UNIVERSITY OF CALGARY

Bandit-based Delay-Aware Service Function Chain Orchestration at the Edge

by

Lei Wang

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

SEPTEMBER, 2020

Abstract

In Mobile Edge Computing (MEC), the network's edge is equipped with computing and storage resources in order to reduce latency by minimizing communication with remote clouds, thereby provide mobile users with both intensive computing resources and proximity to the data sources. However, the available computing capacity at the edge is limited compared to that of a remote cloud. A promising solution for efficient utilization of edge capacity is the fine-grained management via Network Function Virtualization (NFV). In this approach, users express their service demands as a Service Function Chain (SFC), which are composed of virtual network functions. Such service composition allows constituent VNFs to be flexibly deployed at the edge or in the cloud such that the service latency is minimized.

In order to provide the user with a tolerated perceived delay for an SFC-based application, many works have taken aim at optimal placement for SFC in heterogeneous scenarios. Most of the existing work on system-wide placement with the assumption that they have the knowledge of information such as future demands and the whereabouts of the users. The increasing number of users, however, challenges the scalability of system-managed SFC orchestration. To address this problem, we formulate the user-managed SFC placement in MEC as a contextual combinatorial multi-arm bandit (C^2MAB) problem and proposed CHANGE, a bandit based algorithm for online SFC orchestration on edge, which considers user's mobility and service preference while jointly optimizing their perceived latency and service migration delay, and then propose an exact offline approach for the role of performance benchmark using standard optimization solver. To fit the SFC placement problem in a bandit framework, we model the nodes and links to be arms by viewing them as delays

and define them as a set of feature vectors. To balance the exploration and exploitation of arms, we adopt the Upper Confidence Bound (UCB) theory in arm selection to efficiently explore and estimate the arms and use Dynamic Programming (DP) algorithm to exploit the collected arm estimations and compute delay-optimized SFC placements. Then we design and implement a delay estimation framework as an essential component of CHANGE.

At last, we evaluate the proposed algorithm in extensive model-driven simulation and realistic Mininet-WiFi emulation experiments. In the simulation, we study the behavior of the proposed algorithm under varying simulating parameters and problem scales and compare our algorithm with two other online approaches. Numeric simulation results show that the proposed algorithm can achieve close-to-optimum performance and can improve latency performance by nearly 20 percent compared to other approaches. In emulation, we further validate the superior performance of CMAB in realistic Mininet-wifi experiments under different environmental parameters.

Table of Contents

Abstract ii
Table of Contents iv
List of Tables vi
List of Figures vii
List of Symbols viii
1 **introduction** 1
1.1 Motivation 1
1.2 Thesis Objective 6
1.3 Thesis Contribution 7
1.4 Thesis Organization. 9
2 **Backround** 11
2.1 Network Function Virtualization 11
2.1.1 NFV architecture 11
2.1.2 NFV optimization 13
2.2 Mobile Edge Computing 15
2.2.1 MEC architecture 15
2.2.2 MEC research topics 17
2.3 Service Function Chain 19
2.3.1 SFC architecture 19
2.3.2 SFC request model 20
2.3.3 SFC orchestration problem 21
2.3.4 SFC orchestration optimization approaches 22
2.4 Learning algorithm 22
2.4.1 Online learning 23
2.4.2 Bandit online learning 23
2.5 Mathematical Tools 27
2.5.1 Integer Linear problem 27
2.5.2 Dynamic program 28
2.6 Software Tools 31
2.6.1 Networkx 31
2.6.2 Gurobi 32
2.6.3 Mininet-wifi 33
3 **Related Work** 35
3.1 System-managed Edge-agnostic 35
3.2 System-managed Edge-enabled 36
3.2.1 Delay-related objective 36
3.2.2 Resource-related objective 37
3.2.3 Models 38
3.2.4 Heuristic approaches 38
3.3 User-managed Edge-agnostic & Edge-enabled 39
4 **Problem Formulation** 43
4.1 System Model 43

4.1.1 Network Model 44
4.1.2 Service Model 45
4.2 Problem Formulation 47
4.3 Complete Problem Formulation 49
5 **Proposed Method** 51
5.0.1 Bandit Formulation 51
5.0.2 Placement Algorithm 56
5.0.3 Bandit Formulation Example 62
5.0.4 Delay Estimation and Deployment Considerations 63
6 **Delay Estimation framework** 65
6.1 Time stamp in SFC placement 65
6.2 Time-stamped delay estimation example 66
7 **Simulation Results** 69
7.1 Simulation Settings 69
7.1.1 Network Settings 69
7.1.2 SFC Settings 70
7.1.3 User mobility Settings 72
7.1.4 Performance Benchmark. 73
7.1.5 Impact of network delays 73
7.1.6 Impact of Migration Delay 74
7.2 Convergence performance 76
7.2.1 Comparison with an exact offline approach 76
7.2.2 Time average regret 76
7.2.3 Exploration ratio 77
7.2.4 Scalability Analysis 79
7.2.5 Running Time Analysis 79
7.2.6 Comparison with the greedy approaches. 79
8 **Mininet-WiFi Experiments** 82
8.1 Experiment setup. 82
8.2 CHANGE performance in Mininet-WiFi. 84
8.2.1 Response Time 85
8.2.2 User Mobility analysis 86
8.2.3 Different network delays 89
8.2.4 Different length of SFC. 89
9 **Conclusion** 91
9.1 Thesis Summary 92
9.2 Future work 94

List of Tables

3.1 Related work overview 42

4.1 List of main notations in UECOP formulation 46

7.1 Simulation parameters 70
7.2 Requirements for common SFCs 71

8.1 Network and VNF setting 85

List of Figures and Illustrations

1.1 Global device and connection growth (billion) [25] 2
1.2 Example 1: visual cloud computing SFC for 3D light detection and ranging [27] 4
1.3 Example 2: geo-distributed latency-sensitive SFC used for the real-time object tracking pipeline [11] 4

2.1 An illustrative example of NFV paradigm [84] 12
2.2 VNF Chain Embedding [33] 14
2.3 Architecture of the MEC system [51] 16
2.4 A typical SFC architecture [52] 20
2.5 Service Request Model [77] 21

4.1 An example of SFC placement in MEC 44

5.1 An illustrative example used for CMAB formulation 61

6.1 Estimation packet format 66
6.2 Time stamps of SFC placement in MEC 68

7.1 Network topology used in simulation 71
7.2 Random Waypoint mobility model 72
7.3 Effect of network delays 74
7.4 Multiple algorithms under different weights of switching cost 75
7.5 Convergence performance: 1) CHANGE approaching to the offline optimum 2) Algorithm converges around 2000 77
7.6 Total regret with different exploration level 78
7.7 Run time analysis 80
7.8 Performance analysis under different problem scales: a) Number of services on SFC; b) Size of the MEC network 81

8.1 Mininet-WiFi Emulation 83
8.2 CHANGE outperform ϵ-greedy in Mininet-WiFi 86
8.3 Effect of mobility models 88
8.4 Effect of Mininet-WiFi network delays: a) link delay; b) processing delay . . 89
8.5 Effect of the length of SFC 90

List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
LIDAR	3D Light Detection and Ranging
UECOP	User-managed Edge-enabled Chain Orchestration Problem
MEC	Mobile Edge Computing
NFV	Network Function Virtualization
VNF	Virtual Network Function
SFC	Service Function Chain
DP	Dynamic Programming
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
UCB	Upper Confidence Bound
MAB	Multi-arm Bandit
CMAB	Combinatorial Multi-arm Bandit
CCMAB	Combinatorial Contextual Multi-arm Bandit
CCUCB	Contextual Combinatorial Upper Confidence Bandit
ILP	Integer Linear Program
BILP	Binary Integer Linear Program
MILP	Mixed Integer Linear Program
MIQCP	Mixed Integer Quadratically Constrained Program
WGMAP	Weighted Graph Matching Problem
MCCF	Multi Commodity-Chain Flow problem

AP	Access Point
TCP	Transmission Control Protocol
SC	Serving on the cloud
SE	Serving on the edge
QoS	Quality of Service
NAT	Network Address Translator
FW	Firewall
TM	Traffic Monitor
WOC	WAN Optimization Controller
IDPS	Intrusion Detection Prevention System
VOC	Video Optimization Controller
RWP	Random Way Point
RW	Random Walk
RD	Random Direction
TVC	Time Variant Community
GM	Gauss Markov
RP	Reference Point

Chapter 1

introduction

1.1 Motivation

Driven by the rapidly evolving modern communication technologies such as 5G and beyond [4], the next generation of mobile applications such as virtual and augmented reality [63], face recognition [69] and 3D interactive gaming [12], have become more and more prevalent and approachable in our daily life. The increasing development of these mobile applications result from recent growth in mobile device usage. According to the recent Cisco Annual Internet Report [25], the total number of global devices and connections are predicted to increase by 11 million, with a 2 million increase in mobile devices (Smartphones, Non-smartphones, Tablets, etc.). Figure 1.1 depicts the trend of the growth in global device usage. However, mobile devices often suffer from resource limits such as storage, computing capacity, battery lifetime and therefore cannot provide the aforementioned applications with the guaranteed low latency that they require.

In order to address the computational limit of the local mobile devices, Mobile Edge Computing (MEC) [36] has recently been introduced as a promising architecture that has the potential to enable cellular networks to offer low latency to mobile applications. MEC equips cellular base stations (i.e., the edge) with computing and storage resources. Such an architecture allows mobile users to work with services deployed in their vicinity and avoid frequent communication with remote cloud services. However, the amount of available

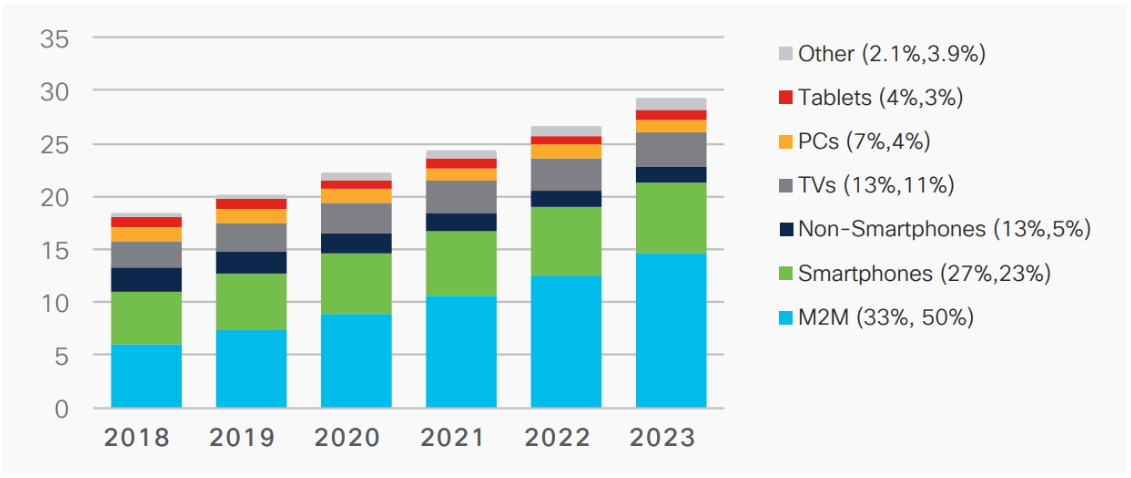


Figure 1.1: Global device and connection growth (billion) [25]

resources at the edge is scarce, and thus it is necessary to manage them efficiently to handle the ever-increasing user demands.

Recently, researchers [11, 27, 78, 80] have proposed to apply NFV to MEC to provide more added-values, such as low cost and high efficiency. Network function virtualization (NFV) [4] has emerged as a networking-computing paradigm that enables efficient utilization of computing and networking resources by applying virtualization technologies to offer network services. In this paradigm, network services are implemented as software modules called virtual network functions (VNFs) that can be dynamically deployed, scaled, and chained together to offer a variety of services to users. In particular, the NFV paradigm is well suited to mobile environments where users freely roam at the edge of the network and dynamically change their point of attachment to the network. In NFV, service demands are expressed in the form of Service Function Chains (SFCs). An SFC specifies a sequence of VNFs that user traffic has to pass through in order to attain the required service. One of the main challenges of implementing SFCs in MEC is the placement of VNFs on the limited resources that are available at the edge, refer to as **SFC Orchestration**. User mobility further complicates the placement decisions, where due to user mobility, it may be possible to migrate an SFC or part of it closer to the new location of the user to reduce the communication delay, or continue to run the VNFs in their current locations to avoid service migration delays. Therefore, a trade-off between communication delay and migration delay is often considered when placing VNFs at the network edge.

Motiviating Example. An example of SFC orchestration in MEC can be illustrated in visual cloud computing for 3D light detection and ranging (LIDAR) [27], which deal with different data processing stages such as: 1) acquisition; 2) preprocessing; 3) analysis; and 4) postprocessing. The LIDAR pipeline is outlined in figure 1. When scanning and modeling moving objects in 3D space, the processing stages are divided into two categories based on computational needs: (1) small instance processing: Camera metadata data processing, static background registration, and 3D rendering; (2) large instance processing: video camera pose computation, motion segmentation, and dynamic object positioning. In this example, we can consider each processing function as a service function in SFC orchestration, and a good strategy is to place small instance processing functions on the edge nodes that have limited computing resources but shorter access latency and place the large instance processing functions on the cloud that has sufficient capacity but longer access latency.

Another example of real-time SFC orchestration is shown in figure 1.3. This shows a geo-distributed latency sensitive SFC for the computer vision of a real-time object tracking pipeline[11]. The pre-processing and Human-Computer Interaction analysis functions are placed on the edge servers while the track function that involves compute-intensive processing is placed on a cloud server.

The majority of previous research on SFC orchestration has focused on data center set-

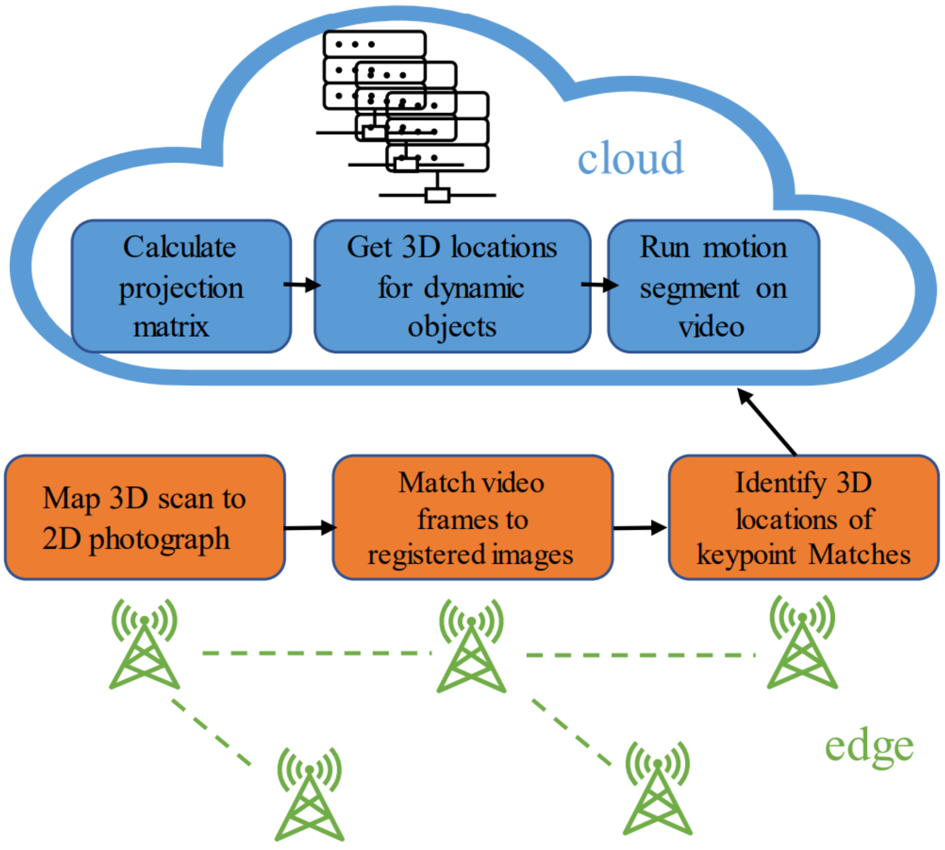

cloud
Calculate projection matrix
Get 3D locations for dynamic objects
Run motion segment on video
Map 3D scan to 2D photograph
Match video frames to registered images
Identify 3D locations of keypoint Matches
edge

Figure 1.2: Example 1: visual cloud computing SFC for 3D light detection and ranging [27]

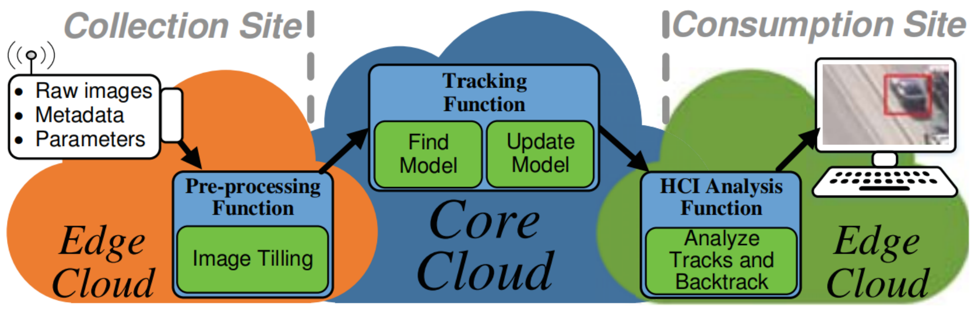


Figure 1.3: Example 2: geo-distributed latency-sensitive SFC used for the real-time object tracking pipeline [11]

tings and does not consider edge resources and user mobility (*e.g.* [58]). Thus, the existing works are not directly applicable to SFC orchestration in MEC. Those works that consider service orchestration in edge-enabled environments belong to one of the two classes: system-managed and user-managed methods. The system-managed methods orchestrate services from a centralized location (*e.g.* [13]). These works have an inherent uncertainty about the users' mobility and face scalability issues as the number of users increases. In contrast, the user-managed methods enable end-users to manage their services in a distributed fashion based on the system feedback (*e.g.* end-to-end delay). Several works use game theory to design user-managed mechanisms [79, 83]. However, these works do not consider the migration cost and can not adapt to system changes. Recently, a few works have applied reinforcement learning and bandit formulation to provide a higher level of adaptability. In the face of problem complexity, these works have focused on the single VNF orchestration to limit the number of so-called *arms* employed in the bandit formulation (*e.g.* [60]). We address this challenge by designing a user-managed SFC orchestration algorithm at the edge that applies reinforcement learning to minimize the user-perceived end-to-end delay while limiting the number of arms required for modeling SFCs by utilizing the theory of combinatorial bandits.

Therefore in this work, in order to take users' behavior into consideration while placing SFCs, we stand on the user-side perspective and propose an original contextual online learning approach that can cope with system uncertainty in edge and user-specific context.

1.2 Thesis Objective

In this thesis, we consider the problem of online SFC orchestration by mobile users with no prior knowledge of system side information (*i.e.*, server capacities and link delays) in an edge-enabled environment, with the objective of minimizing the user-perceived end-to-end latency. More specifically, the uses generate a list of VNF demands representing the SFC request and decide which computing nodes they deploy the SFC on, yet they have no knowledge of the infrastructure information, including server capacities and link delays. The SFC orchestration is to decide, from the pool of resources available on the local device (*i.e.*, hand-held equipment), at the edge and in a remote cloud, where to place each VNF of a service in order to minimize end-to-end service delay. In this optimization problem, there are three main difficulties that need to be considered:

- *Unknown environment*: The user has no knowledge of system information or future information.
- *User's mobility*: User may roam throughout the MEC region during a continuous SFC-related service.
- *VNF's migration*: VNF migration may be needed when the user moves too far from their original served region.

In our formulation of the problem, we consider the end-to-end service delay incurred due to processing, propagation, transmission, and service migration. We call this problem User-managed Edge-enabled Chain Orchestration Problem (UECOP) and formulate it as a Mixed-Integer Program (MIP). To solve the problem, we design an algorithm, called

CHANGE. In CHANGE, we employ the contextual combinatorial bandit framework to enable users to efficiently collect information from the environment and compute delay-optimized service placements. The context allows users to incorporate any available information (e.g., about demands) into the resource allocation formulation. We formulate the problem as a combinatorial bandit to focus on the basic user options (i.e., servers and links) during the learning phase of our algorithm, which significantly reduces the solution space of the problem. A naive formulation would force users to consider all possible solutions, whose number grows exponentially in terms of the number of servers and links (e.g., all the different paths in the network). Therefore, CHANGE can efficiently learn about the environment, i.e., fast convergence, without incurring a high processing penalty. For orchestrating services efficiently, CHANGE uses a fast dynamic programming-based subroutine to make allocation and migration decisions based on the learned information. Finally, while the majority of exiting works in this area use numerical computations and simulations to evaluate their proposed algorithms, we use Mininet-WiFi to implement our proposed scheme and examine its performance and behavior in an emulated environment with a realistic setting and various mobility models.

1.3 Thesis Contribution

In this section, we summarize the main contribution made by this work, including an overview of our proposed formulation and approach for UECOP, and the experiments designed for evaluating the proposed approach.

Customized Problem Formulation. We formulate the problem of user-managed SFC orchestration at the edge as an integer program by considering user demands, mobility, and end-to-end service delay, in which each formulation characterizes the key features in our work such as network models, delay models, and service models. The main constraints we consider in our formulation is designed to meet the network requirements, such as avoiding co-locating placement, single-path routing, and link bandwidth constraints. The objective of our problem is a weighted sum of four different delays we considered in our model. Specifically, we define multiple scale factors to specify the relative importance of different components of the end-to-end delay.

Problem Transformation via Contextual CMAB. Due to the uncertainty of system-wide information and the visibility in the user-side state information, we are able to transform the user-managed SFC orchestration problem into a *contextual combinatorial multi-arm bandit* problem. We formulate the delays in the network as arms that is to be selected and define various parameters to store information such as the unknown system feature and cumulative contextual feature. The contextual combinatorial formulation allows the user to focus on the primitive options, thus greatly reduces the decision space and incorporate any available information to make better decisions. Furthermore, we adopted the *contextual combinatorial upper confidence bound algorithm* (C^{2}UCB) [62] in our decision making to balance exploration and exploitation. Lastly, We present a contextual combinatorial bandit learning algorithm to efficiently learn about the available resources on local equipment, at the edge, and in a remote cloud.

Dynamic programming chain allocation. Our C^{2}MAB formulation for user-managed SFC orchestration in MEC is an online learning problem that is unlikely to solve optimally due to the environment and time span uncertainty. However, at each learning slot, finding the current optimal SFC placement solution can be solved in a reasonable time with a traditional optimization method. Therefore, we proposed a dynamic program (DP) based chain allocation method to optimally allocate SFC at each round, which takes the revealed estimates as input and take user-side specific contextual information into consideration. DP finds the optimal solution by starting from placing one single service and successively moving to the next service while taking the previous solutions into account. Our proposed DP-based approach is able to find the per-time-slot optimum in polynomial time.

Simulation and Mininet-wifi emulation experiments. We present extensive simulation results to study the behavior of our algorithms under different experiment parameters and problem scales. By comparing to the offline optimum, we are able to show that our proposed approach can achieve a close-to-optimum performance. We also compare the performance of our algorithms against two other greedy-based learning approaches and one offline benchmark to demonstrate the superior performance of the proposed algorithm. Furthermore, we simulate a MEC system with user mobility using *mininet-wifi* [23] and implement manageable VNFs for each server node. We then perform a set of experiments in Mininet-WiFi to validate the practical performance of the proposed algorithm.

1.4 Thesis Organization.

The thesis is divided into nine chapters. The content of each chapter is summarized below.

Chapter 1 discussed the motivation for this study, provides a summary of the current research scope and our research objective, as well as an outline of the main contributions in this work.

Chapter 2 provides the necessary background on NFV, MEC, SFC techniques, discusses the online learning approaches as well as the mathematical optimization techniques employed in this study, and gives an overview of the software tools used in experiments.

Chapter 3 reviews the most relevant literature that focuses on SFC placement problems and classifies them into different categories based on whether they are edge-enabled and user-managed.

Chapter 4 describes the system model and present the optimization formulation for offline UECOP.

Chapter 5 presents the contextual combinatorial bandit formulation for the proposed SFC placement problem and the design of our proposed dynamic programming-based online learning algorithm.

Chapter 6 presents the design of a time-stamp based delay estimation method that is required in the bandit formulation.

Chapter 7 presents the extensive simulations results to study the learning behavior of the proposed algorithms and demonstrate their superior performance on scalability compared to the greedy approaches.

