Wireless Networks
https://doi.org/10.1007/s11276-018-1824-y

Circa: collaborative code offloading among multiple mobile devices

Xueling Lin[1] · Jingjie Jiang[1,2] · Calvin Hong Yi Li[3] · Bo Li[1] · Baochun Li[4]

Abstract
For mobile users who seek extra power or computing resources to perform computation-intensive tasks, code offloading to remote infrastructures is a promising solution. However, most of the recent works mainly target on code offloading from single mobile devices to remote cloud servers, which restricts the potential of offloading only to devices with available Internet access. Moreover, offloading to remote cloud computing platforms is not always guaranteed to be time efficient and energy conserving. In this paper, we propose *Circa*, a framework that demonstrates the feasibility of code offloading among multiple mobile devices in the same vicinity, leveraging the presence of *iBeacons*. *Circa* eliminates the costs incurred by connecting to a remote cloud and running virtual machine instances in the cloud. With iBeacons, neighbouring devices can discover and support one another through collaborative code offloading with short-range communication, obviating the need for centralized servers. We also propose task allocation algorithms to select reliable collaborators among nearby mobile devices and disseminate the computation intensive tasks among them efficiently in a fair fashion. The performance of the task allocation algorithm is evaluated based on three different mobility models. We also implement a prototype of the *Circa* framework on an iOS platform and validate its feasibility and efficiency using iOS devices. According to the experimental results, by involving nearby mobile devices as collaborators, *Circa* is able to reduce the total execution time of an offloaded task substantially, while preserving the satisfactory performance of mobile applications.

Keywords Cloud computing · Mobile computing · Remote offload · Mobile communication

1 Introduction

With the incredible prevalence of smartphones, mobile applications are becoming increasingly ubiquitous, taking the leading role in personal communication as well as information processing. Nowadays, numerous native applications on mobile and wearable devices have blossomed to servers as more convenient substitutes to web applications.

Nevertheless, since mobile devices are still resource constrained, the execution of computation intensive or power draining applications still puts a heavy burden on devices, which may lead to a poor performance and enormous energy consumption. For instance, a device with low-end CPU may exceed its computational limits before obtaining the translation results of a long voice message processed locally; a low-battery device may even power off before the computation finishes.

Computation offloading is a promising technique to alleviate such problems. With abundant computing and power resources in cloud computing infrastructures, it is

✉ Xueling Lin
xlinai@cse.ust.hk

Jingjie Jiang
jiang.jingjie@huawei.com

Calvin Hong Yi Li
cli78@jhu.edu

Bo Li
bli@cse.ust.hk

Baochun Li
bli@ece.toronto.edu

[1] Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China

[2] Future Network Theory Lab, Huawei Technologies, Hong Kong, China

[3] Department of Computer Science, Whiting School of Engineering, Johns Hopkins University, Baltimore, USA

[4] Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada

Published online: 05 September 2018

feasible to offload a portion of a computational task of resource-intensive applications to cloud servers [1–3]. This can further enable numerous types of applications, such as natural language translation, face recognition and augmented reality, to run on tiny handhold, wearable or body-implanted platforms. The increasing level of parallelism can also enhance the performance of mobile applications.

Unfortunately, offloading to remote cloud computing platforms is not always guaranteed to be time efficient and energy conserving. When the network bandwidth is fairly limited, it may be too slow to transmit data between mobile devices and remote servers; when the network status is highly unstable, maintaining a connection to a cloud might consume more energy than local computation. To make matters worse, some devices, such as smartwatches, have no access to clouds, due to hardware restrictions or signal attenuation. In addition, the rental expense of cloud servers could be too exorbitant for a mobile user.

In this paper, we propose to detour from the conventional wisdom of offloading to the cloud. Instead, we focus on the concept of collaborative offloading among multiple mobile devices that are in close proximity to one another. With the presence of *iBeacons* [4], an exciting technology enabling new location-aware applications, devices entering into a local area can discover each other through the unique identifications distributed by a common iBeacon. A device without sufficient computing capability or battery life can then turn to its "neighbours" for help. Mobile devices that are in the same vicinity can be involved in the same task and work together.

