2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS) | 978-1-7281-7002-2/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDCS47774.2020.00051

To Cache or Not to Cache: Stable Service Caching in Mobile Edge-Clouds of a Service Market

Zichuan Xu[‡], Yugen Qin[‡], Pan Zhou[$], John C. S. Lui[†], Weifa Liang[¶],
Qiufen Xia[*#], Wenzheng Xu[%], and Guowei Wu[‡]
[‡] School of Software, Dalian University of Technology, Dalian, China, 116621.
[#] International School of Information Science & Engineering, Dalian University of Technology, Dalian, China, 116621.
[$] School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, Hubei China.
[†] Department of Computer Science & Engineering, The Chinese University of Hong Kong, Shatin, N.T, Hong Kong.
[¶] Research School of Computer Science, Australian National University, Canberra, ACT 2601, Australia
[%] School of Computer Science, Sichuan University, Chengdu, Sichuan, China.
Emails: z.xu@dlut.edu.cn, qyg@mail.dlut.edu.cn panzhou@hust.edu.cn, cslui@cse.cuhk.edu.hk wliang@cs.anu.edu.au,
qiufenxia@dlut.edu.cn, wenzheng.xu@scu.edu.cn, wgwdut@dlut.edu.cn.
[*] Corresponding author: Qiufen Xia, email: qiufenxia@dlut.edu.cn.

***Abstract*—Mobile edge computing (MEC) is emerging as an enabling technology of low-latency network services, such as Augmented Reality (AR) and Virtual Reality (VR), by deploying cloudlets in locations close to users. In MEC networks, telco-operators can place their services to cloudlets, such that the service accessing delay of users is minimized. In this paper, we investigate a fundamental problem of caching services that are originally deployed in remote clouds to cloudlets in an MEC network within the proximity of users. Specifically, we focus on the *service caching* problem in a two-tiered MEC network with both remote clouds and cloudlets that are close to users, in which multiple network service providers competing computing and bandwidth resources. This setting is significantly different from existing studies that focused on offloading user tasks from mobile devices to cloudlets in MEC networks that typically do not consider a service market with multiple network service providers. For the service caching problem in a two-tiered MEC network, we propose a novel approximation-restricted framework that guarantees the stableness of the service market. Under the proposed framework, an approximation algorithm with an approximation ratio for the problem with non-selfish players and an efficient, stable Stackelberg congestion game with selfish players have been proposed. We also analyze the Price of Anarchy (PoA) of the proposed Stackelberg congestion game to measure the efficiency of the proposed game degrades due to selfish behavior of network service providers. We finally evaluate the performance of our mechanism on both simulated environments and a real test-bed. Results show that the performance of our proposed mechanism is promising.**

***Index Terms*—Mobile edge computing; Service caching; Task offloading; Stackelberg congestion game; Price of Anarchy.**

I. Introduction

Mobile edge computing (MEC) is envisioned to be able to revolutionize various multimedia applications, such as Augmented Reality (AR) and Virtual Reality (AR). For example, typical AR/VR services intensively need computing resource for rendering and processing [41]. AR/VR services therefore are usually implemented in central clouds with abundant computing resource. However, interactive AR/VR services have very stringent requirements on the motion-to-photon latency, and using central clouds often leads to unacceptable delay (e.g. hundreds of milliseconds [11]) and heavy backhaul resource usage. By deploying VR services in cloudlets at a specific location (such as museums and sport stadium) or even directed at 5G base-stations within the proximity of users, the delay experienced by VR users can be significantly reduced.

Since resources in an MEC are limited, each cloudlet may only be able to host a limited number of services. Therefore, caching services in cloudlets of the MEC temporarily while keeping the original instances of the services is a smart, progressive, and economic approach towards the wide adoption of the MEC technique. In this paper, we consider a fundamental problem of *service caching* in an MEC of a service market with multiple network service providers [19], [37] competing resources in cloudlets of the MEC. Each network service provider is selfish and only aims to maximize its own revenue by caching its services from remote clouds to cloudlets in a two-tiered MEC.

The key technical question of the service caching problem in a service market is how to design stable and performance-guaranteed mechanisms for the market. The challenges are three folds: (1) Network service providers are selfish and want to maximize their own revenue by caching their services to an MEC network with finite resources. This however may lead to significant congestions of some cloudlets of the MEC. Such congestion will eventually push up its processing delay, trading-off the reduced network latency brought by the MEC technique. Therefore, each network service provider needs to strategically decide whether to cache their service or not, considering the congestion levels of the cloudlets in the MEC. That is, how to design efficient mechanisms that could reduce the congestion levels of cloudlets in the MEC network while meeting the requirements of both the infrastructure and selfish network service providers; (2) the selfishness of network service providers in a service market may jeopardize the social benefit of all players [31], thereby leading to an unstable market that

2575-8411/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDCS47774.2020.00051

no player wants to participate in. Therefore, the mechanisms for the market have to be stable and the obtained solution should be close to the optimal one; (3) unlike conventional mobile computing environments, service caching aims not only to place instances of services in cloudlets, but also to keep the cached and original instances consistent via data updates. Specifically, how to place the to-be-cached instances, assign requests to the cached services, and update the data processed by cached instances to their original instances in remote data centers are non-trivial. To tackle these challenges, we investigate the service caching problem in a two-tiered MEC network of a service market by designing a stable and near-optimal service caching mechanism.

While several studies on computation offloading and service placement [5], [3], [6], [10], [28], [13], [30], [32], [18], [35], [22], [37], [23], [27], [36], [24], [25], [26], [40] have been conducted in the past, service caching from remote cloudlets in the core network to cloudlets in MEC networks has been hitherto overlooked. Specifically, these studies only focused on task offloading or service placement from mobile devices to cloudlets. In contrast, we focus on the service caching from remote clouds to cloudlets in a two-tiered MEC network, thereby enabling a smooth transition to the MEC environment. Also, due to the limited capacities of cloudlets, the MEC network may get congested with more and more services being cached in them, thereby trading-off the benefits brought by MEC. Such congestion-aware service caching is largely not considered by existing studies.

To the best of our knowledge, this is the first study that deals with the congestion-aware caching of delay-sensitive services from remote clouds to cloudlets in an MEC network operated by an infrastructure provider who provide Virtual Machines (VMs) to multiple mobile service providers. We provide both an approximation algorithm with an approximation ratio and Stackelberg congestion game with a guaranteed gap of the obtained solution to the optimal one.

The main contributions of this paper are summarized as follows.

- We are the first to consider the service caching problem in a two-tiered MEC of a service market with both an infrastructure provider and multiple selfish mobile service providers.
- We formulate the problem as a Stackelberg game, and propose a novel *approximation-restricted* strategy that guarantees a bounded Price of Anarchy (PoA) of the mechanism via a novel approximation algorithm.
- We devise an approximation algorithm for the problem with non-selfish users. We devise a novel Stackelberg congestion game for the service caching problem in an MEC network with finite resource capacities, and analyze the Price of Anarchy (PoA) of the game.
- We evaluate the performance of our mechanism not only in simulated environments but also in a real test-bed.

The remainder of the paper is arranged as follows. Section II introduces the system model, notations and problem formulation. The proposed Stackelberg congestion game for the service caching problem is described in Section III. Section IV will provide some experimental results on the performance of the proposed algorithm in both simulated environments and a real test-bed. Section V will summarize the state-of-the-arts on the service caching in MEC networks. The paper is concluded in Section VI.

II. Preliminary

In this section, we first introduce the system model and notations. We then formally define the service caching problem.

A. System model

We consider a two-tiered mobile edge-cloud (MEC) network $G = (\mathcal{CL} \cup \mathcal{DC}, E)$ consisting of a set $\mathcal{CL}$ of cloudlets that are deployed in locations within the proximity of users and a set $\mathcal{DC}$ of remote data centers. Let CL_i be a cloudlet in $\mathcal{CL}$. A number of virtualized servers can be instantiated in each cloudlet CL_i to implement mobile services and their related databases/libraries. The computing resource in each CL_i is managed by an infrastructure provider via Virtual Machines (VMs) provisioning. In addition, the cloudlets have a limited capacity of transferring data traffic from/to itself due to the capacity of its incident links. Denote by $C(CL_i)$ and $B(CL_i)$ the computing and bandwidth resource capacities of each cloudlet CL_i, respectively. Here, we do not consider the capacity constraint of each data center in $\mathcal{DC}$, since they usually have abundant resources. E is a set of links that interconnect the cloudlets and data centers in $\mathcal{CL} \cup \mathcal{DC}$.