Chapter 8 presents the Mininet-WiFi experiment results to validate the practical performance of the proposed approach

Chapter 9 concludes the thesis with a summary of the works and discussions for future research orientations.

Chapter 2

Backround

This chapter presents the background knowledge for understanding our work in this thesis. Particularly, Section 2.1 provides a summary of basic knowledge on NFV architecture. Section 2.2 gives an introduction of the newly emerged MEC technologies. Section 2.3 specifically focus on the SFC architecture and its optimization opportunities. Then, Section 2.4 discusses the online learning techniques and especially the C^{2}MABalgorithm adopted in this work in order to solve the user-managed online SFC orchestration problem. Section 2.5 discusses the mathematical techniques employed in this work to model and solve the offline SFC orchestration on edge. Lastly, Section 2.6 gives an overview of the software tools used for simulation and emulation experiments.

2.1 Network Function Virtualization

2.1.1 NFV architecture

Network Function Virtualization (NFV) is a network architecture concept that is first introduced in 2012 by the European Telecommunications Standards Institute (ETSI) [61], in which they propose to softwarize the traditional network appliances that are physically installed (*e.g.*, Deep Packet Inspection (DPI), Firewall, Message Router), and implement these functions in a way that they can be run on a range of industry-standard server, and can be migrated and placed in various locations in the network as required, without the need

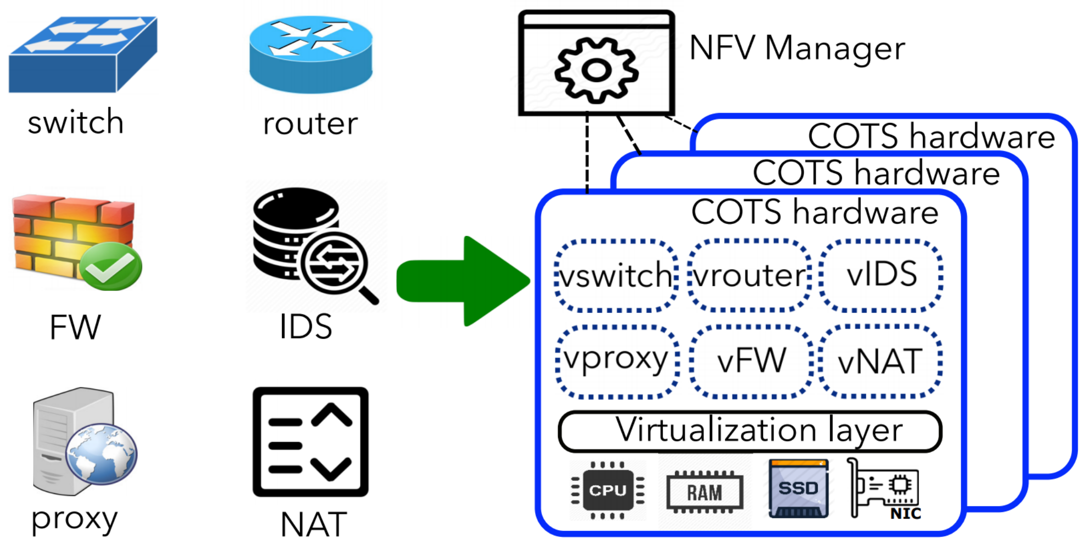


Figure 2.1: An illustrative example of NFV paradigm [84]

of installing new physical equipment. Each softwarized implementations of these network appliances are defined as a virtual network function (VNF). These VNFs can be automatically and remotely installed in a shared computing platform and have functionality as their hardware counterparts in principle. An example of the NFV paradigm is shown in fig 2.1, where specialized middleboxes are replaced with VNFs consolidated on Commodity Off-The-Shelf (COTS) hardware [84]. Virtualising Network Function has potential benefits such as reducing equipment cost, increasing the speed of Time to Market (*i.e.*, the time it takes to implement a network service), and enabling multi-tenancy/multi-version network appliances. Moreover, NFV architecture provides users with enhanced manageability over their personalized applications. Compared to traditional network architectures, the advantages of adopting NFV can be summarized according to NFV white paper [61] as follow: (1) reduced equipment costs, (2) improved operating performance and operational efficiency, (3) optimized network configuration and resource allocation, (4) flexible network function deployment and dynamic operation, and (5) reduced energy consumption.

2.1.2 NFV optimization

Furthermore, NFV architecture provides various optimization opportunities for researchers such as latency, resource consumption, VNF deployment cost minimization, and utility maximization. In a typical resource allocation problem in an NFV-based network (NFV-RA), there are three stages described below.

1) *VNF chain composition*: VNF chain composition, referred to as Service function Chaining in this work, is the problem of dynamically and strategically composing and deploying SFCs on a set of physical works in an NFV architecture, in order to meet a predefined service requirement.

2) *VNF Chain Embedding*: The second stage is called VNF Chain Embedding, also referred to as SFC placement or orchestration in this work, which aims to find the physical node in the network infrastructure to employ the VNFs that suits the requested network services. Figure 2.2 shows an example of an end-to-end VNF chain (Firewall → LoadBalancing → Encryption → PacketInspection → Decryption) embedding.

3) *VNF Chain Scheduling*: The final stage is to schedule the VNFs on the chain properly. Specifically, this stage seeks to minimize the total execution time of the requested network services by scheduling the execution of each VNF, for example, execute some VNFs simultaneously or execute each VNF in order of the chain.

In this work, our focus is mainly on the second stage of an NFV-RA problem, that is, the placement/embedding/orchestration of the VNF chain to physical, we refer to it as **SFC orchestration** in this thesis. In section 2.3 we further discuss the standard SFC architecture,

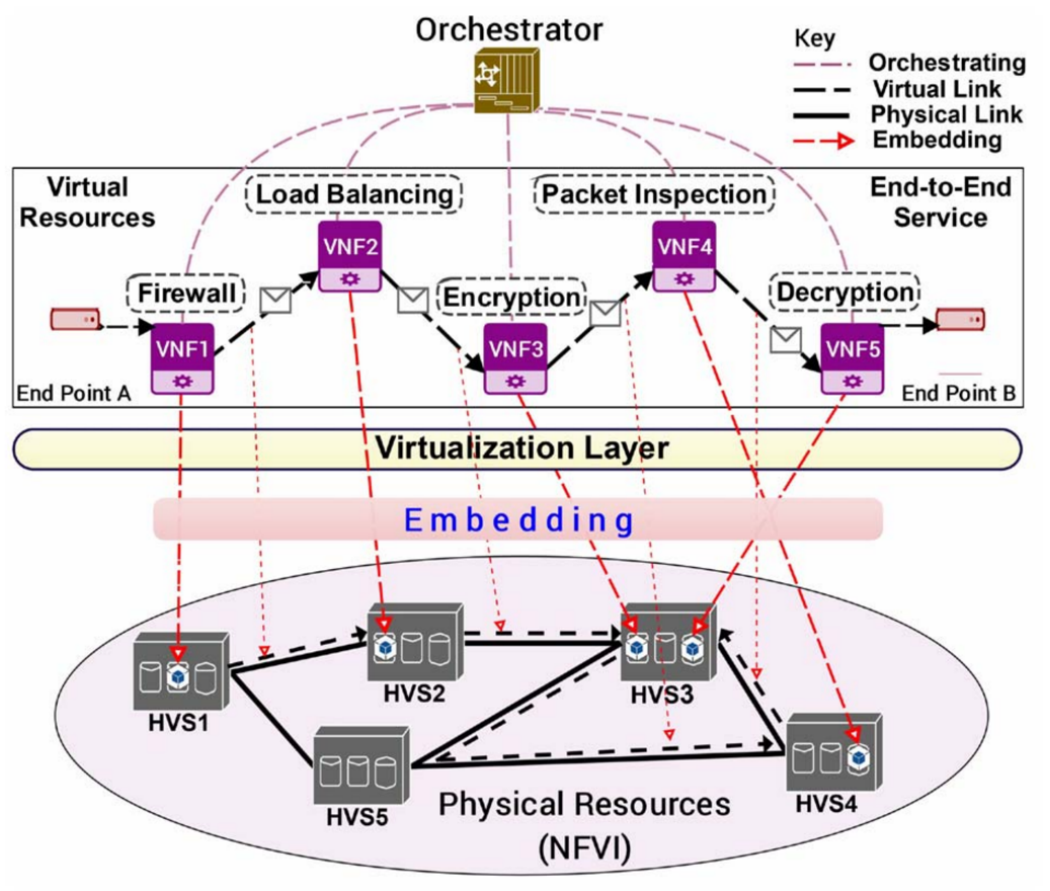


Figure 2.2: VNF Chain Embedding [33]

mathematical and algorithmic approaches for SFC orchestration.

2.2 Mobile Edge Computing

2.2.1 MEC architecture

Mobile Edge Computing (MEC) was first standardized by European Telecommunications Standards Institute (ETSI) and Industry Specification Group (ISG), it is introduced as an integration of edge computing (also known as fog computing) and mobile computing, which empowers Mobile Cloud Computing (MCC) by distributing cloud resources (*e.g.*storage and processing capacity) to the edge servers inside the range of radio access network (RAN). An illustration of a MEC system is shown in fig 2.3, which consists of two major components: mobile devices (e.g. end-users, clients, and service subscribers) and MEC servers. MEC server is usually a small data center deployed by cloud and telecom operators that can be placed close to the end-user and co-located with the wireless APs. The edge servers are connected to the data centers through a gateway via a backbone internet, while mobile devices are connected to the edge servers via a well-established wireless link implemented using advanced wireless communications and network technologies [51]. In section 4.1 we present our system model formulation for MEC that consist of one cloud center and multiple edge servers.

Numerous works in different fields have been discussed in the literature that can be exemplified as a application scenario for MEC, including but not limited to: Health care [82], Video Analytics[35], Mobile Big Data Analytics [44], Connected Vehicles [20], Smart Grid[49], Wireless Sensor and Actuator Networks [41], Smart building Control, SDNs [68],

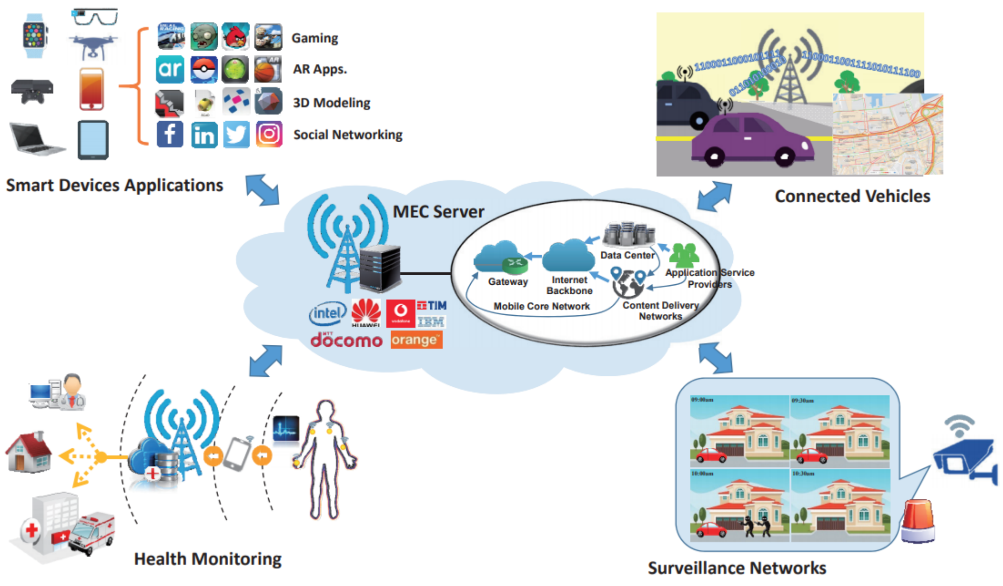


Figure 2.3: Architecture of the MEC system [51]

Ocean Monitoring [3]. According to the white paper published by ETSI [22], a brief summary of the key features of MEC is characterized below.

On-premises. MEC can run isolated from the rest of the network while having access to local resources. This plays an essential role in Machine-to-Machine (M2M) scenarios such as security or safety systems that require high-level resilience.

Proximity. Being close proximity to resources, MEC has an advantage in capturing and analyzing big data. It also provides direct access to devices, which can benefit compute-hungry applications such as augmented reality (AR) and video analytics.

Lower Latency. Edge server is usually deployed at the nearest location of the mobile user, which considerably reduces user-perceived latency to their devices. This can greatly boost user experience and reduce the congestion of other parts of the network.

Location awareness. MEC devices can utilize low-level signaling information to determine the location of other connected devices under different types of the wireless network, which provide mobile users with location-based services and analytics.

Network context awareness. RAN Real-time network information such as the congestion of the radio cell and network bandwidth can be estimated by applications that adopt MEC in their implementation. This offers context awareness that can help to improve the mobile user's experience.

In summary, MEC provides the end-user with swift and powerful computing, energy efficiency, storage capacity, mobility, location, and context awareness support.

2.2.2 MEC research topics

MEC has provided numerous research topics in computer science, and we summarize the main topics that have been studied in this area and discuss where our work stands.

Computational Offloading. The most discussed research topic in the MEC area is computation offloading, which is the process of migrating computing tasks to external resources such as clouds, grids, or clusters [48]. This increases the computing ability of mobile devices by running computing-intensive and resource-demanding applications (e.g. 3-D gaming and video encoding) on edge cloud. For example, offloading web application execution on the edge servers that is within the RAN can accelerate web browsing [71].

Using well-designed algorithms and approaches, the computational offloading process can also be optimized in a radio access environment, which can reduce signal interference and energy consumption [17, 66].

Low Latency. MEC can greatly improve users' experience by lowering the latency of their requested applications and services. For example, latency can be effectively reduced by intelligently scheduling memory replication events in the edge cloud while resolving conflicts for wireless resources[2]. Integrated with 5G technologies, MEC can provide real-time context-aware support for applications such as live remote robotic telesurgery and road accident[59], which require context-information (e.g. geographical information). MEC allows ultralow latency for these applications by satisfying the context demands. MEC can also support vehicular delay-tolerant network by utilizing smart grid devices that communicate with the MEC environment. It addresses the communication complications caused by the high mobility of vehicles by interacting with mobile smart devices that monitor the data sets[42].

Storage support. MEC also provides users with additional storage support when their local device storage is limited. MEC storage capabilities can be further enhanced by integrating with virtualization techniques such as software-defined network, software-defined compute, and software-defined storage[38], which enable the support for storage/computing-intensive applications such as traffic monitoring and mobile gaming.

Energy Efficiency. By running computational tasks on edge cloud architecture, MEC can reduce the energy consumption of users' local devices.

For example, mobile devices can share the energy and computational resources on edge by adequately estimating the network status, which ensures the computation time is synchronized [26]. MEC can significantly prolong the battery lifetime of user devices by offloading computing-intensive tasks like video encoding to the edge servers that employ advanced encoding services[6]. Instead of migrating computing-intensive to the edge, MEC architecture can also coordinate resources between resource-rich mobiles devices and resource-constraint mobile devices in order to escalate the power management of these devices.

In this work, our focus is on computational offloading and latency reduction in MEC architecture. In particular, we consider the optimization of the end-to-end latency to run SFCs in MEC, for which we design an online SFC orchestration algorithm and offload SFC tasks accordingly.

2.3 Service Function Chain

Service Function Chain is defined as a sequence of multiple VNFs that user's traffic has to traverse through to realize their required services. This section provides some background knowledge of SFC, including SFC architecture, SFC request model, SFC Resource allocation problems, and optimization approaches.

2.3.1 SFC architecture

A typical SDN-based SFC architecture consists of the SFC control plane and SFC data plane, as defined by the IETF SFC and ONF working groups. Fig 2.4 shows an illustration of a standard SFC architecture that is explained as follows:

SFC control plane. The control plane is responsible for the management of SFC, such as the management of each service function instance (SFI), the mapping of SFC to a specific service function path (SFP), the administration of forwarding rules, and the adjusting of SFP with regards to the link status.

SFC data plane. SFC data plane consist of: SFC classifier, Service function forwarding (SFF), Service function (SF) and SFC proxy. the SFC classifier differentiates the incoming traffic into flows and tags each flow with an SFP header. An SF executes a particular set of actions on the incoming packets (e.g deep packet inspection or firewall functions). An SFC proxy de-capsulate the packets when the majority of SFs do not recognize the SFC packet

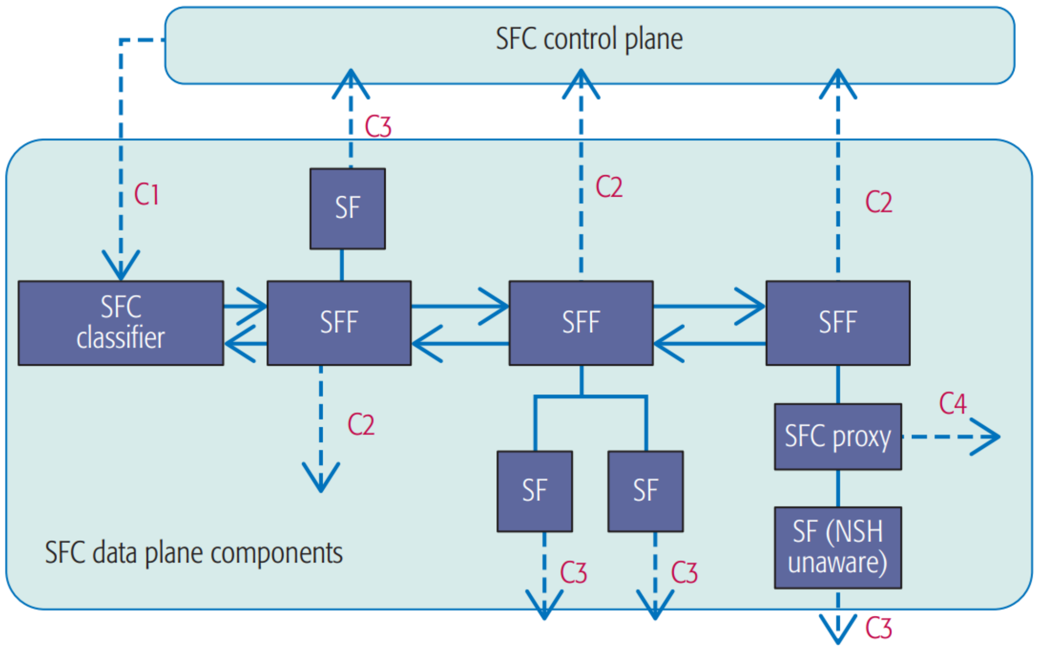


Figure 2.4: A typical SFC architecture [52]

headers.

2.3.2 SFC request model

In order to realize a particular service, the user's traffic flow must be directed through an ordered sequence of VNF instances. For example, an SFC of NAT, FW, and IDS, as depicted in fig 2.5. The set of enabled services represents the operators' services, which is built according to the service agreements between operators and end-users with regards to network policies [77]

An SFC request is a source-to-destination path that contains an ordered list of service functions, and the users may have various bandwidth and computational demands for each service function. We model our SFC demand in section 4.1.2, in which we define λ as the user-generated traffic rate and add customized scale factor α_{s_i} to it with respect to each type

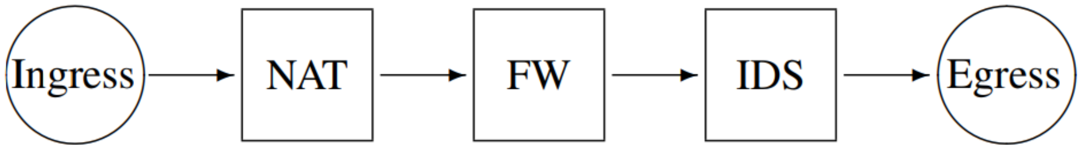


Figure 2.5: Service Request Model [77]

of service s_i.

2.3.3 SFC orchestration problem

Delay Aware. In a delay-aware SFC orchestration problem, a physical network and a set of service requests are given, as well as the number of needed VNF types and demand specifications. The goal is to minimize the total end-to-end delay. Therefore delay-aware SFC orchestration problem has two steps:

1. Calculate the latency on each link and node with regards to the service demand and capacity.

2. Place SFCs to physical nodes and links such the total end-to-end delay is minimized.

In addition, a virtual link on SFC may be placed on several physical links.

Cost Aware. The objective of the cost-aware SFC orchestration problem is to minimize the deployment cost when placing SFCs to physical nodes and links, which is often expressed as the resource and bandwidth consumption.

2.3.4 SFC orchestration optimization approaches

To orchestrate a SFC via an NFV architecture, the user needs to choose a set of computing platforms to install each VNF on a SFC and decide the SFC's routing order while considering various constraints such as fluctuated SFC demands and quality of service (QoS). The problem of SFC orchestration is one of the main challenges in NFV. Therefore many works have taken aims at addressing it, and many optimization approaches have been investigated. They can be summarized as below.

Exact and approximation solutions. SFC orchestration problem can be typically formulated into: Integer Linear Programming (ILP) or Mixed Integer Linear Programming (MILP). We can use traditional exact approaches to solve ILP, *e.g.*, *dynamic programming, branch and bound, cutting plane methods* The work in this thesis uses DP to solve per-time-slot SFC orchestration in a guaranteed time. *Approximation solutions* give a trade-off between optimal solution and algorithm complexity, which achieves a near-optimal but polynomial-time performance.

Heuristic solutions. In practice, SFC orchestration is required to be low latency or real-time. Thus fast heuristics-solution is often preferred due to the hardness of the SFC orchestration problem. *e.g.*, *simulated annealing, tabu search, genetic algorithm, ant colony, best-fit decreasing.*

2.4 Learning algorithm

In this section, we discuss the learning algorithms we applied in this work.

2.4.1 Online learning

Online learning is the process of taking actions given knowledge of rewards of the previous actions and possibly available information. It takes place in continuous rounds. On each round, the learner is presented with a set of actions and is required to take one of them. The goal of online learning is to maximize the accuracy of the learner's predictions so that correct decisions can be made, which is in contrast to a traditional offline learning method that is to learn a model from the entire training data set at once [34]. Therefore, online learning can instantly learn from newly arrived data and are easier to implement than offline learning. In this work, user-managed SFC orchestration can be modeled as an online learning process due to the sequentialness of the user requests.

Online binary learning can be exemplified as a simple online learning, where the reward/outcome of each action is binary: yes or no. Algorithm 1 summarizes a brief procedure of online binary learning, where $\mathbf{x}_t$ denotes the sequential instance and y_t denotes the binary reward label of each instance. At each round, the learning use vector $\mathbf{w}_t$ as a prediction model to calculate the predicted reward label : $\hat{y}_t = \text{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$. Then the learner receives the true reward label y_t after taking the predicted action, and calculate the suffered loss: $l_t(\mathbf{w}_t) = \max(0, 1 - y_t \mathbf{w}_t)^\top \mathbf{x}_t)$. Lastly, the learner strategically update the prediction model based on the loss and true labels received.

2.4.2 Bandit online learning

As an import branch of online learning, Bandit online learning, a.k.a. Multi-Armed Bandit (MAB) problem, has been studied extensively in the online learning community. MAB is normally formulated as a sequential decision-making problem where the decision-maker is

```
Algorithm 1: Online Binary Learning
1 Initialize the prediction function as $\mathbf{w}_1$;
2 for $t = 1, 2, ..., T$ do
3     Receive instance: $\mathbf{x}_t \in \mathbb{R}^d$;
4     Predict $\hat{y}_t = \text{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$;
5     Receive the true reward label: $y_t \in \{-1, +1\}$;
6     Suffer loss: $l_t(\mathbf{w}_t) = \max(0, 1 - y_t \mathbf{w}_t)^\top \mathbf{x}_t)$;
7     Update the prediction model $\mathbf{w}_t$ to $\mathbf{w}_{t+1}$;
8 end
9 Calculate regret: $R_T = \sum_{t=1}^{T} l_t(\mathbf{w_t}) - \min_{\mathbf{w}} \sum_{t=1}^{T} l_t(\mathbf{w})$;
```

presented with m arms to select from at each of n rounds, where T is often unknown at the beginning and decisions are made based only on the feedback from the environment. The distribution of reward on each arm is unknown, and the goal of the problem is to maximize the total reward or minimize the total loss over the course of n rounds, with a_t denoting the action of each round, $r_t(a_t)$ and $l_t(a_t)$ denoting the reward and loss of that action respectively. We can formally define the "regret" as the difference of cumulative loss between the optimal arms and player selected arms:

$$R_T = \sum_{t=1}^{T} l_t(a_t) - \min_{i \in [m]} \sum_{t=1}^{T} l_t(i) \tag{2.1}$$

One of the major challenges of MAB is how to trade-off between *exploitation* and *exploration*, where *exploitation* of past actions ensure high payoffs based on the past knowledge and *exploration* of new actions gives possibly higher payoffs in the future, which is also one of the major concerns in this work.