We design and implement *Circa*, a new collaborative offloading framework that identifies and selects a group of mobile devices in the same vicinity, and involves them into the same offloading task. With such collaborative offloading among nearby devices, the costs involved in running instances in the cloud and setting up connections to the cloud will be eliminated, or at least reduced significantly. Besides, devices that are incapable of connecting to a cloud can also enjoy the benefits brought by computation offloading. Furthermore, since the group of selected mobile devices follow similar movement patterns in a relatively stable mode, the probability of disconnection and transmission errors is much lower.

The key to our framework is to identify and select a group of adjacent mobile devices with the assistance of iBeacons, and involve them into the same offloading task. Our goal is to choose the available devices with abundant computation resources that are potentially connected with each other as the most reliable collaborators. The computation task is then divided and disseminated among these devices in a fair fashion. We propose a task allocation algorithm and a task priority scheduling algorithm to achieve the aim of cutting down the total completion time of the entire task.

To investigate the performance benefits brought by the *Circa* system, we have carried out simulation experiments based on different mobility models, including the Random Waypoint Model, the Gauss–Morkov Model as well as the Reference Point Group Model. We have also implemented *Circa* on the iOS development platform, and evaluated its effectiveness and performance in a number of practical scenarios. As demonstrated by the experimental results, by involving nearby mobile devices as collaborators, the total time spent on the execution of the offloaded task can be reduced substantially.

Organization The remainder of this paper is organized as follows. Section 2 discusses our contributions in the context of related work. In Sect. 3, we introduce the background knowledge of iBeacons and describe the motivating examples of our framework under different scenarios. In Sect. 4, we present the workflow of collaborative offloading, including setting up a connection among devices and determining the most reliable and valuable collaborators. Task allocation and priority assignment algorithms are proposed in Sect. 5 to minimize the overall completion time of a task when it is disseminated among a group of mobile device collaborators. We undertake an extensive evaluation of the *Circa* system in Sect. 6. We evaluate our framework through prototype-based experiments in Sect. 7. Some possible extensions to our framework are discussed after the concluding remarks in Sect. 8.

2 Related work

2.1 Code offloading to remote servers

Most existing works, such as *MAUI* [1], *CloneCloud* [2], *ThinkAir* [3] and *COMET* [5], mainly focus on code offloading between mobile devices and cloud servers. With the objective of reducing local energy consumption, such frameworks try to determine which parts of an application could be offloaded to a remote server at runtime. The offloading granularities vary from function calls to running threads.

Similar working schemes include *Cloudlets* [6], which is a trusted, resource-rich computer or cluster of computers that is well-connected to the Internet and available for nearby mobile devices. *Cuckoo* [7] also presents a feasible framework that offloads mobile applications to a cloud server equipped with a Java stub/proxy model. It offers a simple way for the application user to collect remote resources. However, neither of them is easy to implement as a practical framework in reality, since the mobility nature of mobile devices is not taken into account.

There are also other works that aims at improving the efficiency of the computation offloading task. *Hermes* [8] searches for the optimal strategy that balances between latency improvement and energy consumption of mobile devices during the offloading task. Chen et al. [9] proposes a game theoretic approach for the computation offloading decision making problem among multiple mobile device users for mobile-edge cloud computing. In this work, the problem is formulated as a multi-user computation offloading game, which always admits a Nash Equilibrium. Zhou et al. [10] also provides a context-aware offloading decision algorithm, aiming to selecting wireless medium and target cloud resources based on the device context at runtime. Flores et al. [11] contributes by proposing crowdsourced evidence traces as a novel mechanism to optimize offloading decision making.

