the set of optical links [49; 54]. The set V is the union of the set V_C in which each network node consists of both an AP and a cloudlet, and the set V_A in which each network node consists of an AP only. For the sake of convenience, in the rest of the discussion, we only focus on computing resource consumption in each cloudlet. Each cloudlet $v \in V_C$ has computing capacity c_v. For a node $v \in V_A$ without any computing resource, its capacity is $c_v = 0$.

Denote by $N^+(v)$ the set that contains all neighbors of node v in G and itself, i.e., $N^+(v) = \{u \mid (u, v) \in E\} \cup \{v\}$. Each link $(v, u) \in E$ between nodes v and u has bandwidth capacity $c(v, u)$. Denote by $l(v, u)$ the latency per unit data traffic along link $(v, u) \in E$.

Denoted by F the set of VNFs in the MEC and any SFC consists of some VNFs from F in a certain order. The VNFs are instantiated in virtual machines on cloudlets, and denote by c_f the required amount of computing resource to deploy a specific VNF of type f for all $f \in F$. Denote by l_f^v the processing time per unit data traffic of VNF f on node v.

5.2.2 Uncertainties of demanded computing resource and data rate

Considering a set R of user requests, we assume that the demanded data rate B_i of a request $r_i \in R$ is not precisely measured, and the amount of computing resource c_f of a VNF f is various at different stages of a request implementation. In a realistic scenario, the network service provider is informed of the exact requested SFC instead of the exact resource requirements of a request (e.g., the demanded data rate B_i and the amount of computing resource c_f for a VNF of f are not known until the request is processed to its completion [89]). Although the same types of VNFs are established, they always have similar but different computing resource requirements [80]. The network system can perform some prediction mechanisms to obtain necessary information about resource demands of admitted requests and to estimate the amounts of computing resource and data rate demands by analyzing past traces [26].

We first assume that both the lower bounds and the upper bounds on the expected demanded computing resource and the data rate of each request are given in advance, which can be obtained through analyzing the past traces [26]. Recall that B_i is the data rate of request r_i. Denote by B_i^L and B_i^U the lower and upper bounds on B_i with expectation $\overline{B_i}$, which are given in advance. Similarly, denote by c_f^L and c_f^U the lower and upper bounds on the computing resource demand c_f of a VNF instance of function f with expectation $\overline{c_f}$. However, it is difficult to accurately estimate the expected amount of demanded resources with past traces, due to the uncertainty of actual resource demands in the execution of requests. Therefore, the expected amounts of resources may experience perturbations and the analysis of such perturbations will be given later. We also introduce an adjustable control parameter of cost variation caused by the uncertain resource demands in our model to achieve a desired stable system performance, which will be shown later.

5.2.3 User requests with both SFC and latency requirements

Each request $r_i \in R$ is represented by a tuple $\langle s_i, d_i, B_i, \mathcal{L}_i, sfc_i, pay_i \rangle$, where s_i and d_i are the source and destination nodes of the data traffic of the request, respectively, B_i is the demanded data rate, $\mathcal{L}_i$ is the latency requirement, sfc_i is the service function chain, and pay_i is the payment of the request. For request r_i, its latency consists of the processing times of r_i on cloudlets and the communication latency along links in the routing path of r_i [70].

Although the data traffic of request r_i has to start from node s_i and end at node d_i, the deployed VNF instances in its SFC may be in neither of them. Meanwhile, a specific SFC may go through a node $v \in V$ without any VNF instance deployed on it. Also, the routing path of a request may do retracing multiple times. Therefore, it is complicated to characterize the traffic flow under the context of SFCs, and we herein introduce a set of dummy VNFs, denoted by $\mathbb{D}$, in a routing path construction of data traffic to ensure that when a routing path for request r_i passes through a node $v \in V$, at least one VNF or one dummy VNF is deployed on it. Therefore, the deployment of the dummy VNF helps the Quadratic Integer Programming (QIP) formulation of the problem, which will be formally defined later. And the solution by the QIP formulation serves as the exact solution to the problem. Especially, dummy VNFs g_0 and g_1 are always appended at the start and end of sfc_i. Dummy VNF g_0 is placed on node s_i while dummy VNF g_1 is place on node d_i. Therefore, it is guaranteed that the data flow for request r_i always starts from node s_i and ends at node d_i. It is worth mentioning that dummy VNFs consume neither computing resource nor processing time (i.e., for a dummy VNF $g \in \mathbb{D}$, $c_g = 0$ and $l_g^v = 0$, where c_g and l_g^v are the computing resource cost and processing time per unit data traffic of dummy VNF g on node v). Thus, for any network node $v \in V_A$ that consists of only a single AP, only dummy VNFs can be implemented on it.

Denote by S_i the extended SFC of sfc_i including both g_0 and g_1, and the length of S_i is $(|sfc_i| + 2)$. Let $\Gamma = \mathbb{D} \setminus \{g_0, g_1\}$.

5.2.4 Problem definition

Given an MEC network $G = (V, E)$, a set F of VNFs, a set R of user requests, the admission of each request $r_i \in R$ suffers uncertainties of the amounts of demanded computing resource c_f of VNF f in its SFC and demanded data rate B_i, *the Robust Service Function Chain Placement (RSFCP) problem* is to maximize the expected profit collected of the network service provider, by admitting as many requests in R as possible while meeting the latency requirement of each admitted request, subject to both computing and bandwidth resource capacities in G, where the expected profit will be defined in Section 5.3.

5.3 QIP Formulation

In this section, we first formulate the RSFCP problem as a Quadratic Integer Programming (QIP), and we then show that the problem is NP-hard.

5.3.1 Traffic flow of requests

We deal with VNF placements of SFCs of multiple requests. For a single request $r_i \in R$, we introduce a binary decision variable x_i indicating whether request $r_i \in R$ is admitted ($x_i = 1$) or rejected ($x_i = 0$) by the system.

Recall that S_i is the extended SFC of sfc_i including both g_0 and g_1. To satisfy the network function dependence in S_i of r_i, we divide the SFC into a set of two-VNF sub-chains and then consider these sub-chains separately pair by pair. The core idea behind is to find a routing path between every two adjacent VNFs in S_i first, $\forall r_i \in R$.

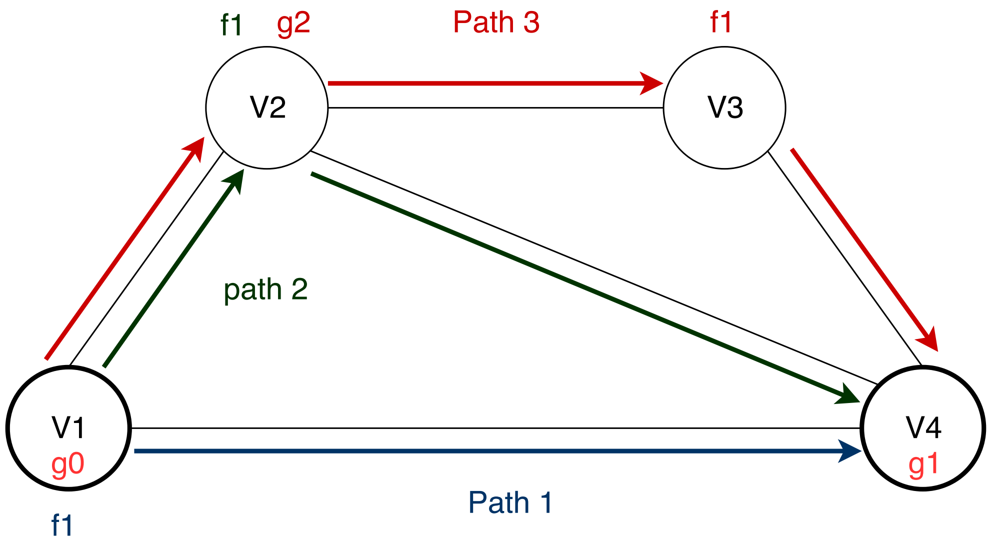


Figure 5.1: An example of a routing path of $g_0 \to f_1 \to g_1$.

Then, the selected routing paths between the VNF instance pairs are concatenated to form an ordered chain. Denote by h_i^k the kth VNF of S_i, e.g., $h_i^1 = g_0$.

We introduce a binary variable $\rho_{i,k}^{v,a,u,b}$ denoting that when constructing the routing path between the kth two-VNF sub-chain $h_i^k \to h_i^{k+1}$, whether the data traffic of request r_i traverses from VNF a on node v to VNF b on node u. Thus, the routing subpaths for r_i are expressed by the values of $\{\rho_{i,k}^{v,a,u,b} \mid \forall r_i \in R, \forall k \in \{1, \ldots, |S_i| - 1\}, \forall v \in V, \forall u \in N^+(v), \forall a \in \{h_i^k\} \cup \Gamma, \forall b \in \{h_i^{k+1}\} \cup \Gamma, a \neq b\}$. For example, as shown in Figure 5.1, for the requested SFC, $g_0 \to f_1 \to g_1$, dummy VNFs g_0 and g_1 are first deployed on source node v_1 and destination node v_4, respectively. The SFC $g_0 \to f_1 \to g_1$ is then considered as two two-VNF sub-chains: $g_0 \to f_1$ and $f_1 \to g_1$. Once the routing paths for two-VNF sub-chains $g_0 \to f_1$ and $f_1 \to g_1$ are decided, they are concatenated together to map the SFC $g_0 \to f_1 \to g_1$ into the network. Meanwhile, several dummy VNFs in Γ may have been inserted to construct the routing path to ensure that when a routing path passes through a node $v \in V$, at least one VNF $f \in S_i \cup \Gamma$ is deployed on it. Figure 5.1 shows the three routing paths.

As multiple VNFs of an SFC could be mapped to a single node, the data traffic of a request can traverse between the VNF instances in the same node multiple times. As shown in Figure 5.1, in *path* 1, both VNF g_0 and f_1 are placed on node v_1, which is represented by setting $\rho_{i,1}^{v_1,g_0,v_1,f_1} = 1$. Also, in Figure 5.1, inserting one dummy VNF g_2 on node v_2 is enough in describing *path* 3, it is not allowed to place multiple dummy VNFs on the same node for the same k, i.e., when considering a two-VNF sub-chain, we have

$$\sum_{a,b \in \mathbb{D}, a \neq b} \rho_{i,k}^{v,a,v,b} = 0, \forall r_i \in R, \forall v \in V, \forall k \in \{1, \ldots, |S_i| - 1\}. \tag{5.1}$$

When we insert dummy VNFs, the routing path of the kth two-VNF sub-chain in S_i goes through at most all nodes. Thus, we let $|\Gamma| = |V|$. Dummy VNF g_0 is placed on node s_i when constructing the routing path for the first two-VNF sub-chain. While dummy VNF g_1 is placed on node d_i when constructing the routing path for the last two-VNF sub-chain. We then have

$$\sum_{u \in N^+(s_i), b \in \{h_i^2\} \cup \Gamma} \rho_{i,1}^{s_i, g_0, u, b} = 1, \tag{5.2}$$

$$\sum_{v \in N^+(d_i), a \in \{h_i^{|S_i|-1}\} \cup \Gamma} \rho_{i,|S_i|-1}^{v, a, d_i, g_1} = 1. \tag{5.3}$$

The construction of a routing path for the kth two-VNF sub-chain in S_i, i.e., $h_i^k \to h_i^{k+1}, \forall r_i \in R, \forall k \in \{1, \dots, |S_i| - 1\}$ is as follows.

