An Offloading Method Using Decentralized P2P-Enabled Mobile Edge Servers in Edge Computing

Wenda Tang[a,b], Xuan Zhao[a,b], Wajid Rafique[a,b], Lianyong Qi[c,*], Wanchun Dou[a,b,*], Qiang Ni[d]

[a] *State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China*
[b] *Department of Computer Science and Technology, Nanjing University, Nanjing, China*
[c] *School of Information Science and Engineering, Qufu Normal University, Rizhao, China*
[d] *School of Computing and Communications, InfoLab21, Lancaster University, Lancaster LA1 4YW, U.K.*

Abstract

Edge computing has emerged as a promising infrastructure for providing elastic resources in the proximity of mobile users. Owing to resource limitations in mobile devices, offloading several computational tasks from mobile devices to mobile edge servers is the main means of improving the quality of experience of mobile users. In fact, because of the high speeds of moving vehicles on expressways, there would be numerous candidate mobile edge servers available for them to offload their computational workload. However, the selection of the mobile edge server to be utilized and how much computation should be offloaded to meet the corresponding task deadlines without large computing bills are topics that have not been discussed much. Furthermore, with the increasing deployment of mobile edge servers, their centralized management would cause certain performance issues. In order to address these challenges, we firstly apply peer-to-peer networks to manage geo-distributed mobile edge servers. Secondly, we propose a new deadline-aware and cost-effective offloading approach, which aims to improve the offloading efficiency for vehicles and allows additional tasks to meet their deadlines. The proposed approach was validated for its feasibility and efficiency by means of extensive experiments, which are presented in this paper.

Keywords: Edge Computing, Computation Offloading, Decentralization, Deadline, Cost-effectiveness

1. Introduction

Owing to the limitations of computation resources and storage capacity in mobile smart devices, a new computing paradigm known as mobile cloud computing (MCC) has been introduced, enabling mobile devices to offload their tasks to a more powerful and resourceful cloud [1]. The fact that MCC is not suitable for scenarios involving near real-time applications and guaranteeing high quality of service (QoS) has led to the birth of the edge computing paradigm.

According to [2], edge computing is an umbrella concept covering a range of practical schemes for its implementation; for example, cloudlets, fog computing, and mobile edge computing (MEC). While these approaches implement the same principles, they differ in terms of various aspects.

Cloudlets were introduced in [3], with the aim of providing a means for resource-poor mobile devices in particular to offload their computationally intensive tasks to the strongest computational machine encountered in the physical proximity. A cloudlet is a self-maintained, relatively limited-resource cloud located within the close vicinity of mobile users [4]. Mobile devices can access the cloudlet node through a wireless network, while the cloudlet itself is strongly connected to the Internet. Cloudlets can be deployed effortlessly and are supposed to be self-maintained. In brief, a cloudlet resembles a "data center in a box", which makes it trivial to deploy on business premises, such as a coffee shop or doctor's office [3].

Fog computing, also known as "cloud at the edge" [5], is one of the famous implementations of the edge computing paradigm. The term "fog computing" was first introduced by Cisco in 2012 [6, 7]. Fog computing is an emerging paradigm for extending the cloud service to the "ground" [8]. The term "fog" is, at times, used interchangeably with the term "fog computing". Fog computing deploys a lightweight computing facility usually known as mobile edge servers (also referred to as fog computing nodes) in the proximity of mobile

*Co-corresponding authors
Email address: douwc@nju.edu.cn (Wanchun Dou)

users. Consequently, mobile users can access these mobile edge servers with only one-hop wireless connection and a higher transmission speed than the speed from the remote cloud platform to end users [8].

MEC is a "hardware + software" system that provides user services and cloud computing functions, creating high performance, low latency, and high bandwidth in the telecom service environment. It accelerates the downloading of contents, services, and applications in the network, and allows consumers to enjoy uninterrupted, high-quality web experiences. Its technical features mainly include proximity, low time delay, high broadband, and location awareness, which have broad application prospects for the future. MEC is the first step towards a flatter network for 5G, as it provides an application programming interface and opens up the underlying network capabilities to third parties, enabling networks to be customized and interactive according to the business needs of third parties [9]. In order to broaden the MEC applicability into heterogeneous networks, including WiFi and fixed access technologies, ETSI ISG has dropped the "mobile" from MEC and renamed it multi-access edge computing since September 2016 [10].

Accordingly, in the edge computing paradigm, mobile edge servers could be considered as fog computing nodes, MEC nodes, and cloudlet nodes in different edge computing implementations. Despite the varying technical features of these edge computing implementations, these mobile edge servers are heterogeneous in nature and can therefore be based on different types of hard devices, including, but not limited to, access points as well as Internet of things (IoT) gateways [11, 12]. In summary, edge computing aims to provide context-aware distributed computing and storage at the edges of networks.

According to [13], the computational tasks of smart devices in the real world are generally not required to be processed immediately, but rather by a certain deadline. Consequently, mobile devices could offload certain computational tasks to mobile edge servers in their proximity to release their computation workloads. Therefore, the quality of experience (QoE) of mobile services could be improved by leveraging edge computing.

Consider a tourist bus with passengers on board that stops in an expressway service area, and numerous passengers may be using their mobile devices for their leisure time in the area. Certain computation-intensive applications, such as augmented reality, virtual reality, and video transcoding, may place great pressure on the edge network. Therefore, the time delay for these applications may be extremely high, which induces unacceptable user experiences. Furthermore, with the increase in the number of smart mobile devices and data-heavy mobile applications, global mobile data traffic has been increasing dramatically in recent years. The QoE of mobile services cannot be guaranteed without high-speed and stable network connections. Fig. 1 illustrates a toy example for this type of scenario. As mentioned previously, edge computing can address this problem by allowing mobile devices to offload their computation-intensive tasks to mobile edge servers, thereby reducing the time delay and improving the QoE for users.

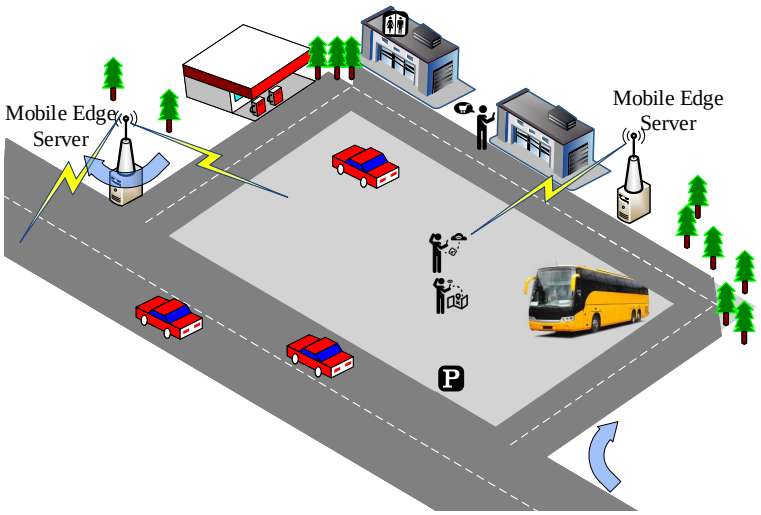


Figure 1: Example of expressway service area with numerous people using mobile devices

As mobile edge servers are geo-distributed and deployed in very common places such as roads, bus terminals, and shopping centers, mobile smart devices can offload part or all of their computation-intensive tasks to mobile edge servers in order to release their computation workloads almost anywhere. However, when a tourist bus is moving on the expressway, the situation becomes more complicated. Suppose that a tourist bus is moving

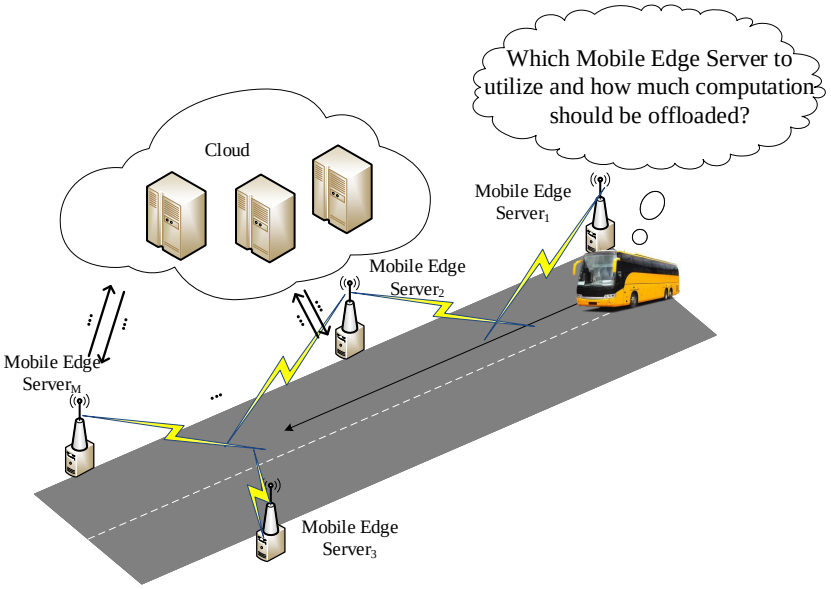


Figure 2: Selecting mobile edge server to offload computation

on an expressway, which is almost covered throughout by mobile edge server services. As the moving speed of the tourist bus is generally fast on the expressway, several candidate mobile edge servers would have to be utilized. Because each mobile edge server workload differs, the manner in which to select a mobile edge server to offload computation should to be addressed properly. Moreover, the amount of computation that should be offloaded from the vehicle remains to be determined for a specific mobile edge server. In fact, the current task deadline and each available mobile edge server current workload should be jointly considered for the offloading scheme. These questions could be illustrated in Fig. 2.

In this paper, we provide three contributions, as follows.

- Firstly, we proposed a peer-to-peer (P2P)-enabled decentralized mobile edge server management scheme for the edge computing environment. This scheme can enable all mobile edge servers to participate in updating information in a decentralized manner. Accordingly, mobile devices can easily obtain the available information of interested mobile edge servers from the nearest network-connected mobile edge server, and therefore make improved offloading decisions for selecting the target mobile edge server.