Nevertheless, most of these works only consider offloading via the Internet to powerful servers in a remote cloud. Involving remote servers, however, requires an Internet connection through a cellular or Wifi network. Since the network status between cloud servers and a mobile device is highly unstable, the device may lose its Internet connection to remote servers before the computation finishes. Moreover, some devices, such as smartwatches, may not even have Internet access due to their hardware constraints.

Our work is closely related to those that also consider a collaborated scheme in cloud computing, which regard devices as a part of cloud, utilizing the remaining resources of mobile devices. Usually the cloud server acts as the controller and scheduler of the collaboration among multiple devices. However, in our framework, one or more initial mobile devices will take the leading role to choose and coordinate the collaborators.

2.2 Code offloading among mobile devices

As mentioned in the previous section, if we only consider offloading via the Internet to powerful remote servers, the long WAN latency and the energy consumed in the transmission process will be an inevitable problem. Recently, there has been a growing interest in utilizing mobile devices in building a mobile computing system. To avoid the long latency which occurs in WANs, the idea of "Mobile Cloud" is proposed in order to take advantage of computing power in proximity to achieve execution and energy saving.

To investigate how mobile devices can function as resource providers and mutually help each other, Doolan et al. in [12] design a mobile version of the standard message passing interface over Bluetooth. They employ a fully interconnected mesh structure so that all nodes can directly communicate with each other. Due to the limited communication range of Bluetooth, only devices in a small physical area can form such a fully-interconnected local network. Nevertheless, it is non-trivial to dynamically identify such fully connected clusters under mobile environments. In contrast, the *Circa* framework utilizes the deployment of iBeacons to discover all the devices within close proximity to one another, and enables local cooperation among these devices.

Derived from Hadoop [13], *Hyrax* [14] also explores the possibility of using mobile phones as resource providers. However, a central server is required to collect device information and coordinate computation jobs. Therefore, *Hyrax* shares the same deficiencies as the remote offloading frameworks: the unpredictable delay between a remote server and a device may lead to severe performance degradation, incurring intolerable computation latencies. Furthermore, *Hyrax* only works for a set of smartphones that connect to each other via plain TCP/IP sockets, and thus cannot be applied to a realistic setting where smartphones do not have global, unrestricted IP addresses. Leveraging iBeacons, devices in our framework can efficiently identify one another through the unique IDs distributed by their neighboring beacons.

Another similar work is presented by Black and Edgar [15], which demonstrates the feasibility of using mobile devices as part of a grid by implementing the BOINC [16] client on Apple iPhones. Work units are downloaded from a BOINC server and executed on an iPhone via a virtual machine emulating an $\times 86$ processor. Afterwards the results are uploaded to the cloud. Similar working schemes include the *Codor* system architecture [17], SETI@home [18] and folding@home [19]. However, there is still no consideration about the mobility patterns and the collaboration schemes of the mobile devices.

Serenity [20] is another work designed for computation offloading among intermittently connected mobile devices. It enables a mobile initiator to use the computational resources of other mobile systems in its environment. The authors in [21] investigates how to map a job to a set of available devices so that the devices can perform a portion of the job with their own resources. Works like [21–26] also share a common objective to allow a mobile device to utilize other computing resources to accomplish its own work. The target of these works is to save computation for the offloading devices that offload computation to other devices.

Specifically, [27] extend the idea of code offloading to wearable computing by proposing a three-layer architecture, which consists of wearable devices, mobile devices and remote cloud servers. In particular, a portion of computation tasks are offloaded from wearable devices to local mobile devices or remote cloud such that even applications with heavy computation load can still be upheld on wearable devices.

However, in all the works mentioned above, the incentive of participants and collaboration is not taken into account. In our framework, before a device borrows computation power from other mobile devices with the same job objective and shares the workload with them, it must also donate parts of its own resources to them. Thus free-riders are prevented.

We proposed the idea of offloading the computation-intensive task to nearby devices with the assistance of iBeacons in [28]. In this paper, we extend the previous idea by formally describing the framework and designing task allocation algorithms in the offloading process, and demonstrating the feasibility and efficiency og the framework with more experimental results.