The source and destination VNFs of the kth routing path are h_i^k and h_i^{k+1}. Then, the outgoing flow from h_i^k and incoming flow to h_i^{k+1} must be 1, which is represented by the following equations.

$$\sum_{\substack{v \in V, u \in N^+(v),\\ b \in \{h_i^{k+1}\} \cup \Gamma}} \rho_{i,k}^{v, h_i^k, u, b} = 1, \forall r_i \in R, \forall k \in \{1, \dots, |S_i| - 1\}. \tag{5.4}$$

$$\sum_{\substack{v \in V, u \in N^+(v),\\ a \in \{h_i^k\} \cup \Gamma}} \rho_{i,k}^{v, a, u, h_i^{k+1}} = 1, \forall r_i \in R, \forall k \in \{1, \dots, |S_i| - 1\}, \tag{5.5}$$

And for any inserted dummy VNF $g \in \Gamma$ in the kth two-VNF sub-chain, the incoming flow at VNF g equals the outgoing flow from it, and each dummy VNF $g \in \Gamma$ appears in the path at most once, i.e.,

$$\sum_{\substack{u \in N^+(v),\\ b \in \{h_i^{k+1}\} \cup \Gamma, a \neq b}} \rho_{i,k}^{v,a,u,b} - \sum_{\substack{u \in N(v),\\ b \in \{h_i^k\} \cup \Gamma, a \neq b}} \rho_{i,k}^{u,b,v,a} = 0,$$
$$\forall r_i \in R, \forall k \in \{1, \dots, |S_i| - 1\}, \forall v \in V, \forall a \in \Gamma. \tag{5.6}$$

$$\sum_{\substack{v \in V, u \in N^+(v),\\ b \in \{h_i^{k+1}\} \cup \Gamma, a \neq b}} \rho_{i,k}^{v,a,u,b} \leq 1, \forall r_i \in R, \forall k \in \{1, \dots, |S_i| - 1\},$$
$$\forall a \in \Gamma. \tag{5.7}$$

Let $y_{i,k}^v$ be a binary variable indicating whether VNF $h_i^k \in S_i$ is placed in cloudlet node v. As the first and last VNFs in S_i are dummy VNFs, we only consider the other VNFs in S_i,

$$y_{i,k}^v = \sum_{\substack{u \in N^+(v),\\ b \in \{h_i^{k+1}\} \cup \Gamma}} \rho_{i,k}^{v, h_i^k, u, b}, \forall r_i \in R, \forall v \in V, k \in [2, |S_i| - 1]. \tag{5.8}$$

Recall that the upper bound of computing resource consumption of VNF h_i^k is $c^U_{h_i^k}$. To ensure that no resource capacity is violated, the computing capacity constraint on each node v is expressed as follows.

$$\sum_{r_i \in R, k \in \{2,\ldots,|S_i|-1\}} x_i \cdot y^v_{i,k} \cdot c^U_{h_i^k} \leq C_v, \forall v \in V. \tag{5.9}$$

The routing path for request r_i may pass through a link multiple times. Let $z_i^{v,u}$ be an integer variable indicating the number of times link (v,u) is contained in the routing path for request r_i. Then, we have

$$\begin{aligned} z_i^{v,u} = & \sum_{\substack{k \in [1,|S_i|-1] \\ \forall a \in \{h_i^k\} \cup \Gamma, \forall b \in \{h_i^{k+1}\} \cup \Gamma}} (\rho_{i,k}^{v,a,u,b} + \rho_{i,k}^{u,a,v,b}), \\ & \forall r_i \in R, \forall e(v,u) \in E. \end{aligned} \tag{5.10}$$

Recall that the upper bound of demanded data rate of request r_i is B_i^U. To ensure that no resource capacity is violated, the link capacity constraint on each link (v,u) for r_i can be expressed as follows.

$$\sum_{r_i \in R} x_i \cdot z_i^{v,u} \cdot B_i^U \leq c(v,u), \forall e(v,u) \in E. \tag{5.11}$$

Note that the latency of request r_i consists of the processing times on nodes and the communication latency along links in the routing path [70]. To admit request r_i, we need to ensure that its latency requirement must be met. With the upper bound of demanded data rate B_i^U of request r_i, the latency constraint of request r_i can be expressed as follows.

$$\begin{aligned} & \sum_{\substack{v \in V, \\ k \in \{1,\ldots,|S_i|-1\}}} x_i \cdot y^v_{i,k} \cdot B_i^U \cdot l^v_{h_i^k} \\ & + \sum_{e(v,u) \in E} x_i \cdot z_i^{v,u} \cdot B_i^U \cdot l(v,u) \leq \mathcal{L}_i, \forall r_i \in R. \end{aligned} \tag{5.12}$$

5.3.2 Admission cost of requests

Recall that the problem objective is to maximize the expected profit collected by the network service provider, which is equivalent to minimizing the accumulative expected admission cost of all admitted requests, where the expected admission cost of a request is the sum of the expected resource usage cost on both computing and bandwidth resource consumptions for implementing the request, which is defined as follows.

Considering the computing resource consumption for placing VNF instances for SFCs of requests in nodes, the computing resource usage cost at each node v, denoted by $\mathcal{E}_v$, is

$$\mathcal{E}_v = \varphi_v \cdot \sum_{r_i \in R, k \in \{2,\ldots,|S_i|-1\}} x_i \cdot y_{i,k}^v \cdot c_{h_i^k}, \tag{5.13}$$

where φ_v is the cost of a unit computing resource on node v with $\varphi_v > 0$. However, $\mathcal{E}_v$ is uncertain due to the uncertainty of computing resource consumption. The expected computing resource usage cost $\mathbb{E}(\mathcal{E}_v)$ of $\mathcal{E}_v$ at node v then is

$$\mathbb{E}(\mathcal{E}_v) = \varphi_v \cdot \sum_{r_i \in R, k \in \{2,\ldots,|S_i|-1\}} x_i \cdot y_{i,k}^v \cdot \overline{c_{h_i^k}}. \tag{5.14}$$

Considering the demanded data rate in links for the data traffic of request r_i, the bandwidth resource usage cost on link (v, u), denoted by $\mathcal{E}_{v,u}$, is

$$\mathcal{E}_{v,u} = \phi_{(v,u)} \cdot \sum_{r_i \in R} x_i \cdot z_i^{v,u} \cdot B_i, \tag{5.15}$$

where $\phi_{(v,u)} > 0$ is the cost of a unit bandwidth resource on link (v, u). However, the value of $\mathcal{E}_{v,u}$ is also uncertain due to the uncertainty of demanded data rates of requests. The expected bandwidth resource usage cost $\mathbb{E}(\mathcal{E}_{v,u})$ of $\mathcal{E}_{v,u}$ on link (u, v) then is

$$\mathbb{E}(\mathcal{E}_{v,u}) = \phi_{(v,u)} \cdot \sum_{r_i \in R} x_i \cdot z_i^{v,u} \cdot \overline{B_i}, \tag{5.16}$$

The total admission cost $\mathcal{E}$ of all admitted requests in R thus is the sum of the resource usage costs on both computing and bandwidth resource consumptions, which is defined as follows.

$$\mathcal{E} = \sum_{v \in V} \mathcal{E}_v + \sum_{e(v,u) \in E} \mathcal{E}_{v,u}. \tag{5.17}$$

The expectation $\mathbb{E}(\mathcal{E})$ of $\mathcal{E}$ is as follows.

$$\mathbb{E}(\mathcal{E}) = \sum_{v \in V} \mathbb{E}(\mathcal{E}_v) + \sum_{e(v,u) \in E} \mathbb{E}(\mathcal{E}_{v,u}). \tag{5.18}$$

The total admission cost has the uncertainty due to unknown demanded computing resource and data rates for the request implementation. In spite of this uncertainty, statistical information provided by experimental studies can be exploited to limit the risk of cost fluctuation. Reducing cost fluctuation is necessary to maintain a desired system performance for different realizations of uncertainty [26].

To model the cost fluctuation, we consider the effect of $\Delta B_i = B_i^U - B_i^L$ and $\Delta c_f = c_f^U - c_f^L$ on the cost variation of $\mathcal{E}$, denoted by $\Delta\mathcal{E}$. Since $\mathcal{E}$ is a linear function of both B_i and c_f, the relationship among $\Delta\mathcal{E}$, ΔB_i and Δc_f is given as follows.

$$\begin{aligned}\Delta\mathcal{E} = \sum_{v\in V} \varphi_v \cdot \sum_{r_i\in R, k\in\{2,\ldots,|S_i|-1\}} x_i \cdot y^v_{i,k} \cdot \Delta c_{h^k_i} \\ + \sum_{e(v,u)\in E} \phi_{(v,u)} \cdot \sum_{r_i\in R} x_i \cdot z^{v,u}_i \cdot \Delta B_i.\end{aligned} \tag{5.19}$$

5.3.3 QIP Formulation

The RSFCP problem can be formulated as a Quadratic Integer Programming (QIP) for a set R of requests with the optimization objective to

$$maximize \quad \sum_{r_i\in R} x_i \cdot pay_i - \mathbb{E}(\mathcal{E}) - \zeta \cdot \Delta\mathcal{E}, \tag{5.20}$$

subject to:

$$\begin{aligned}&(5.1),(5.2),(5.3),(5.4),(5.5),(5.6),(5.7),(5.9),(5.11),(5.12),\\ &x_i, \rho^{v,a,u,b}_{i,k} \in \{0,1\}, \forall r_i \in R, \forall k \in \{1,\ldots,|S_i|-1\},\\ &\forall v,u \in V, \forall a,b \in S_i \cap \Gamma,\end{aligned} \tag{5.21}$$

where variable x_i is a binary decision variable indicating whether request $r_i \in R$ is admitted ($x_i = 1$) or rejected ($x_i = 0$) by the system. $\rho^{v,a,u,b}_{i,k}$ is a binary decision variable denoting that when constructing the routing path between the kth two-VNF sub-chain $h^k_i \to h^{k+1}_i$, whether the data traffic of request r_i traverses from VNF a on node v to VNF b on node u. pay_i is the payment by request r_i if it is admitted and all its constraints are met, and ζ is the adjustable control parameter of cost variation compared with the expected total admission cost to stabilize the total admission cost [26].

5.3.4 NP-hardness of the problem

Theorem 5.1. *The RSFCP problem in an MEC $G(V,E)$ is NP-hard.*

Proof. We prove the NP-hardness of the RSFCP problem through reducing from the well-known knapsack problem.

The knapsack problem is NP-hard [79] and is defined as follows. Given a set of items $\mathcal{N}$, each item $i \in \mathcal{N}$ has a weight ω_i and a profit $p_i > 0$, $\forall i \in \mathcal{N}$. There is a bin with a capacity of $\mathcal{W}$. If item i can be placed in the bin without capacity violation, the associated profit p_i will be collected. The problem is to maximize the profit collected by packing as many items in $\mathcal{N}$ as possible, subject to the bin capacity.

Given an instance of the knapsack problem, we generate an instance of the RS-FCP problem as follows.

We consider a special case of the RSFCP problem and assume that there is a single cloudlet in the MEC network with a capacity of $\mathcal{W}$, we also ignore the bandwidth and latency constraints, and we assume that there exists no resource demand uncertainty.

We then generate a set of requests R and further assume that the processing time of a VNF is negligible. For each item $i \in \mathcal{N}$, there is a corresponding request, which has the amount ω_i of computing resource consumed by the requested SFC. The expected profit p_i of request r_i is calculated by its revenue and admission cost. If a request $r_i \in R$ is admitted, the requested SFC is placed in the cloudlet and the profit p_i is collected. The RSFCP problem is to maximize the expected profit collected by the network service provider by admitting as many requests as possible, subject to computing resource capacity.

It can be seen that a solution to the RSFCP problem returns a solution to the knapsack problem. And it takes polynomial time to reduce an instance of the knapsack problem to an instance of the RSFCP problem. Due to the NP-hardness of the knapsack problem [79], the RSFCP problem is NP-hard, too. $\square$

5.4 A Markov Based Approximation Algorithm

As the RSFCP problem is NP-hard, in this section we devise a near-optimal approximation algorithm for the problem, by adopting the Markov approximation technique [17]. Specifically, we first introduce the log-sum-exp approximation concept, and then construct a time-reversible Markov chain with designed transition rates satisfying the detailed balance equation [37].

5.4.1 Log-Sum-Exp approximation

Recall that there are two binary decision variables in the RSFCP problem, x_i (i.e., acceptance decision) and $\rho_{i,k}^{v,a,u,b}$ (i.e., routing path decision). We then define a scheduling η_i for request r_i that consists of determining the values of x_i and $\rho_{i,k}^{v,a,u,b}, \forall r_i \in R, \forall k \in \{1, \ldots, |S_i| - 1\}, \forall v \in V, \forall u \in N^+(v), \forall a \in \{h_i^k\} \cup \Gamma, \forall b \in \{h_i^{k+1}\} \cup \Gamma, a \neq b$.

To this end, we combine all $\eta_i, \forall r_i \in R$ to form a state, denoted by w, following resource capacity constraints in the MEC network and the latency constraints of requests in R.

Denote by W the set of all states, i.e., the set of all feasible solutions to the RSFCP problem. Let Λ_w be the value achieved by the optimization objective (5.20) under a state w (i.e., the expected profit collected from all admitted requests under the state w). Because Λ_w could be negative (i.e, deficit may exist), we let $\lambda_w = \Lambda_w + \tau$, $\forall w \in W$ where τ is a large positive constant to guarantee $\lambda_w \geq 0$. The problem then is reformulated as *the Maximum Weighted Combinatorial Optimization* (MWCO) problem as follows.

$$Maximize \quad \{\lambda_w \mid w \in W\}. \tag{5.22}$$

It can be seen that the optimization objective (5.22) has the same optimal value as the following problem.

$$\textit{Maximize} \quad \sum_{w \in W} p_w \cdot \lambda_w, \tag{5.23}$$

subject to

$$\sum_{w \in W} p_w = 1, \tag{5.24}$$

where p_w is a real value within the range of $[0, 1]$, which is the fraction of horizontal time (assuming the entire time horizon is 1) that the system stays in state w.