ϵ-Greedy Multi-Arm Bandit. The first simplest MAB based algorithm is ϵ-Greedy MAB introduced in [70], in which the player plays the arm that currently has the highest average reward with probability $1 - \epsilon$ and plays a random arm with probability ϵ, where ϵ is a constant value in (0, 1). ϵ-Greedy Multi-Arm Bandit algorithm is summarized in algorithm

2, where $N_{i_t}(t)$ denote the number of times arm i has been selected by the player until t rounds and μ_{i_t} denotes the mean of the rewards received on arm i.

In this work, ϵ-Greedy Multi-Arm Bandit algorithm is used as a performance comparison in experiments presented in chapter 7 and 8.

Algorithm 2: ϵ-greedy MAB

```
1 INPUT: parameter $\epsilon > 0$
2 INIT: empirical means $\mu_i = 0, \forall i \in [m]$
3 for $t = 1, 2, ..., T$ do
4     with probability 1-$\epsilon$ play the current best arm $i_t = \text{argmin}_{i \in [m]} \mu_i$
5     with probability $\epsilon$ play a random arm
6     receive $l_t(i_t)$ and $r_t(i_t)$
7     update the empirical means $\mu_{i_t} = (\mu_{i_t} * N_{i_t}(t) + r_t(i_t))/(N_{i_t} + 1)$
8 end
```

Contextual Combinatorial Multi-Arm Bandit. In the framework of a Combinatorial multi-armed bandit (CMAB), the decision-maker needs to select a set of arms(referred to as a super arm) and play it, after a super arm is played and the reward of each arm in the super arm and other triggered arms are revealed at each round. When each arm can be characterized by a feature vector that the decision-maker is able to observe, the problem is known as Contextual Combinatorial Multi-arm bandit (C^{2}MAB) problem, which is often used to adapt to user feedback and diverse interest. Our work is inspired by the contextual combinatorial upper confidence bound algorithm (C^{2}UCB) presented in [14], which is a general algorithms for addressing C^{2}MABproblems. C^{2}UCB characterize the user-observed context using a set of feature vectors $\mathbf{X}_t = \{\mathbf{x}_t(i), ..., \mathbf{x}_t(m)\}$ corresponding to m arms at each round t and use vector θ_* to characterize the system feature, which is unknown to user. C^{2}UCB then define the reward function on each arm to be:

$$r_t(i) = \theta_*^T \mathbf{x}_t(i) + \epsilon_t(i), \tag{2.2}$$

where, $\epsilon_t(i)$ is a zero-mean random variable that represent the noise in the system. The goal of C^{2}UCB is to maximize the expected cumulative reward $\mathbb{E}[\sum_{t\in n} R_t(S_t)]$ over n rounds, where $R_t(S_t)$ represent the sum reward received for all the arms on the selected super arm S_t at round t. The algorithm use the feature vectors and previously observed rewards to maintain a confidence set for true parameter θ_*, which is then used to compute an reward upper confidence bound for each arm: $\hat{\mathbf{r}}_t = \hat{r}_t(1), ..., \hat{r}_t(m)$ according to equation 2.2. $\hat{\mathbf{r}}_t$ and $\mathbf{X}_t$ is taken as a input to a computation oracle that computes the optimal or near-optimal super arm S_t. The algorithm plays the returned super arm and update the confidence sets using the observed rewards on the super arm. The pseudocode of C^{2}UCB is summarized in algorithm 3, where α_t is an approximation ratio that shows the proximity of the current solution to the optimal solution, $\mathbf{V}_t$ and $\mathbf{b}_t$ are two auxiliary vectors used to update the system paramter θ_t.

In this work, chapter 5 presents our proposed algorithm that adopts the C^{2}UCB framework, which uses an optimal oracle that is based on dynamic programming (e.g, $\alpha_t = 1$)

Algorithm 3: C^{2}UCB

```
1  Input: $\alpha_1, ..., \alpha_n$
2  Initialize $\mathbf{V}_0 \leftarrow \mathbf{I}_{d\times d}, \mathbf{b}_0 \leftarrow \mathbf{0}_d$
3  for $t \leftarrow 1, ..., n$ do
4      $\hat{\theta}_t \leftarrow \mathbf{V}_{t-1}^{-1}\mathbf{b}_{t-1}$
5      for $i \leftarrow 1, ..., m$ do
6          $\bar{r}_t(i) \leftarrow \hat{\theta}_t^T \mathbf{x}_t(i)$
7          $\hat{r}_t(i) \leftarrow \bar{r}_t(i) + \alpha_t \sqrt{\mathbf{x}_t(i)^T \mathbf{V}_t^{-1} \mathbf{x}_t(i)}$
8      end
9      $\boldsymbol{S}_t \leftarrow \mathcal{O}(\hat{\boldsymbol{r}}_t, \mathbf{X}_t)$
10     Play super arm $\boldsymbol{S}_t$ and observe$\{r_t(i)\}_{i\in S_t}$
11     $\mathbf{V}_t \leftarrow \mathbf{V}_{t-1} + \sum_{i\in S_t} \mathbf{x}_t(i)\mathbf{x}_t(i)^T$
12     $\mathbf{b}_t \leftarrow \mathbf{b}_{t-1} + \sum_{i\in S_t} r_t(i)\mathbf{x}_t(i)$
13 end
```

2.5 Mathematical Tools

In this section, we present the two mathematical tools we used for the optimizations considered in this work.

2.5.1 Integer Linear problem

Integer Linear problem is a type of constrained optimization, which consists of an objective function and a set of constraints. In ILP, all the variables in the objective function are restricted to be integers while all the functions and constraints are linear. In this work, we formulate UECOP as a specific form of ILP that is Binary integer linear problem (BILP), in which all of the variables are binary, that is, they can only take on the value of 0 or 1. This is often used to represent a selection or rejection of an option, *e.g.* a yes/no answer, a turning on/off of a switch, and in our work, a decision of whether or not to select a certain node and link. Formulation 2.3 illustrates an example of a standard constrained binary integer linear problem. In this program, x_i are the *variables* while c_i and $a_{i,j}$ are the *constant coefficient*, the expression 2.3a describes the objective of the BILP as a linear function of variable x_i, expression 2.3b and 2.3c are the inequality and equality constraints that x_i is subjected to respectively , while expression 2.3d restrict x_i to be variables only. An standard form also restricts that the variables are ordered according to the objective function coefficients, which is expressed in constraint 2.3e.

The formulation above may seem restrictive, but it is easy to convert many problems to this form. For example, we can handle the negative objective function by simply replacing x_i with $1 - x_i$. Negative constraints can also be handled by converting $\leq$ form to $\geq$ form,

Formulation 1: A Standard Binary Integer Linear Program

$$\textbf{Minimize Z} = \sum_{i=1}^{n} c_i x_i \tag{2.3a}$$

$$\textbf{Subject to:} \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad \forall i \in 1, 2, ..., h \tag{2.3b}$$

$$\sum_{j=1} a_{ij} x_j = b_i, \quad \forall i \in h+1, h+2, ..., m \tag{2.3c}$$

$$x_i = 0 \text{ or } 1, \quad \forall \in 1, 2, ..., n \tag{2.3d}$$

$$0 \leq c_1 \leq c_2 \leq ... \leq c_n \tag{2.3e}$$

and reordering the variables is easy.

Linearization. A common linearization of BILP is to linearize a product of multiple binary variables, which can be done by a general method showed in formulation 2, where a product of n binary varibles x_i can be represented by a new binary variable z and additional constraints. The linearization concerned in this work is a product of two binary variables, more specifically, we linearize the migration delay part of the objective using this method as expressed in equation 4.12.

Formulation 2: BIP Product Linearization

$$\begin{aligned} z &= \prod_{i=1}^{n} x_i \\ z &\leq x_i \\ z &\geq x_i - (n-1) \end{aligned} \tag{2.4}$$

2.5.2 Dynamic program

A recursive algorithm can be designed to solve the problem more efficiently by taking advantage of the similarities in the substructures of a problem,[9]. This type of algorithm first divides the problem into many subproblems recursively, each of which has the same structure. The algorithm then repeatedly calls itself to solve each subproblem. Because the problem is divided recursively, there are many instances where subproblems in different branches of the recursion tree are exactly identical to one another. Rather than separately solving all subproblems, which wastes computing resources and time, Dynamic Programming proposes to store the results of the subproblems and reuse them when necessary [9]. By apply such a simple technique, the number of recursive steps needs to solve the problem can be reduced from exponential to polynomial. DP is an optimization approach that has been widely used in common problems such as Min/Max Knapsack, Shortest Path, and Shortest Common/Uncommon Subsequence, etc. With respect to this work, Dynamic Programming is applied to solve the recurrence equation **??**, which is used to find the optimal per-time-slot solution of SFC orchestration. The idea of the DP-based SFC orchestration is first to compute the optimal solutions for small subproblems (mapping a virtual service function and its associated virtual) and store those values, which is then used to solve larger subproblems until the overall problem (a whole SFC) is solved. A general DP framework for SFC orchestration is presented in algorithm 4, in which each VNF is assigned with a matrix D_i that stores the cost of sub-optimal SFC orchestrations through iterations and eventually the complete SFC orchestration. Specifically, diagonal elements ($D(j, j)$) of the matrix store node-related cost and none-diagonal elements ($D(k, j)$) store link-related cost.

Algorithm 4: A general DP algorithm for SFC orchestration

```
1  Input: SFC ($s$ services), $n$ computing nodes
2  Output: hostsList: substrate nodes hosting the SFC
3  Initialize $D \leftarrow \mathbf{I}_{s\times n\times n}, C \leftarrow \mathbf{I}_{s\times n}, H \leftarrow \mathbf{I}_{n\times s}$
4  for $i = s-1; i \geq 0; i--$ do
5      Associate a Matrix $D_i$ to each $vnf_i$
6      $\mathrm{D}_i(j,j) = cost(vnf_i, j)$
7      $\mathrm{D}_i(j,k) = cost((vnf_i, vnf_{i+1}), (j,k))$
8      Associate a hostList $H_j$ for each $D_i(j,j)$ to store the computed chain mapping
9      for $j = 1 \rightarrow n$ do
10         $minCost = \infty$
11         for $k = 1 \rightarrow n$ do
12             if $minCost \geq C$ then
13                 $\mathrm{C}(\mathrm{vnf}_i, j) = D_i(j,j) + D_{i+1}(k,k) + D_{i+1}(j,k)$
14                 $\mathrm{minCost} = \mathrm{C}(\mathrm{vnf}_i, j)$
15                 $\mathrm{host}_{i+1} = k$
16             end
17         end
18         $\mathrm{D}_i(j,j) = minCost$
19         Add $host_{i+1}$ to $H_j$
20     end
21 end
22 Extract the minimum SFC cost from $D_1$by computing the minimum diagonal value
23 $\mathrm{C(SFC)} = \min(\mathrm{D}_1(j,j))\ \forall\ j \in [1,n]$
24 Extract the optimal SFC mapping from $H$
```

2.6 Software Tools

In the simulation experiments, networkx[21] is used to simulate different types of network topologies and weighted graphs. In emulation experiments, Mininet-wifi[23] is deployed to create a mobile wireless network as a simulation to MEC. In both simulation and emulation, Gurobi[30] is used as an offline optimization solver.

2.6.1 Networkx

NetworkX is a Python software package used to create, manipulate and study complex network structure, dynamics, and functions [21]. It is mostly used for the purpose of analysis of network-related algorithms and problems. Networkx allows users to create data structures that represent multiple types of networks or graphs, including simple graphs, directed graphs, multigraphs, and order graphs, in which the nodes and edges are feasible to be assigned with attributes or weights that can be various Python data types and structures (integer, string, list, etc.). In addition to the primary data structure, Many standard graph algorithms are provided and implemented by Networkx to calculate network attributes and structural metrics: shortest paths, simple paths, clustering, and traversal, etc. NetworkX can exchange with existing data by reading and writing various graph formats and generate data for classic graphs and models such as the Erdos-Renyi, Small World, and Barabasi-Albert models [31]. Networkx is a Python-language-based package. Therefore it can interact with many other Python packages such as Numpy, Scipy, and Matplotlib. Networkx is used to create and analyze network topologies with different parameters, such as node/edge attributes, which simulates a dynamic MEC network.

2.6.2 Gurobi

The Gurobi Optimizer [30] is developed by the same team that founded CPLEX[1]. It serves as a solver for a wide variety of optimization problems such as linear programming, quadratically programming, mixed integer programming, etc. It allows users to customize the solver's functioning according to the specifics of the problem by modifying parameters such as convergence tolerance, termination conditions, and optimization algorithms. For example, the convergence tolerance can be adjusted to speed up the optimization process in applications where determining the exact solution is not critical, therefore sacrificing a certain degree of accuracy. The termination condition is used to specify the termination criteria for an optimization model when multiple termination parameters are used, Gurobi will stop when it reaches the first one. Gurobi optimizer provides two main optimization algorithms: barrier and simple, the barrier algorithm works faster for large, intricate models, while simple algorithm is a good alternative for problems that are less numerical sensitive. Gurobi also supports parallel optimization and distributed optimization over multiple processors/machines. Gurobi is known as one of the most accessible and user-friendly optimization solvers. It provides interfaces to multiple modeling and programming languages, including AMPL, Matlab, R, Python, C/C++, and Java. User guidelines and API are comprehensively documented and updated on the Gurobi website, and free licenses can be obtained for academic purpose, which has access to all key features. In this study, Gurobi is used to solve the offline optimization introduced in formulation 3

2.6.3 Mininet-wifi

Mininet-WiFi [24] is a wireless network emulator based on Mininet[73]. Mininet is an open-source SDN network emulator for prototyping network systems and conducting experiments on them. It allows creating virtual network hosts, links, and switches that behave like an entire system on a single machine by utilizing the network namespaces and process virtualization of Linux. Mininet-WiFi extends the functionality of Mininet by adding the virtualization of WiFi Stations and Access Points using the standard Linux wireless driver and the 80211_hwism wireless simulation driver, while supporting all the normal SDN emulation of Mininet as well. In addition, it supports multiple mobility models (e.g. RandomWayPoint, TruncatedLevyWalk, GaussMarkov, etc.) and propagation models (e.g, Friis Propagation Loss Model, Log-Distance Propagation Loss Model, Two-Ray Ground Propagation Loss Model), In section 8.2.2, we study how different mobility models can affect our algorithm performance in Mininet-WiFi.

Stations. Stations are devices that are connected to access points through wireless authentication and association. In Mininet-WiFi, Mininet Hosts are customarily connected to an access point and therefore stations are able to communicate with those hosts. In our Mininet-WiFi experiments presented in Chapter 8, stations are implemented as a simulation for mobile devices, which are configured with propagation models and mobility models.

Access points. Access points (AP) are devices that manage associated stations. In Mininet-WiFi, Access points are virtualized using the hostapd daemon and virtual wireless interfaces. In our emulation experiments, Access Points function as both routers and base stations in MEC that connect the server hosts and mobile users, respectively.

The advantages of using Mininet-WiFi as a MEC emulator can be summarized as follows, (i) Mininet-WiFi provides a lightweight virtualization scheme scripted by python API, allowing easy creation of customized network topologies and settings. (ii) Mininet-WiFi provides support for emulations of various mobility models and propagation models. (iii) Virtual wireless systems emulated by Mininet-WiFi are able to interact with external systems and machines just like in real networks. Thus real MEC applications can be deployed at Mininet-WiFi hosts. (iii) Mininet-WiFi makes it possible to test edge-cloud applications in a simulated mobile edge computing network, which can be overly burdensome in a real-world MEC.

Chapter 3

Related Work

This chapter reviews previously proposed works in the literature that solves the SFC placement problem and relevant to our work. Based on their problem model, these algorithms can be broadly grouped into four classes: (i) System-managed Edge-agnostic, (ii) System-managed Edge-enabled, (iii) User-managed Edge-agnostic, (iv)User-managed Edge-enabled. This chapter is divided into four sections, with each section summarizing the algorithms proposes in one class of problem model. The related work overview is summarized in table 3.1.

3.1 System-managed Edge-agnostic

System-managed SFC placement is based on the assumption that future information (*e.g.*u̇ser mobility and future demands) and system-wide information (*e.g.*ċapacity and bandwidth) is know, works in this category are usually able to formulate their model into one of these models: Integer Linear Program (ILP), Mixed Integer Linear Program (MILP), mixed-integer quadratically constrained programming (MIQCP).

A large number of works have studied the system-managed SFC placement in NFV-enabled network that is edge-agnostic. Therefore those works usually do not consider factors such as mobility and migration cost, and the objectives can be categorized into three groups: delay-aware, cost-aware, and hybrid. For example, authors in [72] proposed a heuristic algorithm that addresses resource allocation in a System-managed Edge-agnostic model, and their approach is both cost-effective and delay-aware. Authors in [55] consider a delay-aware objective for multicast and multi resources application in NFV-enabled network such as Internet Protocol Television (IPTV), video streaming platforms. Authors in [40], however, consider an energy-aware objective and used Genetic Algorithm (GA) to solve the SFC placement in a multi-cloud model.

3.2 System-managed Edge-enabled

Recently, many works have proposed to deploy MEC application in NFC as service function chains, those works focus mainly on the system-managed edge-enabled SFC placement. We can further deliberate about these works based on their objective, optimization models, and different heuristics.

3.2.1 Delay-related objective

Many studies address SFC placement intending to minimize the overall delay of SFC. One typical delay model consists of four different communication delays in 1) propagation delay, 2) transmission delay, 3) queuing delay, and 4) processing delay[45]. *e.g.*, Authors in [57] provide a clustered NFV service chaining scheme that computes the optimal number of clusters to minimize the end-to-end delay for MEC services. In dynamic SFC placement, VNF migration or re-location costs are often considered as well[18, 60, 64] due to non-negligible configuration cost and transmission delays. Specifically, authors in [18], and [60] also consider user mobility in the model when computing communication delays.

3.2.2 Resource-related objective

Another primary objective in past publications is the minimization of SFC deployment costs, which is usually expressed as the resources needed to place an SFC. *e.g.*, Authors in [46] consider the SFC resource utilization to be an accumulation of CPU, memory, and bandwidth utilization. The same objective is also pursued in [75], in which the same SFC resource utilization attributes are used to formulate the physical network and SFC requests into weighted graphs. Authors in [54] consider the resource optimization with regards to mono- or multi-tenant context. Authors in [32] however, consider the maximization of the number of satisfied clients as their objective. Authors in [5, 45] consider both delay and resource optimization and try to achieve a balance between these two objectives. Besides the objectives above, some studies also consider quality of service (QoS) requirements (e.g., latency, bandwidth, security) in their placement scheme. [8, 18, 45, 54, 64]. The authors in [45] formulate their objective to consist of the number of computational and communication resources for placing the SFC, and the total delay experienced in the SFC paths. Then they jointly optimize the total objectives and find an optimal trade-off. Rather than minimizing both delay and resources, the work in [5] introduces *Resource-Delay Dependency* to provide a specific end-to-end delay while minimizing resource consumption. In [32], authors aim to maximize the number of satisfied clients, where satisfaction necessitates both the latency constraints and the client's desired network function.

In our work, we focus on user-managed SFC placement. Therefore we consider the minimization of user-perceived end-to-end delay as our objective.

3.2.3 Models

To deal with System-wide SFC placement in cloud & edge architecture, numerous optimization models have been introduced. One standard model of solving the SFC placement problem is to formulate and solve it ILP. Many works have introduced ILP based approaches [18, 45, 46, 54, 64] and solve it as a performance benchmark. To deal with different optimization objectives for SFC placement, authors in [5, 53, 57] formulate their models as MIQCP. In order to jointly minimize a set of requirements, works in [45, 64] introduce a Mixed Integer Programming (MIP) formulation

Besides formulating the problem into traditional optimization models, some works[11, 18, 75] also introduce other LP-based models. Authors in [75] formulate the SFC placement as the Weighted Graph Matching Problem (WGMAP). Authors in [18] apply a dynamic scheduler on their ILP model in order to fit it in a real-world scenario. For an augmented cloud infrastructure, the work in [11] defines the optimal SFC composition as the integer multi commodity-chain flow problem (MCCF).

3.2.4 Heuristic approaches

Besides using the exact model that solves the problem optimally, most works above [45, 46, 54, 64, 75] also propose heuristic-based approaches that can likely achieve a near optimum. These approaches also handle the computational complexity so that it can solve the problem in a polynomial time. *e.g.*, authors in [75] design a Hungarian-based SFC placement algorithm in MEC and compare it with a heuristic-based greedy algorithm, which shows an efficient reduction in execution time.

Some other work[8, 57] only focus on heuristic approaches to solve the problem in order to avoid the complexity of the LP problem or deal with different challenges and contexts, specifically, works in [57] proposed a clustered NFV service chaining scheme in order to reduce the amount of traffic in MEC. As already mentioned before, SFC placement has already been used in different applications such as visual computing[11] and SFC placement can often be solved within a different context in different scenarios. *e.g.*, [54] solves the SFC placement problem allowing adequate management of rare resources to address the multi-tenant issue in edge and core network, some work such as [64] consider the total response time to get the service ensuring user Quality of Service in 5G networks. Authors in [11] address the geo-distributed latency-sensitive SFC placement problem using trace-driven simulations comprising of challenging disaster-incident conditions. However, the works mentioned above mainly focus on the system-wide SFC placement, where the scheduler or agent knows the complete system-wide information and considers one-shot offline optimization. Our work, however, focuses on user-managed online SFC placement that conducts optimization at run time to adapt to network dynamics and unknown future demands and available resources.

3.3 User-managed Edge-agnostic & Edge-enabled

On the user-managed service placement, where the user makes decisions based on their interactions with the environment, many reinforcement-learning-based approaches [29, 39, 47, 50, 56, 65, 76, 85] has been proposed to solve the online SFC placement problem. For example, Q-placement[85] is proposed to optimally place services in an iterative manner with guaranteed performance and convergence rate. Authors in [56] proposed an accelerated RL method that divides the learning process into two steps in order to deal with numerous explorations in real networks. ScaRL is proposed in [39] that leverages reinforcement learning to solve SFC allocation in MEC by using its trial-and-error mechanism. As an improved version of reinforcement learning, deep reinforcement learning (DRL) is proposed in [10, 50, 76] in order to address a more complex and dynamic problem state. e.g. [76] introduce Markov decision process (MDP) model to capture the dynamic network state transitions and proposed DRL to deploy SFCs automatically. In [47], a novel machine learning approach based on quantum physics is proposed to solve SFC placement in massive data scenarios such as dynamic SFC placement on edge clouds. Specifically, [39, 47] both solve the SFC placement in a MEC context and define the computing resources of the edge server as the state set in the learning framework.

However, Most of these works lack considerations of some of the critical features in edge-enabled SFC placement. For example, most of them fail to consider an edge-enabled application scenario: [56, 76, 85] focus on general SDN, [29, 50] focus on traditional core cloud data centers and [65] focus on 5G networks. Most works [10, 39, 47, 50, 56, 65, 76, 85] do not consider VNF migration. Other works like [29, 65] does not consider the user-specific request, and none of the above works consider users' mobility. Besides lacking the context consideration in MEC, none of these solve SFC placement in a combinatorial manner, which means that the reward function of the learning objectives in these work is often defined to be the reward of one complete SFC, this will not only ignore the rewards on independent nodes and links but also make the action space of selecting SFCs exponential. The overview of related work is summarized in table 3.1.

Our work is inspired by the advantage of using multi-armed bandit model [60] to overcome the challenges of lacking both future and system-wide information in UECOP and using contextual combinatorial MAB [62] to characterize the dynamic context of the mobile user and the feature observed on each single arms such as each node and link. To fill the scarcity of MEC-related and user-managed consideration in SFC placement studies, we proposed a novel online-learning SFC orchestration approach with a Combinatorial Contextual Multi-arm Bandit (C^{2}MAB) framework that are able to not only predict users' preference by utilizing user-specific and edge-enabled context but also able to solve the SFC placement problem in a combinatorial manner that allows user to focus on the primitive options.