3 Background and motivation

In this section, we first introduce iBeacons, a new wireless location-based transmitter technology that we adapt in *Circa* to detect mobile devices in its proximity, in order to set up connections among these devices. Afterwards, we illustrate two practical scenarios that motivate the design of the *Circa* framework.

3.1 iBeacon: a proximity detector

How can mobile devices easily discover each other as neighbours? This is one of the major and crucial problems that we need to solve in our framework design. We propose iBeacon, a new wireless location-based transmitter that detects mobile devices in its proximity, as a solution to this problem. In this subsection, we give a brief introduction of the iBeacon technology and demonstrate the workflow that we adapt into the *Circa* working scheme as the key factor to help devices connect with one another.

Introduced in iOS 7, iBeacon is an exciting technology proposed by Apple Inc. which enables new location-aware information and services for mobile devices. Leveraging Bluetooth low-energy (BLE) for short-range communication, a device with iBeacon technology can establish a region around itself. Each iBeacon (called a beacon) has a unique 20-byte ID, which is divided into three sections: proximity UUID (16 bytes), major number (2 bytes) and minor number (2 bytes). A beacon continuously broadcasts its unique ID via BLE to devices in its close proximity. The broadcast coverage of a beacon could be as small as 2 inches and as large as 230 feet away, since its Bluetooth signal can diffract, be interfered with or be absorbed by its surroundings.

Any mobile device entering such beacon coverage can receive its unique ID without *a priori* and explicit pairing procedures. Devices probe a beacon's signal and refresh their relative distance at fixed intervals, which is usually set to 1.0 s in most cases. The concept of beacons is similar to the small light towers that are installed in fixed locations and broadcast their presence to ships (smartphones) in the vicinity.

A related event will be triggered by the application of a mobile device when it enters or exists in the region covered by the beacon, which can be applied to inform the device about its geographical location changes [4]. The coverages of different beacons overlap with each other. Mobile devices in the overlapping area can pick up the unique IDs broadcasted by multiple beacons, and estimate their distances to those beacons by measuring the received signal strength (RSSI) [29]. The closer a device is to a beacon, the stronger the corresponding signal it can receive. Thus, the mobile device will be able to learn its relative location based on the knowledge of its estimated distance to each beacon.

The application scenario of iBeacons is simple and straightforward. As shown in Fig. 1, all the devices residing within a beacon's coverage receive the unique ID of the corresponding beacon. By checking the beacon IDs picked up by others, a device can easily discover the ones that stay in the same beacon region and form a collaborative group with them. As the key to identifying such groups, iBeacons play crucial roles in the *Circa* framework.

3.2 Motivating scenarios

To better demonstrate the highlights of collaborative code offloading, we present two scenarios as motivating examples. In these scenarios, devices can come across their hardware limitations or failures, get rid of the poor network performance resulting from their mobility, and embrace the benefits of code offloading.

Suppose a scholar is attending a meeting held in a conference hall without available WiFi network. Unfortunately, he forgets to bring his smart phone and merely wears a smart watch, which has no cellular network. However, with the mobile collaborative offloading framework, the scholar can still enjoy the speech recognition and

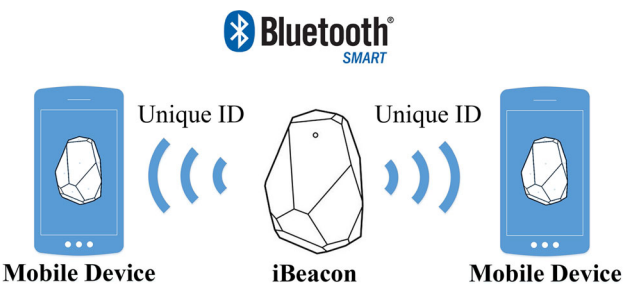


Fig. 1 The way an iBeacon works with mobile devices

face recognition application, by sending the speech and image data to a nearby smart phone that is more powerful via Bluetooth.