Property 5.1. *[17] Given a positive constant $\beta > 0$ and a set of non-negative real variables $\{\gamma_1, \gamma_2, \ldots, \gamma_n\}$, we have*

$$\max_{i \in [1,n]} \gamma_i \leq \frac{1}{\beta} \ln\Big(\sum_{i \in [1,n]} exp(\beta \cdot \gamma_i)\Big) \leq \max_{i \in [1,n]} \gamma_i + \frac{\ln n}{\beta}. \tag{5.25}$$

Following Proposition 5.1, when β approaches infinity, $\frac{\ln n}{\beta} = 0$, we then have

$$\max_{i \in [1,n]} \gamma_i = \lim_{\beta \to \infty} \frac{1}{\beta} \ln\Big(\sum_{i \in [1,n]} exp(\beta \cdot \gamma_i)\Big). \tag{5.26}$$

As λ_w is non-negative for any $w \in W$, we have

$$\max_{w \in W} \lambda_w \approx \frac{1}{\beta} \ln\Big(\sum_{w \in W} exp(\beta \cdot \lambda_w)\Big), \text{ for a very large } \beta. \tag{5.27}$$

Definition 3. [10] Let a be a real vector and b be the dual of a, for $\delta : \mathbb{R}^n \mapsto \mathbb{R}$, then the conjugate function of δ, $\delta^* : \mathbb{R}^n \mapsto \mathbb{R}$ is defined in terms of the supremum as

$$\delta^*(b) = \sup(b^T a - \delta(a)). \tag{5.28}$$

Property 5.2. *[10] The conjugate function of the conjugate function of a convex function is the function itself.*

Lemma 16. When β approaches infinity, the objective function (5.22) has the same value as the value of the following objective function.

$$\textit{Maximize} \sum_{w \in W} p_w \cdot \lambda_w - \frac{1}{\beta} \cdot \sum_{w \in W} p_w \cdot \ln p_w, \tag{5.29}$$

subject to

$$\sum_{w \in W} p_w = 1. \tag{5.30}$$

Proof. Denote by

$$\delta(\boldsymbol{\lambda}) = \frac{1}{\beta}\ln(\sum_{w\in W} exp(\beta \cdot \lambda_w)), \tag{5.31}$$

where $\boldsymbol{\lambda} = [\lambda_w \mid w \in W]$ is a vector of λ_w under each state, i.e., a vector of the expected profit collected, added by τ at each state.

Following Definition 3, the conjugate function of $\delta(\boldsymbol{\lambda})$ is

$$\delta^*(\boldsymbol{P}) = \begin{cases} \frac{1}{\beta} \cdot \sum_{w\in W} p_w \cdot \ln(p_w), & if\ \forall w \in W, p_w \geq 0, \\ & and\ \sum_{w\in W} p_w = 1. \\ \infty & otherwise, \end{cases} \tag{5.32}$$

where $\boldsymbol{P} = [p_w | w \in W]$ is a vector of the time fraction assigned to each state.

As log-sum-exp functions are both closed and convex, following Proposition 5.2, the conjugate of $\delta^*(\boldsymbol{P})$ is $\delta(\boldsymbol{\lambda})$.

Following Definition 3, we have

$$\delta(\boldsymbol{\lambda}) = \sup \sum_{w\in W} p_w \cdot \lambda_w - \frac{1}{\beta} \cdot \sum_{w\in W} p_w \cdot \ln p_w. \tag{5.33}$$

Combining (5.27), (5.31), and (5.33), the objective function (5.22) has the same value as the value of the objective function (5.29) when β approaches infinity. The lemma then follows. □

Following Lemma 16, the approximate value of the objective function of (5.22) can be obtained, by solving (5.29).

Let μ be the Lagrangian multiplier associated with Equation (5.30). Let p_w^* be the optimal solution to (5.29). It can be seen that the Karush-Kuhn-Tucker (KKT) conditions for (5.29) are met. We then have

$$\begin{cases} \lambda_w - \frac{1}{\beta} \cdot \ln(p_w^*) - \frac{1}{\beta} + \mu = 0, \forall w \in W, \\ \sum_{w\in W} p_w^* = 1, \\ \mu \geq 0. \end{cases} \tag{5.34}$$

The value of p_w^* thus is

$$p_w^* = \frac{exp(\beta \cdot \lambda_w)}{\sum_{w'\in W} exp(\beta \cdot \lambda_{w'})}, \ \forall w \in W. \tag{5.35}$$

5.4.2 Markov chain model design

Let p_w^* be the stationary distribution of the designed Markov chain model that is trained to demonstrate the convergence to the specific stationary distribution set in advance, i.e., p^*_w. The authors in [17] showed that there exists at least one time-reversible Markov chain model in which the stationary distribution is p^*_w. Let $q_{w,w'}$ be a non-negative transition rate between two states w and w' with $w \neq w'$ $\forall w, w' \in W$. To construct a time-reversible Markov chain [17], the following two conditions must be met: (1) each state could be transited to any other state; and (2) the detailed balance equation between states w and w' should be satisfied.

$$p^*_w \cdot q_{w,w'} = p^*_{w'} \cdot q_{w',w}, \forall w, w' \in W, w \neq w'. \tag{5.36}$$

Considering (5.35) and (5.36), we then design the transition rate $q_{w,w'}$ $\forall w, w' \in W, w \neq w'$ as follows.

$$\begin{cases} q_{w,w'} = \dfrac{exp(\beta \cdot \lambda_{w'})}{|R| \cdot \gamma \cdot \max\{exp(\beta \cdot \lambda_w), exp(\beta \cdot \lambda_{w'})\}}, \\ q_{w',w} = \dfrac{exp(\beta \cdot \lambda_{w})}{|R| \cdot \gamma \cdot \max\{exp(\beta \cdot \lambda_w), exp(\beta \cdot \lambda_{w'})\}}, \end{cases} \tag{5.37}$$

where R is the set of user requests and γ is a positive constant which is set as the mean time to transit from current state to another state.

5.4.3 Markov Based Approximation Algorithm

The implementation details of the Markov based approximation algorithm are given in `Algorithm` 11. In particular, the system controller creates a thread for each request in parallel, i.e., each thread deals with one request with the aim of the expected profit optimization. Initially, the dedicated thread for each request r_i randomly makes the acceptance decision (i.e., x_i) and the routing path decision (i.e., $\rho^{v,a,u,b}_{i,k}$) to choose a feasible scheduling for itself. The system controller finally combines the scheduling of each request to form the current state w, and calculates the expected profit Λ_w in state w. Recall that $\lambda_w = \Lambda_w + \tau$, where τ is a large positive constant to guarantee $\lambda_w \geq 0$.

For each request $r_i \in R$, the associated thread randomly looks for a new feasible scheduling for itself based on current residual resources in the network to form the next state w' in a parallel manner, where w' is the next state from the current state w through one transition. λ_w and $\lambda_{w'}$ under w and w' then are calculated. Then a random exponentially timer ψ_i for request $r_i \in R$ is triggered with a mean γ, where γ is a positive constant, and the timer ψ_i starts to count down. Let $p_{w,w'}$ be the probability that the system transits from state w to state w'. When the timer ψ_i for request $r_i \in R$ expires, the thread of r_i transits to state w' with probability of $p_{w,w'} = \frac{exp(\beta \cdot \lambda_{w'})}{\max\{exp(\beta \cdot \lambda_w), exp(\beta \cdot \lambda_{w'})\}}$, and broadcasts a Reset signal to the rest threads. For the rest threads, upon receiving the Reset signal, their timers are terminated. Then each thread of requests looks for a new feasible scheduling for itself in parallel based on residual resources in the network and the above steps are repeated until it converges.

Algorithm 11 A Markov Based Approximation Algorithm

```
Require: An MEC network $G = (V, E)$ and a set of requests $R$.
Ensure: A scheduling of requests in $R$, s.t., the collected profit is maximized.
 1: procedure INITIALIZATION
 2:     for $r_i \in R$ do
 3:         Initialize a thread for $r_i$;
 4:         The thread computes a feasible scheduling for $r_i$;
 5:     end for
 6:     The scheduling for each request $r_i$ forms the current state $w$, and the expected
        profit $\Lambda_w$ collected from state $w$ is calculated.
 7:     Then let $\lambda_w = \Lambda_w + \tau$, where $\tau$ is a large constant to guarantee $\lambda_w \geq 0$;
 8:     Execute TRANSIT($r_i$), $\forall r_i \in R$.
 9: end procedure
10: procedure SET TIMER($r_i$)
11:     The thread of $r_i$ creates a random exponentially timer $\psi_i$ with the mean value
        $\gamma$;
12:     The timer $\psi_i$ starts to count down.
13: end procedure
14: procedure TRANSIT($r_i$)
15:     while It has not converged do
16:         The thread of $r_i$ randomly chooses a new feasible scheduling to form next
            state $w'$;
17:         Execute SET TIMER($r_i$).
18:         if $\psi_i$ expires then
19:             the thread of $r_i$ transits to state $w'$ with probability of $p_{w,w'} =
                \frac{exp(\beta \cdot \lambda_{w'})}{\max\{exp(\beta \cdot \lambda_w), exp(\beta \cdot \lambda_{w'})\}}$;
20:             Broadcast RESET signals to other threads.
21:             Execute TRANSIT($r_i$).
22:         else if $\psi_i$ does not expire and the thread of $r_i$ receives a RESET signal then
23:             Terminate current timer.
24:             Execute TRANSIT($r_i$).
25:         end if
26:     end while
27: end procedure
```

5.4.4 Algorithm analysis

In the following, we analyze the performance of the proposed algorithm. We also study the impact of perturbations on the performance of the proposed algorithm.

Lemma 17. Given an MEC network and a set of requests R, there is a Markov based approximation algorithm, `Algorithm` 11, which constructs a time-reversible Markov chain, and the stationary distribution is Equation (5.35).

Proof. As mentioned, we design a Markov chain model and guarantee that in the constructed Markov chain model, each state can be transited to any another state.

We then show that the stationary distribution of the designed Markov chain model is Eq. (5.35). From `Algorithm` 11, it can be seen that we randomly select a state $w' \in W$ as the next state. And we generate a random exponentially timer ψ_i with mean value equal to γ for each request r_i. Furthermore, when the timer expires, the thread of r_i transits from current state w to next state w' with the probability of $p_{w,w'} = \frac{exp(\beta \cdot \lambda_{w'})}{\max\{exp(\beta \cdot \lambda_w), exp(\beta \cdot \lambda_{w'})\}}$. Thus, we are able to calculate the transition rate as follows,

$$q_{w,w'} = \frac{p_{w,w'}}{|R| \cdot \gamma} = \frac{exp(\beta \cdot \lambda_{w'})}{|R| \cdot \gamma \cdot \max\{exp(\beta \cdot \lambda_w), exp(\beta \cdot \lambda_{w'})\}}, \tag{5.38}$$

which is mentioned in Eq. (5.37).

Combining (5.35) and (5.37), it can be seen that $p^*_w \cdot q_{w,w'} = p^*_{w'} \cdot q_{w',w}, \forall w, w' \in W, w \neq w'$. The detailed balance equation of the designed Markov chain model is satisfied, and the designed Markov chain model is time-reversible, and the stationary distribution of which is Equation (5.35) [46]. Hence, the theorem follows. □

Ideally, if the exact value of λ_w of each state $w, \forall w \in W$ can be obtained (i.e., the expected profit collected from each state can be accurately measured), then the designed Markov chain model will always converge to the preset stationary distribution p^*_w. However, the value of λ_w is very likely to be perturbed, and the designed Markov chain model might not be able to achieve the global optimality, instead a sub-optimal stationary distribution will be obtained [86].

In this chapter, we assume the measurement perturbations are caused by the uncertainty of resource demands in the execution of requests, and the expected amounts of resources may experience perturbations as mentioned in Section 5.2.2. In the sequel, it is necessary to analyze the impact of measurement perturbations on the solution and to mitigate the impact of such perturbations on the performance of the proposed approximation algorithm.

We quantify the perturbation error under a state $w \in W$ as a value ranged from $-\theta_w$ to θ_w, by introducing θ_w as a perturbation error bound, and λ_w is drawn from $2 \cdot n_w + 1$ discrete values: $\lambda_w - \theta_w, \ldots, \lambda_w - (1/n_w)\theta_w, \lambda_w, \lambda_w + (1/n_w)\theta_w, \ldots, \lambda_w + \theta_w$, where n_w is a positive constant.

Denote by $\alpha(w, j)$ the probability of the perturbed λ_w taking the value $\lambda_w + (j/n_w) \cdot \theta_w, \forall w \in W, \forall j = \{-n_w, \ldots, n_w\}$, and $\sum_{j \in \{-n_w, \ldots, n_w\}} \alpha(w, j) = 1$.

Lemma 18. Given an MEC network $G(V, E)$, with measurement perturbations, the stationary distribution of the Markov chain model, denoted by $\overline{p}_w$, is

$$\overline{p}_w = \frac{\kappa_w \cdot exp(\beta \cdot \lambda_w)}{\sum_{w' \in W} \kappa_{w'} \cdot exp(\beta \cdot \lambda_{w'})}, \forall w \in W, \tag{5.39}$$

where $\kappa_w = \sum_{j \in \{-n_w, \ldots, n_w\}} \alpha(w, j) \cdot exp(\beta \cdot ((j \cdot \theta_w)/n_w))$.

Proof. The core idea of the proof is to first deliver the modified transition rate with perturbations by treating each state w as $2 \cdot n_w + 1$ states. With (5.35) and the detailed balance equations, the stationary distribution of the Markov chain model with perturbations can be obtained then.