As illustrated in Fig. 1, the two-tiered MEC network G is operated by an infrastructure provider and the resources are leased to a few selfish network service providers. This is a common business in the telecommunication market. For example, large-scale telco providers usually own infrastructure and want to lease their resource to small- and medium-scale network service providers. Since each network service provider is selfish, it competes the resources in the MEC network with other network service providers.

B. Service caching from remote clouds to cloudlets in an MEC network

We consider the provisioning of delay-sensitive network services, e.g., VR/AR, video processing, and online gaming services, which are frequently required by mobile users. To reduce the latency experienced by mobile users, network service providers hope to move the services that are originally deployed in remote data centers to cloudlets in an MEC network. Such service moving from remote data centers to cloudlets is referred to as *service caching*. In contrast to traditional service placement in MEC networks, service caching focuses on caching existing services in data centers to cloudlets in MEC networks. Due to the resource capacity constraints of cloudlets, services are only cached for temporary and their original services are still kept in remote data centers for later use when the cached service is destroyed from cloudlets.

We refer an implementation of service SV_l in a VM of a data center as its *original instance*. If an instance of SV_l is cached

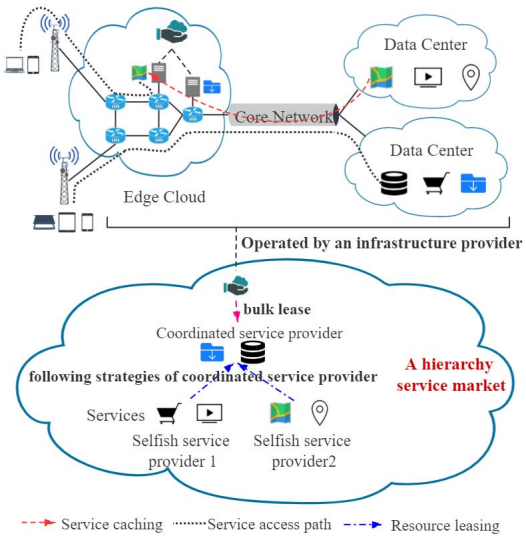


Fig. 1. An example of the two-tiered cloud network.

to a cloudlet in G, all of its user requests will be directed to the *cached instance* of SV_l for processing. Otherwise, the *original instance* of SV_l in its data center will continue serving its user requests, as shown in Fig. 1. Denote by r_l the number of requests that should be served by service SV_l. Assuming that each request of SV_l having a uniform workload, the amount of resource demanded by each service SV_l is $a_l \cdot r_l$. In addition, transmitting data from and to the cached instance of SV_l consume bandwidth resource. Therefore, each network service provider assigns some amount of bandwidth resource to each of its user requests to guarantee its performance. Denote by b_l the bandwidth resource assigned to each of the user requests of SV_l, the bandwidth consumption of service SV_l thus is $b_l \cdot r_l$.

Let sp_l be a network service provider offering services in the MEC network G, and each service provider is selfish and want to maximize its own revenue. Denote by N a set of such network service providers. Each $sp_l \in N$ requests to cache a service SV_l in a cloudlet in G. Such service SV_l may also be removed from the cloudlet, considering the capacity constraint of the cloudlet. Such service caching is usually offered as value-added features of existing cloud services. Therefore, the original service instance of SV_l in its remote data center will not be removed even if it is cached in the MEC network.

C. Cost model of service caching

Since the network service providers in N lease resources from the network infrastructure provider that owns the MEC network G, they usually need to pay for resource usages. Following the resource setting and pricing policies of most infrastructure providers [1], [8], the costs of using VMs are due to the usage of both computing and bandwidth resources of VMs in a cloudlet. The cost of caching a service in a cloudlet thus consists of a service instantiation cost in a cloudlet due to the instantiation of a VM, processing cost due to the processing of data traffic, and the update cost from the cached service to its original service in a remote cloud. These costs are categories into *service caching cost* and *bandwidth consumption cost*.

Service caching cost: Since each service is cached into a VM, the service instantiation cost thus is the cost due to VM instantiation and software setup for the cached service instance. Let c_l^{ins} be the cost of instantiating an instance of service SV_l of network service provider sp_l. If service SV_l is cached into a cloudlet of G, the data of its user requests are directly forwarded to the cloudlet for processing. This incurs costs since data processing is computing-intensive and requires resource usage of computing resource. Since the cached services in cloudlet CL_i are sharing its computing resource, the cost of caching a service in CL_i depends on the workload (i.e., congestion) of CL_i. Intuitively, the cost due to congestion is non-decreasing with the congestion levels. For simplicity, we adopt a proportional congestion model, following many exiting studies [4], [12], [21]. Let σ_i be the set of service providers that have service instances cached in cloudlet CL_i. The service caching cost is calculated by

$$\alpha_i|\sigma_i| + c_l^{ins}, \tag{1}$$

where α_i is a given constant that captures the influence of congestion of computing resource on the cost. It must be mentioned that the derivation technique in the later section does not relay on the assumption of proportional cost model. Instead, it relies only on the non-decreasing of cost with congestion levels. Therefore, the proportional congestion cost model can be easily extended to consider other complicated non-decreasing cost models.

Bandwidth consumption cost: Recall that service SV_l is temporarily cached into a cloudlet, and we need to maintain the original instance of SV_l in the remote cloud. However, to make sure the seamless service transition between of the cached and original service, the processed data of the cached instance needs to be updated/synchronized to its original instance in the remote cloud. Clearly, the cost incurred by such updates is due to the bandwidth resource consumption of cloudlet CL_i. Recall that each network service provider is assigned with an amount $b_l \cdot r_l$ of bandwidth resource. Hence, there is a fixed bandwidth consumption cost, referred to as c_i^{bdw}, for each network service provider in each cloudlet CL_i. In addition, it is clear that the bandwidth consumption cost is affected by other network service providers using the bandwidth resource of CL_i, i.e., the congestion of cloudlet CL_i. We calculate the update cost by

$$\beta_i|\sigma_i| + c_i^{bdw}, \tag{2}$$

where β_i is a given constant that captures the influence of congestion of bandwidth resource of CL_i on the cost. Notice that we focus on the provisioning of network resources for such updating/synchronization, the specific synchronization methods depends on specific services. We thus consider that the design of synchronization methods is out the scope of this paper.

The cost $c_{l,i}$ of caching an instance of service SV_l in CL_i is

$$c_{l,i} = \alpha_i|\sigma_i| + c_l^{ins} + \beta_i|\sigma_i| + c_i^{bdw}. \tag{3}$$

D. Hierarchy service markets

There is an emerging 5G service market which allows multiple network service providers to provide different services, such as VR services to mobile users. To promote network latency experienced by users, these network service providers typically lease VMs from an infrastructure provider that operates the multi-tier MEC network. Specifically, we consider a hierarchy service market with an infrastructure provider offering various resources to a number of selfish network service providers, as shown in Fig. 1. As a *leader*, the infrastructure provider has bulk-lease contracts with several large-scale network service providers; it thus can coordinates them as long as requirements in the bulk-lease contracts are met. Specifically, such a *coordinated network service provider* has huge demands of resources and large quantity of user requests. Therefore, they have significant influence in the service market, and their behavior may affect many medium and small-scale network service providers. In addition, medium and small-scale network service providers lease resources on a pay-as-you-go basis. They are interested in their own revenue. We refer to such network service providers as *selfish network service providers*. Denote by N the set of all network service providers and S the set of coordinated network service providers. $N \setminus S$ then denotes the number of selfish network service providers.

