INTELLIGENT NETWORK MANAGEMENT FOR HETEROGENEOUS SERVICES IN MOBILE EDGE COMPUTING

by

Qiang Liu

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Electrical Engineering

Charlotte

2020

Approved by:

Dr. Tao Han

Dr. Shen-En Chen

Dr. Yu Wang

Dr. Jiang (Linda) Xie

ABSTRACT

QIANG LIU. Intelligent network management for heterogeneous services in mobile edge computing. (Under the direction of DR. TAO HAN)

The proliferation of connected devices creates various use cases and heterogeneous services, e.g., augmented/virtual reality (AR/VR), vehicle-to-everything (V2X), and mobile artificial intelligence. These services and use cases have diverse networking and performance requirements such as throughput, delay, and reliability, which challenge the "one-fit-all" service architecture in current networks. Mobile edge computing (MEC) allows the deployment of computation infrastructures in close proximity to mobile users and is a key technology to effectively serve these services in terms of cost-efficiency, flexibility, and scalability.

In this research, an intelligent network management framework in mobile edge computing is explored. The primary challenges lie in the unique characteristics of heterogeneous services and complicated correlations between network management on multiple technical domains and high-dimension performance requirements of mobile users in complex mobile networks. This research addresses these challenges with two different management approaches, i.e., context-aware service adaptation to network dynamics as service providers and network orchestration intelligence for heterogeneous services as infrastructure providers.

From the perspective of service providers, multiple mobile systems are designed to allow service adaptation under complex network dynamics, e.g., channel variation and traffic workload, which dynamically and adaptively adjust resource allocations and system configurations by exploiting the unique characteristics of individual services. Specifically, two mobile AR/VR systems are proposed in distributed edge computing networks to strike the balance between the quality and round-trip latency (RTT) performance of AR/VR services.

From the perspective of infrastructure providers, multiple network systems are proposed to enable orchestration intelligence with no need for accurate performance modelings of services, which automatically learn to orchestrate multiple domain network resources for supporting various services by exploiting advanced machine learning techniques. First, two resource orchestration systems are proposed, which learn to allocate cross-domain network resources to heterogeneous services by leveraging Gaussian process (GP) regression techniques. Then, a network slicing system is designed to enable intelligent orchestration to network slices under high-dimension complex end-to-end networks by leveraging deep reinforcement learning (DRL) techniques.

The proposed network management framework in this research unveils the promising directions in effectively and efficiently supporting heterogeneous services, and provides important insights for network management design in the next-generation mobile network.

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the supervision of my advisor, advice from my dissertation committee members, assistance from my colleagues, and encouragement from my wife and family.

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Tao Han, for his continuous support, guidance, and supervision throughout these years. He has trained my technical skills, academic writing, and critical thinking in the past four years and ultimately shaped me as a good researcher. The rigorous standards of research and ethic learned from him will continuously drive me to pursue excellence in future research. I am very grateful for the invaluable efforts he spent on me and so honored to be his student.

I would like to thank my dissertation committee members: Dr. Shen-En Chen, Dr. Yu Wang, and Dr. Jiang (Linda) Xie for their generous time and insightful advice along with this dissertation work. Besides, I appreciate the GASP grant from UNC-Charlotte and Research Assistantships from the National Science Foundation (NSF) as the financial assistance for this dissertation.

I would also like to thank my colleagues and friends: Yang Deng, Yongjie Guan, Xueyu Hou, Ephraim Moges, Haoxin Wang, and many others. I am so thankful and appreciative to have been able to receive help from them.

Thanks also to my parents and grandparents, who have raised, supported, and encouraged me with their best wishes through all my life.

Finally, I would like to thank my wife, Hongying Xiao for her support, patience, and encouragement over the past many years.

Great thanks to myself for have been doing hard work over the past five years!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADMM alternating direction method of multipliers

APS Automated car parking system

AR/VR Augmented/virtual reality

CACC Cooperative adaptive cruise control

CN Core network

DDPG Deep deterministic policy gradient

DQN deep Q-network

DRL Deep reinforcement learning

eNB evolved NodeB

ETSI European Telecommunications Standards Institute

gNB next generation NodeB

GPU Graphics processing unit

IoT Internet of things

ITS Intelligent transportation system

LTE Long-Term Evolution

M2M Machine-to-Machine

mAP Mean average precision

MEC Mobile edge computing

MIMO Multiple-input and multiple-output

MINLP Mixed-integer nonlinear programming problem

ML Machine learning

NFV Network functions virtualization

QoA Quality of augmentation

RAN Radio access network

RAP Radio access point

RMSE Root mean square error

RSSI Received signal strength indicator

SDN Software defined networking

TN Transport network

V2X Vehicle-to-everything

CHAPTER 1: INTRODUCTION

1.1 Background on Mobile Edge Computing

The increasing popularity of various connected devices, e.g., smartphones, tablets, laptops, and the Internet of things (IoT), lead to a dramatic explosion of mobile data traffic. By 2022, Ericsson forecasts that more than 1.5 billion IoT devices will be connected with cellular mobile networks [1]. The mobile data traffic is expected to reach 77.5 EB per month by 2022, as forecasted by Cisco's Visual Networking Index (VNI) [2]. Supporting the enormous connectivity and rapidly increasing mobile traffic is an extremely heavy burden for current mobile networks such as 4G Long-Term Evolution (LTE). For example, the number of connected devices in a single cell needed by massive IoT and Machine-to-Machine (M2M) communication could over tens of thousands, which substantially surpluses the original design objective in the typical LTE network [3]. The next-generation mobile network, i.e., 5G, targets to handle the escalating challenges by deploying more cellular base stations with higher wireless bandwidth and more advanced technologies such as massive multiple-input and multiple-output (MIMO) and millimeter-wave [4]. The ever-expanding complex network typologies, e.g., radio access network (RAN), transport network (TN) and core network (CN), along with the heterogeneous network infrastructures, e.g., evolved NodeBs (eNBs) and next-generation NodeBs (gNBs), complicate the network management for the next-generation mobile network.

The proliferation of devices create heterogeneous services and various use cases, e.g., augmented/virtual reality (AR/VR), vehicle-to-everything (V2X), and mobile artificial intelligence, which challenges the "one-fit-all" service architecture in current networks [5]. Unlike the conventional services, these services have highly diverse

networking and performance requirements such as bandwidth, delay, and reliability, and involve multiple technical domains in its lifetime, e.g., radio transmission in RAN, packet transportation in TN, and server computation in edge/cloud. As a result, it is quite a challenge for the mobile network to effectively accommodate these services in terms of scalability, flexibility, availability, and cost-efficiency [6]. For example, mobile AR/VR service requires low-latency and high-bandwidth link to upload the sensed environmental information of mobile users to computation servers, e.g., point clouds and RGB frames, and retrieve the corresponding digital augments and overlays [7], e.g., 3D animation contents. The vehicle communication service needs extremely high-reliable and ultra-low latency radio transmission link to exchange the critical vehicle and road perceptional information, e.g., pedestrian detection and accident reporting.

Mobile edge computing (MEC) is a promising technology to efficiently accommodate these heterogeneous services, which distributes the computation infrastructures, e.g., computing server and storage, to the network edge for supporting massive connections and achieving highly responsive services [8]. The cloud computing offers centralized computing, storage and networking resources in large-scale data centers to realize scalable cloud services for remote mobile users. However, it encounters several noticeable disadvantages, such as long delay, less responsiveness, and heavy backbone traffic, due to the long distance between the service providers and users, and the aggregated backbone traffic from all distributed users [9]. As a result, cloud computing is inappropriate to host delay-sensitive or real-time services, e.g., mobile AR and V2X, because the round-trip time from mobile users to cloud servers could easily reach hundreds of milliseconds. Instead, MEC overcomes these drawbacks of cloud computing by deploying massive edge nodes with computing, storage and networking capabilities to the network edge, e.g., core network or access networks. According to the European Telecommunications Standards Institute (ETSI), MEC [10] is defined

as "Mobile edge computing provides an IT service environment and cloud computing capabilities at the edge of the mobile network, within the radio access network (RAN) and in close proximity to mobile subscribers." Thus, the service providers can instantiate their applications on the distributed edge nodes in close proximity to mobile users to achieve low-latency resource access and high-responsive services. For instance, computing-constrained and battery-limited devices, e.g., disaster surveillance sensors, health monitoring implant, and AR glasses, can save the energy consumption and thus prolong the lifetime by offloading the compute-intensive tasks to edge nodes. The service provider can deploy intelligent transportation system (ITS) applications with local datasets to support ultra-low latency mobility services such as cooperative adaptive cruise control (CACC) and automated car parking system (APS).

Since a massive number of edge nodes are geographically deployed at the network edge, both the network traffic of radio access points (RAPs) and the workload of edge nodes exhibit extremely dynamic spatiotemporal patterns under the unpredictable behaviors of mobile users [11]. Meanwhile, the radio transmission, data transportation, and task computation of heterogeneous services are closely coupled in MEC, which complicates the network management for large-scale mobile networks. Therefore, an intelligent network management framework is needed in mobile edge computing to support these heterogeneous services in terms of efficiency, effectiveness, flexibility, and scalability.

1.2 Problem Statement

1.2.1 Dynamic Adaptive Mobile Augmented Reality

Augmented reality (AR) is able to embed virtual information seamlessly into our physical environment and provide new kinds of experiences where the world is improved by virtual content blending with the real [12]. The key component of AR is a fast and precise detection and understanding of the physical environment so that virtual contents can synchronize with the real world. In AR, the data from the sen-

Figure 1.1: Dynamic Mobile AR with Edge Computing.

sors, e.g., images, is processed by computer vision algorithms to detect objects in the physical environment. However, the computation complexity of computer vision algorithms, e.g., SSD [13], YOLO [14], and Faster R-CNN [15], are usually too high to run on mobile devices. Although there are some low-complexity object detection frameworks and algorithms such as Google MobileNet [16], Tiny YOLO [14] and DeepMon [17] that can perform object detection on mobile devices, the performance of these algorithms in terms of the mean average precision (mAP)[1] and latency are significantly worse than that of the state-of-the-art computer vision algorithms. In addition, because of the hardware limitation, a mobile device cannot perform advanced computer vision analysis, e.g., human action detection [19], which limits the virtual information that can be generated from the physical world.

As illustrated in Fig. 1.1, with MEC, mobile AR clients offload image data to an edge server via wireless access point while the edge server performs environmental understanding, i.e., object detection, and sends the corresponding virtual content back to MAR clients. Since the edge server is placed in close proximity to users, the communication link between the wireless access point and edge server usually has a very low latency in order to provide highly responsive services. However, wireless links between MAR clients and the wireless access point can be highly dynamic, e.g., varying radio channel quality, and thus introduce a long latency during the data offloading and virtual content delivery. Besides, edge server is usually with finite computation capability and thus could be easily overloaded by highly variant

[1]The mean average precision (mAP) is a metric to evaluate the detection accuracy of a visual object detection algorithm [18].

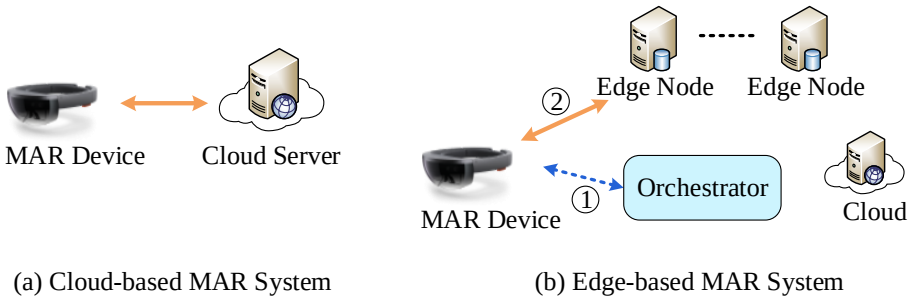

Figure 1.2: The cloud-based vs. edge-based MAR system.

MAR traffic. Therefore, handling the dynamic wireless links and limited computation resources to guarantee the consistent user experience for MAR users is crucial in an edge-assisted MAR system.

1.2.2 Mobile Augmented Reality in Distributed Edge Computing Networks

Mobile augmented reality (MAR) augments a real-world environment by computer-generated sensory information such as text, sound, and graphics. With advanced MAR technologies, the information about a person's surrounding physical environment can be brought out of the digital world and overlaid with the person's perceived real world. Since MAR performs in the semantic context of the real-world, the fast and accurate object analytics is the key component for integrating digital information with environment elements in MAR applications [20].

As illustrated in Fig. 1.2, by offloading the compute-intensive object analysis to powerful edge/cloud servers [21–24], the computational latency and energy consumption of mobile devices can be significantly decreased by exploiting high-end CPU/GPU in edge/cloud servers. Existing works mainly focus on offloading-based MAR systems with a single edge server, which can be easily overloaded by massive MAR services [11]. Multiple distributed edge servers are essential for providing seamless and consistent services for MAR users. Besides, the non-uniform spatiotemporal user distribution [25] can lead to imbalanced workloads among distributed edge servers, which impairs the performance of MAR in terms of the service latency. Therefore,

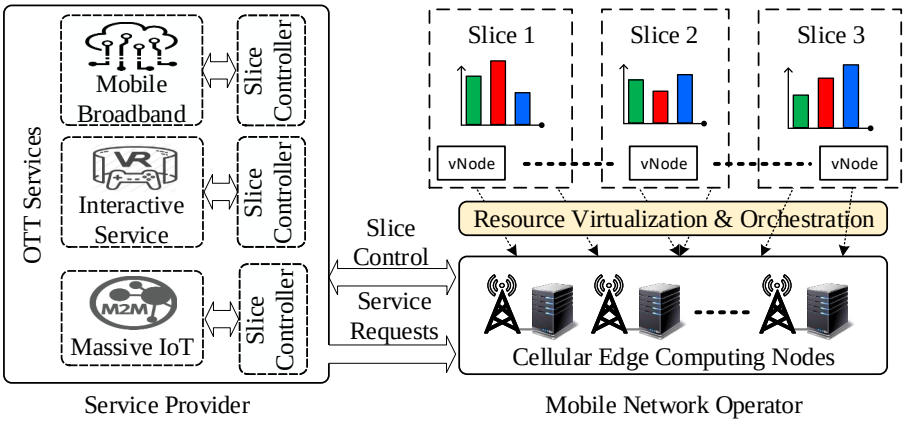

Figure 1.3: The network slicing in mobile edge computing network.

a well-designed network orchestrator, which can dynamically dispatch MAR related computing workloads to distributed edge servers, is needed in an edge-assisted MAR system.

1.2.3 Multiple Domain Virtualization and Orchestration

To efficiently support heterogeneous use cases, network slicing enables the creation of multiple virtual networks on top of a shared physical infrastructure [26, 27], e.g., RAN, TN, and CN. These virtual networks, which are denoted as network slices, are dynamically designed and instantiated to serve the specific needs of vertical industries [28], e.g., MAR and V2X. A network slice usually requires resources from multiple technical domains such as radio access networks and computing servers. Fig. 1.3 shows an example of the network slicing in cellular edge computing networks with a mobile network operator and multiple service providers. The mobile network operator is responsible for managing the physical infrastructures to ensure the co-existence of network slices. The service providers with various over-the-top (OTT) services place service requests to create a network slice for each service and then manage the network slices.

To efficiently utilize the network and computing resources, the network operator needs to virtualize the physical infrastructure to virtual resources and orchestrate

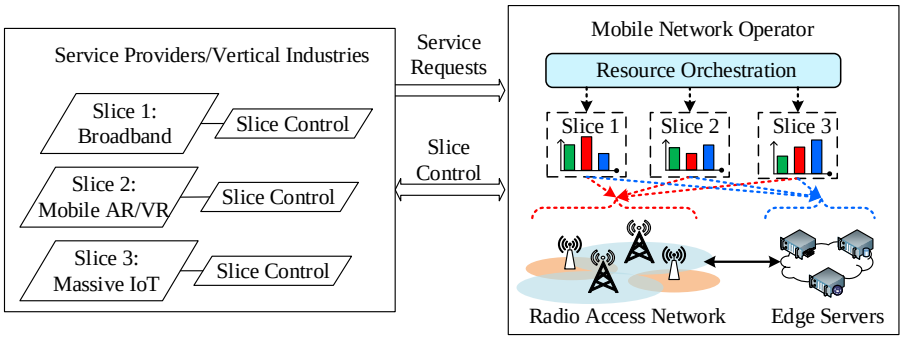

Figure 1.4: Distributed network slicing in mobile edge computing network.

multiple virtual resources to network slices. On one hand, the diverse requirements, e.g., throughput, delay and reliability, of different network slices have to be met, where these requirements are closely coupled in multiple technical domains. On the other hand, the performance and functional isolation among the network slices [26–28] need to be maintained. Here, the performance isolation ensures that the performance of a network slice will not affect or be affected by other network slices that share the same physical network infrastructure. The functional isolation enables each service provider to customize and control its network slice operations independently [29]. Thus, an intelligent resource virtualization and orchestration system is needed to handle the complicated multi-domain correlations of performance requirements of network slices.

1.2.4 Distributed Cross-Domain Resource Orchestration

Network slicing enables mobile network operators to create multiple virtual networks (slices) on top of a common physical network infrastructure, where each slice can be customized to meet a wide variety of network requirements on performance and functionality. As shown in Fig. 1.4, the service providers or vertical industries make the service requests to create network slices and manage the network slices once they are created. The mobile network operator manages the physical network infrastructure and is responsible to create network slices to support diverse requirements from its customers. Since a network slice in cellular edge computing consists

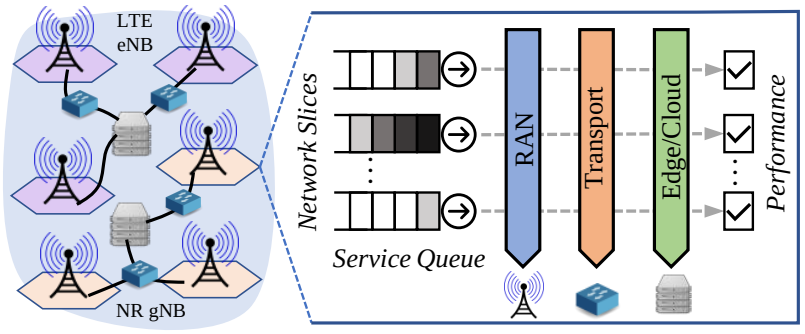

Figure 1.5: An illustration of end-to-end network slicing.

of multiple radio access points (RAPs) and edge servers, the resources allocated to a network slice should be properly distributed among base stations and edge servers to ensure the performance of a network slice and support seamless mobility.

As a critical technology of 5G, network slicing attracts many research efforts from both academia and industry. However, most of the existing works present conceptual network slicing frameworks [26–28, 30, 31], which focus on the virtualization of individual domains and did not investigate how to orchestrate resources from cross-domains such as communication and computing resources. Besides, network slices can be instantiated in large geographic areas to provide seamless coverage for slice users, where the network-wide performances of slices have to be maintained in large-scale mobile networks. Therefore, a distributed cross-domain resource orchestration solution in network slicing is desired to manage network resources to meet the performance requirement of slices.

1.2.5 End-to-End Network Slicing with Decentralized DRL

Leveraging software-defined networking (SDN) and network functions virtualization (NFV), end-to-end network slicing allows individual customization of slices to meet various end-to-end performance requirements of different network services and use cases. Dynamic network slicing, which can dynamically change the resource allocation for slices according to their actual needs, improves the multiplexing efficiency of physical network infrastructures [32].

Dynamic network slicing, as illustrated in Fig. 1.5, faces two research challenges. On one hand, it is almost impossible to obtain the exact correlation between the resources and performance of network slices. A network slice usually requires end-to-end resources from multiple technical domains such as radio access network, transport network, and edge/cloud, which leads to very complex tradeoffs among these resources and slice performances. For example, a short delay in the radio access network might be compensated by accelerated computation in the edge/cloud servers. Besides, the resource orchestration in an end-to-end network slicing system exhibits *Markovian* on serving slice users where a resource orchestration decision affects not only the current but also further network state, e.g., service queues. As a result, conventional model-based approaches [33, 34] cannot effectively handle the high-dim dynamic network slicing problem. We resort to model-free deep reinforcement learning (DRL), which leverages advanced artificial neural networks to deal with high-dim network systems [35].

On the other hand, the spatial diversity of mobile traffic requests the resources of network slices to be properly distributed among base stations and edge/cloud servers in different geographic locations. To meet the network-wide performance of slices, a centralized DRL-based resource orchestration might be deployed, which manages all the network components such as RAPs, TN, and CN. However, it is inefficient to adapt to dynamic network topology and architecture since the input dimension of a neural network based policy is usually fixed.

Thus, a decentralized DRL approach for end-to-end network slicing is desired to automatically learn to orchestrate resources for heterogeneous slices under dynamic mobile networks.

1.3 Overview of the Proposed Research

This research, as shown in Fig. 1.6, proposes an intelligent network management framework in mobile edge computing to support heterogeneous services. It deals

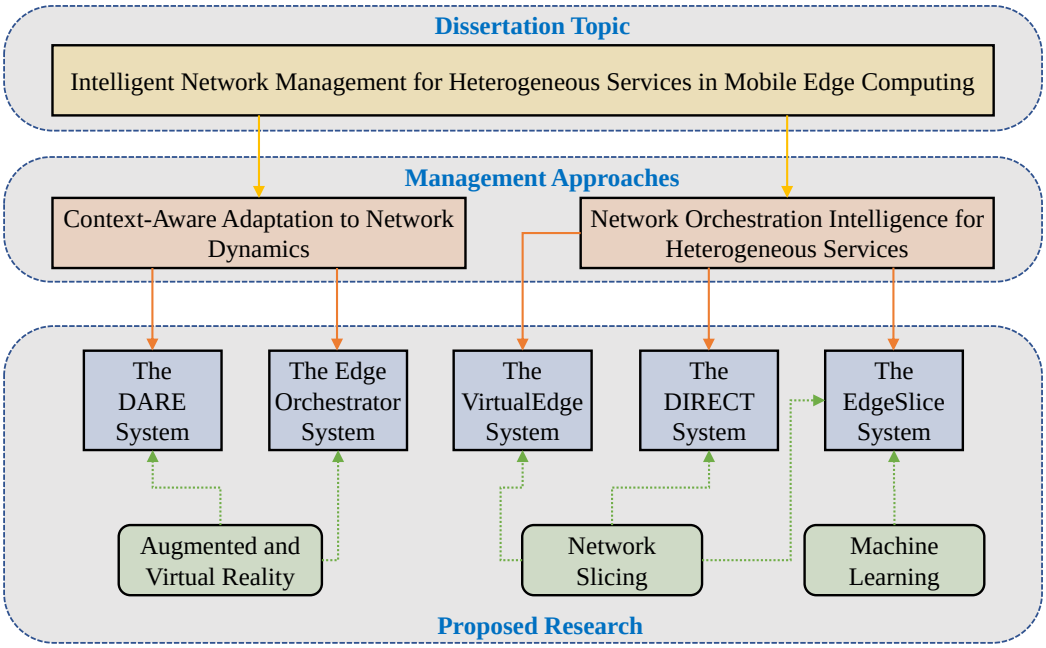

Figure 1.6: Overview of the proposed research.

with these challenges and difficulties with two different management approaches, i.e., context-aware service adaptation to network dynamics and network orchestration intelligence for heterogeneous services from the perspective of service providers and infrastructure providers, respectively.

From the perspective of service providers, multiple mobile systems are designed to allow service adaptation under complex network dynamics, e.g., channel variation and traffic workload, which dynamically and adaptively adjust resource allocations and system configurations by exploiting the unique characteristics of individual services. Specifically, two mobile AR/VR systems are proposed in distributed edge computing networks to strike the balance between the quality and round-trip latency (RTT) performance of AR/VR services.

From the perspective of infrastructure providers, multiple network systems are proposed to enable orchestration intelligence without accurate performance modelings of services, which automatically learn to orchestrate multiple domain network resources for supporting various services by exploiting advanced machine learning techniques.

First, two resource orchestration systems are proposed, which learn to allocate cross-domain network resources to heterogeneous services by leveraging Gaussian process (GP) regression techniques. Then, a network slicing system is designed to enable intelligent orchestration to network slices under high-dimension complex end-to-end networks by leveraging deep reinforcement learning (DRL) techniques.

1.4 Proposal Organization

The remainder of the proposal is organized as follows. In Chapter 2, we briefly review the related work. In Chapter 3, a dynamic adaptive mobile augmented reality system is proposed. In Chapter 4, an edge orchestrator system for MAR is proposed. In Chapter 5, a multiple domain resource orchestration and virtualization system for network slicing is described. In Chapter 6, a distributed cross-domain resource orchestration system for network slicing is designed. In Chapter 7, an automatic resource management system for end-to-end slicing is designed. Following that, the publications and future work are listed in Chapter 8.

CHAPTER 2: RELATED WORK

This chapter discusses the related work of network management in mobile edge computing network.

2.1 Existing Research on Dynamic Adaptive MAR

Since mobile devices are constrained by their computation resources and battery power supplies, most of the existing mobile augmented reality systems are developed by exploiting either cloud or edge servers [21,24,36,37]. To improve the accuracy and latency of object recognition, Jain *et. al.* [20] designed an AR system which utilizes a location-free geometric representation of environments to prune down the visual search space. Chen *et. al.* [22] designed Glimpse which is a continuous real-time object recognition system. In this system, the authors developed an active cache mechanism to improve recognition accuracy and proposed a trigger frame selection method to hide the transmission latency caused by wireless networks. Lee *et. al.* [38] designed a speculative execution system named Outatime to hide the network latency in mobile cloud gaming. The main idea of Outatime is to render speculative frames of future outcomes and deliver the results to the mobile client one round-trip-time ahead of the next input. Cuervo *et. al.* [39] proposed a collaborative rendering system called Kahawai that reduces the requirement of network bandwidth from the server to the mobile cloud gaming client, which uses mobile GPU to render either reduced detail or a subset of frames. Jain *et. al.* [23] proposed a low bandwidth offloading scheme for MAR. This scheme reduces the network latency by only transmitting the distinctive features of images to the server. However, none of the existing works discusses the dynamic adaptation on the AR configurations and edge computation

resource allocations.

2.2 Existing Research on MAR in Distributed MEC

The main objectives in designing offloading-based MAR systems are improving the object recognition accuracy and reducing the service latency [20, 22, 23, 40]. Zhang *et. al.* developed a video analytics system VideoStorm that handles thousands of video analytics queries to tradeoff the variety in quality and latency goals [41]. Yi *et. al.* [42] designed a video analytics system on top of edge computing platform. This system enables the computation offloading from mobile users to edge nodes and exploits the collaboration among edge nodes to reduce the latency of video analytics. On designing offloading-based MAR systems, these works mainly focus on single edge server with sufficient computation capability No solution is provided to properly dispatch computing workloads among heterogeneous edge servers for MAR services.

Jia *et al.* propose a task redirection algorithm to balance workloads among edge cloudlets and show that the load balancing scheme can significantly reduce the service response time of edge cloudlets [43]. Tong *et al.* propose a hierarchical architecture for edge clouds and design a heuristic workload dispatch algorithm to minimize the average program execution delay by adaptively placing workloads among different tires of servers [11]. Tan *et al.* propose an online job dispatching and scheduling algorithm to minimize total weighted service response time in edge clouds. [44]. These existing edge cloud load balancing solutions only focus on reducing the service latency of users. However, the analytics accuracy is as important as the service latency for MAR. Therefore, mitigating the tradeoff between the service latency and analytics accuracy is indispensable in designing workload dispatching algorithms for edge-assisted MAR systems.

2.3 Existing Research on Multiple Domain Virtualization and Orchestration

As a key technology in 5G, the cellular network virtualization has attracted many research efforts [45–47]. Kokku *et. al.* [48] proposed a network virtualization substrate (NVS) which virtualizes uplink and downlink radio resources into slices in WiMAX networks. They also designed CellSlice which slices the radio resources at the gateway level without modifying the MAC scheduler in base stations [45]. Foukas *et. al.* [46] developed FlexRAN which decouples the control and data plane of eNodeB in LTE and provides a programmable control plane for managing radio access networks. The authors also engineered the Orion system that realizes on-the-fly radio access network slicing while maintaining the functional and performance isolation among network slices [29]. Salvat *et al.* [33] developed an end-to-end resource orchestration system, formulated an orchestration problem to maximize the revenue in network slicing, and proposed an optimal Benders decomposition method and a heuristic method. However, the fundamental assumption of these works is that the resource demands of slices and their performance modelings are known as closed-form mathematical expressions to the network operator, which might not be obtained under the complicated and high-dim performance requirements of network slices in complex mobile networks.

2.4 Existing Research on Distributed Cross-Domain Resource Orchestration

Network slicing has attracted extensive research attention from industry and academia [46, 49, 50]. Foukas *et. al.* designed FlexRAN [46] to decouple the control and user plane of LTE, which provides a customized API and programmable control plane for the radio access network management. Han *et. al.* [51] proposed a utility-based admission control mechanism based on multi-queuing systems to improve the resource efficiency for accommodating heterogeneous slices in network slicing. Ghodsi *et. al.* [52] proposed a dominant resource fairness (DRF) algorithm which allocates multi-domain

resources to ensure the max-min fairness among the tenants. Halabian *et al.* [49] showed that non-collaborative slices in the system compromise the fairness performance when maximizing the overall system performance and proposed a distributed solution. Chowdhury *et. al.* [53] proposed a high utilization with guarantees (HUG) algorithm that is an extension of DRF for handling elastic resource demands. Liu *et. al.* [54] designed a multi-domain resource allocation algorithm which maximizes the utility of the system in mobile cloud computing using a Markov decision process. However, none of these works addresses the distributed cross-domain resource orchestration for network slicing in large-scale mobile networks.

2.5 Existing Research on End-to-End Network Slicing with Decentralized DRL

The resource management problem with machine learning techniques in network slicing targets to maximize system performance. Caballero *et al.* [55] constructed a network slicing game in which tenants are selfish to maximize its own performance. The authors proved that this game with such strategic behavior converges to a Nash equilibrium for elastic traffic. To exploit the statistical multiplexing gain of slices, Sciancalepore *et al.* [56] designed STORNS that optimizes the admission control of slices with considering per slice SLA requirement by leveraging stochastic geometry theory. Mao *et al.* [57] designed DeepRM with the deep Q-network technique to optimize the admission control and resource orchestration of users. They obtained a considerable reduction in the average slowdown of user tasks as compared to heuristic solutions. Xu *et al.* [58] utilized the state-of-the-art deep deterministic policy gradient (DDPG) technique to solve the traffic engineering (TE) networking problem, i.e., allocating the bandwidth of network links, and obtained significant end-to-end latency reduction and performance improvement under the unknown performance function. Bega *et al.* [59] proposed DeepCog with deep learning techniques to forecast the network capacity within an individual slice and achieve the balance between resource over-provisioning and service request violations. Yang *et al.* [60] proposed

an adaptive reinforcement learning-based approach for a microservice workflow system that enables model-free resource allocation and improves the response time of microservices. However, these works advocate the centralization of resource management by using a central DRL agent, which cannot adapt to the dynamic typologies and architectures in wireless edge computing networks.

CHAPTER 3: DARE: DYNAMIC ADAPTIVE MOBILE AUGMENTED REALITY WITH EDGE COMPUTING

In this chapter, we design the DARE (dynamic adaptive AR over the edge) protocol that enables the edge-based MAR system to dynamically change AR configurations and computation resource allocations according to wireless channel conditions and available computation resources on the edge server. In the adaptation, the DARE protocol trades off the quality of augmentation (QoA) against the service latency. The intuitive basis of DARE is that performing precise AR (high QoA) requires massive data offloading and intensive computing while allowing selective approximation (medium QoA) can provide disproportionate gains in efficiency.

3.1 DARE Overview

Fig. 3.1 shows the overview of the DARE protocol. In the first step, MAR clients send their service requests and measurements of wireless channel conditions and service latency to an edge server. In the second step, upon receiving the requests and measurements, the optimization engine on the edge server determines the video frame sizes of MAR clients, selects computation models, which are object detection algorithms such as YOLO and SSD [13,14], to serve different MAR clients, and optimizes the computation resource allocation by mapping the computation requests of individual MAR clients to GPUs. The video frame sizes determined by the edge server is sent back to corresponding MAR clients as AR configuration messages. In the third step, MAR clients resize their video frame sizes according to the AR configuration messages. Meanwhile, MAR clients also adapt their video frame rate based on the service latency. After the frame resizing and frame rate adaptation, video frames are

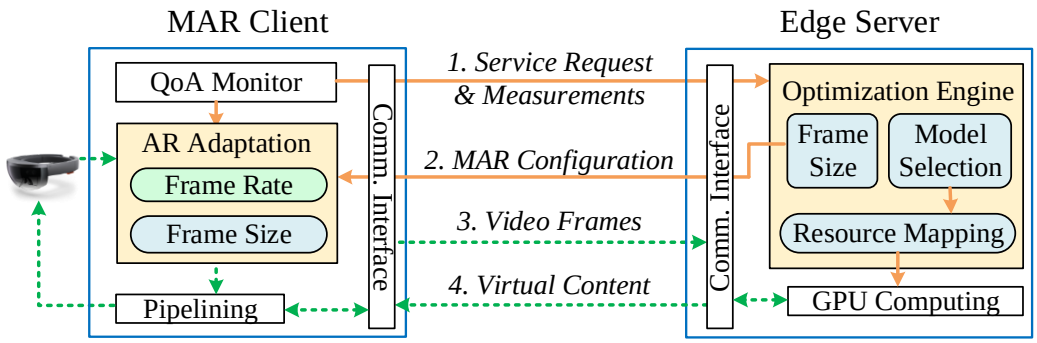

Figure 3.1: The overview of the DARE protocol.

sent to the edge server for processing. In the fourth step, the edge server sends back the virtual content that augments reality.

With the DARE protocol, the MAR clients can adapt their frame rates and video frame sizes. The edge server can change its computation models and resource allocations for different clients. Therefore, on designing the DARE protocol, we need to optimize four parameters in the edge-based MAR system: frame rate, video frame size, computation model selection, and computation resource allocation. Since the video frame sizes are closely related to the computation model and resource allocation, we design an optimization engine on the edge server to jointly optimize the video frame size, computation model selection, and computation resource allocation. The optimal video frame sizes are fed back to the corresponding MAR clients through the AR configuration message. The MAR clients are responsible for optimizing their own video frame rate.

The technical challenges for designing the DARE protocol are as follows: 1) there are no analytical models for characterizing the impact of the image data size and computation model on QoA and the service latency for a multiuser MAR system. 2) Since the edge server is shared by multiple MAR clients, individual clients' AR configurations are coupled with the computation resource allocation on the edge server. Such coupling makes it computationally hard to optimally allocate computation resources and adapt clients' AR configurations.

To address these challenges, we perform experiments to study the tradeoffs between QoA and the service latency in a MAR system. Based on the experiment results, we model QoA and the service latency as functions of the video frame size and computation model. We then formulate the AR reconfiguration and computation resource allocation in the edge-based MAR system as an optimization problem that aims to maximize the average QoA while satisfying all clients' latency requirements. We solve the problem using the cyclic block coordinate gradient projection method and implement this solution on the edge server. On the client side, we design and implement a frame rate adaptation mechanism that adapts the number of AR video frames sent to the edge server per second according to workloads on the edge server. This mechanism helps the client to maximize the amount of virtual content obtained from the edge server. We implement the DARE protocol and evaluate its performance through experiments.

3.2 Tradeoffs in Edge-based MAR System

In this section, we perform experiments to characterize the impact of video frame sizes and computation models, i.e., object detection algorithms, on QoA and the service latency for a multiuser MAR system. These experiment results not only provide the basis for modeling the adaptive MAR system but also make the case for it. Note that we focus on the MAR application in which the MAR client captures the environment information via cameras and sends the information to a server for object detection. The concept of dynamic adaptive AR can be generalized to other types of AR applications such as surface detection and rendering.

3.2.1 Video Frame Size v.s. QoA v.s. Latency

In this experiment, we implement YOLO 544x544 [14], which is the YOLO algorithm tuned at the 544x544 image resolution, as the object detection algorithm on a workstation with an NVIDIA Quadro M4000 GPU for the computation acceleration.

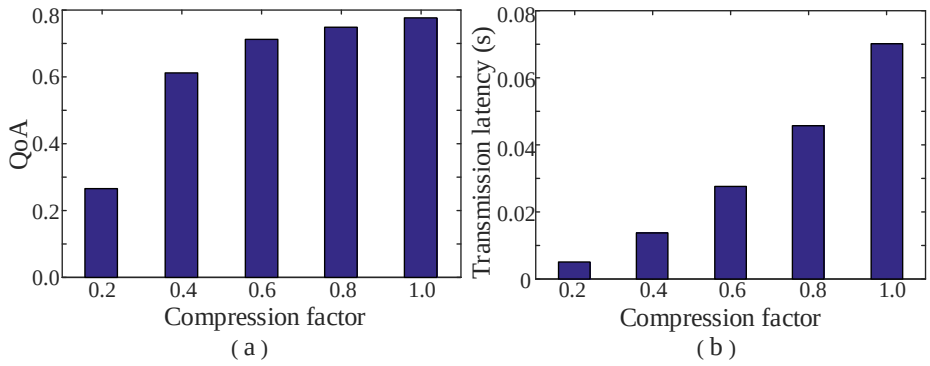

Figure 3.2: The QoA and latency vs. compression factor.

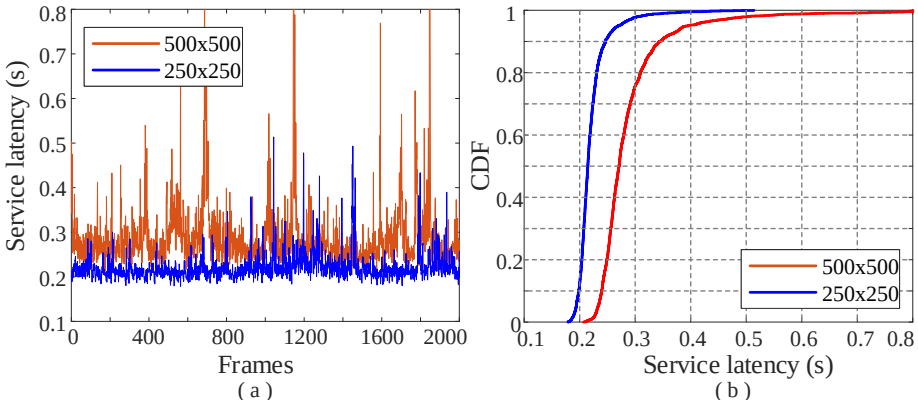

Figure 3.3: The service latency and jitter.