In the case of perturbation, each state w are then treated as $2 \cdot n_w + 1$ states (i.e., w_j, $\forall j = \{-n_w, \ldots, n_w\}$) with perturbation error bound θ_w. Denote by $\lambda_{w,j}$, the expected profit collected in state w_j added by τ, we have

$$\lambda_{w,j} = \lambda_w + (j/n_w) \cdot \theta_w, \forall w \in W, \forall j \in \{-n_w, \ldots, n_w\}. \tag{5.40}$$

The modified transition rate $\overline{q}_{w_j, w'_{j'}}$ with perturbations is

$$\overline{q}_{w_j, w'_{j'}} = \frac{\alpha(w', j') \cdot exp(\beta \cdot \lambda_{w', j'})}{|R| \cdot \gamma \cdot \max\{exp(\beta \cdot \lambda_{w', j'}), exp(\beta \cdot \lambda_{w,j})\}}, \tag{5.41}$$

With regard to the detailed balance equations $p_{w_j} \cdot \overline{q}_{w_j, w'_{j'}} = p_{w'_{j'}} \cdot \overline{q}_{w'_{j'}, w_j}$, we have

$$\frac{p_{w_0}}{\alpha(w,0) \cdot exp(\beta \cdot \lambda_{w_0})} = \frac{p_{w'_{j'}}}{\alpha(w', j') \cdot exp(\beta \cdot \lambda_{w'_{j'}})}, \tag{5.42}$$

where w_0 is state w with no perturbation and $\alpha(w, 0)$ is the probability that no perturbation exists for state w.

From (5.35) and (5.42), we have

$$p_{w_j} = \frac{\alpha(w,j) \cdot exp(\beta \cdot \lambda_{w_j})}{\sum_{w' \in W} \sum_{j' \in \{-n_w, \ldots, n_w\}} \cdot \alpha(w', j') \cdot exp(\beta \cdot \lambda_{w'_{j'}})},$$
$$\forall w \in W, \forall j \in \{-n_w, \ldots, n_w\}. \tag{5.43}$$

Denote by $\kappa_w = \sum_{j \in \{-n_w, \ldots, n_w\}} \alpha(w,j) \cdot exp(\beta \cdot ((j \cdot \theta_w)/n_w))$, we have

$$\begin{aligned}\overline{p}_w &= \sum_{j \in \{-n_w, \ldots, n_w\}} \cdot p_{w_j} \\ &= \frac{\sum_{j \in \{-n_w, \ldots, n_w\}} \alpha(w,j) \cdot exp(\beta \cdot \lambda_{w_j})}{\sum_{w' \in W} \sum_{j' \in \{-n_w, \ldots, n_w\}} \cdot \alpha(w', j') \cdot exp(\beta \cdot \lambda_{w'_{j'}})} \\ &= \frac{\kappa_w \cdot exp(\beta \cdot \lambda_w)}{\sum_{w' \in W} \kappa_{w'} \cdot exp(\beta \cdot \lambda_{w'})}, \text{by (5.40).}\end{aligned} \tag{5.44}$$

Thus, Lemma 18 follows. □

Theorem 5.2. *Given an MEC network $G(V, E)$ and a set of requests R, without analysis perturbation, there is an Markov based approximation algorithm,* `Algorithm` *11, its optimality gap is as follows.*

$$0 \le \Lambda_{max} - \Lambda_{avg} \le \frac{\ln |W|}{\beta}, \tag{5.45}$$

where the optimality gap of an algorithm is the absolute difference between the solution ob-

tained by the algorithm and the optimal solution of the problem. $\Lambda_{max} = \max_{w\in W}\{\Lambda_w\}$ *is the value of the optimal solution,* Λ_{avg} *is the expected profit with the designed Markov chain model and* W *is the collection of all states.*

Proof. We show the optimality gap as follows. Let w_{max} be the state to obtain the maximum profit. Let the time fraction distribution $\tilde{p}_w, \forall w \in W$ be the optimal solution. We then have

$$\tilde{p}_w = \begin{cases} 1, & if\ w = w_{max}, \\ 0, & otherwise. \end{cases} \tag{5.46}$$

Following Lemma 16, as p_w^* is the desired stationary distribution calculated in (5.35),, we have

$$\begin{aligned} &\sum_{w\in W} p_w^* \cdot \lambda_w - \frac{1}{\beta} \cdot \sum_{w\in W} p_w^* \cdot \ln p_w^* \\ \geq &\sum_{w\in W} \tilde{p}_w \cdot \lambda_w - \frac{1}{\beta} \cdot \sum_{w\in W} \tilde{p}_w \cdot \ln \tilde{p}_w = \lambda_{max}, \end{aligned} \tag{5.47}$$

where $\lambda_{max} = \max_{w\in W}\{\lambda_w\}$.

Apply Jensen's inequality [10], we have

$$\begin{aligned} &\sum_{w\in W} p_w^* \cdot \ln p_w^* = -\sum_{w\in W} p_w^* \cdot \ln \frac{1}{p_w^*} \\ \geq &-\ln(\sum_{w\in W} p_w^* \cdot \frac{1}{p_w^*}) = -\ln|W|. \end{aligned} \tag{5.48}$$

Combining equations (5.47) and (5.48), we have

$$\begin{aligned} \lambda_{avg} = &\sum_{w\in W} p_w^* \cdot \lambda_w \leq \sum_{w\in W} p_w^* \cdot \lambda_{max} = \lambda_{max} \\ \leq &\lambda_{avg} - \frac{1}{\beta} \cdot \sum_{w\in W} p_w^* \cdot \ln p_w^* \leq \lambda_{avg} + \frac{1}{\beta} \cdot \ln|W|, \end{aligned} \tag{5.49}$$

where λ_{avg} is the expected value with the designed Markov chain model.

Thus, we have

$$0 \leq \lambda_{max} - \lambda_{avg} \leq \frac{\ln|W|}{\beta}. \tag{5.50}$$

Because $\lambda_w = \Lambda_w + \tau, \forall w \in W$, we have,

$$0 \leq \Lambda_{max} - \Lambda_{avg} \leq \frac{\ln|W|}{\beta}. \tag{5.51}$$

The theorem then follows. □

Theorem 5.3. *Given an MEC network and a set of requests R, with analysis perturbation, there is an Markov based approximation algorithm,* `Algorithm` *11, its optimality gap is given as follows.*

$$0 \le \Lambda_{max} - \overline{\Lambda}_{avg} \le \frac{\ln |W|}{\beta} + \theta_{max}, \tag{5.52}$$

where $\overline{\Lambda}_{avg} = \sum_{w \in W} \overline{p}_w \cdot \Lambda_w$ is the expected profit with the perturbed Markov chain model, and the time fraction distribution $\overline{p}_w$, $\forall w \in W$, is the optimal solution under the perturbation case, and $\theta_{max} = \max_{w \in W}\{\theta_w\}$.

Proof. Recall that $\kappa_w = \sum_{j \in \{-n_w, \ldots, n_w\}} \alpha(w,j) \cdot exp(\beta \cdot ((j \cdot \theta_w)/n_w))$, $\forall w \in W$, we have

$$exp(-\beta \cdot \theta_w) \le \kappa_w \le exp(\beta \cdot \theta_w). \tag{5.53}$$

$$-\theta_w \le \frac{\ln \kappa_w}{\beta} \le \theta_w. \tag{5.54}$$

From Lemma 18, we have,

$$\overline{p}_w = \frac{exp(\beta \cdot (\lambda_w + \frac{\ln \kappa_w}{\beta})}{\sum_{w' \in W} exp(\beta \cdot (\lambda_{w'} + \frac{\ln \kappa_{w'}}{\beta})}, \forall w \in W. \tag{5.55}$$

As a result, for $\lambda_{w''} = \lambda_w + \frac{\ln \kappa_w}{\beta}$, the stationary distribution with perturbation $\overline{p}_w, \forall w \in W$, works as an optimal solution in this case.

From Theorem 5.2, we have,

$$\max_{w'' \in W} \lambda_{w''} - \sum_{w'' \in W} \overline{p}_{w''} \cdot \lambda_{w''} \le \frac{\ln |W|}{\beta}. \tag{5.56}$$

Let $\lambda_{w''} = \lambda_w + \frac{\ln \kappa_w}{\beta}$, we have,

$$\max_{w \in W}(\lambda_w + \frac{\ln \kappa_w}{\beta}) - \sum_{w \in W}(\overline{p}_w \cdot \lambda_w + \frac{\ln \kappa_w}{\beta}) \le \frac{\ln |W|}{\beta}. \tag{5.57}$$

Then,

$$\begin{aligned}\lambda_{max} &= \max_{w \in W} \lambda_w \le \max_{w \in W}(\lambda_w + \frac{\ln \kappa_w}{\beta}) \\ &\le \sum_{w \in W}(\overline{p}_w \cdot \lambda_w + \frac{\ln \kappa_w}{\beta}) + \frac{\ln |W|}{\beta}, \text{by (5.57)}, \\ &\le \sum_{w \in W} \overline{p}_w \cdot \lambda_w + \theta_w + \frac{\ln |W|}{\beta}, \text{by (5.54)}.\end{aligned} \tag{5.58}$$

Also,

$$\begin{aligned}\overline{\lambda}_{avg} &= \sum_{w\in W}\overline{p}_w\cdot\lambda_w \le \sum_{w\in W}\overline{p}_w\cdot\lambda_{max} = \lambda_{max}\\ &\le \overline{\lambda}_{avg} + \theta_w + \frac{\ln|W|}{\beta}, \text{by (5.58)},\end{aligned}$$

where $\overline{\lambda}_{avg}$ is the expected value with the perturbed Markov chain model.

We thus have

$$0 \le \lambda_{max} - \overline{\lambda}_{avg} \le \frac{\ln|W|}{\beta} + \theta_w \le \frac{\ln|W|}{\beta} + \theta_{max}.$$

Because $\lambda_w = \Lambda_w + \tau$, $\forall w \in W$, we have,

$$0 \le \Lambda_{max} - \overline{\Lambda}_{avg} \le \frac{\ln|W|}{\beta} + \theta_{max}.$$

Hence, the theorem follows. □

In the context of Markov Chain, the convergence time is examined by the mixing time [17].

Definition 4. [17] Let $H_t(w)$ be the probability distribution of all states in W at time t with the initial state w, $\boldsymbol{p}^*$ be the stationary distribution of the Markov Chain, and $\epsilon > 0$ be a constant and represent the gap between the converged solution and the optimal one, then the mixing time of the constructed Markov Chain is defined as

$$t_{mix}(\epsilon) := inf\{t \ge 0 : \max_{w\in W}||H_t(w) - \boldsymbol{p}^*||_{TV} \le \epsilon\}, \tag{5.59}$$

where term $||H_t(w) - \boldsymbol{p}^*||_{TV}$ is the total variance distance between $H_t(w)$ and $\boldsymbol{p}^*$.

Denote by $\Lambda_{min} = \min_{w\in W}\{\Lambda_w\}$ and recall that $\Lambda_{max} = \max_{w\in W}\{\Lambda_w\}$.

Theorem 5.4. *Given an MEC network $G(V,E)$ and a set R of requests, there is an Markov based approximation algorithm,* `Algorithm` *11, its convergence time is bounded as follows.*

$$t_{mix} \ge \frac{|R|\cdot\gamma\cdot exp(\beta\cdot(\Lambda_{min} - \Lambda_{max}))}{2\cdot|W|}\cdot\ln\frac{1}{2\cdot\epsilon},$$

and

$$\begin{aligned}t_{mix} \le& 2\cdot|W|^3\cdot|R|\cdot\gamma\cdot exp(5\cdot\beta\cdot(\Lambda_{max} - \Lambda_{min}))\\ &\cdot(\ln\frac{1}{2\cdot\epsilon} + \frac{1}{2}\cdot(\ln|W| + \beta\cdot(\Lambda_{max} - \Lambda_{min}))).\end{aligned}$$

Proof. By stationary distribution (5.35), the minimum probability among the stationary distribution, denoted by p_{min}, is:

$$\begin{aligned} p_{min} := \min_{w \in W} p^*_w \geq & \frac{exp(\beta \cdot \Lambda_{min})}{|W| \cdot exp(\beta \cdot \Lambda_{max})}, \text{by (5.35)}, \\ = & \frac{1}{|W|} \cdot exp(\beta(\Lambda_{min} - \Lambda_{max})). \end{aligned} \tag{5.60}$$

We then adopt the uniformization technique [17]. Denote by $\mathcal{Q} = \{q_{w,w'}\}$ the transition matrix of the constructed Markov Chain. Then a Markov Chain $Z(m)$ is constructed with a transition matrix $P = I + \frac{\mathcal{Q}}{\sigma}$, where I denotes a unit matrix and σ denotes the uniform rate parameter. We then assume that with the Markov Chain $Z(n)$, the system transits its state following the Poisson process $N(t)$ with rate σ [37]. Denote by $Z(N(t))$ the state of the system at time t.