E. Stackelberg congestion game and price of anarchy

In the above-mentioned service market for an MEC, we consider a Stackelberg congestion game where the leader (i.e., infrastructure provider) coordinates a few network service providers while allows the rest network service providers behave selfishly. Without loss of generality, we consider a symmetric Stackelberg where the strategy space of each network service provider is the same. The Stackelberg game is denoted by $\Gamma(N, \mathcal{CL}, (\sigma_l)_{sp_l \in N}, (c_i)_{CL_i \in \mathcal{CL}})$, where $\sigma_l \in 2^{|\mathcal{CL}|} \setminus \{\emptyset\}$ is the strategy space of each network service provider, and c_i is a non-negative and non-decreasing cost function associated with caching network service in each cloudlet CL_i. Then,

$$c_i = \sum\nolimits_{sp_l \in \sigma_i} c_{l,i}. \tag{4}$$

Denote by $c_l(\sigma_l)$ the cost of network service provider sp_l with strategy space σ_l, which can be calculated by

$$c_l(\sigma_l) = \sum\nolimits_{CL_i \in \sigma_l} x_{l,i} \cdot c_i, \tag{5}$$

where $x_{l,i}$ is a binary indicator variable that indicates whether network service provider sp_l chooses cloudlet CL_i to cache an instance of its service SV_l. A *Stackelberg strategy* is an algorithm that chooses a subset of players and assigns them a prescribed strategy with the purpose of mitigating the detrimental effect of the selfish behavior of the remaining uncoordinated players.

Note that the network service providers operate in their own interest, which often leads to great inefficiencies and instabilities in the MEC network. They however are not expecting such inefficiency. Therefore, we need to measure the inefficiency caused by the selfishness of network service providers. The most popular metric for the evaluation of such selfishness is called the *Price of Anarchy* (PoA). It is defined as the proportion between the worst possible social utility from a Nash equilibrium and the optimal social utility in which players are not selfish, not necessarily from a Nash equilibrium.

F. Problem definition

Given an MEC network $G = (\mathcal{CL} \cup \mathcal{DC}, E)$ managed by an infrastructure provider and its resources shared by both coordinated and selfish network service providers. There is also a set of network service providers that have bulk-purchase contracts with the infrastructure provider and they are coordinated by the infrastructure provider. In addition, medium and small scale network service providers hoping to cache their services with the aim of maximizing their own revenue.

The service caching problem in an MEC network of a service market is to cache the services of the N network service providers, by selecting and coordinating a subset of network service providers from N, and allowing the rest to perform selfishly, such that the social cost of the mobile service market is minimized while no players have incentives to deviate from their current strategies, i.e., the *Nash equilibrium* of the mobile service market exists, where the social cost of the mobile service market in a mobile edge-cloud is defined as the total cost of all players in N, i.e.,

$$c = \sum\nolimits_{sp_l \in N} c_l(\sigma_l), \tag{6}$$

subject to the computing and bandwidth resource capacity constraints of the mobile edge cloud.

III. An Efficient and Near-Optimal Mechanism for the Service Caching Problem

We now devise an efficient and near-optimal mechanism for a novel Stackelberg game for the service caching problem. We first propose a novel strategy called *approximation strategy* in the mechanism design, such that the performance of the proposed mechanism is not far from the game when all players are not selfish. We then devise an approximation algorithm for problem with non-selfish players. Based on the approximation strategy and the approximation algorithm, we finally elaborate on the proposed Stackelberg strategy for the service caching problem.

A. Design rationale of the mechanism

We aim to design a mechanism with a guaranteed performance gap with the social optimum. Clearly, the selfish behavior of the selfish network service providers degrades the system performance, as they only care about their own revenue. We thus design a Stackelberg strategy for the infrastructure provider, such that it can find a subset of the players and coordinate them to avoid significant performance deviation caused by the rest selfish network service providers.

The basic idea of our algorithm is to focus on the *optimal-restricted* Stackelberg strategies, and a Stackelberg strategy is optimal-restricted if for a social optimum solution OPT, the strategy assigned to the coordinated players coincides with the

one they adopt in OPT. However, the social optimum solution cannot be obtained in polynomial time, due to the NP-Hardness of the problem. We thus find an approximate solution for the problem when all players are coordinated to approach the social optimum OPT. We then use the obtained approximate solution OPT' to guide the strategies of the coordinated players. We refer this method as *approximation-restricted* strategy.

Our idea is to design an approximation algorithm with a probable approximation ratio for the service caching problem in an MEC network with non-selfish network service providers. In the following, we reduce the problem to a Generalized Assignment Problem (GAP). We describe the approximation-restricted Stackelberg strategy, and analyze the approximation ratio of the approximate solution and performance of the approximation-restricted Stackelberg strategy, in the rest of this section.

For the sake of clarity, we first describe the GAP problem [34]. Given n items and m knapsacks, with each item itm_j can be assigned to a knapsack bin_i at a cost of c_{ij}. The weight of the item is w_{ij} if it is assigned to knapsack bin_i. The accumulative weight of the items that are assigned to bin_i cannot exceed its capacity CAP_i. The objective of the GAP problem is to assign the items to the given knapsacks such that the total assignment cost is minimized.

B. An approximation algorithm for the problem with non-selfish players

We now reduce the problem of service caching in an MEC network with non-selfish players into the GAP problem. It must be mentioned that the difference between the service caching problem and the GAP problem is the cost model. The cost of caching an instance of service in a cloudlet is related to the 'congestion' of the cloudlet, i.e., the number of cached service instances. However, in the GAP problem the cost of assigning an item to a knapsack only depends on the item itself. Reducing the service caching problem thus is to how to map the congestion-aware cost model to the flat cost model in the GAP problem.

Our basic idea is to split each cloudlet into a set of *virtual cloudlets*, with each virtual cloudlet being restricted to be able to only cache a single service instance. The rationale is to ignore the 'congestion' aspect in the cost model first, and consider it later. Specifically, let a_{max} and b_{max} be the maximum demands of computing and bandwidth resources of a service SV_l, i.e., $a_{max} = \arg\max_{sp_l \in N}(a_l)$ and $b_{max} = \arg\max_{sp_l \in N}(b_l)$. Similarly, $a_{min} = \arg\min_{sp_l \in N}(a_l)$ and $b_{min} = \arg\max_{sp_l \in N}(b_l)$. $\frac{a_{max}}{a_{min}}$ and $\frac{a_{max}}{a_{min}}$ usually are given fixed constants. Please note that such ratios largely depends on the number of service types offered by the network service providers, which can be obtained from historical information of services. We then split each cloudlet CL_i into

$$n_i = \min\Big\{\lfloor\frac{C(CL_i)}{a_{max}}\rfloor, \lfloor\frac{B(CL_i)}{b_{max}}\rfloor\Big\} \tag{7}$$

virtual cloudlets with each virtual cloudlet being able to cache a single service SV_l. Let $\{CL'_{1,i}, \cdots, CL'_{k,i}, \cdots, CL'_{n_i,i}\}$ be the set of virtual cloudlets for cloudlet CL_i. Each virtual cloudlet CL'_k has a capacity $\max\{a_{max}, b_{max}\}$ such that any service in N can be cached in it.

We now reduce the problem into the GAP problem, by considering each virtual cloudlet as a knapsack, with its capacity being set to $\max\{a_{max}, b_{max}\}$. Clearly, each virtual cloudlet maximally can cache

$$n'_{max} = \max\Big\{\frac{\max\{a_{max}, b_{max}\}}{a_{min}}, \frac{\max\{a_{max}, b_{max}\}}{b_{min}}\Big\}. \tag{8}$$

services in N. The cost of caching a service in virtual cloudlet $CL'_{k,i}$ is set to

$$\alpha_i + \beta_i + c_l^{ins} + c_i^{bdw}, \tag{9}$$

which means that the contribution of other services in $CL'_{k,i}$ is not considered. That is, the cost of using resource by SV_l in virtual cloudlet $CL'_{k,i}$ is solely depending on the service itself and the location.

Now the problem is reduced into the GAP problem, we then solve the GAP problem by adopting the approximation algorithm in [34]. The obtained solution assigns each service to a virtual cloudlet. This however is not a feasible solution to the original service caching problem. To make it feasible, we assign all the services that are assigned in the virtual cloudlets in $\{CL'_{1,i}, \cdots, CL'_{k,i}, \cdots, CL'_{n_i,i}\}$ to cloudlet CL_i.

The detailed steps of the proposed algorithm are described in algorithm 1.

Algorithm 1 `Appro`

Input: An MEC network G and a number of network service providers wishing to cache their services in G.

Output: A cloudlet to cache each service SV_l of each network service provider.