- Secondly, we introduce a sophisticated computation offloading model for mobile smart devices (particularly for high-speed movement of devices, such as smart vehicles), which involves the task upload and retrieval procedure in terms of when to offload computation to a mobile edge server in the edge computing environment. In particular, the workload transmission under wireless networks, actual computing capabilities, and geo-distribution features of mobile edge servers are explicitly and jointly considered.

- Thirdly, an offloading-efficient optimization problem is formulated, which involves maximizing the offloading efficiency by selecting the optimal mobile edge server, subject to meeting the task deadline constraint. We propose a deadline-aware and cost-effective offloading approach, which aims to improve the offloading efficiency among mobile edge servers and allows more tasks to meet their deadlines. Experiments on emulated data sets confirm the feasibility and efficiency of our approach by means of comparison with other offloading schemes.

The remainder of this paper is organized as follows. Section 2 introduces the background of edge computing and reviews the existing offloading approaches. In section 3, practical decentralized management of mobile edge servers in the edge computing environment is proposed. In section 4, we present the modeling and analysis of computation offloading in a decentralized P2P-enabled edge computing environment. In section 5, experiments and a performance analysis of our approach are presented, followed by a summary and future work in section 6.

2. Preliminary Knowledge and Related Work

2.1. Edge Computing

Edge computing extends cloud computing by introducing an intermediate edge layer between mobile devices and the cloud. Rather than a substitute, edge computing often serves as a complement to cloud computing [14]. Accordingly, this leads to a three-layer mobile-edge-cloud hierarchy [15]. The intermediate edge layer consists of geo-distributed mobile edge servers, located sufficiently near to mobile users. These mobile edge servers are virtualized devices with embedded data storage, computing, and network communication facilities, and therefore provide computing, storage, and communication resources in the close proximity of mobile users [16]. Furthermore, these mobile edge servers can guarantee high QoS to mobile users, owing to local one-hop distances with high-rate wireless network connections.

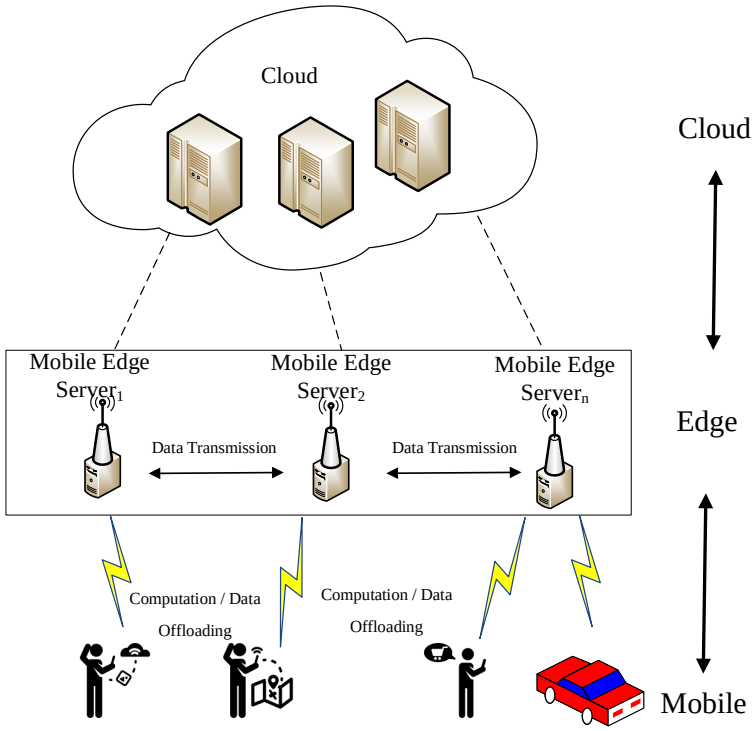


Figure 3: Three-layer mobile-edge-cloud hierarchy

As illustrated in Fig. 3, the edge layer is located between the cloud and mobile device layers. Mobile devices can offload not only data traffic to the edge layer to enlarge their network transmission bandwidths, but also computation tasks to the edge layer to release their workloads. Any existing network components, such as WiFi access points and Femtocell routers, can easily change to edge servers by upgrading their computing and storage resources and reusing the wireless interface. In this paper, we mainly focus on the computation aspect of edge computing.

Edge computing not only reduces the backbone traffic to be sent to the cloud, but also improves the latency for delay-sensitive IoT applications, by reducing the relatively lengthy delay of remote cloud computing [17]. The concept of using edge computing not only provides network resources, but also allows for computational resources closer to users, thereby improving scalability from a computation perspective and QoE in terms of mobile users.

2.2. Computation Offloading in Edge Computing

At present, application offloading is a hot topic, owing to the appearance of cloud computing and MEC. Each mobile application can be offloaded at the coarse-grained application level [18] and fine-grained task level [19]. Application-level offloading means that all of the application functions are executed on the edge node side (for example, by virtual machines (VMs)), and each piece of user equipment (UE) runs as a thin client. In fact, application-level offloading involves simpler operation and lower programming difficulty, but induces a higher

computation workload on the edge node, including the time consumed by application initialization in the VMs. For the task-level offloading method, this could provide the opportunity to release intensive workloads on single edge nodes by distributing tasks to run in parallel on different edge nodes, and therefore improve the performance, although this would require a more sophisticated task scheduler in the system. An ideal offloading scheme could make the necessary task-scheduling decisions so that the overall offloading leads to reduced completion time. Also, the deployment of offloading technology in the real world usually accompanies with many related challenges, e.g., privacy preservation [20, 21, 22, 23, 24] for the offloaded tasks, message/task scheduling [25, 26] in distributed embedded systems, data indexing technique [27], data aggreation protocols [28], and machine learning technique [29].

Most of the available papers on edge computing focus on designing different and effective offloading schemes for resource-limited mobile devices in order to offload computation. Zhou *et al.* [30] proposed an online auction-based stochastic offloading scheme, which uses Lyapunov optimization techniques to achieve a near-optimal, long-term, system-wide utility with several economic properties, including truthfulness, individual rationality, and budget balance. Moreover, Chang *et al.* [31] considered the energy consumption and delay constraint for when to offload certain computation-intensive tasks from mobile users. Considering that mobile devices are usually battery-powered, and ***energy harvesting*** is viewed as a promising and green technology for extending the battery time of mobile devices, Liu *et al.* [32] took into account the social relationships of energy-harvesting mobile devices in the design of a computational offloading scheme in MEC, and aimed to minimize the social group execution cost. Accordingly, they proposed a dynamic computation offloading scheme, designing the offloading process in the fog computing system with energy-harvesting mobile devices. Du *et al.* [33] tackled the computation offloading problem in a mixed fog/cloud system by jointly optimizing the offloading decisions and allocation of computational resources, transmitting power, and radio bandwidth, while guaranteeing user fairness and maximum tolerable delay. The cost of UE is defined as the weighted sum of energy consumption and latency: $Cost_n = \lambda_n^e E_n + \lambda_n^t T_n$, where $\lambda_n^e, \lambda_n^t \in [0, 1], n \in N$ denotes the weights of the energy consumption and delay for UE_n, respectively. In order to ensure the fairness of all UEs, they minimized the maximum cost among all of the UEs, while meeting the maximum delay constraints. Sundar *et al.* [34] studied the scheduling decision for an offloading application consisting of dependent tasks, in a generic cloud computing system comprising a network of heterogeneous local processors and a remote cloud server. They developed a heuristic algorithm known as individual time allocation with greedy scheduling (ITAGS) to overcome the obstacles of task dependency and deadline constraints.

To the best of our knowledge, although the performance evaluations in the above work were demonstrated to be effective, they ignored the result-retrieval procedure during the offloading process. Moreover, for moving vehicles with high speeds, limited time is available for offloading their tasks. Moreover, after the mobile edge server computing for the offloaded computation, the vehicles may already have moved beyond the prior mobile edge server service coverage, and should obtain the computational results from other mobile edge servers, according to their new geographical locations.

Unlike smart phones and tablets, smart vehicles generally use petrol as fuel or even solely use large capacity batteries, which means that energy consumption is not a critical issue for vehicles. Furthermore, wireless power transfer technology [35, 36] can be utilized for deployment on expressways to charge the vehicles continually during driving. Therefore, in this paper, we only consider the manner in which to make efficient offloading decisions for vehicles, and ignore the energy consumption during the offloading process.

3. Towards Practical Management of Mobile Edge Servers in Edge Computing Environment

Actual computation offloading in the edge computing environment generally involves two roles, namely the mobile edge server side and mobile device side. It is essential to manage mobile edge servers efficiently in the edge computing environment before providing computation offloading services for mobile devices. In this paper, we focus on smart vehicles as the mobile devices in the edge computing environment, without loss of generality. Suppose that each mobile edge server processes offloaded tasks from vehicles in a first-in, first-out (FIFO) order. Accordingly, a mobile edge server with a shorter waiting time could be considered as a more suitable offloading target.

3.1. Naive Centralized Management Scheme for Mobile Edge Servers

Intuitively, it is not difficult to consider that there could be one computing server located in the cloud layer, which connects to all mobile edge servers with strong network connectivity by means of a wired network.

Mobile users could establish a network communication to a computing server through a cellular network. The computing server acts as proxy to aid mobile users to query mobile edge server computation workloads.

Mobile users could query their interested mobile edge server computation workloads by requesting information from the computing server (namely, the proxy). When the computing server receives a request from a mobile user, it will establish parallel network connections to all requested mobile edge servers, and aggregate their responses. In order alleviate the network bandwidth pressure, all mobile users only query the information from the centralized proxy to obtain available candidate mobile edge servers in their moving direction.

Theoretically, for each task j arriving at vehicle i in time slot t, suppose that the vehicle wishes to know several mobile edge servers which constitute $\mathcal{R}_j^i(t)$, and tw_k denotes the waiting time for the mobile edge server k availability, for definiteness and without loss of generality, in this paper.