From the transition rate (5.37), we have, $\forall w, w' \in W$,

$$q_{w,w'} \leq \frac{1}{|R| \cdot \gamma} \cdot exp(\beta \cdot (\Lambda_{max} - \Lambda_{min})). \tag{5.61}$$

And we have

$$\sum_{w \neq w'} q_{w,w'} \leq \frac{|W|}{|R| \cdot \gamma} exp(\beta \cdot (\Lambda_{max} - \Lambda_{min})). \tag{5.62}$$

Then we have σ as:

$$\sigma = \frac{|W|}{|R| \cdot \gamma} exp(\beta \cdot (\Lambda_{max} - \Lambda_{min})). \tag{5.63}$$

According to the uniformization theorem [47], the Markov Chain and its counterpart $Z(N(t))$ with discrete-time manner share the same probability distribution. Denote by $\rho2$ the second eigenvalue of transition matrix $\boldsymbol{P}$ for $Z(n)$. We then adopt the spectral gap inequality [47], we have,

$$\begin{aligned} \frac{exp(-\sigma \cdot (1 - \rho2) \cdot t)}{2} \leq & \max_{w \in W} ||H_t(w) - \boldsymbol{p}^*||_{TV} \\ \leq & \frac{exp(-\sigma \cdot (1 - \rho2) \cdot t)}{2 \cdot (p_{min})^{\frac{1}{2}}}. \end{aligned} \tag{5.64}$$

Therefore,

$$\begin{aligned} \frac{1}{\sigma \cdot (1 - \rho2)} \cdot \ln \frac{1}{2 \cdot \epsilon} \leq & t_{mix}(\epsilon) \\ \leq & \frac{1}{\sigma \cdot (1 - \rho2)} \cdot (\ln \frac{1}{2 \cdot \epsilon} + \frac{1}{2} \cdot \ln \frac{1}{p_{min}}). \end{aligned} \tag{5.65}$$

With Cheeger's inequality [25], we bound $\rho 2$ as follows:

$$1 - 2 \cdot \Phi \le \rho 2 \le 1 - \frac{1}{2} \cdot \Phi^2, \tag{5.66}$$

where Φ is the "conductance" of $\boldsymbol{P}$ and is defined as follows,

$$\Phi := \min_{N \subset W, \pi_N \in (0,0.5]} \frac{\mathbb{F}(N, N^c)}{\pi_N}, \tag{5.67}$$

where $\pi_N = \sum_{w \in N} p^*_w$ and $\mathbb{F}(N, N^c) = \sum_{w \in N, w' \in N^c} p^*_w \cdot P(w, w')$. With (5.65) and (5.66), we have,

$$\frac{1}{2 \cdot \sigma \cdot \Phi} \cdot \ln \frac{1}{2 \cdot \epsilon} \le t_{mix}(\epsilon) \le \frac{2}{\sigma \cdot \Phi^2} \cdot (\ln \frac{1}{2 \cdot \epsilon} + \frac{1}{2} \cdot \ln \frac{1}{p_{min}}). \tag{5.68}$$

Then, $\forall N' \subset W, \pi_{N'} \in (0, 0.5]$, we have

$$\begin{aligned}\Phi &:= \min_{N \subset W, \pi_N \in (0,0.5]} \frac{\mathbb{F}(N, N^c)}{\pi_N} \\ &\le \frac{1}{\pi_{N'}} \cdot \sum_{w \in N, w' \in N'^c} p^*_w \cdot P(w, w') \\ &= \frac{1}{\pi_{N'}} \cdot \sum_{w \in N'} \cdot p^*_w \cdot \sum_{w' \in N'^c} P(w, w') \le \frac{1}{\pi_{N'}} \cdot \sum_{w \in N'} \cdot p^*_w = 1\end{aligned} \tag{5.69}$$

From (5.63), (5.68) and (5.69), we have the lower bound of $t_{mix}(\epsilon)$ as follows,

$$\begin{aligned}t_{mix}(\epsilon) &\ge \frac{1}{2 \cdot \sigma} \cdot \ln \frac{1}{2 \cdot \epsilon} \\ &= \frac{|R| \cdot \gamma \cdot exp(\beta \cdot (\Lambda_{min} - \Lambda_{max}))}{2 \cdot |W|} \cdot \ln \frac{1}{2 \cdot \epsilon}.\end{aligned} \tag{5.70}$$

From the transition rate (5.37), we have, $\forall w, w' \in W$,

$$q_{w,w'} \ge \frac{1}{|R| \cdot \gamma} \cdot exp(\beta \cdot (\Lambda_{min} - \Lambda_{max})). \tag{5.71}$$

From (5.67), we have,

$$\begin{aligned}\Phi &\ge \min_{N \subset W, \pi_N \in (0,0.5]} \mathbb{F}(N, N^c) \ge \min_{w \ne w', p(w,w') > 0} \mathbb{F}(w, w') \\ &= \min_{w \ne w', p(w,w') > 0} p^*_w \cdot P(w, w') = \min_{w \ne w', p(w,w') > 0} p^*_w \cdot \frac{q_{w,w'}}{\sigma} \\ &\ge \frac{p_{min}}{\sigma} \cdot \frac{1}{|R| \cdot \gamma} \cdot exp(\beta \cdot (\Lambda_{min} - \Lambda_{max})), \text{by (5.71)}.\end{aligned} \tag{5.72}$$

From (5.68), we have the upper bound of $t_{mix}(\epsilon)$ as follows,

$$\begin{aligned}
t_{mix}(\epsilon) &\leq \frac{2}{\sigma \cdot \Phi^2} \cdot (\ln \frac{1}{2 \cdot \epsilon} + \frac{1}{2} \cdot \ln \frac{1}{p_{min}}) \\
&\leq \frac{2 \cdot |R|^2 \cdot \gamma^2 \cdot \sigma \cdot exp(2 \cdot \beta \cdot (\Lambda_{max} - \Lambda_{min}))}{p_{min}^2} \\
&\cdot (\ln \frac{1}{2 \cdot \epsilon} + \frac{1}{2} \cdot \ln \frac{1}{p_{min}}), \text{by (5.72)}
\end{aligned}$$

$$\begin{aligned}
&= \frac{2 \cdot |W| \cdot |R| \cdot \gamma \cdot exp(3 * \beta \cdot (\Lambda_{max} - \Lambda_{min}))}{p_{min}^2} \\
&\cdot (\ln \frac{1}{2 \cdot \epsilon} + \frac{1}{2} \cdot \ln \frac{1}{p_{min}}), \text{by (5.63)} \\
&\leq 2 \cdot |W|^3 \cdot |R| \cdot \gamma \cdot exp(5 \cdot \beta \cdot (\Lambda_{max} - \Lambda_{min})) \\
&\cdot (\ln \frac{1}{2 \cdot \epsilon} + \frac{1}{2} \cdot (\ln |W| + \beta \cdot (\Lambda_{max} - \Lambda_{min}))), \text{by (5.60).}
\end{aligned}$$

Hence, the theorem follows. □

5.5 Performance Evaluation

In this section, the performance of the proposed algorithm is evaluated by experimental simulations. The impact of parameters on the performance of the proposed algorithm is investigated, too.

5.5.1 Environment Settings

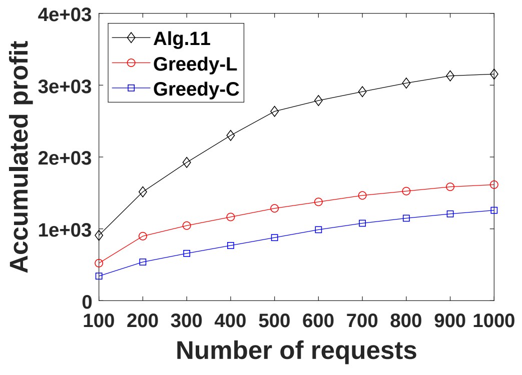
We generate topologies of MEC networks through a tool GT-ITM [34]. We consider an MEC network with 100 APs, and 10 percent of the APs are randomly selected to be co-located with cloudlets. The capacities of cloudlets are randomly drawn between 30,000MHz and 60,000MHz [29]. We further assume that there are 20 types of VNFs, and the expected computing resource demanded by each type of VNFs is ranged from 40MHz to 600MHz [9]. The length of a requested SFC is randomly chosen from 2 to 10 [82] and each VNF instance is randomly drawn from the provisioned VNFs. The bandwidth capacity of a link is randomly drawn from 2,000Mbps to 20,000Mbps [37]. The transmission latency of a link is randomly drawn from 2ms to 7ms [70]. The expected demanded data rate of each request varies from 1 to 10 packets per millisecond, and the size of each packet is 64KB [69]. The upper bound of the demanded computing resource and data rate are randomly set as 105%, 110% or 115% of the expected one, while the lower bound of those are randomly set as 95%, 90% or 85% of the expected one. Considering the perturbation, the actual consumed computing resource and data rate are randomly drawn between the upper bounds and lower bounds. The actual demanded resource is utilized to calculate the actual accumulated profit. The processing rate of each VNF instance varies from 5 to 20 packets per millisecond [69]. The latency requirement of each request is set from 20ms to 100ms randomly [70]. The computing resource cost on each cloudlet is randomly drawn from a range between \$0.05 to \$0.2 per MHz, while the bandwidth cost on each link varies from \$0.002 to \$0.005 per Mbps [107]. Each value in figures is the mean of the results of 20 topologies randomly generalized by GT-ITM [34] with the same size. The payment of each request is randomly drawn from \$10 to \$30. The parameter β in the designed Markov Chain is set as 1 and the control parameter ζ associated with the cost variation is set as 0.1 [26]. The running time of each algorithm is based on a desktop with a 3.60 GHz Intel 8-Core i7-7700 CPU and 16 GB RAM. Unless specified, the above parameters are adopted by default.

To evaluate the performance of `Algorithm` 11 (referred to as `Alg.11`) for the RS-FCP problem, We here introduce two benchmarks against the proposed algorithm. The first is a greedy algorithm considering the least latency on links for each request, referred to as `Greedy-L`. We consider the set R of requests randomly in sequence. For request $r_i \in R$, we first eliminate the nodes and links with insufficient resources for its admission through constructing an auxiliary graph. `Greedy-L` finds a feasible path with sufficient residual resource and the least latency on links from the source node to the destination in the auxiliary graph. Then, we find a feasible placement of the requested SFC on the chosen routing path, following the computing resource constraint and latency constraint. A request is rejected if `Greedy-L` cannot find a feasible placement for its SFC. The other benchmark is a greedy algorithm considering the least cost on the links for each request, referred to as `Greedy-C`. Similarly, for each request r_i, `Greedy-C` first finds a feasible routing path with the least cost on links from the source node to the destination node in the auxiliary graph with sufficient resource for its admission. Then, we find a feasible placement of the requested SFC on the chosen routing path, following the computing resource constraint and latency constraint. A request is rejected if `Greedy-C` cannot find a feasible placement for its SFC. The average result delivered by each algorithm is calculated based on 20 topologies of the same size. According to [87], we assume that the Markov based approximation algorithm converges if the collected profit does not change more than 0.1% of the value obtained at the current state.

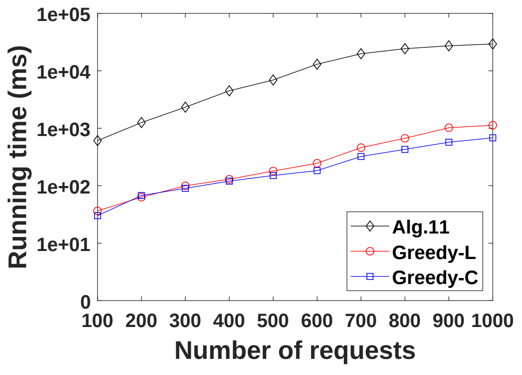
5.5.2 Performance evaluation of different algorithms

We first studied the performance of different algorithms, by varying the number of requests from 100 to 1,000 with the network size of 100. Figure 5.2(a) depicts the accumulated profit with varying number of requests while keeping other settings unchanged. In addition, the associated running time is shown in Figure 5.2(b). It can be seen from Figure 5.2(a) that `Alg.11` outperforms the benchmarks `Greedy-L` and `Greedy-C` in all cases. When the number of requests is 100, the profits collected by `Greedy-L` and `Greedy-C` are 57.4% and 37.6% of that by `Alg.11`, respectively. This is because `Alg.11` has an advantage in lowering the admission cost when the network has enough resource to admit all requests. When the number of requests is

1, 000, the profits collected by `Greedy-L` and `Greedy-C` are 51.2% and 39.8% of that by `Alg.11`, respectively. This is because `Alg.11` delivers a more reasonable scheduling of resource to admit more requests with the limited resource in a network.

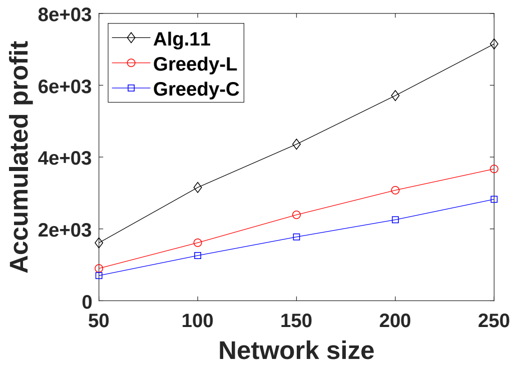


(a) The accumulated profit

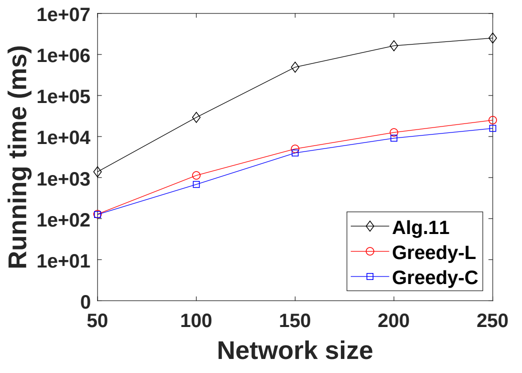


(b) The running time

Figure 5.2: Performance of different algorithms by varying the number of requests from 100 to 1,000 with the network size of 100.