A MAR client is emulated in a laptop which connects to the workstation via a wireless router. We use *VOC 2007 test* dataset [18] for the study. To evaluate the impact of video frame sizes on QoA and the service latency, we preprocess the images in the dataset with five compression factors. For instance, when the compression factor is 0.2, an image with 500x500 pixels is resized to an image with 100x100 pixels.

Fig. 3.2 shows the impact of image data sizes on QoA and the service latency. As expected, a large video frame size leads to a high mAP but a long transmission latency. The gain on mAP becomes smaller as the increase of image data sizes. However, the transmission latency increases much faster with a larger video frame size. When the compression factor decreases from 1.0 to 0.6, the transmission latency decrease about 61% while mAP only drops about 8%. This result supports adaptive AR in trading mAP for the latency reduction.

Transmitting high-fidelity image data, i.e., a large video frame size, introduces a long latency and large jitters under dynamic wireless channel conditions. Fig. 3.3 shows the service latency and jitter of a MAR client with two different video frame sizes under varying wireless channel conditions. As wireless channel condition changes, there are many large spikes in the service latency measurement when transmitting images with 500x500 resolution. These spikes indicate large jitters in the MAR service. The latency jitter is alleviated when the image data size is reduced to 250x250 pixels. Besides, with a small image data size, the service latency of 97.8% images is less than 0.3 s. With a large image size, only about 75.6% images experience less than 0.3 s service latency.

3.2.2 Computation Model v.s. QoA v.s. Latency

In this experiment, we implement four object detection algorithms based on the YOLO framework [14] and two object detection algorithms based on the SSD framework [13] on the edge server. We use the *VOC 2007 test* dataset [18] to evaluate mAP of the object detection and the corresponding computation latency. Fig. 3.4 shows the mAP and the computation latency of these algorithms in a single user MAR system. The computation model that provides a high mAP usually incurs a long computation latency. The gain of mAP and the increase of the computation latency are disproportional. For example, as compared with YOLO 544x544, SSD 512x512 gains about 2.7% mAP but increases about 60% service latency. This result advocates adapting the computation model for reducing the service latency.

Since the number of MAR clients are dynamic, we measure the computation latency of different computation models when serving multiple users. Fig. 3.5 shows that the computation latency increases with the number of users because the GPU computation resources are shared among the users. However, a low-complexity model, i.e., YOLO 416x416, maintains a slower computation latency increase than a high-complexity model, i.e, YOLO 544x544. These observations provide us two insights

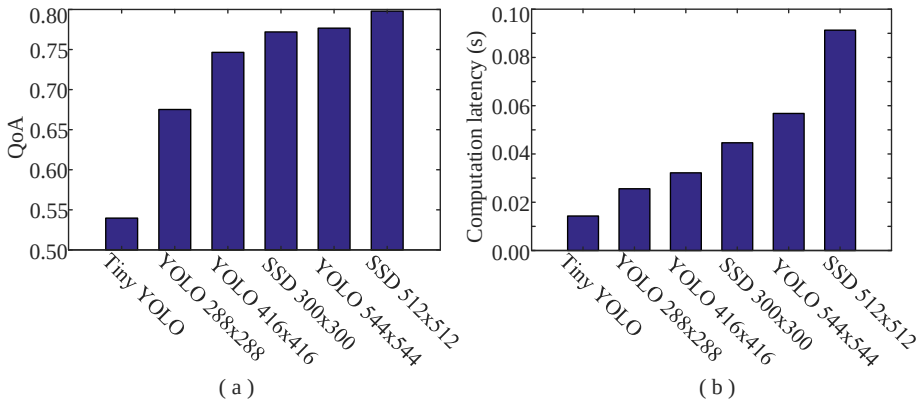

Figure 3.4: The QoA and latency v.s. the computation models.

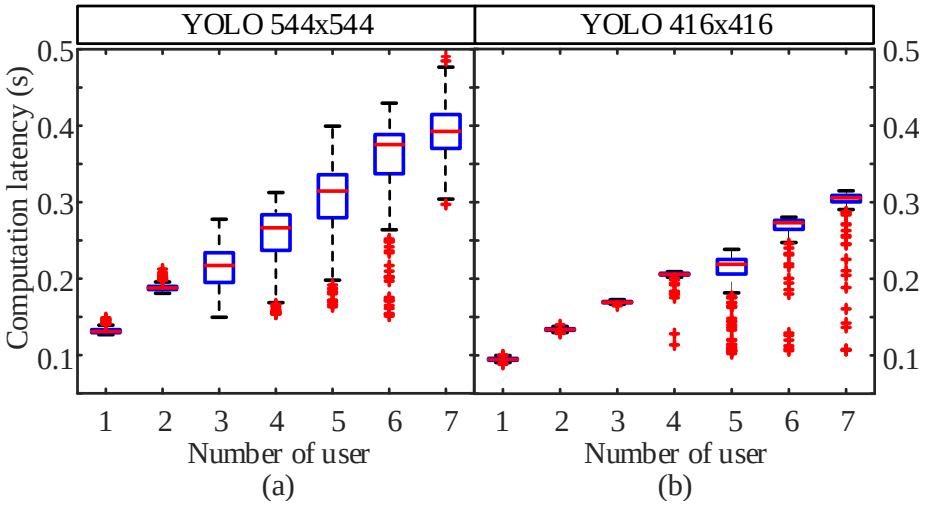

Figure 3.5: Computation model v.s. latency v.s. number of user.

on designing the DARE protocol. The first one is that optimizing the computation resource sharing among users may improve the performance of MAR. The second one is that adapting to a low-complexity computation model can effectively reduce the computation latency when there are multiple MAR clients in the system.

3.3 Optimization Engine on Edge Server

In this section, we describe system models for analyzing the adaptive MAR system, formulate the MAR reconfiguration and computation resource allocation on the edge server as an optimization problem, and propose an adaptive MAR algorithm to solve the problem. On modeling the system, we consider multiple MAR clients and one

edge server. The MAR clients are connected to the edge server via a wireless access point.

3.3.1 System Model

Quality of Augmentation (QoA): Denote the d_k, l_k, Q_k, and $\mathcal{K}$ as the image compression factor, mAP of the computation model, QoA of the kth MAR client, and the set of MAR clients, respectively. Here, we use the mAP value to represent a computation model. That is, given a mAP, we find the computation model that produces the mAP. Since QoA of a MAR client closely relates to the video frame size and the computation model, Q_k can be expressed as $Q_k = \mathcal{Q}(d_k, l_k)$ where $\mathcal{Q}(x, y)$ is a function of x and y. Hence, the average QoA of MAR clients in the system is modeled as

$$Q = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} Q_k. \tag{3.1}$$

Although the computation models have the frame resolution embedded, e.g., YOLO 544x544, we consider the computation model, i.e., mAP, and frame resolution, i.e., image compression factor, as two separate parameters for two reasons. First, given a computation model, the compression factor affects QoA of MAR services as shown in Fig. 3.2. Second, the number of computation models that can be supported in a practical system is limited by the computation resources. Therefore, the corresponding mAPs are discrete. However, the compression factor is continuous and selected by MAR clients according to the system performance.

Service Latency: We model the service latency as the static latency and dynamic latency. The static latency does not change when we adapt the configurations of MAR. For example, the latency of the communication link establishment and image preprocessing are considered as the static latency in a MAR system. The dynamic latency depends on the image data size and computation model of a MAR system. In our system, we consider the transmission latency in transmitting image data and com-

putation latency in performing the object detection as the dynamic latency. Hence, the service latency of the kth MAR client is expressed as

$$T_k = T_k^t + T_k^p + T^c \tag{3.2}$$

where T_k^t is the transmission latency, T_k^p is the computation latency, and T^c is the static latency in the system. In a practical system, the transmission latency consists of the wireless transmission latency between mobile clients and wireless access points and the wired transmission latency between wireless access points to edge servers. Since the wireless transmission latency is expected to be much longer and more dynamic than the wired transmission latency, we focus on the impact of the wireless transmission latency on the QoA of MAR services. Denote r_k as the wireless data rate of the kth MAR client. Let $\mathcal{S}(d_k)$, which is a non-decreasing function with respect to d_k, be the image data size of the kth client. Then, $T_k^t = \mathcal{S}(d_k)/r_k$.

The computation latency depends on the computational complexity of the computation model and available computation resources on the edge server [61]. Let c_k be the computational complexity of the kth client's computation model. We define f_k as the computation resources allocated to the kth client by the edge server. Then, the computation latency experienced by the kth client can be modeled as $T_k^p = \mathcal{P}(l_k)/f_k$. Here, $\mathcal{P}(l_k)$, which is a function of l_k, represents the computational complexity of the client's request.

3.3.2 Problem Formulation

On designing the optimization engine, we aim to maximize the average QoA while satisfying the latency requirements of the MAR clients in the system. The variables

are d_k, l_k, and $f_k, \forall k \in \mathcal{K}$. The optimization problem can be formulated as

$$\begin{aligned}
\mathscr{P}_0: \quad & \max_{\{d_k, l_k, f_k, \forall k \in \mathcal{K}\}} \quad Q = \tfrac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} Q_k \\
& s.t. \\
& C_1: \qquad T_k \leq T_k^{max}, \forall k \in \mathcal{K}; \\
& C_2: \qquad \textstyle\sum_{k \in \mathcal{K}} f_k \leq F^{\max}; \\
& C_3: \qquad d^{\min} \leq d_k \leq d^{\max}, \forall k \in \mathcal{K}; \\
& C_4: \qquad l_k \in \{l^{\min}, ..., l^{\max}\}, \forall k \in \mathcal{K};
\end{aligned} \tag{3.3}$$

where T_k^{max} is the maximum tolerable latency of the kth client, and F^{max} is the total computation resources on the edge server. As the emergence of the network slicing technology, it is reasonable to assume that a network slice is created for supporting a particular service, e.g., the MAR service [27]. Hence, we consider the edge server is dedicated to the MAR service and assume that the total computation resource for the MAR service is known as F^{max}. The constraints C_1 guarantee that the service latency of users are not larger than their maximum tolerable latency; the constraint C_2 means that the allocated computation resources do not exceed the total computation resources on the edge server; the constraints C_3 and C_4 are the constraints of the image compression factor and mAP of the computation model.

Note that l_k is a discrete variable. The values of l_k depend on the available computation models in the system. In other words, each computation model corresponds to a value of l_k. Therefore, deciding l_k equals to selecting the computation model. Hence, l_k acts as an integer variable that selects the computation models in the optimization. As shown in the experiment results in Sec. 3.2, QoA of a MAR client is a nonlinear function of the image data size. Thus, the above optimization problem is a mixed-integer non-linear programming problem (MINLP) which is difficult to solve [62].

3.3.3 Optimization Algorithm

We develop an adaptive MAR optimization algorithm to efficiently solve the above problem $\mathscr{P}_0$. According to the solution, we determine the AR video frame compression factor and select computation models for individual clients.

To solve problem $\mathscr{P}_0$, we relax the discrete variable l_k into continuous variable $\widehat{l}_k$. The problem is relaxed as

$$\begin{aligned} \mathscr{P}_1: \quad & \max_{\{d_k, \widehat{l}_k, f_k, \forall k \in \mathcal{K}\}} && Q \\ & s.t. && C_1, C_2, C_3 \\ & \widehat{C}_4: && l^{\min} \leq \widehat{l}_k \leq l^{\max}, \forall k \in \mathcal{K}. \end{aligned} \tag{3.4}$$

According to the observations in Sec. 3.2, the objective function Q is non-decreasing with respect to d_k and l_k. Therefore, we adopt the cyclic block coordinate gradient projection (CBGP) method to solve problem $\mathscr{P}_1$ [63]. According to the method, we solve problem $\mathscr{P}_1$ by fixing two of three variables and deriving the remaining one. We iterate the process until the value of each variable converges.

Denote $\nabla y(x)$ as the partial derivative of function y corresponding to variable x. Define $P_\Omega(x) = \arg\min_{y \in \Omega} \|x - y\|^2$ as the Euclidean projection of x on Ω. The procedures of the solution can be summarized as follow:

- Given $\widehat{l}_k$ and f_k, we update d_k according to

$$d_k^{(j+1)} = P_{\Omega_d}\left(d_k^{(j)} + \alpha_k \nabla Q_k\left(d_k^{(j)}\right)\right), \forall k \in \mathcal{K}; \tag{3.5}$$

  where $\alpha_k > 0$ is a constant step size and Ω_d is the bounded domain constrained by C_3.

- Given d_k and f_k, we update $\widehat{l}_k$ according to

$$\widehat{l}_k^{(j+1)} = P_{\Omega_{\widehat{l}}}\left(\widehat{l}_k^{(j)} + \beta_k \nabla Q_k\left(\widehat{l}_k^{(j)}\right)\right), \forall k \in \mathcal{K}; \tag{3.6}$$

where $\beta_k > 0$ is a constant step size and $\Omega_{\widehat{l}}$ is the bounded domain constrained by $\widehat{C}_4$.

- Given $\widehat{l}_k$ and d_k, the problem is simplified to

$$\begin{aligned} \max_{\{f_k, \forall k \in \mathcal{K}\}} \quad & Q \\ s.t. \quad & C_1 : T_k \leq T_k^{max}, \forall k \in \mathcal{K}; \\ & C_2 : \textstyle\sum_{k \in \mathcal{K}} f_k \leq F^{\max}; \end{aligned} \tag{3.7}$$

where constraints C_3 and $\widehat{C}_4$ are irrelevant to this problem.

We utilize the Lagrangian dual decomposition method to solve the above problem. The Lagrangian function is

$$\begin{aligned} \mathbb{L}(f_k, \lambda, \mu) \quad & = Q + \mu \left(\textstyle\sum_{k \in \mathcal{K}} f_k - F^{\max} \right) \\ & + \sum_{k \in \mathcal{K}} \lambda_k \left(\frac{\mathcal{S}(d_k)}{r_k} + \frac{\mathcal{P}(l_k)}{f_k} + T^c - T_k^{\max} \right) \end{aligned} \tag{3.8}$$

where λ and μ are the Lagrange multipliers corresponding to constraints C_1 and C_2, respectively. Then, the Lagrangian dual problem is expressed as

$$\begin{aligned} \min_{\{\lambda, \mu\}} \quad & g(\lambda, \mu) = \max_{\{f_k, \forall k \in \mathcal{K}\}} \mathbb{L}(f_k, \lambda, \mu) \\ s.t. \quad & \lambda \geq 0, \mu \geq 0. \end{aligned} \tag{3.9}$$

Here, $g(\lambda, \mu)$ is convex with respect to f_k. Based on the Karush-Kuhn-Tucker (KKT) condition [64], the optimal computation resource allocation for the kth client can be expressed as

$$f_k^* = \sqrt{\frac{\lambda_k}{\mu} \mathcal{P}(l_k)}. \tag{3.10}$$

Next, we use the sub-gradient method [64] to solve the dual problem. Based on the

sub-gradient method, the dual variables in the $(j+1)$th iteration are updated as

$$\lambda_k^{(j+1)} = \left[\lambda_k^{(j)} - \delta_k^{\lambda} \times \nabla g(\lambda_k^{(j)})\right]^+, \forall k \in \mathcal{K} \tag{3.11}$$

$$\mu_k^{(j+1)} = \left[\mu^{(j)} - \delta^{\mu} \times \nabla g(\mu^{(j)})\right]^+, \forall k \in \mathcal{K} \tag{3.12}$$

where $\delta_k^{\lambda} > 0$ and $\delta^{\mu} > 0$ are the step sizes, and $[x]^+ = \max\{0, x\}$.

With these mathematical analysis, we develop an adaptive MAR optimization algorithm that dynamically determines the AR image compression factor, selects the computation models, and allocates computation resources on the edge server. The pseudo code of the algorithm is presented in Algorithm 1. The algorithm is initialized with the simplest computation model and evenly shared computation resources among MAR clients. We then iteratively obtain d_k, $\widehat{l_k}$, and f_k until the algorithm converges. Since $\widehat{l_k}$ is a relaxed mAP of the computation model, it may not match the mAP of any installed computation models in a real system. In this case, we select the computation model whose mAP, l_k^*, is most close to the relaxed mAP (line 9 in the pseudo code of the algorithm). The adaptive MAR algorithm is developed based on the CBGP method, and hence follows the convergence results in [63], we claim that the algorithm converges to a local optimal solution.

3.3.4 GPU Computation Resource Allocation

Although the adaptive MAR optimization algorithm obtains the computation resource allocations for individual clients, it is not trivial to enforce the GPU resource allocation in the system. There are some techniques for virtualizing a physical GPU into multiple virtual GPUs (vGPU) so that the GPU resources can be shared among multiple clients [65]. However, the GPU virtualization incurs the computation overhead and delay. Since the adaptive MAR optimization algorithm runs on a small time scale, e.g., every second, it is inefficient to dynamic generate vGPUs according to the optimization results. Therefore, instead of reconfiguring vGPUs, we assign clients'

Algorithm 1: Adaptive MAR Optimization Algorithm

Input: T_k^{max}, F^{max} and r_k, $\forall k \in \mathcal{K}$.

Output: d_k, $\widehat{l}_k$ and f_k, $\forall k \in \mathcal{K}$.

$f_k \leftarrow F^{max}/|\mathcal{K}|$, $\forall k \in \mathcal{K}$;

$\widehat{l}_k \leftarrow l^{\min}$, $\forall k \in \mathcal{K}$;

while *True* **do**

  $d_k^{(j+1)} = P_{\Omega_d}\left(d_k^{(j)} + \alpha_k \nabla Q_k\left(d_k^{(j)}\right)\right), \forall k \in \mathcal{K}$;

  $\widehat{l}_k^{(j+1)} = P_{\Omega_{\widehat{l}}}\left(\widehat{l}_k^{(j)} + \beta_k \nabla Q_k\left(\widehat{l}_k^{(j)}\right)\right), \forall k \in \mathcal{K}$;

  $f_k \leftarrow$ solving $\mathscr{P}_1$ with fixed d_k and $\widehat{l}_k$, $\forall k \in \mathcal{K}$;

  if *convergence* **then**

    break;

$l_k^* = \underset{l \in \{l^{\min}, \ldots, l^{\max}\}}{\arg\min} \left| l - \widehat{l}_k \right|, \forall k \in \mathcal{K}$;

return d_k, l_k^* *and* f_k, $\forall k \in \mathcal{K}$.

requests to pre-configured vGPUs according to the optimization results.

Denote g_i and $\mathcal{I}$ as the total computation resources of the ith vGPU and the set of vGPUs, respectively. Let f_k be the computation resources allocated to the kth client. We formulate the client-vGPU mapping problem as

$$\begin{aligned} \min_{\{a_{i,k}, \forall k \in \mathcal{K}, i \in \mathcal{I}\}} \quad & \sum_{i \in \mathcal{I}} (g_i - \sum_{k \in \mathcal{K}} a_{i,k} f_k) \\ s.t. \quad & \\ & C_1: \ g_i - \sum_{k \in \mathcal{K}} a_{i,k} f_k \geq 0, \forall i \in \mathcal{I}; \\ & C_2: \ \sum_{i \in \mathcal{I}} a_{i,k} = 1, \forall k \in \mathcal{K}; \\ & C_3: \ a_{i,k} = \{0, 1\}, \forall i \in \mathcal{I}, k \in \mathcal{K}; \end{aligned} \tag{3.13}$$

where $a_{i,k}$ is an indicator function. If the kth client is assigned to the ith vGPU, $a_{i,k} = 1$; otherwise, $a_{i,k} = 0$. The client-vGPU mapping problem is equivalent to a bin packing problem which is NP-complete. We solve this problem with a greedy approximation algorithm whose pseudo code is shown in Alg. 2. In this algorithm, we sort f_k and g_i based on their values from the largest to the smallest, respectively.

Algorithm 2: Greedy Client-vGPU Mapping

Input: f_k and g_i, $\forall k \in \mathcal{K}, i \in \mathcal{I}$.
Output: $a_{i,k}$, $\forall k \in \mathcal{K}, i \in \mathcal{I}$.

```
Sort f_k and g_i from the largest to the smallest;
a_{i,k} ← 0, ∀k ∈ K, i ∈ I;
for k = 1 : |K| do
    for i = 1 : |I| do
        if f_k ≤ g_i then
            a_{i,k} = 1 and g_i = g_i − f_k;
            break;
    if Σ_{i∈I} a_{i,k} = 0 then
        Set a_{j,k} = 1 where j = arg max_{i∈I} g_i;
        g_i = g_i − f_k;
return a_{i,k}.
```

Then, we sequentially assign a client to the vGPU which has sufficient computation resources to serve the client. If no vGPU can serve the client, the algorithm assigns the client to the vGPU with the most residual computation resources.

3.4 Frame Rate Adaptation on Mobile Client

The impact of the AR frame rate on the system performance is implicitly considered in the optimization problem $\mathscr{P}_0$. We assume that the MAR client continuously sends video frames to the edge server. Therefore, the frame rate is determined by the service latency, i.e., the frame rate of the kth client $\omega_k = 1/T_k$. However, based on this frame rate, the computation resources allocated to the kth client are not fully utilized because the server has to wait for the frames from the client. To solve this problem, we design a frame rate adaptation mechanism on the MAR client to perform the traffic flow control in the system. This mechanism enables the MAR client to fully utilize the allocated computing resources on the edge server and helps them to increase the amount of virtual content obtained from the server without impairing the service latency.

As shown in Fig. 3.6, the frame rate adaptation determines how many AR video

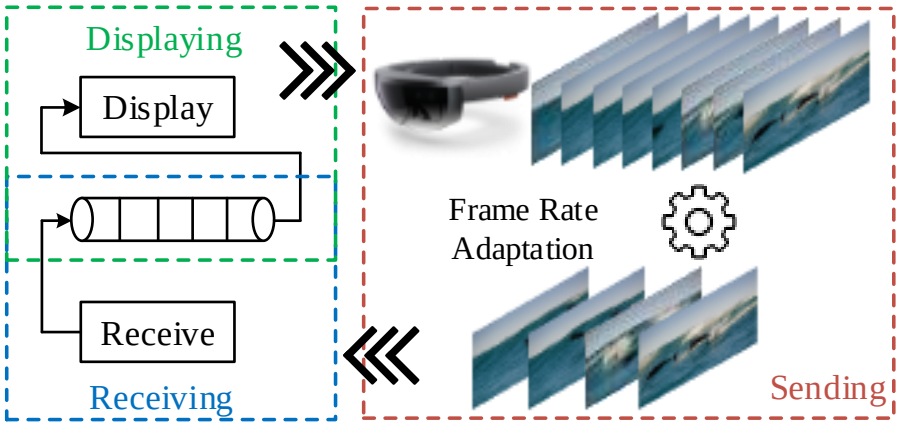

Figure 3.6: Frame rate adaptation and transceiving pipeline.

frames are sent to the edge server per second. With a higher video frame rate, the MAR client can send more frames to the edge server and thus obtain more virtual content. However, if the video frame rate is too high, sending and processing these frames may lead to the traffic congestion in wireless transmissions and the computation congestion on edge server. On the other hand, if the video frame rate is too low, the server waits for the frames from MAR clients which results in a under-utilization of computing resources. Therefore, the frame rate adaptation mechanism determines the AR frame rate based on the estimation of the computation workloads on the edge server. Since the computation workloads are varying, the AR frame rate also changes over time. In the system, we adapt the AR frame rate based on the computation latency. Hence, the frame rate of the kth client is calculated as

$$\omega_k = \frac{1}{\gamma_k T_k^p} \tag{3.14}$$

where T_k^p is the computation latency of the kth client, and γ_k is a control factor determined by the network conditions and computation workloads. A transceiving pipeline is designed to ensure that the sampled AR video frames are sent to the server continuously without waiting for the server's response.

3.5 Performance Evaluation

In this section, we provide an extensive evaluation of the DARE protocol under varying wireless channel conditions and computation workloads.

3.5.1 Protocol Implementation

Edge Server: A database is implemented to store clients' information such as the minimum compression factors, wireless data rates and the latency requirements. This database will be updated based on the information received from the clients. The system periodically optimizes its performance and determines the AR configurations and computation resource allocations. Besides the periodical optimization, three events can trigger the optimization. They are 1) a newly connected client, 2) an optimization request from a client, and 3) a disconnected client. We establish long-standing sockets between clients and the edge server to exchange the control information such as the optimization request, network status report, and AR configurations.

Based on the client-vGPU mapping, clients are assigned to a vGPU using different socket ports. The computation resources on a vGPU are shared among the clients associated with the vGPU. We use multithreading to serve multiple clients simultaneously in the system. When a client connects to the system, a thread is created to serve the client. Using this thread, the edge server receives the client's video frames and pushes them into the computation queue. The tasks in the computation queue are scheduled according to the adaptive MAR optimization algorithm.

MAR Client: The QoA monitor on the MAR client is implemented to monitor the wireless channel conditions and the service latency. The QoA monitor will trigger the optimization request to the edge server if the wireless channel condition changes or the service latency exceeds $T_{\max}$. An optimization request contains the information about the wireless data rate, $T_{\max}$, and $d^{\min}$. We evaluate the wireless channel condition and service latency every second to avoid generating excessive optimization requests

Table 3.1: GPU details

	NVIDIA Quadro M4000	NVIDIA Quadro P2000
CUDA cores	1664	1024
Memory size	8 GB	5 GB
Boost Clock	800 MHz	1470MHz
Power	120 W	75 W
Peak FP32 Performance	2.66 TFLOPS	3.0 TFLOPS

Table 3.2: RSSI vs. Data Rate

RSSI (dBm)	-30	-40	-50	-60	-70
Data Rate (Mbits/s)	25	20	15	10	5

based on the service latency of an individual frame.

The wireless channel conditions of clients are important for optimizing the edge-based MAR system. To effectively estimate wireless data rates, we use the Linux wireless-tool *iwconfig* to fetch received signal strength indicator (RSSI) from the driver of the wireless network adapter. Then, we map the RSSI to the wireless data rate.

3.5.2 Evaluation Setup

We deploy the edge server on a workstation with an Intel Xeon E5-2630v4 2.2GHz CPU, Nvidia Quadro M4000 and P2000 GPUs, and 16GB RAM. The features of the GPUs installed in our edge server are detailed in Table 3.1. The MAR clients are deployed on laptops. We emulate the AR video using *VOC 2007 test* dataset [18] that contains 4952 images. Since we do not have video dataset for evaluating the mAP of different visual object detection algorithms, we construct a video with 4952 images in *VOC 2007 test* dataset to evaluate the proposed protocol. The images in the dataset are resized to 500x500 pixels. We use the resized images to generate AR video streams. The clients are connected to the edge server via a LinkSys WRT1900AC wireless router. In the experiments, the static latency T_c of the edge-based MAR system is 0.04 s. Based on measurements, we derive the mapping between RSSIs and wireless data rates as shown in Table 3.2.

Table 3.3: The mAP of computation models

Computation Models	288x	352x	416x	480x	544x
mAP	69.0	73.7	76.8	77.8	78.6

Table 3.4: Fitting Curves

Functions	Fitting Curve	RMSE
$\mathcal{Q}(d_k, l_k)$	$(-0.13d_k^2 + 0.3d_k + 0.62)\, l_k$	7.77×10^{-4}
$\mathcal{S}(d_k)$	$0.06d_k$	9.1×10^{-3}
$\mathcal{P}(l_k)$	$60.77l_k^2 - 86.1l_k + 30.71$	8.79×10^{-3}

On the edge server, we deployed five computation models based on the YOLO framework. These models are YOLO 288x288, YOLO 352x352, YOLO 416x416, YOLO 480x480, and YOLO 544x544. Here, YOLO 288x288 means that the YOLO object detection algorithm tuned at the image resolution of 288x288. The mAPs of these models are listed in Table 3.3 where we use 288x to represent YOLO 288x288. On the MAR client, the range of the image compression factors are from 0.4 to 1.0.

On implementing the optimization mechanism, the functions of $\mathcal{Q}(d_k, l_k)$, $\mathcal{S}(d_k)$, $\mathcal{P}(l_k)$ are derived based on the measurement data by using the curve fitting toolbox (*cftool 3.5.4*) in Matlab. These functions are listed in Table 3.4.

We compare the performance of the DARE protocol with two baseline protocols:

- **Max QoA**: this system uses the high-fidelity video data and high-complexity computation model to maximize mAP of the object detection for MAR clients.
- **Min Latency**: this system uses the low-resolution video frames and low-complexity computation model to minimize the service latency experienced by MAR clients.

3.5.3 Evaluation Results

Wireless channel conditions: To evaluate the performance of the DARE protocol under dynamic wireless channel conditions, we maintain a static workload on the edge server and carry the MAR client randomly walking in the laboratory. During the random walk, the MAR client experiences different wireless channel conditions. Fig. 3.7 compares the performance of the DARE protocol with two baseline protocols.

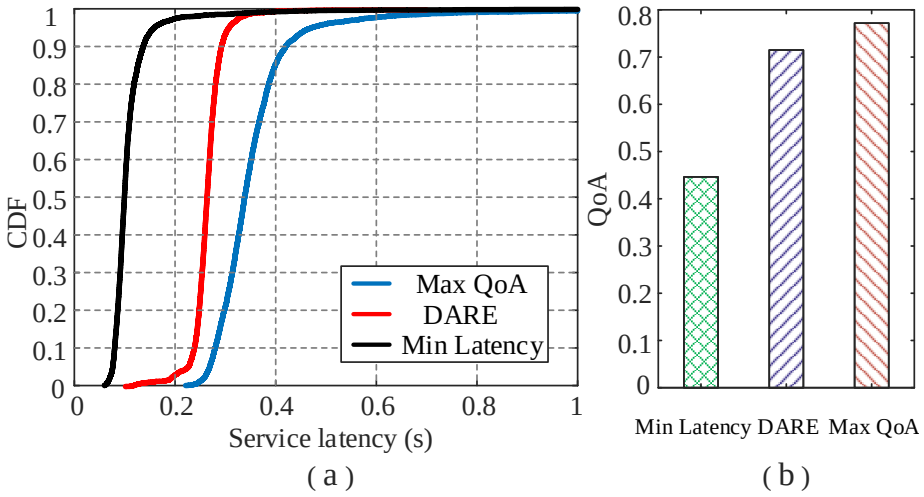

Figure 3.7: The comparison of the protocol performance under varying wireless channel conditions.

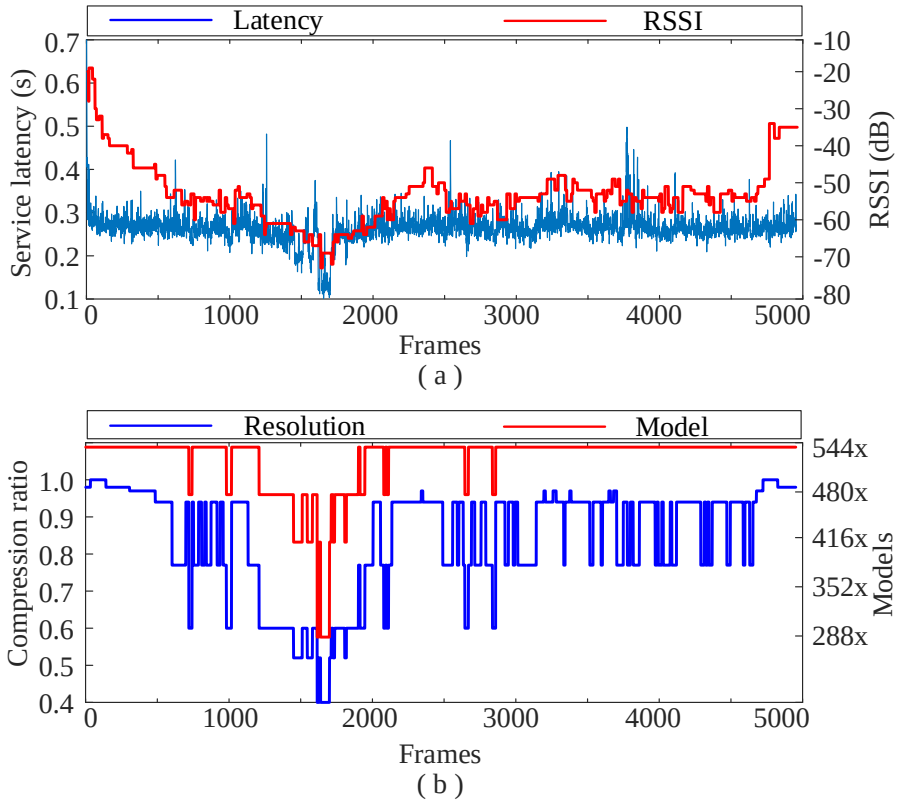

Figure 3.8: The adaptation to wireless channel conditions.

As shown in the figure, the Max QoA protocol has the longest service latency but provide the highest QoA. The Min Latency protocol minimizes the service latency at the cost of a significantly decreased QoA. For most of the frames, the DARE protocol

maintains the MAR client's minimum latency requirement which is 0.3 s in the experiment. At the same time, the DARE protocol maximizes the QoA of the MAR client considering various wireless channel conditions. The QoA achieved by the DARE protocol is only slightly smaller than that achieved by the Max QoA protocol.

Fig. 3.8 shows that the DARE protocol tracks the wireless channel conditions and adapts the video frame size and computation model accordingly. Fig. 3.8 (a) shows the wireless received signal strength indication (RSSI) and the corresponding service latency of the MAR client in the duration of 4952 AR video frames. When RSSI drops, there are a few latency spikes in the system. This is because the DARE protocol performs the optimization every second. If RSSI changes between two consecutive optimizations, the DARE protocol does not respond to the changes immediately. The optimization interval can be configured in the system. A small optimization interval improves the performance of the system in tracking wireless channel variations but also increases the optimization workloads on the server.

As shown in the Fig. 3.8 (b), the video frame size is adapted more frequently than the computation model. This observation indicates that the DARE protocol prefers to adapting the video frame size first when RSSI changes. For most of the frames, the DARE protocol uses the high-complexity computation model to ensure a high QoA and only degrades the computation when the wireless channel condition is very poor.

Server workloads: Fig. 3.9 shows the performance of the DARE protocol with a different number of MAR clients. In this experiment, the clients are stable so that their wireless channel conditions are almost static. When there are more clients in the system, the computation latency increases because all the clients share the limited computation resources on the server. For both the Max QoA and Min Latency protocols, their service latency almost linearly increases with the number of clients. The DARE protocol maintains a low service latency that satisfies the latency requirements (0.3 s) of MAR clients. The cost of maintaining the low latency is dropping

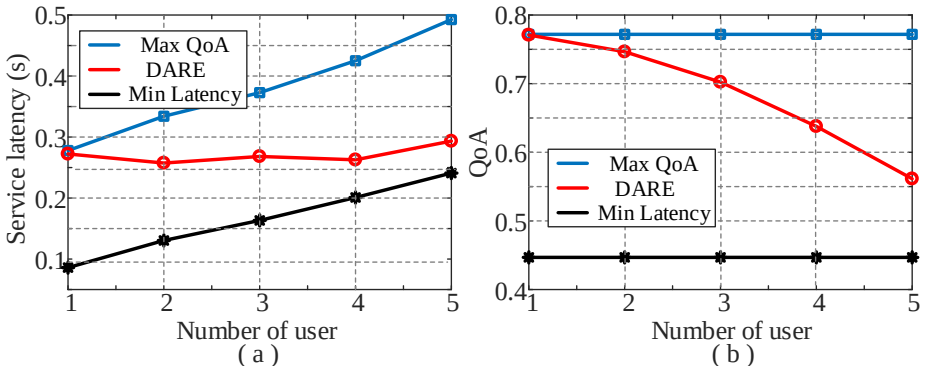

Figure 3.9: The protocol performance with different number of users.

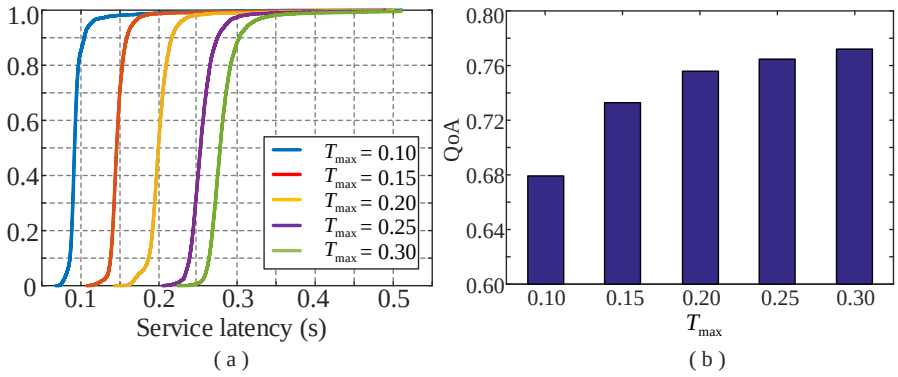

Figure 3.10: The performance of DARE with different T_{max}.

QoA of the object detection of MAR clients. However, the DARE protocol is able to mitigate the tradeoff between QoA and the service latency through optimizations. For example, when the number of users is 3, the DARE protocol trades about 0.06 QoA for about 31% latency reductions as compared with the Max QoA protocol.

Service latency requirements: In this experiment, we vary T_{max} to evaluate the performance of the DARE protocol under different latency requirements. Fig. 3.10 shows that for most of the time, the DARE protocol can satisfy the latency requirements of the mobile client. Under a stringent latency requirement, e.g., 0.1 s, the DARE protocol serves the client with a low QoA at about 0.68.