Table 3.1: Related work overview

Works	**Models**	**Heuristic approaches**	**online**	**Objective**
[18]	ILP	✗	✗	Minimize end-to-end latency from all users to their respective VNFs
[64]	MILP	Ant Colony Optimization	✗	Minimize total VNF relocation and total response time
[45]	MILP	Tabu Search	✗	Minimize the end-to-end communication and the overall deployment cost
[75]	WGMP	Hungarian-based placement	✗	Minimize the total resource consumption and algorithm execution time
[46]	ILP	Priority based Greedy	✗	Minimize the total resource consumption
[57]	MIQCP	cluster based	✗	Minimize the average service time
[54]	ILP	✗	✗	Minimize the total resource consumption
[8]	✗	EdgeUser	✗	Maximize tolerated latency for SFC
[5]	MIQCP	✗	✗	Minimize resource consumption
[11]	MCCF	metapath composite variable	✗	Minimize a sum of SFC demands and corresponding physical resource capacity ratios
[29]	✗	best-fit decreasing	✓	Minimize energy consumption
[85]	✗	Reinforcement Learning	✓	Minimize the average service cost for end users
[60]	ILP	contextual multi-armed bandit	✓	Minimize the total service cost
[39]	ILP	Reinforcement Learning	✓	minimize the transmission latency and processing latency
[32]	ILP	$(1 - 1/e)$ deterministic	✗	Maximize the number of satisfied clients
[53]	MIQCP	✗	✓	Maximize the remaining data
[56]	✗	Reinforcement Learning	✓	throughput latency ratio
[76]	MDP	Deep Reinforcement Learning	✓	minimize the operation cost and maximize the total throughput
[47]	ILP	Quantum machine learning	✓	Minimize the end-to-end delay
[10]	✗	Deep Reinforcement Learning	✓	Minimize the resource cost
[65]	✗	Reinforcement Learning	✓	Minimizing energy consumption of allocating new VNFs
Our work	ILP	BandEdge	✓	Minimize the average response time

Chapter 4

Problem Formulation

This chapter describes different mathematical models we propose for user-managed SFC orchestration in MEC and show how our problem can be formulated as a constrained optimization problem. Specifically, Section 4.1 presents the system model, network model, and service model in user-managed SFC orchestration. Section 4.2 shows the mathematical formulation of the User-managed Edge-enabled Chain orchestration Problem as constrained optimization.

4.1 System Model

In this section, we describe our model. Our model is able to represent existing edge-enabled architectures with required elements for bandwidth and processing resource allocation. We adopt a time-slotted model and use $\mathcal{T} = \{0, \ldots, T\}$ to denote the considered time horizon. Each timeslot $t \in \mathcal{T}$ represents a resource allocation phase with the duration of Θ seconds. In addition, we assume that our system is a user-managed system where each user receives light-weight feedback about the system operation, and is responsible for managing and resource allocation of service individually with the help of the feedback. This system operation scheme keeps the system complexity and scale at a smaller scale, which enables us to achieve better solutions in section 5.

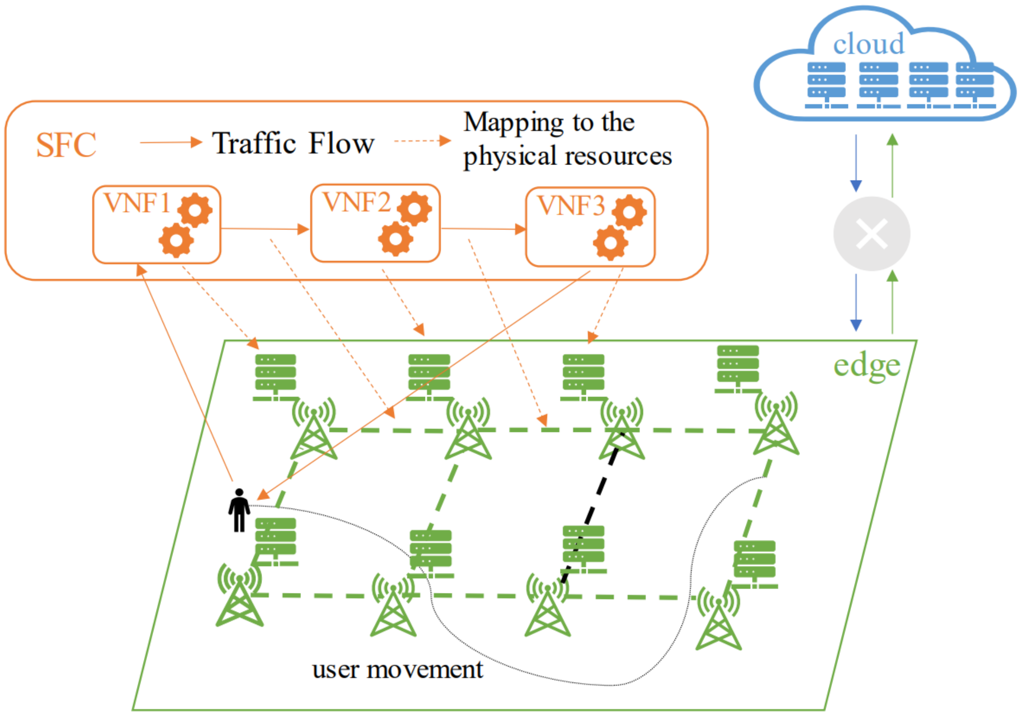


Figure 4.1: An example of SFC placement in MEC

4.1.1 Network Model

As illustrated in figure 4.1, We consider a remote cloud C and a mobile access network consisting of a set of base stations, each of which is bridged with an edge server, and is accessible through wireless channels. Each base station is equipped with an edge server, where the set of all edge servers is denoted by $\mathcal{E}$. The cumulative processing capacity of the cloud and each edge server $e \in \mathcal{E}$ (in terms of the number of CPU cores) in timeslot t is denoted, respectively, by $c_C(t)$ and $c_e(t)$[1]. Edge servers are connected to each other and the remote cloud through a capacitated backbone network $G(\mathcal{R}, \mathcal{L})$, where $\mathcal{R}$ is the set of backbone routers and $\mathcal{L}$ is the set of backbone links. At time t, each link $\ell \in \mathcal{L}$ is associated with the bandwidth capacity $b_\ell(t)$ and propagation delay $d_\ell(t)$. We also refer to a link by its

[1]When cores are heterogeneous we must normalize the numbers with regard to the weakest CPU core in the system

two endpoints, *e.g.*, $d_{a,b}$ is the delay of the link between a and b. Generally, the delay between edge servers and the cloud is significantly higher than the delay between edge servers. While, propagation delay within the cloud is considered to be negligible.

4.1.2 Service Model

We assume that each user who roams in the vicinity of the access network has a mobile device u (*e.g.* Smartphone, tablet, in-vehicle infotainment system) with the processing capacity of $c_u(t)$, which is always connected to the nearest base station (denoted by β). Each user in time slot t generates traffic at rate $\lambda(t)$ megabits per-second (Mbps) and requires it to be processed by a set of pre-determined *services* in strict order (*a.k.a.* a service chain). Each service is a software program (*e.g.* video transcoder, firewall, and logger) that can be deployed in the cloud or an edge server with the means of state-of-the art virtualization methods such as Container [19]. We use $\mathcal{S} = (s_1, \ldots, s_K)$ to denote the list of the user-required services and assume that service s_i incurs π_{s_i} seconds of delay to process 1 Mbps of incoming data by using 1 unit of CPU core. Moreover, each service can scale the user input traffic by a factor of $\alpha_{s_i} \in [0, \infty)$ before sending it to the next service due to operations such as encoding or decoding. Consequently, the traffic rate to the service s_i can be computed by,

$$\lambda_{s_i}(t) = \lambda(t) \times \prod_{j=1}^{i-1} \alpha_{s_j}. \tag{4.1}$$

Note that service migration is inevitable as the user moves in the environment. Let $\mathcal{N} = \mathcal{E} \cup \{C, u\}$ represent all nodes that can host a service and provide the required processing capacity. The *migration delay* $\rho^{s_i}_{a,b}(t)$ is defined as the time that the user has to wait for if her service s_i is migrated from the source node a to the destination node b ($a, b \in \mathcal{N}$) in

Table 4.1: List of main notations in UECOP formulation

Notation	Description
$\mathcal{T}$	Time frame
$\mathcal{E}$	Set of all edge servers
C	Cloud
$\mathcal{L}$	Set of backbone links
$\mathcal{R}$	Set of backbone routers
$b_\ell(t)$	Current bandwidth capacity on link ℓ
$d_\ell(t)$	Current propagation delay on link ℓ
$c_n(t)$	Current CPU core of node n
$\beta(t)$	Current connected base station
$\mathcal{S}$	Set of all available services
$\mathcal{N}$	Set of all selectable nodes
$\lambda(t)$	Current user traffic rate
π_s	Processing delay factor of service s
α_s	Traffic scale factor of service s
$\rho^s_{a,b}(t)$	Migration delay of service s from node a to node b
$\delta^+(r), \delta^-(r)$	Incoming and outgoing links of router r
$x^s_n(t)$	Current placement of service s on node n
$y^{s_i}_\ell(t)$	Usage of link ℓ to route traffic from service s_i to s_{i+1}

timeslot t (clearly, $\rho^{s_i}_{a,a}(t) = 0$). To serve a user request, each service in $\mathcal{S}$ should be deployed in a node with sufficient processing capacity, and the network routing should be adjusted such that user traffic goes through the services in the specified order. Finally, we assume that there is a special service at the end of the chain which is constrained to be placed on the user device. This technique guarantees that the results of the computation is carried back to the user.

4.2 Problem Formulation

In this section, we formally define the problem of SFC orchestration at the edge. Specifically, We consider the problem of service placement, service migration, and traffic routing with the objective of minimizing the user-perceived end-to-end delay.

Placement. To specify the placement of a service chain, we define binary decision variables $x_n^s(t)$, where $x_n^s(t) = 1$ means that service s is placed on the node n in time slot t. We use the following constraint to ensure that every service in the chain is placed on a node,

$$\sum_{n \in \mathcal{N}} x_n^s(t) = 1. \qquad \forall s \in \mathcal{S}, t \in \mathcal{T} \tag{4.2}$$

Recent studies show that co-locating a user's services compromises the system reliability [15]. Thus, we include the following constraint to prevent co-located services,

$$\sum_{s \in \mathcal{S}} x_n^s(t) \leq 1. \qquad \forall n \in \mathcal{N}, t \in \mathcal{T} \tag{4.3}$$

Routing. A single path consisting of intermediate links and routers with enough bandwidth should be provisioned for every consecutive services s_i and s_{i+1} that are placed on separate nodes. To this end, we define binary decision variables $y_\ell^{s_i}(t)$, where $y_\ell^{s_i}(t) = 1$ means that link ℓ is used to route the traffic between hosting nodes of services s_i and s_{i+1}. Additionally, to unify the formulation and present it in a compact manner, we assume that a base station is a router and also assume that a hypothetical router resides on the user hand-held device the connects the device to the corresponding base station. We call this hypothetical router β' and denote the extended set of router by $\mathcal{R}' = \mathcal{R} \cup \{\beta, \beta'\}$. Therefore, we can use following constraint to specify a path,

$$\textstyle\sum_{\ell\in\delta^+(r)} y_\ell^{s_i}(t) - \sum_{\ell\in\delta^-(r)} y_\ell^{s_i}(t) = 0, \tag{4.4}$$

$$\textstyle\sum_{\ell\in\delta^-(n)} y_\ell^{s_i}(t) - \sum_{\ell\in\delta^+(n)} y_\ell^{s_i}(t) = x_n^{s_i} - x_n^{s_{i+1}}, \tag{4.5}$$

where, $\delta^+(r)$ and $\delta^-(r)$ show the incoming and outgoing links of router r,respectively. Furthermore, the following constraint is employed to ensure the capacity of links is respected,

$$\sum_{s_i\in\mathcal{S}} \lambda_{s_{i+1}}(t) y_\ell^{s_i}(t) \leq b_\ell(t). \qquad \forall \ell\in\mathcal{L}, t\in\mathcal{T} \tag{4.6}$$

Delay. The end-to-end chain delay is composed of four fundamental components: (1) *processing delay*, (2) *transmission delay*, (3) *propagation delay*, and (4) *migration delay*. These delays, respectively, are represented by $\Gamma_1(t)$, $\Gamma_2(t)$, $\Gamma_3(t)$, and $\Gamma_4(t)$. The total *processing delay* is computed with regards to the traffic rate assigned to each services on SFC and the CPU cores of the nodes that each service is placed on, expressed as,

$$\Gamma_1(t) = \sum_{s\in\mathcal{S}}\sum_{n\in\mathcal{N}} \frac{x_n^s(t)\pi_s\lambda_s(t)}{c_n(t)}. \tag{4.7}$$

The *transmission delay* is computed with regards to the traffic rate routed to each link on SFC and its bandwidth,

$$\Gamma_2(t) = \sum_{s_i\in\mathcal{S}}\sum_{n\in\mathcal{N}}\sum_{\ell\in\delta^-(r_n)} \frac{y_\ell^{s_i}(t)\lambda_{s_{i+1}}(t)}{b_\ell(t)}. \tag{4.8}$$

The *propagation delay* of each link on SFC is computed as,

$$\Gamma_3(t) = \sum_{s_i\in\mathcal{S}}\sum_{\ell\in\mathcal{L}} y_\ell^{s_{i+1}}(t) d_\ell(t). \tag{4.9}$$

Lastly, notice that when multiple services are migrated in the same timeslot, the transfer happens in parallel and thus the *migration delay* is the maximum time that any of the relocated services need to start its operation. Consequently, the migration delay is computed as,

$$\Gamma_4(t) = max\{\sum_{a,b\in\mathcal{N}} x_a^s(t-1)x_b^s(t)\rho_{a,b}^s(t)\}_{s\in\mathcal{S}}. \tag{4.10}$$

As the objective, the user minimizes its overall weighted delay,

$$\text{Min.} \sum_{t=0}^{\mathcal{T}}\sum_{i=1}^{4}\gamma_i\Gamma_i(t), \tag{4.11}$$

where, γ_i are the scale factors that specifies the relative importance of the different components in the total delays Γ_i.

4.3 Complete Problem Formulation

In an offline setting, given the network topology and delays, demands, user's location at each time slot, our goal is to find a sum minimum delay throughout all time slots while satisfying the constraints for link bandwidth and co-located services prevention. We note that sub-objective 4.11 in the above formulation is non-linear. In this section, we show how to linearize it and formulate the problem above into a binary integer programming (BIP) that can be solved using standard solvers such as Gurobi[30], which allow us to get an offline optimum as a performance benchmark.

The last sub-objective $\Gamma_4(t)$ contains a product of two binary variables: $x_a^s(t-1)$ and $x_b^s(t)$, which can be replaced by introducing a 2-dimension multiplication binary variables $x_{a,b}^s(t)$ and several equivalent linearized constraints, we can rewrite objective 4.11 as follows:

$$
\begin{aligned}
&\Gamma_4(t) = \sum_{s\in\mathcal{S}} \sum_{a,b\in\mathcal{N}} x^s_{a,b}(t)\rho^s_{a,b}(t), \\
&x^s_{a,b}(t) = x^s_a(t-1) * x^s_b(t), \\
&x^s_{a,b}(t) \le x^s_a(t-1), \\
&x^s_{a,b}(t) \le x^s_b(t), \\
&x^s_{a,b}(t) \ge x^s_a(t-1) + x^s_b(t) - 1, \\
&\forall t \in \mathcal{T}, \forall s \in \mathcal{S}, \forall a,b \in \mathcal{N}.
\end{aligned}
\tag{4.12}
$$

Formulation 3 presents the complete problem formulation for the offline placement in the form of a BILP

Formulation 3: Exact offline UECOP

$$
\begin{aligned}
\text{Min.}\quad & \sum_{t=0}^{\mathcal{T}} \sum_{i=1}^{4} \gamma_i \Gamma_i(t) \\
\text{s.t.}\quad & \Gamma_1(t) = \sum_{s\in\mathcal{S}} \sum_{n\in\mathcal{N}} \frac{x^s_n(t)\pi_s\lambda_s(t)}{c_n(t)}. \\
& \Gamma_2(t) = \sum_{s_i\in\mathcal{S}} \sum_{n\in\mathcal{N}} \sum_{\ell\in\delta^-(r_n)} \frac{y^{s_i}_\ell(t)\lambda_{s_{i+1}}(t)}{b_\ell(t)}. \\
& \Gamma_3(t) = \sum_{s_i\in\mathcal{S}} \sum_{\ell\in\mathcal{L}} y^{s_{i+1}}_\ell(t) d_\ell(t). \\
& \Gamma_4(t) = \textstyle\sum_{s\in\mathcal{S}} \sum_{a,b\in\mathcal{N}} x^s_{a,b}(t)\rho^s_{a,b}(t). \\
& \sum_{n\in\mathcal{N}} x^s_n(t) = 1. \qquad \forall s \in \mathcal{S}, t \in \mathcal{T}. \\
& \sum_{s\in\mathcal{S}} x^s_n(t) \le 1. \qquad \forall n \in \mathcal{N}, t \in \mathcal{T}. \\
& \sum_{\ell\in\delta^+(r)} y^{s_i}_\ell(t) - \sum_{\ell\in\delta^-(r)} y^{s_i}_\ell(t) = x^{s_i}_{n_r}(t) - x^{s_{i+1}}_{n_r}(t). \\
& \qquad\qquad\qquad\qquad \forall r \in \mathcal{R}', t \in \mathcal{T}. \\
& \sum_{s_i\in\mathcal{S}} \lambda_{s_{i+1}}(t) y^{s_i}_\ell(t) \le b_\ell(t). \qquad \forall \ell \in \mathcal{L}, t \in \mathcal{T}. \\
& x^s_{a,b}(t) \le x^s_a(t-1), \\
& x^s_{a,b}(t) \le x^s_b(t), \\
& x^s_{a,b}(t) \ge x^s_a(t-1) + x^s_b(t) - 1, \\
& \forall t \in \mathcal{T}, \forall s \in \mathcal{S}, \forall a,b \in \mathcal{N}. \\
& \text{constants} : \gamma, \pi, \lambda, \rho, c, b, \mathcal{T}, \mathcal{N}, \mathcal{S}, \mathcal{L}
\end{aligned}
\tag{4.13}
$$

Chapter 5

Proposed Method

In this chapter, we present the design of `CHANGE`, our proposed bandit-based algorithm for online SFC orchestration at the edge.

We first present the contextual combinatorial bandit formulation [62] of the chain placement problem by characterizing the environment with a set of *arms* (*i.e.* placement options) and specifying their relations in subsection 5.0.1. The formulation reduces the bandit solution's computational complexity and allows the user to handle the uncertainty in the environment and through repeated interactions to find efficient service placement and resource allocation schemes. Specifically, the context allows the user to incorporate the available information into the resource allocation procedure (*e.g.* demand). The combinatorial formulation allows the user to focus on the primitive options (*i.e.* servers and links) during the learning procedure, which significantly reduces the problem space. Otherwise, the user would be forced to consider all the solutions, which are exponential in terms of the number of servers and links (*e.g.* all the different paths in the network). Then, we show how to reduce the uncertainty about the quality of arms and use the result to compute a service efficiently placement by dynamic programming, which is presented in subsection 5.0.2

5.0.1 Bandit Formulation

At the beginning of each timeslot t, user-side information such as their service demands $\lambda(t)$ during the timeslot, the current location $\beta(t)$, and the placement of their service chain in the previous timeslot($i.e.$ $x_n^s(t-1)$) is known. In contrast, the system-level information such as processing and bandwidth capacity of servers and links are not accessible, thereby are required to be learned by the user. Therefore, to learn about the environment, we consider each server and link as an option that can be used for placing a service, which is analogous to the concept of *arm* in bandit theory. The available options in each timeslot are defined to be $O_t \subseteq \mathcal{L} \cup \mathcal{N}$, which is the set of all links, user's device, edge servers, and the remote cloud. At the beginning of each timeslot, based on the information learned so far, the user selects a subset of feasible options for the service chain's placement and minimizes the end-to-end delay. At the end of each timeslot, the user observes each option's contribution to the delay objective. Specifically, for each selected option $o \in O_t$, the user observes some value $\delta_o(t)$. This value is only a noisy estimate of the true value of the option $o \in O_t$. Our goal is to quickly and efficiently estimate the true values of delays incurred by each option. According to our delay formulation, each option contributes *linearly* to the user's perceived end-to-end delay. Following the framework of contextual bandits, we assume that in each timeslot t, the significance of each option depends only on an unknown system parameter vector θ and a contextual feature vector $\boldsymbol{\chi}_o(t)$ that is fully known to the user. For example, the edge server capacities are unknown parameters but the user's current location is a known contextual feature. Thus, the observed value $\delta_o(t)$ is characterized as follows:

$$\delta_o(t) = \theta^\intercal \times \boldsymbol{\chi}_o(t) + \epsilon_o(t), \tag{5.1}$$

where, $\epsilon_o(t)$ is a zero-mean random variable representing the noise delay in the network.

Note that the user can select arm o as a part of service placement resources but only observes the resulting delay $\delta_o(t)$ at the end of the timeslot. Specifically, the user, in each timeslot, using the historical information, should select a subset of options $O_t \subseteq \mathcal{L} \cup \mathcal{N}$ that (1) is feasible for the placement of the service chain and (2) minimizes the overall delay perceived by the user. The user's goal is to minimize the expected cumulative delay in the period of service placement, which is defined as,

$$\mathbb{E}\left[\sum_{t \in \mathcal{T}} \sum_{o \in O_t} \delta_o(t)\right]. \tag{5.2}$$

In the following, we fully describe the system parameter vector θ and contextual feature vector $\boldsymbol{\chi}_o(t)$.

System Parameters. The parameter vector of the system (which is unknown to the user) is composed of four sub-vectors, where each sub-vector contains parameters that are related to a specific type of delay.

1. θ_c is the sub-vector of parameters that determine the processing delay based on the number of CPU cores in each computing node:

$$\theta_c = [\tfrac{1}{c_n}, \quad \forall n \in \mathcal{N}]. \tag{5.3}$$

2. θ_b is the sub-vector of parameters that determine the transmission delay based on the bandwidth of links:

$$\theta_b = [\tfrac{1}{b_\ell}, \quad \forall \ell \in \mathcal{L}]. \tag{5.4}$$

3. θ_d is the sub-vector of parameters that determine the propagation delay:

$$\theta_d = [d_\ell, \quad \forall \ell \in \mathcal{L}]. \tag{5.5}$$

4. θ_ρ is the sub-vector of parameters that determine the service migration delay:

$$\theta_\rho = [\rho^s_{a,b}, \quad \forall a, b \in \mathcal{N}, s \in \mathcal{S}]. \tag{5.6}$$

Finally, it is possible to construct the system parameter vector θ from the concatenation of the aforementioned sub-vectors:

$$\theta = \theta_c \oplus \theta_b \oplus \theta_d \oplus \theta_\rho, \tag{5.7}$$

where, $\oplus$ is the concatenation operator.

Notation. In the following, we use the notation 1_p to represent an indicator function that is equal to 1 when the logical predicate p is true and is equal to 0 otherwise.

Contextual Features. We define a contextual feature vector to represent the known information at the user's side for each placement option $o \in O_t$. Each option is either a link $o = \ell \in \mathcal{L}$ or a computation node $o = n \in \mathcal{N}$. Since the information about links and nodes are different, we define a feature vector for nodes and links separately. The contextual vector, similar to the parameter vector, comprises four sub-vectors that correspond to four types of delay. The link contextual sub-vectors are defined as follows:

1. $\boldsymbol{\chi}^b_\ell(t)$ is the sub-vector corresponding to the transmission delay. $\boldsymbol{\chi}^b_\ell(t)$ contains contextual information about the user's transmission rate.

$$\boldsymbol{\chi}^b_\ell(t) = [1_{\ell'=\ell, \exists n \in \mathcal{N}: \ell' \in \delta^-(n)} \lambda_s(t), \quad \forall \ell' \in \mathcal{L}]. \tag{5.8}$$

   Note that if the user uses link ℓ to transfer the result of computation out of the computation node n, the multiplication of this sub-vector by the system parameter vector computes the expected transmission delay.

2. When the user selects a link, its corresponding propagation delay is incurred. Thus, the contextual propagation delay sub-vector of link ℓ, denoted by $\boldsymbol{\chi}^d_\ell(t)$, is defined as:

$$\boldsymbol{\chi}^d_\ell(t) = [1_{\ell'=\ell}, \quad \forall \ell' \in \mathcal{L}], \tag{5.9}$$

Note that when this sub-vector is multiplied by the corresponding system sub-vector the propagation delay of ℓ is obtained.