(a) The accumulated profit



(b) The running time

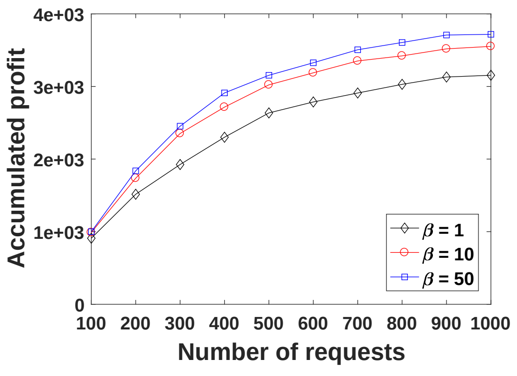
Figure 5.3: Impact of network size on different algorithms by varying the number of nodes from 50 to 250 with 1, 000 requests.

5.5.3 Impact of different parameters on the performance of the proposed algorithm

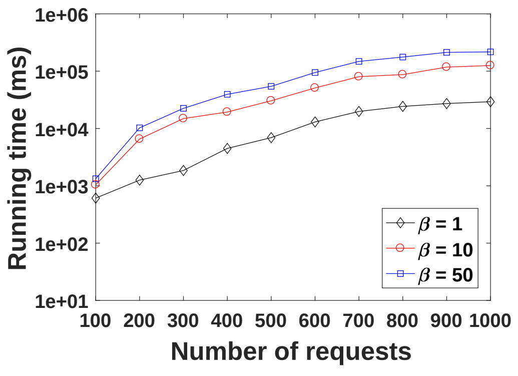
We then investigated the impacts of important parameters on the performance of the proposed algorithm, such as the network size, the parameter β, the uncertainty of demanded resource and the control parameter ζ. Recall that β is an important parameter in designing a Markov chain, and the control parameter ζ is related to the cost variation.

We started by investigating the impact of network size on the performance of the proposed algorithm against the benchmarks `Greedy-L` and `Greedy-C`, by varying the network size from 50 to 250 with 1,000 requests. Recall that the number of cloudlets is set as 10% of the network size. Figure 5.3(a) depicts the accumulated profit with varying numbers of network size, while Figure 5.3(b) depicts the related running time. It can be seen from Figure 5.3(a) that when the network size is 250, the profit collected by `Greedy-L` is 52.3% of that by `Alg.11`, while the profit collected by `Greedy-C` is 38.5% of that by `Alg.11`. This can be justified that when the network size is large, compared with the greedy algorithms, `Alg.11` achieves better utilization of computing resource and bandwidth resource to avoid the overloading on links and cloudlets.

We then studied the impact of parameter β on the performance of the proposed algorithm, by varying the number of requests from 100 to 1,000 with the network size of 100. Figure 5.4(a) demonstrates the impact of parameter β on the collected profit of `Alg.11` while Figure 5.4(b) demonstrates the convergence time with different β. With 1,000 requests, when parameter $\beta = 50$, the proposed algorithm achieves the best performance, which is 17.8% higher than that by the proposed algorithm with parameter $\beta = 1$. However, in this case, it takes the longest convergence time. The rationale behind this is that when the value of parameter β is small, the optimality gap is enlarged by Theorem 5.2. However, following the convergence time analysis by Theorem 5.4, it consumes much less time to achieve convergence. It implies that `Alg.11` demonstrates good flexibility for us to set parameter β with a reasonable value to achieve a good trade-off between the performance and convergence time.



(a) The accumulated profit



(b) The running time

Figure 5.4: Impact of parameter β on the proposed algorithm by varying the number of requests from 100 to 1,000 with the network size of 100.

We thirdly evaluated the impact of the uncertainty of demanded resource on the performance of the proposed algorithm, by varying the number of requests from 100 to 1,000 with the network size of 100. Figure 5.5(a) depicts the accumulated profit obtained with different uncertainties of the demanded resource. E.g., the uncertainty of demanded resource is ±5% when the upper bounds of the demanded computing

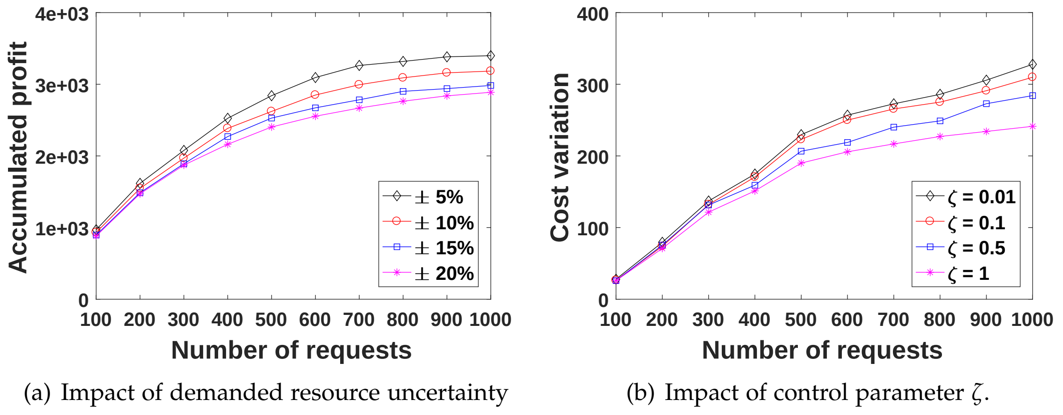


Figure 5.5: Impact of uncertainty of demanded resource and control parameter ζ on the proposed algorithm by varying numbers of requests from 100 to 1,000 with the network size of 100.

resource and data rate are set as 105% of the expected value, while the lower bounds of the demanded computing resource and data rate are set as 95% of the expected value. The actual consumed computing resource and data rate are randomly drawn between the upper bounds and lower bounds. It can be seen from Figure 5.5(a) that the proposed algorithm performs better with lower uncertainty of demanded resource. With 1,000 requests, when the uncertainty is $\pm 20\%$, `Alg.11` achieves 84.9% of the accumulated profit by itself when the uncertainty is $\pm 5\%$. The reason is that higher uncertainty of demanded resource not only enlarges the cost variation, but also perturbs the stationary distribution of the designed Markov chain by Lemma 18.

We finally studied the impact of the control parameter ζ on the performance of the proposed algorithm, by varying the number of requests from 100 to 1,000 with the network size of 100. Figure 5.5(b) depicts the cost variation with varying control parameter ζ. Recall that the control parameter ζ is used to stabilize the total admission cost as a coefficient of cost variation. With 1,000 requests, when control parameter $\zeta = 1$, `Alg.11` achieves 73.7% of the cost variation by itself when control parameter $\zeta = 0.01$. It can be seen from Figure 5.5(b) that a larger ζ leads to a smaller cost variation, and the total admission cost becomes more stable with a larger ζ. The reason is that we assign a higher weight to cost variation compared with the accumulated cost.

5.6 Summary

In this chapter, we studied user service request admissions with both SFC and latency requirements in an MEC network. We first formulated a novel RSFCP problem with the aim to maximize the expected profit of the network service provider through admitting as many user requests as possible. We then formulated a QIP exact solution to the problem when its size is small or moderate. Furthermore, we developed a Markov based approximation algorithm, which can deliver a near-optimal solution with a moderate bounded gap for the problem without measurement perturbation. We also extended the proposed approach to the measurement perturbation case, for which we showed that the proposed approximation algorithm is still applicable, and the solution delivered has a near-optimal gap with a guaranteed error bound. We finally evaluated the performance of the proposed algorithm through experimental simulations with practical settings. Experimental results demonstrated that the proposed algorithm is promising, and outperforms the mentioned benchmarks. Several potential topics based on this study can be further explored in the future. For example, the problem can be extended to an online setting where requests arrive one by one without the knowledge of future arrival information. Furthermore, the mobility of mobile users can also be taken into account when dealing with robust service provisioning.

Chapter 6

Conclusions and Future Work

This chapter summarizes the contributions we made in this thesis, followed by discussing potential research topics derived from this work.

6.1 Summary of Contributions

In this thesis, we have systematically studied efficient virtual service provisioning for Internet of Things (IoT) applications in Mobile Edge Computing (MEC) environments. Novel concepts, system models and optimization techniques were proposed to facilitate efficient resource allocation and ease the implementation of IoT services in MEC networks, through bridging the theory-practice gap. Extensive experiments have been conducted in this thesis by simulations using both real and synthetic datasets to evaluate all proposed algorithms. Furthermore, the devised algorithms, as well as the proposed algorithm design and analysis techniques, will be of independent interests in many other domains, especially in the combinatorial optimization domain. We proposed a novel metric to measure user satisfaction of using delay-aware IoT services, and devised efficient approximation and online algorithms with performance guarantees to maximize user service satisfaction in MEC environments. To provide cost-effective services in MEC for multi-source IoT applications, we built the very first IoT-driven service provisioning framework for multi-source IoT applications to meet Service Function Chain (SFC) requirements, while minimizing the total operational costs of service providers. We then devised performance-guaranteed approximation and online algorithms for admitting delay-aware Deep Neuron Network (DNN) inference service requests in MEC environments. We also tackled the uncertain computing resource consumption and demanded data rates in service request implementation with the aim of maximizing the expected profit, and proposed a Markov based approximation algorithm for robust and efficient IoT service provisioning in MEC.

The main contributions of this thesis are summarized as follows.

- We investigated the user service satisfaction for delay-sensitive IoT applications in an MEC environment, by offloading user service requests to either the remote cloud or local cloudlets in an MEC network. We first formulated two novel optimization problems and showed their NP-hardness. Especially, we proposed an approximation algorithm and a heuristic algorithm for the total utility maximization problem without and with the bandwidth capacity constraint, respectively. We also devised efficient online algorithms for the dynamic total utility maximization problem for dynamic request admissions without the knowledge of future request arrivals.

- We explored the IoT-driven service provisioning in an MEC network for multi-source IoT applications with SFC requirements. We first formulated the cost minimization problems for a single multi-source IoT application and a set of multi-source IoT applications, respectively. We then showed that the problems are NP-hard. Furthermore, we proposed a novel IoT-driven service provisioning framework for multi-source IoT applications, which includes uploading stream data from multiple IoT sources, VNF instance placement and sharing, in-network aggregation of data streams, and workload balancing among cloudlets. We also devised efficient algorithms for the defined problems.

- We studied the DNN inference service provisioning with inference delay requirements in an MEC environment. We first formulated a novel DNN inference throughput maximization problem with the aim to maximize the number of delay-aware DNN service requests admitted, by accelerating each DNN inference through jointly exploring DNN model partitioning and inference parallelism of DNN inference. To this end, we showed that the problem is NP-hard and then devised a constant approximation algorithm for it. We also proposed an online algorithm with a provable competitive ratio for the dynamic DNN inference throughput maximization problem where a sequence of delay-aware DNN inference requests arrives one by one without the knowledge of future arrivals.

- We dealt with admitting user requests with both SFC and latency requirements in an MEC network. We first formulated a novel Robust Service Function Chain Placement (RSFCP) problem with the aim to maximize the expected profit of the network service provider, under the uncertainty assumption of both computing resource and data rate demanded by request executions. We formulated the RSFCP problem as a Quadratic Integer Programming (QIP) and showed that the problem is NP-hard. We then developed a near-optimal algorithm for it by adopting the Markov approximation technique. We also analyzed the proposed approximation algorithm with the optimality gap, the bounds on the convergence time and perturbation caused by inaccurate measurements.

- We conducted extensive experimental simulations, using both real and synthetic datasets to evaluate all proposed algorithms and investigate the impact of constraint parameters on their performance. Experimental results demonstrated that the proposed algorithms are promising, and outperform existing ones significantly in a series of aspects, such as maximizing user service satisfaction, minimizing operational costs, maximizing network throughput, and maximizing expected profit.

6.2 Future Directions

So far, we have studied several fundamental issues to enable efficient virtual service provisioning for delay-sensitive IoT applications in MEC environments. Derived from the studies in this thesis, several potential topics can be explored in future.

Firstly, we will investigate how to handle the massive mobility of mobile users to provide seamless IoT services in MEC networks. To this end, we will consider user mobility modelling to capture the features of dynamic user movement. Then we will devise an effective prediction mechanism to accurately predict user movements in the future by adopting machine learning-based methods. For example, given the current location of a mobile user, we are able to leverage the devised prediction mechanism and obtain the probabilities of the user's future movements. Then the service of the user will be migrated to the identified location with the highest movement probability to mitigate service interruption. On the other hand, we can proactively place service replicas on strategic locations, depending on the movement probabilities of the user.