Number of computation models: The available computation modes on the server impact the performance of the adaptation. To study such an impact, we evaluate the performance of the DARE protocol under three settings of the computation

models. In the first setting, we install one computation model, YOLO 416x416, on the server. In the second setting, we install three computation models: YOLO 352x352, 416x416 and 480x480. In the third setting, we install five computation models: YOLO 288x288, 352x352, 416x416, 480x480 and 544x544.

The experiment results are shown in Fig. 3.11. When there is only one computation model on the edge server, the DARE protocol can only adjust the client's video frame size. Since the service latency is smaller than the latency requirement which is $T_{\max} = 0.2$ in the experiment, the DARE protocol adopts the largest video frame size to maximize QoA. As a result, under this scenario, the DARE protocol has a similar performance as the Max QoA protocol. The Min Latency protocol adopts the minimum image data size to minimize the server latency. Hence, its service latency is smaller, but its QoA is also lower than the DARE protocol and the Max QoA protocol.

When there are three computation models in the system, the service latency of the Max QoA protocol is still less than the latency requirement of the client (0.2 s). Hence, the DARE protocol adopts the maximum video frame size and the high-complexity computation model to maximize its QoA. When there are five computation models in the system, the service latency of the Max QoA protocol is larger than the latency requirement of the client (0.2 s). Under this scenario, the DARE protocol maximizes QoA with the constraint of the client's latency requirement. Therefore, the service latency of the DARE protocol nearly equals to 0.2 s, and the QoA is slightly less than that of the Max QoA protocol. For the Min Latency protocol, it always selects the low-complexity computation model with the smallest video frame size. Therefore, the service latency as well as QoA of the Min Latency protocol decreases when a computation model with a lower complexity is available in the system.

Video frame rate and pipelining: With the DARE protocol, the MAR client dynamically changes the AR frame rate according to workloads on the edge server.

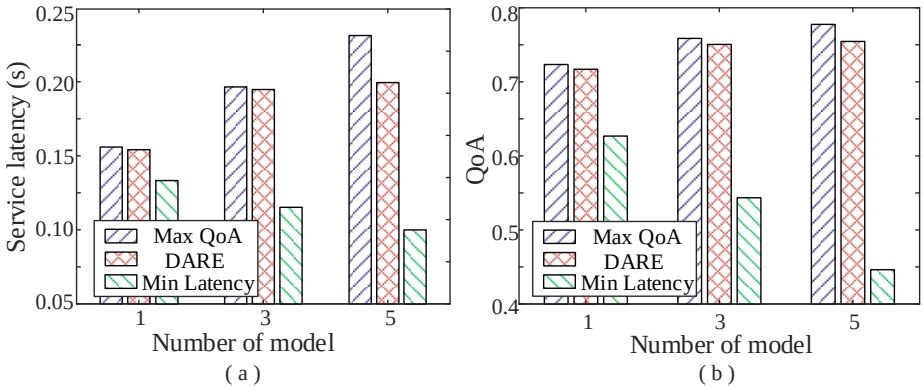

Figure 3.11: The protocol performance with different number of computation models.

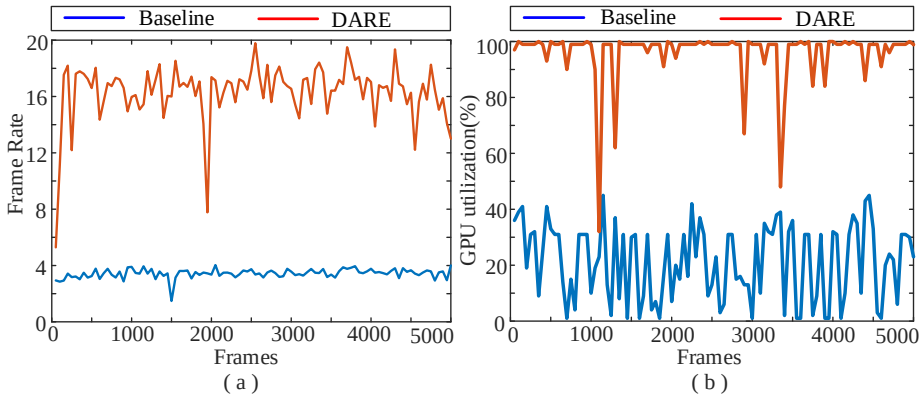

Figure 3.12: The performance of adaptive frame rate mechanism.

Fig. 3.12 shows the impact of the video frame rate on the server workload. In the experiment, the service latency is about 0.3s. Without the dynamic video frame rate mechanism (baseline), the AR video frames are sent after receiving the virtual content from the server. Therefore, the client can only send about 3 AR video frames to the server per second which is determined by the service latency. With the adaptive frame rate mechanism, the MAR client can send about 16 AR video frames per second on average to the server. In Fig. 3.12(b), with the adaptive frame rate mechanism on the MAR client, the GPU utilization on the server is almost 100%. This indicates this mechanism is able to exploit the allocated computation resources on the server to maximize the frame rate of the MAR client.

CHAPTER 4: AN EDGE NETWORK ORCHESTRATOR FOR MOBILE AUGMENTED REALITY

In this chapter, we design and implement an edge network orchestrator that enables fast and accurate object analytics in an edge-based MAR system. We model the network latency, computational latency, and analytics accuracy in an edge-based MAR system according to the performance measurements obtained from our MAR testbed. Then, we formulate a multi-objective optimization problem that aims to mitigate the tradeoff between the network latency, computation latency, and analytics accuracy by optimizing the edge server assignment and video frame resolution selection for MAR users. We develop a fast and accurate object analytics (FACT) algorithm which solves the multi-object optimization problem based on convex optimization theory. We evaluate the FACT algorithm through both network simulations and experiments with our edge-based MAR system implementation.

4.1 Analytical Model of Edge-based MAR System

In this section, we describe the system model for analyzing the edge-based MAR system. The system model includes network latency, computational latency, and analytics accuracy models. The computational latency and analytics accuracy models are derived based on the performance measurements obtained from our MAR testbed.

We consider a mobile edge network with K MAR users and N heterogeneous servers including both the cloud and edge servers. Denote $\mathcal{K}$ and $\mathcal{N}$ as the set of MAR users and servers, respectively. The MAR users communicate with servers via wireless access points such as cellular base stations and WiFi hotspots. The object analytics is performed on either edge servers or cloud server. The average service latency of

the kth MAR user can be defined as

$$L_k = L_k^w + L_k^t + L_k^p, \tag{4.1}$$

where L_k^w is the wireless latency incurred by sending a video frame from the kth user to its associated wireless access point; L_k^t is the core network latency caused by transferring the frame from the wireless access point to the server assigned to the user; and L_k^p is the computational latency of the object analytics on the server.

4.1.1 Network Latency Model

The network latency is composed of the wireless and core network latency. The wireless latency is determined by the user's video frame resolutions and wireless data rates. Since the data size of analytics results is usually small, we do not model the latency caused by transmitting the analytics results [61]. We assume that the AR video of the kth user is preprocessed into video frames with the resolution of $s_k \times s_k$ pixels. Here, we use s_k^2 (the number of pixels) to represent the video frame resolution of the kth MAR user. Denote σ as the number of bits required to represent the information carried by one pixel. Denote $\mathcal{S} = \{s_k^2 \,|k \in \mathcal{K}\}$ as the set of users' frame resolutions. The data size of a video frame is calculated as σs_k^2 bits. Let R_k be the average wireless data rate of the kth user. The wireless latency experienced by the kth user is modeled as

$$L_k^w = \frac{\sigma s_k^2}{R_k}. \tag{4.2}$$

Note that we consider the simplified wireless latency model (Eq. 4.2) because we focus on the system level performance rather than wireless link level performance.

Since the core network usually has very high transmission capacity, its latency is mainly determined by the aggregated traffic loads in the network and the geo-distance between the wireless access points and the servers. The impact of the video frame size of a single user on the link latency of the core network is negligible. Therefore, we do

not consider such an impact in our core network latency model. Denote $a_{k,n} \in \{0, 1\}$ as the server assignment indicator which indicates the kth user is served by the nth server if $a_{k,n} = 1$. Denote $\mathcal{A} = \{a_{k,n} | k \in \mathcal{K}, n \in \mathcal{N}\}$ as the set of users' server assignments. Here, we set $a_{k,n}$ as a binary variable to restrict that a user can be served by only one server at a time. Let $l_{k,n}$ be the core network latency between the kth user's associated wireless access point and the nth server, the core network latency of the kth user can be expressed as

$$L_k^t = \sum_{n \in \mathcal{N}} a_{k,n} l_{k,n}. \tag{4.3}$$

4.1.2 Computational Latency Model

The computational latency is closely related to the computational complexity of a user's task and available computational resources on servers [61]. Let c_k and f_n be the computational complexity of analyzing the kth user's video frame and the available computational resources on the nth server. We assume that the available computational resources on a server are evenly shared by the users associated with the server. Then, $f_n / \sum_{m \in \mathcal{K}} a_{m,n}$ is the computational resources allocated to one user on the nth server. Therefore, the computational latency experienced by the kth user can be modeled as

$$L_k^p = \sum_{n \in \mathcal{N}} a_{k,n} \frac{c_k}{f_n} \sum_{m \in \mathcal{K}} a_{m,n}. \tag{4.4}$$

In order to characterize the computational latency, we have to figure out the relationship between the computational complexity c_k and the video frame resolution $s_k \times s_k$. To do so, we implement two object recognition algorithms, YOLO [66] and SSD [67], on our workstation with Nvidia Quadro M4000 GPU. The original YOLO algorithm resizes incoming video frames to a predefined resolution, e.g., 448 x 448, before performing the object analytics. In order to measure the impact of the video

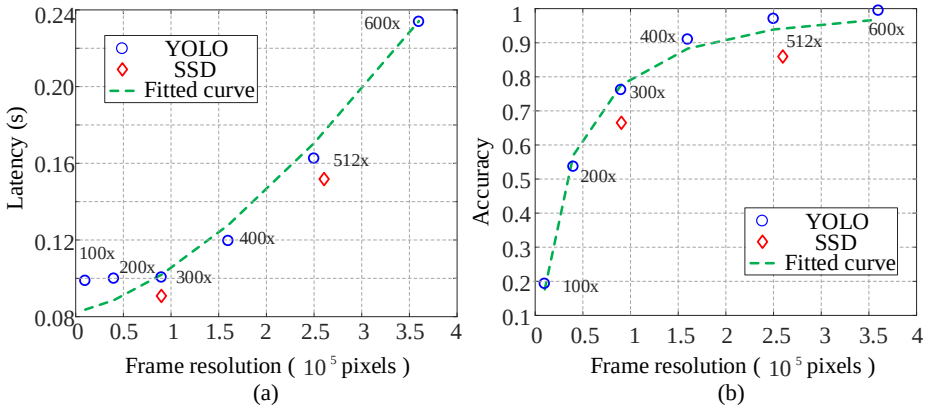

Figure 4.1: The latency and accuracy vs. the frame resolution.

frame resolution on the object analytics performance, we enable the YOLO algorithm to resize incoming video frames to different resolutions and perform the object analytics based on resized video frames. The computational latency measured in the experiments reflects the computational complexity of the object recognition under different video frame resolutions.

Fig. 4.1 (a) shows that the computational latency increases when the video frame resolution becomes higher. The speed of the latency increase becomes faster at a higher video frame resolution. Such a relationship between the computational complexity and the video frame resolution can be characterized by a convex function. For example, the measurement data can be fitted by a convex function $f(s_k^2) = 7 \times 10^{-10} {s_k}^3 + 0.083$ with the root mean square error (RMSE) of 0.01. Based on these observations, we model the computational complexity $c_k = \psi(s_k^2)$ where $\psi(s_k^2)$ is convex with respect to the video frame resolution s_k^2.

4.1.3 Analytics Accuracy Model

The analytics accuracy highly depends on the resolutions of the AR video frame. The higher video frame resolution usually results in a better mean average precision of an object recognition function [68]. Therefore, we model the analytics accuracy as a function of the video frame resolution. We define the analytics accuracy as the

ratio between the number of correctly recognized objects and that of total objects in a video frame. We build the analytics accuracy model based on the aforementioned performance measurement. On calculating the accuracy, we assume that the YOLO algorithm can detect all objects in a video frame when the video frame resolution is 600×600 pixels.

Fig. 4.1 (b) shows the analytics accuracy of the YOLO and SSD algorithm under different video frame resolutions. There are two observations. The first one that a higher video frame resolution enables a better analytics accuracy. The second observation is that the performance gain narrows down at a high video resolution. Based on these observations, we can use a concave function to define the relationship between the analytics accuracy and the video frame resolution. For example, the concave function $f(s_k^2) = 1 - 1.578e^{-6.5\times10^{-3}s_k}$ can fit the measurement data with less than 0.03 RMSE. Therefore, we model the analytics accuracy $A_k = \xi(s_k^2)$ where $\xi(s_k^2)$ is a concave function with respect to video frame resolution s_k^2.

4.1.4 Problem Formulation

Based on the analytical model, the total service latency of the MAR users is

$$L = \sum_{n\in\mathcal{N}} \sum_{k\in\mathcal{K}} \left[\frac{\sigma s_k^2}{R_k} + a_{k,n} \left(l_{k,n} + \frac{\psi(s_k^2)}{f_n} \sum_{m\in\mathcal{K}} a_{m,n} \right) \right], \tag{4.5}$$

and the summation of the analytics accuracy of the MAR users is

$$A = \sum_{k\in\mathcal{K}} \xi(s_k^2). \tag{4.6}$$

On designing the edge network orchestrator, we aim to minimize the overall service latency and maximize the total analytics accuracy of the MAR users. Therefore, designing the orchestrator is a multi-objective optimization problem [69]. There is a tradeoff between the service latency and analytics accuracy. In order to characterize

the tradeoff, we introduce a positive weight parameter β which reflects the preference between the service latency and analytics accuracy in optimizing the MAR system. We adopt the weighted sum method [70] to express the multi-object optimization problem as

$$\begin{aligned} \mathscr{P}1: \quad & \min_{\{\mathcal{A},\mathcal{S}\}} \quad F = L - \beta A \\ & s.t. \quad C_1: \ \xi(s_k^2) \geq \delta_k, \forall k \in \mathcal{K}, \\ & \qquad C_2: \ \textstyle\sum_{n\in\mathcal{N}} a_{k,n} = 1, \forall k \in \mathcal{K}, \\ & \qquad C_3: \ a_{k,n} \in \{0,1\}, \forall k \in \mathcal{K}, \forall n \in \mathcal{N} \end{aligned} \tag{4.7}$$

where δ_k is the minimum analytics accuracy requirement of the kth user; the constraints C_2 and C_3 ensure that an individual user is assigned to one and only one server. The weight parameter β controls the latency-accuracy tradeoff. For example, a larger β indicates that the MAR system prefers a higher accuracy. As a result, the optimal solution of Problem $\mathscr{P}1$ trades the average service latency for enhancing the analytics accuracy.

4.2 The FACT Algorithm

Problem $\mathscr{P}1$ is a mixed-integer nonlinear programming problem (MINLP) which is difficult to solve [62]. We develop the FACT algorithm to solve the problem based on the block coordinate descent method [71].

To solve Problem $\mathscr{P}1$, we relax binary variables $a_{k,n}$ to continuous variables $\tilde{a}_{k,n}$. Denote $\tilde{\mathcal{A}} = \{\tilde{a}_{k,n} \,|k \in \mathcal{K}, n \in \mathcal{N}\}$. The relaxed problem is

$$\begin{aligned} \mathscr{P}2: \quad & \min \qquad F = L - \beta A \\ & s.t. \quad C_1, C_2, 0 \leq \tilde{a}_{k,n} \leq 1. \end{aligned} \tag{4.8}$$

Lemma 1. *The Problem $\mathscr{P}2$ is strictly convex with respect to the relaxed server assignments $\tilde{\mathcal{A}}$.*

Proof: For any feasible $\tilde{a}_{m,n}, \tilde{a}_{i,j}, \forall m, i \in \mathcal{K}, \forall n, j \in \mathcal{N}$,

$$\frac{\partial^2 F}{\partial \tilde{a}_{i,j} \partial \tilde{a}_{m,n}} = \begin{cases} \frac{2\psi(s_i^2)}{f_j}, & i = m \text{ and } j = n, \\ 0, & i \neq m \text{ or } j \neq n, \end{cases} \tag{4.9}$$

The Hessian matrix $\mathbf{H} = \left(\frac{\partial^2 F}{\partial \tilde{a}_{i,j} \partial \tilde{a}_{m,n}}\right)_{KN \times KN}$ is symmetric and positive definite. The constraints C_2 and C_3 are linear. The constraints C_1 are irrelevant to $\tilde{\mathcal{A}}$. Therefore, $\mathscr{P}2$ is strictly convex with respect to $\tilde{\mathcal{A}}$ [64]. ■

Lemma 2. *The Problem $\mathscr{P}2$ is strictly convex with respect to frame resolution $\mathcal{S}$.*

Proof: For any feasible variable $s_i^2, s_j^2, \forall i, j \in \mathcal{K}$, we have

$$\frac{\partial^2 F}{\partial s_i^2 \partial s_j^2} = \begin{cases} \frac{\partial^2 \psi}{\partial s_i^2 \partial s_j^2} \sum\limits_{n \in \mathcal{N}} \frac{\tilde{a}_{i,n}}{f_n} \sum\limits_{m \in \mathcal{K}} \tilde{a}_{m,n} - \beta \frac{\partial^2 \xi}{\partial s_i^2 \partial s_j^2}, & i = j, \\ 0, & i \neq j, \end{cases} \tag{4.10}$$

Since $\psi(s_k^2)$ is convex function, $\frac{\partial^2 \psi}{\partial s_i^2 \partial s_j^2}$ is non-negative. Since $\xi(s_k^2)$ is concave function, $\frac{\partial^2 \xi}{\partial s_i^2 \partial s_j^2}$ is non-positive. Hence, the Hessian matrix $\mathbf{H} = \left(\frac{\partial^2 F}{\partial s_i^2 \partial s_j^2}\right)_{K \times K}$ is symmetric and positive definite. Constraints C_1 are convex, and Constraints C_2 and C_3 do not apply to $\mathcal{S}$. Therefore, Problem $\mathscr{P}2$ is strictly convex with respect to $\mathcal{S}$ [64]. ■

Lemmas 1 and 2 lay the foundation for solving Problem $\mathscr{P}2$ with the block coordinate descent method [71]. Based on this method, we develop a fast and accurate object analytics (FACT) algorithm which solves Problem $\mathscr{P}2$ by sequentially fixing one variable, i.e., $\tilde{\mathcal{A}}$ or $\mathcal{S}$, and updating the other one. The pseudo code of the FACT algorithm is presented in Algorithm 3. At the beginning of the algorithm, the weight β and video frame resolution $\mathcal{S}$ are initialized. With the fixed video frame resolution, we solve Problem $\mathscr{P}2$ to obtain the optimal server assignment $\tilde{\mathcal{A}}$. Then, based on the optimized $\tilde{\mathcal{A}}$, we optimize the video frame resolution. After each iteration, the value of the objective function F is calculated. The FACT algorithm iteratively optimizes $\tilde{\mathcal{A}}$ and $\mathcal{S}$ until the objective function F is converged. In the algorithm, we introduce

Algorithm 3: The FACT Algorithm

Input: The weight β, the convergence condition τ, the initial set of frame resolutions $\mathcal{S}_0$.

Output: The set of server assignments $\mathcal{A}$, the set of frame resolutions $\mathcal{S}$.

1 $\mathcal{S} \leftarrow \mathcal{S}_0, i \leftarrow 0$;
2 **while** *True* **do**
3 $\tilde{\mathcal{A}} \leftarrow$ solve Problem $\mathscr{P}2$ with fixed $\mathcal{S}$;
4 $\mathcal{S} \leftarrow$ solve Problem $\mathscr{P}2$ with fixed $\tilde{\mathcal{A}}$;
5 $F_i \leftarrow L - \beta A$;
6 **if** $|(F_i - F_{i-1})/F_i| \leq \tau$ **then**
7 **break**;
8 $i \leftarrow i + 1$;
9 **for** $k \in \mathcal{K}$ **do**
10 **for** $n \in \mathcal{N}$ **do**
11 **if** $n = \arg\max_{j \in \mathcal{N}} \tilde{a}_{k,j}$ **then**
12 $a_{k,n} \leftarrow 1$;
13 **else**
14 $a_{k,n} \leftarrow 0$;
15 $\mathcal{S} \leftarrow$ solve Problem $\mathscr{P}2$ with $\mathcal{A}$;
16 **return** $\mathcal{A}, \mathcal{S}$

an arbitrary small positive number τ to evaluate the convergence of the objective function as shown in line 6 of the pseudo code. After solving Problem $\mathscr{P}2$, $\tilde{a}_{k,n}$ are converted to $a_{k,n}$ according to

$$a_{k,n} = \begin{cases} 1, & n = \arg\max_{j \in \mathcal{N}} \tilde{a}_{k,j}, \\ 0, & \text{otherwise.} \end{cases} \tag{4.11}$$

Theorem 1. *The FACT algorithm converges to the optimal solution.*

Proof: The convergence of the FACT algorithm to optimal solution can be proved by showing that Problem $\mathscr{P}2$ is strictly convex with respect to each block of variables, e.g., $\tilde{\mathcal{A}}$ and $\mathcal{S}$ [71]. The convexity of Problem $\mathscr{P}2$ is proved by lemma 1 and 2. Hence, the optimality and convergence of the FACT algorithm is guaranteed. ■

Lemma 3. *The FACT algorithm has a sub-linear convergence rate.*

Proof: The block coordinate descent method ensures a sub-linear convergence rate [72]. The FACT algorithm is designed based on the block coordinate descent method. Thus, it has a sub-linear convergence rate. ■

4.3 Simulation Results

In this section, we evaluate the FACT algorithm through large-scale simulations. We simulate an edge network with 50 wireless access points and 20 servers. The distribution of the wireless access points and the user traffic are derived based on network traffic traces collected from an operating mobile network consisting of 10000 base stations and 50000 mobile users. The edge servers are randomly deployed in the network. The network latency between wireless access points and edge servers are generated according to the normal distribution with the mean value of 50ms. The network latency between the wireless access points and the cloud server are also generated based on the normal distribution but with a mean value of 150ms. The wireless data rates of users are uniformly distributed between 1 to 10 Mbps. Based on the measurements from our experiments, we adopt $c_k = \psi(s_k^2) = 7 \times 10^{-10} {s_k}^3 + 0.083$ TFLOPS as the computational complexity of analyzing a $s_k \times s_k$ video frame. In the simulation, the computing capacities of the cloud and edge servers are set to 10 and 2 TFLOPS, respectively. We model as $A_k = \xi(s_k^2) = 1 - 1.578e^{-6.5 \times 10^{-3} s_k}$ as the analytics accuracy function on both cloud and edge servers. The default value of β is 20, and the minimum video frame resolution is 40000 pixels (200×200).

In the simulation, we compare the FACT algorithm with the three categories of algorithms summarized in Table 4.1.

- **Baseline:** the baseline algorithm has a fixed video frame resolution and randomly assigns servers to MAR users.
- **Server assignment optimized:** this category of algorithms optimize the server assignments, but the video frame resolution is predefined. We implement maximum accuracy (maxA) and minimum latency (minL) algorithms which adopt

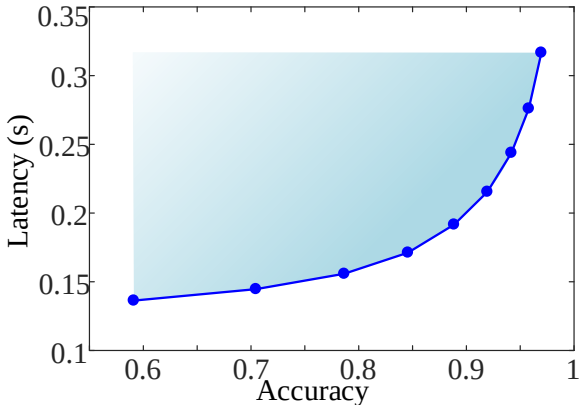

Figure 4.2: The Pareto boundary derived by the FACT algorithm.

Table 4.1: Algorithm Comparison

	frame resolution		server selection		
	fixed	optimized	random	greedy	optimized
FACT		x			x
Baseline	x		x		
MaxA	x				x
MinL	x				x
LoadS		x		x	
RandS		x	x		

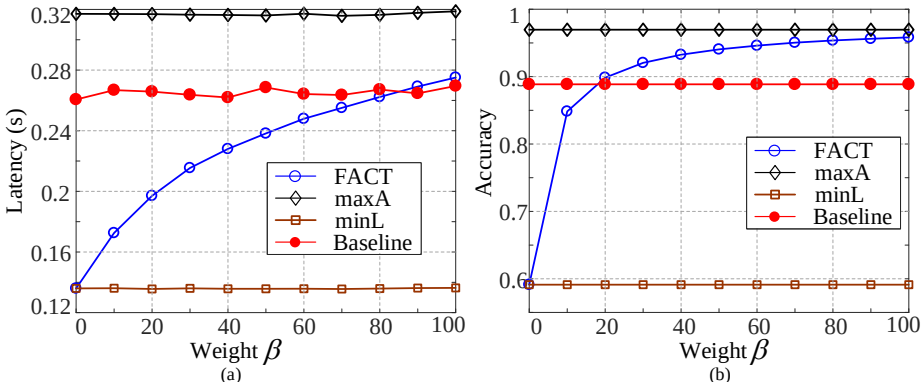

Figure 4.3: The system performance vs. β.

the largest and smallest frame resolutions, respectively.

- **Frame resolution optimized:** these algorithms optimize the frame resolution, but the server assignments are not optimized. We implement two methods: random server selection (RandS) and least workload server selection (LoadS).

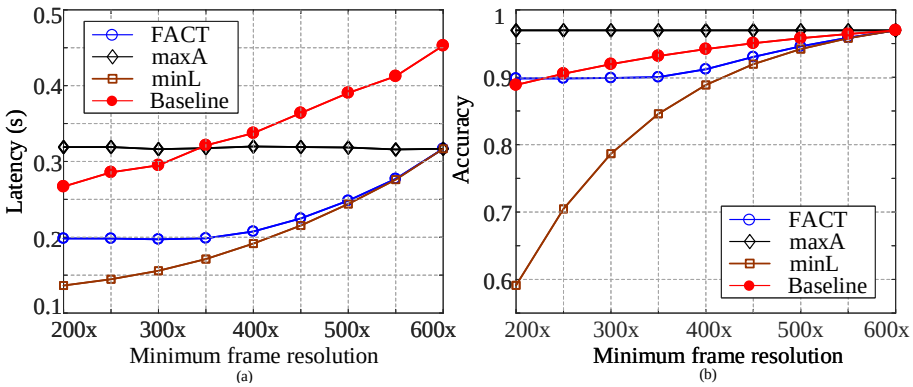

Figure 4.4: The system performance vs. frame resolution.

4.3.1 The impact of β

The value of β impacts the tradeoff between the service latency and analytics accuracy in the edge based MAR system. As shown in Fig. 4.2, by varying the value of β, we derive the Pareto boundary which characterizes the latency-accuracy tradeoff obtained by the FACT algorithm.

Fig. 4.3 shows the impact of β on the service latency and analytics accuracy of different algorithms. When β increases, the MAR system emphasizes the performance of the analytics accuracy. As a result, the FACT algorithm trades the service latency for the analytics accuracy. Since the maxA, minL, and baseline algorithms do not optimize the video frame resolution, the variation of β does not change the latency-accuracy tradeoff. Hence, the performances of these algorithms do not show many changes versus β. The latency fluctuation of the baseline algorithm is because of the random server assignments. This simulation result also shows that, as compared to the baseline algorithm, the FACT algorithm is able to reduce about 26% service latency while maintaining the similar analytics accuracy ($\beta = 20$), and enhance about 8% analytics accuracy when ensuring a similar service latency ($\beta = 90$).

4.3.2 The impact of AR video frame resolution

The AR video frame resolution impacts not only the transmission and computational latency but also the analytics accuracy. In order to evaluate such impacts, we

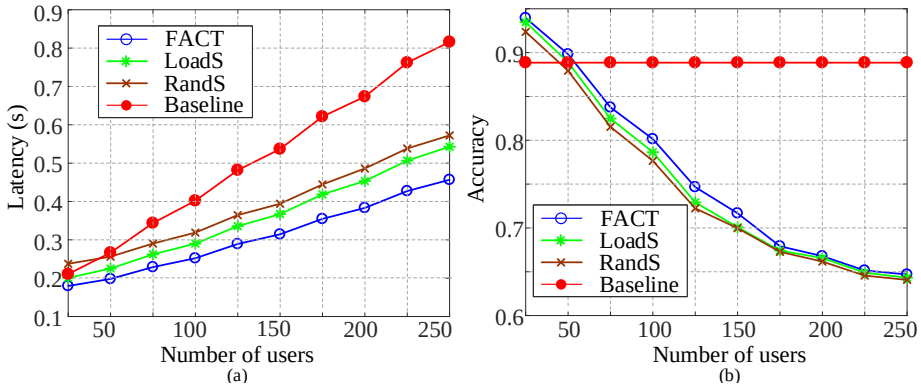

Figure 4.5: The system performance vs. number of users.

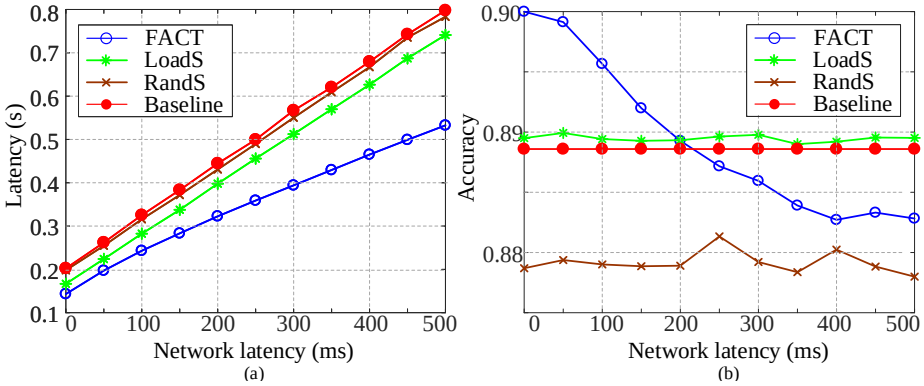

Figure 4.6: The system performance vs. core network latency.

vary the minimum video frame resolutions in the simulation. The maximum video frame resolution is fixed at 600×600 pixels. The optimized video frame resolution is between the minimum and maximum video frame resolutions. The video frame resolution of the baseline algorithm is defined as the mean value of the minimum and maximum video frame resolutions. As shown in Fig. 4.4 (a), the service latency of the minL and baseline algorithm increases versus the minimum video frame resolution. When the minimum video frame resolution is less than 350×350 pixels, the FACT algorithm maintains the system performance (both the service latency and analytics accuracy) because of the frame resolution optimization. When the minimum video frame resolution is larger than 500×500 pixels, the FACT algorithm has the same service latency as the minL algorithm because the optimal frame resolution equals the minimum frame resolution. Fig. 4.4 (b) shows that the FACT algorithm obtains a

lower latency at the cost of analytics accuracy. However, the latency-accuracy trade-off is mitigated. For example, as compared with the baseline algorithm, the FACT algorithm reduces about 40% latency at the cost of around 3% accuracy when the minimum video frame resolution is 400×400 pixels.

4.3.3 The impact of the number of users

Fig. 4.5 evaluates the impact of the number of users on the performance of the edge-based MAR system. When the number of users increases, the edge servers experience more workloads. Thus, the computational latency increases. As compared with the other algorithms, the FACT algorithm achieves the smallest latency. Since the FACT, LoadS and RandS algorithms optimize the video frame resolution, the performance differences between the FACT algorithm and the other two algorithms show the improvement gained from the optimal server assignment. Both RandS and baseline algorithms adopt the random server selection. Hence, the performance differences between these algorithms reflect the advantages of the frame resolution optimization.

As compared with the baseline algorithm, the FACT algorithm gains up to 38% service latency reduction with less than 10% loss of analytics accuracy when the number of users is 100. When the number of users increases, the servers are overloaded. In order to maintain a low service latency, the FACT algorithm aggressively reduces the video frame resolution, which is the reason for the 10% loss in the analytics accuracy. As compared with LoadS and RandS algorithms, the FACT algorithm improves both the latency and accuracy performance.

4.3.4 The impact of the average network latency

Fig. 4.6 shows the impact of the average network latency on the performance of the edge-based MAR system. Here, the average network latency reflects the average transmission delay between the wireless access points and servers. As shown in the

figure, the service latency of the MAR system increases versus the average network latency. However, the FACT algorithm is able to minimize the service latency at the cost of a slight decrease of the analytics accuracy. The latency gap between the FACT, LoadS and RandS algorithms reflects the gain obtained through optimizing the server assignment. The latency gap between the baseline and RandS algorithms is due to the video frame resolution optimization.

4.4 The System Implementation and Experiments

In this section, we implement the edge-based MAR system including the edge network orchestrator, MAR clients and MAR servers, and develop the corresponding MAR communication protocol. We conduct experiments based on the implementation to validate the performance of the edge network orchestrator and MAR communication protocol.

4.4.1 The edge-base MAR system implementation

Fig. 4.7 overviews the edge-based MAR system which consists of three major components: the edge network orchestrator, MAR client and MAR server.

The edge network orchestrator: the orchestrator is responsible for optimizing the server assignment and video frame resolution for MAR users. The core of the orchestrator is the FACT algorithm which performs the optimization. In order to realize the FACT algorithm, three auxiliary modules are implemented. The first one is the request handler which puts the incoming service requests into a FIFO queue and dispatches the server assignments and video frame resolutions derived from the FACT algorithm to the corresponding MAR users. The second module is the scheduler which manages the service queue and invokes the FACT algorithm based on the queue length and predefined optimization interval. In the implementation, the FACT algorithm is invoked if the queue length equals to 10 or a 100ms timer is expired after the last optimization. The optimization trigger, e.g., the queue length and timer, is

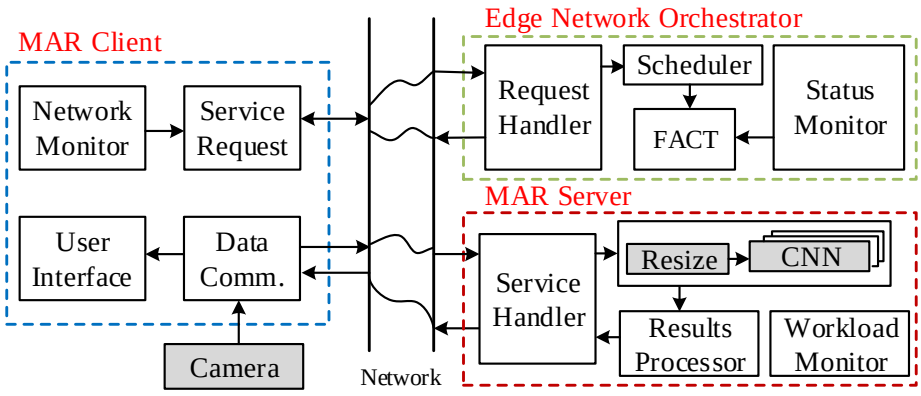

Figure 4.7: The overview of edge-based MAR system.

adjustable in the implementation. The third module is the system status monitor which monitors and collects the system information including users' wireless data rates, network latency between users and servers, and workloads on servers. This information is inputted to the FACT algorithm for optimizing the server assignment and video frame resolution. The orchestrator is deployed at the network edge and can be accessed by MAR users with very short latency. We also implement the baseline, maxA, minL, LoadS, and RandS algorithms in the network orchestrator for the performance comparison.

The MAR Client: The MAR client captures the real-world video, sends frames to a server and overlays the received information with its corresponding objects. There are four functional modules implemented in the MAR client. The first one is the network status monitoring module which measures wireless data rates and the network latency between the user and servers. This model also triggers a service request to the orchestrator for the performance optimization when the user's service quality is lower than a threshold. The second one is the user interface that captures the real-world environments and displays the analytics results received from the server. The third one is the service request module which sends the service request to the orchestrator. The service request message contains the information of the user's wireless data rate and network latency toward edge and cloud servers. It also parses

the control information about the frame resolution and server assignment received from the orchestrator. The fourth module is the data communication module which streams the AR video frames to an assigned server, and receives the analytics results from the server. This module is also responsible for resizing the video frames according to the frame resolution selected by the orchestrator. Since the current implementation of our MAR server only supports six video frame resolutions, we select one of the supported resolutions to approximate the video frame resolution determined by the orchestrator.

The MAR Server: The MAR server is developed to process the video frames and send the analytics results back to MAR users. The server is designed to serve multiple users simultaneously through multi-threading. There are four major modules implemented on the server. The first one is the service handler module which performs the authentication and establishes a socket connection with MAR users. This module is also responsible for dispatching the analytics results to corresponding MAR users.

The second one is the object analytics module, which decompresses the frames and performs the object analytics for MAR users. The object analytics module is designed based on the YOLO framework with the GPU acceleration [66]. In order to allow the object analytics module to analyze video frames with different resolutions, we loaded six models with different input resolutions. Therefore, the implemented object analytics module supports six video frame resolutions. If the resolution of incoming video frames does not match any of these video frame resolutions, the object analytics module resizes the input video frames into one of the supported resolutions and then analyzes the resized video frames.

The third one is the results processing module which prepares the information related to the recognized objects. The fourth module is the workload monitor which monitors workloads on the server and updates the workload information to the orchestrator periodically.