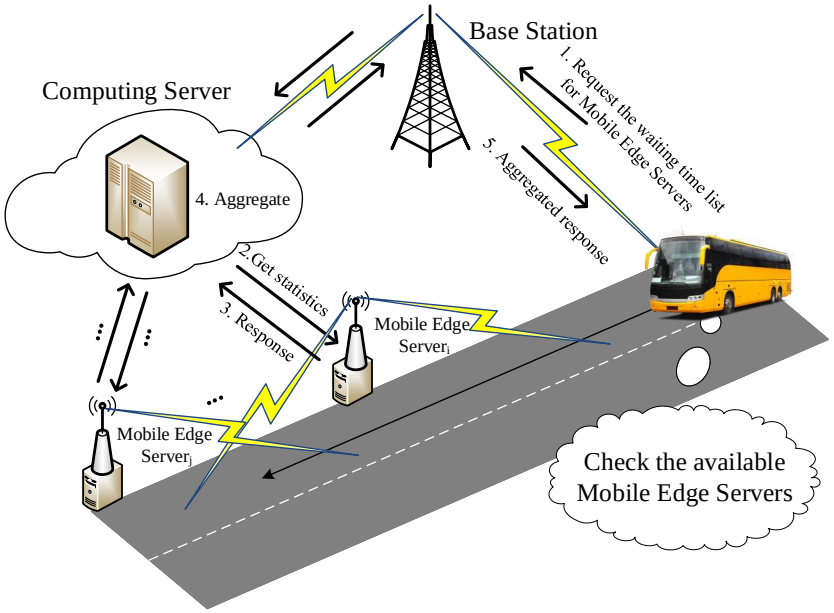
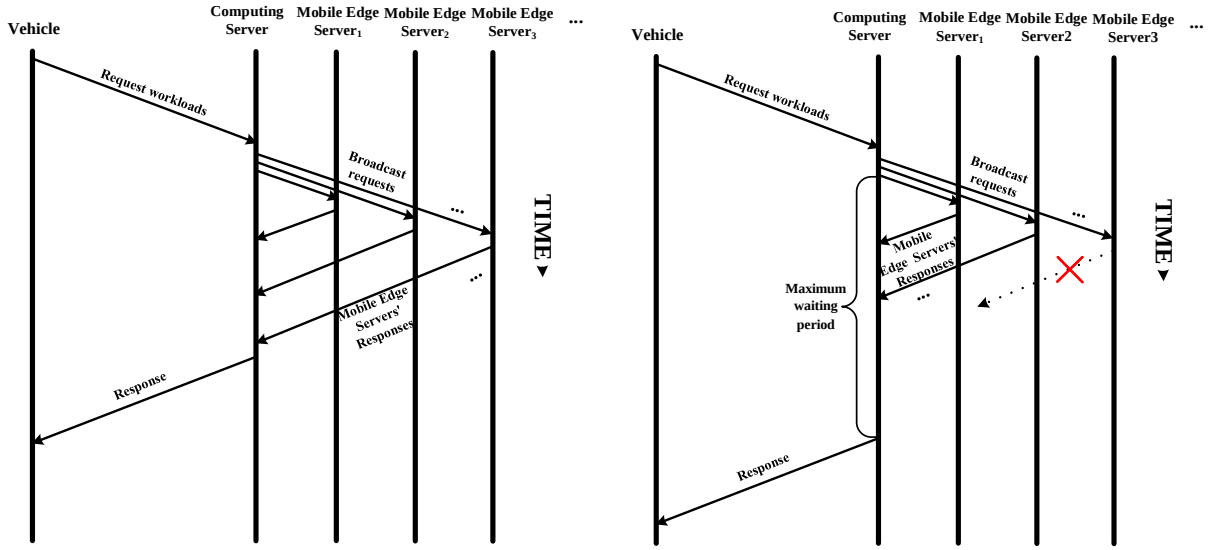


Figure 4: Collecting available mobile edge servers by querying information from computing server

As illustrated in Fig. 4, when a moving vehicle wishes to obtain information regarding the waiting time list for the forthcoming mobile edge servers, it should follow these five steps:

1) The vehicle sends the query request to the computing server regarding the $\mathcal{R}_j^i(t)$ mobile edge servers via the cellular network.
2) The computing server broadcasts the requests to all $\mathcal{R}_j^i(t)$ mobile edge servers and waits for all responses via the wired network.
3) Each mobile edge server k associated with $\mathcal{R}_j^i(t)$ reports its tw_k to the computing server via the wired network.
4) The computing server aggregates all of the responses and updates the mobile edge server status in its database.
5) The computing server sends the aggregated response, which contains all of the query results, to the vehicle via the cellular network.

However, with an increasing number of mobile devices in the mobile layer (as depicted in Fig. 3), this type of method may not be feasible for practical deployment. The problem with this centralized design method is that the proxy may not be able to respond to all query requests from mobile edge servers sufficiently quickly. The details of this problem are illustrated in Fig. 5, where the computing server responds to the query requests from all of the vehicles, and broadcasts the query requests to pull the updated workloads in the mobile edge servers. At times, the queried mobile edge server cannot respond to the requests immediately, owing to the fact that it should answer near-simultaneous requests from different vehicles one after the other. Moreover, if the response packet from mobile edge server is lost in the network, the computing server will not receive the response in time. The computing server should adjust the maximum waiting period for each broadcast frequently, according to the current network condition. Furthermore, certain attackers may establish



(a) Ideal query for workloads

(b) Delayed response of actual workloads

Figure 5: Query for updated workloads of candidate mobile edge servers

distributed denial of service attacks to place additional network pressure on the centralized proxy, which can easily induce a breakdown of the entire offloading system.

To this end, all of the questions above motivated us to design a decentralized management scheme for mobile edge servers and an offloading approach in the edge computing environment.

3.2. P2P-Enabled Decentralized Management Scheme for Mobile Edge Servers

As mentioned in the previous sections, with the rapid development of edge computing, numerous geo-distributed mobile edge servers would exist in the entire city areas.

P2P networking is a decentralized application architecture that partitions workloads among peers. Peers in such networks have equal privileges. The decentralized nature of P2P networks increases robustness, because it removes the single point of failure that may be inherent in a client-server-based system [37]. Furthermore, because the roles of all peers in the network are the same, unstructured P2P networks are highly robust in the face of high "churn" rates; that is, when large numbers of peers frequently join and leave the network [38].

To this end, we explore the manner in which P2P networking can be used in the MEC environment. In our opinion, P2P networks enables distributed nodes to cooperate with one another and maintain a reliable network; therefore, it is nearly impossible for the entire system to crash. Accordingly, P2P networking is a promising solution for the decentralized management of mobile edge servers for offloading services in the MEC environment.

As the computation offloading service is a virtual service, paying for the service via electronic coins has become a direct measure. Suppose that vehicles pay for the service using ***Edgecoin*** as a virtual currency, which can be exchanged in local currency. Mobile edge servers and vehicles can store the entire history of the Edgecoin transactions (every transaction by every vehicle, ever), and every workload update transaction by every online mobile edge server.

All of the public keys of authorized mobile edge servers and mobile devices are permanently stored in the database of the edge computing service provider. This would be helpful for performing service liquidation, verifying the authority of every mobile edge server in the edge computing environment, and charging mobile users for offloading services. Moreover, if one mobile edge server leaves the network owing to occurrences such as a power failure or earthquake, its private key would be regenerated when it reboots and rejoins the network.

Mobile edge servers use their private keys to signature the actual updated workloads and their geographical positions, and the updated information will be propagated as the workload transaction to the entire P2P network. The propagation delay on the network could be enhanced by leveraging BCBPT protocol [39], which is a proximity-aware extension of Bitcoin protocol. Bitcoin is also based on a P2P network, with the peers mining Bitcoins. It is well known that Bitcoin follows a distributed trust mechanism, which relies on distributed validation and tracking of transactions. BCBPT aims to increase the proximity of connectivity among nodes

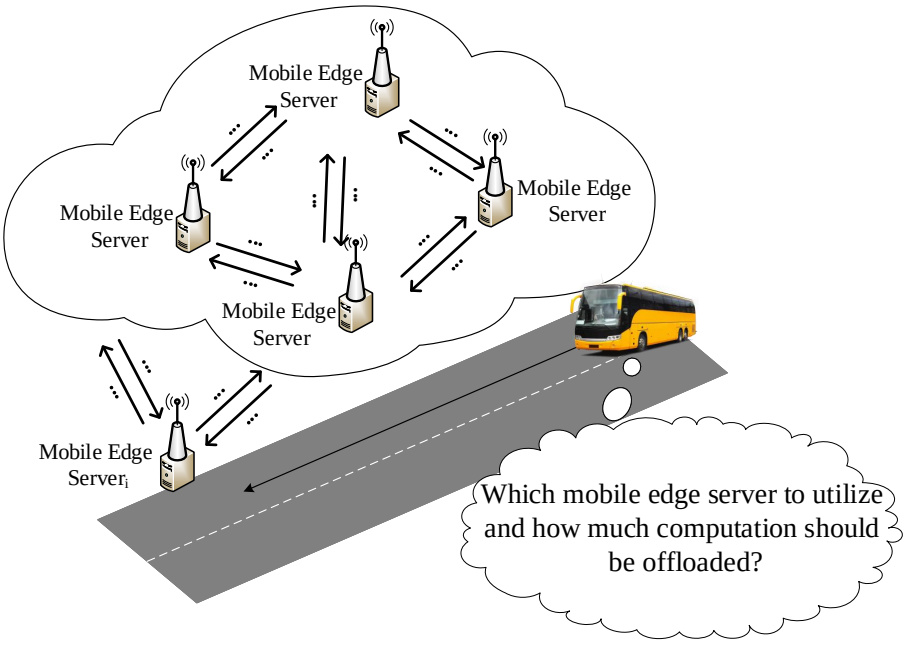


Figure 6: P2P-enabled decentralized management of mobile edge servers

in the Bitcoin network, based on round-trip ping latencies. Likewise, the workload fluctuation of each mobile edge server in the edge computing environment could be also be considered as transactions that should be broadcasted to the peers in the network proximity. By leveraging BCBPT protocol, the propagation delay in the P2P-enabled decentralized management of mobile edge servers could be improved.

