Selective Offloading in Mobile Edge Computing for Green Internet of Things

Xinchen Lyu, Hui Tian, Li Jiang, Alexey Vinel, Sabita Maharjan, Stein Gjessing, and Yan Zhang

***Abstract*—Mobile Edge Computing (MEC) provides the radio access networks with cloud computing capabilities to fulfill the requirements of the Internet of Things (IoT) services such as high reliability and low latency. Offloading services to edge servers can alleviate the storage and computing limitations and prolong the lifetimes of the IoT devices. However, offloading in MEC faces scalability problems due to the massive number of IoT devices. In this article, we present a new integration architecture of the cloud, MEC and IoT, and propose a lightweight request and admission framework to resolve the scalability problem. Without coordination among devices, the proposed framework can be operated at the IoT devices and computing servers separately, by encapsulating latency requirements in offloading requests. Then, a selective offloading scheme is designed to minimize the energy consumption of devices, where the signalling overhead can be further reduced by enabling the devices to be self-nominated or self-denied for offloading. Simulation results show that our proposed selective offloading scheme can satisfy the latency requirements of different services and reduce the energy consumption of the IoT devices.**

***Index Terms*—Mobile edge computing, Internet of things, Selective offloading, Scalability, Energy efficiency**

I. Introduction

THE Internet of Things (IoT) is proposed to equip everyday objects with electronics, software, sensors and network connectivity, and bring the vision of a connected world into reality [1], [2]. However, computation-intensive services, such as eHealth, automatic driving and industrial automation, are fast developing and outgrowing the computing and storage capabilities of the IoT devices. Cloud computing offers enormous storage, computing facilities and data sharing opportunities. By offloading the computation and storage from the IoT devices to the cloud through mobile networks, mobile cloud computing can alleviate the computation and storage limitations and prolong the lifetimes of the IoT devices [3].

As the computing units in the core network use shared backhaul resources, mobile cloud computing may not be able to meet the reliability and latency requirements for the IoT services. For instance, the emergency IoT services, such as mobile vehicular connectivity, eHealth and industrial automation, require ultra low latency and extremely high reliability. In addition, the services from smart sensors generate high volumes of data. Uploading the sensed data to the cloud may waste energy and cause traffic congestion in the core network.

Mobile Edge Computing (MEC) is introduced to provide the radio access networks with cloud computing capabilities [4]. For instance, macro/pico/femto base stations may be connected to co-located edge servers, so as to reduce latency, ease the traffic on backhaul links, and deliver reliable services. Typical characteristics of MEC include proximity, high energy efficiency, low latency, high throughput, mobility support and location awareness [4]. These features are highly inline with the requirements of the IoT services.

Offloading incurs extra energy consumption and latency due to the communication between devices and servers. Earlier works on task offloading focus on single-device decision-makings, where the devices make offloading decisions independently to minimize either latency [5] or energy consumption [6]. Due to the resource bottlenecks of edge servers, scalability becomes a key problem in MEC [7]–[10], i.e., there exists a tradeoff between the scale of offloading and the quality of services. Especially in the era of IoT, the offloaded services from millions of devices will exhaust the computational resources at the edge servers, which leads to the increase of processing latency that violates the requirements of the emergency IoT services.

The existing studies resolve the scalability problem by either executing the load balancing among edge servers to aggregate and sustain the workloads [7], [8], or implementing the coordination among mobile devices to select services for offloading [9], [10]. On the other hand, cross-platform IoT services have been enabled via transparent computing in [11], and offloading decisions and resource allocations have been jointly optimized in [12]. However, the tremendous scale of IoT devices necessitates the efficient service discovery and lightweight resource management for heterogeneous IoT services.

In this article, we present a novel three-layer integration architecture including the cloud, MEC and IoT, and propose a lightweight request and admission framework to resolve the scalability problem. Following the proposed framework, a selective offloading scheme is developed to minimize the energy consumption of the IoT devices and further reduce the signalling overhead of MEC. Our main contributions are summarized as follows:

- **Integration architecture of the cloud, MEC and IoT:** The cloud and the geo-distributed edge servers can be complemented with each other to fulfill the various requirements of the IoT services. In the integration, by exploiting the location awareness of edge servers and

Xinchen Lyu and Hui Tian are with State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, China.

Li Jiang is with Guangdong University of Technology, China.

Alexey Vinel is with Halmstad University, Sweden.

Sabita Maharjan, Stein Gjessing and Yan Zhang are with the Simula Research Laboratory and University of Oslo, Norway.