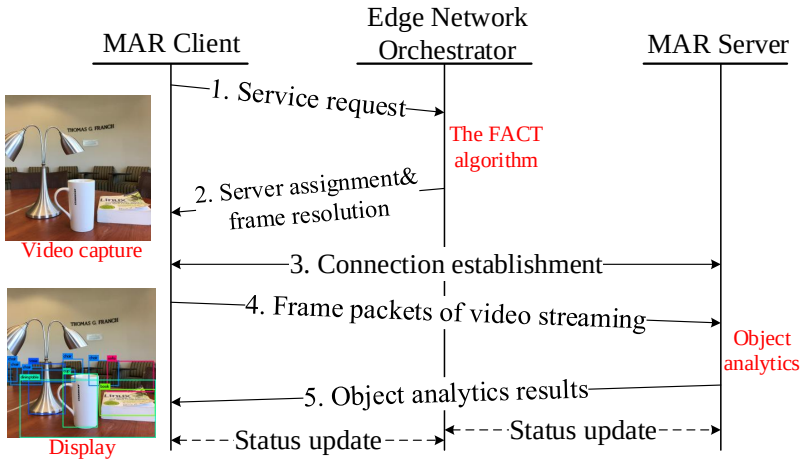

Figure 4.8: The MAR communication protocol.

4.4.2 The edge-based MAR communication protocol

As illustrated in Fig. 4.8, the proposed communication protocol enables the MAR service in five steps.

1. The MAR user sends a service request to the orchestrator. The service request message also includes the user's wireless data rates and network latency measurements. After the service is initialized, the service request message is used to periodically update the user's wireless data rates and network latency measurements to the orchestrator.
2. Upon receiving the service request, the orchestrator decides the server assignment and frame resolution, and sends them back to the user.
3. The user establishes a connection to the assigned server, and informs the server of its configuration information.
4. After the connection is established, the user sends its AR video frames to the server for the object analytics.
5. The MAR server detects and recognizes objects in the video frames, and sends the results back to the user.

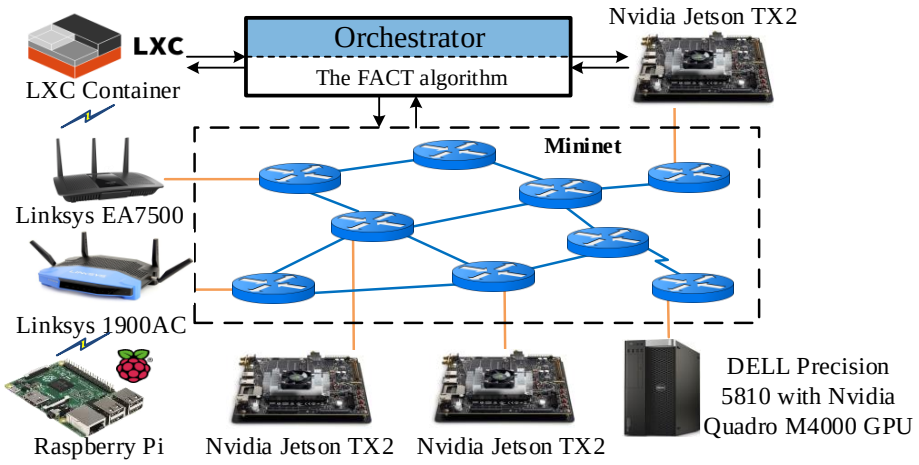

Figure 4.9: The edge-base MAR testbed.

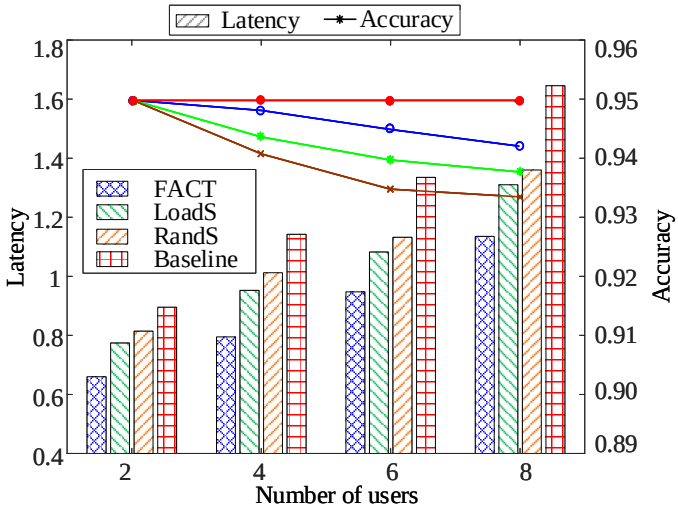

Figure 4.10: The system performance measurements with different number of users.

4.4.3 Performance evaluation

An edge-based MAR testbed as shown in Fig. 4.9 is developed to evaluate the performance of the proposed edge network orchestrator. The MAR clients are implemented in Raspberry Pis and LXC containers connected to the emulated network via wireless routers. The edge network is emulated by Mininet [73]. Using Mininet, we can configure the core network latency and evaluate its impact on the system performance. Three edge MAR servers are implemented using three NVIDIA Jetson TX development kits, and one cloud MAR server is implemented on a Dell workstation

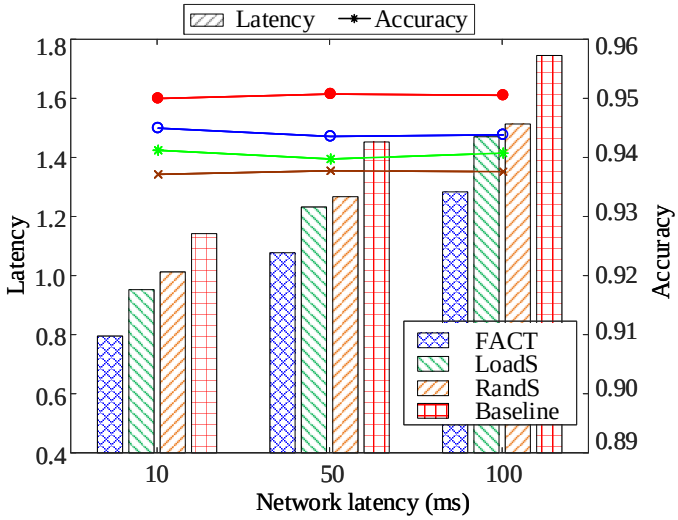

Figure 4.11: The system performance measurements with different network latency.

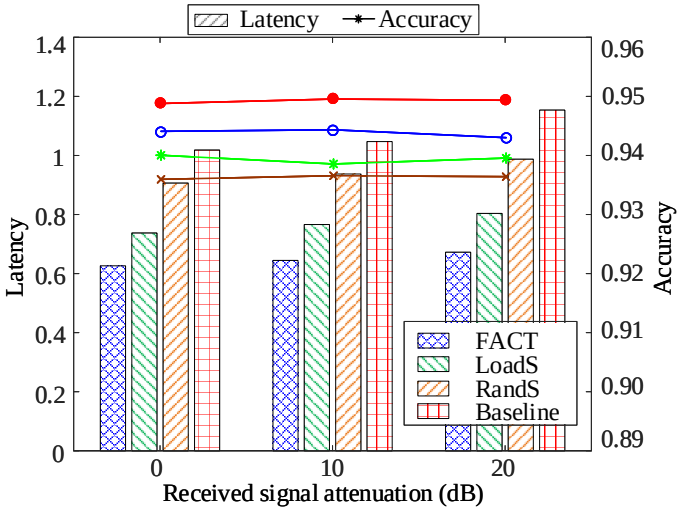

Figure 4.12: The system performance measurements with different RSSIs.

with Nvidia Quadro M4000 GPU.

Fig. 4.10 shows the latency and accuracy versus the number of users. Although the latency increase with the growth of the number of users, the FACT algorithm achieves significant latency reduction as compared to the other algorithms. For example, when the number of users is 8, the FACT algorithm reduces 27% latency with only 1% accuracy loss as compared to the baseline algorithm.

Fig. 4.11 shows the service latency and analytics accuracy versus the core network

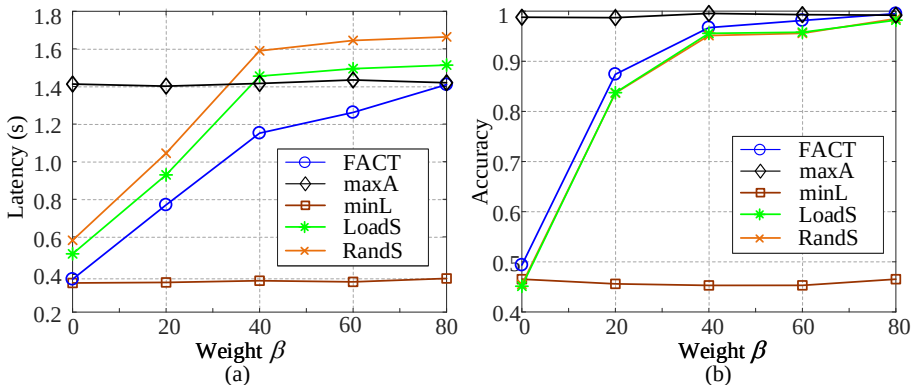

Figure 4.13: The performance measurements vs. β.

latency. The service latency increases with the growth of the core network latency. However, the FACT algorithm outperforms the other algorithms. For instance, it achieves about 25% latency reduction at the cost of less than 1% accuracy loss when the average core network latency is 100 ms.

Fig. 4.12 evaluates the impact of wireless channel conditions on the service latency and analytics accuracy. We use the Keysight J7211A attenuation control unit to vary the channel fading between the wireless access point and a MAR user. In this way, we evaluate the system performance under different wireless channel conditions. A larger channel fading increases the signal attenuation and leads to a lower wireless data rate. As a result, the service latency increases. Owing to the video frame resolution optimization, the FACT algorithm is able to achieve low service latency while maintaining a high analytics accuracy.

Fig 4.13 shows the tradeoff between the latency and accuracy with different β. A larger β indicates the system prefers a higher analytics accuracy. When β is small, e.g., $\beta = 20$, the FACT algorithm trades the analytics accuracy for the service latency. Therefore, it achieves more than 40% service latency reduction at the cost of about 10% analytics accuracy loss as compared with the maxA algorithm. When β is large, e.g., $\beta = 60$, the FACT algorithm prioritizes the analytics accuracy improvement. As a result, it achieves an almost perfect analytics accuracy. At the same time, the FACT

algorithm achieves lower latency than the maxA, LoadS, and RandS algorithms.

CHAPTER 5: VIRTUALEDGE: MULTI-DOMAIN RESOURCE ORCHESTRATION AND VIRTUALIZATION

In this chapter, we propose the *VirtualEdge* system to address the challenges of the multi-domain resource orchestration and virtualization. The *VirtualEdge* system dynamically creates a virtual node (vNode) on top of a physical cellular edge computing node to serve the traffic and workloads of a network slice. To mitigate the conflict between the isolation and radio resource efficiency, we introduce the concept of virtual resources and design a two-step multi-domain resource orchestration framework. To tackle the problems of multi-domain resource correlation and unknown utility functions of vNodes, we develop a novel learning-assisted multi-domain resource orchestration algorithm. The idea is to construct a probabilistic model as the black-box function that represents the relationship between the vNode performance and multi-domain resource allocations, improve the accuracy of the model with observed data, and estimate the gradient of the black-box function for optimizing the multi-domain resource orchestration.

5.1 *VirtualEdge* Overview

The objective of *VirtualEdge* is to efficiently manage multiple resources in the cellular edge computing node to maximize the performance of individual vNodes. Meanwhile, *VirtualEdge* provides functional and performance isolation among vNodes. Toward this end, *VirtualEdge* realizes dynamic multi-domain resource slicing in a cellular edge computing node in two steps. First, a multi-domain resource orchestration orchestrator is designed to allocate virtual resources to vNodes. Second, a multi-domain resource hypervisor is developed to map the virtual resources to phys-

ical resources.

Fig. 5.1 outlines the *VirtualEdge* system. To create network slices, service providers send slice requests describing their resource requirements to the mobile network operator who performs an admission control based on the resource availability. The *VirtualEdge* System creates vNodes in the physical cellular edge computing node for admitted slice requests. Given the resource requirements, the multi-domain resource orchestrator allocates virtual resources to individual vNodes to maximize their utilities. Since the relationship between the performance of a vNode and the resource allocation is unknown, the utility function of each vNode is a black-box function to the resource orchestrator. Hence, we design a novel learning-assisted multi-domain resource orchestration algorithm to solve the problem. In particular, we use a probabilistic model to represent the black-box function and exploit the model to learn the properties of the function [74]. Based on the learned properties of the utility functions, the resource orchestration problem is solved by using the proximal gradient method.

The service provider allocates virtual resources to its users based on a customized resource management algorithm that maximizes the utility of the vNode. After this step, in each vNode, the users' traffic and workloads are mapped to virtual resources. The multi-domain resource hypervisor takes these mappings as inputs to map virtual resources to physical resources. In the virtual-to-physical resource mapping, since the user information, e.g., wireless channel conditions, is available, the multi-domain resource hypervisor leverages this information to exploit the multi-user diversity and thus improve the efficiency of slicing radio resources. After the virtual-to-physical resource mapping, vNodes update and feed their utilities back to the resource orchestrator for the next round of the virtual resource allocation. The multi-domain resource orchestration algorithm converges after a few iterations, and then the performance of vNodes will be stable. We detail the design of the multi-domain resource

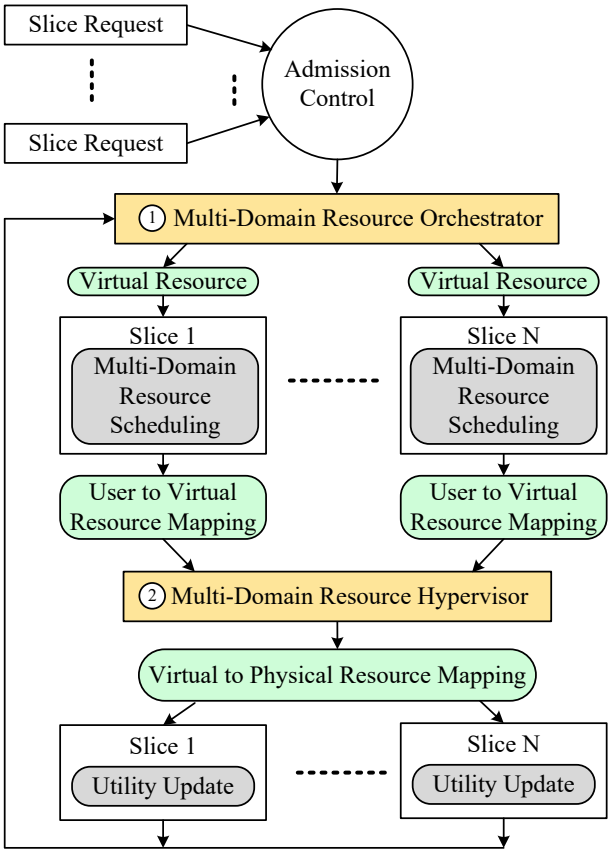

Figure 5.1: The overview of *VirtualEdge*.

orchestrator and hypervisor in Sections 5.2 and 5.3, respectively.

5.2 Multi-Domain Resource Orchestration

In this section, we present the multi-domain resource orchestration algorithm that dynamically allocates virtual resources to vNodes

5.2.1 Resource Orchestration Problem Formulation

We consider a cellular edge computing node which is composed of a base station and an edge server. Multiple vNodes are created in the cellular edge computing node to serve users' traffic and workloads of corresponding network slices. To provide service to their users, vNodes need resources from multiple technical domains. Here, we consider three resources from two technical domains. The resources are the uplink radio resource (UR), downlink radio resource (DR), and computing resource (CR).

Denote m_i, n_i, and c_i as UR, DR, and CR allocated to the ith vNode, respectively. The utility function of the ith vNode can be expressed as $f_i(m_i, n_i, c_i)$. Let $\mathcal{I}$ be the set of vNodes, and denote M^{tot}, N^{tot}, and C^{tot} as the total uplink, downlink, and computing resources, respectively. The multi-domain resource orchestration problem can be formulated as

$$\begin{aligned} &\max_{\{m_i,n_i,c_i,\forall i\in\mathcal{I}\}} \quad \sum_{i\in\mathcal{I}} \alpha_i f_i(m_i, n_i, c_i) \\ &s.t. \\ &C_1: \qquad \sum_{i\in\mathcal{I}} m_i \leq M^{tot}, \\ &C_2: \qquad \sum_{i\in\mathcal{I}} n_i \leq N^{tot}, \\ &C_3: \qquad \sum_{i\in\mathcal{I}} c_i \leq C^{tot}, \end{aligned} \tag{5.1}$$

where constraints C_1, C_2, and C_3 restrict that the resources allocated to vNodes should not exceed the total resources. $\alpha_i \geq 0$ is a weight for prioritizing the ith vNode in the resource allocation.

5.2.2 The Multi-Domain Resource Orchestration Algorithm

It is difficult to solve the above optimization problem because the utility function of each network slice is unknown. That is, the objective function of the problem is a black-box function which is intractable when the orchestrator allocates the resource to vNodes. However, we observe that it is not necessary to obtain the precise expression of utility function for the resource allocation. The gradients of utility functions are sufficient for updating the resource allocation iteratively with gradient-based methods. Toward this end, we construct a probabilistic model with Gaussian Process to characterize the utility functions and learn its properties with observed data. The accuracy of the model can be improved gradually with the increasing amount of observed data. Based on the model, we define the predictive gradients of utility function which are calculated based on the prediction of utilities. Then, with the predictive gradients, we design the multi-domain resource orchestration algorithm based on the

proximal gradient descent method that maximizes the utility of vNodes.

Denote $\mathbf{x}_i = [m_i, n_i, c_i]^T$ and $f_i(\mathbf{x}_i)$ as the allocated resources and utility of the ith vNode. With the *VirtualEdge* system (Fig. 5.1), given $\mathbf{x}_i$, the resource orchestrator can observe $y_i = f_i(\mathbf{x}_i) + \epsilon_i$ that is corrupted with Gaussian noise $\epsilon \sim \mathcal{N}(0, \delta^2_{noise})$ [75]. Denote $\mathbf{x}_i^{1:t}$ and $y_i^{1:t}$ as the set of resource allocations and corresponding observations in the t iterations, respectively. As the observations $\mathcal{D}_i^{1:t} = \{\mathbf{x}_i^{1:t}, y_i^{1:t}\}$ accumulate, the posterior distribution of the black-box function can be obtained by combining the prior distribution $P(f_i(\mathbf{x}_i))$ and the likelihood function $P(\mathcal{D}_i^{1:t}|f_i(\mathbf{x}_i))$:

$$P(f_i(\mathbf{x}_i)|\mathcal{D}_i^{1:t}) \propto P(\mathcal{D}_i^{1:t}|f_i(\mathbf{x}_i))P(f_i(\mathbf{x}_i)). \tag{5.2}$$

The posterior updates the beliefs about the black-box function.

A Gaussian process [75] is completely specified by its mean function μ and covariance function k,

$$f_i(\mathbf{x}_i) \sim \mathcal{GP}(\mu(\mathbf{x}_i), k(\mathbf{x}_i, \mathbf{x}_i')) \tag{5.3}$$

where

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{1}{2}||\mathbf{x}_i - \mathbf{x}_j||^2). \tag{5.4}$$

Given a potential resource allocation $\mathbf{x}_i^*$, we can derive the predictive posterior distribution

$$P(f_i(\mathbf{x}_i^*)|\mathcal{D}_i^{1:t}, \mathbf{x}_i^*) \sim \mathcal{N}(\mu^t(\mathbf{x}_i^*), \sigma_i^2(\mathbf{x}_i^*)) \tag{5.5}$$

where

$$\mu^t(\mathbf{x}_i^*) = \mathbf{k}^T[\mathbf{K} + \delta^2_{noise}I]^{-1}y_i^{1:t} \tag{5.6}$$

and

$$\sigma_i^2(\mathbf{x}_i^*) = k(\mathbf{x}_i^*, \mathbf{x}_i^*) - \mathbf{k}^T[\mathbf{K} + \delta^2_{noise}I]^{-1}\mathbf{k} \tag{5.7}$$

are the sufficient statistics of predictive posterior distribution $P(f_i(\mathbf{x}_i^*)|\mathcal{D}_i^{1:t}, \mathbf{x}_i^*)$. Here,

$$\mathbf{k} = [k(\mathbf{x}_i^*, \mathbf{x}_i^1), k(\mathbf{x}_i^*, \mathbf{x}_i^2), \cdots, k(\mathbf{x}_i^*, \mathbf{x}_i^t)], \tag{5.8}$$

and

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_i^1, \mathbf{x}_i^1) & \cdots & k(\mathbf{x}_i^1, \mathbf{x}_i^t) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_i^t, \mathbf{x}_i^1) & \cdots & k(\mathbf{x}_i^t, \mathbf{x}_i^t) \end{bmatrix}. \tag{5.9}$$

Based on the predictive posterior distribution, we define the predictive gradient of the black-box function at $\mathbf{x}_i^t$ as

$$\begin{aligned} \nabla f_i(\mathbf{x}_i^t) &\coloneqq [\nabla f_i(m_i^t), \nabla f_i(n_i^t), \nabla f_i(c_i^t)]^T \\ &\coloneqq \begin{bmatrix} \frac{1}{\Delta m}(\mu^t([m_i^t + \Delta m, n_i^t, c_i^t]) - y_i^t) \\ \frac{1}{\Delta n}(\mu^t([m_i^t, n_i^t + \Delta n, c_i^t]) - y_i^t) \\ \frac{1}{\Delta c}(\mu^t([m_i^t, n_i^t, c_i^t + \Delta c]) - y_i^t) \end{bmatrix} \end{aligned} \tag{5.10}$$

where Δm, Δn, and Δc are small positive constants.

Since a vNode usually has better service performance when allocated more resources, it is reasonable to assume that the utility functions of vNodes are non-decreasing versus the allocated resources. Therefore, the predictive gradients are non-negative and provide guidance on the resource allocation. For example, when $\nabla f_i(m_i^t)$ is larger than $\nabla f_i(n_i^t)$ and $\nabla f_i(c_i^t)$, it indicates that uplink radio resources is the performance bottleneck in the ith vNode. Meanwhile, if $\nabla f_i(m_i^t)$ is larger than $\nabla f_i(m_j^t), \forall j \in \mathcal{I}$, it means that allocating more uplink radio resource to the ith vNode can improve the overall utility of the vNodes created in the cellular edge computing node.

With the predictive gradients, we solve the multi-domain resource orchestration problem using the proximal gradient descent method [64]. At the beginning, resources

are evenly allocated to vNodes. Then, the resource orchestrator observes the utilities of vNodes. Based on the observed utilities of the ith vNode for t iterations, the resource orchestrator updates the resource allocations to the ith vNode as follows:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha_i \lambda \cdot \nabla f_i(\mathbf{x}_i^t) \tag{5.11}$$

where $\lambda = [\lambda_m, \lambda_n, \lambda_c]^T$ is the vector of positive step sizes for searching the optimal solution, and $\cdot$ is the dot product operator.

Since the total radio and computing resources are constrained, we have to ensure that the allocated resources do not exceed the total resources in the cellular computing node. Therefore, we project the resource allocations into a bounded domain that satisfies the total resource constraints in the physical cellular edge computing node. Define $P_\Omega(x) = \arg\min_{y \in \Omega} \|x - y\|^2$ as the Euclidean projection of x on Ω. Denote $\mathbf{m}^{t+1} = \{m_i^{t+1} | \forall i \in \mathcal{I}\}$, $\mathbf{n}^{t+1} = \{n_i^{t+1} | \forall i \in \mathcal{I}\}$ and $\mathbf{c}^{t+1} = \{c_i^{t+1} | \forall i \in \mathcal{I}\}$. Then, the resource allocations are projected to corresponding domains as $\mathbf{m}^{t+1} = P_{\Omega_m}(\mathbf{m}^{t+1})$, $\mathbf{n}^{t+1} = P_{\Omega_n}(\mathbf{n}^{t+1})$, and $\mathbf{c}^{t+1} = P_{\Omega_c}(\mathbf{c}^{t+1})$, respectively. Here, Ω_m, Ω_n and Ω_c are the bounded domains of uplink, downlink and computing resources, respectively. The bounded domains are defined to ensure that the resource allocations satisfy the constraints C_1, C_2, and C_3 in the multi-domain resource orchestration problem.

The pseudo-code of the multi-domain resource orchestration algorithm is summarized in Algorithm 4. When the amount of observed data is insufficient, the prediction about the posterior distribution is not accurate, and the predictive gradients are close to zero. To solve this problem, we use random gradients for calculating future resource allocations at the beginning of the algorithm until sufficient data are observed. The algorithm stops when the resource allocations converge.

Proposition 1. *$\nabla \bar{f}_i(\boldsymbol{x}_i), \forall i \in \mathcal{I}$ are the controllably accurate gradient approximations of $\nabla f_i(\boldsymbol{x}_i), \forall i \in \mathcal{I}$ if $f_i(\boldsymbol{x}_i), \forall i \in \mathcal{I}$ are Lipschitz continuous.*

Algorithm 4: Multi-Domain Resource Orchestration

Input: M^{tot}, N^{tot}, C^{tot}, and η.
Output: m_i, n_i, c_i, $\forall i \in \mathcal{I}$.

Initialize m_i, n_i, c_i, $\forall i \in \mathcal{I}$, and set $t \leftarrow 1$;
while *True* **do**
  $/ * *query\ the\ utility\ of\ vNodes * */$;
  $f_i\left(m_i, n_i, c_i\right), \forall i \in \mathcal{I}$;
  $/ * *update\ posterior\ distribution\ for\ vNodes * */$;
  $\mu^t(\mathbf{x}_i^t) \leftarrow Eq.\ 5.6, \sigma_i^2(\mathbf{x}_i^t) \leftarrow Eq.\ 5.7, \forall i \in \mathcal{I}$;
  $/ * *compute\ predictive\ gradients * */$;
  $\nabla f_i(\mathbf{x}_i^t) \leftarrow Eq.\ 5.10, \forall i \in \mathcal{I}$;
  $/ * *update\ allocation\ based\ on\ gradients * */$;
  $\mathbf{x}_i^{t+1} \leftarrow Eq.\ 5.11$;
  $/ * *project\ resource\ allocation * */$;
  $\mathbf{m}^{t+1} \leftarrow P_{\Omega_m}(\mathbf{m}^{t+1})$, $\mathbf{n}^{t+1} \leftarrow P_{\Omega_n}(\mathbf{n}^{t+1})$, $\mathbf{c}^{t+1} \leftarrow P_{\Omega_c}(\mathbf{c}^{t+1})$;
  $/ * *determine\ algorithm\ convergence * */$;
  if $||\mathbf{x}_i^{t+1} - \mathbf{x}_i^t|| \leq \eta, \forall i \in \mathcal{I}$ **then**
    break;
  $t \leftarrow t + 1$;
return m_i, n_i, c_i, $\forall i \in \mathcal{I}$

Proof: Denote $f_i(\mathbf{x}_i)$ and $\bar{f}_i(\mathbf{x}_i)$ as the utility and predicted utility of the ith slice at $\mathbf{x}_i$, respectively. The predictive gradients are defined as $\nabla \bar{f}_i(\mathbf{x}_i) = (\bar{f}_i(\mathbf{x}_i + \tau) - f_i(\mathbf{x}_i))/\tau = (\mu(\mathbf{x}_i + \tau) - f_i(\mathbf{x}_i))/\tau, \forall i \in \mathcal{I}$ (Eq. 5.10). Then,

$$|\nabla \bar{f}_i(\mathbf{x}_i) - \nabla f_i(\mathbf{x}_i)| = \frac{1}{\tau}|(\mu(\mathbf{x}_i + \tau) - f_i(\mathbf{x}_i + \tau))| \tag{5.12}$$
$$= \frac{1}{\tau}(|\mathbf{k}^T[\mathbf{K} + \delta_{noise}^2 I]^{-1} y_i^{1:t} - f_i(\mathbf{x}_i + \tau)|),$$

where $y_i = f_i(\mathbf{x}_i) + \epsilon$. Since the utility functions $f_i(\mathbf{x}_i), \forall i \in \mathcal{I}$ are Lipschitz continuous [64], there exists a positive real constant κ such that, for all real $\mathbf{x}_i^1$ and $\mathbf{x}_i^2$,

$$|f_i(\mathbf{x}_i^1) - f_i(\mathbf{x}_i^2)| \leq \kappa |\mathbf{x}_i^1 - \mathbf{x}_i^2|, \forall i \in \mathcal{I}. \tag{5.13}$$

Hence, we obtain

$$|\nabla \bar{f}_i(\mathbf{x}_i) - \nabla f_i(\mathbf{x}_i)| \leq \Delta, \tag{5.14}$$

where Δ is a constant. Accoridng to Definition 10.1 in [76], the $\nabla \bar{f}_i(\mathbf{x}_i), \forall i \in \mathcal{I}$ are the controllably accurate gradient approximations of $\nabla f_i(\mathbf{x}_i), \forall i \in \mathcal{I}$. ■

Theorem 2. *The multi-domain resource orchestration algorithm converges if* $f_i(\boldsymbol{x}_i), \forall i \in \mathcal{I}$ *are non-decreasing and Lipschitz continuous.*

Proof: The resource orchestration algorithm is designed based on proximal gradient method. According to Theorem 10.6 in [76], the convergence of the algorithm can be proved by showing 1) the predictive gradients $\nabla \bar{f}_i(\mathbf{x}_i), \forall i \in \mathcal{I}$ are the controllably accurate gradient approximations of $\nabla f_i(\mathbf{x}_i), \forall i \in \mathcal{I}$; and 2) the step size is positive in the gradient-based descent method. The first condition has been proved in *Proposition* 1. Since the step size in the algorithm is positive, the algorithm satisfies both conditions required for the convergence. Therefore, the convergence of Alg. 4 is proved. ■

5.3 Multi-Domain Resource Hypervisor

The multi-domain resource hypervisor is developed to provide a unified multi-domain resource abstraction view so that the resource orchestrator and vNodes can efficiently manage resource allocations for the vNodes and users, respectively. The hypervisor performs multi-domain resources, i.e., radio and computing, virtualization which maps virtual resources in multiple domains to the underlying physical resources. In resource virtualization, the design objective is to ensure the isolation among vNodes while maximizing utilization of physical resources, e.g., exploiting diversity gains in wireless communications.

5.3.1 Radio Resource Virtualization

In the system, we let vNodes share the same control plane and focus on virtualizing resources in the user plane, i.e., PUSCH/PDSCH in LTE. For the control plane, we add a function in the eNodeB, i.e., radio base station, to associate users to corresponding vNodes; other than that, the control plane operations follow the standard

LTE protocol implemented in OAI LTE platform [77]. For the user plane, we realize the virtual-to-physical resource mapping by managing the uplink/downlink resources (URs/DRs), i.e., physical resource blocks (PRBs), in the MAC layer as shown in Fig. 5.2. Here, the URs and DRs are referred to as the uplink and downlink bandwidth, respectively. During the virtual-to-physical resource mapping, the URs and DRs are converted to the corresponding number of PRBs.

Each vNode performs its resource scheduling and allocates URs and DRs to its users. Then, we calculate the number of PRBs, N_k, required by the kth user. Here, we use RR (radio resource) to represent either URs or DRs. Since the bandwidth of a PRB in LTE is 180kHz, $N_k = \lfloor RR/180kHz \rfloor$ where $\lfloor x \rfloor$ returns the greatest integer less than or equal to x. After the RRs are allocated to users, the virtual resources allocated to users of all vNodes are known. Hence, instead of allocating physical resources to each vNode statically, we are able to map the virtual resources of users to physical resources dynamically, which allows the exploitation of the diversity gain in wireless communications.

Due to the dynamics of the wireless channel, mobile users have varying signal-to-noise-plus-interference ratios (SINRs) on different PRBs. Based on the SINR, the number of bits carried by the jth PRB for the kth user $r_{j,k}$ can be derived using the modulation and coding scheme (MCS) in LTE. Denote the set of scheduled users and PRBs as $\mathcal{K}$ and $\mathcal{J}$, respectively. The virtual-to-physical radio resource mapping problem can be expressed as

$$\begin{aligned} &\max_{\{x_{k,j}, k\in\mathcal{K}, j\in\mathcal{J}\}} \quad \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{J}} x_{k,j} r_{k,j} \\ &s.t. \\ &C_1: \qquad \textstyle\sum_{k\in\mathcal{K}} x_{k,j} = 1, \forall j \in \mathcal{J}, \\ &C_2: \qquad \textstyle\sum_{j\in\mathcal{J}} x_{k,j} \geq N_k, \forall k \in \mathcal{K}, \\ &C_3: \qquad x_{k,j} \in \{0,1\}, \forall k \in \mathcal{K}, j \in \mathcal{J}, \end{aligned} \tag{5.15}$$

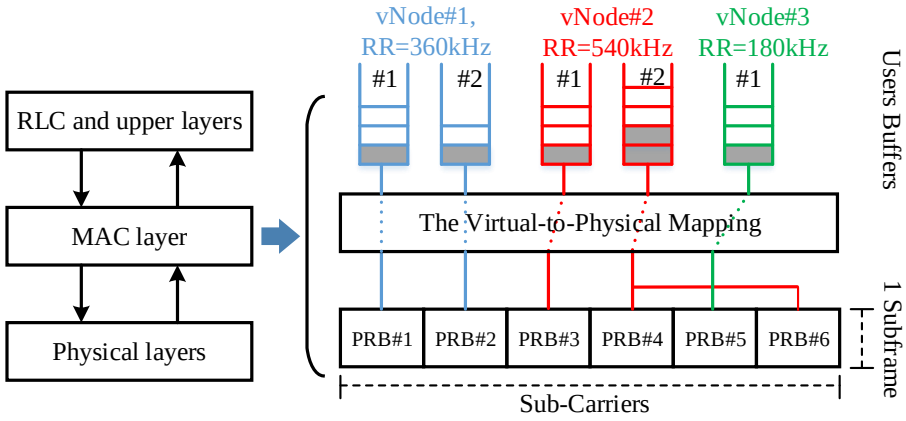

Figure 5.2: An illustration of radio resource virtualization.

where the integer variable $x_{k,j} = 1$ indicates that the jth PRB is allocated to the kth user. The above problem is an integer programming and difficult to solve [64].

We propose a heuristic algorithm to solve the above problem for uplink and downlink virtual-to-physical resource mapping, respectively. The basic idea is to select the user with the best channel condition for each PRB. In the case of downlink, for the jth PRB, we assign it to the user with maximum $r_{k,j}$ from the set of users whose DRs are not fully mapped to PRBs. If there are surplus PRBs, we assign them to the user who has the best channel condition on these PRBs. In the uplink, since the SC-FDMA (single carrier FDMA) is adopted in LTE, the PRBs allocated to a user must be contiguous in frequency domain [78]. For the jth PRB, we select the user with the best channel condition from the set of users whose URs are not fully mapped to PRBs. If the user requires N PRBs, the jth to the $(j + N - 1)$th PRBs are allocated to the user to ensure that the frequencies are allocated contiguously. If there are surplus PRBs, they are allocated to users whose current PRBs and the surplus PRBs are contiguous.

5.3.2 Computing Resource Virtualization

Since most of the compute-intensive applications rely on GPUs to accelerating the computation, we consider GPUs as the edge computing processors and develop com-

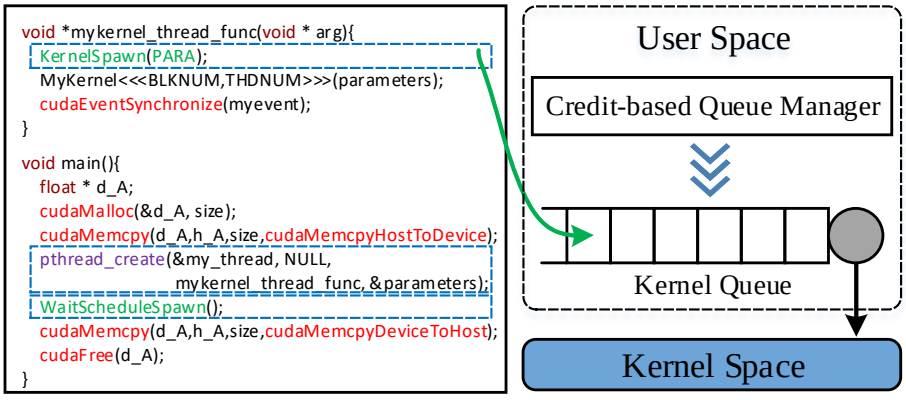

Figure 5.3: An illustration of GPU Virtualization.

puting resource virtualization on CUDA GPU computing platform. In the CUDA programming model, an application can launch multiple *kernels* that can be concurrently executed by CUDA *threads* [79]. Multiple *threads* are grouped into a *warp* which is the smallest scheduling granularity on CUDA-enable GPU. Since the technical details of the warp scheduler are not publicly available, it is challenging to dynamically virtualize GPU resources. To realize the computing resource virtualization, we introduce an FIFO queue to buffer the *kernel* command in the user space and design a credit-based queue management scheme to manage the dispatch of these commands. In the user space, the *KernelSpawn* function is developed to push *kernel* commands into the FIFO queue. The *kernel* commands are popped out of the queue when the *KernelSpawn* function receives a permission from the queue manager. The *kernel* is executed after the corresponding *kernel* command is popped out of the queue. This process is illustrated in Fig. 5.3. Since we create a system thread with the non-blocking property for running the *KernelSpawn* function, the executions of *kernels* remain asynchronous.

In the credit-based queue manager, the credit of a user at time t is updated as

$$C_t = C_{t-1} + S_t - \sum_j T_j. \tag{5.16}$$

where S_t is the credits generated every millisecond and T_j is the number of requested parallel *threads* of the jth running *kernel*. Here, S_t is determined by the virtual computing resource (CRs) allocated to the user. When the user has sufficient credits, the queue manager sends a permission message to let the *KernelSpawn* function pop a *kernel* command out of the FIFO queue. Since the credit update needs the execution status of *kernels* which are not available in CUDA, we create *cudaEvents* and use the *cudaEventSynchronize()* function to record and track the execution status of *kernels*, respectively.

