CHANGE: Delay-Aware Service Function Chain Orchestration at the Edge

Lei Wang
University of Calgary
lei.wang2@ucalgary.ca

Mahdi Dolati
University of Tehran
mahdidolati@ut.ac.ir

Majid Ghaderi
University of Calgary
mghaderi@ucalgary.ca

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. However, the available computing capacity at the edge is limited compared to that of remote clouds. A promising solution for efficient utilization of the limited capacity at the edge is fine-grained processing of user demands via Virtual Network Functions (VNFs). In this approach, user service demands are expressed as Service Function Chains (SFCs), 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. The increasing number of users, however, challenges the scalability of system-managed SFC orchestration. To address this problem, we propose a user-managed online SFC orchestration framework at the edge of the network, called CHANGE, that minimizes service latency by jointly considering the effect of user mobility, edge capacity and service migration. We first present the theoretical foundations of CHANGE and then evaluate its performance via model-driven simulations and realistic Mininet-WiFi emulations. Our results show that CHANGE can improve latency performance by nearly 20% compared to other approaches.

I. Introduction

The next generation of mobile applications such as virtual and augmented reality [1] require extremely low latency to operate effectively. Such low latency requirements are challenging even for modern communication technologies such as 5G cellular communications [2]. A promising architecture that has the potential to enable cellular networks to offer low latency to mobile applications is Mobile Edge Computing (MEC) [3]. 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 resources at the edge is scarce and thus it is necessary to manage them efficiently to handle the ever-increasing user demands.

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. A critical challenge of using NFV at the edge is the orchestration of service function chains (SFCs). Each SFC is an ordered sequence of VNFs that are chained together to process the user traffic in order to implement a specific network service. SFC orchestration is the problem of placement, routing, and migration of VNFs to minimize the user-perceived service latency with acceptable operational cost.

The majority of previous research on SFC orchestration has focused on data center settings and does not consider edge resources and user mobility (*e.g.*, [5]). 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.*, [6]). 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 [7], [8]. 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.*, [9]). 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.

To this end, 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. We consider available resources on the local device (*i.e.*, hand-held equipment), at the edge, and in a remote cloud. Moreover, we present a Mixed-Integer Program (MIP) formulation to minimize the end-to-end service delay composed of processing, propagation, transmission, and service migration delays. To solve the problem, we employ the contextual combinatorial bandit framework to design an algorithm called CHANGE. Our proposed algorithm uses contextual information to incorporate available information about the user's demand and location into the resource allocation mechanism. CHANGE uses the combinatorial formulation to focus on the basic options (*i.e.*, servers and links). Thus, the action space (*i.e.*, number of arms) remains polynomial in the input size. 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 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 existing 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.

Our key contributions are summarized as follows:

- 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.
- 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.
- We develop a dynamic programming-based algorithm to compute delay-optimized SFC placements.
- We evaluate the performance of our algorithm using simulations and Mininet-WiFi emulations, which show that our algorithm is able to achieve near-optimal performance.

II. Related Work

In this section, we review the most relevant studies to our work. We first investigate the works that orchestrate SFC in edge-enabled environments. Then, we discuss the reinforcement learning (RL) based approaches.

SFC orchestration in MEC. Previous studies on SFC orchestration consider different aspects such as end-to-end delay [10], [11], deployment cost [11]–[14], and utility [15]. Authors in [10] provide a clustered NFV service chaining scheme that computes the optimal number of clusters to minimize the end-to-end delay for MEC services. Authors in [12] formulate the SFC orchestration problem as an ILP to minimize the resource and bandwidth consumption when deploying VNFs, and propose a heuristic to solve it. In [11], a parallel algorithm is designed to minimize deployment cost and end-to-end delay. Authors in [15] consider the utility goal and provide an approximation algorithm to maximize the number of satisfied clients. These studies focus on system-wide SFC orchestration, which is one shot offline optimization. Our work, however, focuses on user-managed online SFC orchestration that conducts optimization at run time to adapt to

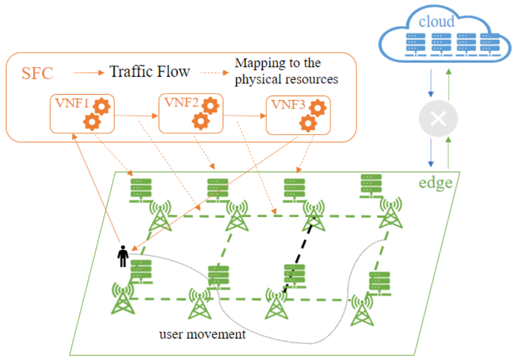


Fig. 1: An example of SFC placement in MEC.

network dynamics and unknown future demands and available resources.

RL-based approaches. Reinforcement learning is a powerful tool that is used in previous studies to handle uncertainty and design efficient online decision making strategies. Authors in [16] applied the contextual combinatorial bandit theory to design an online algorithm that allows application service providers to learn the demand patterns and rent edge resources to their end users. A system-managed learning mechanism based on proximal policy optimization to control the instantiation and deletion of VNFs across distributed data centers located close to the cellular base stations is presented in [17]. Authors in [18] used Q-learning to design a system-managed service orchestration algorithm in software-defined networks. Authors in [19] propose an accelerated RL method that divides the learning process into two steps to increase the scalability in real networks. ScaRL [20] leverages reinforcement learning to solve SFC allocation in MEC by using its trial-and-error mechanism. To handle the state space size, deep reinforcement learning (DRL) is employed in [21]–[24]. NFVdeep [21] models the system as a Markov Decision Process (MDP) and applies DRL to deploy SFCs. In [25], a quantum learning approach is proposed to handle the dynamic SFC orchestration in edge clouds. However, several of these works do not consider edge-enabled application scenarios and focus on traditional core cloud data centers (*e.g.*, [18], [19], [21]). Most of the works that are designed for the MEC environment do not consider VNF migration (*e.g.*, [17]–[25]). Other works like [24], [26] do not consider user-specific requests and mobility.