Similarly, vehicles use their private keys to signature the offloading transactions, which involves the actual payment for the offloading service. As each vehicle stores the entire history of the offloading and workload transactions, it could easily determine which mobile edge server should be selected for utilization to offload its specific computation by using certain offloading strategies.

On this basis, we argue that each vehicle could easily know every mobile edge server actual workload at any time by leveraging BCBPT protocol. Suppose that all mobile edge servers constitute the full set $\mathcal{R}$, and each vehicle can at most offload its current task to one mobile edge server. Obviously, in a specific time slot t, the number of candidate mobile edge servers $\mathcal{R}^i(t)$ for vehicle i is quite small. Moreover, it could easily be found that a shorter network distance between the mobile edge server and mobile device would promote higher offloading performance. In common with [39], we argue that two nodes N_i and N_j are considered close to one another if $D_{i,j} \leq D_{thd}$, where $D_{i,j}$ is the distance between N_i and N_j, measured by the round-trip latency in the network, and D_{thd} is the latency threshold.

4. Modeling and Analysis of Computation Offloading in P2P-Enabled Edge Computing Environment

In this paper, we consider an edge computing environment as illustrated in Fig. 6. In our model, each vehicle i mostly has only one computational task, which can be described in three terms $T_j = \{d_j^i, c_j^i, t_j^{i,M}\}$ at any time, and $i, j \in \mathcal{N} = \{1, 2, ..., N\}$. Here, d_j^i denotes the size of the input data for computation, c_j^i is the amount of computation resources measured in million instructions per second (MIPS) that should be satisfied to finish complete j in vehicle i, and $t_j^{i,M}$ is the deadline for the corresponding task j. Although every passenger or vehicle itself may create tasks simultaneously, tasks could be considered as one larger task in every time slot. Moreover, we use $t_j^{i,L}$ to denote the time cost while executing task j locally in vehicle i itself, with its computational capability denoted by q^i, and this can be expressed by the following equation:

$$t_j^{i,L} = \frac{c_j^i}{q^i}, \tag{1}$$

which would be considered as the *local execution time*. It should be easy to note that, if we wish to allow the execution time of a specific task j to be acceptable, $t_j^{i,L} \leq t_j^{i,M}$ should be satisfied. Furthermore, it should be mentioned that the number of smart vehicles is increasing sharply, and their resource-hungry tasks pose significant challenges in providing convenient and reliable vehicular services. Although a vehicle can perform computations locally at any time, the computational workload cannot always be zero when several new tasks arrive simultaneously, and the computation time may be significantly lengthy, which means that $t_j^{i,L} > t_j^{i,M}$ usually holds. Fortunately, with the aid of mobile computation offloading techniques, vehicles can offload part or all of their task computation to mobile edge servers to release their workload, in order to meet the task deadlines if necessary, thereby saving energy for vehicles and improving the QoE for passengers.

We consider numerous existing mobile edge servers located alongside roads, with their IDs numbered from $\{1, 2, ..., N\}$, and all the mobile edge servers constitute the aforementioned full set $\mathcal{R}$. In fact, a vehicle that runs through the road may associate with its nearest mobile edge server in terms of its geographical location. Whenever it continues to move, its geographical location may move away from the previous connected mobile edge server and closer to the next mobile server that has a stronger wireless signal, following which the vehicle would hand off the wireless connection to the new mobile server. Normally, the computing and mobile edge servers are operated by certain profit-making organizations. Thus, these operators would like to charge for the actual resource consumption by users [1]. Each smart vehicle can establish the service coverage, computational capability, and computation unit price of each mobile edge server in advance along the vehicle mobility trajectories with the aid of the transportation department, which possesses all of the information along the expressway.

In this paper, the computational capability of each mobile server is denoted by $f_1, f_2, ..., f_M$, which can be considered as the service infrastructure of edge computing, while the computation unit price is represented by $p_1, p_2, ..., p_M$.

In our assumption, each vehicle can at most offload its current task to one mobile edge server when it runs through the road. Let $x_{j,k}^i = 1$ indicate that vehicle i selects to offload part or all of its task j to mobile edge server k, and $x_{j,k}^i = 0$ otherwise. Thus, we can obtain $\sum_{k=1}^{\mathcal{R}} x_{j,k}^i \leq 1, i, j \in \mathcal{N}$.

4.1. Offloading Time Cost

We argue that, if a moving vehicle wishes to offload its computation to a forthcoming mobile edge server in its vicinity, the time cost should consist of three parts. Firstly, the vehicle should spend time to move sufficiently close to the specific mobile edge server to hand off its wireless connection thereto, and then spend time to upload the required data of the task to be processed. Secondly, the vehicle should wait a certain time for the result data from the mobile edge server. Lastly, the corresponding result data should be obtained from the mobile edge server, which will also cost time.

In general, the offloading time cost for a specific task j offloaded from vehicle i to mobile edge server k is expressed as:

$$ts_{j,k}^i = \frac{l_k^i}{v} + \frac{d_j^i}{r_k}, \tag{2}$$

and the computing time for a specific task j of vehicle i to be executed on mobile edge server k is

$$tc_{j,k}^i = \frac{\alpha_{j,k}^i c_j^i}{f_k}, \tag{3}$$

where the parameter $\alpha_{j,k}^i$ $(0 \leq \alpha_{j,k}^i \leq 1)$ denotes the offloading ratio of task j, l_k^i is the distance from the mobile edge server k to the location of the current vehicle i, and v and f_k are the vehicle speed and computational capability of mobile edge server k, respectively. In this paper, we assume that the data transmission rate of the wireless channel in the service coverage of a specific mobile edge server is a statistically average value, and let r_k denote the statistically average data transmission rate of mobile edge server k.

Following remote computation, owing to the limited contact opportunity with mobile edge servers (tens of seconds), the vehicle may have moved far from the service coverage area of mobile edge server k, and near to another mobile edge server z. Certainly, z could also be k when the vehicle does not leave the wireless area of mobile edge server k. Then, the result data for the task should be transmitted from the mobile edge server to the vehicle. We use

$$tr_{j,z}^i = \frac{d_j^{i'}}{r_z} \tag{4}$$

to denote the backward transmission time for the specific result data and ignore the transmission time between mobile edge servers k and z ($k \neq z$), in that the communication between mobile edge servers occurs through a high bandwidth wired connection, and the time delay of the wired communication is sufficiently negligible compared to contemporary wireless technologies. Then, we obtain the total time for the offloading part of the execution time of task j as follows:

$$t_{j,k\sim z}^{i,\text{offload}} = ts_{j,k}^{i} + tc_{j,k}^{i} + tr_{j,z}^{i}. \tag{5}$$

As the α_j^i part of the task is executed remotely, the other part of the computation of task j should be executed locally to constitute the entire task completion, which would cost

$$t_j^{i,\text{local}} = \frac{(1-\alpha_{j,k}^i)c_j^i}{q^i}. \tag{6}$$

The completion time of task j should be

$$t_j^{i,\text{actual}} = \max\{t_{j,k\sim z}^{i,\text{offload}}, t_j^{i,\text{local}}\}, \tag{7}$$

in that only both complete parts (local and remote) can enable the entire task to be considered as completed, and the slower part work would cause the bottleneck of the overall process. However, the completion time of task i can be considered as acceptable if and only if $t_j^{i,\text{actual}} \leq t_j^{i,M}$. The difference between $t_j^{i,M} - t_j^{i,\text{actual}}$ can shed light for offloading under our approach to improve the offloading efficiency and user QoE.

4.2. Generation for Candidate Mobile Edge Servers in Every Computation Offloading

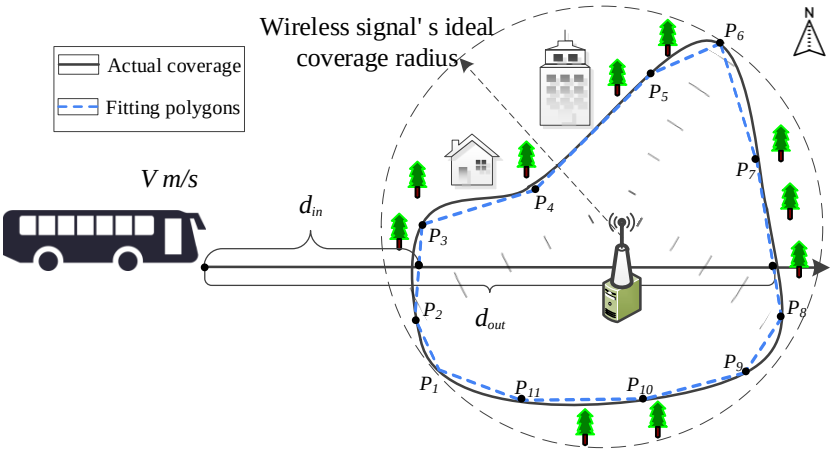


Figure 7: Toy example of moving through service coverage of mobile edge server

Consider the scenario in which a vehicle is moving towards service coverage of a mobile edge server, and it can know the coverage range of the mobile edge server with the aid of contributions of other vehicles in advance. Suppose that the ideal service coverage of the mobile edge server is denoted by the dotted circle in Fig. 7, and the mobile edge server resides in the circle center. In fact, there are always trees, buildings, and houses in downtown areas, which have a significant impact on the wireless coverage range. The actual coverage can be the irregular area, and the near-actual coverage can be fitted by the polygons denoted by the blue dotted line in Fig. 7. The polygons are composed of 11 vertexes, denoted by $\{P_1, P_2, P_3, ..., P_{11}\}$ in this figure, and more accurate coverage can be fitted when using additional vertexes.