For another example, a group of users are staying in front of a piece of glorious artwork in a museum. Each user uses their mobile device to take a snap of the artwork from their own unique angle. Thanks to the help of the mobile collaborative offloading framework, they can easily distinguish each other as nearby collaborators, and are thus capable of building a 3D model of the artwork together by allocating the 3D modeling task among one another.

The first example above implies the potential benefits brought by collaborative offloading when hardware failure or network disconnection appears in some devices. In contrast, the second example demonstrates the value of collaborative offloading in location-based activities, where the collaborating devices share the same objective to complete a computationally intensive task in a parallel fashion.

One essential incentive of collaborative offloading is that all of the participating devices should provide some types of original resource before they can enjoy the result. In the first example, the user with the smart watch shares the recording file of a talk among the nearby users before he can obtain the translation result. As for the second example, everyone donates a snap of the artwork from their own angle before obtaining the complete 3D model. Therefore, in the mechanisms for both scenarios, the free rider problem is inherently avoided.

4 System design and job model

In this section, we present the design of the *Circa* framework. *Circa* consists of a code offloading workflow that involves three basic phases. In the first phase, all mobile devices that stay in the region of a beacon establish connections with one another. In the second phase, *Circa* determines which devices should be involved in the task as reliable collaborators according to different factors, including their battery levels, memories and mobility patterns. Lastly, we apply a task allocation algorithm to distribute subtasks among each of the participating devices to proceed via Bluetooth.

4.1 Establish connections based on iBeacons

In this phase, we aim at setting up a connection between each pair of devices that are within the region of a beacon.

There are several beacons that broadcast their unique IDs in a certain area. Each mobile device that enters in the Bluetooth transmission regions of the beacons will automatically pick up the ID of each beacon. Once a device starts our application, it can detect all beacons in its proximity and estimate its distance to those beacons. Devices probe the signal of beacons every second.

Afterwards, a peer discovery mode inherited from the GameKit framework [30] is applied to establish connections between devices. Specifically, a GKSession object provides the devices with the ability to discover and connect to nearby mobile devices using Bluetooth. Devices connected to an ad-hoc wireless network are known as peers. A peer is synonymous with a session object running inside our application. Each peer creates a unique peer identification string (session ID), which is used to identify itself to other peers in the network [31]. In the *Circa* framework, sessions discover other peers based on a session mode, which is set when the session is initialized according to the role of the devices as explained below.

A mobile device should choose one of the following three roles:

Server A mobile device acts as a server of a session when it needs to offload a portion or all of its current task to other devices, if at least one of the following 4 conditions is satisfied:

a) its remaining battery capacity falls below a certain threshold;
b) according to its historic profiling, both of the energy consumption and time duration of the requested task exceed specific thresholds;
c) due to hardware constraints, offloading the current task to other devices is of necessity, e.g. the speech recognition task run by the smartwatch mentioned in Sect. 3;
d) it is required to participate in a task that demands collaboration with others, e.g., the collaborative 3D modeling task mentioned in Sect. 3.

Client A device serves as a client if it is willing to accept an offloading request, when both of the following two requirements are met:

a) its remaining battery capacity is over a certain threshold;
b) according to its historic profiling, both the energy consumption and time duration of the requested task are below specific thresholds, or it is a task that demands collaboration with others.

Bystander A device acts as a bystander, if it is neither in demand for offloading tasks to others, nor willing to accept offloading requests from others. A bystander does nothing in the whole process.

A server advertises its service type with a session identification string (sessionID) which is the same as the unique ID of the beacon that it picks up. If a server collects more than one beacon ID, it will advertise a set of session

IDs that contains every beacon ID it obtained. A client records all the beacon IDs it has picked up as session IDs. It keeps listening for the session IDs broadcasted by other servers and only connects to those servers that have one or more matching session ID. Thus, devices are able to establish connections with one another if and only if they stay in the region of the same beacon.