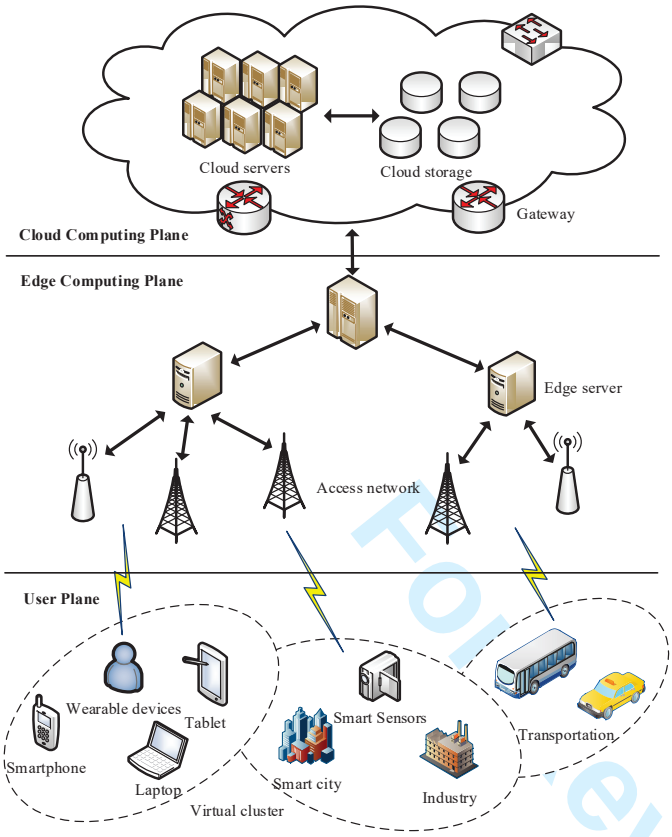

Fig. 1: The three-layer integration architecture of the cloud, MEC and IoT.

low latency interconnects between them, the IoT devices from a wide range can be grouped into virtual clusters for efficient service discovery, and the edge servers are organized in a hierarchical structure to sustain the peak workloads by aggregating services across different tiers of servers.

- **Lightweight request and admission framework:** We encapsulate the latency requirements determined at each device in their offloading requests, so as to decouple the dependency of task partitioning of different devices. The proposed request and admission framework resolves the intrinsic scalability problem of MEC, and can be operated at the devices and edge servers separately, without the need of coordination among devices.
- **Selective offloading scheme:** We propose a selective offloading scheme under the request and admission framework to minimize the energy consumption of the IoT devices, while satisfying the latency requirements of different services. The signalling overhead of MEC can be further reduced by enabling the devices to be self-nominated or self-denied for offloading.

The rest of this article is organized as follows. We present the three-layer integration architecture in Section II, and propose the request and admission framework in Section III. Section IV illustrates our selective offloading scheme and Section V evaluates its efficiency through numerical results. Finally, we conclude the article in Section VI.

II. Proposed Integration Architecture of the Cloud, MEC and IoT

MEC can complement with the cloud to fulfill the various requirements of IoT services, such as low latency, high reliability, location awareness, and bandwidth demanding. Offloading can save the energy of local execution and stretch the storage and computational capacities of the IoT devices. However, the integration of the cloud, MEC and IoT remains challenging in terms of service discovery, service supply and load aggregation. In this section, we present our proposed three-layer integration architecture design, and illustrate the scalability problem in MEC for IoT.

A. Three-layer Integration Architecture

Fig. 1 shows the proposed three-layer integration architecture for the integration of the cloud, MEC and IoT:

1) The *user plane* is the bottom layer consisting of both the mobile users (e.g., smartphones, tablets and laptops) and IoT devices, such as industrial actuators, wearable devices and smart sensors. These devices can be grouped into virtual clusters based on their ownership, and co-location and co-service relationships.
2) The *edge computing plane* is in close proximity to the users. MEC enables cloud computing capabilities within the radio access networks to fulfill the requirements of the IoT services. The geo-distributed edge servers can be organized in a hierarchical structure to efficiently utilize the resources, aggregate the services, and sustain the workloads during peak hours.
3) The *cloud computing plane* is in the core network, and constitutes multiple cloud servers and data centers, which are capable of processing and storing enormous amounts of data.