1: Split each cloudlet into a set of n_i virtual cloudlets, with each virtual cloudlet having the ability of caching only a limited number of services;
2: Consider each virtual cloudlet as a knapsack in the GAP problem [34], and use the cost function that does not incorporate the 'congestion', i.e., $\alpha_i + \beta_i + c_l^{ins} + c_i^{bdw}$;
3: Invoke the approximation algorithm for the GAP problem in [34];
4: Move all the network service providers that are assigned to virtual cloudlets in $\{CL'_{1,i}, \cdots, CL'_{k,i}, \cdots, CL'_{n_i,i}\}$ to cloudlet CL_i;

C. The Stackelberg strategy of the proposed mechanism

We now describe the Stackelberg strategy given the approximate solution to the service caching problem with non-selfish network service providers. Some network service providers in the mobile service market are selfish. To avoid performance degradation due to selfish behavior, we coordinate a subset of players in N by assigning strategies in the approximate solution to them, that is we restrict our attention to an approximation-restricted Stackelberg strategy. This means that the coordinated players will make decisions according to the approximate solution, while the rest uncoordinated players will selfishly select cloudlets that would minimize their own cost.

Let ξ be the percentage of the network service providers that are coordinated by the leader in the mobile service market (i.e., the infrastructure provider), where the $0 < \xi < 1$. The proposed Stackelberg strategy consists of two steps. In the first step, a number of $\lfloor \xi|N| \rfloor$ network service providers are

selected to be coordinated by the infrastructure provider. Since the services of network service providers have different resource demands, they thus have different impacts on the social cost. Therefore, to enlarge the influence of coordinated network service providers, we select $\lfloor \xi |N| \rfloor$ network service providers that incur the highest cost of caching their services. We refer this strategy as Largest Cost First (LCF). The second step of the Stackelberg game is to allow the rest $(1-\xi)|N|$ network service providers selfishly select cloudlets that incur the lowest cost for them. The proposed Stackelberg strategy is shown in algorithm 2.

Algorithm 2 LCF

Input: An MEC network G and a number of network service providers wishing to cache their services in the cloudlets of the cloud

Output: A cloudlet to cache each service SV_l of each network service provider.

1: Find an approximate solution of the service caching problem in an MEC network with non-selfish players according to algorithm Appro;
2: Find a number of $\xi|N|$ network service providers with the maximum cost of caching their services in a cloudlet of the MEC network;
3: Let N_s be the set of such coordinated network service providers;
4: **for** each network service provider $sp_l \in N_s$ **do**
5: Use the location in the approximate solution to cache its service;
6: **for** each network service provider $sp_l \in N \setminus N_s$ **do**
7: Use the location that could incur a minimum cost for itself to cache its service SV_l;

D. Analysis

We now analyze the performance of the proposed algorithms Appro and LCF in the following.

Lemma 1: The solution obtained by algorithm Appro is feasible for the service caching problem in an MEC network with non-selfish network service providers, assuming that the resource capacities are far greater than the maximum resource demands of the services of service providers in N.

Proof Showing the feasibility of algorithm Appro is to show that each service SV_l of network service provider is cached into a cloudlet, and the computing and bandwidth capacities of each cloudlet $C(CL_i)$ and $B(CL_i)$ are met.

In algorithm Appro, we divide each cloudlet CL_i into a list of virtual cloudlets, i.e., $\{CL'_{1,i}, \cdots, CL'_{k,i}, \cdots, CL'_{n_i,i}\}$. The services of network service providers N then are assigned to those virtual cloudlets using the approximate solution to the GAP problem. We then adjust the obtained solution, by assigning all the services that are assigned to virtual cloudlets in $\{CL'_{1,i}, \cdots, CL'_{k,i}, \cdots, CL'_{n_i,i}\}$ to cloudlet CL_i. Therefore, each service is cached only to a single cloudlet.

Let us now show that both computing and bandwidth capacities of cloudlets are met. For the computing capacity constraint $C(CL_i)$ of each cloudlet CL_i, it is clear that each cloudlet is divided into a number of virtual cloudlets according to the maximum demand (either computing or bandwidth) of a service, this guarantees that any virtual cloudlet has enough resource to cache a service. Considering that each knapsack in the GAP problem has a capacity that is not violated by its solution, the capacity of each virtual cloudlet is not violated as well. After obtaining the solution due to algorithm in [34], we move the services cached into virtual cloudlet $CL'_{k,i}$ to cloudlet CL_i. Since the computing capacity of $CL'_{k,i}$ is not violated, the computing capacity of each cloudlet CL_i is not violated. Similar derivation can be performed for the bandwidth capacity of each cloudlet CL_i. Therefore, the solution due to algorithm Appro is feasible.

Lemma 2: The approximation ratio of the solution due to algorithm Appro is $2 \cdot \delta \cdot \kappa$, where $\delta = \frac{C(CL_i)}{a_{max}}$ and $\kappa = \frac{B(CL_i)}{b_{max}}$.

Proof We have showed the approximation ratio of the proposed algorithm Appro. Denote by C' the obtained social cost for the problem with non-selfish network service providers under cost function $\alpha_i + \beta_i + c_l^{ins} + c_i^{bdw}$. Let OPT' be the optimal cost. Similarly, let C be the social cost due to algorithm Appro and let OPT be the optimal cost for the service caching problem in a two-tiered MEC network. We know that the approximation ratio due to algorithm in [34] for the GAP problem is 2. This means that we have

$$C'/OPT' = 2. \tag{10}$$

We then find the relation between the social cost C' under cost function $\alpha_i + \beta_i + c_l^{ins} + c_i^{bdw}$ and the social cost under the cost function in the original problem. It must be mentioned that social cost C' does not consider the cost due to 'congestion' of a cloudlet. After invoking the approximation algorithm due to [34], there are at most n'_{max} services that are assigned to each virtual cloudlet $CL'_{k,i}$, since the capacity of each virtual cloudlet is set to $max\{a_{max}, b_{max}\}$. Recall that there are at most n_i virtual cloudlets for each cloudlet CL_i. All the services assigned to these n_i will be moved to a single cloudlet CL_i. The cost due to such movement increases as it increases the congestion of cloudlet CL_i. Clearly, there can be at most $n_i \cdot n'_{max}$ services that are involved in the movement, which means that

$$\begin{aligned} C &\leq \sum\nolimits_{CL_i \in \mathcal{CL}} (n_i \cdot n'_{max} \cdot (\alpha_i + \beta_i) + c_l^{ins} + c_i^{bdw}) \\ &= n_i \cdot n'_{max} \cdot \sum\nolimits_{CL_i \in \mathcal{CL}} (\alpha_i + \beta_i + \frac{c_l^{ins} + c_i^{bdw}}{n_i \cdot n'_{max}}) \end{aligned}$$

Clearly we know that $n_i > 1$ and $n'_{max} > 1$. We then have

$$\begin{aligned} C &< n_i n'_{max} \sum_{CL_i \in \mathcal{CL}} (\alpha_i + \beta_i + c_l^{ins} + c_i^{bdw}) \\ &= n_i \cdot n'_{max} \cdot C'. \end{aligned} \tag{11}$$

Assuming that $\frac{C(CL_i)}{a_{max}}$ and $\frac{B(CL_i)}{b_{max}}$ are small constants, let $\frac{C(CL_i)}{a_{max}} = \delta$ and $\frac{B(CL_i)}{b_{max}} = \kappa$. We get

$$C < \delta \cdot \kappa \cdot C' = 2 \cdot \delta \cdot \kappa \cdot OPT', \tag{12}$$

due to Eq. (10). Since OPT considers the congestion of each cloudlet, we have $OPT' < OPT$, which means that

$$C < 2 \cdot \delta \cdot \kappa \cdot OPT. \tag{13}$$

The approximation ratio of the proposed algorithm Appro thus is $2 \cdot \delta \cdot \kappa$.

We now analyze the performance of the proposed Stackelberg

strategy LCF in the following lemmas and theorem.

Lemma 3: There exists at least one NE of the proposed Stackelberg game.

Proof The proposed Stackelberg game deals with a set of coordinated service providers and another set of selfish service providers. Recall that the coordinated service providers follow the decisions produced by algorithm Appro, and their decisions will not be affected by the selfish network service providers. We only need to show that a NE could be achieved for the selfish network service providers. Recall that the cost of caching a service in cloudlet is calculated by $\alpha_i + \beta_i + c_l^{ins} + c_i^{bdw}$, which is an affline function. Following existing results in [33], affline congestion games admits at least one NE. This concludes the proof.