In the CUDA programming model, *cudaDeviceSynchronize()* builds an explicit barrier to wait the completion of all dispatched *kernels*. With the credit-based queue management, the barrier can be falsely cleared by *cudaDeviceSynchronize()* when the credit generation rate is slow. For example, assume that a user has enqueued 100 *kernel* commands. 80 of them are dispatched to the kernel space, and the remaining 20 are queued waiting for credits. If the dispatched *kernels* are completed before any new *kernel* commands are dispatched, *cudaDeviceSynchronize()* will clear the barrier. We introduce a new function named *WaitScheduleSpawn* to address this problem. The *WaitScheduleSpawn* function ensures all previous *kernels* being dispatched and then calls the *cudaDeviceSynchronize()* to wait the completion of *kernels*. On designing the credit-based queue manager, we do not modify the source codes of *kernels* and the CUDA drivers. Therefore, the proposed computing resource virtualization solution can be utilized to virtualize GPU resources for any CUDA-based programs.

5.4 Performance Evaluation

We implement the *VirtualEdge* system and evaluate its performance and various properties through both experiments and network simulations.

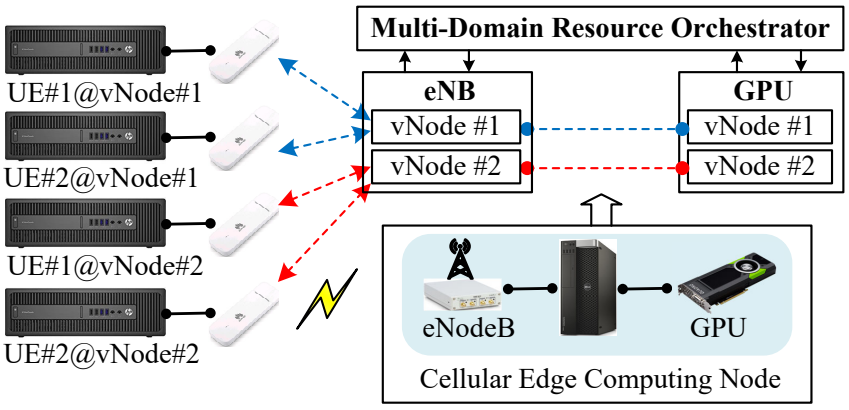

Figure 5.4: The overview of the system prototype.

5.4.1 System Implementation

The orchestrator maintains a context information table that consists of the vNode identifier, user identifier, and resource allocations. The vNode identifier indicates which vNode a user belongs to. We use the international mobile subscriber identity (IMSI) and the radio network temporary identifier (RNTI) to identify a user in the system. In order to build the context information table, we extract a user's IMSI from the user's message sent from eNodeB to mobile management entity (MME) via S1 Application Protocol (S1AP). We obtain the user's RNTI in the MAC layer once it is available and match the RNTI with the IMSI. The RNTI-to-IMSI matching is necessary because our radio resource virtualization is implemented on the MAC layer where users can only be identified by their RNTIs. To identify a user in computing resource virtualization, the mobile application is responsible for sending the user's IMSI to the edge computing node. In order to determine the virtual-to-physical resource mapping in the hypervisor, the resources allocated to users are recorded in the context information table. If the context information of the vNode changes, e.g., new vNode creation, the updated context information is pushed to the hypervisor through a TCP socket.

5.4.2 System Prototype

We develop a small-scale prototype as shown in Fig. 5.4 to evaluate the performance of the *VirtualEdge* system. The system prototype consists of a cellular edge computing node composed of an eNodeB and a GPU server and four user equipment (UEs). On the cellular edge computing node, two vNodes are created, and each vNode has two UEs. The eNodeB is implemented using Ettus USRP B210 [80] based on the OpenAirInterface (OAI) LTE platform [77], and the core network is built with OpenAir-CN that is an open-source implementation of the LTE evolved packet core (EPC) [81]. The UEs are emulated using LTE dongles [82]. The carrier frequencies of the LTE system are 2.655GHz for downlink and 2.535GHz for uplink. The total uplink and downlink bandwidth are 5MHz (25 PRBs), respectively. The computing resource virtualization is implemented using a NVIDIA GeForce GTX 1080 Ti with CUDA GPU computing platform [79, 83].

5.4.3 Compute-intensive Applications

To evaluate the *VirtualEdge* system, we develop two compute-intensive mobile applications based on the YOLO object detection algorithm [84]: mobile augmented reality (MAR) and video analytics and streaming (VAS). The first and second vNodes support MAR and VAS applications, respectively.

- **Mobile Augmented Reality (MAR):** A client continuously sends the camera-captured frames to the server and retrieves the object detection results. The server performs object detection using the YOLO algorithm, and sends detection results back to the client. The computation model is YOLO 288x288. Here, YOLO 288x288 means the YOLO object detection algorithm tuned at the image resolution of 288x288. MAR represents the applications that have heavy uplink traffic loads and moderate computing workloads.

- **Video Analytics and Streaming (VAS):** A client sends a streaming request to the server. The server retrieves the real-time camera frames, processes it with YOLO 544x544, and then sends the frames with detection results back to the client. The VAS represents applications that have heavy downlink traffic loads and intensive computing workloads. Here, YOLO 544x544 has higher computation complexity than YOLO 288x288.

5.4.4 Experimental Evaluation

In the experiments, the vNode latency (vNL) is defined as the sum latency of all users in a vNode; the instantaneous system latency (iSysL) is defined as the summation of the weighted vNL of all vNodes at each iteration; the average system latency (aSysL) is defined as the average iSysL over all iterations. We adopt the minus vNode latency as the utility of a vNode. Therefore, maximizing the vNode utility equals to minimizing the vNode latency. We compare the *VirtualEdge* system with the following systems:

- **Static allocation (*Static*):** *Static* evenly allocates the multi-domain resources to every vNode.
- **Traditional Bayesian optimization (TBO):** TBO treats the overall system performance as a black-box function of resource allocations in all vNodes, models the distribution of the black-box function using the Gaussian process, and uses the expected improvement (EI) as the acquisition function to maximize the probability of improving over the best current value [85]. The pseudo-code is summarized in Alg. 5. We implement the TBO and apply it to solve the multi-domain resource orchestration problem.

Convergence: Fig. 5.5 (a) shows the iSysL versus the number of iteration under three systems. Both *VirtualEdge* and TBO converge after about 20 iterations and achieve about 27% reduction of iSysL as compared to *Static*. *Static* maintains an almost constant iSysL performance since it evenly allocates resources to all vNodes.

Algorithm 5: Traditional Bayesian Optimization

for $t = 1, 2, ..., T_{max}$ **do**
  Select new point $\mathbf{x}^t$ by maximizing acquisition function under resource constraints C_1, C_2, C_3;
  Query the utility $y^t = \sum_{i \in \mathcal{I}} f_i(\mathbf{x}^t)$;
  Augment data set $\mathcal{D}^{1:t} = \{\mathcal{D}^{1:t-1}; (\mathbf{x}^t, y^t)\}$;
  Update posterior distribution with Eq. 5.6 and Eq. 5.7;
return x

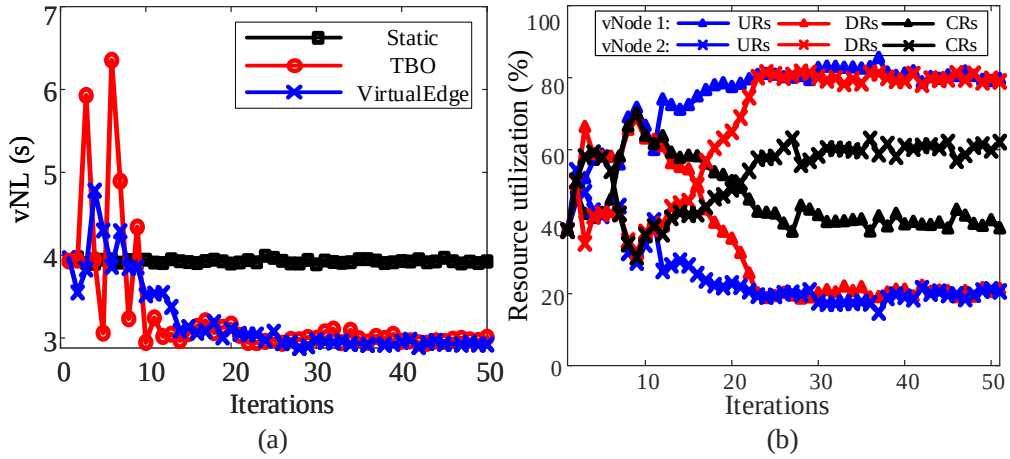

Figure 5.5: The convergence performance of the *VirtualEdge* system.

TBO has a larger jitter than the *VirtualEdge* system for two reasons. First, the black-box function of *TBO* has more variables, and thus it requires more data points to learn the properties of the black-box function. Second, the multi-domain resource orchestration algorithm in the *VirtualEdge* system provides a more accurate and stable estimation of the black-box functions.

Application awareness: Fig. 5.5 (b) illustrates the resource utilization of vNodes versus the number of iterations in the *VirtualEdge* system. It shows that the first vNode utilizes more URs but less DRs than the second vNode. This is because the MAR application in the first vNode has heavier traffic loads in the uplink while the VAS application in the second vNode has heavy traffic loads in the downlink. Since the computing demand of the VAS application is higher than that of the MAR application, more computing resources are allocated to the second vNode to improve

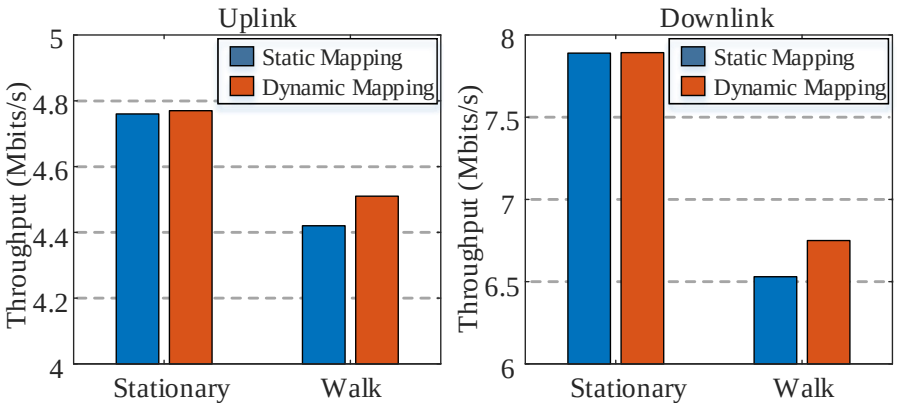

Figure 5.6: The comparison between static and dynamic mapping.

the utility. This result implies that the orchestrator is able to allocate multi-domain resources according to the characteristics of applications in vNodes.

Radio resource virtualization efficiency: To evaluate the efficiency of virtualizing radio resource, we compare the uplink and downlink throughput with static and dynamic mapping of virtual-to-physical radio resources under stationary and random walk scenarios. Here, the larger throughput reflects the higher radio resource efficiency. The VirtualEdge system develops the dynamic mapping of radio resources for vNodes. Other existing radio access network virtualization system such as Orion [29] and NVS [48] uses the static mapping between the virtual and physical radio resources.

As shown in Fig. 5.6, when the users are stationary, the wireless channels are relatively stable, the frequency diversity that can be exploited is small. As a result, the throughput with static and dynamic mapping is almost the same. However, in the random walk scenario, the dynamic mapping, i.e., VirtualEdge, outperforms the static mapping, e.g., Orion and NVS, in terms of throughput. This is because, with the dynamic mapping, the Virtual Edge system can exploit the frequency diversity among users to maximize the throughput. Note that when there are a large number of users, the gain of dynamic mapping will be more significant.

Functional isolation: The functional isolation ensures that each vNode can customize its resource management strategy. Figs. 5.7 (a) and (b) show that $VirtualEdge$

provides the functional isolation in the radio and computing resource virtualization, respectively. Given the total amount of resources including both radio and computing resources, a vNode can freely allocate these resources to its users. Here, 50%50% means that the resources are evenly allocated to both users, while 66%33% means that 66% and 33% of the resources are allocated to the first and second users, respectively. The performance of the users varies according to resource allocations.

Performance isolation: The performance isolation mainly ensures that the performance of different vNodes is independent. Figs. 5.7 (c) and (d) show the performance isolation of $VirtualEdge$. The performance of vNodes is regulated by the orchestrator through resource allocations. Under the scenario of 66%33%, the first vNode has much better performance than the second vNode in terms of both the network throughput and computing latency. Although the vNodes share the same physical cellular edge computing node, the second vNode, which has a worse performance, does not grab resources from the first vNode.

Prioritization: With the *VirtualEdge* system, we can change the priority of the vNodes in the resource orchestration by modifying the weights of the vNode' utility. Fig. 5.8 (a) shows that modifying the weight can effectively change the vNL.

Utility observations: In each iteration of the resource orchestration, both TBO and $VirtualEdge$ have to observe the utility of vNodes by querying the value of the corresponding black-box functions with current resource allocation. Since the network system is dynamic, the utility observations may contain some noises. These noises can be reduced by obtaining the average of multiple observations with the same resource allocation. Fig. 5.8 (b) shows the impact of the number of observations on the system performance. We can see that more observations per iteration lead to a lower aSysL because the observed utility is more reliable after the noise is reduced. The disadvantage of having multiple observations per iteration is the additional time spent on querying the utility. Hence, with multiple observations per iteration, both

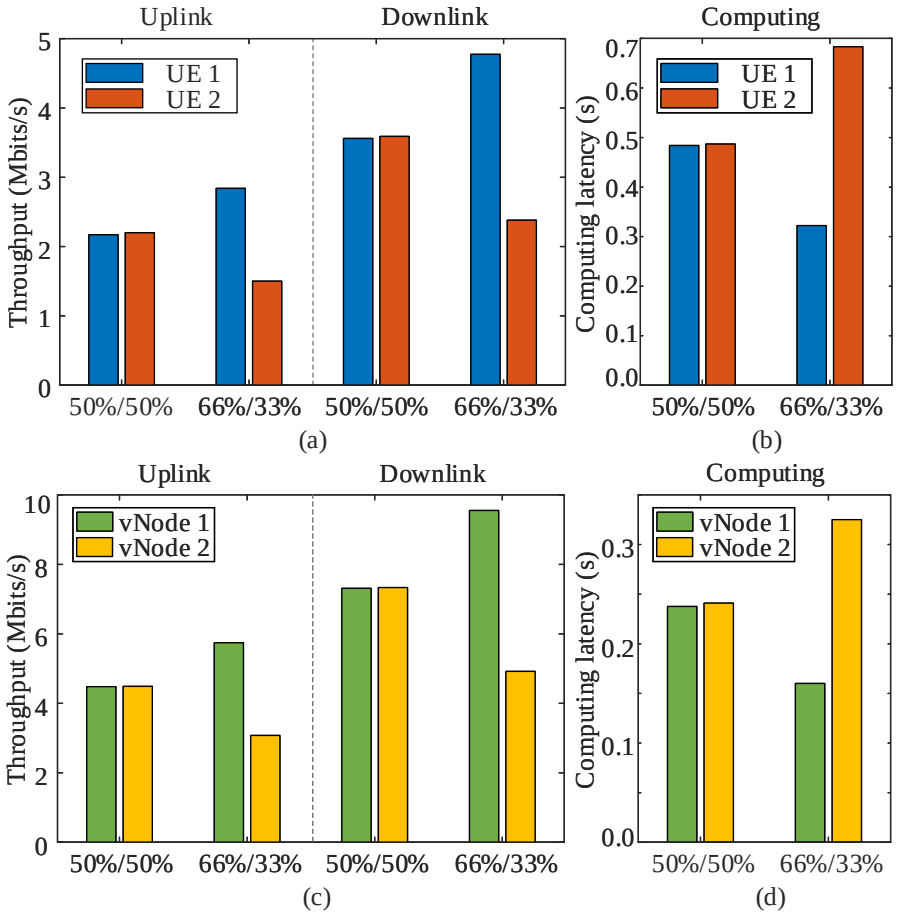

Figure 5.7: The functional and performance isolation.

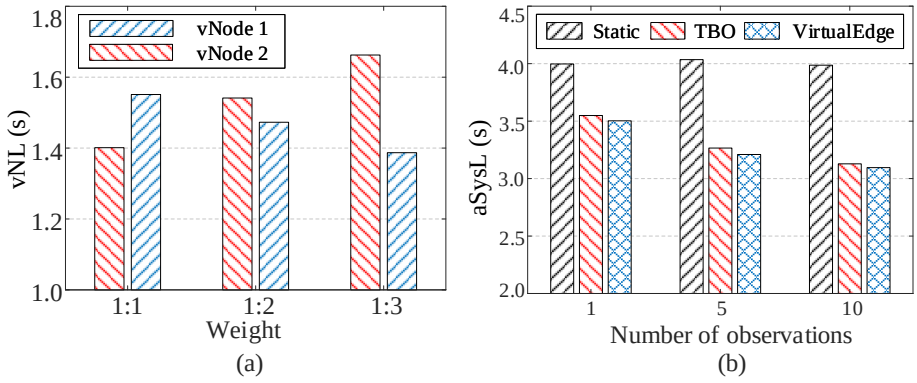

Figure 5.8: The impact of the weight and utility observations .

VirtualEdge and *TBO* need a longer time to converge to a relatively stable resource allocation solution.

5.4.5 Simulation Evaluation

The objective of the simulation is to evaluate the scalability of the multi-domain resource orchestration algorithm developed in the $VirtualEdge$ system. In the simulation, there are ten vNodes, and the number of users in vNode is randomly selected between one to five. The utility of the ith vNode is defined as the minus vNode latency

$$U_i = -\sum_{k \in \mathcal{K}} \left(\frac{D_k^u}{f_k(m_i)} + \frac{D_k^d}{g_k(n_i)} + \frac{C_k}{h_k(c_i)} \right) \tag{5.17}$$

where $f_k(m_i)$, $g_k(n_i)$ and $h_k(m_i)$ are the uplink data rate, downlink data rate, and virtual computing resources of the kth user, respectively. D_k^u, D_k^d and C_k are the corresponding uplink, downlink and computing resource demands in $Mbps$, $Mbps$, and $Credit/ms$ (credits per millisecond), respectively. We consider three types of applications whose $[D_k^u, D_k^d, C_k]$ are $[10, 1, 1]$, $[1, 10, 1]$ and $[1, 1, 10]$, respectively. We define the heterogeneity of an application as the ratio of the dominant resource to the average of the non-dominant resources. The total uplink and downlink resources are 100 $Mbps$, respectively. The total computing credit is generated at a speed of 100 $Credit/ms$.

To evaluate the performance of the proposed multi-domain resource orchestration algorithm, we implement a brute-force multi-domain resource orchestration algorithm based on the branch-and-bound search (BnBS) [64]. In the simulation, we use *BnBS* to represent the brute-force algorithm. Note that the brute-force algorithm is not practical in a real system due to its very high complexity. However, this algorithm provides a lower bound of the system latency.

Convergence: In Fig. 5.9 (a), we validate the convergence of the proposed algorithm. For the seek of simplicity, we use *VirtualEdge* to represent the multi-domain resource orchestration algorithm developed in the VirtualEdge system in the simulation results. In the simulation, *VirtualEdge* converges in about 40 iterations and

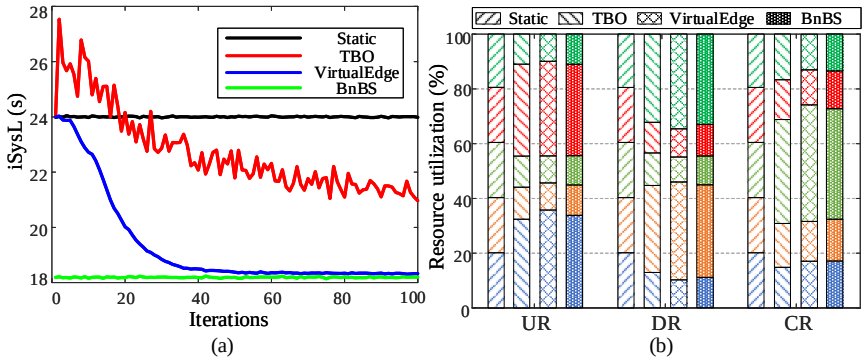

Figure 5.9: The simulation evaluation of the resource orchestration algorithm.

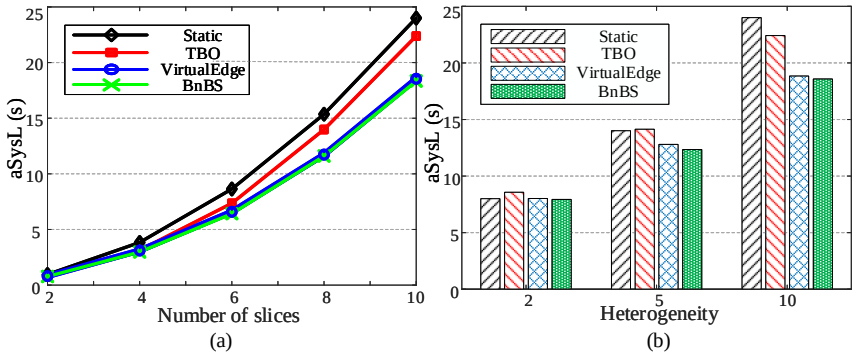

Figure 5.10: The simulation evaluation of the resource orchestration algorithm.

closely approximate the performance of *BnBS* in terms of iSysL. *TBO* can not converge because of the difficulty in learning the black-box function with a large number of variables. Besides, *VirtualEdge* significantly outperforms *TBO*, e.g., 20% reduction of aSysL. Fig. 5.9 (b) shows a snapshot on the multi-domain resource allocation of all systems. As shown in the figure, *VirtualEdge* and *BnBS* have very similar multi-domain resource allocations.

Scalability: Fig. 5.10 (a) evaluates the performance of *VirtualEdge* under different number of vNodes. Since the total resource is constrained, all systems have a longer average system latency (aSysL) under a large number of vNodes. However, the performance of *VirtualEdge* approximates that of *BnBS* very well. We also observe that the performance gap between *VirtualEdge* and *TBO* enlarges with the

increment of the number of vNodes. This is because the black-box function defined in the *TBO* has much more variables than that defined in *VirtualEdge*. For *TBO*, the number of variables in the black-box function equal to the number of vNodes multiplies by the number of resource types. For *VirtualEdge*, the number of variables in the black-box function is the number of resource types. With a large number of vNodes, *TBO* cannot efficiently solve the black-box optimization problem with the limited data observations and iterations, and thus results in a worse performance.

Heterogeneity: Fig. 5.10 (b) shows the impact of the application heterogeneity on the multi-domain resource orchestration. When the heterogeneity is small, there is not much room for the resource orchestration because an application requests a similar amount of resources from all the technical domains. Hence, the performance of the multi-domain orchestration algorithms, i.e., *VirtualEdge*, *TBO*, and *BnBS*, are similar to that of *Static*. When the heterogeneity is large, the multi-domain algorithms can learn the resource demands from vNodes and optimizes the resource allocation accordingly. Therefore, *VirtualEdge* achieves a significantly lower aSysL than the *Static* when the heterogeneity equals to ten. In the simulation, the demands of non-dominant resources are set to 1 $Mbps$ and 1 $Credit/ms$ for radio and computing resources, respectively. For example, if the uplink radio resource of an application is dominant and the heterogeneity of the application is 5, the demand of the uplink radio resource will be 5 $Mbps$. Therefore, in the simulation, when the application heterogeneity is larger, the vNode requires more resources. This is the reason why aSysL increases with respect to the heterogeneity of the applications in the simulation.

CHAPTER 6: DIRECT: DISTRIBUTED CROSS-DOMAIN RESOURCE ORCHESTRATION

In this chapter, we design the DIRECT protocol based on the alternating direction method of multipliers (ADMM) method and a new learning-assisted optimization (LAO) approach. To address the cross-domain virtual resource orchestration problem, we apply ADMM to decompose the problem into two subproblems. The first subproblem handles the resource orchestrations within an edge node while the second subproblem coordinates the resource allocation among edge nodes. Since the slice performance model is not available, we represent the performance of a slice using a black-box function. Therefore, the first subproblem becomes a black-box optimization problem. We solve this problem by designing a new learning-assisted optimization algorithm that constructs a probabilistic model for the black-box function and iteratively learn the gradients of the function with the observed data for the optimization. The second subproblem is a standard quadratic programming problem, and we solve it based on convex optimization.

6.1 System Model

In cellular edge computing, the performance of a network slice depends on resources from multiple technical domains. Here, we mainly consider the radio resources and computing resources from a radio access network and edge servers, respectively. On modeling the system, we introduce a logic network unit, edge node, which is composed of a cellular base station and a certain amount of computing resources. Then, cellular edge computing is composed of a collection of edge nodes. To provide seamless mobility and continuous services, network slices need resources from all edge nodes.

Let $\mathcal{I}$, $\mathcal{J}$ and $\mathcal{K}$ be the set of network slices, edge nodes and resources, respectively. Denote $x_{i,j,k}$ as the amount of the kth resource allocated to the ith network slice on the jth edge node, and let $\mathcal{X} = \{x_{i,j,k} | \forall i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}\}$ be the set of the resource allocations. Denote $\mathbf{x}_{i,j} = \{x_{i,j,k} | \forall k \in \mathcal{K}\}$ as the set of resources allocated to the ith network slice on the jth edge node. We define the utility function, i.e., performance, of the ith network slice on the jth edge node as $f_{i,j}(\mathbf{x}_{i,j})$. Denote $R_{j,k}^{tot}$ as the total amount of the kth resource on the jth edge node. The service provider pays the mobile network operator for running the network slice according to service level agreement. Denote $\gamma_{j,k}$ and Q_i^{tot} as the unit price of the kth resource on the jth edge node and the total payment of the ith network slice, respectively.

The objective of the mobile network operator is to maximize the sum utility of network slices on all edge nodes. Therefore, the cross-domain resource orchestration problem is formulated as

$$\begin{aligned} &\max_{\{x_{i,j,k} \geq 0\}} \quad \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} f_{i,j}(\mathbf{x}_{i,j}) \\ &s.t. \\ &C_1: \qquad \textstyle\sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} x_{i,j,k} \gamma_{j,k} \leq Q_i^{tot}, \forall i \in \mathcal{I}, \\ &C_2: \qquad \textstyle\sum_{i \in \mathcal{I}} x_{i,j,k} \leq R_{j,k}^{tot}, \forall j \in \mathcal{J}, k \in \mathcal{K}. \end{aligned} \tag{6.1}$$

Here, constraints C_1 ensure that the cost of the resources allocated to a network slice do not exceed the network slice's payment; constraints C_2 restrict that the amount of the kth resource allocated to network slices on the jth edge node should be less than the total amount of the kth resource on the jth edge node for all $j \in \mathcal{J}$ and $k \in \mathcal{K}$. For the sake of simplicity, we let $\gamma_{j,k} = 1$, $\forall j \in \mathcal{J}, k \in \mathcal{K}$ and simplify constraints C_1 as $\sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} x_{i,j,k} \leq Q_i^{tot}, \forall i \in \mathcal{I}$.

6.2 Distributed Resource Orchestration

In this section, we present the distributed resource orchestration algorithm that solves the problem in Eq.6.1. This problem is difficult to solve because all utility functions $f_{i,j}(\mathbf{x}_{i,j}), \forall i \in \mathcal{I}, j \in \mathcal{J}$ are unknown, have various mathematical properties, and are coupled by the constraints. To solve this problem, we decompose it to two subproblems using the ADMM method. The first subproblem is to optimize the resource orchestration within an edge node, where the utility functions of the network slices are unknown. This falls into the realm of black box optimization. However, the classic black box optimization methods, e.g., genetic algorithms and pattern search methods, have very high computation complexity and are not appropriate for solving the resource orchestration problem. Therefore, we design a new learning-assisted resource orchestration algorithm to solve the subproblem. The second subproblem is to coordinate the resource orchestration among network slices and can be efficiently solved by convex optimization methods.

6.2.1 Problem Decomposition

To decompose the problem, we introduce an auxiliary variable $z_{i,j,k}$ and let $\mathcal{Z} = \{z_{i,j,k} | \forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}\}$. Then, the problem in Eq.6.1 is equivalent to

$$
\begin{aligned}
&\max_{\{x_{i,j,k}, z_{i,j,k}\}} \quad \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} f_{i,j}(\mathbf{x}_{i,j}) \\
&s.t. \\
&C_1: \quad \textstyle\sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} z_{i,j,k} \le Q_i^{tot}, \forall i \in \mathcal{I}, \\
&C_2: \quad \textstyle\sum_{i\in\mathcal{I}} x_{i,j,k} \le R_{j,k}^{tot}, \forall j \in \mathcal{J}, k \in \mathcal{K}, \\
&C_3: \quad x_{i,j,k} = z_{i,j,k}, \forall i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}.
\end{aligned}
\tag{6.2}
$$

After the transformation, the problem has two set of variables, i.e., $\mathcal{X}$ and $\mathcal{Z}$, which are coupled by constraints C_3. Based on the ADMM method, we decompose the problem in Eq. 6.2 into two subproblems with variables $\mathcal{X}$ and $\mathcal{Z}$, respectively. Here,

constraints C_2 apply to the first subproblem whose variables are $\mathcal{X}$, while constraints C_1 apply to the second subproblem whose variables are $\mathcal{Z}$. Hence, we derive the augmented Lagrangian of the problem as

$$\begin{aligned}\mathcal{L}_u =& \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} f_{i,j}\left(\mathbf{x}_{i,j}\right) \\ &- \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} \frac{\rho}{2}\left\|x_{i,j,k} - z_{i,j,k} + u_{i,j,k}\right\|_2^2,\end{aligned} \tag{6.3}$$

where $\rho \geq 0$ is a positive constant, and $u_{i,j,k}$ is the scaled dual variable. Here, the augmented Lagrangian incorporates the coupling constraints C_3, and constraints C_1 and C_2 apply when the corresponding subproblems are solved. According to the ADMM method, the above problem is solved by alternatively solving the following subproblems

$$\mathcal{SP}1: \quad x_{i,j,k}^{(n+1)} = \arg\max_{x_{i,j,k}\in C_2} \mathcal{L}_u(x_{i,j,k}, z_{i,j,k}^{(n)}, u_{i,j,k}^{(n)}), \tag{6.4}$$

$$\mathcal{SP}2: \quad z_{i,j,k}^{(n+1)} = \arg\max_{z_{i,j,k}\in C_1} \mathcal{L}_u(x_{i,j,k}^{(n+1)}, z_{i,j,k}, u_{i,j,k}^{(n)}), \tag{6.5}$$

and updating the dual variables

$$u_{i,j,k}^{(n+1)} = u_{i,j,k}^{(n)} + (x_{i,j,k}^{(n+1)} - z_{i,j,k}^{(n+1)}). \tag{6.6}$$

6.2.2 Resource Orchestration on Edge Node

Since the constraints to $\mathcal{SP}1$ (constraints C_2) only restrict the resource allocation within an edge node, $\mathcal{SP}1$ can be solved independently on each edge node. Therefore,

to solve $\mathcal{SP}1$, each edge node addresses the following problem:

$$\begin{aligned} \max_{\{x_{i,j,k}\}} \quad & \sum_{i\in\mathcal{I}} f_{i,j}(\mathbf{x}_{i,j}) \\ & - \sum_{i\in\mathcal{I}} \sum_{k\in\mathcal{K}} \frac{\rho}{2} \left\| x_{i,j,k} - z_{i,j,k} + u_{i,j,k} \right\|_2^2 \\ s.t. \quad C_2: \quad & \textstyle\sum_{i\in\mathcal{I}} x_{i,j,k} \leq R_{j,k}^{tot}, \forall k \in \mathcal{K} \end{aligned} \tag{6.7}$$

where $z_{i,j,k}$ is assumed known. Since $f_{i,j}(\mathbf{x}_{i,j})$, which is the utility of the ith network slice on the jth edge node, is an unknown function, we develop a probabilistic model to represent it and iteratively learn its properties from observed data. Based on the properties, we design a gradient-based optimization algorithm to solve the problem [74].

Since the problem is solved within an edge node, for the sake of simplicity, we omit the subscript j in the math expression when deriving the solution in this section. Hence, $\mathbf{x}_i = \{x_{i,j,k} | \forall k \in \mathcal{K}\}$ and $f_i(\mathbf{x}_i)$ equal to $\mathbf{x}_{i,j} = \{x_{i,j,k} | \forall k \in \mathcal{K}\}$ and $f_{i,j}(\mathbf{x}_{i,j})$, respectively. Given $\mathbf{x}_i$, mobile network operator can observe $y_i = f_i(\mathbf{x}_i) + \epsilon$ which contains Gaussian noises $\epsilon \sim \mathcal{N}(0, \delta^2)$. Let $\mathbf{x}_i^{1:t}$ and $y_i^{1:t}$ as the set of resource allocations and the corresponding observation in t iterations. With the observations $\mathcal{D}_i^{1:t} = \{\mathbf{x}_i^{1:t}, y_i^{1:t}\}$, the posterior distribution of $f_i(\mathbf{x}_i)$ can be expressed as

$$P(f_i(\mathbf{x}_i)|\mathcal{D}_i^{1:t}) \propto P(\mathcal{D}_i^{1:t}|f_i(\mathbf{x}_i))P(f_i(\mathbf{x}_i)). \tag{6.8}$$

We adopt the Gaussian process (GP) to model the prior distribution of $f_i(\mathbf{x}_i)$ [75]. Hence, $f_i(\mathbf{x}_i)$ can be described as $f_i(\mathbf{x}_i) \sim \mathcal{GP}(\mu(\mathbf{x}_i), c(\mathbf{x}_i, \mathbf{x}_i'))$ where $\mu(\mathbf{x}_i)$ and $c(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{1}{2}||\mathbf{x}_i - \mathbf{x}_j||^2)$ are the mean function and covariance function, respectively.

Given a resource allocation $\mathbf{x}_i^*$, the posterior distribution at the tth iteration can be derived as

$$P(f_i(\mathbf{x}_i^*)|\mathcal{D}_i^{1:t}, \mathbf{x}_i^*) \sim \mathcal{N}(\mu(\mathbf{x}_i^*), \sigma_i^2(\mathbf{x}_i^*)), \tag{6.9}$$

where

$$\mu(\mathbf{x}_i^*) = \mathbf{c}^T[\mathbf{C} + \delta^2 I]^{-1} y_i^{1:t}. \tag{6.10}$$

$\sigma_i^2(\mathbf{x}_i^*)$ can be derived as

$$\sigma_i^2(\mathbf{x}_i^*) = c(\mathbf{x}_i^*, \mathbf{x}_i^*) - \mathbf{c}^T[\mathbf{C} + \delta^2 I]^{-1}\mathbf{c} \tag{6.11}$$

where $\mathbf{c} = [c(\mathbf{x}_i^*, \mathbf{x}_i^1), c(\mathbf{x}_i^*, \mathbf{x}_i^2), \cdots, c(\mathbf{x}_i^*, \mathbf{x}_i^t)]$ and

$$\mathbf{C} = \begin{bmatrix} c(\mathbf{x}_i^1, \mathbf{x}_i^1) & \cdots & c(\mathbf{x}_i^1, \mathbf{x}_i^t) \\ \vdots & \ddots & \vdots \\ c(\mathbf{x}_i^t, \mathbf{x}_i^1) & \cdots & c(\mathbf{x}_i^t, \mathbf{x}_i^t) \end{bmatrix}. \tag{6.12}$$

Based on the posterior distribution, we define the predictive gradient of $f_i(\mathbf{x}_i^t)$ as

$$\nabla \bar{f}_i(\mathbf{x}_i^t) \coloneqq \frac{1}{\tau} \begin{bmatrix} \mu([\mathbf{x}_{i,1}^t + \tau, \mathbf{x}_{i,2}^t, ..., \mathbf{x}_{i,K}^t]) - y_i^t \\ \mu([\mathbf{x}_{i,1}^t, \mathbf{x}_{i,2}^t + \tau, ..., \mathbf{x}_{i,K}^t]) - y_i^t \\ \vdots \\ \mu([\mathbf{x}_{i,1}^t, \mathbf{x}_{i,2}^t, ..., \mathbf{x}_{i,K}^t + \tau]) - y_i^t \end{bmatrix}, \tag{6.13}$$

where $\mathbf{x}_{i,k}^t$ is the amount of the kth resource allocated to the ith network slice at the tth iteration, and τ is small positive constant.

With the predictive gradient, we solve the resource orchestration problem on the edge node using the proximal gradient descent method [64]. According to this method, the resource allocation to the ith slice is updated as follow

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \lambda \cdot (\nabla \bar{f}_i(\mathbf{x}_i^t) - \rho(\mathbf{x}_i^t - \mathbf{z}_i^t + \mathbf{u}_i^t)), \tag{6.14}$$

where $\lambda = [\lambda_1, \lambda_2, ..., \lambda_K]^T$ are appropriate step sizes, and $\cdot$ is the dot product operator. To prevent the updated resource allocation $\mathbf{x}_i^{t+1}, \forall i \in \mathcal{I}$ from violating resource

Algorithm 6: Resource Orchestration: Edge Node

Input: $R_{j,k}^{tot}$, $u_{i,j,k}$ and $z_{i,j,k}$, $\forall i \in \mathcal{I}, k \in \mathcal{K}$.
Output: $\mathbf{x}_i, \forall i \in \mathcal{I}$, $u_{i,j,k}$,$\forall i \in \mathcal{I}, k \in \mathcal{K}$.