In this three-layer architecture, the data centers in the cloud can perform complex computing and data analysis, and hence, is responsible for processing the delay-tolerant services that require a large number of storage and computational resources to augment the task processing of the edge computing plane. In specific, the IoT devices sense a multitude of data and offload their services only to the edge servers, instead of offloading directly, so as to the cloud to reduce the required signalling and corresponding energy consumption for the decision-making. The edge servers in proximity to the user plane collect the offloaded services, prioritize the processing of delay-sensitive services to ease traffic on the backhaul links, and offload delay-tolerant services to the cloud based on their workloads. The advantages of the integration architecture mainly include:

- **Efficient service discovery:** In millions of IoT devices, the service discovery, i.e., the search for the right device that can provide the desired data or service, is challenging due to its large scale. In the paradigm of Social Internet of Things (SIoT) [13], the devices can establish social relationships based on their ownerships, locations and services, so as to enhance the process of service discovery. The edge servers are aware of the locations and services of these IoT devices, and then, are responsible for grouping these devices into virtual clusters. The devices in a

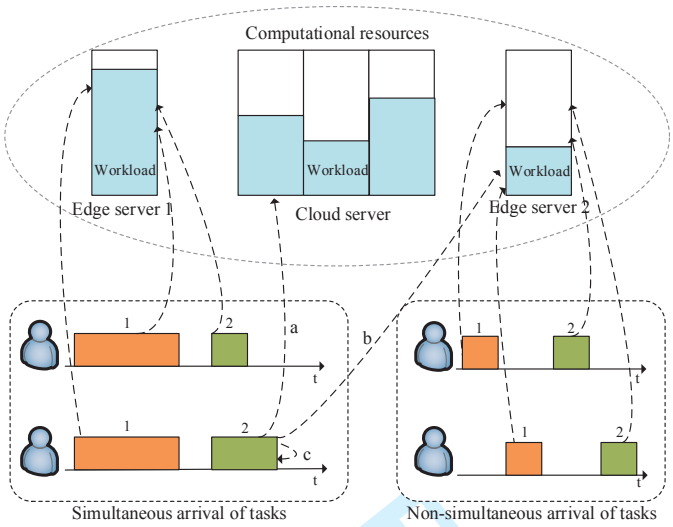

Fig. 2: Scalability problem and selective offloading.

virtual cluster are of similar services, and can aggregate the sensed data to a cluster head to further enhance energy efficiency. Moreover, in the edge computing plane, the edge servers in hierarchical structure can exploit the low latency interconnects between them to address the devices from a wide range of locations and improve the service visibility for the IoT devices.

- **High-performance computing as a service:** The IoT services may have significantly various requirements in terms of latency, data volumes and reliability. Traditional mobile cloud computing cannot fulfill the requirements of low latency and high reliability for industrial automation, eHealth and automatic driving. With the deployment of edge servers, MEC has the potential to make high-performance computing as a service that resolves the latency fluctuation and delivers reliable services.
- **Workload aggregation:** To complement the cloud that cannot fulfill the reliability and latency requirements of the IoT services, the edge servers are deployed in proximity to the users with high flexibility of geo-distribution. However, the deployment of edge servers faces a tradeoff between resource efficiency and service provision during peak workloads. Specifically, the scarce deployment of edge servers will introduce excessive delay due to lack of computational resources, but provisioning more resources through dense deployment could result in poor resource utilization. The tree-structured hierarchical architecture of the edge servers ensures efficient resource utilization by aggregating services across different tiers of servers, and can sustain heavy workloads even during peak hours [14].

However, considering the resource bottlenecks in the edge servers, scalability is an inherent problem in the proposed architecture.

B. Scalability Problem

Services can be executed locally or offloaded to the cloud or edge servers. A large number of tasks that arrive at the same edge server, will exhaust the computational resources and face scalability problems. Fig. 2 illustrates the scalability problem in the proposed integration architecture. When tasks arrive non-simultaneously, the offloaded tasks can utilize the computational resources in turns, and achieve low latency and high reliability as desired. However, when tasks arrive simultaneously, offloading all the tasks to the same edge server may undergo severe resource scarcity and may suffer much longer service latency in consequence.

Especially in the IoT scenario, thousands of devices may wake up concurrently and compete for the limited resources in the edge servers. This not only significantly degrades the user experience for the services that require ultra low latency and high reliability, but also hampers the lifetimes of the IoT devices, since the devices have to stay active when waiting for the computation results. As a result, the resource bottlenecks and workloads of the edge servers introduce a tradeoff between the number of offloaded tasks and the quality of service. In addition, the heterogeneity among millions of devices and their services would intensify the scalability problem.