We now establish the PoA of the proposed Stackelberg game under the LCF strategy in the following theorem.

Theorem 1: Denote by ζ the obtained approximation solution due to algorithm Appro, let s be any approximation-restricted Stackelberg caching of services of an MEC network, and let ω be the service caching that leads to the worst cost due to selfish behavior of the rest $(1-\xi)|N|$ network service providers. We then have the PoA of the proposed strategy is $\frac{2\delta\kappa}{1-v}(\frac{1}{4v}+1-\xi)$ with $v \in (0,1)$.

Proof Denote by N_s be the set of coordinated network service providers determining the approximation-restricted Stackelberg strategy. Let ϕ be the worst pure NE induced by s. We then know that $\omega = s + \phi$.

Recall that ϕ is a NE with respect to the cost function $\alpha_i|\sigma_i| + \beta_i|\sigma_i| + c_l^{ins} + c_i^{bdw}$ for each of the $(1-\xi)|N|$ network service providers. We have

$$\begin{aligned}&(\alpha_{\phi(l)} + \beta_{\omega(l)})|\sigma_{\omega(l)}| + c_l^{ins} + c_{\omega(l)}^{bdw}\\ &< \alpha_{\zeta(l)}|\sigma_{\zeta(l)}+1| + \beta_{\zeta(l)}|\sigma_{\zeta(l)}+1| + c_l^{ins} + c_{\zeta(l)}^{bdw}, \quad (14)\end{aligned}$$

where $\phi(l)$ and $\zeta(l)$ are the cloudlets that are used to cache service SV_l of network service provider sp_l of the NE and the approximate solution due to algorithm Appro.

If we sum up the inequalities (14) for all uncoordinated network service providers in $N \setminus N_s$, we get

$$\begin{aligned}&\sum\nolimits_{sp_l \in N\setminus N_s} (\alpha_{\omega(l)} + \beta_{\omega(l)})|\sigma_{\omega(l)}| + c_l^{ins} + c_{\omega(l)}^{bdw}\\ &< \sum_{sp_l \in N\setminus N_s} (\alpha_{\zeta(l)} + \beta_{\zeta(l)})|\sigma_{\zeta(l)}+1| + c_l^{ins} + c_{\zeta(l)}^{bdw}. \quad (15)\end{aligned}$$

With a little abuse of using symbols, we use $\sigma_{i,\omega}$ to denote the number of service providers that are assigned to cloudlet CL_i. We have

$$\begin{aligned}&\sum\nolimits_{CL_i \in \mathcal{CL}} \sigma_{i,\phi} \cdot \big((\alpha_i + \beta_i)|\sigma_{i,\omega}| + c_l^{ins} + c_i^{bdw}\big)\\ &< \sum_{CL_i \in \mathcal{CL}} (\sigma_{i,\zeta} - \sigma_{i,s}) \cdot \big((\alpha_i + \beta_i)|\sigma_{i,\omega}+1| + c_l^{ins} + c_i^{bdw}\big). \quad (16)\end{aligned}$$

Adding the cost $\sigma_{i,s}\big((\alpha_i+\beta_i)\sigma_{i,\omega} + c_l^{ins} + c_i^{bdw}\big)$ of coordinated players to both sides of the above inequality, we obtain

$$\begin{aligned}C(\omega) &= \sum_{CL_i \in \mathcal{CL}} (\sigma_{i,\phi} + \sigma_{i,s}) \cdot \big((\alpha_i + \beta_i)\sigma_{i,\omega} + c_l^{ins} + c_i^{bdw}\big)\\ &< \sum_{CL_i \in \mathcal{CL}} \big((\alpha_i + \beta_i)\sigma_{i,\omega}\sigma_{i,\zeta} + c_l^{ins}\sigma_{i,\zeta} + c_i^{bdw}\sigma_{i,\zeta}\\ &\qquad + (\sigma_{i,\zeta} - \sigma_{i,s})(\alpha_i + \beta_i)\big). \quad (17)\end{aligned}$$

Given the inequality $xy \le vx^2 + \frac{1}{4v}y^2$ that is valid for all $x, y \in \mathbb{R}$ and $v \in (0,1)$, we then derive the following inequality

$$\begin{aligned}&\sum\nolimits_{CL_i \in \mathcal{CL}} \big((\alpha_i + \beta_i)\sigma_{i,\omega}\sigma_{i,\zeta} + c_l^{ins}\sigma_{i,\zeta} + c_i^{bdw}\sigma_{i,\zeta}\big)\\ &< v \sum_{CL_i \in \mathcal{CL}} (\alpha_i + \beta_i)(\sigma_{i,\omega})^2 + \frac{1}{4v} \sum_{CL_i \in \mathcal{CL}} (\alpha_i + \beta_i)(\sigma_{i,\zeta})^2\\ &\qquad + \sum_{CL_i \in \mathcal{CL}} (c_l^{ins} + c_i^{bdw})\sigma_{i,\zeta}\\ &= vC(\omega) - v \sum_{CL_i \in \mathcal{CL}} (c_l^{ins} + c_i^{bdw})(\sigma_{i,\omega}) +\\ &\quad \frac{1}{4v} \sum_{CL_i \in \mathcal{CL}} (\alpha_i + \beta_i)(\sigma_{i,\zeta})^2 + \sum_{CL_i \in \mathcal{CL}} (c_l^{ins} + c_i^{bdw})\sigma_{i,\zeta}\\ &= vC(\omega) + \frac{1}{4v} vC(\zeta) + \sum_{CL_i \in \mathcal{CL}} (c_l^{ins} + c_i^{bdw})\sigma_{i,\zeta}\\ &- \big(v + \frac{1}{4v}\big) \sum_{CL_i \in \mathcal{CL}} (c_l^{ins} + c_i^{bdw})\sigma_{i,s}\\ &- \frac{1}{4v} \sum_{CL_i \in \mathcal{CL}} (c_l^{ins} + c_i^{bdw})(\sigma_{i,\zeta} - \sigma_{i,s}), \quad \text{since } \sigma_{i,\omega} \ge \sigma_{i,s}\\ &\le vC(\omega) + \frac{1}{4v} C(\zeta) +\\ &\quad (1 - \frac{1}{4v}) \sum_{CL_i \in \mathcal{CL}} (c_l^{ins} + c_i^{bdw})(\sigma_{i,\zeta} - \sigma_{i,s}),\\ &\qquad \text{since } v + \frac{1}{4v} \ge 1.\end{aligned}$$

Combining inequalities (17) and (18), we have

$$\begin{aligned}&(1-v)C(\omega)\\ &\le \frac{1}{4v} C(\zeta) + (1 - \frac{1}{4v}) \sum_{CL_i \in \mathcal{CL}} (c_l^{ins} + c_i^{bdw})(\sigma_{i,\zeta} - \sigma_{i,s})\\ &\qquad + \sum\nolimits_{CL_i \in \mathcal{CL}} (\sigma_{i,\zeta} - \sigma_{i,s})(\alpha_i + \beta_i).\\ &\le \frac{1}{4v} C(\zeta) + \sum\nolimits_{CL_i \in \mathcal{CL}} (c_l^{ins} + c_i^{bdw})(\sigma_{i,\zeta} - \sigma_{i,s})\\ &\qquad + \sum\nolimits_{CL_i \in \mathcal{CL}} (\alpha_i + \beta_i)(\sigma_{i,\zeta} - \sigma_{i,s})\\ &\le \frac{1}{4v} C(\zeta) + \sum\nolimits_{CL_i \in \mathcal{CL}} \Big((\alpha_i + \beta_i)(\sigma_{i,\zeta} - \sigma_{i,s})^2\\ &\quad + (c_l^{ins} + c_i^{bdw})(\sigma_{i,\zeta} - \sigma_{i,s})\Big),\\ &\text{since } (\sigma_{i,\zeta} - \sigma_{i,s}) \le (\sigma_{i,\zeta} - \sigma_{i,s})^2 \text{ for non-negative integers}\\ &\le \frac{1}{4v} C(\zeta) + \sum_{CL_i \in \mathcal{CL}} \big((\alpha_i + \beta_i)\sigma_{i,\zeta} + (c_l^{ins} + c_i^{bdw})\big)\\ &(\sigma_{i,\zeta} - \sigma_{i,s}), \text{since } (\sigma_{i,\zeta} - \sigma_{i,s})^2 \le \sigma_{i,\zeta}(\sigma_{i,\zeta} - \sigma_{i,s})\end{aligned}$$

$$\leq \frac{1}{4v}C(\zeta) + (1-\xi)C(\zeta) \leq 2\delta\kappa(\frac{1}{4v} + 1 - \xi)OPT. \quad (18)$$

The PoA of the proposed approximation-restricted Stackelberg strategy thus is $\frac{2\delta\kappa}{1-v}(\frac{1}{4v} + 1 - \xi)$.