3. Since links have no effect on the computational and migration delays, the corresponding sub-vectors, denoted by $\boldsymbol{\chi}_\ell^c(t)$ and $\boldsymbol{\chi}_\ell^\rho(t)$, respectively, are zero vectors:

$$\boldsymbol{\chi}_\ell^c(t) = [0, \quad \forall n \in \mathcal{N}], \tag{5.10}$$

$$\boldsymbol{\chi}_\ell^\rho(t) = [0, \quad \forall a, b \in \mathcal{N}, s' \in \mathcal{S}]. \tag{5.11}$$

Thus, the contextual feature vector of each link ℓ that is going to be used to route the traffic towards service $s \in \mathcal{S}$ is obtained from the concatenation of these sub-vectors:

$$\boldsymbol{\chi}_\ell(t) = \boldsymbol{\chi}_\ell^c(t) \oplus \boldsymbol{\chi}_\ell^b(t) \oplus \boldsymbol{\chi}_\ell^d(t) \oplus \boldsymbol{\chi}_\ell^\rho(t), \tag{5.12}$$

The contextual feature vector of each computation node is defined based on four sub-vectors as follows:

1. The contextual sub-vector of computation features, denoted by $\boldsymbol{\chi}_n^c(t)$, is defined as follows:

$$\boldsymbol{\chi}_n^c(t) = [1_{n'=n} \times \pi_s \lambda_s(t), \quad \forall n' \in \mathcal{N}]. \tag{5.13}$$

Note that if we multiply this sub-vector by the sub-vector of system parameters for computation, the computation delay of running the service s on node n is obtained.

2. The migration contextual sub-vector contains information about the placement of services in the previous timeslot. We denote this sub-vector by $\boldsymbol{\chi}_n^\rho(t)$ and define it as follows:

$$\boldsymbol{\chi}_n^\rho(t) = [1_{s'=s,b=n} x_a^s(t-1), \quad \forall a, b \in \mathcal{N}, s' \in \mathcal{S}]. \tag{5.14}$$

3. Since computation nodes do not affect the transmission and propagation delays, the corresponding contextual feature sub-vectors, denoted by $\boldsymbol{\chi}_n^b(t)$ and $\boldsymbol{\chi}_n^d(t)$, respectively, are zero vectors:

$$\boldsymbol{\chi}_n^b(t) = [0, \quad \forall \ell \in \mathcal{L}], \tag{5.15}$$

$$\boldsymbol{\chi}_n^d(t) = [0, \quad \forall \ell \in \mathcal{L}]. \tag{5.16}$$

Finally, the contextual feature of each node n where service $s \in \mathcal{S}$ is placed is obtained by concatenating the mentioned sub-vectors:

$$\boldsymbol{\chi}_n(t) = \boldsymbol{\chi}_n^c(t) \oplus \boldsymbol{\chi}_n^b(t) \oplus \boldsymbol{\chi}_n^d(t) \oplus \boldsymbol{\chi}_n^\rho(t), \tag{5.17}$$

5.0.2 Placement Algorithm

In this subsection, we present the design of our proposed bandit-based algorithm that efficiently estimates each arm's value, which is equivalent to the expected delay of each placement option. Our proposed bandit-based algorithm for SFC placement on edge is summarized in Algorithm 5, called CHANGE. CHANGE repeatedly examines different placement strategies and adjusts the estimates based on the noisy feedback it receives from the system. Then, CHANGE uses the expected estimated delay values to compute a placement with the minimum end-to-end delay. CHANGE is outlined in Algorithm 5. CHANGE gets the service chain, the contextual features, and available nodes and links as the input. To estimate the expected delay for each placement option, the algorithm has to estimate the system parameter vector θ. We use $\hat{\theta}(t)$ and $\hat{\delta}_o(t)$ to represent the estimated parameter vector and estimated delay for placement option o at timeslot t. To compute these values, we applied the confidence bound-based algorithm in [62].

The algorithm starts with an initial estimate for the covariance matrix $\boldsymbol{V}_0$ and mean vector $\boldsymbol{b}_0$ of system parameters expressed as follows (line 2-3),

$\mathbf{b}_0 \leftarrow \mathbf{0}_d$.

In each timeslot, the estimated system parameter vector is obtained from the covariance matrix and the mean vector. The estimated system parameter vector is then used to compute the excepted delay of each placement option, which is expressed as follows (line 5-6),

$$\hat{\theta}(t) \leftarrow \boldsymbol{V}_{t-1}^{-1}\boldsymbol{b}_{t-1}, \tag{5.18}$$

$$\bar{\delta}_o(t) \leftarrow \hat{\theta}(t)^\intercal \boldsymbol{\chi}_o(t), \forall o \in O_t. \tag{5.19}$$

Then, a lower bound for each delay values in computed as follows (line 7),

$$\hat{\delta}_o(t) \leftarrow \bar{\delta}_o(t) - \sqrt{\boldsymbol{\chi}_o(t)^\intercal \boldsymbol{V}_t^{-1} \boldsymbol{\chi}_o(t)}. \tag{5.20}$$

Then, the algorithm selects a set of placement options P_t at time t based on a placement strategy, which is expressed as follows(line 8),

$$P_t \leftarrow \mathbf{Place}(\hat{\delta}_o(t), \mathcal{S}[t]). \tag{5.21}$$

The placement strategy, based on dynamic programming, is outlined in Algorithm 4.

Finally, the computed placement is implemented and the algorithm observes the delay values incurred as a result. The algorithm the update the covariance matrix and the mean

Algorithm 5: CHANGE: Chain Orchestrator at the Edge

```
1  Input: $\mathcal{N}, \mathcal{L}, \mathcal{S}, \boldsymbol{\chi}$
2  $\boldsymbol{V}_0 \leftarrow \boldsymbol{I}_{d\times d}$
3  $\boldsymbol{b}_0 \leftarrow \boldsymbol{0}_d$
4  for $t \in \mathcal{T}$ do
5      $\hat{\theta}(t) \leftarrow \boldsymbol{V}_{t-1}^{-1}\boldsymbol{b}_{t-1}$
6      $\bar{\delta}_o(t) \leftarrow \hat{\theta}(t)^\intercal \boldsymbol{\chi}_o(t), \forall o \in O_t$
7      $\hat{\delta}_o(t) \leftarrow \bar{\delta}_o(t) - \sqrt{\boldsymbol{\chi}_o(t)^\intercal \boldsymbol{V}_t^{-1}\boldsymbol{\chi}_o(t)}$
8      $P_t \leftarrow \textbf{Place}(\hat{\delta}_o(t), \mathcal{S}[t])$
9      $\boldsymbol{V}_t \leftarrow \boldsymbol{V}_{t-1} + \sum_{o\in P_t} \boldsymbol{\chi}_o(t)\boldsymbol{\chi}_o(t)^\intercal$
10     $\boldsymbol{b}_t \leftarrow \boldsymbol{b}_{t-1} + \sum_{o\in P_t} \delta_o(t)\boldsymbol{\chi}_o(t)$
11 end
```

vector accordingly as follows (line 9-10),

$$\boldsymbol{V}_t \leftarrow \boldsymbol{V}_{t-1} + \sum_{o\in P_t} \boldsymbol{\chi}_o(t)\boldsymbol{\chi}_o(t)^\intercal, \tag{5.22}$$

$$\boldsymbol{b}_t \leftarrow \boldsymbol{b}_{t-1} + \sum_{o\in P_t} \delta_o(t)\boldsymbol{\chi}_o(t). \tag{5.23}$$

Exploration vs Exploitation. In online learning, *Exploration* allows user to improve their knowledge about each arm that is likely to lead to a long-term benefit while *Exploitation* allows user to exploit their current knowledge by choosing the current largest estimated arms. In equation 5.20, $\bar{\delta}_n(t)$ denotes the current estimates at timeslot t and therefore represent the exploitation, while $\sqrt{\boldsymbol{\chi}_n(t)^\intercal \boldsymbol{V}_t^{-1}\boldsymbol{\chi}_n(t)}$ denotes the adjustment made on each arm and therefore represent exploration. Additionally, we associate a constant ratio c to exploration term to determine the exploration-exploitation level, refer to as *exploration ratio. e.g.*, bigger c leads to a smaller lower bound on the estimates and encourages exploration, smaller c leads to a larger lower bound and encourages exploitation. We further evaluate the effect of the exploration ratio c in section 7.2.3.

SFC placement algorithm. The SFC placement algorithm, presented in Algorithm 6 is based on the dynamic programming (DP) principle, which, given the delay estimates $\hat{\delta}_o(t)$ and required service set $\mathcal{S}[t]$, calculates the optimal resource allocation in polynomial time by defining the cost of an optimal solution recursively in a bottom-up fashion. DP finds the optimal solution by starting from placing one single service and successively moving to next service while taking the previous solutions into account. There are three recurrence relations for the minimum cost to place service s_i on node j as explained below (line 5-12):

- s_i is the first service on SFC:

$$\widetilde{D}(s_i, j) = \hat{\delta}_{l_{\beta(t),j}(t)} + \hat{\delta}_j(t) + min\{\hat{\delta}_{l_{j,k}}(t) + \widetilde{D}(s_{i+1}, k)\}_{k \in \mathcal{N}}. \tag{5.24}$$

- s_i is the last service on SFC:

$$\widetilde{D}(s_i, j) = \hat{\delta}_j(t) + \hat{\delta}_{l_{j,\beta(t)}}(t) \tag{5.25}$$

- s_i is neither the first and nor the last service on SFC:

$$\widetilde{D}(s_i, j) = \hat{\delta}_j(t) + min\{\hat{\delta}_{l_{j,k}}(t) + \widetilde{D}(s_{i+1}, k)\}_{k \in \mathcal{N}}. \tag{5.26}$$

The optimal placement of a SFC is then determined by the node with the minimum cost to place the first service, that is:

$$\text{Cost(SFC)} = min\{\text{Cost}(1, n)\}_{n \in \mathcal{N}}. \tag{5.27}$$

The solution is obtained by extracting the host list associate with it, that is:

$$P_t \leftarrow H(1, n) \tag{5.28}$$

With probability $1 - \epsilon$, the regret of the proposed algorithm is less than

$$O(Slog(n)\sqrt{T} + \sqrt{TSlog(T/\epsilon)}), \tag{5.29}$$

Algorithm 6: Place($\hat{\delta}_o(t)$, $\mathcal{S}[t]$)

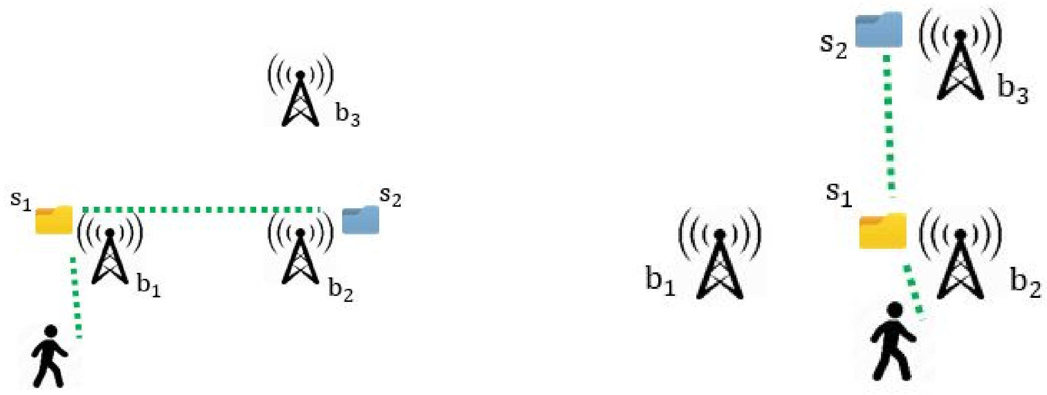
```
1  $H(i,j) \leftarrow \mathbf{0} \quad \forall i \in \mathcal{S}[t], \forall j \in \mathcal{N}$
2  $\widetilde{D}(s_i,j) \leftarrow 0 \quad \forall i \in \mathcal{S}[t], \forall j \in \mathcal{N}$
3  for $i \in S..1$ do
4      foreach $j \in \mathcal{N}$ do
5          if $i = 1$ then
6              $\widetilde{D}(s_i,j) = \hat{\delta}_{l_{\beta(t),j}(t)} + \hat{\delta}_j(t) + min\{\hat{\delta}_{l_{j,k}}(t) + \widetilde{D}(s_{i+1},k)\}_{k \in \mathcal{N}}$.
7          else
8              if $i = S$ then
9                  $\widetilde{D}(s_i,j) = \hat{\delta}_j(t) + \hat{\delta}_{l_{j,\beta(t)}}(t)$
10             else
11                 $\widetilde{D}(s_i,j) = \hat{\delta}_j(t) + min\{\hat{\delta}_{l_{j,k}}(t) + \widetilde{D}(s_{i+1},k)\}_{k \in \mathcal{N}}$.
12             end
13         end
14         $H(i,j) \leftarrow H(i-1,k).append(k)$
15     end
16 end
17 Extract the node with the minimum cost to place $s_1$: $n \leftarrow \text{argmin}\{\widetilde{D}(s_1,j)\}_{j \in \mathcal{N}}$
18 Get the corresponding host list as the solution $P_t \leftarrow H(1,n)$
19 Return $P_t$
```

where, S shows the length of the chain and T is the lifetime of the chain (*i.e.* total timeslot). For detailed analysis refer to [62].

Theorem 1. *With combinatorial bandit formulation, the decision space of the user is reduced by the following factor compared to a general bandit formulation:*

$$\frac{n+l}{\sum_{k=0,\dots,s} \frac{n!s!}{k!(s-k)!(n-s+k)!}} \tag{5.30}$$

Proof. A general Combinatorial Multi-armed Bandit (CMAB) algorithm selects the arm (a set of underlying arms) of round t to play based on the outcomes of revealed arms selected in previous rounds, without knowing the expectation vector μ of all individual arms [16]. We denote S_t to be the super arm selected in round t, which can be a random super arm chosen depending on the outcomes of arms in previous rounds and potential randomness in the



(a) Service placement at T_1 (b) Service placement at T_2

Figure 5.1: An illustrative example used for CMAB formulation

algorithm itself. The objective of CMAB is to maximize the expected reward of all rounds up to round n, that is, $\mathrm{Min}(\mathbb{E}_S[\sum_{t\in\mathcal{T}} r(S_t)])$, where $\mathbb{E}_S$ denotes taking expectation only among all random events generating the super arm S_t's, $r(S_t)$ is a non-negative variable denoting the reward of round t when super arm S is played. In order to formulate the UECOP into a general CMAB problem, the size of the arm set that the user can choose can be analyzed as follow, during each timeslot, the user is presented with the choice of selecting a set of nodes from mobile edge or cloud to deploy SFC and run applications. Therefore we define a set of nodes as a super arm. For example, we consider n edge nodes, l links, one cloud in the MEC network, and s service requested. Since in the core network, the computing resources are considered to be limitless and are capable of running all the services, we assume that there are at least s nodes to choose from in the cloud, by combinations the total decision space can be calculated as the bottom part of eq.5.30. With our proposed CMAB formulation, the decision space is reduced to $n + l$ because we view individual links and nodes as candidates to select. $\square$

5.0.3 Bandit Formulation Example

Assume there are 3 base stations as shown in Fig. 5.1. At time T_1, the user is in the vicinity of b_1. The first service s_1 is located in the edge server at b_1 and the second service s_2 is located in the edge server at b_2. Assume that after some time the user moves to the neighborhood of the base station b_2 at time T_2. As a result, he changes his service placement as shown in the Fig. 5.1b. To get this placement, the user has selected the nodes at the base stations b_2 and b_3 (n_1 and n_2) as well as the links between himself and the base station b_2 and between base stations b_2 and b_3, which are referred to by ℓ_1 and ℓ_2. The unknown system parameter is expressed as follows:

$$\theta_c^{\intercal} = [\frac{1}{c_1}, \frac{1}{c_2}, \frac{1}{c_3}]$$

$$\theta_b^{\intercal} = [\frac{1}{b_{\ell_1}}, \frac{1}{b_{\ell_2}}, \frac{1}{b_{\ell_3}}, \frac{1}{b_{\ell_4}}]$$

$$\theta_d^{\intercal} = [d_{\ell_1}, d_{\ell_2}, d_{\ell_3}, d_{\ell_4}]$$

$$\theta_\rho^{\intercal} = [\rho^{s1}_{e1,e_2}, \rho^{s1}_{e1,e_3}, \rho^{s1}_{e_2,e_3}, \rho^{s2}_{e1,e_2}, \rho^{s2}_{e1,e_3}, \rho^{s2}_{e_2,e_3}]$$

$$\theta^{\intercal} = \theta_c^{\intercal} \oplus \theta_b^{\intercal} \oplus \theta_d^{\intercal} \oplus \theta_\rho^{\intercal}$$

However, as the user does not know this parameter vector, starts with an estimate that is obtained by equations (**??**), (5.0.2), and (5.18)-(5.20). Then, the user assumes $\hat{\theta}(t)$ is the correct parameter vector and use it to construct proper contextual feature vectors $\boldsymbol{\chi}_{b_2}$ to compute the expected delays. Then, the user uses the computed expected delays to find the best chain placement. For example, the user, in order to evaluate the expected delay of the base station b_2 for placement of his first service s_1, uses the following contextual feature vector:

$$\boldsymbol{\chi}_{b_2}^{c} = [0, \pi_{s_2}\lambda_{s_1}, 0]$$

$$\boldsymbol{\chi}_{b_2}^{b} = [0, 0, 0, 0]$$

$$\boldsymbol{\chi}_{b_2}^{d} = [0, 0, 0, 0]$$

$$\boldsymbol{\chi}_{b_2}^{\rho} = [1, 0, 0, 0, 0, 0]$$

$$\boldsymbol{\chi}_{b_2} = \boldsymbol{\chi}_{b_2}^{c} \oplus \boldsymbol{\chi}_{b_2}^{b} \oplus \boldsymbol{\chi}_{b_2}^{d} \oplus \boldsymbol{\chi}_{b_2}^{\rho}$$

The user, then, uses the employed feature vectors and observed delay values to update the vectors $\boldsymbol{V}_t$ and $\boldsymbol{b}_t$ by using equation (5.22) and (5.23).

5.0.4 Delay Estimation and Deployment Considerations

Algorithm 5 needs to know the contribution of each server and each link in the solution O_t in the end-to-end delay (*i.e.* $\{\delta_n(t)\}$). There are multiple hardware and software-based solutions to obtain these delays. Designing another mechanism for this problem is not the focus of this work. However, we briefly explain the viability of obtaining these estimates by employing current studies. A simple and lightweight approach is using time-stamped packets, where a special packet is sent along the same path that carries the service traffic. Each server adds a timestamp that specifies the packet's arrival time and another timestamp that specifies a departure time, which is set to be after the mean processing time of the service packets. A simple daemon that is deployed on servers can accomplish this task. In order to compute the mean service time for each service more precisely, it is possible to employ the approach that is presented in [28]. Further, it is possible to use available tools (*e.g.* [74]) to achieve the desired level of time synchronization that is enough for the accuracy of timestamped-based calculations. However, it is possible to employ the ideas from the network tomography, which is deeply studied, to avoid network synchronization complications and overheads. With approaches like [81], it is possible to obtain link-level delay estimates from end-to-end measurements. The processing delays can be obtained with the same approach as before, and the servers can directly send their measurements to the user.

In chapter 6, we present our design of a simple timestamp-based estimation framework.

Chapter 6

Delay Estimation framework

As already mentioned before, to decide on a solution (super arm), we need first to gain estimation for each individual's arms. In our problem, we assume that by selecting a super arm, we are able to get an estimation of every single arm in it (i.e: computing delay and link delay) and update the estimated values accordingly. In this chapter, we present our design of a time-stamped based estimation framework to measure the delay in the network, which we implement and apply for each Mininet-WiFi experiment conducted in chapter 8.

6.1 Time stamp in SFC placement

Now we present our proposed time-stamp based estimation framework in SFC placement to measure the processing delays on each computing node and the transmission delay on each link. To estimate the delays in the network, we time-stamp[] one packet and tag it along with the packets we send for each SFC request while keeping track of the time. The estimation is consist of two parts:

1) **An initial estimation of the whole network.** The user first sends a request to each computing node. Upon receiving the request, the server will each send packets to the other servers and time stamp them. After receiving all the returned packets, the server parses the timestamps in them, calculates the delays, and sends that information back to the user. The user collects all the delay estimations from the servers and saves them as a initial estimation required in the CMAB algorithm.

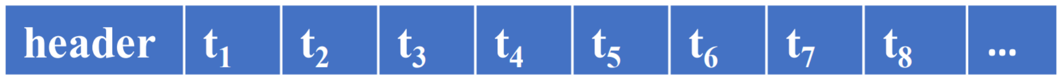


Figure 6.1: Estimation packet format

2) **Estimation of delays on the nodes and links that have been selected for placeing SFCs.** Like the initial estimation, we also timestamp the packets we send in each SFC request. After the packet is returned from the SFC, the user parses the timestamps and calculate the delays on the SFC. Those estimations will then be used to update the initial estimation according to algorithm 5. We consider a SFC with three different services in this section as an example. The packet format of the packet that's used for estimation is shown in figure 6.1. Each server on the path appends the timestamps to the packet as the packet traverses. Depending on the length of SFC, say l, each estimation packet would occupy $(l + 1) * 18$ bytes of payload to store all the timestamps because each time stamps takes 18 bytes to store in our implementation.

6.2 Time-stamped delay estimation example

We consider a scenario that the VNFs are placed on node A, B, C at the moment and that the user is connected to node D, as shown in figure 6.2. The SFC request from a mobile user will be time-stamped and saved as a vector as we defined before, as the request is handled along the chain, each processing node will time stamp the packet when: 1) When it receives a packet, 2) After it is done processing. The probing process is described as follow:

1) User generates SFC request packets with size λ_i bytes for each service i, while inserting the current time to t_1 in the last packet.

2) Node A receives the request first in the SFC. Upon receiving the last packet, it updates the packet by inserting its current time to t_2 and starts processing the packets. When the processing is done, node A inserts the current time t_3 to the same packet. Then it sets the destination to node B as the next service handler and forwards all the packets.

3) Node B process the second requested task and inserts timestamps t_4, t_5 before and afterward the task, respectively. Then it sets the destination to C and forwards the packets.

4) Node C processes the last request. It repeats the time-stamping (t_6, t_7) and sends a SFC response along with a packet that stores all the timestamps in order.

5) the user will then get a response for their services and a list of timestamps while recording the current time t_8 upon receiving. Based on the timestamps and SFC packet sizes, the user is able to calculate the corresponding link delay and processing delay.

When the time stamps is returned to user, they can analyze the packet and calculate the delays on the SFC path as follow:

$$\begin{aligned}
LinkDelay_{D\to A} &= t_2 - t_1 \\
LinkDelay_{A\to B} &= t_4 - t_3 \\
LinkDelay_{B\to C} &= t_6 - t_5 \\
LinkDelay_{C\to D} &= t_8 - t_7 \\
ProcessingDelay_A &= (t_3 - t_2)/\lambda_1 \\
ProcessingDelay_B &= (t_5 - t_4)/\lambda_2 \\
ProcessingDelay_C &= (t_7 - t_6)/\lambda_3
\end{aligned} \tag{6.1}$$

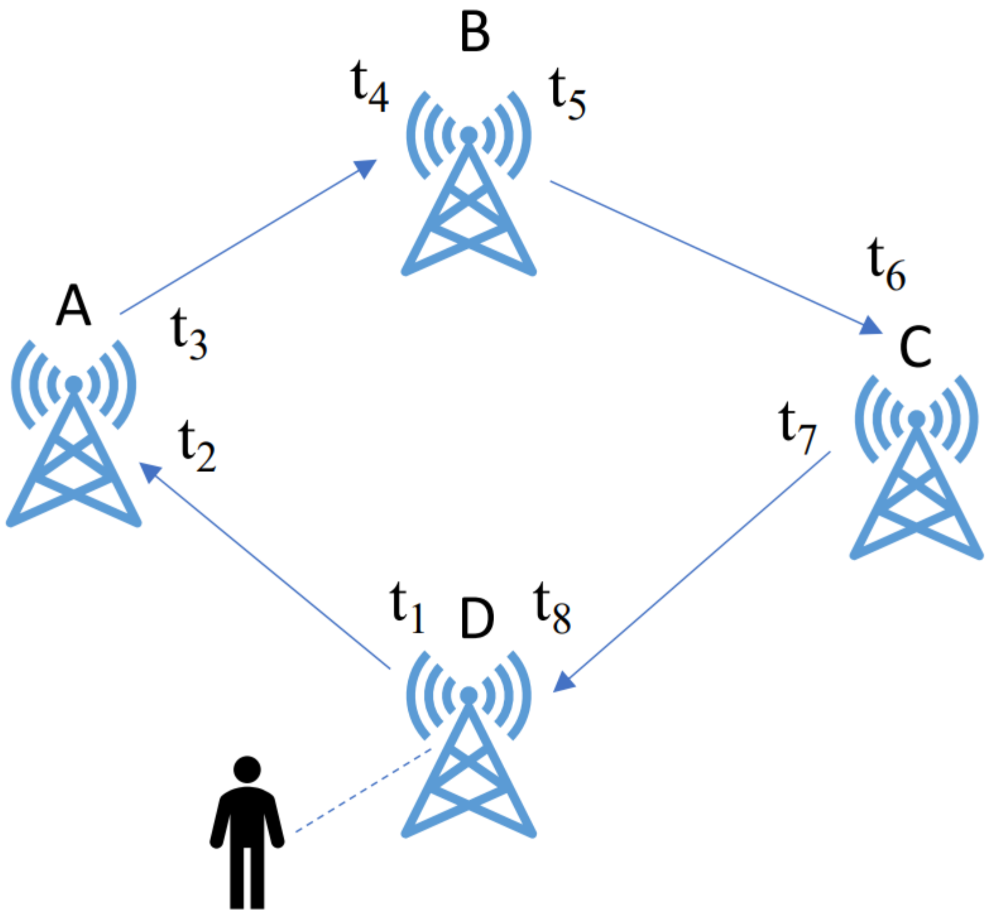


Figure 6.2: Time stamps of SFC placement in MEC

Chapter 7

Simulation Results

In this chapter, we conduct extensive simulations to demonstrate the performance of our proposed online learning algorithm CHANGE for user-managed SFC placement on edge, in terms of the convergence performance, scalability, and relative performance against other algorithms under varying parameters. All algorithms are implemented in Python 2.7 and run on an Intel Core i5-7400CPU@3.00GHz machine with 8.00 GB RAM. We utilize the Python interface of Networkx [21] to simulate customized network and Gurobi [30] to solve the optimal offline optimizations.