In Fig. 2, the blue parts of the edge severs and the cloud server denote their workloads (i.e., the congestion of tasks). Specifically, edge server 1 is congested by the offloaded tasks and cannot serve the offloaded tasks promptly, i.e., face the scalability problem. As dictated in Fig. 2, using selective offloading, a device can a) execute its task locally; b) offload its task to edge server 2 under light workload, e.g., via dual connectivity [15]; or c) offload its task to the cloud through the edge computing plane, so as to alleviate the scalability problem and balance the workloads among edge servers.

III. Proposed Request and Admission Framework for the Green IoT

Green networking, which aims at reducing energy consumption and minimizing operational costs, plays an important role in the IoT paradigm. This is even more crucial for the energy-constrained sensors, which are expected to run autonomously for long periods. The longer active time, as described in the scalability problem, would hamper the lifetimes of the IoT devices, which necessitates selective offloading for energy saving. However, the selective offloading schemes in [9], [10] require the coordination among devices, which results in a waste of energy in millions of IoT devices. In this section, we discuss the key challenges in offloading in MEC, and propose a request and admission framework for the green IoT.

A. Challenges of Offloading in MEC

Offloading strategies for task partitioning have been extensively studied in mobile cloud computing, e.g., [5] and [6]. In [5], a dynamic programming approach was developed to minimize the execution latency under cost constraints. In [6], an adaptive receding horizon offloading strategy among multiple devices was proposed, where the solver can adjust its offloading decision according to environmental dynamics (e.g., fluctuating latency). However, compared to the cloud, the edge servers in MEC are heterogeneous and rather limited in terms of storage and computational capabilities. The competition for

the computational resources introduces couplings of decision-makings among devices. As a result, offloading in MEC is more challenging than that of mobile cloud computing.

The studies in [9], [10] verify the scalability problem due to the resource bottlenecks and propose selective offloading schemes. In particular, in [9], the offloading competition among multiple devices was modeled as a sequential offloading game, where the mobile devices made offloading decisions sequentially to obtain a stable offloading result. Assuming that tasks are extremely resource demanding, in [10], only one task was selected for offloading at the same time, and both offline and online algorithms were proposed to optimize the allocation of wireless and computational resources. In [12], a heuristic scheme based on a submodular optimization method was proposed to jointly optimize offloading decisions and resource allocation, but only for delay-tolerant services.

Developing efficient offloading schemes in MEC for IoT faces the following challenges:

- **Coordination costs:** Coordination among devices consumes energy and incurs further latency due to communication overhead. Moreover, the coordination costs increase exponentially with the number of devices, and therefore, enabling coordination may be cost-prohibitive when the scale of IoT (millions of devices) is considered.
- **Couplings among task partioning:** Current selective offloading schemes in MEC [9], [10], [12] only consider offloading services as a whole, instead of offloading part of a service to increase efficiency as in [5], [6]. This is because the resource bottlenecks of the edge servers introduce strong couplings among task partitioning of different devices. Solving the problem optimally requires full knowledge on both the devices and edge servers, including future task arrivals and channel conditions.
- **Heterogeneity of edge servers and devices:** Both edge servers and the IoT devices are heterogeneous in terms of computing and storage capabilities, as well as desirable services. The heterogeneity makes the selection of the offloaded devices even more challenging. For instance, the cloud with abundant computational resources may prefer to execute resource-intensive and delay-tolerant services, while offloading the delay-sensitive services with large volumes of data to edge servers may achieve high reliability and low latency, and reduce the energy consumption on backhaul links.

In summary, the coordination costs necessitate the development of lightweight scheme for the IoT devices, while the couplings among task partitioning require frequent communication among devices and even the exact prediction of future task arrivals and channel conditions to resolve the scalability problem. Moreover, the selection of offloaded devices from a large number of IoT devices is even more challenging due to the heterogeneity of edge servers and devices.