IV. Experiments

In this section, we evaluate the performance of the proposed algorithms.

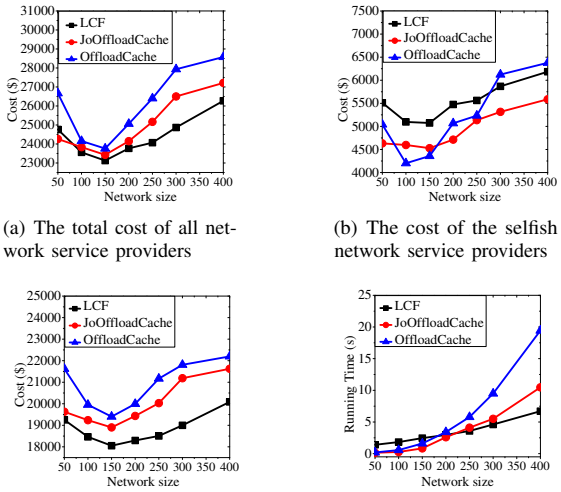
A. Parameter settings

We consider a two-tiered cloud network with its size varying from 50 to 400 switch nodes and 5 remote data centers, where each network topology is generated using GT-ITM [9]. The number of cloudlets in the mobile edge network is set to 10% of the network size, which are randomly distributed in the network edge. We also use a real network topologies AS1755 from [29]. The number of VMs provided by each cloudlet/data center is randomly generated from [15, 30]. The bandwidth capacity of each VM is drawn from the range of [10Mpbs, 100Mbps]. The costs of transmitting and processing 1 GB of data are set within [$0.05, $0.12] and [$0.15, $0.22], respectively. The traffic volume of each request is randomly drawn from $[10, 200]$ Megabytes. The data volume of each service caching request is varied from 1 Gigabyte (GB) to 5 GB. The values for α_i and β_i of each cloudlet CL_i are randomly drawn in the range of $[0, 1]$. The data volume of consistency updating from a cached instance to the original instance of a service is set to 10% of the service's data volume. The running time of each algorithm is obtained based on a machine with a 3.70GHz Intel i7 Hexa-core CPU and 16 GiB RAM. Unless otherwise specified, these parameters will be adopted in the default setting.

We compare the proposed mechanism with the following algorithms: (1) the first benchmark is the algorithm in [23], which provides efficient solution to the problem of joint service caching and task offloading in MEC networks. One difference of this study with ours is that we consider a two-tier MEC network that jointly considers task offloading, service caching, and data updating. The data updating however is not considered in [23]. In addition, since the algorithm in [23] does not consider the mobile service market with multiple network service providers, we consider that each network service provider runs the algorithm in [23], without communicating with each other. For simplicity, we refer such joint offloading and caching algorithms as `JoOffloadCache`; (2) the second benchmark is a greedy algorithm, in which each network service provider considers offloading and service placement/cache separately [20]. Specifically, the algorithm simply selects the cloudlets for each request that could achieve an optimal offloading cost. Based on the assignment, services are instantiated in the cloudlets (or close cloudlets) with their requests. This algorithm is referred to as `OffloadCache`.

B. Simulations

We first evaluate the performance of mechanism `LCF` with algorithms `JoOffloadCache` and `OffloadCache`, by varying the network size from 50 to 400 with a number of 100 network service providers and fixing $(1-\xi)$ to 0.3. Fig. 2 shows the results in terms of the social cost, the cost of coordinated network service providers, the cost of selfish network service providers, and the running time. From Fig. 2 (a), we can see that `LCF` consistently delivers the minimum social cost among the three algorithms, while algorithm `OffloadCache` has the highest social cost. The reason is that algorithm `LCF` coordinates a number of coordinated network service providers to avoid significant performance degradation while algorithms `JoOffloadCache` and `OffloadCache` allows each network service provider makes decisions selfishly. In addition, `LCF` optimizes the data uploading cost while the rest two algorithms do not consider such costs. The running times of algorithms `LCF`, `JoOffloadCache`, and `OffloadCache` are shown in Fig. 2 (d).



(a) The total cost of all network service providers

(b) The cost of the selfish network service providers

(c) The cost of the coordinated network service providers

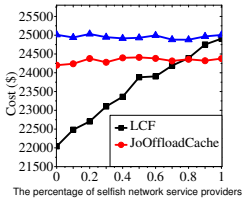
(d) The running times

Fig. 2. Algorithm performance in GT-ITM generated networks with sizes varied from 50 to 400.

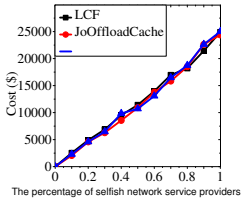
We then study the impact of the ratio of the number of selfish network service providers and the total number, i.e., $1-\xi$, by fixing the network size to 250 and varying the value of $(1-\xi)$ from 0 to 1. Results are shown in Fig. 3. From Fig. 3 (a), we can see that the social cost of all network service providers by algorithm `LCF` increases with the growth of $(1-\xi)$. The reason is that with more and more selfish players, less network service providers can be coordinated, making the obtained cost deviating further from the social optimum. This can also be evidenced by Figures 3 (a) and (b). We can also see from Fig. 3 (a) that the total cost by **LCF** is significantly smaller than algorithms `JoOffloadCache` and `OffloadCache` until $(1-\xi)$ is increased to 0.8. It must be mentioned this is a relative high percentage of selfish network service providers. Also, algorithms `JoOffloadCache` and `OffloadCache` do not consider selfishness of network service providers at all.

C. Implementations in a test-bed

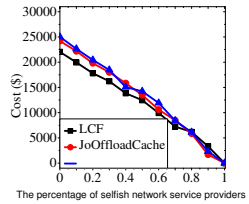
Testbed settings: We build a test-bed consisting of both an underlay with hardware switches and an overlay with virtual



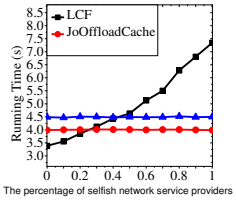
(a) The total cost of all network service providers



(b) The cost of the selfish network service providers

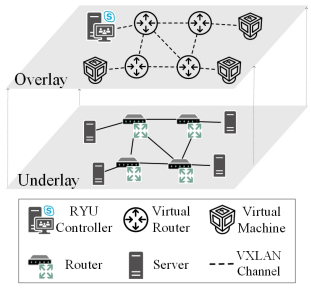


(c) The cost of the coordinated network service providers



(d) The running times

Fig. 3. The impact of $(1-\xi)$ on the algorithm performance in a GT-ITM generated network with size 250.



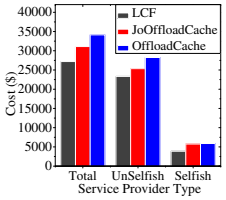
(a) The underlay and overlay of the test-bed

(b) The physical deployment of the hardware switches

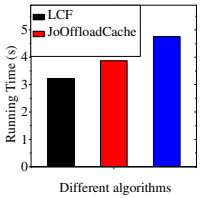
Fig. 4. A test-bed with both hardware switches and virtual resources.

switches, as shown in Fig. 4. The physical underlay consists of five switches, i.e., Huawei S5720-32C-HI-24S-AC, H3C S5560-30S-EI, Ruijie RG-5750C-28Gt4XS-H, CISCO 3750X-24T, and Centec aSW1100-48T4X. Each switch is connected to at least two other switches that can guarantee the network data can still be transmitted if one switch is down. It also has five servers with i7-8700 CPU and 16G RAM. Netconf and SNMP protocols are used to manage the switches and the links that interconnect them. Using VXLAN, we virtualize an overlay network with a number of Open vSwitch (OVS) [15] nodes and VMs. The underlay can be seen as a resource pool with computing and bandwidth resources which can be used to build overlay networks. In the overlay, we create VMs and connect them with the OVS. The overlay network is built following the real topology AS1755. Its OVS nodes and VMs are controlled by a Ryu [17] controller. The proposed algorithms are implemented as Ryu applications. All the rest settings are the same as the simulations in the previous subsection.