III. System Model

In this section, we describe the system model. Our model is able to represent existing edge-enabled architectures with necessary elements for bandwidth and processing resource allocation. We assume that each user is responsible for managing and resource allocation of her service, with the help of light-weight feedback about the system operation. 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.

TABLE I: Related work overview

Works	Models	Heuristic approaches	Online	Objective
[27]	ILP	✗	✗	Minimize end-to-end latency from all users to their respective VNFs
[28]	MILP	Ant Colony Optimization	✗	Minimize total VNF relocation and total response time
[11]	MILP	Tabu Search	✗	Minimize the end-to-end communication and the overall deployment cost
[29]	WGMP	Hungarian-based placement	✗	Minimize the total resource consumption and algorithm execution time
[12]	ILP	Priority based Greedy	✗	Minimize the total resource consumption
[10]	MIQCP	cluster based	✗	Minimize the average service time
[30]	ILP	✗	✗	Minimize the total resource consumption
[31]	✗	EdgeUser	✗	Maximize tolerated latency for SFC
[32]	MIQCP	✗	✗	Minimize resource consumption
[33]	MCCF	metapath composite variable	✗	Minimize a sum of SFC demands and corresponding physical resource capacity ratios
[26]	✗	best-fit decreasing	✓	Minimize energy consumption
[18]	✗	Reinforcement Learning	✓	Minimize the average service cost for end users
[9]	ILP	contextual multi-armed bandit	✓	Minimize the total service cost
[20]	ILP	Reinforcement Learning	✓	minimize the transmission latency and processing latency
[15]	ILP	$(1-1/e)$ deterministic	✗	Maximize the number of satisfied clients
[34]	MIQCP	✗	✓	Maximize the remaining data
[19]	✗	Reinforcement Learning	✓	throughput latency ratio
[21]	MDP	Deep Reinforcement Learning	✓	minimize the operation cost and maximize the total throughput
[25]	ILP	Quantum machine learning	✓	Minimize the end-to-end delay
[22]	✗	Deep Reinforcement Learning	✓	Minimize the resource cost
[17]	✗	Reinforcement Learning	✓	Minimizing energy consumption of allocating new VNFs
[16]	✗	COERR	✓	Maximize the utility of ASP
[27]	✗	Deep Reinforcement Learning	✓	Minimize the total end-to-end delay
[35]	ILP	Machine Learning	✓	minimize the overall user-to-sfc end-to-end latency
Our work	ILP	CHANGE	✓	Minimize the average response time

Network Model. As illustrated in Fig. 1, we consider a remote cloud C and a mobile access network consisting of a set of base stations. 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 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, delays inside the cloud are negligible.

Service Model. Each user in the vicinity of the mobile access network has a mobile device u (with the processing capacity of $c_u(t)$) which is always connected to the nearest base station (denoted by $\beta(t)$). Each user in timeslot t generates traffic at rate $\lambda(t)$ 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.*, a video transcoder) that can be deployed in the cloud or on an edge server with the means of state-of-the-art virtualization methods such as containers [36]. We use $\mathcal{S} = (s_1, \dots, s_k)$ to denote the list

[1]If the cores are heterogeneous, we must normalize the numbers concerning the weakest CPU core in the system.

TABLE II: List of main notations in the formulation

Notation	Description
C	Cloud
$\mathcal{T}$	Time frame
$\mathcal{L}$	Set of backbone links
$\mathcal{R}$	Set of backbone routers
$\mathcal{E}$	Set of all edge servers
$\mathcal{N}$	Set of all selectable nodes
$\mathcal{S}$	Set of all available services
α_s	Traffic scale factor of service s
π_s	Processing delay factor of service s
$\lambda(t)$	Current user traffic rate
$c_n(t)$	Current CPU cores of node n
$\beta(t)$	Current connected base station
$d_\ell(t)$	Current propagation delay on link ℓ
$b_\ell(t)$	Current bandwidth capacity on link ℓ
$\rho^s_{a,b}(t)$	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}

of 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 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 $\lambda_{s_i}(t) = \lambda(t) \times \prod_{j=1}^{i-1} \alpha_{s_j}$. Note that service migration is inevitable as the user moves in the environment. Let $\mathcal{N} = \mathcal{E} \cup \{C, u\}$ denote the set of nodes that can host a particular service and provide the required processing capacity. The *migration delay* $\rho^{s_i}_{a,b}(t)$ (seconds) 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 timeslot t (clearly, $\rho^{s_i}_{a,a}(t) = 0$). To serve a user request, each service in $\mathcal{S}$ should be deployed on a node with sufficient processing capacity, and the network routing layout should be adjusted such that user traffic goes through the services in the specified order. To ensure that the user receives the service result, we assume that there is a special service at the end of the chain which is constrained to be placed on the user device.

IV. Problem Formulation

In this section, we formally define the problem of minimum delay SFC orchestration at the edge.