B. Request and Admission Framework for the Green IoT

The proposed request and admission framework is lightweight in terms of signalling overhead, where the devices can send offloading requests independently while the servers only admit selected requests. This is because the dependency of task partitioning among multiple devices can be decoupled by encapsulating the latency requirement in the offloading requests to the computing servers. The latency requirements are set as the deadlines of each task determined by the task partitioning schemes, and the edge servers make best efforts to satisfy the requirements such that the tasks can be executed without delays. Besides, the task partitioning schemes in [5], [6] can help devices select the offloaded server among multiple edge servers. The working procedure of the proposed request and admission framework consists of three stages:

1) Each mobile device partitions its tasks independently, and sends an offloading request to the selected computing server including the latency requirements and other intrinsic features of the device and its service (e.g., the memory requirement, the thread CPU time, and the needed CPU cycles of the service).
2) Each server receives the offloading requests, only admits the selected users for offloading, and pre-allocates the computational resources to satisfy latency requirements.
3) Mobile devices offload their tasks according to the admission results.

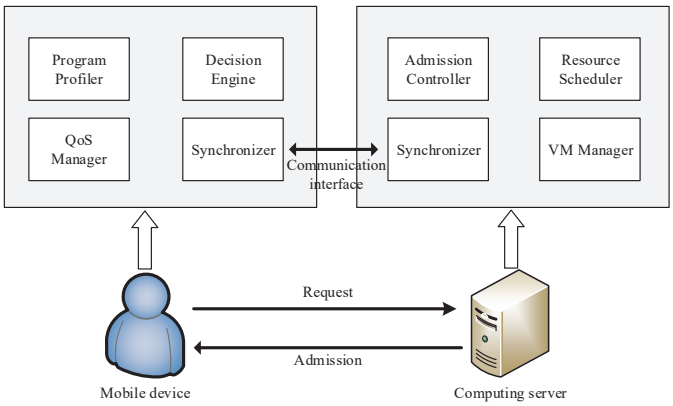

Fig. 3: The proposed request and admission framework.

The proposed lightweight framework enables the selection functionality in both servers and devices to reduce signalling overhead, where only the information on offloading requests and admission results is required to be exchanged through the communication interface. As will be shown in Section IV-C, the signalling overhead can be further reduced by enabling the devices to be self-nominated and self-denied for offloading.

Fig. 3 shows the major blocks of the proposed request and admission framework, which can be operated at the devices and servers separately. The blocks in mobile devices mainly include program profilers, QoS managers, decision engines and synchronizers. In particular, the program profiler monitors the program parameters such as execution time, acquired memory, thread CPU time, number of instructions and method calls; the QoS manager determines the service requirements (e.g., the latency, energy consumption and reliability) and estimates the required latency and energy consumption for execution of the service; the decision engine is responsible for the task partitioning and selects the desired server to send

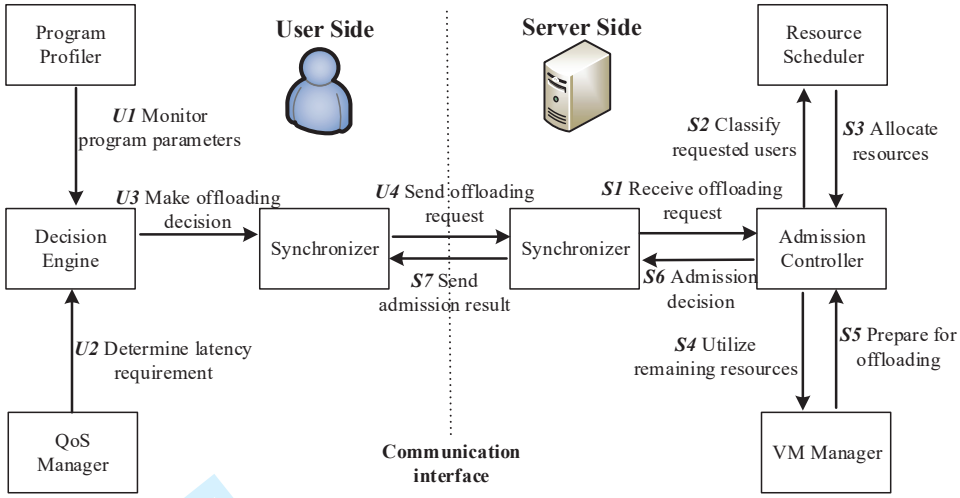

Fig. 4: Flow diagram of our proposed selective offloading scheme.

offloading requests; the synchronizer handles the communication and synchronization between devices and servers in order to ensure integrity of the offloaded data. On the other hand, the blocks in computing servers consist of synchronizers, admission controllers, resource schedulers and Virtual Machine (VM) managers. Specifically, the synchronizer receives the offloading requests; the admission controller selects the devices for offloading according to its current available resources; the resource scheduler and the VM manager allocate the computational resources and activate the VM to prepare for the offloading from the selected devices.