7.1 Simulation Settings

7.1.1 Network Settings

We conduct experiments on a simulated 2km $\times$ 2km grid network area with 25 edge nodes and a cloud node, as shown in figure 7.1, this setting is in line with [60]. Cloud and each edge server is equipped with computation ability, The ratio of processing delay factors π_s to node capacities is set appropriately to achieve a processing delay of [0.5, 1] and 0.1 milliseconds per Kbits of data flow for each edge computing node and 0.1 milliseconds per Kbits of data flow for the cloud. The link delay of user-to-edge and edge-to-cloud links are uniformly distributed in [10, 50] milliseconds and [100, 200] milliseconds, respectively. The migration delay is proportional to the link delays and thereby is distributed in [10, 200] milliseconds

Table 7.1: Simulation parameters

Parameter	Value
Number of nodes	26 (25 edge, 1 cloud)
Edge VNF proc. delay (ms/kbit)	0.5 - 1 (uniform)
Cloud VNF proc. delay (ms/kbit)	0.1
Number of links	65
Link delay (ms) on edge	10 - 50 (uniform)
Link delay (ms) edge2cloud	100 - 200 (uniform)
Migration delay (ms)	10 - 200 (uniform)
VNFs per SFC requests	3 - 5
VNF request (bits)	[250, 2500, 10000]
Mobility model	Random Waypoint

depending on the link that the VNF is migrated through.

We set these values based on [67] so that the environment can provide the conventional end-to-end delays for existing applications (*e.g.* web service 500ms, video streaming 100ms, and online gaming 60ms).

7.1.2 SFC Settings

According to [43], we consider SFC with 3 to 5 different service VNF request. For each request in the SFC, we roughly pick a random choice among three categories: heavy (10000 bytes), medium (2500 bytes), and light (250 bytes) at each timeslot, with regards to the requirements of common service chains such as web services and video streaming, which is summarized in table 7.2.

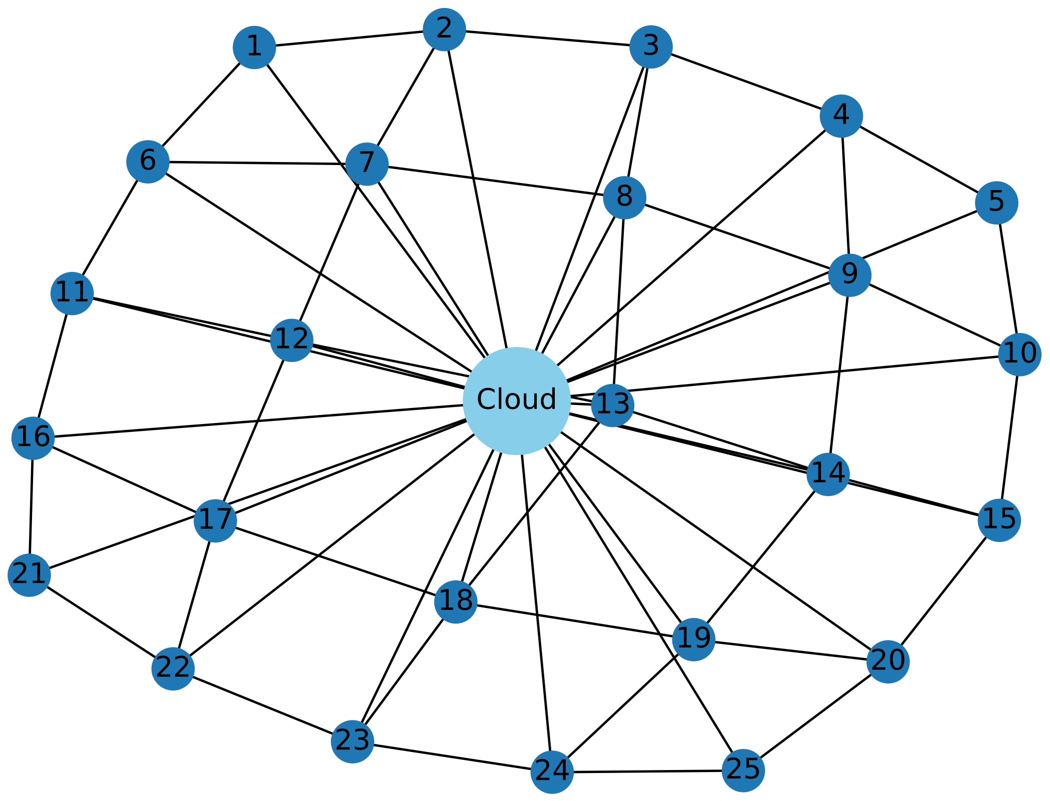


Figure 7.1: Network topology used in simulation

Table 7.2: Requirements for common SFCs

Service Chain	Chained VNFs	Requested Bandwidth	Maximum Tolerated Latency
Web Service	NAT-FW-TM-WOC-IDPS	100 kbit/s	500ms
VoIP	NAT-FW-TM-FW-NAT	64 kbit/s	100ms
Video Streaming	NAT-FW-TM-VOC-IDPS	4 Mbit/s	100ms
Online Gaming	NAT-FW-VOC-WOC-IDPS	50 kbit/s	60ms

NAT: *Network Address Translator*, FW: *Firewall*, TM: *Traffic Monitor*, WOC: *WAN Optimization Controller*, IDPS: *Intrusion Detection Prevention System*, VOC: *Video Optimization Controller*

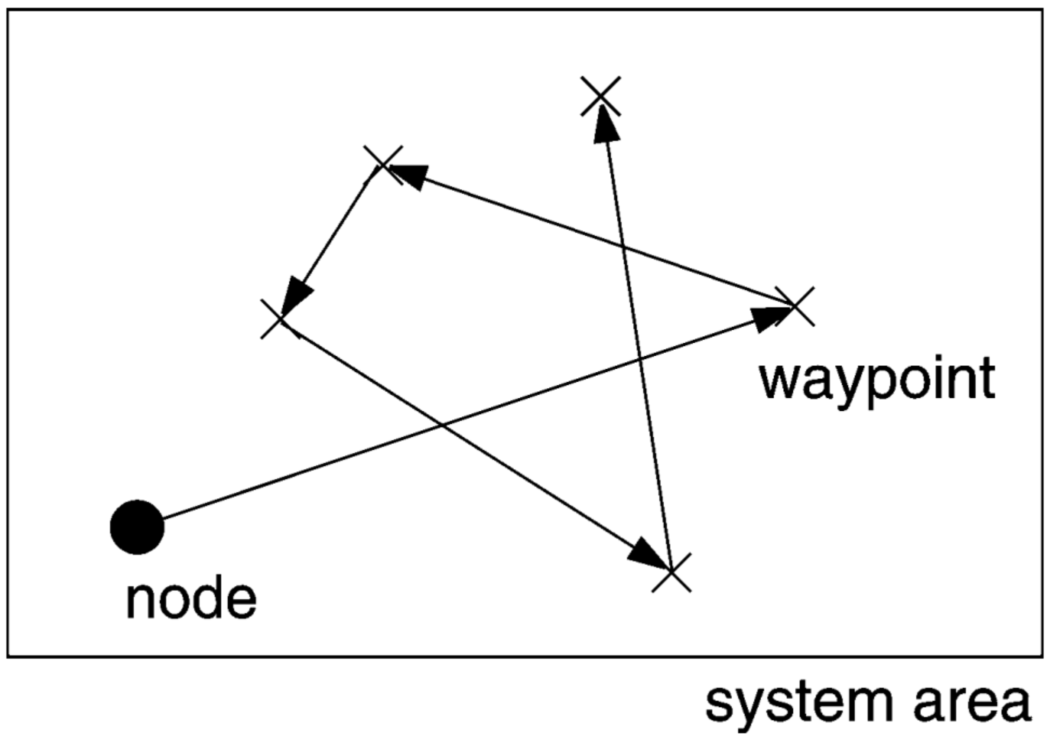


Figure 7.2: Random Waypoint mobility model

7.1.3 User mobility Settings

To simulate the user's mobility, we use *Random Waypoint* mobility model because of its simplicity and wide availability. The random waypoint model has been commonly used as a mobility model in wireless network simulations [7]. In this model, the user will pause for a fixed number of seconds and then choose a random destination within the network area and a random speed between 1 m/s and 1.5 m/s. The user moves to this destination and again pauses for a period before another random location and speed, as depicted in figure 7.2. Throughout the movement, the user is automatically connected to the closest radio base station for communication.

7.1.4 Performance Benchmark.

To evaluate the performance of our algorithm, we compare it to the overall offline optimum that's obtained by solving the optimization formulated in section 4.2 using *gurobi*[30], and the following two edge/cloud-only derivatives of CHANGE:

- **Serving on the cloud(SC):** the mobile user would always put the services on the remote cloud to run in order to maximize the computation delay
- **Serving on the edge(SE):** the mobile user would always put the services on the edge servers to run in order to avoid high transmission latency to the cloud.

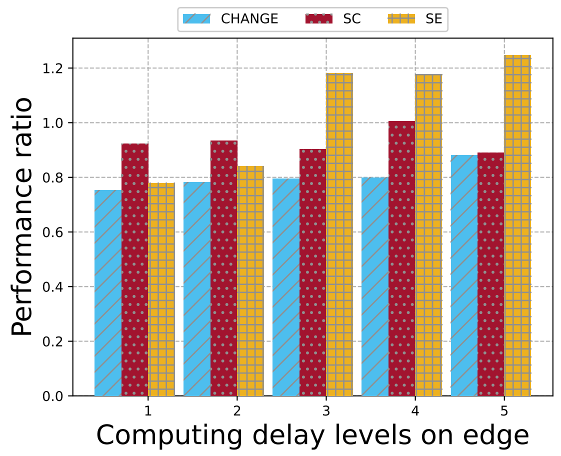
To show the strength of CHANGE in terms of scalability, we compare it with the following two greedy learning algorithms:

- **ϵ-greedy.** with a probability of ϵ, randomly choose a super arm, otherwise choose the arm with a maximum average reward
- **Adaptive greedy.** Extension of ϵ-greedy, in which ϵ is decreasing by time to avoid over-exploring.

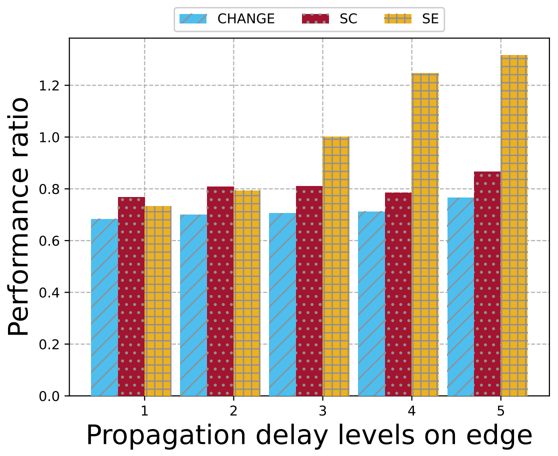
The parameter used in the simulation is summarized in Table 7.1. Nevertheless, all numeric results are normalized with an appropriate maximum or minimum value from each experiment.

7.1.5 Impact of network delays

We considered five levels of computation and transmission delays to investigate their effect on the service quality. Figure 7.3a and figure 7.3b show that CHANGE achieves about $15-20\%$



(a) Multiple algorithms under different computation settings on edge



(b) Multiple algorithms under different transmission settings on edge

Figure 7.3: Effect of network delays

reduction in end-to-end delay compared with other methods. For low delays, SE achieves a comparable result because SFCs can be served on the edge with no switching delay due to migration from and to the cloud. SC becomes more competitive when the delays are higher because the computation power of the cloud becomes more significant. Nevertheless, CHANGE achieves a proper balance between the resources located at the edge and the cloud.

7.1.6 Impact of Migration Delay

Next, we analyze how the switching cost in the network settings affect the service performance. We design five weights of the switching cost to represent the relative importance of migration delay on performance cost while the other setting (*i.e.*, computation and transmission delays) remains the same. As shown in figure 7.4, the relative performance of all three approaches is similar to that seen in the experiments of delays, which shows that CHANGE outperform SE and SC in most instances by an overall 20% improvement. We notice that when the switching weight is high enough (e.g .4.0 or 5.0), CHANGE has a very similar perfor-

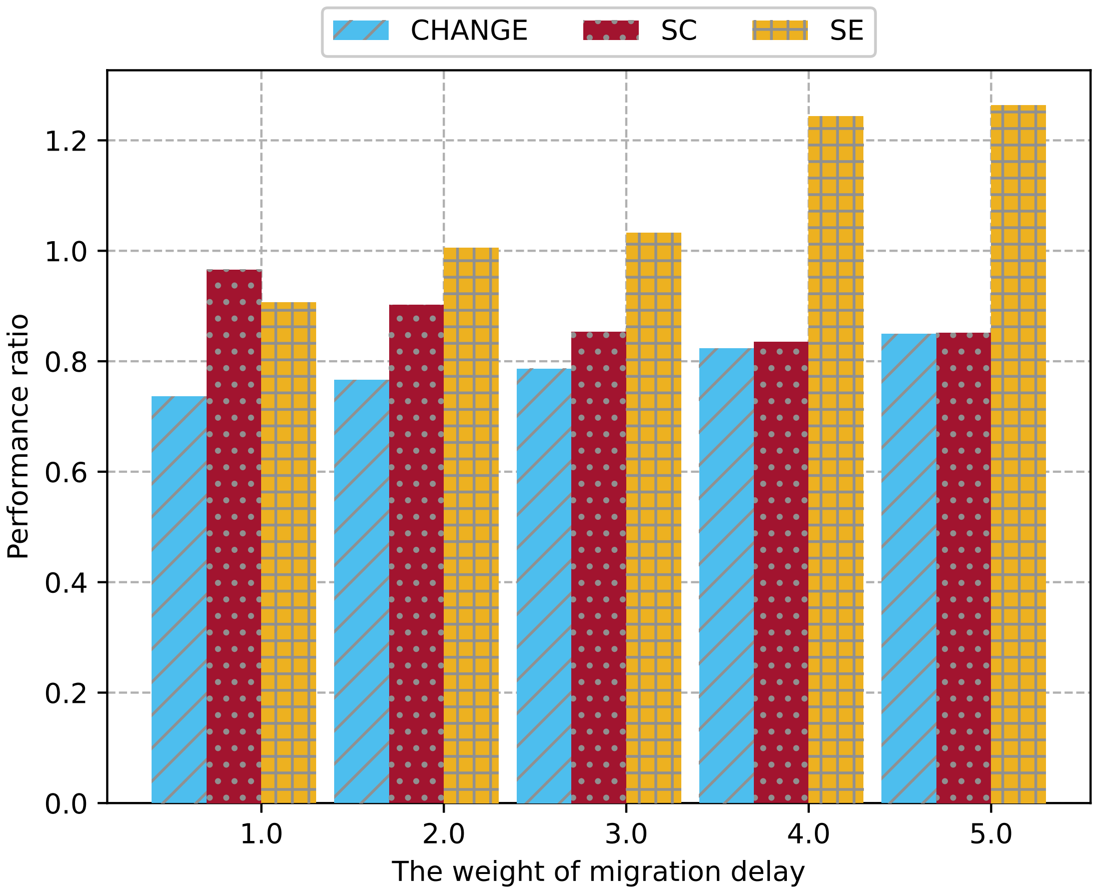


Figure 7.4: Multiple algorithms under different weights of switching cost

mance as SC. This is because user prefer to put its services on the same computation node due to high service migration cost and cloud has the most computation capacity.

Overall, figure 7.3 and figure 7.4 shows that `CHANGE` is able to jointly consider different types of delay and optimize the performance with regards to them by jointly utilizing the edge and cloud.

7.2 Convergence performance

7.2.1 Comparison with an exact offline approach

To evaluate the convergence performance of our proposed algorithm, We first trace the average cost of SFC at each timeslot and compare it with a *offline* optimum, which is obtained by solving the exact offline optimization constructed in formulation 3. Specifically, the offline optimum is calculated after each run of the online algorithm given the whole problem data from the beginning to the end of the learning slots, such as user's demand, mobility, and network capacity on the edge cloud of each online learning experiment.

Figure 7.5 shows that during the learning process, our proposed SFC placement algorithm using `CHANGE` gets a decreasing average cost with the time increasing and is gradually converging to a fixed value that is close to the optimum after about 500 time slots. The downtrend indicates that `CHANGE` is able to learn the system dynamics and make near-optimal decisions.

7.2.2 Time average regret

We define the "regret" as the difference between the cumulative estimate cost of the predicted optimal solution at each round calculated using the delay estimates and the actual cost received after playing the predicted solution, denoted by $\hat{c}_t$ and c_t respectively. We can formally define the regret as:

$$R_T = \sum_{t=0}^{T} abs(c_t - \hat{c}_t) \tag{7.1}$$

From figure 7.5 We can also see a similar trend as the time average cost in total regret during the learning slots, that it increases sharply with learning slots due to the system uncertainty

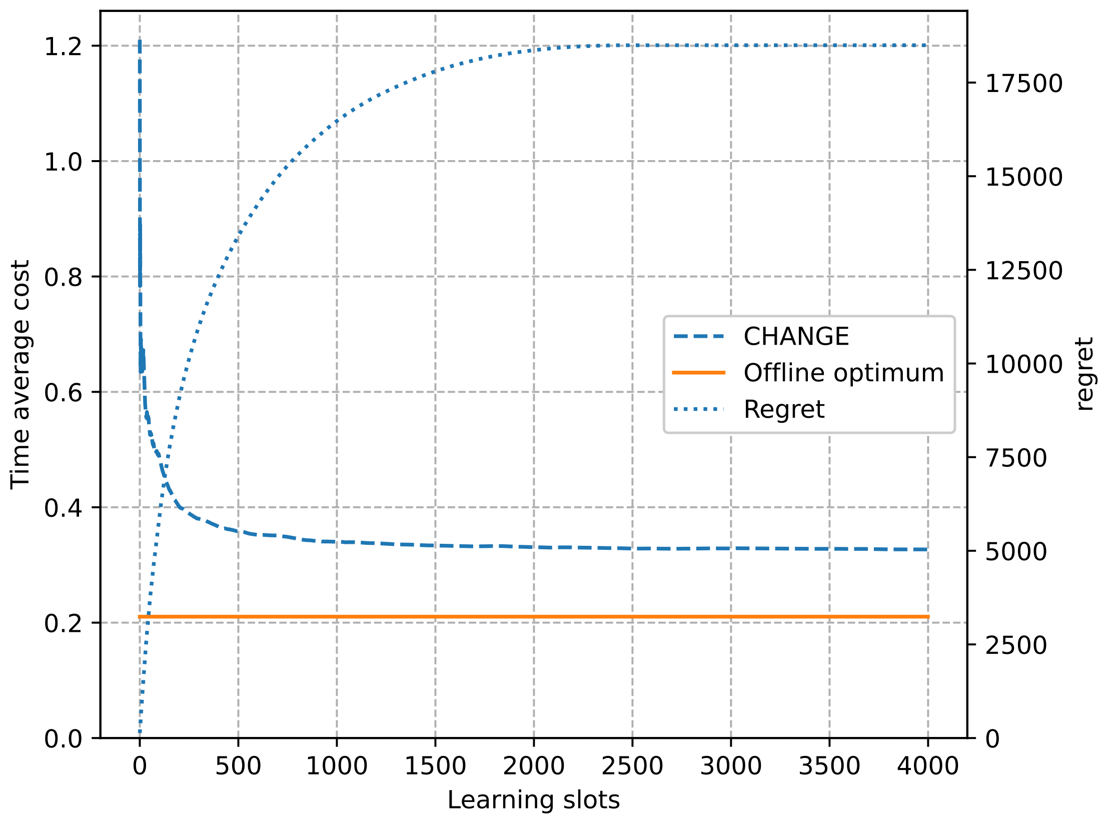


Figure 7.5: Convergence performance: 1) CHANGE approaching to the offline optimum 2) Algorithm converges around 2000

and randomness in the decision making, and gradually converges to a constant value when the time slot is around 2000 times.

7.2.3 Exploration ratio

As mentioned in section5.0.2, exploration level is determined by the ratio c in equation 5.20 when calculating the lower confidence bound for each arm. Figure 7.6 shows how different exploration ratio c influence the total regret of our CHANGE algorithm. It is observed that a smaller c (e.g, $c = 0.05$) may lead to a lack of exploration, and a bigger c (e.g. $c = 1.0$) may lead to over-exploring. We can also see that when $c = 0.3$, the regret converges the fastest. Thus, it can be used as a fixed value for other experiments.

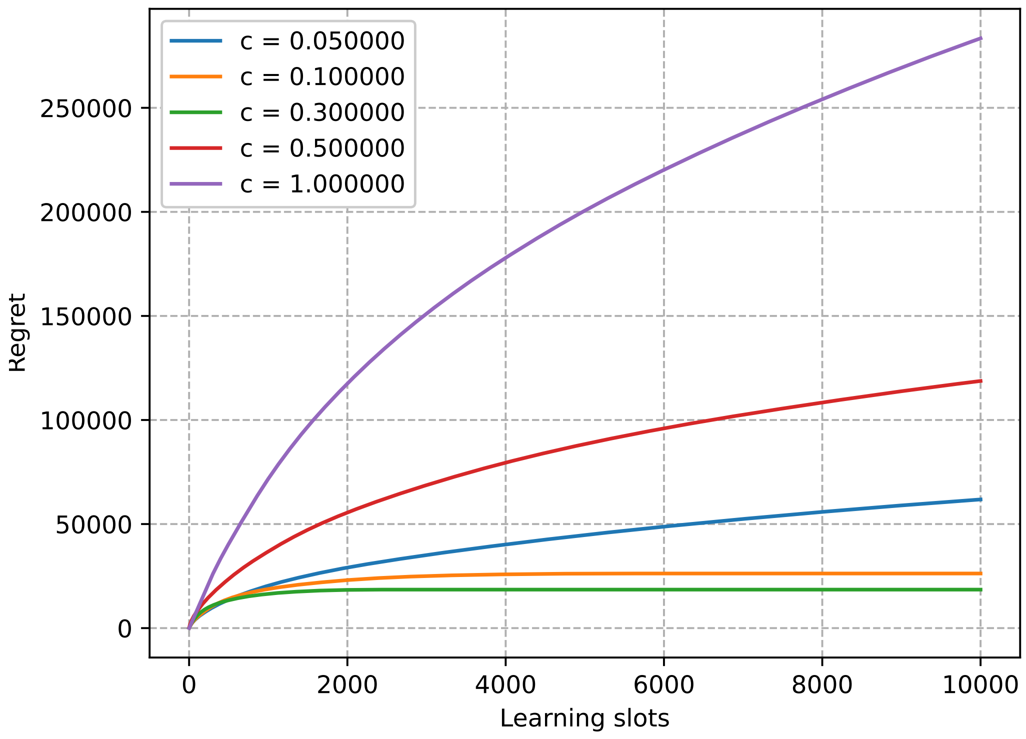


Figure 7.6: Total regret with different exploration level

7.2.4 Scalability Analysis

In this section we analyze how CHANGE is affected by the scale of the problem and compare its performance with the two aforementioned greedy online approaches.