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

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

Recent studies show that co-locating a user's services compromises the system reliability [37]. Thus, we include the following constraint to prevent co-located services,

$$\sum_{s \in \mathcal{S}} x_n^s(t) \leq 1. \qquad \forall n \in \mathcal{N}, t \in \mathcal{T} \tag{2}$$

Routing. A single path consisting of intermediate links and routers with enough bandwidth should be provisioned for every consecutive service s_i and s_{i+1}. 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 device that connects the device to the corresponding base station. We call this hypothetical router $\beta^{'}$ and denote the extended set of routers by $\mathcal{R}^{'} = \mathcal{R} \cup \{\beta(t), \beta^{'}\}$. The following constraints specify a path,

$$\sum_{\ell \in \delta^+(r)} y_\ell^{s_i}(t) - \sum_{\ell \in \delta^-(r)} y_\ell^{s_i}(t) = 0, \tag{3}$$

$$\sum_{\ell \in \delta^-(n)} y_\ell^{s_i}(t) - \sum_{\ell \in \delta^+(n)} y_\ell^{s_i}(t) = x_n^{s_i} - x_n^{s_{i+1}}, \tag{4}$$

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

$$\sum_{s_i \in \mathcal{S}} \lambda_{s_{i+1}}(t) y_\ell^{s_i}(t) \leq b_\ell(t). \qquad \forall \ell \in \mathcal{L}, t \in \mathcal{T} \tag{5}$$

Delay. The end-to-end chain delay is composed of four fundamental components: (1) *processing delay*[2], (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 $\Gamma_4(t)$, and are computed as follows:

$$\Gamma_1(t) = \sum_{s \in \mathcal{S}} \sum_{n \in \mathcal{N}} \frac{x_n^s(t) \pi_s \lambda_s(t)}{c_n(t)}, \tag{6}$$

$$\Gamma_2(t) = \sum_{s_i \in \mathcal{S}} \sum_{n \in \mathcal{N}} \sum_{\ell \in \delta^-(n)} \frac{y_\ell^{s_i}(t) \lambda_{s_{i+1}}(t)}{b_\ell(t)}, \tag{7}$$

$$\Gamma_3(t) = \sum_{s_i \in \mathcal{S}} \sum_{\ell \in \mathcal{L}} y_\ell^{s_{i+1}}(t) d_\ell(t), \tag{8}$$

$$\Gamma_4(t) = \sum_{a,b \in \mathcal{N}} x_a^s(t-1) x_b^s(t) \rho_{a,b}^s(t). \tag{9}$$

The user objective is to minimize weighted end-to-end delay,

$$\text{Min.} \sum_{t=0}^{\mathcal{T}} \sum_{i=1}^{4} \gamma_i \Gamma_i(t), \tag{10}$$

where, γ_i are the scale factors that specify the relative importance of different components of the end-to-end delay.

[2]We compute the processing delay using a simplified version of the Amdahl's law.

V. Proposed Method

In this section, we present the design of CHANGE, our proposed bandit-based algorithm for online SFC orchestration at the edge. To reduce the computational complexity of the bandit solution, we apply the contextual bandit formulation [38] by characterizing the environment with a set of *arms* (*i.e.*, placement options) and specifying their relations in subsection V-A. Then, we show how to reduce the uncertainty about the quality of arms and use the result to efficiently compute a service placement by dynamic programming in subsection V-B.

A. Bandit-based Environment Characterization

To learn about the environment, we consider each server and link as an option that can be used for placing a service. This 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 placement of the service chain and minimizes the end-to-end delay. At the end of each timeslot, the user observes the contribution of each option 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. We adopt the contextual information used in [9]. Thus, the observed value $\delta_o(t)$ is characterized as follows:

$$\delta_o(t) = \theta^\intercal \times \boldsymbol{\chi}_o(t) + \epsilon_o(t), \tag{11}$$

where, $\epsilon_o(t)$ is the zero-mean measurement noise. The user's goal is to minimize the expected cumulative delay in the period of service placement, which is defined as,

$$\mathbb{E}\left[\sum_{t \in \mathcal{T}} \sum_{o \in O_t} \delta_o(t)\right]. \tag{12}$$

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:

$$\theta_c = [\frac{1}{c_n}, \quad \forall n \in \mathcal{N}]. \tag{13}$$

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

$$\theta_b = [\frac{1}{b_\ell}, \quad \forall \ell \in \mathcal{L}]. \tag{14}$$

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

$$\theta_d = [d_\ell, \quad \forall \ell \in \mathcal{L}]. \tag{15}$$

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

$$\theta_\rho = [\rho^s_{a,b}, \quad \forall a, b \in \mathcal{N}, s \in \mathcal{S}]. \tag{16}$$

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

$$\theta = \theta_c \oplus \theta_b \oplus \theta_d \oplus \theta_\rho, \tag{17}$$

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.

$$\boldsymbol{\chi}^b_\ell(t) = [1_{\ell'=\ell, \exists n \in \mathcal{N}: \ell' \in \delta^-(n)} \lambda_s(t), \quad \forall \ell' \in \mathcal{L}]. \tag{18}$$

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:

$$\boldsymbol{\chi}^d_\ell(t) = [1_{\ell'=\ell}, \quad \forall \ell' \in \mathcal{L}], \tag{19}$$