Secondly, we will study the DNN inference service provisioning in MEC by exploring more DNN inference acceleration techniques. Although we have investigated the joint adoption of DNN model partitioning and inference parallelism in meeting the inference delay requirements of users, we can also investigate other DNN inference acceleration techniques, such as model compression and model early-exist techniques. The model compression and model early-exist techniques accelerate the DNN inference at the expense of the inference accuracy, and a deep reinforcement learning method can be leveraged to achieve the finest trade-off between the inference delay and the accuracy, according to different scenarios. In the future work, we will address how to adopt various DNN inference acceleration techniques to provide efficient DNN inference services while meeting the Quality of Service (QoS) requirements in MEC environments.

Finally, we will look at the federated learning technique for DNN training service provisioning in MEC networks. Pushed by the development of MEC, federated learning is a recently emerging paradigm to utilize distributed computation resource for distributed model training with privacy protection, thereby relieving the centralized workload [24]. In other words, multiple edge servers collaborate with each other to establish a global learning model and aggregate model updates in federated learning. Since we have explored the DNN inference service provisioning in MEC in this thesis, we will investigate how to design a efficient federated learning framework for DNN training acceleration in MEC environments to achieve edge intelligence, with a series of optimization objectives, such as minimizing the operational cost, maximize the number of requests admitted, and maximizing the accumulated profit.

Bibliography

1. M. Abadi *et al.* Tensorflow: A system for large-scale machine learning. *OSDI*, vol. 16, pp. 265 – 283, 2016. (cited on pages 15 and 76)

2. N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450 – 465, 2018. (cited on pages 2 and 49)

3. H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi. Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing. *IEEE J. Selected Areas in Communications*, vol. 37, no. 3, pp. 668 – 681, 2019. (cited on page 10)

4. O. Alhussein, P. T. Do, J. Li, Q. Ye, W. Shi, W. Zhuang, X. Shen, X. Li, and J. Rao. Joint VNF placement and multicast traffic routing in 5G core networks. *Proc. of Globecom'18*, IEEE, 2018. (cited on page 12)

5. S. Ali, A. A. Maciejewski, H. J. Siegel, and J. Kim. Measuring the robustness of a resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 7, pp. 630 — 641, 2004. (cited on pages 17, 18, and 101)

6. A. M. Alwakeel, A. K. Alnaim, and E. B. Fernandez. A survey of network function virtualization security. Proc. of SoutheastCon'18, pp. 1 – 8. 2018. (cited on page 4)

7. Amazon Web Services, Inc., Amazon EC2 Instance Types, 2021. [Online]. Available: https://aws.amazon.com/ec2/instance-types/ (cited on page 95)

8. S. Arisdakessian, O. A. Wahab, A. Mourad, H. Otrok, and N. Kara. FoGMatch: An intelligent multi-criteria IoT-fog scheduling approach using game theory. *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1779 – 1789, 2020. (cited on page 10)

9. M. T. Beck and J. F. Botero. Coordinated allocation of service function chains. *Proc. of GLOBECOM'15*, IEEE, 2016. (cited on pages 16, 18, 101, and 122)

10. S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004. (cited on pages 111 and 117)

11. Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1124 – 1137, 2004. (cited on pages 89 and 94)

12. A. Bozorgchenani, F. Mashhadi, D. Tarchi, and S. Salinas. Multi-objective computation sharing in energy and delay constrained mobile edge computing environments. *IEEE Transactions on Mobile Computing*, vol. 20, no. 10, pp. 2992 – 3005, 2021. (cited on page 14)

13. A. Ceselli, M. Premoli, and S. Secci. Mobile edge cloud network design optimization. *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1818 – 1831, 2017. (cited on page 12)

14. M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed Steiner problems. *Journal of Algorithms*, vol. 33, no. 1, pp. 73 — 91, 1999. (cited on page 55)

15. H. Chen, S. Xu, X. Wang, Y. Zhao, K. Li, Y. Wang, W. Wang, and L. M. Li. Towards optimal outsourcing of service function chain across multiple clouds. *Proc. of ICC'16*, IEEE, 2016. (cited on pages 17 and 18)

16. M. Chen, W. Liang, and J. Li. Energy-efficient data collection maximization for UAV-assisted wireless sensor networks. *Proc. of WCNC'21*, IEEE, 2021. (cited on page 3)

17. M. Chen, S. C. Liew, Z. Shao, and C. Ka. Markov approximation for combinatorial network optimization. *Proc. of INFOCOM'10*, IEEE, 2010. (cited on pages 110, 111, 113, 119, and 120)

18. X. Chen, L. Jiao, W. Li, and X. Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795 – 2808, 2016. (cited on pages 80 and 95)

19. Cisco annual Internet report (2018-2023) white paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html, 2020. (cited on page 3)

20. R. Cohen, L. Eytan, J. Naor, and D. Raz. On the effect of forwarding table size on SDN network utilization. *Proc. of INFOCOM'14*, IEEE, 2014. (cited on page 49)

21. R. Cohen, L. Eytan, J. Naor, and D. Raz. Near optimal placement of virtual network functions. *Proc. of INFOCOM'15*, IEEE, 2015. (cited on page 49)

22. R. Cohen, L. Katzir, and D. Raz. An efficient approximation for the generalized assignment problem. *Information Processing Letters*, vol. 100, pp. 162 – 166, 2006. (cited on pages 27, 28, 29, 31, 81, 85, 86, 88, and 89)

23. S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457 – 7469, 2020. (cited on page 6)

24. Y. Deng, F. Lyu, J. Ren, Y. Chen, P. Yang, Y. Zhou, and Y. Zhang. FAIR: Quality-aware federated learning with precise user incentive and model aggregation. *Proc. of INFOCOM'21*, IEEE, 2021. (cited on page 131)

25. P. Diaconis and D. Stroock. Geometric bounds for eigenvalues of Markov chains. *The Annals of Applied Probability* vol. 1, no. 1, pp. 36 — 61, 1991. (cited on page 121)

26. N. Eshraghi, and B. Liang. Joint offloading decision and resource allocation with uncertain task computing requirement. *Proc. of INFOCOM'19*, IEEE, 2019. (cited on pages 103, 108, 109, and 123)

27. F. Esposito *et al.* Necklace: An architecture for distributed and robust service function chains with guarantees. *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 152 – 166, 2021. (cited on pages 17 and 18)

28. ETSI. Network functions virtualisation (NFV): Architectural framework, *ETSI GS NFV 002 v1.2.1*, 2014. (cited on pages xvii, 4, and 5)

29. J. Fan, C. Guan, Y. Zhao, and C. Qiao. Availability-aware mapping of service function chains. *Proc. of INFOCOM'17*, IEEE, 2017. (cited on pages 5 and 122)

30. H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molish. Approximation algorithms for the NFV service distribution problem. *Proc. of INFOCOM'17*, IEEE, 2017. (cited on pages 12 and 13)

31. A. Goldsmith. *Wireless communications*. Cambridge university press, 2005. (cited on page 80)

32. Z. Gong, H. Ji, C. W. Fletcher, C. J. Hughes, S. Baghsorkhi, and J. Torrellas. SAVE: Sparsity-aware vector engine for accelerating DNN training and inference on CPUs. *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020. (cited on page 77)

33. R. Gouareb, V. Friderikos, and A. Aghvami. Virtual network functions routing and placement for edge cloud latency minimization. *IEEE J. Selected Areas in Communications*, vol. 36, no. 10, pp. 2346 – 2357, 2018. (cited on page 9)

34. GT-ITM. http://www.cc.gatech.edu/projects/gtitm/, 2019. (cited on pages 42, 67, 122, and 123)

35. A. Gushchin, A. Walid, and A. Tang. Scalable routing in SDN-enabled networks with consolidated middleboxes. *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, ACM, 2015. (cited on page 49)

36. C. Hu, W. Bao, D. Wang, and F. Liu. Dynamic adaptive DNN surgery for inference acceleration on the edge. *Proc. of INFOCOM'19*, pp. 1423 – 1431, IEEE, 2019, (cited on pages 15, 75, and 95)

37. H. Huang, S. Guo, W. Liang, K. Li, B. Ye, and W. Zhuang. Near-optimal routing protection for in-band software-defined heterogeneous networks. *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 11, pp. 2918 — 2934, 2016. (cited on pages 110, 120, and 122)

38. M. Huang, W. Liang, Y. Ma, and S. Guo. Throughput maximization of delay-sensitive request admission via virtualized network function placements and migrations. *Proc of ICC'18*, IEEE, 2018. (cited on page 9)

39. L. T. Hwang and T.S.J. Horng. 3D IC and RF SiPs: Advanced stacking and planar solutions for 5G mobility. *Wiley*, 2018. (cited on page 101)

40. M. Jalalitabar, E. Guler, G. Luo, L. Tian, and X. Cao. Dependence-aware service function chain design and mapping. *Proc. of GLOBECOM'17*, IEEE, 2017. (cited on pages 16, 18, and 101)

41. Y. Jararweh, A. Doulat, O. AlQudah, E. Ahmed, M. Al-Ayyoub, and E. Benkhelifa. The future of mobile cloud computing: Integrating cloudlets and mobile edge computing. *Proc. 23rd Int. Conf. Telecommun. (ICT)*, pp. 1 – 5, 2016. (cited on page 2)

42. M. Jia, J. Cao, and W. Liang. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. *IEEE Transactions on Cloud Computing*, vol. 5, no. 4, pp. 725 – 737, 2017. (cited on pages 9 and 12)

43. 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. *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 732 – 745, 2018. (cited on page 4)

44. M. Jia, W. Liang, Z. Xu, and M. Huang. Cloudlet load balancing in wireless metropolitan area networks. *Proc. of INFOCOM'16*, IEEE, 2016. (cited on page 12)

45. Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 615 – 629, 2017. (cited on pages 15, 75, and 95)

46. F. Kelly. *Reversibility and stochastic networks*. Cambridge University Press, 2011. (cited on page 115)

47. D. A. Levin and Y. Peres. *Markov chains and mixing times*. American Mathematical Soc., vol. 107, 2017. (cited on page 120)

48. H. Li, G. Shou, Y. Hu, and Z. Guo. Mobile edge computing: Progress and challenges. *Proc. 4th IEEE Int. Conf. Mobile Cloud Comput.Services Eng. (MobileCloud)*, pp. 83 – 84, 2016. (cited on page 1)

49. J. Li, W. Liang, M. Chen, and Z. Xu. Mobility-aware dynamic service placement in D2D-assisted MEC environments. *Proc. of WCNC'21*, IEEE, 2021. (cited on page 102)

50. J. Li, W. Liang, M. Huang, and X. Jia. Providing reliability-aware virtualized network function services for mobile edge computing. *Proc. of ICDCS'19*, pp. 732–741, IEEE, 2019. (cited on page 3)

51. J. Li, W. Liang, M. Huang, and X. Jia. Reliability-aware network service provisioning in mobile edge-cloud networks. *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 7, pp. 1545 – 1558, 2020. (cited on pages 3 and 8)

52. J. Li, W. Liang, Y. Li, Z. Xu, and X. Jia. Delay-aware DNN inference throughput maximization in edge computing via jointly exploring partitioning and parallelism. *Proc. of LCN'21*, IEEE, 2021. (cited on page 75)

53. J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo. Throughput maximization of delay-aware DNN inference in edge computing by exploring DNN model partitioning and inference parallelism. To appear in *IEEE Transactions on Mobile Computing*, 2021, doi: 10.1109/TMC.2021.3125949. (cited on page 75)

54. J. Li, W. Liang, and Y. Ma. Robust service provisioning with service function chain requirements in mobile edge computing. *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 2138 – 2153, 2021. (cited on page 102)

55. J. Li, W. Liang, W. Xu, Z. Xu, X. Jia, W. Zhou, and J. Zhao. Maximizing user service satisfaction for delay-sensitive IoT applications in edge computing. *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 5, pp. 1199 – 1212, 2022 (cited on page 22)

56. J. Li, W. Liang, W. Xu, Z. Xu, and J. Zhao. Maximizing the quality of user experience of using services in edge computing for delay-sensitive IoT applications. *Proc. of MSWiM'20*, ACM, 2020. (cited on page 22)

57. J. Li, W. Liang, Z. Xu, X. Jia, and W. Zhou. Service provisioning for multi-source IoT applications in mobile edge computing. *ACM Transactions on Sensor Networks*, vol. 18, no. 2, article. 17, pp. 17:1 – 17:25, 2022. (cited on page 77)

58. J. Li, W. Liang, Z. Xu, and W. Zhou. Provisioning virtual services in mobile edge computing for IoT applications with multiple sources. *Proc. of LCN'20*, IEEE, 2020. (cited on page 77)

59. Y. Li, W. Liang, and J. Li. Profit maximization for service placement and request assignment in edge computing via deep reinforcement learning. *Proc. of MSWiM'21*, ACM, 2021. (cited on page 2)

60. Y. Li, W. Liang, W. Xu, and X. Jia. Data collection of IoT devices using an energy-constrained UAV. *Proc. of IPDPS'20*, pp. 644 – 653, IEEE, 2020. (cited on page 3)

61. W. Liang, Y. Ma, W. Xu, X. Jia, and S. Chau. Reliability augmentation of requests with service function chain requirements in mobile edge-cloud networks. *Proc. of ICPP'20*, ACM, 2020. (cited on page 4)