In fact, assume that vehicle i moves on the road with a speed of v_i, which is a constant value. Moreover, suppose that task j arrives in time slot t, and should be scheduled immediately. As the deadline of task j is $t_j^{i,M}$, the distance between the locations at which task j arrives and expires should be $v_i t_j^{i,M}$. Considering the high speeds of moving vehicles and limited service coverage of mobile edge servers, it is difficult not to notice that $v_i t_j^{i,M}$ may cross the service coverage of several mobile edge servers. Assume that all of these mobile edge servers compose the mobile edge server set $\mathcal{R}_j^i(t)$, and $\mathcal{R}_j^i(t) \subseteq \mathcal{R}$ should clearly be satisfied. In order to obtain $\mathcal{R}_j^i(t)$ from $\mathcal{R}$, a brute force method involves checking each mobile edge server in $\mathcal{R}$ to determine

whether its service coverage would be crossed by the moving vehicle. However, the search space of this method is excessively large and time consuming, in that the size of $\mathcal{R}$ is significantly large.

To improve the query performance for $\mathcal{R}_j^i(t)$, we construct an R-tree from the service coverage of all of the mobile edge servers. The R-tree organizes data in a tree-shaped structure known as an R-tree index. The index uses a bounding box, which is a rectilinear shape that completely contains the bounded object or objects. Each bounding box can enclose a data object or another bounding box. The key concept of the R-tree data structure is to group adjacent objects and represent them with their minimum bounding rectangle (MBR) in the tree parent node [40]. In general, the MBR for a two-dimensional (2D) circle is a square with a side equal to the circle diameter. The minimum bounding box for a three-dimensional sphere is a cube with an edge equal to the sphere diameter.

Intuitively, the service coverage of every mobile edge server can be covered by a rectangle in the 2D (x, y) coordinate system, and the MBR is a 2D case of the minimum bounding box. Fig. 8 presents a simple example of the service coverage distribution of the mobile edge servers, and each service coverage is covered by an MBR, denoted by R3, R4, R6, R7, and R8 in the figure. Moreover, the moving vehicle trajectory is denoted by the red shaded area, which overlaps with R3, R4, and R7, and we can use the shaded area as the query input for the R-tree. As all service coverages lie within this bounding rectangle, a query that does not intersect the bounding rectangle also cannot intersect any of the contained objects. An R-tree example is illustrated in Fig. 9, according to Fig. 8. At the leaf level of the tree, each rectangle describes a single service coverage. As indicated in Fig. 8, bounding boxes can enclose a single data object or one or more bounding boxes. For example, bounding box R4, which is at the leaf level, contains a service coverage; bounding box R5, which is at the branch level of the tree, contains bounding boxes R4 and R6; and bounding box R1, which is at the root level, contains bounding boxes R3 and R5.

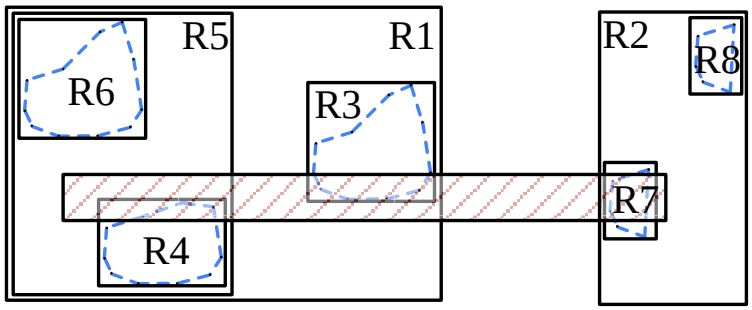


Figure 8: Simple example of mobile edge server service coverage distribution with minimum bounding rectangles

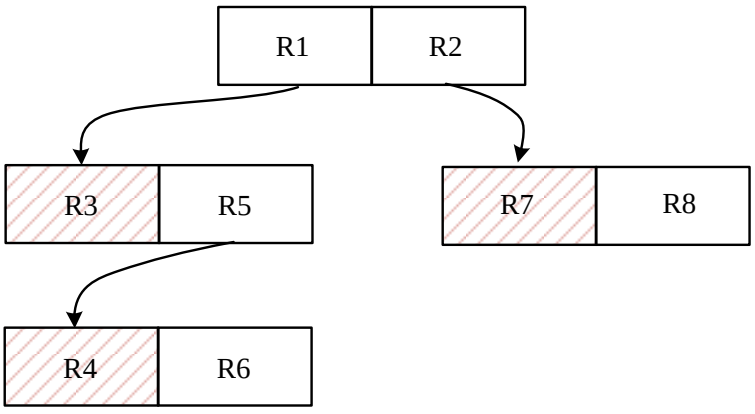


Figure 9: R-tree for example mobile edge server service coverage

We assume that the vehicle moves across service coverages along the horizontal right arrow, with a speed Vm/s, and the current time is 0. The distances from the current location of the vehicle to the coverage area

entrance and exit are denoted by d_{in} and d_{out}, respectively. Thus, the vehicle will require a time of

$$t_{\text{internal}} = (d_{\text{out}} - d_{\text{in}})/V \tag{8}$$

to move across the coverage. Obviously, $t^k_{\text{internal}} \geq d^i_j/r_k$ and $t^z_{\text{internal}} \geq d^{i'}_j/r_z$ should be satisfied for the upload and download stages when offloading to a specific mobile edge server k.

In order to obtain d_{in} and d_{out}, the main problem is the manner in which to obtain the coverage entrance and exit. By converting the longitude/latitude into the 2D (x, y) coordinate system for all locations in Fig. 7, it is easy to observe that the vehicle mobility trajectory can be described as a straight line with the general form $ax+by+c=0$ during a tiny time slot in the converted 2D (x, y) coordinate system. To calculate t_{internal} in Fig. 7, the coordinates of the two intersection points of the line and polygons should be determined, and the coordinates divide the entire polygons into two parts. One part contains the $\{P_3, P_4, P_5, P_6, P_7\}$ vertex set, while the other part contains the $\{P_8, P_9, P_{10}, P_{11}, P_1, P_2\}$ vertex set. It is obvious that the two intersection points of the line and polygons are on segments of two pairs of points from the two sets.

The *cross-product* technique is an easy method for obtaining the two segments, and when using this method, neither division nor trigonometric functions are required, both of which are prone to problems with round-off errors [41]. In the research area of space analytic geometry, considering the vectors $\overrightarrow{OP_1} = (x_1, y_1)$ and $\overrightarrow{OP_2} = (x_2, y_2)$, their cross-product can be formed as

$$\overrightarrow{OP_1} \times \overrightarrow{OP_2} = x_1 y_2 - x_2 y_1. \tag{9}$$

The absolute value of $\overrightarrow{OP_1} \times \overrightarrow{OP_2}$ denotes the area of the parallelogram formed by points $(0,0), P_1, P_2$, and $OP_1 + OP_2 = (x_1 + x_2, y_1 + y_2)$. Moreover, if $\overrightarrow{OP_1} \times \overrightarrow{OP_2}$ is positive, $\overrightarrow{OP_1}$ is clockwise from $\overrightarrow{OP_2}$; if this cross-product is negative, $\overrightarrow{OP_1}$ is counterclockwise from $\overrightarrow{OP_2}$.

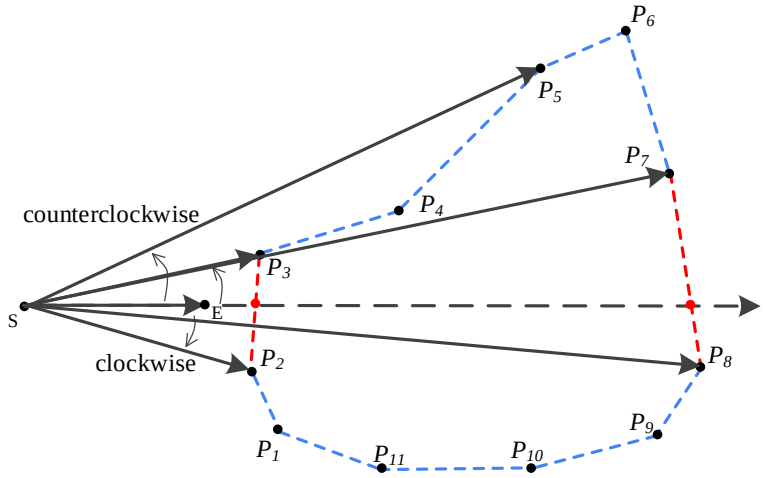


Figure 10: Determining intersection segments of line and polygons

Suppose that $\vec{z}$ is a direction vector for the straight line, which is illustrated as the directed segment $\overrightarrow{SE}$ in Fig. 10. We can enumerate all vertexes of the polygons to verify whether each directed segment (such as $\overrightarrow{SP_3}$ and $\overrightarrow{SP_2}$) is clockwise or counterclockwise relative to $\overrightarrow{SE}$. For example, $\overrightarrow{SE} \times \overrightarrow{SP_3} > 0$ means that $\overrightarrow{SP_3}$ is counterclockwise with respect to $\overrightarrow{SE}$, while $\overrightarrow{SE} \times \overrightarrow{SP_2} < 0$ means that $\overrightarrow{SP_2}$ is clockwise with respect to $\overrightarrow{SE}$. Thus, every vertex can be classified into two parts: the clockwise side part and counterclockwise side part. If we enumerate all vertexes (edges) of the polygons in clockwise order, there would be two edges composed of two vertexes from different vertex parts. In Fig. 10, the two edges are denoted by a red dotted line, and the two intersection points (denoted by red points on the two edges) are indeed the corresponding points of the coverage entrance and exit for the vehicle in Fig. 7. As we can easily obtain the intersection points of the vehicle mobility trajectory and mobile edge server service coverage, each d_{in} and d_{out} of the mobile edge servers can also be measured.

Algorithm 1 illustrates the generation of all candidate mobile edge server(s) in the vehicle moving direction. By running **Algorithm 1** on the moving vehicle, all of the candidate mobile edge server(s) to be utilized

Algorithm 1 Candidate mobile edge server(s) generation

Input: Current time slot t, vehicle moving speed v_i, task T_j, and latency threshold D_{thd}.
Output: $\mathcal{R}_j^i(t)$.