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

Alg. 1: 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      $\hat{\theta}(t) \leftarrow \boldsymbol{V}^{-1}_{t-1}\boldsymbol{b}_{t-1}$
6      $\bar{\delta}_o(t) \leftarrow \hat{\theta}(t)^\intercal \boldsymbol{\chi}_o(t), \forall o \in O_t$
7      $\hat{\delta}_o(t) \leftarrow \bar{\delta}_o(t) - \sqrt{\boldsymbol{\chi}_o(t)^\intercal \boldsymbol{V}^{-1}_t \boldsymbol{\chi}_o(t)}$
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
```

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

$$\boldsymbol{\chi}^c_\ell(t) = [0, \quad \forall n \in \mathcal{N}], \tag{20}$$
$$\boldsymbol{\chi}^\rho_\ell(t) = [0, \quad \forall a, b \in \mathcal{N}, s' \in \mathcal{S}]. \tag{21}$$

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:

$$\boldsymbol{\chi}_\ell(t) = \boldsymbol{\chi}^c_\ell(t) \oplus \boldsymbol{\chi}^b_\ell(t) \oplus \boldsymbol{\chi}^d_\ell(t) \oplus \boldsymbol{\chi}^\rho_\ell(t), \tag{22}$$

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 $\boldsymbol{\chi}^c_n(t)$, is defined as follows:

$$\boldsymbol{\chi}^c_n(t) = [1_{n'=n} \times \pi_s \lambda_s(t), \quad \forall n' \in \mathcal{N}]. \tag{23}$$

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 $\boldsymbol{\chi}^\rho_n(t)$ and define it as follows:

$$\boldsymbol{\chi}^\rho_n(t) = [1_{s'=s, b=n} x^s_a(t-1), \quad \forall a, b \in \mathcal{N}, s' \in \mathcal{S}]. \tag{24}$$

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

$$\boldsymbol{\chi}^b_n(t) = [0, \quad \forall \ell \in \mathcal{L}], \tag{25}$$
$$\boldsymbol{\chi}^d_n(t) = [0, \quad \forall \ell \in \mathcal{L}]. \tag{26}$$

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

$$\boldsymbol{\chi}_n(t) = \boldsymbol{\chi}^c_n(t) \oplus \boldsymbol{\chi}^b_n(t) \oplus \boldsymbol{\chi}^d_n(t) \oplus \boldsymbol{\chi}^\rho_n(t), \tag{27}$$

B. Delay-aware SFC Placement

In this subsection, we present the design of CHANGE that efficiently estimates each arm's value, which is equivalent to the expected delay of each placement option. 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 1. 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 [38]. The algorithm starts with an initial estimate for the covariance matrix and mean vector of system parameters at lines 2 and 3. The expected values show our resource capacity estimates and the covariance matrix characterizes the uncertainty about the estimates. We gradually improve our estimations and update the covariance matrix accordingly. In each timeslot, the estimated system parameter vector is obtained from the covariance matrix and the mean vector in 5. The estimated system parameter vector is used to compute the excepted delay of each placement option in line 6. A lower bound for delay values is computed in line 7. A placement strategy is computed based on the obtained lower bounds in line 8. The placement strategy, based on dynamic programming, is outlined in Algorithm 2. Finally, the computed placement is implemented and the algorithm observes the delay values incurred as a result. In lines 9 and 10, the covariance matrix and the mean vector are updated accordingly.

The dynamic programming-based placement strategy in Algorithm 2 gets the delay estimates $\hat{\delta}_o(t)$ and required service set $\mathcal{S}[t]$ and calculates the optimal SFC placement by defining the cost of an optimal solution recursively in a bottom-up fashion. Our algorithm 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. There are three recurrence relations for the calculation of the minimum delay cost $\widetilde{D}(s_i, j)$ to place service s_i on node j as described on line 5-12, where $l_{j,k}$ denotes the link between node j and k. Each delay cost $\widetilde{D}(s_i, j)$ depends on $\widetilde{D}(s_{i+1}, j)$ and $\hat{\delta}_o(t)$. The algorithm then determines $\mathcal{S}[t]$'s optimal placement by extracting the node with the minimum delay cost to place the first service and the host list associated with it (line 18 and 19).

VI. Performance Evaluation

In this section, first, we use simulations to investigate different aspects of CHANGE with an emphasis on scalability. Then, we use Mininet-WiFi [39], which supports WiFi access points and user mobility, to study the performance of CHANGE under realistic environmental dynamics.

Alg. 2: 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 $P_t$
```

A. Simulation Settings

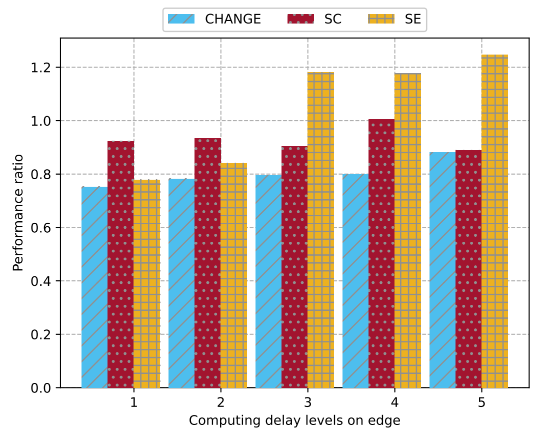
Network Settings. Similar to [9], we simulated a 2km $\times$ 2km grid network area with 25 edge servers and a remote cloud. The propagation delay of user-to-edge and edge-to-cloud links are uniformly distributed in $[10, 50]$ and $[100, 200]$ milliseconds, respectively. The service migration delay is uniformly distributed in $[10, 200]$ milliseconds. 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 in edge servers and cloud, respectively. We set these values based on [40] 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).