Initialize $\mathbf{x}_i, \mathcal{D}_i, \forall i \in \mathcal{I}$, and set $t \leftarrow 1$;
while *True* **do**
  $/ * *$*query the utility of slices* $* */$;
  Given $\mathbf{x}_i$, query to observe $f_i(\mathbf{x}_i), \forall i \in \mathcal{I}$;
  $/ * *$*update the observation set* $* */$;
  $\mathcal{D}_i^{1:t} = \{\mathbf{x}_i^{1:t}, y_i^{1:t}\}, \forall i \in \mathcal{I}$;
  $/ * *$*update prediction for slices* $* */$;
  $\mu(\mathbf{x}_i^t) \leftarrow Eq.\ 6.10, \sigma_i^2(\mathbf{x}_i^t) \leftarrow Eq.\ 6.11, \forall i \in \mathcal{I}$;
  $/ * *$*compute predictive gradients* $* */$;
  $\nabla \bar{f}_i(\mathbf{x}_i^t) \leftarrow Eq.\ 6.13, \forall i \in \mathcal{I}$;
  $/ * *$*update allocation based on gradients* $* */$;
  $\mathbf{x}_i^{t+1} \leftarrow Eq.\ 6.14, \forall i \in \mathcal{I}$;
  $/ * *$*project allocation with constraints* $* */$;
  $\mathbf{x}_k^{t+1} = P_{\Omega_k}\left(\mathbf{x}_k^{t+1}\right), \forall k \in \mathcal{K}$;
  if $||\mathbf{x}_i^{t+1} - \mathbf{x}_i^t|| \leq \eta, \forall i \in \mathcal{I}$ **then**
    break;
  $t \leftarrow t + 1$;
return $\mathbf{x}_i, \forall i \in \mathcal{I}$, $u_{i,j,k}$,$\forall i \in \mathcal{I}, k \in \mathcal{K}$.

constraints on the edge node, we project $\mathbf{x}_i^{t+1}, \forall i \in \mathcal{I}$ into a bounded domain that satisfies the constraint of each type resource, i.e., constraints C_2 in the problem. We define $P_\Omega(x) = \arg\min_{y \in \Omega} \|x - y\|^2$ as the Euclidean projection of x on bounded domain Ω. Denote $\mathbf{x}_k^{t+1} = \{\mathbf{x}_{i,k}^{t+1} | \forall i \in \mathcal{I}\}$ as the amount of the kth resource allocated to all network slices on the edge node. Then, $\mathbf{x}_k^{t+1}$ is projected as

$$\mathbf{x}_k^{t+1} = P_{\Omega_k}\left(\mathbf{x}_k^{t+1}\right), \tag{6.15}$$

where Ω_k is the bounded domain of the kth resource. The algorithm stops when the resource allocations satisfy $||\mathbf{x}_i^{t+1} - \mathbf{x}_i^t|| \leq \eta, \forall i \in \mathcal{I}$. The pseudo code of the algorithm on the edge node is shown in Alg. 6.

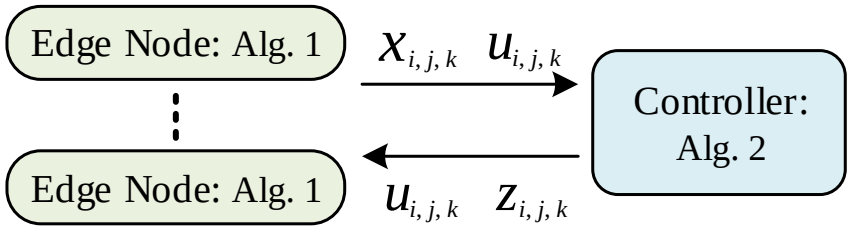

Figure 6.1: The distributed resource orchestration.

6.2.3 Resource Orchestration on Controller

The controller is responsible to solve $\mathcal{SP}2$. Since $\mathcal{X}$ is assumed known, $\mathcal{SP}2$ can be equivalently transformed to

$$\begin{aligned} \min_{\{z_{i,j,k}\}} \quad & \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} \tfrac{\rho}{2}\left\|x_{i,j,k}-z_{i,j,k}+u_{i,j,k}\right\|_2^2 \\ s.t. \quad C_1: \quad & \textstyle\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} z_{i,j,k} \le Q_i^{tot}, \forall i\in\mathcal{I}. \end{aligned} \tag{6.16}$$

This is a standard quadratic programming problem, and we solve it using convex optimization tools, e.g., CVX [64]. After solving the problem, the controller updates the dual variables $u_{i,j,k}^{(n+1)}$ according to Eq. 6.6. The pseudo code of the controller side algorithm is shown in Alg. 7.

6.2.4 Algorithm Analysis

Fig. 6.1 illustrates the distributed resource orchestration algorithm. In the beginning, each edge node derives an initial resource allocation to the network slices and sends the resource allocation and the dual variables to the controller. After receiving the initial resource allocations from all edge nodes, the controller coordinates the resource orchestration by updating the auxiliary variables, $z_{i,j,k}$, and the dual variables. The updated variables are fed back to edge nodes for the next round of resource allocation. The orchestration process stops when the resource allocations converge. We prove the convergence in Corollary 1.

Proposition 2. *$\nabla\bar{f}_i(\boldsymbol{x}_i), \forall i\in\mathcal{I}$ are the controllably accurate gradient approximations*

Algorithm 7: Resource Orchestration: Controller

Input: Q_i^{tot}, $x_{i,j,k}$ and $u_{i,j,k}$, $\forall i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}$.
Output: $x_{i,j,k}$, $u_{i,j,k}$, $z_{i,j,k}$, $\forall i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}$.

/ * * *determine algorithm convergence* * */;
if $||\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} x_{i,j,k} - Q_i^{tot}|| \leq \eta, \forall i \in \mathcal{I}$ **then**
    return $x_{i,j,k}$, $\forall i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}$;
else
    / * * *update z in the controller* * */;
    $z_{i,j,k} \leftarrow \arg\max_{z_{i,j,k}\in C_1} \mathcal{L}_u(x_{i,j,k}, z_{i,j,k}, u_{i,j,k})$;
    / * * *update u in the controller* * */;
    $u_{i,j,k} \leftarrow u_{i,j,k} + (x_{i,j,k} - z_{i,j,k})$;
    return $z_{i,j,k}$ *and* $u_{i,j,k}$, $\forall i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}$.

of $\nabla f_i(\boldsymbol{x}_i), \forall i \in \mathcal{I}$ *if* $f_i(\boldsymbol{x}_i), \forall i \in \mathcal{I}$ *are Lipschitz continuous.*

Proof. Denote $f_i(\mathbf{x}_i)$ and $\bar{f}_i(\mathbf{x}_i)$ as the utility and predicted utility of the ith slice at $\mathbf{x}_i$, respectively. The predictive gradients are defined as $\nabla\bar{f}_i(\mathbf{x}_i) = (\bar{f}_i(\mathbf{x}_i + \tau) - f_i(\mathbf{x}_i))/\tau = (\mu(\mathbf{x}_i + \tau) - f_i(\mathbf{x}_i))/\tau, \forall i \in \mathcal{I}$ (Eq. 6.13). Then,

$$|\nabla\bar{f}_i(\mathbf{x}_i) - \nabla f_i(\mathbf{x}_i)| = \frac{1}{\tau}|(\mu(\mathbf{x}_i + \tau) - f_i(\mathbf{x}_i + \tau))| \tag{6.17}$$
$$= \frac{1}{\tau}(|\mathbf{c}^T[\mathbf{C} + \delta^2 I]^{-1} y_i^{1:t} - f_i(\mathbf{x}_i + \tau)|),$$

where $y_i = f_i(\mathbf{x}_i) + \epsilon$. Since the utility functions $f_i(\mathbf{x}_i), \forall i \in \mathcal{I}$ are Lipschitz continuous [64], there exists a positive real constant κ such that, for all real $\mathbf{x}_i^1$ and $\mathbf{x}_i^2$,

$$|f_i(\mathbf{x}_i^1) - f_i(\mathbf{x}_i^2)| \leq \kappa|\mathbf{x}_i^1 - \mathbf{x}_i^2|, \forall i \in \mathcal{I}. \tag{6.18}$$

Hence, we obtain

$$|\nabla\bar{f}_i(\mathbf{x}_i) - \nabla f_i(\mathbf{x}_i)| \leq \Delta, \tag{6.19}$$

where Δ is a constant. According to Definition 10.1 in [76], the $\nabla\bar{f}_i(\mathbf{x}_i), \forall i \in \mathcal{I}$ are the controllably accurate gradient approximations of $\nabla f_i(\mathbf{x}_i), \forall i \in \mathcal{I}$. □

Corollary 1. *The distributed resource orchestration algorithm converges to a local*

optimum if $f_i(\boldsymbol{x}_i), \forall i \in \mathcal{I}$ are non-decreasing and Lipschitz continuous.

Proof. The distributed resource orchestration algorithm consists of the controller-side and edge-node-side algorithms which iteratively exchanges information. The controller-side algorithm, i.e., Alg. 7, simply updates the auxiliary and dual variables, and there is no need to prove its convergence. Hence, to show the convergence of the distributed resource orchestration algorithm, we prove that the edge-node-side algorithm, i.e., Alg. 6, converges, and the iterative information exchanges lead to a converged resource allocation.

Convergence of Alg. 6: This algorithm is designed based on gradient-based optimization. According to Theorem 10.6 in [76], the convergence of the algorithm can be proved by showing 1) the predictive gradients $\nabla \bar{f}_i(\mathbf{x}_i), \forall i \in \mathcal{I}$ are the controllably accurate gradient approximations of $\nabla f_i(\mathbf{x}_i), \forall i \in \mathcal{I}$; and 2) the step size is positive in the gradient-based descent method. The first condition has been proved in *Proposition* 2. Since the step size in the algorithm is positive, the algorithm satisfies both conditions required for the convergence. Therefore, the convergence of Alg. 6 is proved.

Convergence of iterative information exchanges: The iterative information exchanges are designed based on the ADMM method. Therefore, we prove the convergence of the iterative information exchanges based on the convergence proofs of the ADMM method [86, 87]. First, we prove the intermediate variables, $\mathcal{X}$, $\mathcal{Z}$ and $\mathcal{U}$, and augmented Lagrangian $\mathcal{L}_u$ are bounded. Second, we prove that there exists positive constants α_x and α_z such that $\mathcal{L}_u^{t+1} - \mathcal{L}_u^t \leq \alpha_x|\mathbf{x}^{t+1} - \mathbf{x}^t| + \alpha_z|\mathbf{z}^{t+1} - \mathbf{z}^t|$; Thus, $\mathcal{L}_u$ is monotonically non-decreasing and lower bounded. Third, we prove that there exists positive constants α_x, α_z and $d^* \in \partial\mathcal{L}$ such that $|d^*| \leq \alpha_x|\mathbf{x}^{t+1} - \mathbf{x}^t| + \alpha_z|\mathbf{z}^{t+1} - \mathbf{z}^t|$. Therefore, when $t \to \infty$, $|d^*| \to 0$. Fourth, we prove that if $(\mathcal{X}^*, \mathcal{Z}^*, \mathcal{U}^*)$ is the limit point of generated sequence $(\mathcal{X}^{1:t}, \mathcal{Z}^{1:t}, \mathcal{U}^{1:t})$, we obtain $\mathcal{L}_u(\mathcal{X}^*, \mathcal{Z}^*, \mathcal{U}^*) = \lim_{t\to\infty} \mathcal{L}_u(\mathcal{X}^t, \mathcal{Z}^t, \mathcal{U}^t)$. Therefore, the iterative information exchanges between edge nodes and the controller

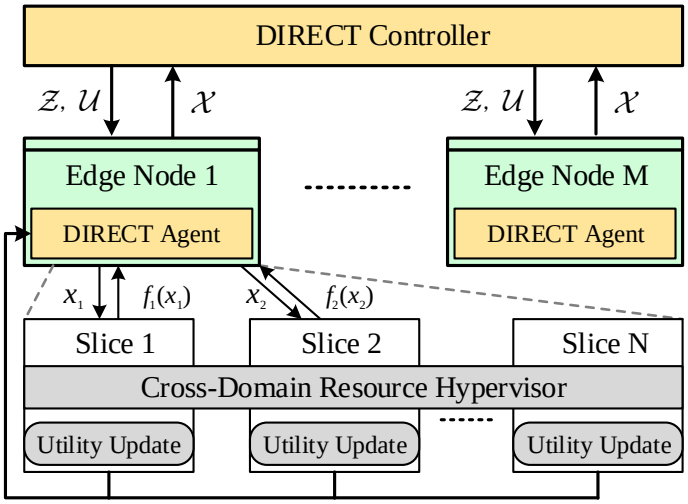

Figure 6.2: The design of DIRECT protocol.

converge. Due to the space limitation, we omit the detail convergence proof.

Since both Alg. 6 and the iterative information exchanges are convergent, the convergence of the distributed resource orchestration algorithm is proved. □

6.3 Protocol Design and Implementation

In this section, we design the DIRECT protocol according to the distributed resource orchestration algorithm and implement the protocol in a small-scale testbed developed based on the LTE and GPU computing platforms [77, 83].

6.3.1 Protocol Design

Fig. 6.2 illustrates the design of the DIRECT protocol which mainly consists of a DIRECT controller and multiple DIRECT agents on edge nodes. To realize the protocol, we also develop a cross-domain resource hypervisor to manage the resource virtualization. The DIRECT controller runs the controller-side algorithm, i.e., Alg. 7, to coordinate the cross-domain resource orchestration among network slices and ensure that each network slice is properly served according to its service level agreement. It coordinates the resource orchestration by controlling the auxiliary variables, $\mathcal{Z}$, and the dual variables, $\mathcal{U}$, which are fed back to the DIRECT agent. On the edge node,

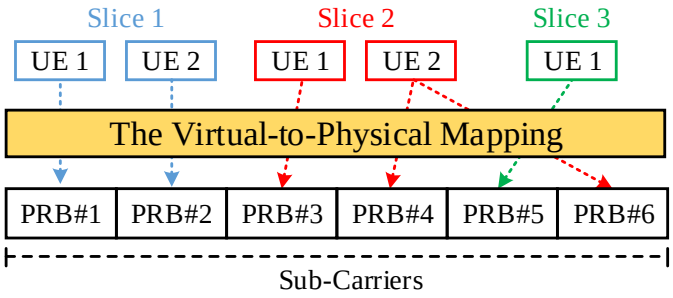

Figure 6.3: The radio resource virtualization.

the DIRECT agent runs the node-side algorithm, i.e., Alg. 6, to allocate the multiple resources to network slices for maximizing the sum utility under resource constraints of the edge node. We develop a cross-domain resource hypervisor on each edge node to enable the coexistence of multiple network slices on the same physical infrastructure, e.g., base station and edge server. Hence, the agent can maintain network slices operations, such as creation, suspension and deletion. The DIRECT agent manages the hypervisor for the resource virtualization as well as the slice creation, suspension and deletion. On the edge node, the resources allocated to network slices are mapped to physical infrastructure at runtime, and the utility of network slices are reported to the DIRECT agent.

6.3.2 Protocol Implementation

To realize the DIRECT protocol, we develop a cross-domain resource hypervisor to dynamically manage the physical resource according to the resource allocation. Meanwhile, the hypervisor ensures the performance and functional isolations among network slices during the resource orchestration. In the protocol implementation, we consider the radio and computing resources in the context of LTE and CUDA GPU programming, respectively.

6.3.2.1 Radio Resource Hypervisor

In the implementation, the network slices on an edge node share the same control plane operations following standard LTE protocols. To differentiate users among net-

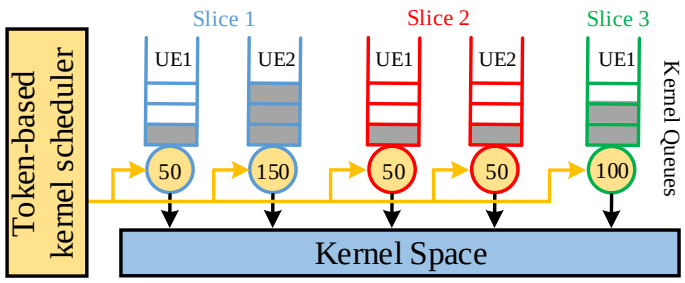

Figure 6.4: The computing resource virtualization.

work slices, we implement a user-association function in the control plane to associate users to their corresponding network slices. The radio resource hypervisor focuses on managing the uplink/downlink resources (URs/DRs), i.e., physical resource blocks (PRBs) of PUSCH/PDSCH, in the LTE user plane. We define the resource allocated to users by network slices as the virtual resource. As shown in Fig. 6.3, the radio resource hypervisor maps the virtual radio resources that allocated to users by network slices to PRBs. During the resource mapping, we maximize the network throughput by selecting the user with the best channel condition for each PRB. After all virtual resources are mapped, the surplus PRBs are allocated to the users who have the best channel condition. Since the single carrier-FDMA (SC-FDMA) is the access method in LTE uplink, the PRBs allocated to a single user must be contiguous in frequency domain [78]. Hence, the hypervisor allocates the surplus PRBs to the users whose current PRBs and the surplus PRBs are contiguous.

6.3.2.2 Computing Resource Hypervisor

In the CUDA programming model, an application can launch multiple *kernels* that can be concurrently executed by massive CUDA *threads*. To realize dynamic computing resource management, we develop a token-based kernel scheduler to control the execution of *kernels*. As illustrated in Fig. 6.4, the kernel scheduler dispatches the *kernels* commands according to the computing tokens of network slices. In the user space, we add a *KernelSpawn* function to push the *kernel* commands into a FIFO

queue. A user's *kernel* command is pulled out of the queue and enter the kernel space if the user has sufficient tokens. Here, the DIRECT agent allocates virtual resources to a network slice, and the network slice distributes the resources to its users. The computing resource hypervisor covers a user's virtual computing resources into tokens. The *KernelSpawn* function is running in a system thread with non-blocking property to ensure the asynchronous executions of *kernels*.

6.4 Performance Evaluation

In this section, we validate the performance of the DIRECT protocol through experiments in a small-scale system prototype and network simulations.

6.4.1 System Prototype

We develop a small-scale prototype as shown in Fig. 6.5 to evaluate the performance of the DIRECT protocol. In the prototype, we consider three resources from two technical domains: for the radio access network, we consider uplink and downlink radio resources; for edge computation, we consider the GPU resources for hardware-accelerated computing. The radio and computing resources are essential for killer applications in 5G such as mobile cross reality and autonomous driving [31]. The prototype consists of two edge nodes, and each edge node is composed of an eNodeB and a GPU. We place two eNodeBs in a different room to emulate a cellular network with limited co-channel interference. The radio access network is implemented based on the OpenAirInterface (OAI) LTE platform [77], and the core network is built with openair-cn [81]. The computing platform is NVIDIA CUDA-enable GPU [83]. We use a dell alienware desktop (Intel i7 8700 @3.2GHz, 64GB RAM) with two NVIDIA GEFORCE GTX 1080Ti (3584 CUDA cores, 11G RAM) for deploying the DIRECT controller, DIRECT agents and mobile core network. We use two desktop computers with a low-latency kernel (Intel i5 4590@3.3GHz, 16 RAM) to deploy the eNodeBs and emulate mobile users with Huawei E3372h LTE dongles. Two Ettus USRP B210

Table 6.1: User Association

	Slice 1	Slice 2	Slice 3
Edge Node 1	1	0	1
Edge Node 2	0	1	1

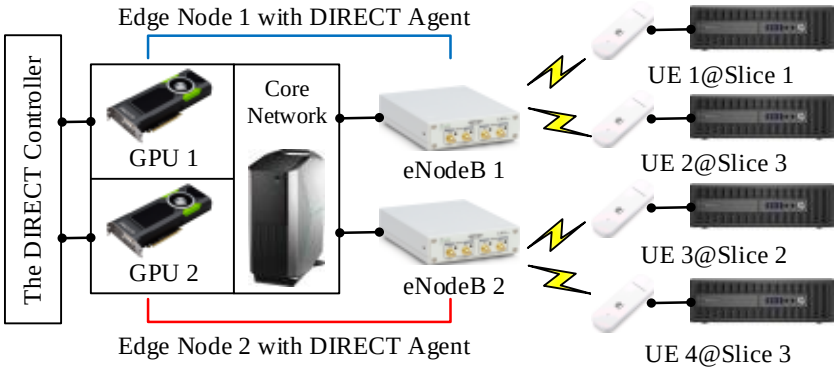

Figure 6.5: The testbed implementation of DIRECT protocol.

SDR boards are used as the RF front-end of eNodeBs. The uplink and downlink carrier frequencies are 2.655GHz and 2.535GHz, respectively (LTE Band 7). Both the uplink and downlink bandwidths are 5MHz (25 PRBs).

In the experiment, we create three network slices to serve four mobile users on the edge nodes[1]. The user-slice-edge node association is listed in Table 6.1.

We adopt the negative slice-latency as the utility of a slice, i.e., $f_{i,j}(\mathbf{x}_{i,j}), \forall i \in \mathcal{I}, j \in \mathcal{J}$. Here, the slice-latency is defined as the sum latency of all users in a slice across all edge nodes. The edge-latency is defined as the summation of the weighted slice-latency of all network slices in an edge node. The system-latency is the summation of the slice-latency of all slices. Hence, maximizing the utility of slices is equivalent to minimizing the slice-latency of slices. Here, we do not study the user scheduling algorithm within a network slice and thus assume that a network slice evenly allocates resources to its users.

[1]Since the numbers of network slices and users are small in the system prototype, we evaluate the scalability of the DIRECT protocol in simulations.

6.4.2 Compute-intensive Applications

In the experiments, we implement two compute-intensive mobile applications based on the YOLO object detection algorithm to evaluate the performance of the DIRECT protocol [84]. These applications are mobile augmented reality (MAR) and video analytics and streaming (VAS). The first and second network slices support the MAR application, and the third slice supports the VAS application.

Mobile Augmented Reality (MAR): A client continuously sends video frames with the resolution of 1280x720 to the server and receives the detection results. The server receives the frames, executes the YOLO 608x608 algorithm, and sends the detection results back to the client. MAR represents the type of applications that have heavy uplink traffic loads and intensive computing workloads. Here, YOLO 608x608 means the YOLO algorithm tuned at the image resolution of 608x608.

Video Analytics and Streaming (VAS): A client sends a streaming request to the server. The server retrieves the real-time camera frames with the resolution of 1280x720, processes it with the YOLO 416x416 algorithm, and sends the frames with detection results back to the client. VAS represents the type of applications that have heavy downlink traffic loads and moderate computing workloads. Here, YOLO 416x416 is less compute-intensive than YOLO 608x608.

6.4.3 Comparison Protocols

We compare DIRECT with following protocols:

- **Static:** With Static, a network slice evenly distributes its payment, Q_i^{tot}, to acquire resources from all technical domains on all edge nodes.
- **PSwarm:** PSwarm is a global optimization solver [88] for bounded and linear constrained derivative-free problems. In this protocol, PSwarm replaces Alg. 1 in the distributed resource orchestration.
- **TOMLAB:** The TOMLAB with *glcSolve* is a optimization solver [89] that han-

dles the global mixed-integer nonlinear programming problems. In this protocol, TOMLAB replaces Alg. 1 in the distributed resource orchestration.

PSwarm and TOMLAB have very high computation complexity and communication overhead and are impractical in a real system. Therefore, we compare the performance of DIRECT with those protocols in network simulations.

6.4.4 Experimental Evaluation

Convergence: Figs. 6.6 (a) and (b) show the system-latency and resource allocation gap versus the number of iterations. In the experiment, the resource allocation gap is defined as $\sum_{i\in\mathcal{I}} |\sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} x_{i,j,k} - Q_i^{tot}|$. When the resource allocation gap reaches zero, the resource orchestration algorithm converges. The experiment result shows that DIRECT converges after 5 iterations. DIRECT reduces about 21% system-latency as compared to the Static protocol which has an almost constant system-latency because of its static resource allocation. The DIRECT protocol has a small fluctuation after the convergence because of the dynamic wireless channel conditions. Fig. 6.6 (c) shows the edge-latency of edge nodes versus the number of iterations and reflects the convergence of Alg. 6. It shows that Alg. 6 converges after 15 iterations and significantly reduces the edge-latency.

Resource allocation: Fig. 6.7 (a) shows resource allocations of network slices on different edge nodes. We can observe that all resources of the first and second slices are allocated to the first and second edge nodes, respectively. This resource allocation matches the traffic workload in the experiment where the first and second slices do not have users in the second and first edge node, respectively. Fig. 6.7 (b) shows the resource utilization in different technical domains of network slices. It can be seen that the third slice utilizes more downlink radio resources than other types of resources because the slice supports VAS applications which have heavier downlink traffic loads These results indicate that DIRECT is able to allocate multi-domain resources according to the traffic workload among edge nodes and the characteristics

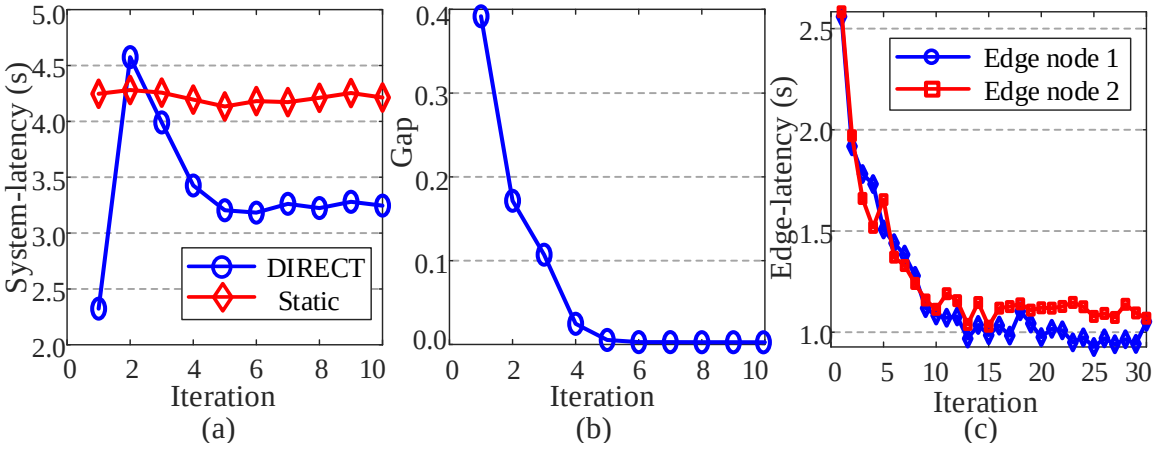

Figure 6.6: The convergence of the DIRECT protocol.

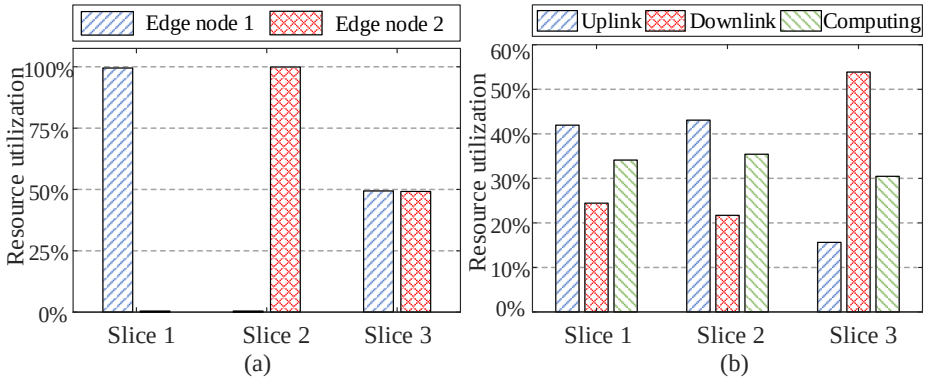

Figure 6.7: The cross-domain resource allocation.

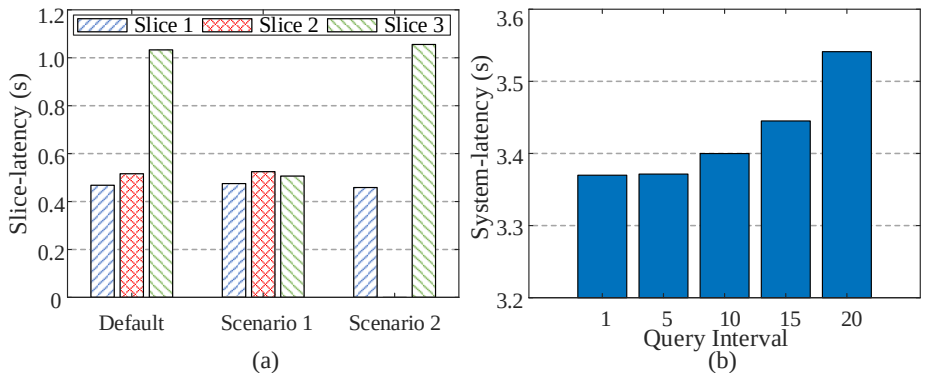

Figure 6.8: The query interval and isolation performance.

of applications served by the slices.

Performance Isolation: Fig. 6.8 (a) shows the slice-latency under different scenarios with the DIRECT protocol. In the first scenario, we remove a user from slice 3. The slice-latency of slice 3 is reduced since the DIRECT protocol adapts to the

traffic workload of slice among edge nodes and allocates all its resources to the single user. In the second scenario, we remove a user from slice 2. As a result, the utility of slice 2 is zero since there are no associated users. From the experiment, it can be observed that the slice-latency of a network slice will not affect or be affected by the traffic variations of other slices, which means that DIRECT ensures the performance isolation among network slices.

Query Interval: In the DIRECT protocol, the resource orchestration algorithm on edge nodes, i.e., Alg. 6, queries the utility of network slices, i.e., $f_i\left(\mathbf{x}_i\right), \forall i \in \mathcal{I}$, for the resource allocation. The frequency of the utility queries impacts the system latency and the convergence speed of the algorithm. More utility queries lead to a lower system latency but a slower convergence speed. Fig. 6.8 (b) shows the system-latency versus different query intervals. A large query interval means a lower query frequency. It can be seen from the figure that a larger query interval leads to a longer system-latency. This because a larger query interval may incur inaccurate predictions of the predictive gradients.

6.4.5 Simulation Evaluation

Here, we aim to evaluate the scalability of the DIRECT protocol through network simulations. In the simulation, there are 4 edge nodes and 6 network slices with 3 types of resources. The number of users in each network slice and edge node is random with a range of one to five. The utility function of the ith slice in jth edge node is defined as

$$f_{i,j}(\mathbf{x}_{i,j}) = \sum\nolimits_{k\in\mathcal{K}} A_k \cdot (x_{i,j,k})^{\beta}, \tag{6.20}$$

where $x_{i,j,k}$ is the kth resource of the ith slice in the jth edge node, A_k is the weight for the kth resource, and β is a parameter controls the property of the utility function. The utility functions are used by individual slices to calculate their utilities given resource allocations in the simulation. The resource orchestrator does not know the

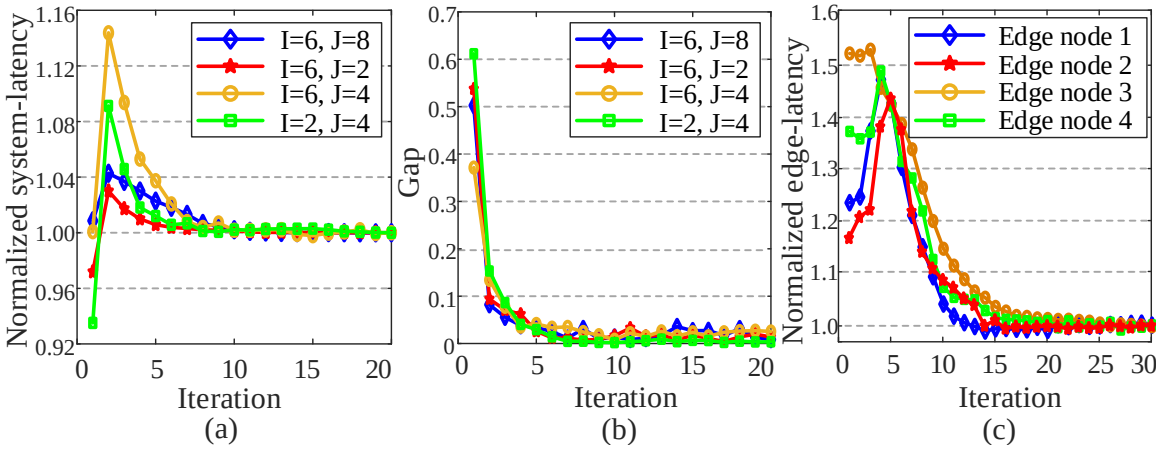

Figure 6.9: The convergence of the DIRECT protocol.

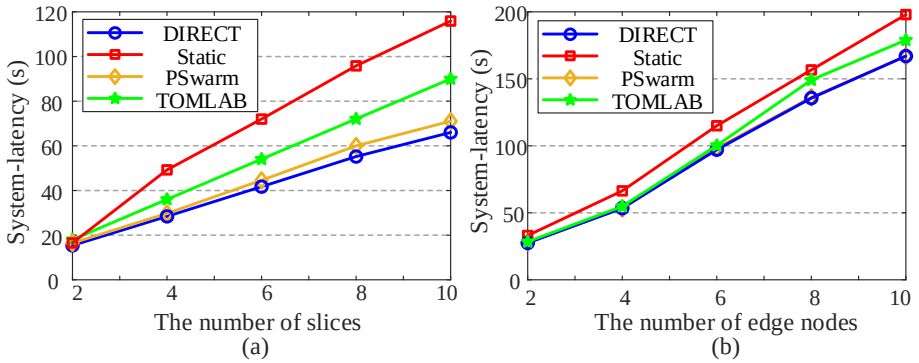

Figure 6.10: The scalability of the DIRECT protocol.

utility functions. When β is positive, the utility of slices $f_{i,j}$ is positively correlated to the allocated resources $x_{i,j,k}$, e.g., more resources lead to a larger utility. For example, the network throughput can be such a utility. The network aims to maximize the throughput. When β is negative, the utility of slices $f_{i,j}$ is negatively correlated to the allocated resources $x_{i,j,k}$, e.g., more resources result in a smaller utility. For example, the network latency can be such a utility. The network aims to minimize the latency. In the simulations, the default value of β is -1. The total amount of the kth resource on the jth edge nodes is constrained by $R_{j,k}^{tot} = 100, \forall j \in \mathcal{J}, k \in \mathcal{K}$. The weights, $A_k, \forall k \in \mathcal{K}$, are generated according to a uniform distribution between one and ten.

Convergence: Fig. 6.9 validates the convergence of DIRECT under different number of slices and edge nodes. In the simulation, the system latency under a simulation

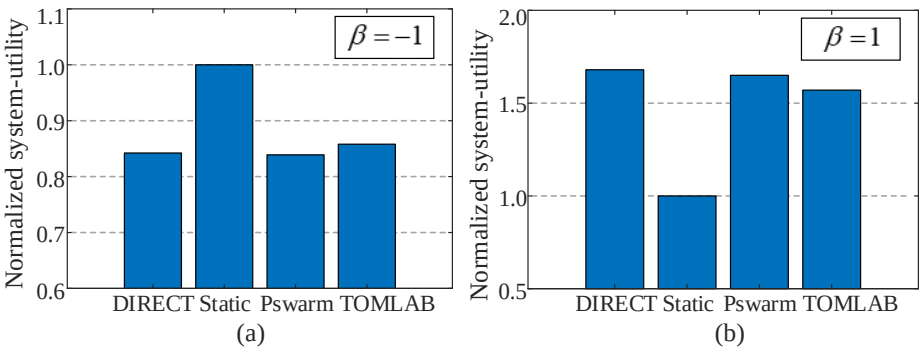

Figure 6.11: The impact of utility function.

setting is normalized with respect to the optimal system latency derived by DIRECT under the same simulation setting. The DIRECT protocol converges in about ten iterations with nearly zero gap, for all simulation settings. Meanwhile, the convergence of the Alg. 6 and service latency in edge nodes are also shown in Fig. 6.9 (c). The simulation results show that the properties of black-box function can be learned in several iterations, and the proposed algorithm gradually converges and significantly reduces the edge-latency of edge nodes.

Scalability: Fig. 6.10 evaluates the performance of the DIRECT protocol under different number of slices and edge nodes. The performance gap between the DIRECT and Static protocol enlarges with the increment of number of slices and edge nodes. This is because the Static protocol cannot adapt to the resource requirements of slice service that results in a high system-latency. The simulation results also show that the DIRECT protocol outperforms both the PSwarm and TOMLAB. Although the performance of the DIRECT is only slightly better than that of the PSwarm, both PSwarm and TOMLAB have a very high communication overhead and complexity and cannot be implemented in a practical system.

Utility functions: Fig. 6.11 shows the performance of the DIRECT protocol with different utility functions. When $\beta = -1$, i.e., lower utility is preferred, the DIRECT protocol reduces the utility of system by 17% as compared to the Static protocol. When $\beta = 1$, i.e., higher utility is preferred, the DIRECT protocol improves 70%

utility of system as compared to the Static protocol.

CHAPTER 7: EDGESLICE: SLICING NETWORK WITH DECENTRALIZED DEEP REINFORCEMENT LEARNING

In this chapter, we design a decentralized resource orchestration system named EdgeSlice for dynamic end-to-end network slicing. EdgeSlice introduces a new decentralized deep reinforcement learning (D-DRL) method to efficiently orchestrate end-to-end resources. D-DRL is composed of a performance coordinator and multiple orchestration agents. The performance coordinator manages the resource orchestration policies in all the orchestration agents to ensure the service level agreement (SLA) of network slices. The orchestration agent learns the resource demands of network slices and orchestrates the resource allocation accordingly to optimize the performance of the slices under the constrained networking and computing resources. We design radio, transport and computing manager to enable dynamic configuration of end-to-end resources at runtime.

7.1 EdgeSlice Overview

EdgeSlice automates dynamic network slicing in wireless edge computing networks through decentralized deep reinforcement learning. Fig. 7.1 outlines the design of the EdgeSlice system. To automate the network slicing process, EdgeSlice leverages machine learning, i.e., deep reinforcement learning, to learn end-to-end resource demands of network slices and then orchestrates the resource allocations to network slices accordingly. Owing to the temporal and spatial dynamics of the slice traffic and the complex tradeoffs between the performance of network slices and the resource orchestration, it is inefficient to use a centralized learning agent to orchestrate resource allocations to network slices. Besides, a centralized learning agent needs to

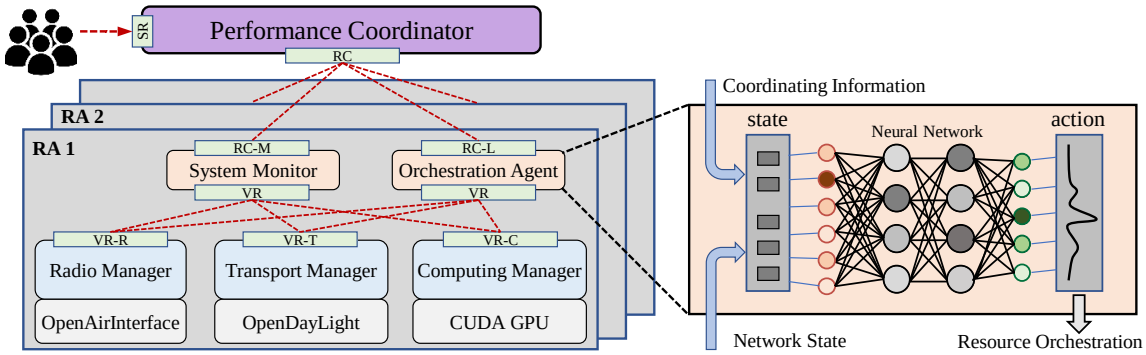

Figure 7.1: The EdgeSlice System.

obtain network performance data from all the network nodes, which introduces excessive communication overhead and delay. Toward this end, EdgeSlice introduces a new decentralized deep reinforcement learning method for network slicing in wireless edge computing networks.