Performance results: We investigate the performance of the algorithms in the test-bed, by fixing $(1-\xi)$ to 0.3. The results are shown in Fig. 5, from which we can see that algorithm



(a) The total cost of all network service providers



(b) The running times

Fig. 5. Performance in the testbed with both physical underlay and virtual overlay.

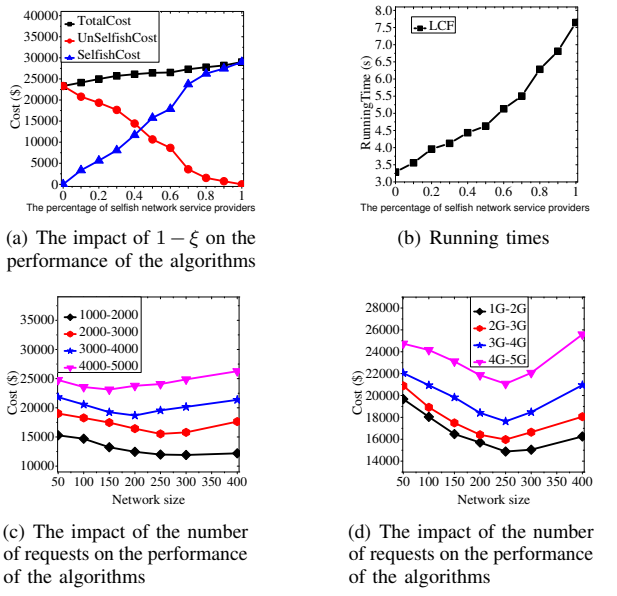
`LCF` has a much lower social cost than that of algorithms `JoOffloadCache` and `OffloadCache`.

We then study the impact of various parameters on the performance of algorithms `LCF`, `JoOffloadCache` and `OffloadCache` in the test-bed. Fig. 6 (a) shows the impact of $1-\xi$, from which we can see that the social cost of algorithm `LCF` increases with the growth of $(1-\xi)$. This is because the higher percentage of selfish network service providers has a higher impact of worsening the social optimum, considering that each network service provider only cares about its own cost instead of the social cost. Fig. 6 (c) illustrates the impact of the number of service caching requests on the performance of the algorithms. From the figure, we can see that a higher number of requests means a higher total cost. In addition, we can see that the total cost decreases first when the network size is increased from 50 to 200, and then increases afterwards. The rationale behind is that with the growth of network sizes more services have higher opportunities to be cached into cloudlets with lower costs. When the network size keeps growing, services will also have a higher probability being assigned to cloudlets further to remote clouds, thereby increasing the cost due to bandwidth resource consumption. Fig. 6 (d) depicts the impact of the amount of update data between cloudlets and remote clouds on the algorithm performance. It can be seen that a larger amount of to-be-updated data incurs a higher total cost due to the higher bandwidth consumption.

We finally investigate the impact of maximum demands of computing and bandwidth resource demands, i.e., a_{max} and b_{max}, on the performance of algorithms `LCF`, `JoOffloadCache` and `OffloadCache` in the test-bed. Fig. 7 (a) shows the impact of a_{max}, from which we can see that the obtained cost is increasing with the growth of a_{max}. The rationale behind is that in the Stackelberg game each cloudlet is partitioned into n_i $(= \{\lfloor \frac{C(CL_i)}{a_{max}} \rfloor, \lfloor \frac{B(CL_i)}{b_{max}} \rfloor\})$ virtual cloudlets, and the number of virtual cloudlets is decreasing if a_{max} grows. This means that the algorithm has a higher probability to reject some requests in the adjustment procedure. Also, this verifies the correctness of Lemma 2. Similar trends can be found for b_{max} as shown in Fig. 7 (b).

V. Related Work

With the emerging of mobile edge computing, various service providers are adopting the notion of caching their services (originally in remote data centers) to locations of a mobile



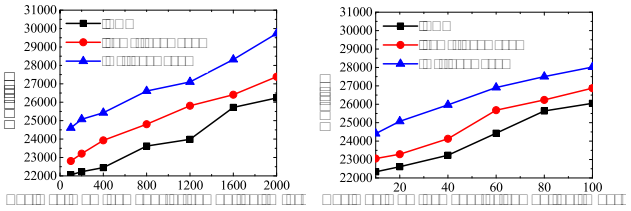
(a) The impact of $1-\xi$ on the performance of the algorithms

(b) Running times

(c) The impact of the number of requests on the performance of the algorithms

(d) The impact of the number of requests on the performance of the algorithms

Fig. 6. Results of the impact of the ratio of the number of coordinated network service providerrs and selfish ones in the test-bed.



(a) The impact of a_{max} on the performance of the algorithms

(b) The impact of b_{max} on the performance of the algorithms

Fig. 7. Results of the impact of maximum demands of computing and bandwidth resource demand in the test-bed.

edge-cloud within the proximity of users. This approach has an equal importance with the task offloading from mobile devices to edge-clouds. The related studies on service caching can be either about computation offloading or on service placement.

For the studies on computation offloading, most studies focus on the user side by deciding which tasks should be offloaded to which cloudlet [7], [32], [27], [39], [40], such that the user computing capacity is saved or the offloading cost is minimized. These works either do not consider service placement/caching or ignore the impact of congestion of cloudlets during task offloading. For example, Jia et al. [13] considered a task offloading problem in augmented reality games with an aim to minimize the delay suffered by end users. Yu et al. [27] considered an application provisioning and data routing problem, with bandwidth and delay guarantees for data sources, i.e., each data source should receive bandwidth that meets its data generation rate, and the transmission delay of each channel should be within the delay tolerance of the application. Misra *et al.* [32] recently studied the task offloading problem in a software-defined network, where Internet-of-Things (IoT) devices are connected to fog computing nodes by multi-hop IoT access-points (APs). Both exact and efficient heuristics are proposed. Zhou *et al.* [40] studied the joint task offloading and scheduling optimization problem, by considering wireless network connections and mobile device mobility.

Existing studies on service placement focus on placing a given set services to cloudlets of an MEC network with limited resources [6], [10], [28], [13], [30], [18], [22], [23], [27]. Most of these studies however do not consider a service market with both infrastructure providers and service providers. In addition, they do not consider a two-tiered cloud network with both data centers and cloudlets. For example, Ascigil et al. [2] proposed a centralized algorithm for the service placement problem in mobile edge-clouds, by adopting multiple criteria, such as least recently used, strictest deadline first, etc. Xu *et al.* [37] investigated the interactions among content service providers under a novel 'sponsored content' scheme, by proposing a Stackelberg game. The delay and cost issues however are not considered, and their method cannot be directly applied to the service caching problem. Farris *et al.* [6] devised methods for service replication and migration for mobile users with an objective to minimize the degradation of the quality of experience (QoE) degradation and the cost of service replication. Hou *et al.* [10] investigated the problem of content caching in mobile networks and devised efficient algorithms to predict content popularity. Similarly, Jiang et al. [28] considered a content caching and delivery problem by placing popular contents in base stations and user equipments, such that the access latency of users is minimized, subject to the capacity constraints of base stations. Xu *et al.* [23] investigated a problem of service placement in MEC-enabled cellular networks, and proposed algorithms based on Lyapunov optimization and Gibbs sampling, to reduce computation latency for end users. Wang et al. [22] presented a problem of provisioning a social virtual reality application in MEC networks to serve a number of users to minimize the total cost for placing services in cloudlets and provisioning placed services to users. Zhang *et al.* [38] presented a dynamic service placement problem in a distributed cloud, they aim to minimize the cost for server allocation and reconfiguration, subject to the constraints of delay and resource demands. Li *et al.* [30] considered a cell caching for mobile networks, each cell (base station) can cache popular contents for minimizing delay experienced of users. However, they did not consider cache cost [10], [28], or neglected hierarchical framework of the network [28], or did not cover updating activities between local edge servers and remote datacenters [30].