SFC Settings. We consider SFCs of length 3 to 5 (similar to [41]) and the traffic rate of each chain in each time slot is randomly selected from the set $\{10, 2.5, 0.25\}$ Kbps.

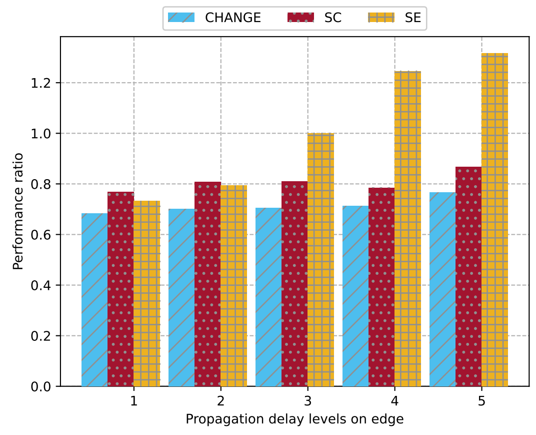
Mobility Model. The user selects random starting and destination points in the simulated area and walks with a constant speed chosen randomly from $[1, 1.5]$ meters per second. When the user reaches the destination, a new random destination is selected. Throughout the movement, the user is automatically connected to the closest radio base station.

Implemented Methods. We compare CHANGE with the offline optimum obtained by solving our optimization formulation with Gurobi [42] and the following online methods:

- **Serving on the cloud (SC):** The user always uses the remote cloud to minimize the computation delay.



(a) Computation delay.



(b) Propagation delay.

Fig. 2: Impact of computation and propagation delays.

- **Serving on the edge (SE):** The user always places the chain on the edge servers to avoid the latency to reach the cloud.
- **ϵ-greedy:** The user chooses a random placement with probability ϵ and otherwise computes the best solution based on delays observed in previous time slots.
- **Adaptive greedy:** Same as ϵ-greedy except that the value of ϵ decreases over time to avoid over-exploration.

The simulations parameters are summarized in Table III.

B. Simulation Results

In this subsection, we present our simulation results. All numeric results are normalized with an appropriate maximum or minimum value from each experiment.

TABLE III: 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]

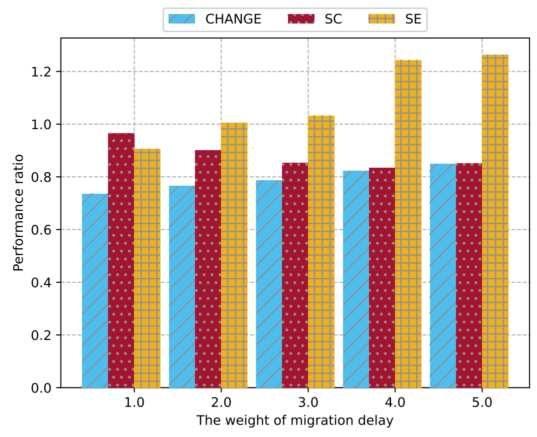


Fig. 3: Migration cost.

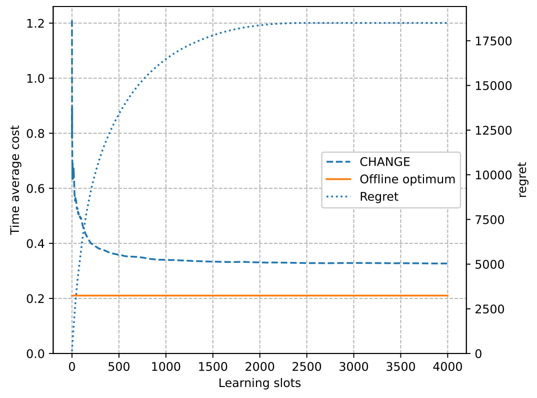
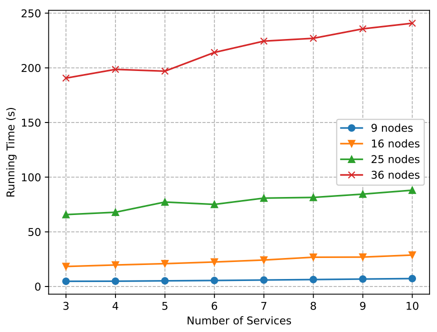


Fig. 4: Convergence analysis.

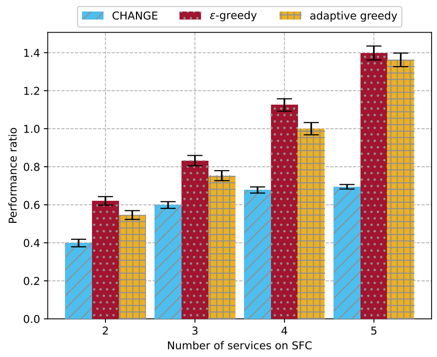
Impact of Network Delays. We considered five levels of computation and transmission delays to investigate their effect on service quality. Figs. 2(a) and 2(b) show that CHANGE achieves about $15 - 20\%$ reduction in end-to-end delay compared with other methods. For low delays, SE achieves a comparable 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.

Impact of Migration Delay. We considered five levels to represent the relative importance of migration delay to other types of delay. Fig. 3 shows that CHANGE outperforms SE and SC in all scenarios. Specifically, our proposed algorithm achieves an overall 20% improvement compared with other methods. Overall, Figs. 2 and 3 show that CHANGE is able to jointly consider different types of delay and optimize the performance with regards to them.