We illustrate the first phase through an example shown in Fig. 2. Suppose there are two mobile devices, A and B. They stay in the region of a certain beacon, whose unique ID is '7' for short. When A and B start our application, they can detect all beacons in proximity as well as their updated distances. Meanwhile, A and B record the session ID which is exactly the same as the unique ID of the beacon ('7'). When A needs to offload tasks, it directly advertises '7' as its session ID. If B is willing to collaborate with others, it will start searching for other devices that are broadcasting '7' as the session ID. Then A and B are able to connect to each other successfully as a peer with the session ID '7', and proceed to the next phase.

4.2 Determine the potential collaborators

For a holistic view, our goal is to minimize the overall time and energy consumption of all users accomplishing the job. We assume that all users involved share the common goal of finishing the job. What we need to do in this phase is to pick out some of the mobile devices that have already built up a connection to the server as the clients of the whole computationally intensive task.

Although only devices that are geographically close to each other can set up connections, it is still possible to encounter disconnection problems if some collaborators walk away during the offloading process without a prompt or just simply shut down their devices. To avoid disconnection in the middle of an offloading process, we need to conduct a reliability analysis to select the most reliable devices as actual collaborators. If an unstable node could be identified beforehand, the system can take precautions by promoting task redundancy.

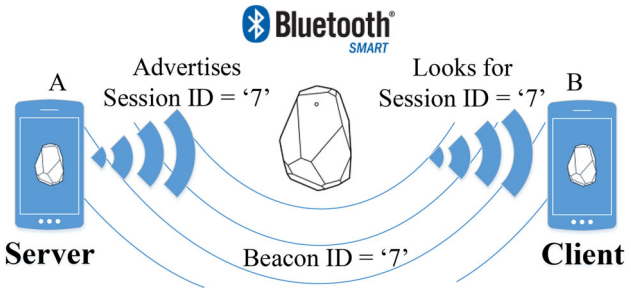


Fig. 2 Peer connection example

There are a few principles that a mobile device should satisfy to become a qualified collaborator. Besides, there are also some key observations that guide our selection process. We discuss these principles and observations in the rest of this section.

4.2.1 Communication relationship

In the *Circa* framework, the server device processes the input data and splits the data into multiple subtask segments, which are passed to the clients. These parallel tasks are executed independently on the clients. The workload of each subtask is determined in the allocation algorithm, which basically relies on the computation ability and resource of the devices that are supposed to execute these tasks. In our design scheme, the workload of each of the subtasks is set to be equivalent to each other. Afterwards, the computation results will be transferred back to the server. Therefore, the determination of the potential and reliable collaborators is a key problem in the *Circa* framework. For each server device, we consider all the other devices in its communication relationships as potential collaborators.

To describe the communication relationships of the potentially collaborative mobile users, for each beacon, we use a resource graph $\mathbb{G}^r = (\mathbb{V}, \mathbb{L})$, with the set of nodes $\mathbb{V} = \{v_0, v_1, \ldots, v_{i-1}\}$. Each of the nodes in the graph represents a mobile device that stays in the transmission region of this beacon. The internal node represents the server and the leaf nodes represent the clients. Since one client will only match one server, we will not consider the communication relationship between the clients. Each of the links $l_{i,j} \in \mathbb{L}$ of the graph represents the communication channel from user i to user$j$. If $l_{i,j} = 1$, it means device i and device j are able to exchange resource with each other. In the beginning, there will be edges between some pairs of nodes in $\mathbb{V}$ that stay in the same region and automatically set up a connection with each other. Since devices in the *Circa* framework probe a beacon's signal and refresh their relative distances at fixed intervals, the graph will also update at fixed intervals. In the *Circa* framework, the resource graph updates every second.

We assume that currently there are m mobile devices connected to a server as clients. After connections are established, each client sends a tiny piece of message to the server, which includes its battery level (in percentage), available memory (in MB) and the time span (in seconds) that it has been in the current region. Based on the reliability analysis, the server then generates a ranked list of the m devices.

4.2.2 Neighbourhood

We give the following definitions of neighbourhoods of a mobile device in our framework.