1: Clustering candidate mobile edge servers with network distances less than D_{thd} as $\mathcal{R}_{j,network}^i(t)$.
2: Aggregate all results $\mathcal{R}_{j,geography}^i(t)$ from R-tree when approximate trajectory area is provided as R-tree input.
3: $\mathcal{R}_j^i(t) \leftarrow \mathcal{R}_{j,network}^i(t) \cap \mathcal{R}_{j,geography}^i(t)$.
4: **for** each ϕ in $\mathcal{R}_j^i(t)$ **do**
5: **if** $t_{\text{internal}}^{\phi} < d_j^{i'}/r_\phi$ **then**
6: $\mathcal{R}_j^i(t) \leftarrow \mathcal{R}_j^i(t) \setminus \{\phi\}$.
7: **end if**
8: **end for**
9: **return** $\mathcal{R}_j^i(t)$.

in its moving direction can be obtained. Firstly, it clusters the candidate mobile edge servers with network distances less than D_{thd} by leveraging the BCBPT protocol: $\mathcal{R}_{j,network}^i(t)$, and then it aggregates all of the mobile edge servers that are geographically near to the vehicle in the moving direction. Finally, it filters out the mobile edge servers from $\mathcal{R}_j^i(t)$ that cannot be utilized owing to limited service coverage.

370 This is the first stage of our proposed offloading approach. Thereafter, we progress to the next stage, namely optimal selection of the mobile edge server.

4.3. Optimal Selection of Mobile Edge Server

Once the candidate mobile edge server(s) in the moving direction have been obtained, the following stage involves determining which mobile edge server should be selected in an appropriate metric. We can obtain the utility function for vehicle i to offload its task j to mobile edge server k when the task arrives in time slot t:

$$u_j^i(t) = \sum_{k}^{\mathcal{R}_j^i(t)} x_{j,k}^i[\mu_i(t_j^{i,M} - t_j^{i,\text{actual}}) - p_k \alpha_{j,k}^i c_j^i], \tag{10}$$

where μ_i is a vehicle-specific parameter that illustrates the sensitivity of vehicle i to the reduction of the task completion time, and $\mu_i > 0$ for all cases.

In fact, the vehicles are rational, and all of them wish to maximize their utilities by selecting the offloading target mobile edge servers with a corresponding specific offloading share. In the case where the price set $\{p_k, k \in \mathcal{R}_j^i(t)\}$ is known, the optimization problem for vehicle i to offload its j-th task on the mobile edge server is

$$\begin{aligned} &\max_{\{x_{j,k}^i, i,j \in \mathcal{N}, k,z \in \mathcal{R}_j^i(t)\}} u_j^i(t) \\ &s.t. \quad \begin{cases} x_{j,k}^i = \{0,1\}, & i,j \in \mathcal{N}, k \in \mathcal{R}_j^i(t), \\ \sum_k^{\mathcal{R}_j^i(t)} x_{j,k}^i \le 1, & i,j \in \mathcal{N}, \\ \alpha_{j,k}^i \in [0,1], & i,j \in \mathcal{N}, k \in \mathcal{R}_j^i(t), \\ l_z \ge l_k, & z,k \in \mathcal{R}_j^i(t), \\ t_{\text{internal}}^k \ge d_j^i/r_k, & i,j \in \mathcal{N}, k \in \mathcal{R}_j^i(t), \\ t_{\text{internal}}^z \ge d_j^{i'}/r_z, & i,j \in \mathcal{N}, z \in \mathcal{R}_j^i(t). \end{cases} \end{aligned} \tag{11}$$

375 In particular, l_k will be equal to l_z when $k = z$, and this case will have the constraint $t_{\text{internal}}^k \ge t_j^{i,\text{actual}}$.

It should be noted that this utility is only for a specific task j. In reality, numerous tasks with different deadlines may arrive at each mobile vehicle during each time slot t. We assume that the tasks arriving at different time slots are independent and identically distributed (i.i.d.). Let $Arr^i(t)$ be the computation of the corresponding task that arrives at mobile vehicle $i \in \mathcal{N}$. In the real world, the value of $Arr^i(t)$ would have a jitter, and we assume that it has a certain peak, which is denoted by $Arr^{i,max}$. Thus, we have $Arr^i(t) \le Arr^{i,max}, \forall i \in \mathcal{N}$, and

$$Arr^i(t) = \begin{cases} (1 - \alpha_j^i)c_j^i, & j\text{-th task arrives at time slot } t, \\ 0, & \text{no task arrives}, \end{cases} \tag{12}$$

where α_j^i means the actual offloading share of the computation from task j, for whichever mobile edge server is selected.

Each vehicle maintains a task queue for the computation of arrived tasks, and the computation of tasks in the queue is processed in a FIFO order. If task j local part computation has not been fully processed prior to task $j+1$, a computation backlog may occur, which should be taken seriously.

We use $Q^i(t)$ to denote the *queue backlog* for the waiting computation generated by the previous tasks in vehicle i during time slot t. Therefore, we can obtain:

$$Q^i(t) = \begin{cases} 0, & t = 0, \\ max\{Q^i(t-1) + Arr^i(t) - q^i, 0\}, & t > 0. \end{cases} \tag{13}$$

Thereafter, equation (10) can be derived as follows:

$$\begin{aligned} u_j^i(t) &= \sum_k^{\mathcal{R}_j^i(t)} x_{j,k}^i [\mu_i (t_j^{i,M} - t_j^{i,\text{actual}}) - p_k \alpha_{j,k}^i c_j^i] \\ &= \sum_k^{\mathcal{R}_j^i(t)} x_{j,k}^i [\mu_i (t_j^{i,M} - max\{t_{j,k\sim z}^{i,\text{offload}}, t_j^{i,\text{local}}\}) - p_k \alpha_{j,k}^i c_j^i] \\ &= \sum_k^{\mathcal{R}_j^i(t)} x_{j,k}^i [\mu_i (t_j^{i,M} - max\{max\{ts_{j,k}^i, tw_k\} + \frac{\alpha_{j,k}^i c_j^i}{f_k} \\ &\quad + tr_{j,z}^i, \frac{(1-\alpha_{j,k}^i)c_j^i + Q^i(t-1)}{q^i}\}) - p_k \alpha_{j,k}^i c_j^i], \end{aligned} \tag{14}$$

Lemma 1. *For each task j arriving in vehicle i in time slot t, the utility function $u_j^i(t)$ should have a specific maximum value.*

Proof. Let

$$f(\alpha_{j,k}^i) = max\{ts_{j,k}^i, tw_k\} + \frac{\alpha_{j,k}^i c_j^i}{f_k} + tr_{j,z}^i$$

and

$$g(\alpha_{j,k}^i) = \frac{(1-\alpha_{j,k}^i)c_j^i + Q^i(t-1)}{q^i}.$$

If vehicle i wishes to offload the $\alpha_{j,k}^i$ part of the computation of task j to a specific mobile edge server k in $\mathcal{R}_j^i(t)$, and its utility is denoted by $u_j^i(t,k)$,

$$\begin{aligned} u_j^i(t,k) &= \mu_i(t_j^{i,M} - max\{f(\alpha_{j,k}^i), g(\alpha_{j,k}^i)\}) - p_k \alpha_{j,k}^i c_j^i \\ &= \begin{cases} \mu_i(t_j^{i,M} - f(\alpha_{j,k}^i)) - p_k \alpha_{j,k}^i c_j^i, & f(\alpha_{j,k}^i) \geq g(\alpha_{j,k}^i), \\ \mu_i(t_j^{i,M} - g(\alpha_{j,k}^i)) - p_k \alpha_{j,k}^i c_j^i, & f(\alpha_{j,k}^i) < g(\alpha_{j,k}^i) \end{cases} \end{aligned}$$

and

$$u_j^i(t) = maximize \quad u_j^i(t,k).$$

The optimal mobile edge server is $\arg\max_k u_j^i(t,k)$. As all parameters in $f(\alpha_{j,k}^i)$ and $g(\alpha_{j,k}^i)$ are positive constants for a specific mobile edge server k, except for $\alpha_{j,k}^i$ with the domain $[0,1]$, $f(\alpha_{j,k}^i)$ and $g(\alpha_{j,k}^i)$ are linear functions. Obviously, $f(\alpha_{j,k}^i)$ and $g(\alpha_{j,k}^i)$ may be an increasing and a decreasing function in terms of $\alpha_{j,k}^i$, respectively. In general, we suppose that $f(\alpha_{j,k}^i)$ would be equal to $g(\alpha_{j,k}^i)$ at a specific value of $\hat{\alpha}_{j,k}^i$, and the domain $[0,1]$ would be divided into two parts: ($[0, \hat{\alpha}_{j,k}^i)$ and $[\hat{\alpha}_{j,k}^i, 1]$). Thus, we can obtain the absolute maximum value of $u_j^i(k,t)$ by comparing the two local maximum values from these two domain parts. At times, $f(\alpha_{j,k}^i)$ may also always be greater than $g(\alpha_{j,k}^i)$ in $[0,1]$, and vise versa; therefore, the absolute maximum value of $u_j^i(t,k)$ may occur at $\alpha_{j,k}^i = 0$ or 1 owing to $u_j^i(k,t)$ being a strictly monotonically decreasing/increasing function in $[0,1]$. Clearly, for each mobile edge server k, we can easily obtain the maximum value of $u_j^i(t,k)$. Thus, $u_j^i(t)$ can also obtain the maximum value from all candidate mobile edge servers in $\mathcal{R}_j^i(t)$. □

Algorithm 2 Optimal mobile edge server selection and offloading ratio determination

Input: Current time slot t, μ_i, v_i, and T_j.

Output: Optimal mobile edge server $\hat{k}$ and corresponding parameter α_j^i.

1: Calculate candidate mobile edge server set $\mathcal{R}_j^i(t)$ based on v_i and $t_j^{i,M}$ by running **Algorithm 1**.
2: For each mobile edge server in $\mathcal{R}_j^i(t)$, query its waiting available time tw_θ, $\theta \in \mathcal{R}_j^i(t)$ from computing server.
3: Maximize $u_j^i(t,k)$ for each mobile edge server k in $\mathcal{R}_j^i(t)$ with corresponding optimal $\alpha_{j,k}^i$.
4: $\hat{k} \leftarrow \arg\max_k u_j^i(t,k)$ and $\alpha_j^i \leftarrow \alpha_{j,\hat{k}}^i$.
5: **return** $\hat{k}, \alpha_j^i$.