62. S. Lin, W. Liang, and J. Li. Reliability-aware service function chain provisioning in mobile edge-cloud networks. *Proc. of ICCCN'20*, IEEE, 2020. (cited on page 2)

63. J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu. On dynamic service function chain deployment and readjustment. *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 543 – 553, 2017. (cited on pages 16, 18, and 101)

64. T. Liu, L. Fang, Y. Zhu, W. Tong, and Y. Yang. A near-optimal approach for online task offloading and resource allocation in edge-cloud orchestrated computing. To appear in *IEEE Transactions on Mobile Computing*, 2020, doi: 10.1109/TMC.2020.3045471. (cited on page 14)

65. Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen. Toward edge intelligence: Multiaccess edge computing for 5G and internet of things. *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6722 – 6747, 2020. (cited on page 6)

66. Y. Liu, Y. Wang, R. Yu, M. Li, V. Sharma, and Y. Wang. Optimizing CNN model inference on CPUs. *Proc. of USENIX Annu. Tech. Conf.*, pp. 1025 – 1040, 2019. (cited on pages 15, 76, 77, and 95)

67. X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Zhang, and R. P. Liu. Energy-efficient admission of delay-sensitive tasks for mobile edge computing. *IEEE Transactions on Communications*, vol. 66, no. 6, pp. 2603 – 2616, 2018. (cited on page 9)

68. Y. Ma, W. Liang, J. Li, X. Jia, and S. Guo. Mobility-aware and delay-sensitive service provisioning in mobile edge-cloud networks. *IEEE Transactions on Mobile Computing*, vol. 21, no. 1, pp. 196 – 210, 2022. (cited on page 77)

69. Y. Ma, W. Liang, J. Wu and Z. Xu. Throughput maximization of NFV-enabled multicasting in mobile edge cloud networks. *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 2, pp. 393 – 407, 2020. (cited on pages 12, 13, 67, 122, and 123)

70. Y. Ma, W. Liang, Z. Xu, and S. Guo. Profit maximization for admitting requests with network function services in distributed clouds. *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 5, pp. 1143 – 1157, 2019. (cited on pages 12, 42, 67, 103, 107, 122, and 123)

71. P. Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628 – 1656, 2017. (cited on page 1)

72. T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco. Distributed inference acceleration with adaptive DNN partitioning and offloading. *Proc. of INFOCOM'20*, pp. 854 – 863, IEEE, 2020. (cited on pages 14, 75, and 95)

73. C. Mouradian, N. Jahromi, and R. Glitho. NFV and SDN-based distributed IoT gateway for large-scale disaster management. *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 4119 – 4131, 2018. (cited on page 12)

74. M. Nguyen, M. Dolati, and M. Ghaderi. Deadline-aware SFC orchestration under demand uncertainty. *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2275 – 2290, 2020. (cited on pages 17 and 18)

75. W. Niu *et al.* Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. *Proc. of ASPLOS'20* , pp. 907 – 922, 2020. (cited on page 15)

76. A. V. Nori, R. Bera, S. Balachandran, J. Rakshit, O. J. Omer, A. Abuhatzera, B. Kuttanna, and S. Subramoney. Proximu$: Efficiently scaling DNN inference in multi-core CPUs through near-cache compute. *arXiv preprint arXiv:2011.11695*, 2020. (cited on pages 15 and 77)

77. A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. *Proc. of NIPS Workshop*, 2017. (cited on pages 15 and 76)

78. M. Patel *et al.* Mobile-edge computing—Introductory technical white paper. White Paper, Mobile-Edge Computing (MEC) Industry Initiative, 2014. (cited on page 2)

79. D. Pisinger. Algorithms for knapsack problems. *Citeseer*, 1995. (cited on pages 109 and 110)

80. K. Psychas and J. Ghaderi. Scheduling jobs with random resource requirements in computing clusters. *Proc. of INFOCOM'19*, IEEE, 2019. (cited on pages 17, 18, and 103)

81. H. Ren, Z. Xu, W. Liang, Q. Xia, P. Zhou, O. Rana, A. Galis, and G. Wu. Efficient algorithms for delay-aware NFV-enabled multicasting in mobile edge clouds with resource sharing. *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2050 – 2066, 2020. (cited on pages 12, 13, and 67)

82. G. Sallam, G. R. Gupta, B. Li, and B. Ji. Shortest path and maximum flow problems under service function chaining constraints. *Proc. of INFOCOM'18*, IEEE, 2018. (cited on pages 4 and 122)

83. A. Samanta and J. Tang. Dyme: Dynamic microservice scheduling in edge computing enabled IoT. *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6164 – 6174, 2020. (cited on page 10)

84. Y. Sang, B. Ji, G. Gupta, X. Du and L. Ye. Provably efficient algorithms for joint placement and allocation of virtual network functions. *Proc of GLOBECOM'17*, IEEE, 2017.

85. A. Shakarami, M. Ghobaei-Arani, and A. Shahidinejad. A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. *Computer Networks.*, vol. 182, no. 107496, 2020. (cited on page 3)

86. Z. Shao, X. Jin, W. Jiang, M. Chen, and M. Chiang. Intra-data-center traffic engineering with ensemble routing. *Proc. of INFOCOM'13*, IEEE, 2013. (cited on page 115)

87. Z. Shao, H. Zhang, M. Chen, and K. Ramchandran. Reverse-engineering bittorrent: A Markov approximation perspective. *Proc. of INFOCOM'12*, IEEE, 2012. (cited on page 123)

88. W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637 – 646, 2016. (cited on page 3)

89. D. B. Shmoys, J. Wein, and D. P. Williamson. Scheduling parallel machines online. *SIAM Journal on Computing*, vol. 24, no. 6, pp. 1313 – 1331, 1995. (cited on pages 18, 101, and 103)

90. Y. Song, S. S. Yau, R. Yu, X. Zhang, and G. Xue. An approach to QoS-based task distribution in edge computing networks for IoT applications. *Proc. of the International Conference on Edge Computing (EDGE)*, pp. 32 – 39, IEEE, 2017. (cited on pages 10, 11, and 42)

91. H. Soni, W. Dabbous, T. Turletti, and H. Asaeda. NFV-based scalable guaranteed-bandwidth multicast service for software-defined ISP networks. *IEEE Transactions on Network and Service Management*, vol. 14, no. 5, pp. 1157 – 1170, 2017. (cited on pages 12 and 13)

92. G. Sun, Y. Li, D. Liao, and V. Chang. Service function chain orchestration across multiple domains: A full mesh aggregation approach. *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1175 – 1191, 2018. (cited on pages 16, 18, and 101)

93. X. Sun and N. Ansari. EdgeIoT: Mobile edge computing for the internet of things. *IEEE Communications Magazine*, vol. 54, no. 12, pp. 22 – 29, 2016. (cited on page 3)

94. Y. Sun, S. Zhou, and J. Xu. EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks. *IEEE Journal on Selected Areas in Communications*, vol.35, no.11, pp. 2637 – 2646, 2017. (cited on pages 77 and 95)

95. X. Tang, X. Chen, L. Zeng, S. Yu and L. Chen. Joint multiuser DNN partitioning and computational resource allocation for collaborative edge intelligence. *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9511 – 9522, 2021. (cited on page 14)

96. A. Tomassilli, N. Huin, F. Giroire, and B. Jaumard. Resource requirements for reliable service function chaining. *Proc. of ICC'18*, IEEE, 2018. (cited on pages 17, 18, and 101)

97. T. X. Tran and D. Pompili. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856 – 868, 2018. (cited on page 42)

98. D. Tse and P. Viswanath. *Fundamentals of Wireless Communication*. Cambridge University Press, 2005. (cited on page 24)

99. L. Wang, G. V. Laszewski, A. Younge, X. He, M. Kunze, J. Tao, and C. Fu. Cloud computing: A perspective study. *New Generation Computing*, vol. 28, pp. 137 – 146, 2010. (cited on page 1)

100. M. Wang, B. Cheng, S. Wang, and J. Chen. Availability-and traffic-aware placement of parallelized SFC in data center networks, *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 182 – 194, 2021. (cited on pages 17 and 18)

101. N. Wingfield. Amazon's profits grow more than 800 percent, lifted by cloud services. *The New York Times*. 2016. [Online]. Available: http://www.nytimes.com/2016/07/29/technology/amazon-earnings-profit.html?r=0 (cited on page 1)

102. C. Wu *et al.* Machine learning at Facebook: Understanding inference at the edge. *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 331 – 344, 2019. (cited on page 6)

103. W. Wu, S. Pirbhulal, A. K. Sangaiah, et al. Optimization of signal quality over comfortability of textile electrodes for ECG monitoring in fog computing based medical applications. *Future Generation Computer Systems*, vol. 86, pp. 515 – 526, 2018. (cited on page 42)

104. Q. Xia, W. Liang, and W. Xu. Throughput maximization for online request admissions in mobile cloudlets. *Proc of LCN'13*, IEEE, 2013. (cited on pages 9 and 12)

105. Y. Xiang and H. Kim. Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference. *2019 IEEE Real-Time Systems Symposium (RTSS)*, pp. 392 – 405, 2019. (cited on page 15)

106. Z. Xu, W. Gong, Q. Xia, W. Liang, O. F. Rana, and G. Wu. NFV-enabled IoT service provisioning in mobile edge clouds. *IEEE Transactions on Mobile Computing*, vol. 20, no. 5, pp. 1892 – 1906, 2021. (cited on page 10)

107. Z. Xu, W. Liang, M. Huang, M. Jia, S. Guo, and A. Galis. Efficient NFV-enabled multicasting in SDNs. *IEEE Transactions on Communications*, vol. 67, no. 3, pp. 2052 – 2070, 2019. (cited on pages 13, 67, and 123)

108. Z. Xu, W. Liang, M. Jia, M. Huang, and G. 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. (cited on pages 10 and 42)

109. Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo. Efficient algorithms for capacitated cloudlet placements. *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, pp. 2866 – 2880, 2016. (cited on pages 10, 12, and 42)

110. Z. Xu, Z. Zhang, W. Liang, Q. Xia, O. F. Rana, and G. Wu. QoS-aware VNF placement and service chaining for IoT applications in multi-tier mobile edge networks. *ACM Transactions on Sensor Networks*, vol. 16, no. 3, article 23:1 – 23:27, 2020. (cited on page 12)

111. Z. Xu, L. Zhao, W. Liang, O. Rana, P. Zhou, Q. Xia, W. Xu, and G. Wu. Energy-aware inference offloading for DNN-driven applications in mobile edge clouds. *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 4, pp. 799 – 814, 2021. (cited on pages 14, 75, 79, and 80)

112. S. Yang, F. Li, S. Trajanovski, R. Yahyapour, and X. Fu. Recent advances of resource allocation in network function virtualization. *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 295 – 314, 2020. (cited on pages 4 and 5)

113. B. Yi, X. Wang, K. Li, S. K. Das, and M. Huang. A comprehensive survey of network function virtualization. *Computer Networks*, vol. 133, pp. 212 – 262, 2018. (cited on page 3)

114. F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell. BDD100K: A diverse driving dataset for heterogeneous multitask learning. *Proc. of CVPR'20*, pp. 2633 – 2642, 2020. (cited on page 95)

115. R. Yu, G. Xue, and X. Zhang. Application provisioning in fog computing-enabled internet-of-things: A network perspective. *Proc of INFOCOM'18*, pp. 783 – 791, IEEE, 2018. (cited on pages 10, 11, and 42)

116. L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang. CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices. To appear in *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595 – 608, 2021. (cited on pages 15 and 75)

117. J. Zhang, D. Zeng, L. Gu, H. Yao, and M. Xiong. Joint optimization of virtual function migration and rule update in software defined NFV networks. *Proc. of GLOBECOM'17*, IEEE, 2017. (cited on pages 17 and 18)

118. S. Q. Zhang, Q. Zhang, H. Bannazadeh, and A. L. Garcia. Network function virtualization enabled multicast routing on SDN. *Proc. of ICC'15*, IEEE, 2015. (cited on pages 12 and 13)

119. X. Zhang, Z. Xu, L. Fan, S. Yu, and Y. Qu. Near-optimal energy-efficient algorithm for virtual network function placement. To appear in *IEEE Transactions on Cloud Computing*, 2019, doi: 10.1109/TCC.2019.2947554. (cited on pages 17, 18, and 101)

120. X. Zhang, R. Zhou, Z. Zhou, J. Lui, and Z. Li. An online learning-based task offloading framework for 5G small cell networks. *Proc. of ICPP'20*, ACM, 2020. (cited on pages 17, 18, and 101)

121. D. Zheng, C. Peng, X. Liao, L. Tian, G. Luo, and X. Cao. Towards latency optimization in hybrid service function chain composition and embedding. *Proc. of INFOCOM'20*, IEEE, 2020. (cited on pages 17, 18, and 101)

122. Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *in Proc. of IEEE*, vol. 107, no. 8, pp. 1738 – 1762, 2019. (cited on pages 6 and 75)

123. Z. Zhu, H. Lu, J. Li, and X. Jiang. Service function chain mapping with resource fragmentation avoidance. *Proc. of GLOBECOM'17*, IEEE, 2017. (cited on pages 4, 17, 18, and 101)