IV. Implementing the Selective Offloading Scheme

In this section, we demonstrate the implementation of the proposed request and admission framework in a multi-user MEC scenario, where the LTE macro base station (BS) is co-located with an edge server of limited computational resources, denoted by f_0. The proposed selective offloading scheme follows the working procedure of the request and admission framework, as summarized in Fig. 4. The working procedure mainly consists of (1) forming the offloading requests at the devices, (2) allocating the resources under the latency requirements at the resource scheduler, and (3) exploiting the heterogeneity among devices to select the energy-saving services for offloading at the admission controller. In the proposed scheme, the delay-sensitive tasks are given high priority for processing, and hence, the delay-tolerant tasks are queued at the edge server under heavy workload. As a result, the edge server only processes delay-tolerant tasks under light workload, and offloads the queueing tasks to the cloud to avoid excessive queueing delay under heavy workload. In the following, we analyze these steps in detail to illustrate the selective offloading scheme.

A. Offloading Request Formation

The offloading requests are formed at each mobile device independently. A task can be described in terms of:

1) Input D_i, including system settings, program codes, and input parameters;
2) The number of CPU cycles required to accomplish the task, denoted by C_i.

The information about D_i and C_i can be obtained through the program profiler. The latency and energy consumption of local execution, denoted by T_i^l and E_i^l, respectively, can be obtained at the QoS manager [9], [10]. Besides, the QoS manager can determine the latency requirement T_i^{req} based on the deadlines determined by the task partitioning schemes [5], [6].

A task can also be offloaded for remote execution to the servers. A typical remote computing approach consists of three stages: (1) uploading the input, (2) remote execution at the edge server, and (3) receiving the computation result. The size of computation result is much smaller than that of input, and the overhead can be neglected [9]. As a result, the total remote computation time of device i can be obtained as $T_i^r = T_i^t + T_i^e$, which is composed of two parts: the uplink transmission time $T_i^t = D_i/R_i$ and the remote execution time $T_i^e = C_i/f_i$. R_i is the achieved data rate of device i for the uplink transmission, and f_i is the allocated computational resources by the edge server. The energy consumption for remote computation of device i can be given by $E_i^r = (p_i/\zeta_i)T_i^t$, where ζ_i is the power amplifier efficiency of device i.

Then, the decision engine can apply the existing offloading strategies in [5], [6], to determine the server to send offloading requests. The offloading request of device i consists of both the latency requirement and intrinsic features of the device.

B. Computational Resource Allocation

At the server side, the synchronizer receives the offloading requests. Then, the problem of interest becomes selecting the offloaded tasks and allocating the limited resources to minimize the system energy consumption, while satisfying the latency requirements of all the offloading requests.

Note that the tasks have to be accomplished before the deadlines, i.e., the latency requirements, determined by task partitioning. As a result, the allocated computational resources

should satisfy $f_i \geq f_i^{\min} = C_i/(T_i^{req} - T_i^t)$, where $f_i^{\min}$ denotes the minimum resources allocated to device i under the latency requirements. In order to enhance the scalability and save energy, the resource scheduler allocates the minimum computational resources to the admitted tasks according to

$$f_i = s_i f_i^{\min} = s_i C_i/(T_i^{req} - D_i/R_i), \quad (1)$$

where $s_i \in \{0, 1\}$ denotes whether the task is admitted for offloading or not (i.e., the task is offloaded when $s_i = 1$).

C. Offloading Decision

The heterogeneity of devices and their IoT services makes offloading more beneficial for some devices, while local execution more beneficial for others. For instance, a delay-sensitive service at a resource-restrained device will benefit from offloading. Therefore, we introduce the following condition to prioritize emergency tasks for offloading.

Condition 1. If $T_i^l > T_i^{req}$, the admission controller selects device i for offloading.

The resource-restrained devices with delay-sensitive tasks satisfying Condition 1 are prioritized for offloading, since their local computing capabilities cannot fulfill the latency requirements, i.e., $T_i^l > T_i^{req}$. Then, the edge server pre-allocates $f_i^{\min}$ resources to these devices, determines its remaining resources $\tilde{f}_0 = f_0 - \sum_i^{T_i^l > T_i^{req}} f_i^{\min}$, and checks the following condition to exclude some devices from offloading.

Condition 2. If $(T_i^r)_{\min} = T_i^t + C_i/\tilde{f}_0 > T_i^{req}$ or $E_i^r \geq E_i^l$, device i executes its task locally.