7.2.5 Running Time Analysis

In this experiment, we gradually increased the problem size and computed the running time and cost.

We increase the network size from 3x3 to 6x6 nodes and specify the number of services on SFC from 3 to 10, then run CHANGE for 1000 learning slots and calculate the running time. The results are shown in figure 7.7, as expected from the complexity of algorithm 5, the running time is dominated by the number of available edge servers. Considering that there are 1000 times in this simulation, the average per-time-slot running time is less than 0.3 seconds, which is acceptable for any practical usages.

7.2.6 Comparison with the greedy approaches.

In this experiment, we further compare the cost of CHANGE with two greedy approaches to show our advantages on performance in terms of problem complexity. As it may be apparent, the difficulty of solving SFC orchestration in our problem depends on the number of services on SFC and the size of the network. Thus, we change these two parameters and see how it affects our algorithm and the greedy algorithms. As shown in figure 7.8, with the complexity of the problem increases, the performance of CHANGE is much less influenced than both greedy algorithms, which shows that our proposed algorithm has better scalability compared to the other two algorithms.

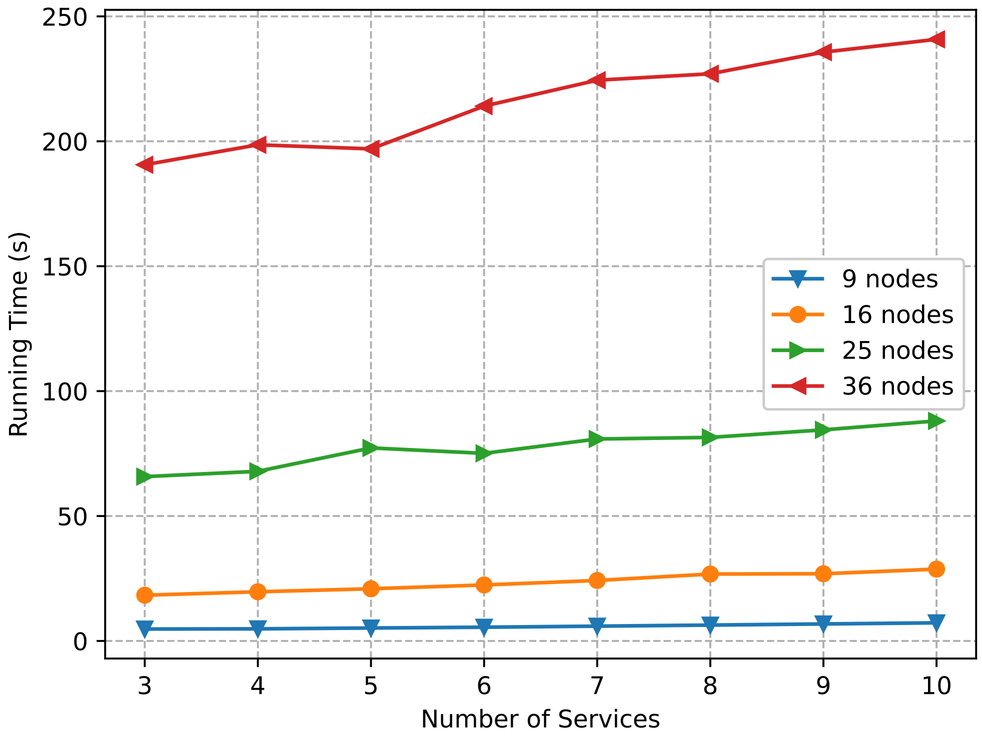
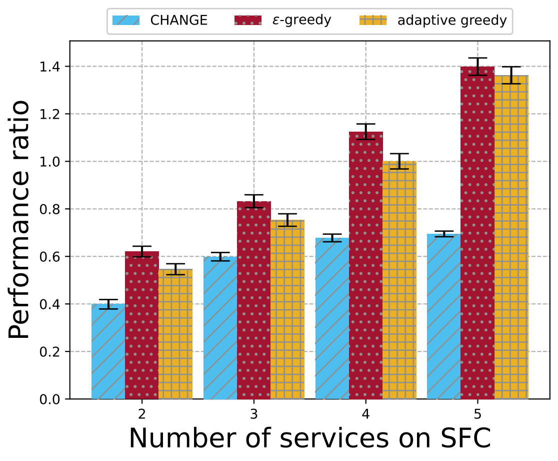
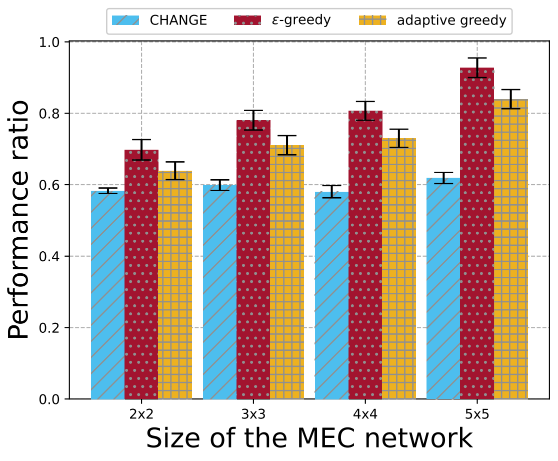


Figure 7.7: Run time analysis



(a) Number of services on SFC



(b) Size of the MEC network

Figure 7.8: Performance analysis under different problem scales: a) Number of services on SFC; b) Size of the MEC network

Chapter 8

Mininet-WiFi Experiments

This chapter describes the Mininet-WiFi experiment we conducted to validate the performance of our proposed user-manged edge-enabled SFC placement in practice. The MEC emulation is implemented as an extension to Mininet-WiFi[23], Mininet-WiFi is a tool that allows researchers to emulate a wireless network environment using SDN technologies. It has supports for the wireless station, Access points, and mobility model. We started by describing the simulation setup and parameters in section 8.1. Then we present the results from the experiments we did in Mininet-WiFi including the learning behavior of CHANGE in Mininet-WiFi (section 8.2) and comparison with greedy algorithms under different environmental parameters (section 8.2.2, 8.2.3, 8.2.4).

8.1 Experiment setup.

In our Mininet-WiFi emulation, we create an edge-enabled network with nine wireless base stations that are scattered in a 100×100 squared meter area, as shown in figure 8.1. Each base station is connected to a server that provides computation services. The user, who is connected to the closet base station automatically, moves with a constant speed that is selected randomly from the interval [1, 5] meters per second. We use the log-distance propagation loss model for wireless connections. We use sockets that run on the edge servers to emulate the computation delay. The user repeatedly sends packets to the closest base station, where the packet is then forwarded to traverse the services in the chain. The last

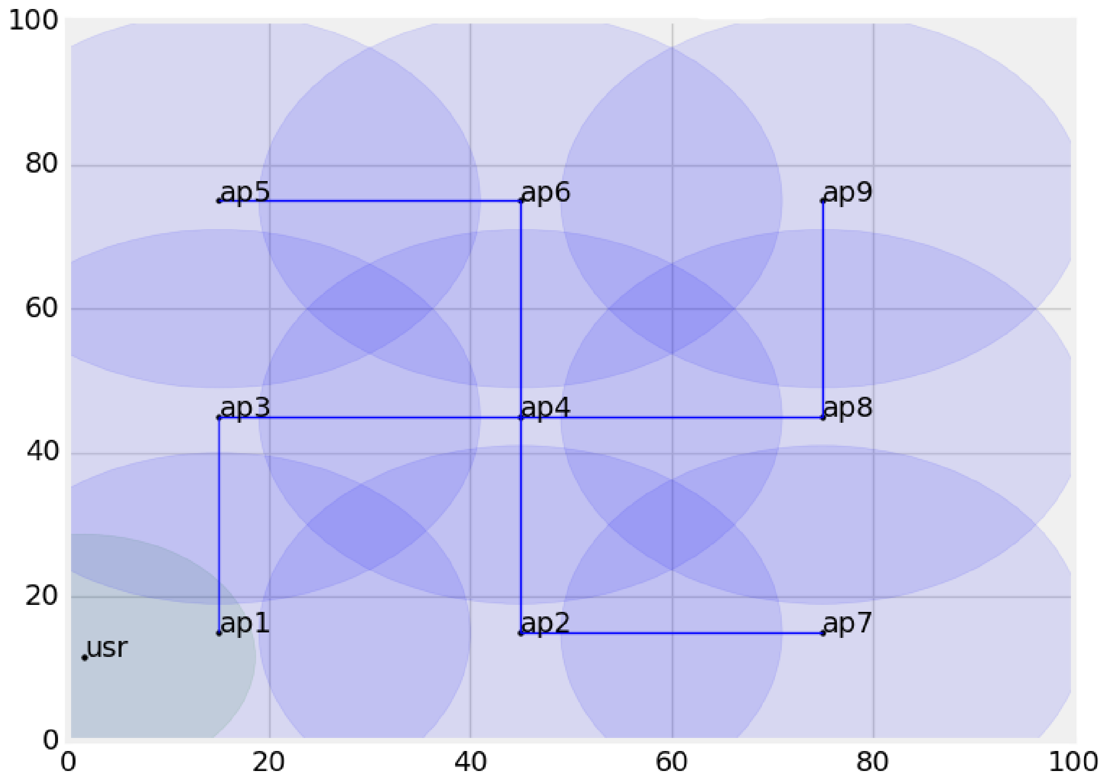


Figure 8.1: Mininet-WiFi Emulation

service in the chain sends a response back that allows the user service in the chain sends a response back that allows the user to collect the end-to-end delay (by time-stamping the packets). The Mininet-WiFi environment is set up in a VM running Ubuntu 16.04.2 LTS with 2 cpu and 4GB RAM. The VM is deployed on a windows 10 machine with i5-7400 and 8GB of RAM.

The parameters used in Mininet-WiFi simulation are summarized as follow:

- **Duration** —Each experiments ran for 30-60 minutes
- **Area** —the spatial area has a dimension 100 $\times$ 100 meters
- **Mobile stations** —there is one mobile station representing the mobile user with a mobility model

- **Packet generation rate** —25-100 packets per time slot.
- **Packet size** —fixed to 1 byte.
- **Access Points** —9 APs with fixed positions show in figure 8.1 with a connection range of 20 meters, which makes 99 % of the simulation area covered by WiFi, with each of them connected to an edge server.
- **Mobility model** —five different mobility models analyzed in section 8.2.2, with each configured with a maxmum speed of 5 meters/second and minimum speed of 1 meters/second.
- **Handover policies** —Active handover, a client station switches WiFi connections to an AP as soon as it finds another connection with closer range, this means that the client is always connected to the closet AP, which is the same as Networx simulation.
- **Propagation model** —The Log-Distance Propagation Model.

8.2 CHANGE performance in Mininet-WiFi.

In this experiment, we perform the actual placement of the SFCs on an emulated Mininet-WiFi network topology, as shown in figure 16. Specifically, we first run the time-stamp estimation on the whole network and gain an initial estimation. Based on the initial estimation, we apply CHANGE to calculate an optimal SFC and perform the placement. When the SFC servers are ready to listen for the traffic, we then assign a corresponding amount of packets to each server on SFC according to the SFC request, forward them through the SFC and record their end-to-end delay s (*i.e.*, by time-stamping the data packets). We keep

Table 8.1: Network and VNF setting

Parameter	Value
VNFs per SFC	2-4
VNF request (byte)	[10, 25, 100]
Link delay (ms)	100 - 500
Migration delay (ms)	400 - 2000
VNF proc.delay (ms/byte)	10 - 50
Minimum speed	1 m/s
Maximum speed	5 m/s

the trace of the average response time of each request in this experiment and compare the result to the aforementioned offline optimum and ϵ-greedy algorithm.

The network settings we use in our experiment are shown in table 8.1. More specifically, each service request on the SFC is randomly chosen from [10, 25, 100] bytes (e.g. for a SFC with three services: [25, 10, 100] bytes). In our emulation, each packet contains one byte of data (e.g. 100 packets will be sent in a SFC request of 100 bytes), we configure the computing delays to be the time it takes for a server to process one packet, which varies from 10 to 50 ms/bytes, and the switch delays to be proportional to link delays, which vary from the range of [100, 500]ms and [400, 2000]ms, respectively.

8.2.1 Response Time

Figure 8.2 shows the convergence performance of CHANGE algorithm in an emulated wireless network using the network settings shown in table 8.1. As anticipated, despite getting unstable averages initially, CHANGE gets a decreasing average cost as the learning slot increases and converges to a stable level, which has similar traits as the convergence performance in simulations. The results also show that CHANGE outperforms ϵ-greedy over the entire course

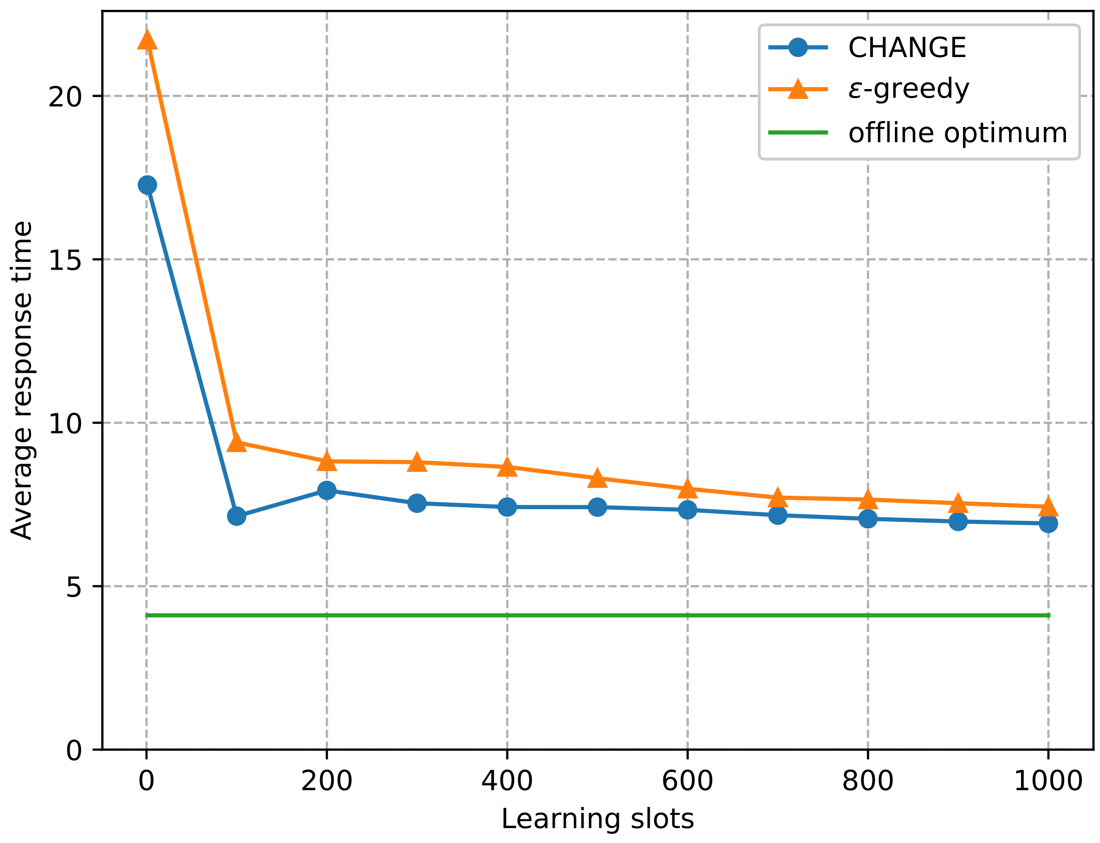


Figure 8.2: CHANGE outperform ϵ-greedy in Mininet-WiFi

of 1000 learning slots. This is because ϵ-greedy does not select arms combinatorially, and therefore has more extensive decision space than CHANGE, which results in a worse learning performance than CHANGE.

In the next sections, we evaluate the total response time of CHANGE in Mininet-WiFi under different environmental parameters such as Mobility models, network delay, and length of the SFC, then compare the respnse time with the two aforementioned greedy approaches.

8.2.2 User Mobility analysis

We first investigate how the mobility model affects the end-to-end latency. Specifically, we consider the following mobility models:

RW. random walk, a variant of the random waypoint model[37], where the user moves directly towards the next waypoint at a certain velocity and choose the next waypoint and velocity randomly. Compared to the random waypoint model, the random walk model has a constant velocity and does not have a wait time.

RD. random direction, a variant of the random waypoint model, compared to the random waypoint, the only difference is that the user has no wait time. The user travel with a velocity that is uniformly distributed between the minimum velocity and maximum velocity.

TVC. Time-Variant Community, In this model, the user will periodically re-appear at the same location defined as "communities".

GM. Gauss Markov, in this model, the velocity of the mobile user is assumed to be correlated over time and modeled as a Gauss-Markov stochastic process.

RP. Reference Point, this model simulates a group behavior where the user follows a group leader and are randomly distributed around a reference point. The user has a velocity and direction randomly derived from that of the group leader.

Figure 8.3 shows the performance of CHANGE under different mobility models in Mininet-WiFi, from which we can see that different mobility model can result in various average response time for both greedy and CHANGE algorithms, this is due to multiple parameters in a mobility model such as average velocity, wait time and mobility range. For example, the RD model results in a higher response time than RW because the user's velocity is continuously changing in RD while the RW model sets the user in a constant velocity.

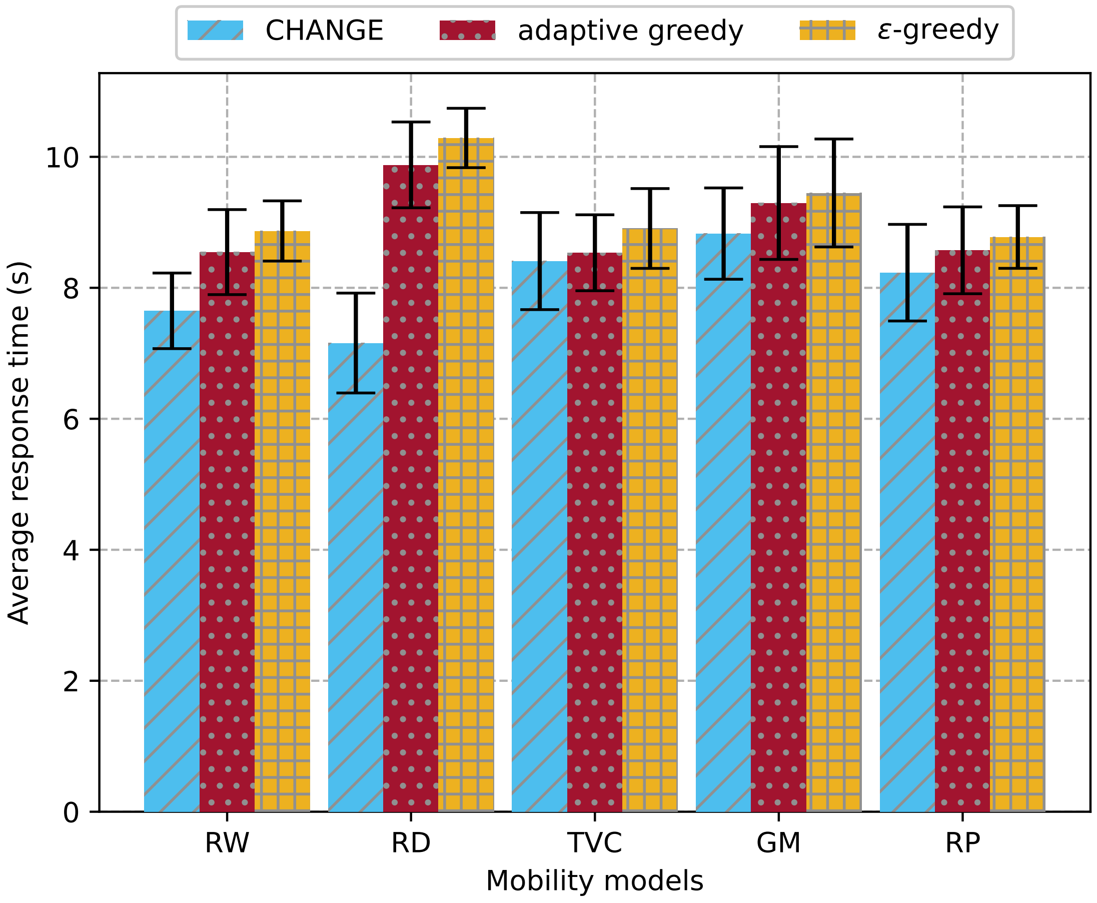
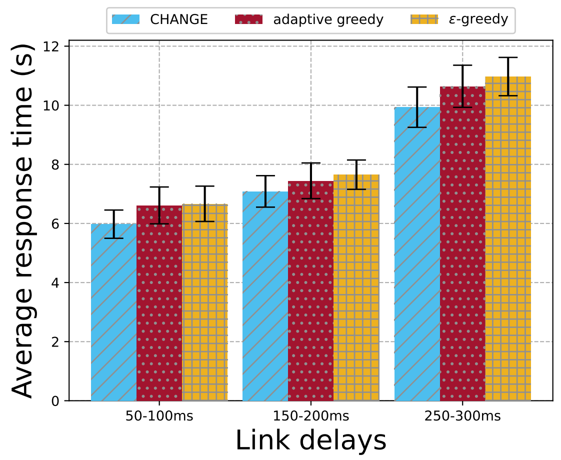
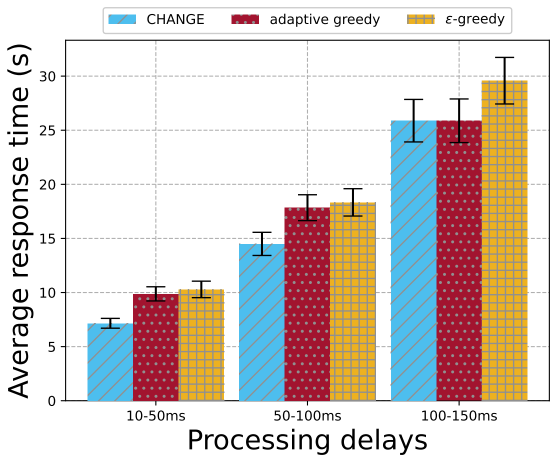


Figure 8.3: Effect of mobility models



(a) Effect of link delay



(b) Effect of processing delay

Figure 8.4: Effect of Mininet-WiFi network delays: a) link delay; b) processing delay

8.2.3 Different network delays

In this experiment, we analyze the impact of different delay settings (e.g. per-packet computing delay and transmission delay) on the average response time after 500 timeslots. The response time increase as the link delay increases. The results are shown in figure 8.4a and figure 8.4b ,respectively. It can also be observed that the processing delay has a greater impact on the overall response time than transmission delay in Mininet-WiFi emulation, which is often the case in most real-world SFC applications.

8.2.4 Different length of SFC.

Figure 8.5 shows the effect of SFC length (e.g the number of VNFs on each SFC) on average response time. As expected, response times increases as the length of SFC increases.

Overall, figure 8.4 and 8.5 shows that CHANGEoutruns the two greedy approaches by about 1 to 5 seconds under varying environmental settings.