Therefore, we obtain the algorithm for the selection of the optimal mobile edge server with the optimal offloading computation ratio for the j-th task of vehicle i in time slot t, and its pseudocode can be observed in **Algorithm 2**. The worst-case running time of this algorithm depends on the size of $\mathcal{R}_j^i(t)$, and the upper bound is $\mathcal{O}(\mathcal{R}_j^i(t))$. Each vehicle should notify the corresponding mobile edge server of the actual computation by using the computing server as communication proxy; therefore, each mobile edge server can maintain its computation backlog updated.

5. Numerical Results

In this section, we firstly evaluate the effectiveness and feasibility of the proposed decentralized P2P-based management of mobile edge servers in the edge computing environment, and then investigate the optimal mobile edge server selection strategy.

5.1. Environment Setup for Query Time Comparison

We consider a scenario in which 10 mobile edge servers are randomly located on a 1000 m road, and each mobile edge server service coverage on the road randomly moves from [1, 100] m. We use ns-3 as a discrete event simulator to evaluate the performance of our approach. In such a simulator, each event is associated with its execution time, and the simulation proceeds by executing events in the temporal order of the simulation time. In our simulation, the average network transmission speed is 10 Mb/s. Each mobile edge server constitutes one unstructured P2P network via wired cables. The simulation environment setting details are presented in Table 1.

Table 1: Environment settings

	Description
Hardware	Intel Core i5-7300M CPU @2.60 GHz and 8.00 GB memory
Software	Windows 10 Home 1803

Fig. 11 presents an example network topology of the mobile edge servers and vehicle. The moving vehicle wishes to collect certain mobile edge server workloads so as to draw comparisons and make an improved offloading decision.

Table 2: Comparison results between centralized and decentralized methods in terms of average query time

Number of mobile edge servers	**Average query time (ms) in centralization**	**Average query time (ms) in decentralization**	**Performance improvement ratio**
1	21.02	4.2	400.5%
2	23.14	4.8	382.18%
3	24.35	5.2	368.43%
4	25.20	5.6	350.17%
5	25.85	6	330.87%
6	26.42	6.4	312.85%
7	26.81	6.79	294.91%
8	27.21	7	288.85%
9	27.49	7.2	281.86%

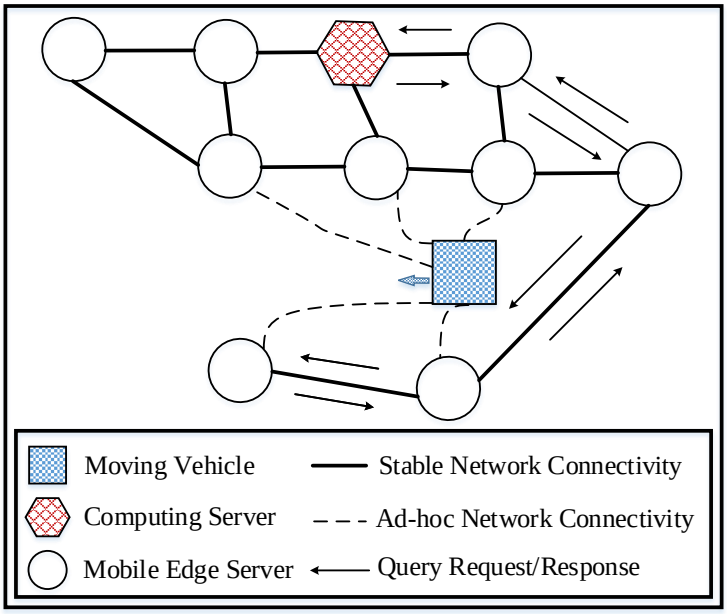


Figure 11: Network topology of simulation environment

Table 2 displays the comparison results between these two approaches in the simulation environment. For every number of available candidate mobile edge servers, we ran 10000 trials to calculate the average query time performance of both approaches under the aforementioned network topology. It can easily be observed that the decentralized style approach has a shorter average query time delay under a different number of candidate mobile edge servers. Moreover, with the increasing number of candidate mobile edge servers during the offloading decision time slot t, both approaches would spend more query time for collecting the workloads of all mobile edge servers. However, in contrast to the centralized style approach, the increasing trend of the query time delay for the decentralized style approach is not obvious.

5.2. Environment Setup for Offloading Utilization

Based on the aforementioned mobile edge server distribution settings, we suppose that there are at most 15 moving vehicles on the road, and their speed is 120 km/h. For wireless transmission settings, we consider that the average wireless transmission speed is 10 Mb/s. We developed an expressway traffic simulator to evaluate the performance of our approach, and the runtime environment settings are the same as those of the environment setup for the query time comparison (Table 2).

Table 3: Simulation parameters

Parameter	**Value**
Length of road (m)	1000
Number of mobile edge servers	10
Service coverage of each mobile edge server (m)	[1, 100]
Number of moving vehicles	[1, 15]
Moving speed of vehicles (km/h)	120
Wireless transmission speed (Mb/s)	10
Task data size (MB)	[2, 5]
Task result size (MB)	[0.5, 1]
Task computational instructions (millions)	[5, 20]
Computational capability of mobile edge servers (MIPS)	[3, 10]
Computational capability of vehicles (MIPS)	[1, 3]
Computation unit price of mobile edge servers (Edgecoin)	[10, 40]
Sensitivity (μ) of task completion time	500

Task setup: The computation instructions of each task from all vehicles are randomly distributed in [5, 20]. For each task j in vehicle i, its deadline constraint $t_j^{i,M}$ is set to be $1.5 \times t_j^{i,L}$, where $t_j^{i,L}$ is the *local execution*

time of task j in vehicle i, as discussed in section 4. Moreover, the tasks in each vehicle are assumed to arrive in a Poisson process, and we set the parameter λ in the Poisson process as $\frac{MeanOfMobileEdgeServerCapability}{MeanOfTaskComputation}$ by default to avoid computation overloading or underloading, where λ can also be adjusted to simulate the computational workload of the vehicles. For the sake of convenience, Table 3 summarizes the parameter settings in our experiments.

Metrics: In order to evaluate our proposed offloading approach, we should firstly determine the suitable μ for vehicles in our simulation environment. Different values of parameter μ may have varying selections of the mobile edge server. As the vehicles should pay for the mobile edge server computation services, the value μ with the highest cost-effectiveness should be selected according to the mobile edge server distributions and vehicle speeds. In our simulation environment settings, we set the parameter μ as 500 for each vehicle.

We evaluate the effectiveness of our proposed offloading approach by means of comparison with the following schemes in our simulations.

1) *Fully local*: Execute the entire computation locally in the vehicle itself for each arriving task.
2) *Fixed ratio remote*: Offload a fixed ratio of computation for each task to the nearest available mobile edge server.

Because the average computation capability of the mobile edge server is three times that of vehicles in our simulation environment, different offloading ratios of task computations may exhibit different performances. We evaluate four fixed ratios: 25%, 50%, 75%, and 100%. In fact, the fully local scheme can be considered as the 0% offloading ratio scheme. In order to simulate the heavy traffic congestion scenario, we let 1 to 15 vehicles randomly move on the road in parallel. We simulate each traffic congestion scenario 1000 times under different mobile edge server distributions, generated by the parameters in Table 3, to obtain the average performance.

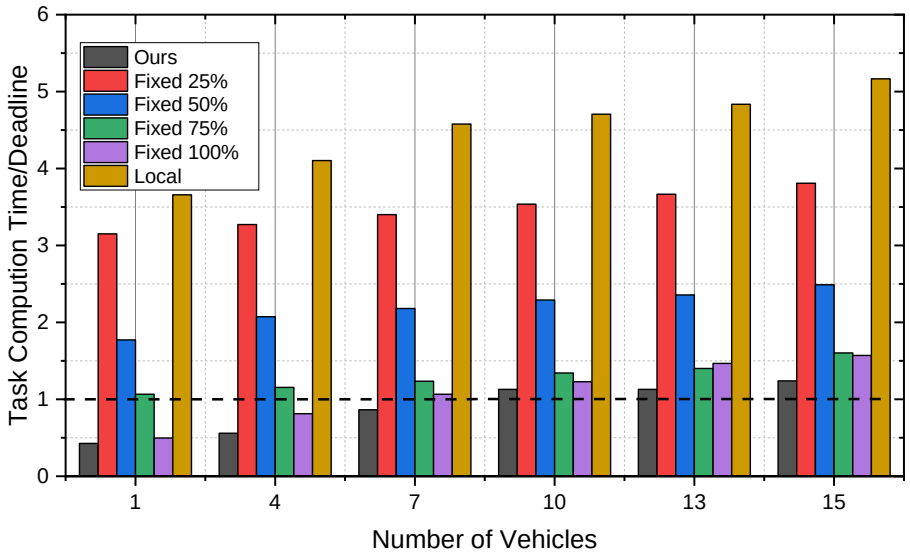


Figure 12: Statistics for average $\frac{\text{task completion time}}{\text{Deadline}}$ under different vehicle traffic congestion scenarios comparing to fixed schemes

Results: It can be observed from Fig. 12 that our approach can outperform all of the offloading schemes under different traffic congestion scenarios. With the increasing number of vehicles moving on the same road, all of the available mobile edge servers have a higher computation workload, thereby inducing certain tasks to missing their deadlines inevitably. In contrast with other schemes, our approach can provide the best to allow additional tasks to be completed before their deadlines. This is because our approach aims to offload each task computation to the optimal nearby mobile edge server with a lighter computation workload. Only by offloading computation to the mobile edge server with a lighter computation workload can the corresponding offloaded computation be completed as soon as possible.

Compared to other schemes, the second-best scheme should be the 100% fixed ratio remote scheme in terms of $\frac{\text{Task completion time}}{\text{Deadline}}$. However, according to Table 4, our approach provides a lower average expense per task among all of the vehicles compared to this scheme, in that our approach utilizes the optimal mobile edge server for offloading the optimal computation ratio to improve the cost-efficiency.