We define a resource autonomy (RA) as a set of network infrastructures such as BSs and edge servers in a geographic area, and thus the network can be partitioned into multiple RAs. An orchestration agent is designed based on deep reinforcement learning to manage multi-domain resources in each RA and operates on a short timescale, e.g., seconds, to enable dynamic network slicing. The orchestration agent (detailed in Sec. 7.3.2) can track the network state (queue length, traffic), learn the resource orchestration policy from experience and orchestrate resources to slices autonomously.

A centralized performance coordinator is designed to coordinate the resource orchestration in all the RAs and optimizes the performance of the network on a much larger timescale. Meanwhile, the performance coordinator ensures that all the constraints related to the resource orchestration, e.g., SLAs and system capacity, are satisfied (detailed in Sec. 7.3.1). The performance coordinator only exchanges slight coordinating information with orchestration agents, which substantially decreases the communication overheads.

To realize EdgeSlice, resource managers, i.e., middleware, are developed to manage resources in radio access network, transport network, and edge computing servers at

Table 7.1: Notations

entity	symbol	entity	symbol
network slice	i	resource autonomy (RA)	j
network resource	k	time interval	t
slice queue length	l	time period	$\mathcal{T}$
slice performance	$\mathbf{U}$	resource orchestration	x
min. performance	$\mathbf{U}^{\min}$	total resource	R^{tot}
auxiliary variable	z	dual variable	y

runtime according to the resource orchestration decision made by orchestration agents (detailed in Sec. 7.4).

7.2 System Model and Problem Statement

To design the EdgeSlice system, we first mathematically model the wireless edge computing network and formalize the statement of end-to-end resource orchestration problem.

7.2.1 System Model

We consider an end-to-end wireless edge computing network which is composed of a radio access network (RAN) with multiple base stations (BSs), edge/cloud computing servers, and a transport network connecting the RAN and computing servers. As shown in Fig. 1.5, there are multiple network slices that request end-to-end resources in every RA, in order to enable seamless service coverage and support their users mobility. In each RA, network slices have service queues that buffer the arrival traffic of their slice users. We consider the network is time-slotted, and network operator can observe the performance[1] of network slices and dynamically change its resource orchestration with a minimum t time interval.

Let $\mathcal{I}$, $\mathcal{J}$ and $\mathcal{K}$ be the sets of network slices, RAs and network resources, respectively. Denote $\mathbf{x}_{i,j}^{(t)} = [x_{i,j,k}^{(t)} | \forall k \in \mathcal{K}]$ where $x_{i,j,k}^{(t)}$ is the kth resource allocated to the ith slice on the jth RA and $\mathbf{U}_{i,j}^{(t)}$ is the performance of network slice.

[1]Network slices could have various metrics on evaluating their performances, e.g., latency, throughput, queue status.

7.2.2 Problem Statement

The objective of network slicing is to maximize the performance of network slices in the system, and the objective of the network slicing can be expressed as

$$\max_{\{\mathbf{x}_{i,j}^{(t)}\}} \lim_{\tau\to\infty} \frac{1}{\tau}\sum_{t=0}^{\tau}\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\mathbf{U}_{i,j}^{(t)}. \tag{7.1}$$

As $\tau \to \infty$, the problem is an infinite time horizon stochastic programming problem. A general method to solve the problem is to transform it into a problem within a finite time period $\mathcal{T}$, e.g., a day [33, 90]. Hence, the resource orchestration problem is formulated as

$$\begin{aligned} \mathscr{P}_0: \quad & \max_{\{\mathbf{x}_{i,j}\geq 0\}} \quad \sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\mathbf{U}_{i,j}^{(t)} \\ & s.t. \qquad (7.3), (7.4). \end{aligned} \tag{7.2}$$

In the context of network slicing, the resource orchestration problem subjects to two practical constraints. The first constraint is that the network-wide performance of a network slice should meet the SLA made between the slice tenant and network operator. Denote $\mathbf{U}_i^{\min}$ as the minimum performance requirement of the ith slice according to the SLA. Thus, the performance constraint can be written as

$$\sum\nolimits_{t\in\mathcal{T}}\sum\nolimits_{j\in\mathcal{J}}\mathbf{U}_{i,j}^{(t)} \geq \mathbf{U}_i^{\min}, \qquad \forall i\in\mathcal{I}. \tag{7.3}$$

The second constraint is that the resources in each RA are limited. Denote $R_j^{tot} = [r_{j,k}^{tot}|\forall k\in\mathcal{K}]$ as the total amount of each resource in the jth RA. Then, the resource allocated to network slices in the jth RA should be less than R_j^{tot}, and the constraint can be expressed as

$$\sum\nolimits_{i\in\mathcal{I}}\mathbf{x}_{i,j}^{(t)} \leq R_j^{tot}, \qquad \forall j\in\mathcal{J}, t\in\mathcal{T}. \tag{7.4}$$

The difficulties in solving problem $\mathscr{P}_0$ are two-fold. First, the problem involves the

end-to-end resource orchestration to network slices within each RA and the performance coordination across all RAs to maintain network-wide performance of network slices. The coupling between the intra-RA and inter-RAs resource management highly complicates the problem. Second, due to the varying network dynamics and the diversity of resource demands of network slices, the slice performance becomes a complex stochastic function. In real systems, it is almost impossible to derive an accurate mathematical model for such correlation [35]. Moreover, the resource orchestration in the network slicing system exhibits *Markovian* on serving slice users where a resource orchestration policy affects not only the current but also further network state, e.g., service queues.

7.3 EdgeSlice Design: Coordinator and Agents

In this section, we present the design of performance coordinator and orchestration agents in the EdgeSlice system.

7.3.1 Performance Coordinator

Since the performance of a network slice depends on the resource orchestration in multiple RAs, the central performance coordinator is designed to coordinate the resource orchestration among RAs and thus optimizes the performance of the network slices. To design the performance coordinator, we transform problem $\mathscr{P}_0$ by introducing auxiliary variables $\mathcal{Z} = \{z_{i,j}, \forall i \in \mathcal{I}, j \in \mathcal{J}\}$ where

$$z_{i,j} = \sum\nolimits_{t \in \mathcal{T}} \mathbf{U}_{i,j}^{(t)}, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}. \tag{7.5}$$

Then, the constraint (7.3) are equivalent to

$$\sum\nolimits_{j \in \mathcal{J}} z_{i,j} \geq \mathbf{U}_i^{\min}, \qquad \forall i \in \mathcal{I}. \tag{7.6}$$

Hence, problem $\mathscr{P}_0$ is equivalently transformed to

$$\begin{aligned} \mathscr{P}_1: \quad & \max_{\{\mathbf{x}_{i,j}\geq 0, z_{i,j}\}} \quad \sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} \mathbf{U}_{i,j}^{(t)} \\ & s.t. \qquad\qquad (7.4), (7.5), (7.6). \end{aligned} \tag{7.7}$$

Problem $\mathscr{P}_1$ has two sets of variables, $\mathcal{X}$ and $\mathcal{Z}$ which are coupled by constraint (7.5). Next, we derive augmented Lagrangian of problem $\mathscr{P}_1$ as

$$\mathcal{L}_y = \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\left(\sum_{t\in\mathcal{T}}\mathbf{U}_{i,j}^{(t)} - \frac{\rho}{2}\|\sum_{t\in\mathcal{T}}\mathbf{U}_{i,j}^{(t)} - z_{i,j} + y_{i,j}\|_2^2\right), \tag{7.8}$$

where $\rho \geq 0$ is a positive constant, and $\mathcal{Y} = \{y_{i,j}, \forall i \in \mathcal{I}, j \in \mathcal{J}\}$ is the scaled dual variables. Here, the augmented Lagrangian incorporates the constraint (7.5) which couples variables $\mathcal{Z}$ and $\mathcal{X}$.

According to the alternating direction method of multipliers (ADMM) method [91], problem $\mathscr{P}_1$ is solved by iteratively solving the following problems:

$$\mathbf{x}_{i,j} = \arg\max_{\mathbf{x}_{i,j}\in(7.4)} \mathcal{L}_y(\mathbf{x}_{i,j}, z_{i,j}, y_{i,j}), \tag{7.9}$$

$$z_{i,j} = \arg\max_{z_{i,j}\in(7.6)} \mathcal{L}_y(\mathbf{x}_{i,j}, z_{i,j}, y_{i,j}), \tag{7.10}$$

$$y_{i,j} = y_{i,j} + (\sum\nolimits_{t\in\mathcal{T}}\mathbf{U}_{i,j}^{(t)} - z_{i,j}), \tag{7.11}$$

where problem in Eq. 7.9 focuses on the resource orchestration. Problem in Eq. 7.11 and Eq. 7.10 update auxiliary and dual variables, respectively, which require all the resource orchestrations in the system.

Therefore, we design the performance coordinator to solve the problem in Eq. 7.10 and Eq. 7.11 based on the resource orchestration and slice performance collected from orchestration agents in the system. Since $\mathcal{X}$ and $\mathcal{Z}$ are obtained, the problem

in Eq. 7.10 is equivalent to

$$\begin{aligned} \mathscr{P}_2: \quad & \min_{\{z_{i,j}\}} \quad \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} \| \sum_{t\in\mathcal{T}} \mathbf{U}_{i,j}^{(t)} - z_{i,j} + y_{i,j} \|_2^2 \\ & s.t. \quad (7.6). \end{aligned} \tag{7.12}$$

This problem is a standard quadratic programming problem which can be solved by using convex optimization tools, e.g., CVX [64]. By solving the problem, the performance coordinator obtains auxiliary variables $\mathcal{Z}$ and then updates dual variables $\mathcal{Y}$ according to Eq. 7.11. We define the auxiliary variables $\mathcal{Z}$ and the dual variables $\mathcal{Y}$ as the coordinating information between the performance coordinator and orchestration agents.

7.3.2 Orchestration Agent

The orchestration agents are designed to orchestrate the end-to-end resources for network slices under the supervision of the performance coordinator, i.e., solving the problem in Eq. 7.9. Since the constraint of the problem only restricts the resource orchestration within a RA, it can be solved individually within each RA, i.e. decentralized. Hence, we rewrite the problem in Eq. 7.9 within the jth RA as

$$\begin{aligned} \mathscr{P}_3: \quad & \max_{\{\mathbf{x}_{i,j}\geq 0\}} \quad \sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}} \mathbf{U}_{i,j}^{(t)} \\ & \qquad\qquad -\tfrac{\rho}{2}\sum_{i\in\mathcal{I}} \| \sum_{t\in\mathcal{T}} \mathbf{U}_{i,j}^{(t)} - z_{i,j} + y_{i,j} \|_2^2 \\ & s.t. \qquad (7.4). \end{aligned} \tag{7.13}$$

The major challenge of solving the above problem is that the slice performance is very complex and without a closed-form mathematical model because of the varying network dynamic and the complicated end-to-end resource demands of network slices. Moreover, the current resource orchestration impacts both slice users in service queues and further network state. To address this challenge, we resort to deep reinforcement learning (DRL) techniques that enable model-free machine learning [92]

when designing orchestration agents.

We consider a general reinforcement learning setting where an agent interacts with an environment in discrete decision epochs. At each decision epoch t, the agent observes a state $\mathbf{s}_t$, takes an action $\mathbf{a}_t$, e.g., resource orchestration, based on its policy $\mu(\mathbf{s})$, and receives a reward $r(\mathbf{s}_t, \mathbf{a}_t)$. Then, the environment transits to the next state $\mathbf{s}_{t+1}$, e.g., queue status changes, based on the action taken by the agent. The objective is to find the optimal policy $\mu^*(\mathbf{s})$ mapping states to actions, that maximizes the discounted cumulative reward $\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)$. Here, $\gamma \in [0, 1)$ is a discounting factor.

Although DRL techniques have been extensively studied in many areas such as robotic control [93], traffic control [58], and chess games [94], the existing DRL models are not appropriate to solve problem $\mathscr{P}_3$ for two reasons. First, most of the DRL models are designed to solve constraint-free problems [58, 95]. However, the problem consists of multiple linear constraints. Second, the existing DRL models are unable to adjust their policies based on coordinating information from an external control [57]. However, to maintain the network-wide performance of network slices, the agent in EdgeSlice needs to orchestrate resources according to the coordinating information derived from the coordinator.

7.3.2.1 Design of Agents

Therefore, we design a new DRL model with customized state space, action space and reward function. In the DRL model, the constraint (7.4) are re-weighted and incorporated into its reward function so that the reward is affected by whether the constraints are satisfied or not. The coordinating information is augmented into state space to allow external control from the coordinator.

State Space: The state is concatenated by two parts. The first part is $[l_j^{(t)}, \forall i \in \mathcal{I}]$ which represents the current network state, i.e., queue status of network slices. The second part is $[z_{i,j} - y_{i,j}, \forall i \in \mathcal{I}]$ which is the coordinating information from the

coordinator. Thus, the state in the jth RA at time interval t can be expressed as

$$\mathbf{s}_t = \left[l_j^{(t)}, z_{i,j} - y_{i,j}, \forall i \in \mathcal{I} \right]. \tag{7.14}$$

Action Space: The action at time interval t is defined as the resource allocations to network slices in the RA:

$$\mathbf{a}_t = \left[\mathbf{x}_{i,j}^{(t)}, \forall i \in \mathcal{I} \right]. \tag{7.15}$$

Reward: The reward at time interval t is defined as

$$\begin{aligned} r(\mathbf{s}_t, \mathbf{a}_t) = & \sum_{i \in \mathcal{I}} \left(\mathbf{U}_{i,j}^{(t)} - \frac{\rho}{2} \| \mathbf{U}_{i,j}^{(t)} - \frac{1}{\mathcal{T}} (z_{i,j} + y_{i,j}) \|_2^2 \right) \\ & - \beta \sum\nolimits_{j \in \mathcal{J}} \left[\sum\nolimits_{i \in \mathcal{I}} \mathbf{x}_{i,j}^{(t)} - R_j^{tot} \right]^+, \end{aligned} \tag{7.16}$$

where $[x]^+ = \max(0, x)$, and β is a positive constant. Here, we approximate the objective function of problem $\mathscr{P}_3$ with identical sub-objective functions in the time domain.

Moreover, we incorporate the constraints (7.4) into the sub-objective functions with reward shaping technique [96]. Therefore, there will be a penalty added into the reward if the constraints are violated.

7.3.2.2 Training of Agents

We follow deep deterministic policy gradient (DDPG), a state-of-the-art reinforcement learning technique that is capable of handling continuous and high-dimensional action spaces [92], to train our orchestration agents. As shown in Fig. 7.2, DDPG integrates deep Q-network (DQN) [93] and actor-critic method [97], and maintains a parameterized actor $\mu(\mathbf{s}_t|\theta^\mu)$ and a parameterized critic $\pi(\mathbf{s}_t, \mathbf{a}_t|\theta^\pi)$. The critic estimates the value function of state-action pairs, and the actor specifies the current policy by mapping a state to a specific action.

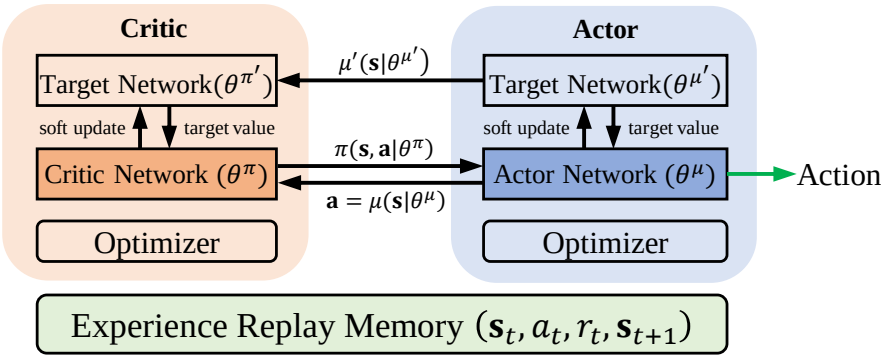

Figure 7.2: The DDPG architecture.

The critic is implemented using a DQN. We define the value function $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ as the expected discounted cumulative reward when the agent starts with the state-action pair $(\mathbf{s}_t, \mathbf{a}_t)$ at decision epoch t and then acts according to the policy π. Then, the value function can be expressed as $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\pi}[R_t]$, where $R_t = \sum_{k=t}^{T} \gamma^{(k-t)} r(\mathbf{s}_k, \mathbf{a}_k)$. Based on the Bellman equation [98], the optimal value function is $Q^*(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}_{t+1}} Q^*(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$.

To obtain the optimal policy, DQN is trained by minimizing the mean-squared Bellman error (MSBE)

$$L(\theta^{\pi}) = \mathbb{E}\left[(g_t - Q(\mathbf{s}_t, \mathbf{a}_t|\theta^{\pi}))^2\right], \tag{7.17}$$

where θ^{π} are parameters of the critic network, and $\mathcal{D}$ is a replay memory. g_t is the target value estimated by a target network, and can be expressed as

$$g_t = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}_{t+1}} Q(\mathbf{s}_{t+1}, \mu(\mathbf{s}_{t+1}|\theta^{\mu'})|\theta^{\pi'}), \tag{7.18}$$

where $\theta^{\pi'}$ are parameters of the target network. The target network has the same architecture as the critic network, and its parameters $\theta^{\pi'}$ are slowly updated to track that of the critic network.

The actor is implemented using another DQN which learns a deterministic policy

Algorithm 8: The EdgeSlice Resource Orchestration

Input: $\mathbf{U}_i^{\min}, \forall i \in \mathcal{I}$; $R_j^{tot}, \forall i \in \mathcal{I}$; ρ, β.
Output: $\mathcal{X}, \mathcal{Z}, \mathcal{Y}$.

Initialize $\mathcal{Z}$ and $\mathcal{Y}$;
while *True* **do**
  $/* *$ *optimize* $\mathcal{X}$ *in each agent* $* */$;
  for $j \in \mathcal{J}$ (*decentralized*) **do**
    $\mathbf{x}_{i,j}^{(t)}, \forall i \in \mathcal{I}, t \in \mathcal{T} \leftarrow$ the ith orchestration agent;
    $\mathbf{U}_{i,j}^{(t)}, \forall i \in \mathcal{I}, t \in \mathcal{T} \leftarrow$ the ith slice performance;
  $/* *$ *update* $\mathcal{Z}$ *in the coordinator* $* */$;
  $z_{i,j} \leftarrow \arg \max_{z_{i,j} \in (7.6)} \mathcal{L}_y(\mathbf{x}_{i,j}, z_{i,j}, y_{i,j})$;
  $/* *$ *update* $\mathcal{Y}$ *in the coordinator* $* */$;
  $y_{i,j} \leftarrow y_{i,j} + (\sum_{t \in \mathcal{T}} \mathbf{U}_{i,j}^{(t)} - z_{i,j})$;
  $/* *$ *if algorithm converges* $* */$;
  if *convergence* **then**
    return $\mathcal{X}, \mathcal{Z}, \mathcal{Y}$;

$\mu(\mathbf{s}_t|\theta^\mu)$ to maximize the cumulative reward of the actor, i.e., $J = \mathbb{E}_\mu[R_t]$. Since the action space is continuous, the value function is assumed to be differentiable with respect to the action. Thus, the actor network can be trained by applying the chain rule to the expected cumulative reward with respect to the actor parameters θ^μ:

$$\begin{aligned} \nabla_{\theta^\mu} J &\approx \mathbb{E}\left[\nabla_{\theta^\mu} Q(\mathbf{s}, \mathbf{a}|\theta^\pi)|_{\mathbf{s}=\mathbf{s}_t, \mathbf{a}=\mu(\mathbf{s}_t|\theta^\mu)}\right] \\ &= \mathbb{E}\left[\nabla_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}|\theta^\pi)|_{\mathbf{s}=\mathbf{s}_t, \mathbf{a}=\mu(\mathbf{s}_t)} \cdot \nabla_{\theta^\mu} \mu(\mathbf{s}|\theta^\mu)|_{\mathbf{s}=\mathbf{s}_t}\right]. \end{aligned} \tag{7.19}$$

7.3.3 The Workflow of EdgeSlice

The workflow of the EdgeSlice system is summarized in Alg. 8. The resource orchestration starts by initializing the coordinating information, i.e., $\mathcal{Z}$ and $\mathcal{Y}$. The orchestration agent in each RA orchestrates resources to network slices based on its parameterized policy under the coordinating information for time intervals in $\mathcal{T}$. At the end of a time period $\mathcal{T}$, the orchestration agent collects the performance of network slices $\mathbf{U}$. Given $\mathcal{X}$ and $\mathbf{U}$, the performance coordinator generates the coordinating

information ($\mathcal{Y}$ and $\mathcal{Z}$), which are fed back to orchestration agents in all RAs. It continues until the convergence of the resource orchestration.

7.4 EdgeSlice Design: Resource Manager

In this section, we design radio, transport, and computing manager that allocates the resources orchestrated by agents to network slices at runtime, as shown in Fig. 7.1. These managers are integrated with OpenAirInterface (OAI), OpenDayLight (ODL), and CUDA GPU computing platform to enable dynamic configuration of resources in radio access network, transport network, and edge/cloud computing, respectively.

7.4.1 Radio Manager

The radio manager is designed to work with OpenAirInterface (OAI) to allocate radio resources to slice users in both uplink (UL) and downlink (DL) radio access network. In EdgeSlice, the total radio resources (bandwidth) can be used by a network slice is determined by the orchestration agent. Once a network slice obtains its radio resources, it allocates these resources to its users. As a result, the allocated radio resources of all slice users are known by the radio manager. Hence, the radio manager should schedule users according to their allocated resources at runtime, which is not supported by vanilla OAI.

We fulfill such functionality by developing a new user scheduling method in the MAC layer to manage physical resource blocks (PRBs) in PUSCH/PDSCH. We schedule the slice users consecutively and map their radio resources to PRBs. The users without any radio resources will not be scheduled. To support the information exchange between the orchestration agent and the radio manager at runtime, we develop the VR-R (virtual resource - radio) and VR (virtual recourse) interfaces in the radio manager and orchestration agent, respectively. The association between a mobile user and a network slice is identified by the user's international mobile subscriber identity (IMSI). The IMSI information is extracted from the S1AP message sent from the base

station to mobile management entity (MME). The information extraction does not need any modification on the mobile user's side.

7.4.2 Transport Manager

Taking advantaging of the separation of data and control plane in SDN switches, we allocate the bandwidth of links between RAN and edge/cloud computing servers with an OpenDayLight [99] controller through OpenFlow (Southbound API) and RESTful (Northbound API) [100]. The OpenFlow protocol currently supports user bandwidth modification with *meters*. However, these meters and their attached flows should be deleted and reinitialize if the user bandwidth needs to be changed. As a result, when changing the user bandwidth allocation at runtime, the switch network is broken during the deletion-creation interval [101].

To enable dynamic modification of bandwidth while keeping the switches network alive, we create a new configuration that parallels with the current one when a new user bandwidth allocation is received from the orchestration agent. Only if the new configuration is available in switches, we release the current configuration to transition to the new one accordingly so that we can hide the deletion-creation interval. In addition, the information exchange between the transport manager and orchestration agent is support through the VR-T (virtual resource - transport) interface and the VR interface. The association of users and slices in the transport network are identified by using their source and destination IP addresses.

7.4.3 Computing Manager

The computing manager is designed to dynamically allocate computing resources, e.g., the number of CUDA *threads*, in the CUDA-based GPU computing platform. In the CUDA programming model, an application can launch multiple *kernels*, where every kernel can be concurrently executed by massive CUDA threads [79]. The number of threads required by a kernel is specified in its execution configuration syntax.

The execution of these kernels in the kernel space follows the order of their callings in the user space. With the multiple-processes service (MPS), multiple applications or processes can share the GPU simultaneously. However, the resource scheduling strategies of user applications are nontransparent and not revealed by NVIDIA. As a result, the resource usage of user applications can not be effectively controlled.

To address this issue, we develop a kernel-split mechanism to control the GPU computing resources by managing the maximum concurrent number of threads occupied by every user application. The kernel-split mechanism splits a kernel that requests a large number of threads into multiple small and consecutive kernels with a specific number of threads. We heavily modify the kernels of user applications to dynamically split the kernels according to the user's virtual resources at runtime. Since the execution of kernels are in-order and consecutive, the number of threads occupied by a user application always less than its virtual resources. We develop the VR-C (virtual resource - computing) interface in the computing manager for exchanging information with the orchestration agent. The association between a mobile user and the network slice is identified by the IP address.

7.4.4 System Monitor

The system monitor is designed to collect information of network state, e.g., traffic load and slice performance, by using a dataset. The database also records the user-slice association based on the users' IMSIs and IP addresses. The system monitor uses the VR interface to communicate with radio, transport and computing manager.

The RC (resource coordination) interface is developed to allow the central performance coordinator to communicate with orchestration agents and system monitors through the RC-L (resource coordination - learning) and RC-M (radio coordination - monitoring), respectively. The SR (slice request) interface is developed to enable the slice tenants to request and configure their network slices. For example, slice tenants can make and modify their service-level agreements (SLAs) with network operator.

Table 7.2: Details of the Prototype

Component	Hardware	Software
UEs	4x Samsung smartphones with band selection capability	Android 7.0
eNodeBs	2x Intel i5 Computer with low-latency kernel 3.19	OpenAirInterface (OAI) [77]
RF Front-End	2x Ettus USRP B210	N/A
Transport	6x OpenFlow 1.3 Ruckus switches	OpenDayLight-Boron [99]
Core Network	Intel i7 desktop computer	openair-cn [81]
Edge Servers	2x NVIDIA GEFORCE GTX 1080Ti	CUDA 9.0 [79]

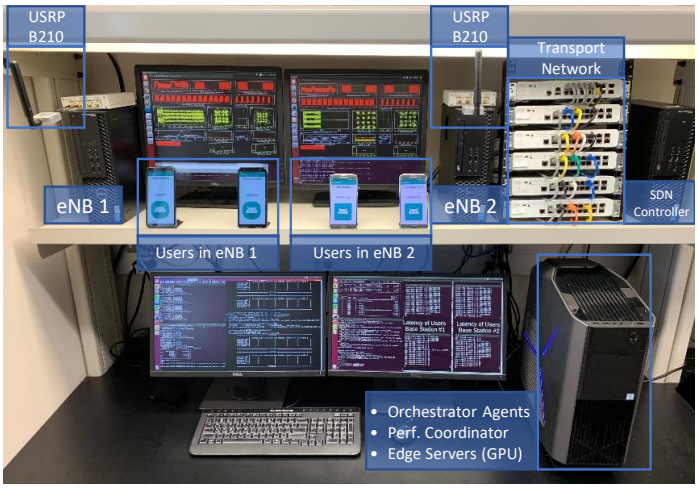

Figure 7.3: The overview of prototype.

The SLAs will be enforced during the resource orchestrations.

7.5 System Implementation

7.5.1 Hardware Details

We develop a prototype of the EdgeSlice system as depicted in Fig. 7.3. It is composed of a RAN with 2 eNodeBs, a transport network with 6 OpenFlow switches, a core network, and 2 edge servers with CUDA GPUs. The details of hardware are summarized in Table 7.2. To eliminate the co-channel interference, eNodeBs are operating at different frequency bands, i.e., LTE Band 7 and Band 38. We configure the band selection option on smartphones so that the users in eNodeB 1 and 2 can only search for band 7 and band 38, respectively.

In the prototype, there are 2 RAs, 2 slices and 4 mobile users (1 user per slice per RA), where a RA is the set of an eNodeB, an edge server and a transport link.

The orchestration agents and performance coordinator are implemented in the core network (Alienware R7 desktop) with Python 3.5. The optimization toolbox used in the performance coordinator is CVXPY 1.0 [102]. The radio manager is deployed in every eNodeB. The transport manager is deployed on an individual desktop computer. The computing manager is implemented on the edge server for every RA. Both eNodeBs are with 5MHz (25 PRBs) wireless bandwidth. The total bandwidth between an eNodeB and its corresponding edge server is 80Mbps. The total amount of the computing resource for each RA is 51200 CUDA threads.

We implement orchestration agents with Tensorflow 1.10 [103]. We use a 2-layer fully-connected neural network in both actor and critic networks. Both layers adopt Leaky Recifier [104] activation functions with 128 neurons. In the output layer, we use *sigmoid* [104] as the activation function. On training orchestration agents, we conduct extensive and empirical tunings on the hyper-parameters. We randomly generate $z_{i,j} - y_{i,j}$ between 0 and R_j^{tot} to train the agents under different coordinating information. The parameter $\beta = 20$ to have sufficient weight on enforcing the total orchestrated resources constraint (7.4). The learning rates of both actor and critic networks are 0.001. The batch size is 512. The total training step is 1E6. The discounted factor for cumulative reward is $\gamma = 0.99$. We add the decaying Gaussian noise on actions during the training phase for balancing the exploitation and exploration. The noise starts from $\mathcal{N}(0, 1)$ and decays with factor 0.9999 per update step.

7.5.2 Simulated Network Environment

The orchestration agents are trained offline by using a simulated network environment as shown in Fig. 7.4. In the environment, we implement a first-in first-out (FIFO) queue for services in individual network slices, and the performance function of each slice can be customized. In each time interval, the traffic, i.e., service tasks, in the network slices is generated according to real network traffic traces [105]. The service time of each task is determined by the end-to-end resource orchestration.

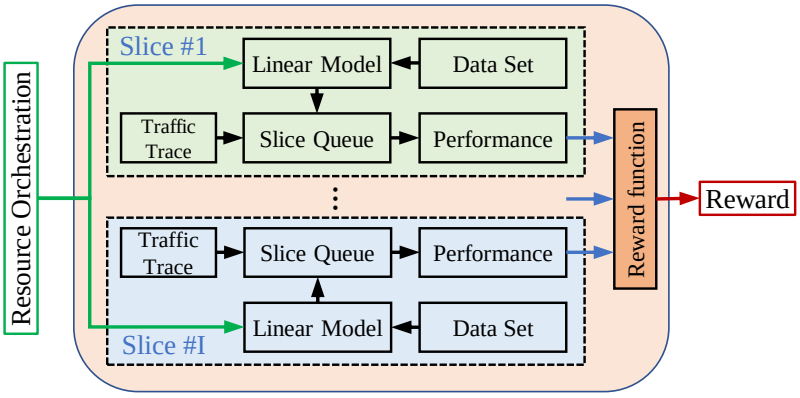

Figure 7.4: The simulated network environment.

With the simulated network environment, we generate the training dataset by traversing all possible orchestration actions using the grid search method for radio, transport and computing resources, respectively. Due to the large number of orchestration actions, we conduct the experiments with resource granularity 10% for all the resources, which means the dataset only contains discrete orchestration actions. During the training of agents, it may produce orchestration actions that are not contained in the training dataset. To solve this problem, we build a linear regression model with *scikit-learn* [106] tool to approximate the correlations between orchestration actions and the slice performance. Given a resource orchestration action such as $[12, 38, 22]\%$, we use adjacent orchestration actions in the dataset, e.g., $[10, 30, 20]\%$ and $[10, 40, 20]\%$, to fit the linear model. Once the linear model is fitted, it makes the prediction for the service time under the orchestration action. The service time determines the traffic departure in service queues. At the end of each time interval, the reward is derived based on the performances of all network slices and the design of reward function in Eq. 7.16.

7.6 Performance Evaluation

In this section, we evaluate the performance of EdgeSlice with both prototype experiments and network simulations. At each time interval, the ith slice on the jth

RA reports its performance to orchestration agent according to $\mathbf{U}_{i,j}^{(t)} = -(l_{i,j}^{(t)})^{\alpha}, \forall i \in \mathcal{I}, j \in \mathcal{J}, t \in \mathcal{T}$, where $\alpha = 2$ and $l_{i,j}^{(t)}$ is the queue length. Note that the performance function is defined to evaluate whether EdgeSlice can learn the optimal resource orchestration policy. In other words, neither the performance coordinator or orchestration agent know the closed-form expression of the performance function. Besides, various performance functions are evaluated in simulations. The performance requirements of slices are defined as $\mathbf{U}_{i}^{\min} = -50, \forall i \in \mathcal{I}$ and $\rho = 1.0$ [107].

7.6.1 Mobile Application

To evaluate the system performance, we develop a mobile application which offloads computation tasks to the edge/cloud servers. Here, the computation tasks are the video analysis based on the YOLO object detection framework [84]. The basic procedures of these applications are: 1) a user sends a video frame with a specific resolution to server and waits to receive the processed results; 2) the server receives the frame from the user and executes the YOLO algorithm with a specific computation model to analyze the frame; 3) the server sends the analysis results back to the user. The mobile application can use different frame resolutions, e.g., 100x100, 300x300 to 500x500, and select computation models, e.g., YOLO 320x320, YOLO 416x416 to YOLO 608x608. Here, the application with a higher frame resolution has heavier transmission traffic, and the application with a larger computation model requires a more intensive computation workload.

7.6.2 Comparison Algorithms

In the performance evaluation, we compare the EdgeSlice resource orchestration with the following algorithms:

Traffic-Aware Resource Orchestration (TARO): TARO is the baseline algorithm in which all the resources are proportionally shared by slices according to the current queue length. In other words, $\mathbf{x}_{i,j}^{(t)} = R_j^{tot} \cdot l_{i,j}^{(t)} / \sum_{i \in \mathcal{I}} l_{i,j}^{(t)}, \forall j \in \mathcal{J}$. This sharing

scheme applies to all the RAs in the system.

EdgeSlice-Non-Traffic (EdgeSlice-NT): EdgeSlice-NT is a simplified version of EdgeSlice in which the orchestration agent manages resources only based on the coordination information from the performance coordinator. Therefore, the state space of the orchestration agent of EdgeSlice-NT is $\mathbf{s}_t = [z_{i,j} - y_{i,j}, \forall i \in \mathcal{I}]$. In other words, EdgeSlice-NT does not use queue length of network slices as the state in the DRL model. By comparing EdgeSlice and EdgeSlice-NT, we can evaluate the impact of the state space design, i.e. whether including traffic load or not, on the performance of network slices.

7.6.3 Experimental Results

Here, we present the experimental results and evaluate the performance of the EdgeSlice system from different angles. In the experiment, there are 2 slices, 2 RAs and 3 types of resources. The mobile application in the first slice uses 500x500 frame resolution and selects YOLO 320x320 as the computation model. This application represents the type of applications that have heavy transmission traffic load and moderate computation workload. The mobile application in the second slice uses 100x100 frame resolution and selects YOLO 608x608 as the computation model. This application represents the type of applications that have light transmission traffic load and intensive computation workload.

In the experiments, the time interval t is 1 second and the time period $\mathcal{T}$ is composed of 10 time intervals. During the time intervals, the task arrival of network slices follow the Poisson process with average arrival rate[2] 10.

7.6.3.1 Convergence

In the EdgeSlice system, the performance coordinator coordinates multiple orchestration agents via the coordinating information $[z_{i,j} - y_{i,j}, \forall i \in \mathcal{I}]$. We first evaluate

[2]The slice traffic is normalized based on the hardware capability of the prototype such as bandwidth and GPU on accommodating the mobile applications.

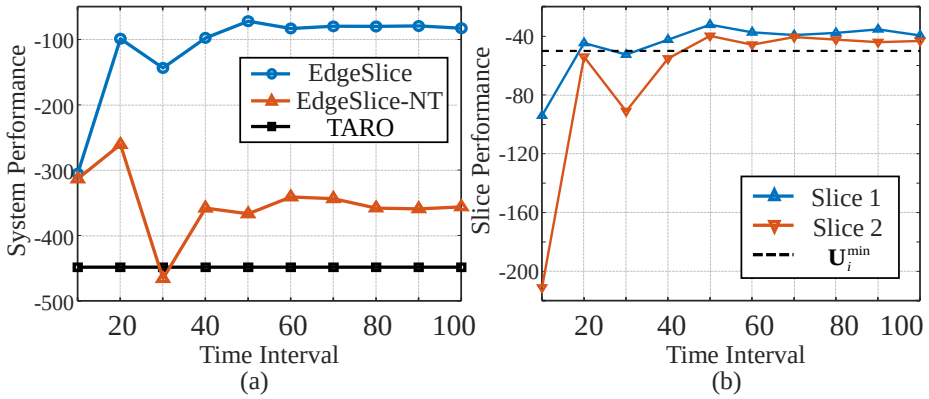

Figure 7.5: The convergence of algorithms: (a) system performance vs. time interval; (b) slice performance vs. time interval.

how fast the interaction between the coordinator and orchestration agents can converge. As depicted in Fig. 7.5 (a), both EdgeSlice and EdgeSlice-NT are able to converge after several time periods. This result also reveals that orchestration agents can effectively orchestrate resources to slices under different coordinating information. EdgeSlice obtains 3.69x and 2.74x improvement on the system performance as compared to TARO and EdgeSlice-NT, respectively. The performance gain over TARO proves that EdgeSlice can effectively learn the optimal resource orchestration policy based on the current network state and the coordinating information. The performance gain over EdgeSlice-NT indicates that observing the traffic load of slices by orchestration agents can significantly improve the system performance. In addition, as shown in Fig. 7.5 (b), the EdgeSlice system ensures that both network slices meet their minimum performance requirements.