Definition 1 One-hop neighbourhood ($\mathbb{N}_1(v_i)$)

$\forall v_i \in \mathbb{V}$, the 1-hop neighbourhood of node v_i is defined as

$$\mathbb{N}_1(v_i) = \{v_j \mid v_j \in \mathbb{V}, l_{i,j} = 1\}$$

The physical meaning of 1-hop neighbourhood is the set of the devices that can be directly reached from device v_i via 1-hop neighbours within the transmission regions of one or more beacons.

Definition 2 h-hop neighbourhood ($\mathbb{N}_h(v_i)$)

$\forall v_i \in \mathbb{V}$, the h-hop neighbourhood of node v_i is defined as

$$\mathbb{N}_h(v_i) = \left\{ v_k \middle| \exists v_j \in \mathbb{N}_{h-1}(v_i), v_k \in \mathbb{N}_1(v_j), v_k \notin \bigcup_{i=1}^{h-1} \mathbb{N}_i(v_i) \right\}$$

The physical meaning of 1-hop neighbourhood is the set of devices that can be reached from device v_i via its multiple-hop neighbours within the transmission region of several beacons, by passing at most h links.

Observation 1 Devices that are in the h-hop neighbourhood ($h > 1$) of the server should be considered to be less reliable and ranked lower.

Suppose that there is a device v_m, who is in the h-hop neighbourhood ($h > 1$) of the server v_s. v_m can only exchange resource and information with v_s by offloading its resource to one of the devices, v'_m, which is in both $\mathbb{N}_1(v_m)$ and $\mathbb{N}_{h-1}(v_s)$. Then v'_m should stay on to pass the resource to v''_m, which is in $\mathbb{N}_1(v'_m)$ and $\mathbb{N}_{h-2}(v_s)$. All in all, the resource must be transmitted h times from one device to another before it reaches the server v_s. The transmission process will be successful if and only if all the intermediate devices stays and works properly in their current region. If one of the intermediate devices walks away during the computation and offloading process, the results would never be transferred to the destination. In other words, the communication between the devices in the h-hop neighbourhood ($h > 1$) of the server depends on the devices in the i-hop neighbourhood ($0 < i < h - 1$) of the server.

To simplify the whole communication process, we do not take the devices in the h-hop neighbourhood ($h > 1$) of the server into consideration. Instead, we only choose the devices that are in 1-hop distance from the server as our potential collaborators. However, during the computation and offloading process, when one collaborator, v_j, disconnects with the server and if it still maintains the connection to one of the 1-hop neighbours of the server, v'_j, v_j will consider transferring the results of its subtask to v'_j and let v'_j and the server communicate with each other. The details of this process are discussed in the next section.

4.2.3 Battery level and available memory

We make the following observation based on the battery level and available memory of a device.

Observation 2 Devices with higher battery and available memory should be considered to be more reliable and ranked higher.

Since the probability that the aforementioned devices running out of battery or memory during the offloading process is much lower, it is less likely for us to encounter disconnection if they are chosen to be the collaborators. Let b_i be the battery level (in percentage) of the i-th mobile device among the $1, 2, \ldots, m$ devices. Let a_i be the available memory (in MB) of the i-th mobile device. Thus the energy level E_i of the i-th mobile device can be formulated as follows:

$$E_i = b_i \times a_i \tag{1}$$

4.3 Residence time

Definition 3 Residence Time is the time duration that a client stays in the 1-hop neighbourhood of a certain server.

Observation 3 Devices whose residence time are too short or too long should be considered to be less reliable and ranked lower.

Mobile devices that enter in the 1-hop neighbourhood of the server for just a few seconds are more likely to be passers-by, who will leave in a short time. On the contrary, those with longer residence time tend to leave soon, since they may finish their tour in the current region soon, and would like to move to another region. Hence the probability of disconnection would be higher if we select them as collaborators.