Table 4: Different average expenses per task under different traffic congestion scenarios compared to fixed scheme

	Average expense per task (Edgecoin)		
Number of Vehicles	**Ours**	**Fixed 100%**	**Improvement ratio**
1	147.3800	194.6607	24.29%
2	196.3063	243.3467	19.33%
3	189.9887	215.4687	11.83%
4	171.0208	193.8082	11.76%
5	190.1935	211.7705	10.19%
6	189.3634	209.8057	9.74%
7	172.8491	190.5770	9.30%
8	158.7493	174.9143	9.24%
9	180.5503	197.8293	8.73%
10	175.5615	191.4335	8.29%
11	156.0627	170.1980	8.31%
12	179.9744	196.0651	8.21%
13	156.4169	171.5644	8.83%
14	143.5924	156.5101	8.25%
15	154.0668	167.0073	7.75%

6. Summary and Future Work

In this paper, we firstly proposed a P2P-enabled decentralized mobile edge server management scheme in order to provide efficient offloading services in the edge computing environment. Thereafter, we proposed a deadline-aware and cost-effective offloading approach in the MEC environment, which aims to improve the offloading efficiency among mobile edge servers. The simulation results confirm the efficiency and effectiveness of the proposed approach. In our future work, we plan to determine improved means of estimating execution time of a task, whether it runs on the remote mobile edge server or local vehicle, and thereby improve the performance of our approach.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China, under Grant Nos. 61672276 and 61872219, and the Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing University, and the Royal Society project under Grant IEC170324. Besides, this paper is also partially supported by the scholarship from China Scholarship Council (CSC) under Grant CSC No. 201706190187.

References

[1] K. Zhang, Y. Mao, S. Leng, S. Maharjan, Y. Zhang, Optimal delay constrained offloading for vehicular edge computing networks, in: 2017 IEEE International Conference on Communications (ICC), 2017, pp. 1–6.

[2] A. C. Baktir, A. Ozgovde, C. Ersoy, How can edge computing benefit from software-defined networking: A survey, use cases, and future directions, IEEE Communications Surveys Tutorials 19 (4) (2017) 2359–2391.

[3] M. Satyanarayanan, P. Bahl, N. Davies, The case for vm-based cloudlets in mobile computing, IEEE Pervasive Computing 8 (4) (2009) 14–23.

[4] M. Whaiduzzaman, A. Naveed, A. Gani, Mobicore: Mobile device based cloudlet resource enhancement for optimal task response, IEEE Transactions on Services Computing 11 (1) (2018) 144–154.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet of Things Journal 3 (5) (2016) 637–646.

[6] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the first edition of the MCC workshop on Mobile cloud computing, ACM, 2012, pp. 13–16.

[7] S. N. Shirazi, A. Gouglidis, A. Farshad, D. Hutchison, The extended cloud: Review and analysis of mobile edge computing and fog from a security and resilience perspective, IEEE Journal on Selected Areas in Communications 35 (11) (2017) 2586–2595.

[8] L. Gao, T. H. Luan, S. Yu, W. Zhou, B. Liu, Fogroute: Dtn-based data dissemination model in fog computing, IEEE Internet of Things Journal 4 (1) (2017) 225–235.

[9] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, D. Sabella, On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration, IEEE Communications Surveys Tutorials 19 (3) (2017) 1657–1681.

[10] I. Morris, ETSI drops "mobile" from MEC. Light Reading, New York, NY, USA, Sep. 2016.

[11] P. Kai, V. C. M. Leung, L. Zheng, S. Wang, L. Tao, Intrusion detection system based on decision tree over big data in fog environment, Wireless Communications & Mobile Computing 2018 (5) (2018) 1–10.

[12] T. Wang, J. Zeng, Y. Lai, Y. Cai, H. Tian, Y. Chen, B. Wang, Data collection from wsns to the cloud based on mobile fog elements, Future Generation Computer Systems. doi:10.1016/j.future.2017.07.031.

[13] D. Huang, P. Wang, D. Niyato, A dynamic offloading algorithm for mobile computing, IEEE Transactions on Wireless Communications 11 (6) (2012) 1991–1995.

[14] C. Matt, Fog computing, Business & Information Systems Engineering 60 (4) (2018) 351–355.

[15] S. Yi, Z. Qin, Q. Li, Security and privacy issues of fog computing: A survey, in: International Conference on Wireless Algorithms, Systems, and Applications, Springer, 2015, pp. 685–695.

[16] T. Wang, J. Zhou, M. Huang, M. Z. A. Bhuiyan, A. Liu, W. Xu, M. Xie, Fog-based storage technology to fight with cyber threat, Future Generation Computer Systems 83 (2018) 208–218.

[17] H. Shah-Mansouri, V. W. S. Wong, Hierarchical fog-cloud computing for iot systems: A computation offloading game, IEEE Internet of Things Journal 5 (4) (2018) 3246–3257. doi:10.1109/JIOT.2018.2838022.

[18] K. Liu, J. Peng, X. Zhang, Z. Huang, A combinatorial optimization for energy-efficient mobile cloud offloading over cellular networks, in: 2016 IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–6. doi:10.1109/GLOCOM.2016.7841488.

[19] Y. Kao, B. Krishnamachari, M. Ra, F. Bai, Hermes: Latency optimal task assignment for resource-constrained mobile computing, IEEE Transactions on Mobile Computing 16 (11) (2017) 3056–3069. doi:10.1109/TMC.2017.2679712.

[20] L. Qi, R. Wang, C. Hu, S. Li, Q. He, X. Xu, Time-aware distributed service recommendation with privacy-preservation, Information Sciences 480 (2019) 354 – 364. doi:10.1016/j.ins.2018.11.030.

[21] Y. Xu, L. Qi, W. Dou, J. Yu, Privacy-preserving and scalable service recommendation based on simhash in a distributed cloud environment, Complexity 2017. doi:10.1155/2017/3437854.

[22] L. Qi, X. Zhang, W. Dou, C. Hu, C. Yang, J. Chen, A two-stage locality-sensitive hashing based approach for privacy-preserving mobile service recommendation in cross-platform edge environment, Future Generation Computer Systems 88 (2018) 636 – 643. doi:10.1016/j.future.2018.02.050.

[23] W. Gong, L. Qi, Y. Xu, Privacy-aware multidimensional mobile service quality prediction and recommendation in distributed fog environment, Wireless Communications and Mobile Computing 2018. doi:10.1155/2018/3075849.

[24] L. Qi, X. Zhang, W. Dou, Q. Ni, A distributed locality-sensitive hashing-based approach for cloud service recommendation from multi-source data, IEEE Journal on Selected Areas in Communications 35 (11) (2017) 2616–2624.

[25] M. Zhang, N. Zheng, H. Li, Z. Gu, A decomposition-based approach to optimization of ttp-based distributed embedded systems, Journal of Systems Architecture 91 (2018) 53–61.

[26] Q. Zhao, Z. Gu, H. Zeng, Design optimization for autosar models with preemption thresholds and mixed-criticality scheduling, Journal of Systems Architecture 72 (2017) 61–68.

[27] S. Wan, Y. Zhao, T. Wang, Z. Gu, Q. H. Abbasi, K.-K. R. Choo, Multi-dimensional data indexing and range query processing via voronoi diagram for internet of things, Future Generation Computer Systems 91 (2019) 382–391.

[28] S. Wan, Y. Zhang, J. Chen, On the construction of data aggregation tree with maximizing lifetime in large-scale wireless sensor networks, IEEE Sensors Journal 16 (20) (2016) 7433–7440.

[29] Z. Gao, D. Wang, S. Wan, H. Zhang, Y. Wang, Cognitive-inspired class-statistic matching with triple-constrain for camera free 3d object retrieval, Future Generation Computer Systems 94 (2019) 641 – 653. doi:10.1016/j.future.2018.12.039.

[30] C. Zhou, C. K. Tham, M. Motani, Online auction for truthful stochastic offloading in mobile cloud computing, in: GLOBECOM 2017 - 2017 IEEE Global Communications Conference, 2017, pp. 1–6.

[31] Z. Chang, Z. Zhou, T. Ristaniemi, Z. Niu, Energy efficient optimization for computation offloading in fog computing system, in: GLOBECOM 2017 - 2017 IEEE Global Communications Conference, 2017, pp. 1–6.

[32] L. Liu, Z. Chang, X. Guo, Socially aware dynamic computation offloading scheme for fog computing system with energy harvesting devices, IEEE Internet of Things Journal 5 (3) (2018) 1869–1879. doi:10.1109/JIOT.2018.2816682.

[33] J. Du, L. Zhao, J. Feng, X. Chu, Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee, IEEE Transactions on Communications 66 (4) (2018) 1594–1608. doi:10.1109/TCOMM.2017.2787700.

[34] S. Sundar, B. Liang, Offloading dependent tasks with communication delay and deadline constraint, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 37–45. doi:10.1109/INFOCOM.2018.8486305.

[35] H. Dai, X. Wang, A. X. Liu, H. Ma, G. Chen, Optimizing wireless charger placement for directional charging, in: INFOCOM 2017 - IEEE Conference on Computer Communications, IEEE, 2017, pp. 1–9.

[36] H. Dai, H. Ma, A. X. Liu, G. Chen, Radiation constrained scheduling of wireless charging tasks, IEEE/ACM Transactions on Networking 26 (1) (2018) 314–327.

[37] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, S. Lim, A survey and comparison of peer-to-peer overlay network schemes, IEEE Communications Survey & Tutorials 7 (2) (2005) 72–93.

[38] X. Jin, S. H. G. Chan, Unstructured Peer-to-Peer Network Architectures, Springer US, 2010.

[39] M. F. sallal, G. Owenson, M. Adda, Proximity awareness approach to enhance propagation delay on the bitcoin peer-to-peer network, in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 2017, pp. 2411–2416.

[40] A. Guttman, R-trees: A dynamic index structure for spatial searching, Vol. 14, ACM, 1984.

[41] M. D. Berg, O. Cheong, M. V. Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, Springer Publishing Company, Incorporated, 2000.