If this condition is satisfied, even allocating all the remaining resources to device i cannot satisfy its latency requirements, or offloading will not save energy. Thus, the device is excluded from offloading and chooses to execute task locally.

Note that the classification of requested users in *S2* of Fig. 4 can be distributed to the IoT devices to further reduce the signalling overhead of MEC. Particularly, the devices, satisfying Condition 1, can be self-nominated to send an indication to the edge server for offloading prioritization. Then, the edge server broadcasts its remaining resources $\tilde{f}_0$ to the devices. After receiving $\tilde{f}_0$, the devices satisfying Condition 2, can be self-denied for offloading without sending offloading requests. As a result, only the undetermined devices that are neither self-nominated nor self-denied send offloading requests to the edge server, which leads to the reduction of signalling overhead in implementation.

After receiving the offloading requests from the undetermined devices, the resource scheduler allocates the minimum computational resources by Eq. (1). Then, the selective offloading problem can be reduced to a binary linear programming problem, which can be efficiently solved through a branch and bound algorithm.

V. Numerical Results

In this section, numerical results are presented to demonstrate the performance improvements brought by our proposed selective offloading scheme. We consider a single macrocell network with the radius of 250m, which is co-located with an edge server with $f_0 = 10$GHz. The radio communication parameters follow the 3GPP specification. As an example of a complex application, we adopt the face recognition application [9], where $D = 420$kB and $C = 1000$MCycles. The computational capability of devices is uniformly distributed in $[0.5, 1.5]$GHz. We set the latency requirements T^{req} as 1s or 1.5s for delay-sensitive and delay-tolerant applications, respectively. Next, we evaluate the average latency and energy consumption of selective offloading, local execution, and total offloading, when the number of offloading requests N varies from 5 to 20.

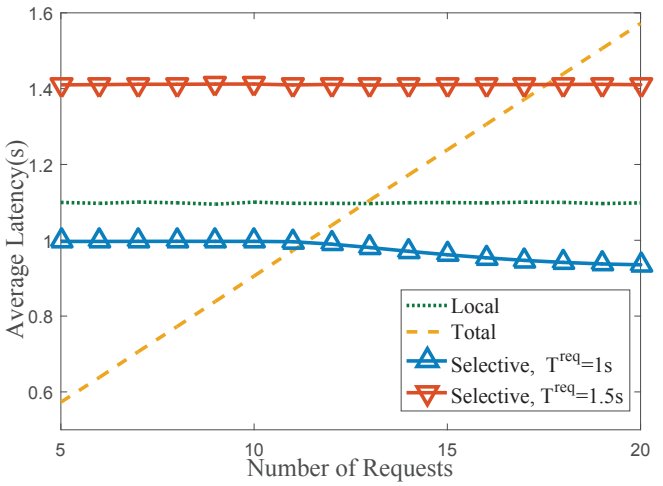

Fig. 5: Comparison of average latency. Selective offloading approaches to the latency requirements.

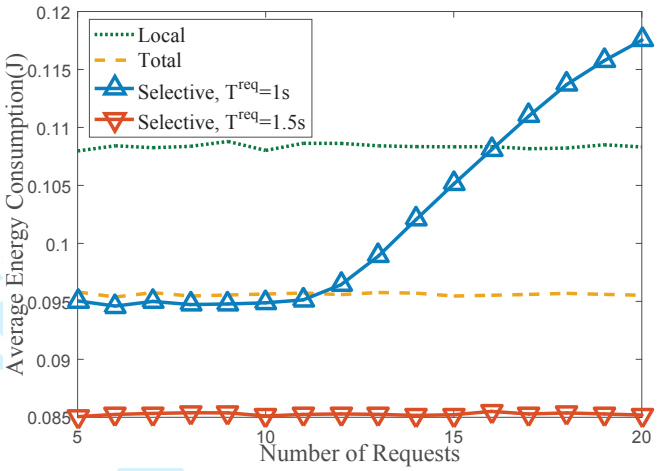

Fig. 6: Comparison of average energy consumption. The tradeoff between latency and energy consumption is inherent.

Fig. 5 shows the average per-user latency in both delay-sensitive and delay-tolerant applications. Our scheme can leverage the computational resources in the devices and edge servers, and make effective task admission to satisfy both stringent and loose latency requirements. In particular, the average latency of our scheme approaches to $T^{\text{req}} = 1$s of delay-sensitive applications, and is about 1.4s for delay-tolerant applications. Irrespective of N, the average latency

of local execution is 1.1s, violating the latency requirement of delay-sensitive applications with $T^{\text{req}} = 1$s. The average latency of total offloading grows linearly with N, and is up to 1.6s when $N = 20$. This is because the scalability problem due to resource scarcity is intensified with increasing N.