VI. Conclusion

In this paper, we investigated the service caching problem in a two-tiered MEC network of a mobile service market. The market consists of an infrastructure provider and multiple network service providers. We developed a novel approximation-restricted optimization framework that could guarantee the stable and near-optional operation of the mobile market. Within the framework, we first designed an approximation algorithm with an approximation ratio for the problem with non-selfish players. We also devised an efficient, stable Stackelberg

congestion game with a provable Price of Anarchy (PoA) as the second part of the framework. We evaluated the performance of our mechanism by simulations. We also performed a set of real experiments in a real test-bed.

ACKNOWLEDGMENTS

The work of Zichuan Xu, Qiufen Xia, and Guowei Wu is partially supported by the National Natural Science Foundation of China (Grant No. 61802048, 61802047, 61772113, and 61872053), the fundamental research funds for the central universities in China (Grant No. DUT17RC(3)061, DUT17RC(3)070, DUT19RC(4)035, and DUT19GJ204), DUT-RU Co-Research Center of Advanced ICT for Active Life, and the "Xinghai Scholar Program" in Dalian University of Technology, China. The work by Weifa Liang is supported by the Australian Research Council Discovery Project (Grant No. DP200101985).

REFERENCES

[1] Azure Pricing. *https://azure.microsoft.com/en-in/pricing/*, accessed in June 2019.
[2] O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, and G. Pavlou. On Uncoordinated Service Placement in Edge-Clouds. *Proc. of CloudCom*, IEEE, 2017.
[3] M. Chen, B. Liang, and M. Dong. Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point. *Proc. of INFOCOM*, IEEE, 2019.
[4] A. Drucker, F. Kuhn, and R. Oshman. On the power of the congested clique model. *Proc. of PODC*, ACM, 2014.
[5] N. Eshraghi and B. Liang. Joint offloading decision and resource allocation with uncertain task computing requirement. *Proc. of INFOCOM*, IEEE, 2019.
[6] I. Farris, T. Taleb, M. Bagaa, and H. Flick. Optimizing Service Replication for Mobile Delay-sensitive Applications in 5G Edge Network. *Proc. of ICC*, IEEE, 2017.
[7] B. Gao, Z. Zhou, F. Liu, and Fei Xu: Winning at the starting line: Joint network selection and service placement for mobile edge computing. *Proc. of INFOCOM*, IEEE, 2019.
[8] Google Cloud Pricing. *https://cloud.google.com/pricing/*, accessed in June 2019.
[9] http://www.cc.gatech.edu/projects/gtitm/.
[10] T. Hou, G. Feng, S. Qin, and W. Jiang. Proactive Content Caching by Exploiting Transfer Learning for Mobile Edge Computing. *Proc. of Globecom*, IEEE, 2017.
[11] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, and Z. Chen. Quantifying the impact of edge computing on mobile applications. *Proc. of ACM APSys*, ACM, 2016.
[12] J. W. Hegeman, G. Pandurangan, S. V. Pemmaraju, V. B. Sardeshmukh, and M. Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and MST. *Proc. of PODC*, ACM, 2015.
[13] M. Jia, and W. Liang. Delay-Sensitive Multiplayer Augmented Reality Game Planning in Mobile Edge Computing. *Proc. of MSWIM*, ACM, 2018.
[14] OpenFlow. https://www.opennetworking.org.
[15] Open vSwtich. *https://www.openvswitch.org*
[16] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas. Joint service placement and request routing in multi-cell mobile edge computing networks. *Proc. of INFOCOM*, IEEE, 2019.
[17] Ryu SDN Controller. *https://osrg.github.io/ryu/*
[18] L. Pu, L. Jiao, X. Chen, L. Wang, Q. Xie, and J. Xu. Online Resource Allocation, Content Placement and Request Routing for Cost-Efficient Edge Caching in Cloud Radio Access Networks. *IEEE Journal on Selected Areas in Communications*, Vol. 36, No. 8, pp. 1751-1767, IEEE, 2018.
[19] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili. Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges. *IEEE Communications Magazine*, Vol. 55, No. 4, pp. 54–61, IEEE, 2017.
[20] J. R. Thomsen, M. L. Yiu and C. S. Jensen. Effective caching of shortest paths for location-based services. *Proc. of SIGMOD*, ACM, 2012.
[21] B. W. Wie, T. L. Friesz, and R. L Tobin. Dynamic user optimal traffic assignment on congested multidestination networks. *Transportation Research Part B: Methodological*, 24(6), 431-442.
[22] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser. Service Entity Placement for Social Virtual Reality Applications in Edge Computing. *Proc. of INFOCOM*, IEEE, 2018.
[23] J. Xu, L. Chen, and P. Zhou. Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks. *Proc. of INFOCOM*, IEEE, 2018.
[24] Z. Xu, W. Liang, M. Jia, M. Huang, and G. Mao. Task offloading with network function services in a mobile edge-cloud network. *IEEE Transactions on Mobile Computing*, Vol.18, No.11, pp.2672 – 2685, 2019.
[25] Z. Xu, W. Liang, M. Huang, M. Jia, S. Guo, and A. Galis. Approximation and online algorithms for NFV-enabled multicasting in SDNs. *Proc of ICDCS*, IEEE, 2017.
[26] Z. Xu, L. Zhou, S. Chau, W. Liang, Q. Xia and P. Zhou. Collaborate or separate? Distributed service caching in mobile edge clouds. To appear in *Proc of INFOCOM20*, IEEE, 2020.
[27] R. Yu, G. Xue, and X. Zhang. Application Provisioning in Fog Computing-enabled Internet-of-Things: A Network Perspective. *Proc. of INFOCOM*, IEEE, 2018.
[28] W. Jiang, G. Feng, and S. Qin. Optimal Cooperative Content Caching and Delivery Policy for Heterogeneous Cellular Networks. *IEEE Transactions on Mobile Computing*, Vol. 16, No. 5, pp. 1382-1393, IEEE, 2017.
[29] S. Knight *et al.* The internet topology zoo. *Journal of Selected Areas in Communications*, Vol. 29, pp. 1765–1775, IEEE, 2011.
[30] X. Li, X. Wang, S. Xiao, and V. C. M. Leung. Delay Performance Analysis of Cooperative Cell Caching in Future Mobile Networks. *Proc. of ICC*, IEEE, 2015.
[31] Q. Ma, E. Yeh, and J. Huang. How bad is selfish caching?. *Proc. of MobiHoc*, ACM, 2019.
[32] S. Misra and N. Saha. Detour: Dynamic Task Offloading in Software-Defined Fog for IoT Applications. *IEEE Journal on Selected Areas in Communications*, Vol. 37, No. 5, pp. 1159–1166, IEEE, 2019.
[33] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic game theory*, Cambridge, 2007
[34] D. B. Shmoys and É. Tardos. An approximation for the generalized assignment problem. *Mathematical programming*, Vol. 62, pp. 461–474, North-Holland, 1993.
[35] Q. Sun, S. Ren, and C. Wu. A provably-efficient online algorithm for re-utilizing unused VM resources for edge providers. *Proc. of ICC*, IEEE, 2019.
[36] R. Xia, H. Dai, J. Zheng, R. Gu, X. Wang, and G. Chen. SAFE: Service availability via failure elimination through VNF scaling. *Proc. of ICPP*, IEEE, 2019.
[37] X. Xiong, *et al*. Joint sponsored and edge caching content service market: A game-theoretic approach. *IEEE Transactions on Wireless Communications*, Vol. 18, No. 2, pp.1166–1181, IEEE, 2019.
[38] Q. Zhang, Q. Zhu, M. F. Zhani, and R. Boutaba. Dynamic Service Placement in Geographically Distributed Clouds. *Proc. of ICDCS*, IEEE, 2012.
[39] S. Zang, W. Bao, P. L. Yeoh, B. Vucetic, and Y. Li. Filling two needs with one deed: Combo pricing plans for computing-intensive multimedia applications. *IEEE Journal on Selected Areas in Communications*, Vol. 37, No. 7, pp. 1518–1533, IEEE, 2019.
[40] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya. An online algorithm for task offloading in heterogeneous mobile clouds. *ACM Transactions on Internet Technology*, Vol. 18, No. 2, Article 23, ACM, 2018.
[41] P. Zhou, K. Wang, J. Xu, and D. Wu. Differentially-private and trustworthy online social multimedia big data retrieval in edge computing. *IEEE Transactions on Multimedia*, vol.21, no. 3, pp.539–554, IEEE, 2019.