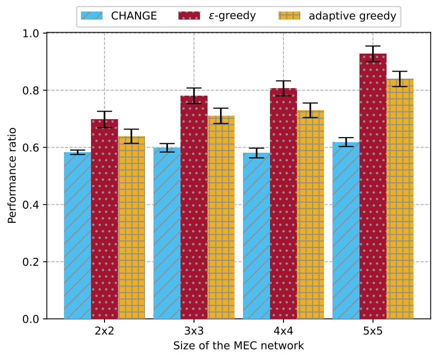
Convergence Analysis. To investigate the convergence of CHANGE, we trace the average SFC placement cost in each timeslot and compare it with the offline optimum. Fig. 4 shows CHANGE converges to a fixed value close to the offline optimum after about 500 time slots. Also, we presented the value of time-average regret that gradually converges to a constant value around 2000-th timeslot. The down-trend



(a) Run time analysis



(b) Number of services in SFC



(c) Size of the MEC network

Fig. 5: Scalability analysis by measuring runtime and cost for different network sizes and chain lengths.

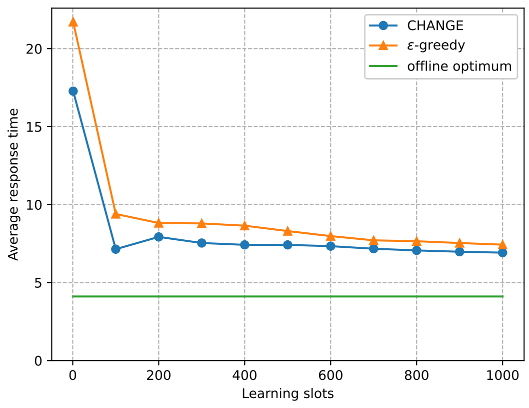


Fig. 6: Comparison of CHANGE and ϵ-greedy convergence in Mininet-WiFi.

indicates that CHANGE can quickly learn the system dynamics and make near-optimal decisions.

Scalability. To investigate the scalability, we gradually increased the problem size by considering grid networks with sizes 3×3 to 6×6 and chains with sizes 3 to 10. Fig. 5(a) shows that the runtime is dominated by the number of available edge servers. Considering that there are 1000 time slots in this simulation, the average per-time-slot running time is less than 0.3 seconds, which is acceptable for any practical usages. Then, we compare the cost of different algorithms for different chain and edge network sizes. Figs. 5(b) and 5(c) show that the performance of CHANGE is much less influenced by problem instances' complexity compared to other algorithms, which demonstrates that our algorithm is more scalable.

C. Mininet-WiFi Settings

Network and SFC Settings. We create an edge-enabled network with 9 wireless base stations in a 100×100 squared meter area. Each base station is connected to a server that provides computation services. The user, who is connected to the closest 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, sockets and artificial delay to emulate VNFs and computation delay, respectively. 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 service in the chain sends a response back that allows the user to collect the end-to-end-delay (by time-stamping the packets). Mininet-WiFi is set up in a VM that runs Ubuntu $16.04.2$ LTS with 2 CPU cores and 4GB of RAM. The VM runs on a Windows 10 machine with i5-7400 and 8GB of RAM.

Mobility Model. To further investigate the effect of mobility, we consider the following mobility models:

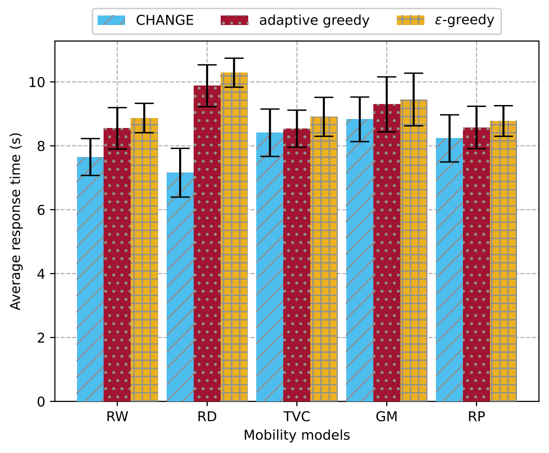
1) **Random Walk (RW)**: A variant of the random waypoint model [43], where the user moves with a constant velocity.
2) **Random Direction (RD)**: A variant of the random waypoint model, where the user has no wait time.
3) **Time Variant Community (TVC)**: In this model, the user will periodically re-appear at the same location.
4) **Gauss Markov (GM)**: In this model, the velocity of the user is assumed to be correlated over time and modeled as a Gauss-Markov stochastic process
5) **Reference Point (RP)**: This model simulates a group behavior where the user follows a group leader and are randomly distributed around a reference point.