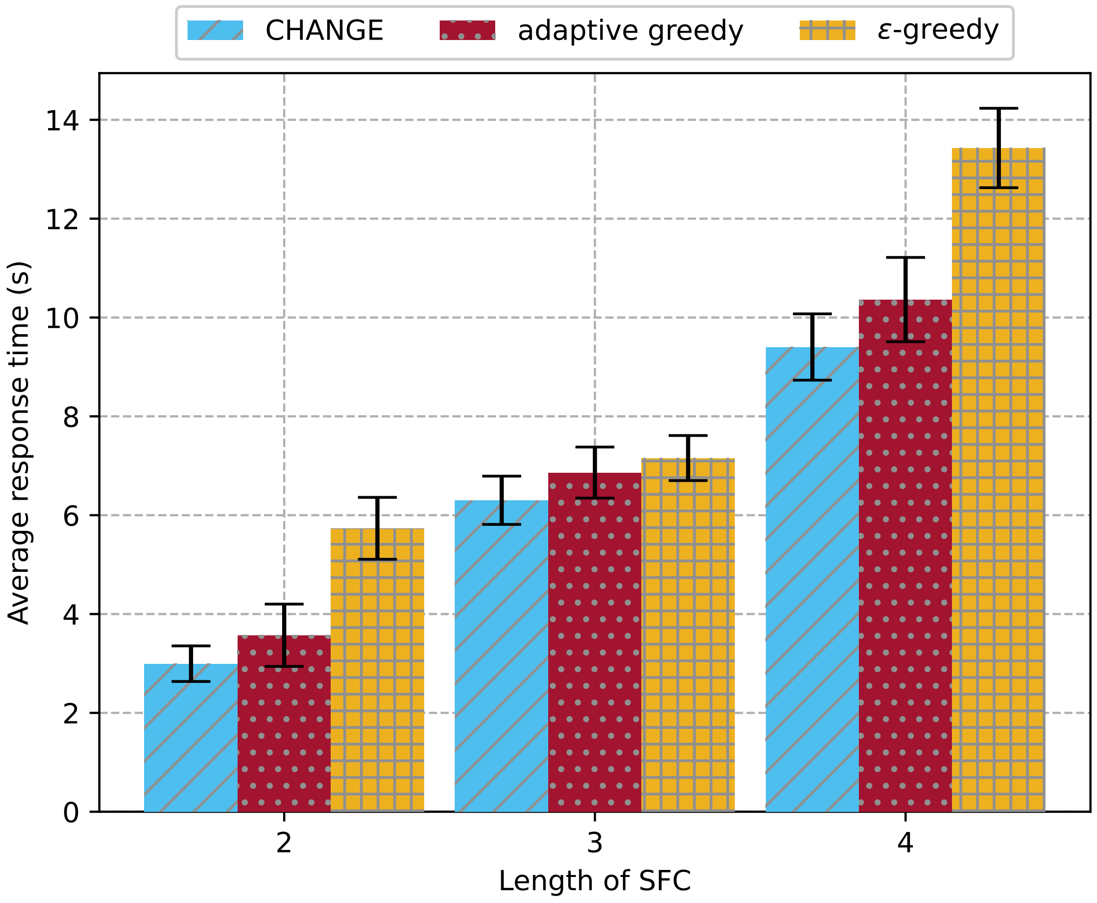


Figure 8.5: Effect of the length of SFC

Chapter 9

Conclusion

Mobile Edge Computing (MEC) empowers cloud computing by distributing cloud resources (*e.g.*storage and processing capacity) to the edge servers inside the range of radio access network (RAN) and bring them closer to end-users. It provides end-users with swift and powerful computing, energy efficiency, storage capacity, mobility, location, and context awareness support. Network Function Virtualization (NFV) enables scalable resource allocation, agile deployment, and efficient management of network services. It largely reduces the deploying and managing cost. Applying NFV to MEC will not only reduce the overall latency for end-users but also provide them with lower cost and flexible management over network services. One of the main considerations when applying NFV to MEC is the placement of Service Function Chains (SFC) on edge, where more changes such as service migration and user mobility need to be addressed.

In this work, we addressed the problem of user-managed service function chain orchestration with the objective of end-to-end delay optimization in mobile edge computing while considering the service migration cost and user mobility. We formulated the problem as an integer linear program and used its solution as an offline performance benchmark. In order to handle the uncertainties in the real environment, we applied the theory of contextual bandits, which reduces the problem complexity and utilizes the available information to make efficient decisions. Then, we used an efficient dynamic programming method to perform the chain orchestration task, which computes delay-optimized SFC placements in polynomial time. At last, through extensive simulations and emulations, we analyzed the utility and performance of our algorithm.

Section 9.1 of this chapter summarize the thesis with regards to our research focus and Section 9.2 discuss the future research directions.

9.1 Thesis Summary

In Chapter 1, we discussed the concept of NFV and MEC, as well as the advantages and challenges when applying NFV to MEC. There are three challenges in user-managed SFC placement on edge: Unknown system-wide information such as dynamic network capacities, user's mobility within MEC, and VNF migrations due to the user's mobility. We also briefly introduced the approaches taken to address the aforementioned challenges and the main contributions of this work.

Chapter 2 provided readers with the background information for acknowledging the work in this thesis. Particularly, we provided a summary of basic knowledge on NFV, SFC, and MEC architecture, then, we introduced the online learning techniques concerned in this work, including Greedy Multi-Arm Bandit learning and Combinatorial Contextual Multi-arm Bandit learning, and the mathematical techniques applied in this work including Binary Integer Linear Program, Binary Product Linearization and Dynamic Programming. We also summarized the software tools used in simulation and emulation, including Networkx simulator, Gurobi Optimizer, and Mininet-WiFi emulator.

In Chapter 3, we review and categorize the previous works that solve the SFC placement into four classes based on their problem model: system-managed edge-agnostic, system-managed edge-enabled, user-managed edge-agnostic, and user-managed edge-enabled. For each work reviewed, we discuss their system model, objective, and the applied optimization approaches and algorithms, as well as their weakness and relevance to our work.

Chapter 4 describes the mathematical models we adopt in this work in order to modelize the SFC placement in the mobile edge computing system, including network model and demand model, then we formally defined the user-managed edge-enabled SFC placement as a contrained optimization problem with the objective of minimizing the total end-to-end delay, which is formulated as a Binary Integer Linear Program problem.

In chapter 5, we present our proposed contextual combinatorial bandit formulation for the user-managed edge-enabled SFC placement problem. Specifically, we first utilize an unknown parameter vector for each server and link to capture and learn the system-wide information, including processing and bandwidth capacities. We then associate each server and link with a feature vector that characterizes user's side contextual information such as their service demand, current location, and previous SFC placement. Then, we define a linear reward function that utilizes the two aforementioned vectors to calculate the delays on each arm (*i.e.* each server and link). We further prove that with the proposed combinatorial bandit formulation, the decision space of the user can be considerably reduced compared to a general multi-arm bandit formulation. Finally, we present our proposed bandit-based algorithm CHANGE, that adopts the *upper confidence bound* theory to maintain confidence bound for each arm during the learning slots and uses dynamic programming algorithm to compute the optimal placement solution in polynomial time at every single time slot.

Chapter 6 provides a brief idea and design of the delay estimation framework that is required in the bandit formulation. The main idea of this framework is to use timestamps to record and keep track of the delays experienced on each node and link when network packets are routed through SFCs.

Chapter 7 presents extensive simulation results that demonstrate the performance of our proposed algorithm. The results show that CHANGE can cope with the system uncertainty and make balanced decisions in terms of migration, exploration, and exploitation under different system dynamics. We then study the learning behavior and convergence performance by analyzing the time average cost and total regrets during the learning slots and compare it with an optimal offline benchmark. We further studied the performance of our proposed algorithm against greedy approaches under different problem sizes. The comparison results show that our proposed algorithm outperforms the greedy approaches by at least 50 percent in terms of scalability.

Chapter 8 reported the results of realistic Mininet-wifi experiments we conduct in order to validate the superior performances of our proposed algorithm under the emulated wireless network environment. The collected empirical data indicate that our proposed algorithm has a similar learning behavior as simulation. We further study the performance of CHANGE under different network delay settings, mobility models, and SFC lengths. The result of this experiment shows that the performance observed in simulation is indeed achievable in practice.

9.2 Future work

SFC placement Strategies. We resort to using a dynamic programming approach to compute the placement solution at every single time slot. While we can attain the exact optimum using this approach, developing approximate placement algorithms that achieve near-optimum performance can result in less run time cost when the problem size is very large.

Network Topologies. The edge computing network topology we used in our simulation and emulation experiments were designed to be squared grid structures. An interesting research direction would be to consider other types of network topologies (*e.g.*, cluster topology) in the experiments.

Cooperation in Practice. In this work, we analyze the performance of CHANGE on both model-driven simulation and realistic Mini-WiFi emulations. A valuable next step is to implement and deploy such algorithm in a real-world MEC network for more empirical analysis.

Bibliography

[1] Cplex, 2019. URL: `http://www.gurobi.com`.

[2] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, and Taieb Znati. Replisom: Disciplined tiny memory replication for massive iot devices in lte edge cloud. *IEEE Internet of Things Journal*, 3(3):327–338, 2015.

[3] Ejaz Ahmed and Mubashir Husain Rehmani. Mobile edge computing: Opportunities, solutions, and challenges. *Future Generation Computer Systems*, 70:59 – 63, 2017.

[4] I. F. Akyildiz, A. Kak, and S. Nie. 6g and beyond: The future of wireless communications systems. *IEEE Access*, 8:133995–134030, 2020.

[5] A. Alleg, T. Ahmed, M. Mosbah, R. Riggio, and R. Boutaba. Delay-aware vnf placement and chaining based on a flexible resource allocation approach. In *Proc. 13th International Conference on Network and Service Management (CNSM)*, pages 1–7, 2017.

[6] Michael Till Beck, Sebastian Feld, Andreas Fichtner, Claudia Linnhoff-Popien, and Thomas Schimper. Me-volte: Network functions for energy-efficient video transcoding at the mobile edge. In *2015 18th International Conference on Intelligence in Next Generation Networks*, pages 38–44. IEEE, 2015.

[7] Christian Bettstetter, Hannes Hartenstein, and Xavier Pérez-Costa. Stochastic properties of the random waypoint mobility model. *Wireless Networks*, 10(5):555–567, 2004.

[8] Antonio Brogi, Stefano Forti, and Federica Paganelli. Probabilistic qos-aware placement of VNF chains at the edge. *CoRR*, abs/1906.00197, 2019. `arXiv:1906.00197`.

[9] T. H. Cormen C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to algorithms.*, volume 1. MIT press, Cambridge, MA, 2001.

[10] H. Chai, J. Zhang, Z. Wang, J. Shi, and T. Huang. A parallel placement approach for service function chain using deep reinforcement learning. In *Proc. IEEE 5th International Conference on Computer and Communications (ICCC)*, pages 2123–2128, 2019.

[11] D. Chemodanov, P. Calyam, and F. Esposito. A near optimal reliable composition approach for geo-distributed latency-sensitive service chains. In *Proc. IEEE INFOCOM*, pages 1792–1800, 2019.

[12] De-Yu Chen and Magda El-Zarki. A framework for adaptive residual streaming for single-player cloud gaming. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 15(2):1–23, 2019.

[13] L. Chen, J. Xu, S. Ren, and P. Zhou. Spatio–temporal edge service placement: A bandit learning approach. *IEEE Trans. Wireless Commun.*, 17(12):8388–8401, 2018.

[14] Lixing Chen, Jie Xu, and Zhuo Lu. Contextual combinatorial multi-armed bandits with volatile arms and submodular reward. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 3247–3256. Curran Associates, Inc., 2018.

[15] W. Chen, K. Ye, and C. Xu. Co-locating online workload and offline workload in the cloud: An interference analysis. In *Proc. IEEE HPCC/SmartCity/DSS*, pages 2278–2283, 2019.

[16] Wei Chen, Yajun Wang, and Yang Yuan. Combinatorial multi-armed bandit: General framework and applications. In *Proc. International Conference on Machine Learning*, pages 151–159, 2013.

[17] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, 2015.

[18] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros. Dynamic, latency-optimal vnf placement at the network edge. In *Proc. IEEE INFOCOM*, pages 693–701, April 2018. `doi:10.1109/INFOCOM.2018.8486021`.

[19] R. Cziva and D. P. Pezaros. Container network functions: Bringing nfv to the network edge. *IEEE Communications Magazine*, 55(6):24–31, 2017.

[20] Soumya Kanti Datta, Christian Bonnet, and Jérôme Härri. Fog computing architecture to enable consumer centric internet of things services. pages 1–2, 06 2015. `doi:10.1109/ISCE.2015.7177778`.

[21] NetworkX developers. Network analysis in python. URL: `https://networkx.github.io/`.

[22] M. Patel et al. Mobile-edge computing—introductory technical white paper. In *White paper, Mobile-Edge Computing (MEC) Industry Initiative*, 2014.

[23] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos, and C. E. Rothenberg. Mininet-wifi: Emulating software-defined wireless networks. In *Proc. 11th International Conference on Network and Service Management (CNSM)*, pages 384–389, 2015.

[24] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos, and C. E. Rothenberg. Mininet-wifi: Emulating software-defined wireless networks. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 384–389, 2015. doi:10.1109/CNSM.2015.7367387.

[25] GMDT Forecast. Cisco visual networking index: global mobile data traffic forecast update, 2017–2022. *Update*, 2017:2022, 2019.

[26] Wei Gao. Opportunistic peer-to-peer mobile cloud computing at the tactical edge. In *2014 IEEE Military Communications Conference*, pages 1614–1620. IEEE, 2014.

[27] R. Gargees, B. Morago, R. Pelapur, D. Chemodanov, P. Calyam, Z. Oraibi, Y. Duan, G. Seetharaman, and K. Palaniappan. Incident-supporting visual cloud computing utilizing software-defined networking. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(1):182–197, Jan 2017. doi:10.1109/TCSVT.2016.2564898.

[28] S. Geissler, S. Lange, F. Wamser, T. Zinner, and T. Hoßfeld. Komon — kernel-based online monitoring of vnf packet processing times. In *Proc. International Conference on Networked Systems (NetSys)*, pages 1–8, March 2019. doi:10.1109/NetSys.2019.8854514.

[29] Mostafa Ghobaei-Arani, Ali Asghar Rahmanian, Mahboubeh Shamsi, and Abdolreza Rasouli-Kenari. A learning-based approach for virtual machine placement in cloud data centers. *International Journal of Communication Systems*, 31(8):e3537, 2018. e3537 IJCS-17-0421.R1. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.3537, doi:10.1002/dac.3537.

[30] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2019. URL: `http://www.gurobi.com`.

[31] Aric Hagberg, Pieter Swart, and Daniel Chult. Exploring network structure, dynamics, and function using networkx. 01 2008.

[32] D. Harris, J. Naor, and D. Raz. Latency aware placement in multi-access edge computing. In *Proc. IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 132–140, June 2018. `doi:10.1109/NETSOFT.2018.8459997`.

[33] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in nfv: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[34] Steven CH Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey. *arXiv preprint arXiv:1802.02871*, 2018.

[35] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Mobile fog: A programming model for large-scale applications on the internet of things. pages 15–20, 08 2013. `doi:10.1145/2491266.2491270`.

[36] YunChao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing: A key technology towards 5g. In *ETSI White paper*, 2015.

[37] Esa Hyytiä and Jorma Virtamo. Random waypoint mobility model in cellular networks. *Wirel. Netw.*, 13(2):177–188, April 2007. `doi:10.1007/s11276-006-4600-3`.

[38] Yaser Jararweh, Ahmad Doulat, Ala Darabseh, Mohammad Alsmirat, Mahmoud Al-Ayyoub, and Elhadj Benkhelifa. Sdmec: Software defined system for mobile edge

computing. In *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, pages 88–93. IEEE, 2016.

[39] Q. Jin, S. Ge, J. Zeng, X. Zhou, and T. Qiu. Scarl: Service function chain allocation based on reinforcement learning in mobile edge computing. In *Proc. Seventh International Conference on Advanced Cloud and Big Data (CBD)*, pages 327–332, Sep. 2019. doi:10.1109/CBD.2019.00065.

[40] S. Kim, Y. Han, and S. Park. An energy-aware service function chaining and reconfiguration algorithm in nfv. In *Proc. IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 54–59, 2016.

[41] Paweł Kułakowski, Eusebi Calle, and Jose L. Marzo. Performance study of wireless sensor and actuator networks in forest fire scenarios. *International Journal of Communication Systems*, 26(4):515–529, 2013.

[42] Neeraj Kumar, Sherali Zeadally, and Joel JPC Rodrigues. Vehicular delay-tolerant networks for smart grid data management using mobile edge computing. *IEEE Communications Magazine*, 54(10):60–66, 2016.

[43] J. Kwak, Y. Kim, J. Lee, and S. Chong. Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems. *IEEE Journal on Selected Areas in Communications*, 33(12):2510–2523, 2015.

[44] Juha Laurila, Daniel Gatica-Perez, Imad Aad, Jan Blom, Olivier Bornet, T.-M.-T Do, Olivier Dousse, Julien Eberle, and Markus Miettinen. The mobile data challenge: Big data for mobile computing research. nokia research center. 01 2012.

[45] Aris Leivadeas, George Kesidis, Mohamed Ibnkahla, and Ioannis Lambadaris. Vnf placement optimization at the edge and cloud †. *Future Internet*, 11(3), 2019. URL: `https://www.mdpi.com/1999-5903/11/3/69`, `doi:10.3390/fi11030069`.

[46] Defang Li, Peilin Hong, Kaiping Xue, and Jianing Pei. Virtual network function placement and resource optimization in nfv and edge computing enabled networks. *Computer Networks*, 152:12 – 24, 2019. `doi:https://doi.org/10.1016/j.comnet.2019.01.036`.

[47] Y. Liu, H. Lu, X. Li, D. Zhao, W. Wu, and G. Lu. A novel approach for service function chain dynamic orchestration in edge clouds. *IEEE Communications Letters*, pages 1–1, 2020.

[48] Xiaoqiang Ma, Yuan Zhao, Lei Zhang, Haiyang Wang, and Limei Peng. When mobile terminals meet the cloud: computation offloading as the bridge. *IEEE Network*, 27(5):28–33, 2013.

[49] R. Mahmud, R. Vallakati, A. Mukherjee, P. Ranganathan, and A. Nejadpak. A survey on smart grid metering infrastructures: Threats and solutions. In *Proc. IEEE International Conference on Electro/Information Technology (EIT)*, pages 386–391, 2015.

[50] W. Mao, L. Wang, J. Zhao, and Y. Xu. Online fault-tolerant vnf chain placement: A deep reinforcement learning approach. In *Proc. IFIP Networking Conference*, pages 163–171, 2020.

[51] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled Letaief. Mobile edge computing: Survey and research outlook. 01 2017.

[52] Ahmed M Medhat, Tarik Taleb, Asma Elmangoush, Giuseppe A Carella, Stefan Covaci, and Thomas Magedanz. Service function chaining in next generation networks: State of the art and research challenges. *IEEE Communications Magazine*, 55(2):216–223, 2016.

[53] S. Mehraghdam, M. Keller, and H. Karl. Specifying and placing chains of virtual network functions. In *Proc. IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pages 7–13, 2014.

[54] Cedric Morin, Géraldine Texier, Christelle Caillouet, Gilles Desmangles, and Cao-Thanh Phan. VNF placement algorithms to address the mono- and multi-tenant issues in edge and core networks. In *Proc. 8th IEEE CLOUDNET*, Coimbra, Portugal, November 2019. URL: `https://hal.archives-ouvertes.fr/hal-02313403`.

[55] A. Muhammad, L. Qu, and C. Assi. Delay-aware multi-source multicast resource optimization in nfv-enabled network. In *Proc. IEEE International Conference on Communications (ICC)*, pages 1–7, 2020.

[56] M. Nakanoya, Y. Sato, and H. Shimonishi. Environment-adaptive sizing and placement of nfv service chains with accelerated reinforcement learning. In *Proc. IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 36–44, 2019.

[57] Y. Nam, S. Song, and J. Chung. Clustered nfv service chaining optimization in mobile edge clouds. *IEEE Communications Letters*, 21(2):350–353, Feb 2017. `doi:10.1109/LCOMM.2016.2618788`.

[58] M. Nguyen, M. Dolati, and M. Ghaderi. Deadline-aware SFC orchestration under demand uncertainty. *IEEE TNSM*, 17(4):2275–2290, 2020.

[59] Swaroop Nunna, Apostolos Kousaridas, Mohamed Ibrahim, Markus Dillinger, Christoph Thuemmler, Hubertus Feussner, and Armin Schneider. Enabling real-time context-aware collaboration through 5g and mobile edge computing. In *2015 12th International Conference on Information Technology-New Generations*, pages 601–605. IEEE, 2015.

[60] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang. Adaptive user-managed service placement for mobile edge computing: An online learning approach. In *Proc. IEEE INFOCOM*, pages 1468–1476, April 2019. `doi:10.1109/INFOCOM.2019.8737560`.

[61] NFV White Paper. Network functions virtualisation: An introduction, benefits, enablers, challenges call for action. issue 1. October 2012.

[62] Lijing Qin, Shouyuan Chen, and Xiaoyan Zhu. *Contextual Combinatorial Bandit and its Application on Diversified Online Recommendation*, pages 461–469. 04 2014. `doi:10.1137/1.9781611973440.53`.

[63] P. Ren, X. Qiao, Y. Huang, et al. Edge ar x5: An edge-assisted multi-user collaborative framework for mobile web augmented reality in 5g and beyond. *IEEE Trans. Cloud Comput.*, pages 1–1, 2020.

[64] Palash Roy, Anika Tahsin, Sujan Sarker, Tamal Adhikary, Md. Abdur Razzaque, and Mohammad Mehedi Hassan. User mobility and quality-of-experience aware placement of virtual network functions in 5g. *Computer Communications*, 150:367 – 377, 2020.

[65] Guto Leoni Santos and Patricia Takako Endo. Using reinforcement learning to allocate and manage service function chains in cellular networks, 2020. `arXiv:2006.07349`.

[66] Stefania Sardellitti, Gesualdo Scutari, and Sergio Barbarossa. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, 1(2):89–103, 2015.

[67] M. Savi, M. Tornatore, and G. Verticale. Impact of processing costs on service chain placement in network functions virtualization. In *Proc. IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pages 191–197, 2015.

[68] Seungwon Shin and Guofei Gu. Attacking software-defined networks: A first feasibility study. In *Proc. ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, page 165–166, 2013.

[69] Tolga Soyata, Rajani Muraleedharan, Colin Funai, Minseok Kwon, and Wendi Heinzelman. Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *Proc. 2012 IEEE symposium on computers and communications (ISCC)*, pages 000059–000066. IEEE, 2012.

[70] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[71] Noriyuki Takahashi, Hiroyuki Tanaka, and Ryutaro Kawamura. Analysis of process assignment in multi-tier mobile cloud computing and application to edge accelerated web browsing. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 233–234. IEEE, 2015.

[72] F. Tashtarian, M. F. Zhani, B. Fatemipour, and D. Yazdani. Codec: A cost-effective and

delay-aware sfc deployment. *IEEE Transactions on Network and Service Management*, 17(2):793–806, 2020.

[73] Mininet Team. Mininet an instant virtual network on your laptop (or other pc), 2012.

[74] M. T. Thi, S. Ben Hadj Said, and M. Boc. Sdn-based management solution for time synchronization in tsn networks. In *Proc. 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 361–368, 2020.

[75] Meng Wang, Bo Cheng, and Junliang Chen. Poster: A linear programming approach for sfc placement in mobile edge computing. In *Proc. 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3300061.3343394`.

[76] Yikai Xiao, Qixia Zhang, Fangming Liu, Jia Wang, Miao Zhao, Zhongxing Zhang, and Jiaxing Zhang. Nfvdeep: Adaptive online service function chain deployment with deep reinforcement learning. In *Proc. International Symposium on Quality of Service*, IWQoS '19, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3326285.3329056`.

[77] Yanghao Xie, Zhixiang Liu, Sheng Wang, and Yuxiu Wang. Service function chaining resource allocation: A survey, 2016. `arXiv:1608.00095`.

[78] Zichuan Xu, Wanli Gong, Qiufen Xia, Weifa Liang, Omer Rana, and Guowei Wu. Nfv-enabled iot service provisioning in mobile edge clouds. *IEEE Transactions on Mobile Computing*, 2020.

[79] B. Yang, W. K. Chai, Z. Xu, et al. Cost-efficient NFV-enabled mobile edge-cloud for low latency mobile applications. *IEEE Trans. Netw. Service Manag.*, 15(1):475–488, 2018.

[80] Song Yang, Fan Li, Stojan Trajanovski, Xu Chen, Yu Wang, and Xiaoming Fu. Delay-aware virtual network function placement and routing in edge clouds. *IEEE Transactions on Mobile Computing*, 2019.

[81] Curtis Yu, Cristian Lumezanu, Abhishek Sharma, Qiang Xu, Guofei Jiang, and Harsha V. Madhyastha. Software-defined latency monitoring in data center networks. In Jelena Mirkovic and Yong Liu, editors, *Passive and Active Measurement*, pages 360–372, Cham, 2015. Springer International Publishing.

[82] Yu Cao, Songqing Chen, Peng Hou, and D. Brown. Fast: A fog computing assisted distributed analytics system to monitor fall for stroke mitigation. In *Proc. IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 2–11, 2015.

[83] D. Zhang, Y. Ma, C. Zheng, et al. Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing. In *IEEE/ACM SEC*, pages 243–259, 2018.

[84] Tianzhu Zhang. Nfv platform design: A survey. *arXiv e-prints*, pages arXiv–2002, 2020.

[85] Z. Zhang, L. Ma, K. K. Leung, L. Tassiulas, and J. Tucker. Q-placement: Reinforcement-learning-based service placement in software-defined networks. In *Proc. IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1527–1532, July 2018. `doi:10.1109/ICDCS.2018.00159`.