Fig. 7.6 shows the normalized usage of multiple resources, i.e., radio, transport and computing resources, with the EdgeSlice system. In the experiments, slice 1 has a higher demand of radio and transport resources and a lower demand of computing resources than slice 2 does. Hence, we observe that EdgeSlice allocates more radio and transport resources to slice 1 (blue area). Since slice 2 serves compute-intensive applications, it requires more computing resources. Therefore, in the beginning, slice

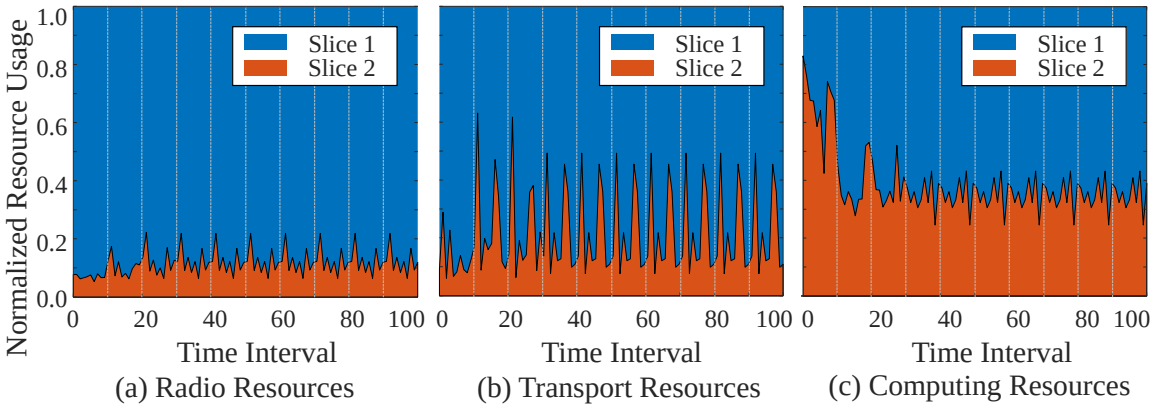

Figure 7.6: The multiple resource orchestrations of EdgeSlice: (a) radio resource; (b) transport resource; (c) computing resource.

2 is allocated more computing resources. Later, EdgeSlice observes that the performance requirement of slice 1 cannot be met although it is allocated almost all the radio and transport resources. Thus, EdgeSlice starts to allocate more computing resources to slice 1 and then the resource orchestration converges. Moreover, we observe the resources orchestrations becomes stable after 6 interactions, which corresponds to the observations in Fig. 7.5 (a).

7.6.3.2 Resource Orchestration

We evaluate the orchestration agent without any central coordination to understand its resource orchestration policy. Fig. 7.7 (a) depicts the cumulative distribution function (CDF) of the slice performance under randomly generated slice traffic loads. We can see that EdgeSlice substantially outperforms both TARO and EdgeSlice-NT in terms of the slice performance. For example, 80% of the slice performance is larger than -30 using EdgeSlice while it is only 11% and 55% using TARO and EdgeSlice-NT, respectively. The performance difference between EdgeSlice and EdgeSlice-NT is smaller than that shows in Fig. 7.5 (a). The reason is that the performance deficiency of the orchestration agent in EdgeSlice-NT accumulates during the iterative interactions between the agents and the coordinator.

Fig. 7.7 (b) and Fig. 7.8 show the average resource usage ratio between slice 1 and

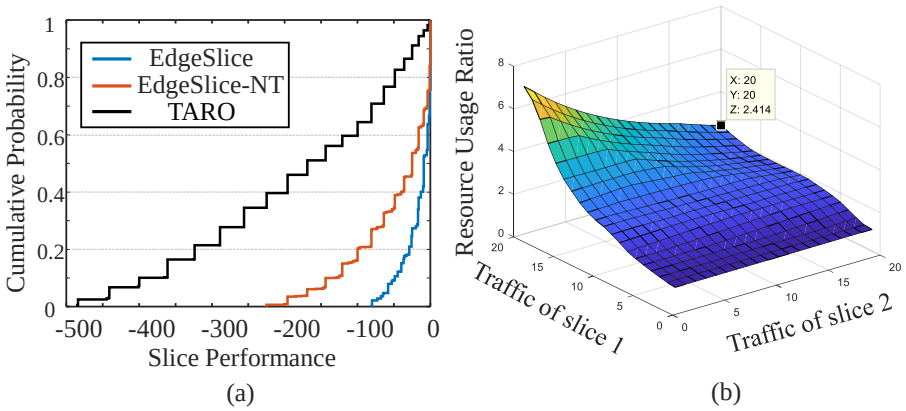

Figure 7.7: The performance of orchestration agents: (a) the CDF of performance; (b) η_1/η_2 vs. slice traffic under EdgeSlice.

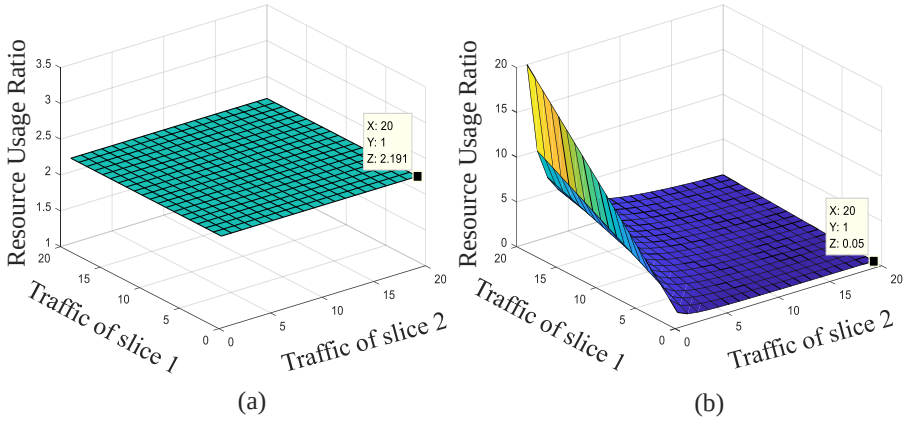

Figure 7.8: The performance of orchestration agents: (a) η_1/η_2 vs. slice traffic under EdgeSlice-NT; (b) η_1/η_2 vs. slice traffic under TARO.

slice 2 obtained by using EdgeSlice under different traffic loads. The average resource usage of a slice is calculated as $\eta_i = \sum_{k \in \mathcal{K}} x_{i,j,k}/r^{tot}_{j,k}$. It can be observed that EdgeSlice allocates resources to slices based on both traffic load and the application's resource needs in different domains. For example, when traffic loads of slice 1 and slice 2 are 20 and 5, respectively, the average resource usage ratio is about 5. This example shows the traffic-awareness of EdgeSlice. Since the orchestration agent in EdgeSlice-NT does not learn the slice traffic load in the resource orchestration, the resource usage ratio is a constant as shown in Fig. 7.8 (a). TARO allocates resources purely based on the slice traffic and is not aware of the actual resource needs from each domain. The

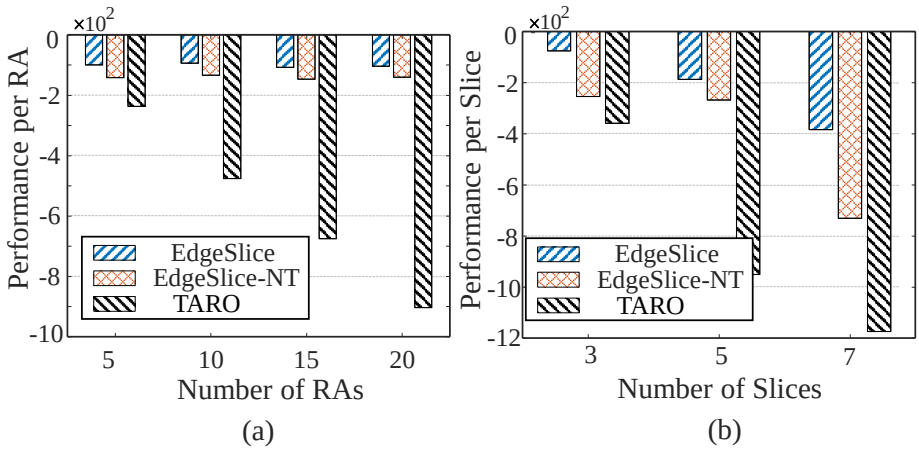

Figure 7.9: The scalability of EdgeSlice: (a) performance per RA vs. the number of RAs; (b) performance per slice vs. the number of slices.

resource usage ratio with TARO is shown in Fig. 7.8 (b). The comparison between EdgeSlice and TARO shows that EdgeSlice is aware of the multi-domain resource needs of an application. These results validate that orchestration agents of EdgeSlice are able to autonomously orchestrate end-to-end resources under varying slice traffic.

7.6.4 Simulation Results

We set up network simulations to evaluate EdgeSlice in terms of scalability and working with different training techniques and performance functions. In the simulation, there are 5 slices, 10 RAs, and 3 types of resources. The applications served by the network slices randomly select the frame resolutions, e.g., 100x100, 300x300, or 500x500, and computation models, e.g., 320x320, 416x416, 608x608. We use the network trace from an Italy mobile network over the Province of Trento [105] to generate the traffic in network slices. The network trace contains 154.8M entries with a minimum 10 minutes time interval collected in December 2013. Each entry includes the counts of phone calls, SMS, Internet traffic, and the geographic square area id. We obtain the average calling traffic in 24 hours under different geographic areas, and use them for the traffic of network slices. In the simulation, the time interval t is 1 hour and the time period $\mathcal{T}$ is composed of 24 time intervals.

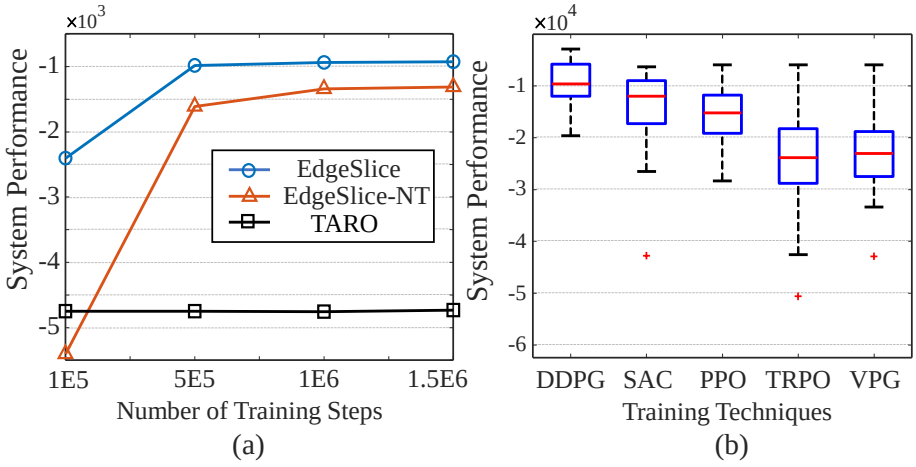

Figure 7.10: The impact of training techniques: (a) system performance vs. the number of training steps of orchestration agents; (b) system performance vs. various training techniques.

7.6.4.1 Scalability of EdgeSlice

We evaluate the scalability of EdgeSlice by varying the number of slices and RAs. As shown in Fig. 7.9 (a), both EdgeSlice and EdgeSlice-NT maintain similar performance per RA as the number of RAs increases, while the performance per RA of TARO decreases substantially. This result indicates the EdgeSlice agents learn much superior resource orchestration policy than TARO in each RA. Besides, EdgeSlice is capable of scaling to large network sizes without noticeably sacrificing system performance. Fig. 7.9 (b) shows the performance per slice versus different number of network slices. As the number of slices increases, the system performance decrease because the resource demand is higher and the average allocated resources of slices are reduced. Nevertheless, EdgeSlice is still able to obtain a better performance than the others. These results validate the scalability of the EdgeSlice system.

7.6.4.2 Training Techniques of Agents

We study the impact of various techniques on training the orchestration agents in the EdgeSlice system. As depicted in Fig. 7.10 (a), the system performance drops remarkably when the training steps of agent is insufficient such as 1E5. In general,

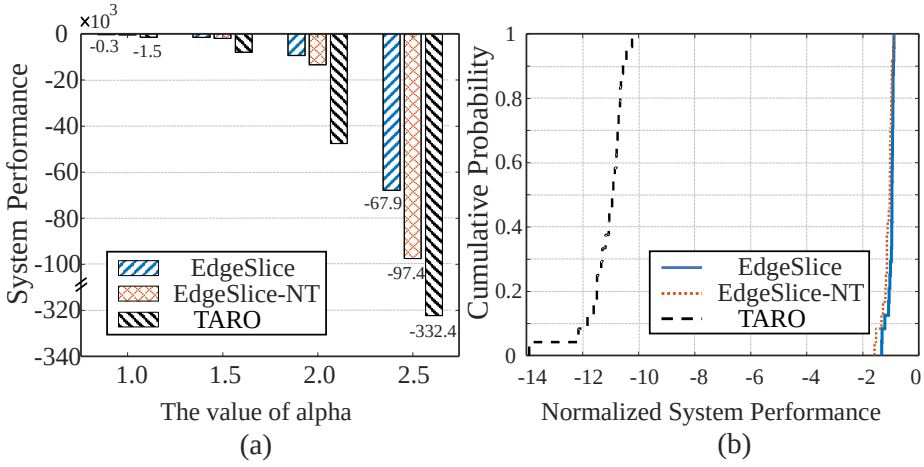

Figure 7.11: The compatibility of EdgeSlice: (a) system performance vs. the value of α, (b) CDF of normalized system performance.

a learning-based agent with a large number of training steps has better performance than that with a small number of training steps. We can see that the performance of EdgeSlice and EdgeSlice-NT can be worse than that of TARO if the number of training steps is 1E5 or less. This means that if the agent is not well trained, it could lead to very poor performance. Moreover, various techniques, e.g., SAC [108], PPO [109], TRPO [110], and VPG [111], have been proposed to improve the performance of agents. We evaluate the system performance of EdgeSlice under different training techniques as shown in Fig. 7.10 (b). The training setting and hyper-parameters are the same as mentioned in Sec. 7.5. The orchestration agent trained using DDPG exhibits better performance than that trained by the other techniques. These results show the importance of the training techniques in developing the EdgeSlice system.

7.6.4.3 Handling different performance functions

We evaluate the performance of EdgeSlice under different performance functions of network slices. As shown in Fig. 7.11 (a), we vary the value of α in the performance function. The large α indicates slice reports worse performance under the same queue length. The EdgeSlice outperforms the others for all conditions, which implies EdgeSlice can automatically learn superior resource orchestration policy under vary-

ing performance functions. Furthermore, we define another performance function as the negative service time of slice users without considering traffic in slice queue. As shown in Fig. 7.11 (b), EdgeSlice and EdgeSlice-NT achieve almost the same system performance. Because we intentionally eliminate the impact of slice traffic on the slice performance function. As a result, the network state, i.e., queue length, observed by EdgeSlice is not helpful on learning the correlations. In contrast, the performance of TARO is much worse. These results indicate that when the performance function is less dependent on the network state, learning-based EdgeSlice and EdgeSlice-NT still have much performance gain over TARO. These results verify the capability of EdgeSlice on handling various performance functions of slices.

CHAPTER 8: CONCLUSION

This research proposed an intelligent network management framework in mobile edge computing to support heterogeneous services. It deals with the challenges and difficulties with two different management approaches, i.e., context-aware service adaptation to network dynamics from the perspective of service providers and network orchestration intelligence for heterogeneous services from the perspective of infrastructure providers.

8.1 Completed Work

First, we proposed the DARE protocol to provide high quality MAR service with edge computing. The unique feature of the DARE protocol is that it can dynamically adapt the AR configurations and the computation resource allocations on the edge server according to the wireless channel conditions and computation workloads. We studied the tradeoff between the quality of augmentation and service latency in the edge-based MAR system and built analytical models to characterize such a tradeoff. We developed optimization mechanisms to guide the AR configuration and server computation resource allocation. We implemented the DARE protocol and validated that the protocol ensures the service latency of MAR clients and maximize the quality of augmentation under varying network conditions and computation workloads.

Second, we proposed an edge network orchestrator to improve the responsiveness and analytics accuracy of the edge-based MAR system. We built analytical models for studying the latency-accuracy tradeoff in edge-based MAR systems, and developed the FACT algorithm to improve the system performance by optimizing the server assignment and frame resolution selection. The performance of the FACT algorithm

was evaluated through network simulations. In addition, we implemented the edge-based MAR system with the proposed network orchestrator and the corresponding communication protocol. The performance of the edge network orchestrator and the edge-based MAR system were validated in our experiments.

Third, we proposed the *VirtualEdge* system that enables multi-domain resource orchestration and virtualization in the cellular edge computing node. *VirtualEdge* introduces a two-step resource orchestration framework that guarantees functional and performance isolation among slices with a high radio resource efficiency. In the system, we developed a multi-domain resource orchestration algorithm, a heuristic radio resource virtualization algorithm, and a credit-based queue management scheme for computing resource virtualization. The performance of the *VirtualEdge* system and corresponding algorithms were validated in both prototype implementations and network simulations.

Forth, we proposed the DIRECT protocol that realizes the cross-domain resource orchestration for cellular edge computing. As a key component of the protocol, a new distributed resource orchestration algorithm was developed by integrating the ADMM method and LAO. The proposed algorithm addresses the challenge of unknown performance model of network slices and efficiently orchestrates multi-domain resources among network slices across edge nodes. We implemented and validated the DIRECT protocol in a small-scale system prototype based on the OpenAirInterface LTE and CUDA GPU computing platforms. We also evaluated the DIRECT protocol in network simulations.

In the end, we proposed EdgeSlice, a new decentralized resource orchestration system, to automate dynamic network slicing in wireless edge computing networks. To realize EdgeSlice, we developed a novel decentralized deep reinforcement learning method which consists of a central performance coordinator and multiple orchestration agents. The orchestration agent learns the optimal resource orchestration policy

for network slicing under the coordination of the central performance coordinator. We also designed new radio, transport and computing resource manager that enables dynamic configuration of end-to-end resources at runtime. We developed a prototype of EdgeSlice with OpenAirInterface (OAI) in radio access network, OpenDayLight (ODL) in transport network, and CUDA GPU computing in edge/cloud servers. The performance of EdgeSlice was validated through both prototype implementation and network simulations.

8.2 Future Work

My Ph.D. research shed light on both theoretical and practical research possibilities in supporting heterogeneous service and use cases, in network virtualization and management, and in other related areas towards the next-generation mobile network. I would like to direct my future efforts to the following research topics.

8.2.1 Network slicing in end-to-end networks

Network slicing allows cost-efficient accommodation for diverse use cases and services in the next-generation mobile network. The network slicing is far from consolidation and freezing, where many unresolved problems and issues need to be answered. For example, most existing network slicing systems only provide the performance isolation among network slices, and the functional isolation of slices, which allows slice tenant to control its slice individually, cannot be accomplished yet. Moreover, these systems only enable a very limited number of configurations for network slices, e.g., bandwidth allocation and server assignment, which are constrained by existing network virtualization techniques. A low-cost and fast network virtualization technique could substantially accelerate the design and deployment of end-to-end network slicing systems in the next-generation mobile network.

8.2.2 Machine learning in wireless systems

Machine learning techniques have shown their great potential in managing large-scale network systems, e.g., data center networks and radio access networks, by leveraging the advanced neural network architecture. However, directly apply ML techniques in network management could lead to significant performance degradation due to the massive training steps of neural network policies and unpredictable management actions. The stability and robustness of ML techniques must be extensively studied before it can be deployed in the real network systems. For example, a well-designed safety mechanism, which organically integrates robust model-based approaches and model-free ML solutions, is preferred to exploit the ML techniques while maintaining a low-risk of SLA violation of customers.

8.3 Published and Submitted Work

The following list is a summary of my publications and submissions.

23. Q. Liu, T. Han, L. Xie, B. Kim, "LiveMap: Real-Time Dynamic Map in Automotive Edge Computing", *submitted to* IEEE International Conference on Computer Communications (INFOCOM), Vancouver, Canada, May 2021

22. Q. Liu, T. Han, N. Choi, "SafeSlicing: Exploring Optimal Network Slicing without Violating Service Level Agreement", *submitted to* IEEE International Conference on Computer Communications (INFOCOM), Vancouver, Canada, May 2021

21. Q. Liu, T. Han, E. Moges, "EdgeSlice: Slicing Wireless Edge Computing Network with Decentralized Deep Reinforcement Learning", in IEEE International Conference on Distributed Computing Systems (ICDCS), Singapore, Dec. 2020

20. Q. Liu, T. Han, N. Zhang, Y. Wang, "DeepSlicing: Deep Reinforcement Learning Assisted Resource Allocation for Network Slicing", in IEEE Globe Commu-

nications Conference (GLOBECOM), Taipei, Taiwan, Dec. 2020

19. Q. Liu, T. Han, "DIRECT: Distributed Cross-Domain Resource Orchestration in Cellular Edge Computing", in ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC), Catania, Italy, Jul. 2019

18. Q. Liu, T. Han, "VirtualEdge: Multi-Domain Resource Orchestration and Virtualization in Cellular Edge Computing", in IEEE International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, Jul. 2019

17. Q. Liu, T. Han, "DARE: Dynamic Adaptive Mobile Augmented Reality with Edge Computing", in IEEE International Conference on Network Protocols (ICNP), Cambridge, UK, Sep. 2018

16. Q. Liu, S. Huang, J. Opadere, T. Han, "An Edge Network Orchestrator for Mobile Augmented Reality", in IEEE International Conference on Computer Communications (INFOCOM), Honolulu, HI, Apr. 2018

15. J. Opadere, Q. Liu, N. Zhang, T. Han, "Joint Computation and Communication Resource Allocation for Energy-Efficient Mobile Edge Networks", in IEEE International Conference on Communications (ICC), Shanghai, China, May 2019 (Best Paper Award)

14. Q. Liu, T. Han, "Energy-Efficient On-demand Cloud Radio Access Networks Virtualization", in IEEE Globe Communications Conference (GLOBECOM), Abu Dhabi, UAE, Dec. 2018

13. Q. Liu, T. Han, N. Ansari, "Joint Radio and Computation Resource Management for Low Latency Mobile Edge Computing", in IEEE Globe Communications Conference (GLOBECOM), Abu Dhabi, UAE, Dec. 2018

12. J. Opadere, Q. Liu, T. Han, "Energy-Efficient RRH Sleep Mode for Virtual Radio Access Networks", in IEEE Globe Communications Conference (GLOBECOM), Singapore, Dec. 2017

11. S. Huang, Q. Liu, T. Han, N. Ansari, "Data-Driven Network Optimization in Ultra-Dense Radio Access Networks", in IEEE Globe Communications Conference (GLOBECOM), Singapore, Dec. 2017

10. Q. Liu, G. Wu, Y. Guo, Y. Zhang, S. Hu, "Energy Efficient Resource Allocation for Control Data Separated Heterogeneous-CRAN", in IEEE Globe Communications Conference (GLOBECOM), Washington DC, Dec. 2016

9. Q. Liu, T. Han, G. Wu, "Computing Resource Aware Energy Saving Scheme for Cloud Radio Access Networks", in IEEE Sustainable Computing and Communications (SustainCom), Atlanta, GA, Oct. 2016

8. Q. Liu, T. Han, N. Ansari, "Learning-Assisted Secure End-to-End Network Slicing for Cyber-Physical Systems", in IEEE Network Magazine, vol. 34, no. 3, pp. 37-43, May 2020

7. J. Opadere, Q. Liu, T. Han, N. Ansari, "Energy-efficient Virtual Radio Access Networks for Multi-Operators Cooperative Cellular Networks", in IEEE Transactions on Green Communications and Networking (TGCN), vol. 3, no. 3, pp. 603-614, Sep. 2019

6. Q. Liu, T. Han, N. Ansari, "Energy-Efficient On-demand Resource Provisioning in Cloud Radio Access Networks", in IEEE Transactions on Green Communications and Networking (TGCN), vol. 3, no. 4, pp. 1142-1151, Jul. 2019

5. Q. Liu, T. Han, N. Ansari, G. Wu, "On Designing Energy-Efficient Heterogeneous Cloud Radio Access Networks", in IEEE Transactions on Green Communications and Networking (TGCN), vol. 2, no. 3, pp. 721-734, May 2018

4. Q. Liu, T. Han, "When Network Slicing meets Deep Reinforcement Learning", in ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT) Student Workshop, Orlando, FL, Dec. 2019

3. Q. Liu, T. Han, "Demo Abstract: Themis: Cross-Domain Resource Orchestration and Virtualization in Cellular Computing Networks", in IEEE International Conference on Network Protocols (ICNP), Cambridge, UK, Sep. 2018

2. Q. Liu, S. Huang, T. Han, "Demo Abstract: Fast and Accurate Object Analysis at the Edge for Mobile Augmented Reality", in ACM/IEEE Symposium on Edge Computing (SEC), San Jose, CA, Oct. 2017

1. Q. Liu, S. Huang, Y. Deng, T. Han, "Demo Abstract: MExR: Mobile Edge Resource Management for Mixed Reality Applications", in IEEE International Conference on Computer Communications (INFOCOM), Atlanta, GA, Apr. 2017

REFERENCES

[1] Ericsson, "Ericsson mobility report," *White paper*, 2018.

[2] Cisco, "Cisco visual networking index: Forecast and trends," *White paper*, 2017.

[3] M. Agiwal, A. Roy, and N. Saxena, "Next generation 5g wireless networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 1617–1655, 2016.

[4] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, "5g: A tutorial overview of standards, trials, challenges, deployment, and practice," *IEEE journal on selected areas in communications*, vol. 35, no. 6, pp. 1201–1221, 2017.

[5] D. A. Chekired, M. A. Togou, L. Khoukhi, and A. Ksentini, "5g-slicing-enabled scalable sdn core network: Toward an ultra-low latency of autonomous driving service," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1769–1782, 2019.

[6] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[7] T. Braud, F. H. Bijarbooneh, D. Chatzopoulos, and P. Hui, "Future networking challenges: The case of mobile augmented reality," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1796–1807.

[8] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[9] 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.

[10] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing a key technology towards 5G," *ETSI*, vol. 11, no. 11, pp. 1–16, 2015.

[11] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *The 35th Annual IEEE International Conference on Computer Communications, IEEE INFOCOM 2016*, San Francisco, CA, USA, April 2016, pp. 1–9.

[12] InterDigital, "Realism in Augmented Reality." [Online]. Available: http://www.interdigital.com/white_papers/realism-in-augmented-reality

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[14] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[15] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[17] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile GPU-based deep learning framework for continuous vision applications," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '17, Niagara Falls, New York, USA, 2017, pp. 82–95.

[18] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun 2010.

[19] R. Hou, C. Chen, and M. Shah, "Tube Convolutional Neural Network (T-CNN) for Action Detection in Videos," *arXiv preprint arXiv:1703.10664*, 2017.

[20] P. Jain, J. Manweiler, and R. Roy Choudhury, "OverLay: Practical Mobile Augmented Reality," in *ACM, MobiSys 2015*, Florence, Italy, May 2015, pp. 331–344.

[21] Z. Huang, W. Li, P. Hui, and C. Peylo, "Cloudridar: A cloud-based architecture for mobile augmented reality," in *Proceedings of the 2014 workshop on Mobile augmented reality and robotic technology-based systems*. ACM, 2014, pp. 29–34.

[22] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices," in *ACM Sensys 2015*, Seoul, South Korea, Nov. 2015, pp. 155–168.

[23] P. Jain, J. Manweiler, and R. Roy Choudhury, "Low Bandwidth Offload for Mobile AR," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, Irvine, California, USA, Dec. 2016, pp. 237–251.

[24] R. Shea, A. Sun, S. Fu, and J. Liu, "Towards Fully Offloaded Cloud-based AR: Design, Implementation and Experience," in *ACM MMSys 2017*, Taipei, Taiwan, Jun. 2017, pp. 321–330.

[25] B. Cici, M. Gjoka, A. Markopoulou, and C. T. Butts, "On the decomposition of cell phone activity patterns and their connection with urban ecology," in *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Hangzhou, China, Jun. 2015, pp. 317–326.

[26] A. Ksentini and N. Nikaein, "Toward enforcing network slicing on RAN: Flexibility and resources abstraction," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 102–108, 2017.

[27] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega *et al.*, "Network slicing to enable scalability and flexibility in 5g mobile networks," *IEEE Communications magazine*, vol. 55, no. 5, pp. 72–79, 2017.

[28] K. Katsalis, N. Nikaein, E. Schiller, A. Ksentini, and T. Braun, "Network slices toward 5G communications: Slicing the lte network," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 146–154, 2017.

[29] X. Foukas, M. K. Marina, and K. Kontovasilis, "Orion: RAN slicing for a flexible and cost-effective multi-service mobile network architecture," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. ACM, 2017, pp. 127–140.

[30] H. Zhang, N. Liu, X. Chu, K. Long, A.-H. Aghvami, and V. C. Leung, "Network slicing based 5G and future mobile networks: mobility, resource management, and challenges," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 138–145, 2017.

[31] Global Mobile Suppliers Association, "5G network slicing for vertical industries," http://www.huawei.com/minisite/5g/img/5g-network-slicing-for-vertical-industries-en.pdf, 2017.

[32] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "How should I slice my network?: A multi-service empirical evaluation of resource sharing efficiency," in *MobiCom*. ACM, 2018, pp. 191–206.

[33] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, and X. Costa-Perez, "Overbooking network slices through yield-driven end-to-end orchestration," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. ACM, 2018, pp. 353–365.

[34] S. D'Oro, L. Bonati *et al.*, "Sl-EDGE: Network slicing at the edge," *arXiv preprint arXiv:2005.00886*, 2020.

[35] H. Mao, M. Schwarzkopf *et al.*, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*. ACM, 2019, pp. 270–288.

[36] D. Chatzopoulos, C. Bermejo, Z. Huang, A. Butabayeva, R. Zheng, M. Golkarifard, and P. Hui, "Hyperion: A wearable augmented reality system for text extraction and manipulation in the air," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, Taipei, Taiwan, Jun. 2017, pp. 284–295.

[37] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 756–764.

[38] K. Lee, D. Chu, E. Cuervo, J. Kopf, Y. Degtyarev, S. Grizan, A. Wolman, and J. Flinn, "Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, Florence, Italy, 2015, pp. 151–165.

[39] E. Cuervo, A. Wolman, L. P. Cox, K. Lebeck, A. Razeen, S. Saroiu, and M. Musuvathi, "Kahawai: High-Quality Mobile Gaming Using GPU Offload," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '15, Florence, Italy, 2015, pp. 121–135.

[40] Q. Liu and T. Han, "Dare: Dynamic adaptive mobile augmented reality with edge computing," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 2018, pp. 1–11.

[41] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance." in *NSDI*, 2017, pp. 377–392.

[42] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 15.

[43] M. Jia, W. Liang, Z. Xu, and M. Huang, "Cloudlet load balancing in wireless metropolitan area networks," in *The 35th Annual IEEE International Conference on Computer Communications, IEEE INFOCOM 2016*, San Francisco, CA, USA, April 2016, pp. 1–9.

[44] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *The 36th Annual IEEE International Conference on Computer Communications, IEEE INFOCOM 2017*, Atlanta, GA, USA, May 2017, pp. 1–9.

[45] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan, "Cellslice: Cellular wireless resource slicing for active ran sharing," in *2013 Fifth International Conference on Communication Systems and Networks (COMSNETS)*. IEEE, 2013, pp. 1–10.

[46] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "FlexRAN: A flexible and programmable platform for software-defined radio access networks," in *ACM CoNEXT 2016*, 2016, pp. 427–441.

[47] Q. Liu and T. Han, "Direct: Distributed cross-domain resource orchestration in cellular edge computing," in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2019, pp. 181–190.

[48] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan, "NVS: A substrate for virtualizing wireless resources in cellular networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 5, pp. 1333–1346, 2012.

[49] H. Halabian, "Distributed resource allocation optimization in 5G virtualized networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 627–642, 2019.

[50] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.

[51] B. Han, V. Sciancalepore *et al.*, "A utility-driven multi-queue admission control solution for network slicing," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 55–63.

[52] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *Nsdi*, vol. 11, no. 2011, 2011, pp. 24–24.

[53] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "HUG: Multi-resource fairness for correlated and elastic demands." in *NSDI 2016*, 2016, pp. 407–424.

[54] Y. Liu, M. J. Lee, and Y. Zheng, "Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system," *IEEE Transactions on Mobile Computing*, vol. 15, no. 10, pp. 2398–2410, 2016.

[55] P. Caballero, A. Banchs, G. De Veciana, and X. Costa-Pérez, "Network slicing games: Enabling customization in multi-tenant mobile networks," *IEEE/ACM Transactions on Networking*, 2019.

[56] V. Sciancalepore, M. Di Renzo, and X. Costa-Perez, "STORNS: Stochastic radio access network slicing," *arXiv preprint arXiv:1901.05336*, 2019.

[57] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016, pp. 50–56.

[58] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1871–1879.

[59] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Deepcog: Cognitive network management in sliced 5G networks with deep learning," 2019.

[60] Z. Yang *et al.*, "MIRAS: Model-based reinforcement learning for microservice resource allocation over scientific workflows," in *IEEE ICDCS*. IEEE, 2019, pp. 122–132.

[61] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[62] M. R. Bussieck and A. Pruessner, "Mixed-Integer Nonlinear Programming," *SIAG/OPT Newsletter: Views & News*, vol. 14, no. 1, pp. 19–22, 2003.

[63] S. Bonettini, "Inexact block coordinate descent methods with application to non-negative matrix factorization," *IMA journal of numerical analysis*, vol. 31, no. 4, pp. 1431–1452, 2011.

[64] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[65] Y. Suzuki, S. Kato, H. Yamada, and K. Kono, "GPUvm: Why not virtualizing GPUs at the hypervisor?" in *USENIX Annual Technical Conference*, 2014, pp. 109–120.

[66] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[67] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *ECCV*, 2016.

[68] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," *arXiv preprint arXiv:1611.10012*, 2016.

[69] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001, vol. 16.

[70] R. T. Marler and J. S. Arora, "The weighted sum method for multi-objective optimization: new insights," *Structural and multidisciplinary optimization*, vol. 41, no. 6, pp. 853–862, 2010.

[71] L. Grippo and M. Sciandrone, "On the convergence of the block nonlinear gauss–seidel method under convex constraints," *Operations research letters*, vol. 26, no. 3, pp. 127–136, 2000.

[72] A. Beck and L. Tetruashvili, "On the convergence of block coordinate descent type methods," *SIAM journal on Optimization*, vol. 23, no. 4, pp. 2037–2060, 2013.

[73] Team Mininet, "Mininet-An Instant Virtual Network on your Laptop (or other PC)," 2014.

[74] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[75] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced lectures on machine learning*. Springer, 2004, pp. 63–71.

[76] C. Audet and W. Hare, *Derivative-free and blackbox optimization*. Springer, 2017.

[77] OpenAirInterface Software Alliance. OpenAirInterface repository. https:gitlab.eurecom.fr/oai/openairinterface5g, 2017.

[78] S.-B. Lee, I. Pefkianakis, A. Meyerson, S. Xu, and S. Lu, "Proportional fair frequency-domain packet scheduling for 3GPP LTE uplink," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 2611–2615.

[79] C. Nvidia, "Nvidia CUDA C programming guide," *Nvidia Corporation*, vol. 120, no. 18, p. 8, 2011.

[80] "Universal software radio peripheral (USRP) B210," https://www.ettus.com/product/details/UB210-KIT.

[81] OpenAirInterface Software Alliance. Openair-cn repository. https:gitlab.eurecom.fr/oai/openair-cn, 2017.

[82] "Huawei LTE Dongle E3372," https://consumer.huawei.com/en/mobile-broadband/e3372/.

[83] M. Harris, "An even easier introduction to CUDA," https://devblogs.nvidia.com/even-easier-introduction-cuda/, 2017.

[84] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[85] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[86] H. Attouch, J. Bolte, and B. F. Svaiter, "Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized Gauss–Seidel methods," *Mathematical Programming*, vol. 137, no. 1-2, pp. 91–129, 2013.

[87] Y. Wang, W. Yin, and J. Zeng, "Global convergence of ADMM in nonconvex nonsmooth optimization," *Journal of Scientific Computing*, pp. 1–35, 2015.

[88] A. I. F. Vaz and L. N. Vicente, "PSwarm: a hybrid solver for linearly constrained global derivative-free optimization," *Optimization Methods & Software*, vol. 24, no. 4-5, pp. 669–685, 2009.

[89] K. Holmström, "The TOMLAB optimization environment in Matlab," 1999.

[90] P. Kall *et al.*, *Stochastic programming*. Springer, 1994.

[91] S. Boyd, N. Parikh *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[92] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[93] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[94] D. Silver, T. Hubert *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[95] L. Chen, J. Lingys *et al.*, "AuTO: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *SIGCOMM 2018*. ACM, 2018, pp. 191–205.

[96] S. Griffith *et al.*, "Policy shaping: Integrating human feedback with reinforcement learning," in *Advances in neural information processing systems*, 2013, pp. 2625–2633.

[97] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[98] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[99] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven SDN controller architecture," in *IEEE WoWMoM 2014*. IEEE, 2014, pp. 1–6.

[100] N. McKeown, T. Anderson *et al.*, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[101] O. S. Specification, "Openflow switch specification version 1.5.1, https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf," 2013.

[102] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[103] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[104] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[105] T. Italia, "Telecommunication activity dataset," https://dandelion.eu/datagems/SpazioDati/telecom-sms-call-internet-tn/description/, 2013.

[106] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[107] M. Hong and Z.-Q. Luo, "On the linear convergence of the alternating direction method of multipliers," *Mathematical Programming*, vol. 162, no. 1-2, pp. 165–199, 2017.

[108] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[109] J. Schulman, F. Wolski *et al.*, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[110] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[111] R. S. Sutton, D. A. McAllester *et al.*, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.