Fig. 6 demonstrates the average per-user energy consumption. The energy consumption of total offloading and local execution is relatively stable, and can be lower than that of selective offloading with increasing N when $T^{req} = 1$s. This is at the cost of violating latency requirements, as shown in Fig. 5. We also see that the energy consumption of selective offloading in delay-tolerant applications stays low (0.085J), which is 22% less than local execution. However, for delay-sensitive applications, the energy consumption of selective offloading only stays low and stable when $N < 12$. The energy consumption increasingly grows (up to 11% higher than that of local execution), as N increases from 12 to 20, since offloading the tasks of some resource-restrained devices may not save energy. This reveals the inherent tradeoff between latency and energy consumption in MEC, i.e., latency and energy consumption cannot be minimized at the same time.

VI. Conclusion

In this article, we propose a three-layer integration architecture of the cloud, MEC and IoT, and develop a lightweight request and admission framework to resolve the scalability problem by offloading only selected services. By encapsulating latency requirements in offloading requests, the framework can be operated at devices and edge servers separately without the need to coordinate among devices. The proposed selective offloading scheme can minimize the energy consumption of devices under latency requirements, and the signalling overhead can be further reduced by enabling the devices to be self-nominated or self-denied for offloading. Numerical results show that, by prioritizing the emergency offloading requests, selective offloading is able to satisfy the latency requirements of different services and save energy for the IoT devices.

References

[1] X. Chen, W. Ni, X. Wang, and Y. Sun, "Optimal quality-of-service scheduling for energy-harvesting powered wireless communications," *IEEE Trans. Wireless Commun.*, vol. 15, no. 5, pp. 3269–3280, May 2016.

[2] Y. Zhang, J. Ren, J. Liu, C. Xu, H. Guo, and Y. Liu, "A survey on emerging computing paradigms for big data," *Chinese J. Electron.*, vol. 26, no. 1, pp. 1–12, 2017.

[3] J. Ren, Y. Zhang, K. Zhang, and X. Shen, "Exploiting mobile crowdsourcing for pervasive cloud services: challenges and solutions," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 98–105, March 2015.

[4] "Mobile-edge computing introductory technical white paper," *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

[5] Y. H. Kao, B. Krishnamachari, M. R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Trans. Mobile Comput.*, 2017.

[6] X. Lyu and H. Tian, "Adaptive receding horizon offloading strategy under dynamic environment," *IEEE Commun. Lett.*, vol. 20, no. 5, pp. 878–881, May 2016.

[7] R. Yu, J. Ding, S. Maharjan, S. Gjessing, Y. Zhang, and D. Tsang, "Decentralized and optimal resource cooperation in geo-distributed mobile cloud computing," *IEEE Trans. Emerg. Topics Comput.*, 2016.

[8] R. Yu, X. Huang, J. Kang, J. Ding, S. Maharjan, S. Gjessing, and Y. Zhang, "Cooperative resource management in cloud-enabled vehicular networks," *IEEE Trans. Ind. Electron.*, vol. 62, no. 12, pp. 7938–7951, Dec 2015.

[9] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, October 2016.

[10] W. Labidi, M. Sarkiss, and M. Kamoun, "Joint multi-user resource scheduling and computation offloading in small cell networks," in *IEEE Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2015, pp. 794–801.

[11] J. Ren, H. Guo, C. Xu, and Y. Zhang, "Serving at the edge: A scalable iot architecture based on transparent computing," *IEEE Netw.*, 2017.

[12] X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Trans. Veh. Tech.*, vol. 66, no. 4, pp. 3435–3447, April 2017.

[13] A. M. Ortiz, D. Hussein, S. Park, S. N. Han, and N. Crespi, "The cluster between internet of things and social networks: Review and research challenges," *IEEE Internet Things J.*, vol. 1, no. 3, pp. 206–215, June 2014.

[14] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE INFOCOM 2016*, April 2016, pp. 1–9.

[15] Y. Wu, K. Guo, J. Huang, and X. S. Shen, "Secrecy-based energy-efficient data offloading via dual connectivity over unlicensed spectrums," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3252–3270, Dec 2016.