D. Mininet-WiFi Results

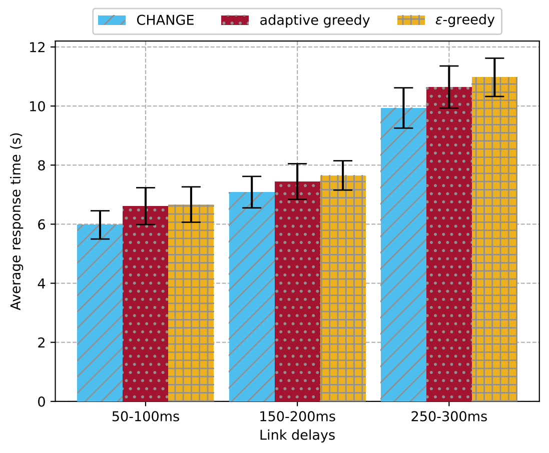
Response Time. Fig. 6 shows the convergence performance of CHANGE in an emulated wireless network using the settings shown in Table IV. Despite unstable averages at the beginning, CHANGE's average cost decreases as the learning slot increases and converges to a stable level around 1000-th time slots, which is similar to convergence results in the simulations. Also, CHANGE outperforms ϵ-greedy over the entire course of 1000 learning slots. This is because ϵ-greedy does not select

TABLE IV: Mininet-WiFi setting

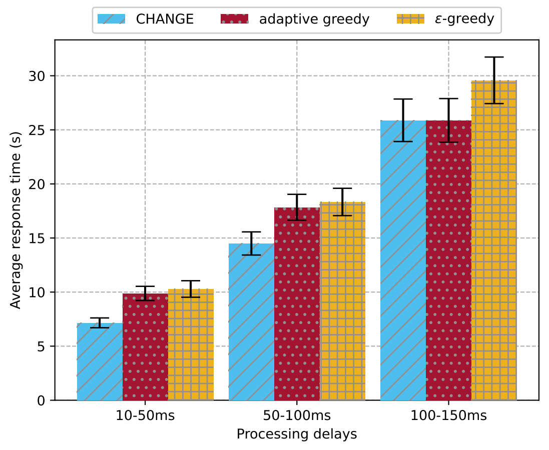
Parameter	**Value**
VNFs per SFC	2-4
VNF request (byte)	[10, 25, 100]
Link delay (ms)	100-500
Switch delay (ms)	400 - 2000
Proc.delay (ms/byte)	10 - 50
Maximum speed	5 m/s
Minimum speed	1 m/s



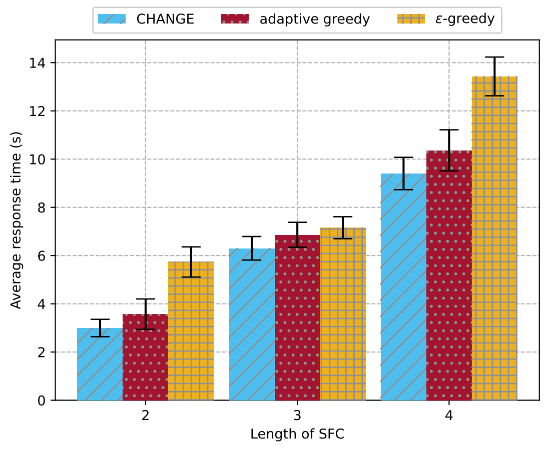
(a) Mobility Models



(b) Link Delay



(c) Processing Delay



(d) SFC Length

Fig. 7: Effects of environmental parameters in Mininet-WiFi.

arms combinatorially, and therefore has a larger decision space than CHANGE, which degrades its learning performance.

Mobility. Figure 7(a) shows the performance of CHANGE under different mobility models in Mininet-WiFi, from which we can see that different mobility models can result in various average response times 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, RD model results in a higher response time than RW because the user's velocity is continuously changing in RD, while RW model sets the user in a constant velocity.

Different Network Delays. In this experiment, we analyze the impact of different delays (*e.g.*, per-packet computing delay and transmission delay) on the average response time after 500 timeslots. The response time increases as the link delay and processing delay increase. The results are shown in Fig. 7(b) and Fig. 7(c), 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.

Different Length of SFC. Figure 7(d) shows the effect of SFC length (*e.g.*, the number of VNFs in each SFC) on average response time. As expected, response time increases as the length of SFC increases. Overall, Fig. 7 shows that CHANGE outruns the two greedy approaches by about 1 to 5 seconds under varying environmental settings.

VII. Conclusion

In this work, we addressed the problem of 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. To handle the uncertainties in the real environment we applied the theory of contextual bandits. Then, we used an efficient dynamic programming method to perform the chain orchestration task. Through extensive simulations and emulations, we analyzed the utility and performance of our algorithm.

In the future, we will consider multiple users and corporate different users' information into our service orchestrator and evaluate our proposal in larger network size and realistic network platforms. We will also consider more complex service models other than the chain model

References

[1] 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 TCC*, pp. 1–1, 2020.

[2] I. F. Akyildiz, A. Kak, and S. Nie, "6G and beyond: The future of wireless communications systems," *IEEE Access*, vol. 8, pp. 133 995–134 030, 2020.

[3] Y. Hu, M. Patel, D. Sabella *et al.*, "Mobile edge computing: A key technology towards 5G," *ETSI White Paper*, 2015.

[4] M. Chiosi, D. Clarke, P. Willis *et al.*, "An introduction, benefits, enablers, challenges & call for action," *ETSI White Paper*, 2012.

[5] M. Nguyen, M. Dolati, and M. Ghaderi, "Deadline-aware SFC orchestration under demand uncertainty," *IEEE TNSM*, vol. 17, no. 4, pp. 2275–2290, 2020.

[6] L. Chen, J. Xu, S. Ren, and P. Zhou, "Spatio–temporal edge service placement: A bandit learning approach," *IEEE Trans. Wireless Commun.*, vol. 17, no. 12, pp. 8388–8401, 2018.

[7] 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.*, vol. 15, no. 1, pp. 475–488, 2018.

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

[9] 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*, 2019, pp. 1468–1476.

[10] Y. Nam, S. Song, and J. Chung, "Clustered NFV service chaining optimization in mobile edge clouds," *IEEE COMML*, vol. 21, no. 2, pp. 350–353, 2017.

[11] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "VNF placement optimization at the edge and cloud," *Future Internet*, vol. 11, no. 3, 2019.

[12] D. Li, P. Hong, K. Xue, and J. Pei, "Virtual network function placement and resource optimization in NFV and edge computing enabled networks," *Comput. Netw.*, vol. 152, pp. 12 – 24, 2019.

[13] P. Jin, X. Fei, Q. Zhang *et al.*, "Latency-aware VNF chain deployment with efficient resource reuse at network edge," in *Proc. IEEE INFOCOM*, 2020, pp. 267–276.

[14] V. Farhadi, F. Mehmeti, T. He *et al.*, "Service placement and request scheduling for data-intensive applications in edge clouds," in *IEEE INFOCOM*, 2019, pp. 1279–1287.

[15] D. Harris, J. Naor, and D. Raz, "Latency aware placement in multi-access edge computing," in *Proc. IEEE NetSoft*, 2018, pp. 132–140.

[16] L. Chen and J. Xu, "Budget-constrained edge service provisioning with demand estimation via bandit learning," *IEEE J-SAC*, vol. 37, no. 10, pp. 2364–2376, 2019.

[17] G. L. Santos, J. Kelner, D. Sadok, and P. T. Endo, "Using reinforcement learning to allocate and manage SFC in cellular networks," in *Proc. IEEE CNSM*, 2020, pp. 1–5.

[18] Z. Zhang, L. Ma, K. K. Leung *et al.*, "Q-placement: Reinforcement-learning-based service placement in software-defined networks," in *Proc. IEEE ICDCS*, 2018, pp. 1527–1532.

[19] M. Nakanoya, Y. Sato, and H. Shimonishi, "Environment-adaptive sizing and placement of NFV service chains with accelerated reinforcement learning," in *Proc. IFIP/IEEE IM*, 2019, pp. 36–44.

[20] 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. CBD*, 2019, pp. 327–332.

[21] Y. Xiao, Q. Zhang, F. Liu *et al.*, "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *Proc. IEEE/ACM IWQoS*, 2019.

[22] 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 ICCC*, 2019, pp. 2123–2128.

[23] W. Mao, L. Wang, J. Zhao, and Y. Xu, "Online fault-tolerant VNF chain placement: A deep reinforcement learning approach," in *Proc. NETWORKING*, 2020, pp. 163–171.

[24] Y. Liu, H. Lu, X. Li, Y. Zhang, L. Xi, and D. Zhao, "Dynamic service function chain orchestration for nfv/mec-enabled iot networks: A deep reinforcement learning approach," *IEEE IoT-J*, 2020.

[25] 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 Wireless Commun. Lett.*, pp. 1–1, 2020.

[26] M. Ghobaei-Arani, A. A. Rahmanian, M. Shamsi, and A. Rasouli-Kenari, "A learning-based approach for virtual machine placement in cloud data centers," *Int. J. Commun. Syst.*, vol. 31, no. 8, p. e3537.

[27] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal VNF placement at the network edge," in *Proc. IEEE INFOCOM*, 2018, pp. 693–701.

[28] P. Roy, A. Tahsin, S. Sarker, T. Adhikary, M. A. Razzaque, and M. M. Hassan, "User mobility and quality-of-experience aware placement of virtual network functions in 5G," *Comput. Commun.*, vol. 150, pp. 367 – 377, 2020.

[29] M. Wang, B. Cheng, and J. Chen, "Poster: A linear programming approach for SFC placement in mobile edge computing," in *Proc. MobiCom*, 2019.

[30] C. Morin, G. Texier, C. Caillouet, G. Desmangles, and C.-T. Phan, "VNF placement algorithms to address the mono- and multi-tenant issues in edge and core networks," in *Proc. IEEE CloudNet*, 2019.

[31] A. Brogi, S. Forti, and F. Paganelli, "Probabilistic qos-aware placement of VNF chains at the edge," *CoRR*, vol. abs/1906.00197, 2019.

[32] 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. CNSM*, 2017, pp. 1–7.

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

[34] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Proc. IEEE CloudNet*, 2014, pp. 7–13.

[35] T. Subramanya, D. Harutyunyan, and R. Riggio, "Machine learning-driven service function chain placement and scaling in mec-enabled 5g networks," *Comput. Netw.*, vol. 166, p. 106980, 2020.

[36] R. Cziva and D. P. Pezaros, "Container network functions: Bringing NFV to the network edge," *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 24–31, 2017.

[37] 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*, 2019, pp. 2278–2283.

[38] L. Qin, S. Chen, and X. Zhu, "Contextual combinatorial bandit and its application on diversified online recommendation," in *Proc. SDM*, 2014, pp. 461–469.

[39] 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. CNSM*, 2015, pp. 384–389.

[40] M. Savi, M. Tornatore, and G. Verticale, "Impact of processing costs on service chain placement in network functions virtualization," in *Proc. IEEE NFV-SDN*, 2015, pp. 191–197.

[41] J. Kwak, Y. Kim, J. Lee, and S. Chong, "Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE J-SAC*, vol. 33, no. 12, pp. 2510–2523, 2015.

[42] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2019. [Online]. Available: http://www.gurobi.com

[43] E. Hyytiä and J. Virtamo, "Random waypoint mobility model in cellular networks," *Wirel. Netw.*, vol. 13, no. 2, p. 177–188, 2007.