We use a *normal distribution* to represent the availability of mobile device, depending on the time that it has already stayed in the 1-hop neighbourhood of the server. Let μ be the average time span (in seconds) that those qualified collaborators have been connected to the server. Let σ be the standard deviation of the time span (in seconds). Let t be the time span (in seconds) that the i-th mobile device has been connected to the server as its 1-hop neighbour. Thus the availability $\varphi_i(t)$ of the i-th mobile device can be formulated as follows:

$$\varphi_i(t) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{2}$$

In the *Circa* framework, each server will record the average time duration that its collaborators have stayed in its 1-hop neighbourhood as well as the standard deviation of the time span. These two values will update after the resource graph is updated at a fixed time interval.

4.3.1 Ranking

Finally, let R_i be the reliability level of the i-th mobile device in the 1-hop neighbourhood of the server, which can be formulated as follows:

$$R_i = E_i + \alpha\varphi_i(t) = b_i a_i + \alpha \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{3}$$

In this function, α is introduced to combine the energy level and availability of a single device. Therefore, arranging reliability of each device from high to low, a ranked list $\mathbb{P} = \{m_0, m_1, m_2, m_3, \ldots\}$ will be generated. A server will choose those devices with higher rankings as its collaborators.

Algorithm 1 ReliabilityRanking

Require: $s, \mathbb{T}, threshold, tolerance, \alpha$
/* s is the server, $\mathbb{T}$ is the subtask set delivered by the server, $threshold$ is the threshold of the collaborators quantity, $tolerance$ is the delay tolerance of unreceived subtask result, α is used to combine the energy level and availability of a single device. */
s.connectionDurationList = $\emptyset$
$\mathbb{T}$.suspendedSubtasksList = $\emptyset$
$\mathbb{T}$.executingSubtasksList = $\emptyset$
$\mathbb{T}$.finishedSubtasksList = $\emptyset$
$\mathbb{T}$.oldCollaboratorsList = $\emptyset$
while finishedSubtasksList.size() $<\mathbb{T}$.size() **do**
  $\mathbb{P} = \emptyset$
  $\mathbb{N}_1(s)$ = oneHopNeighbours(s)
  for all $v \in \mathbb{N}_1(s)$ **do**
    $\mu \leftarrow$ connectionDurationList.getAverage()
    $\sigma \leftarrow$ connectionDurationList.getStandardDeviation()
    v.ranking $\leftarrow v$.getBatterry() $* v$.getMemory() $+\alpha*$ normalDistribution(μ, σ, v.timeDuration)
    $\mathbb{P} \leftarrow v$
  end for
  collaboratorsSort($\mathbb{P}, threshold$)
  if $v \notin \mathbb{N}_1'(s)$ **then**
    s.connectionDurationList.add(v.timeDuration)
  end if
  if contact.controllbleState = TRUE **then**
    CONTROLLABLECONTACTALLOCATION($\mathbb{T}, \mathbb{P}$) /* will be discussed in Section 5.1.1 */
  else
    UNCONTROLLABLECONTACTALLOCATION($\mathbb{T}, \mathbb{P}, tolenrance$) /* will be discussed in Section 5.1.2 */
  end if
end while

We use a threshold here to determine the quantity of mobile devices we need to involve. In order to achieve the best performance with the least energy consumption and transmission latency, it is important for us to decide how many devices should be picked out as collaborators from the ranked list, which is obtained in the previous step.

The main challenge of this problem is to balance the trade-off among energy cost, transmit latency and application performance. More collaborators lead to better application performance for the following reasons. First of all, after offloading part of its computation task to other devices, a single device incurs less energy cost. Moreover, since the collaborators share a common object to achieve a positive outcome, the performance can be enhanced if the execution sequence of the application can be reordered so as to increase the level of parallelism. The total time required to finish the whole task could also be cut down. However, involving more nearby devices as potential collaborators will inevitably cause longer delay, since it takes extra time for all the devices to connect to each other.

As shown in Algorithm 1, our goal is to keep the quantity of collaborators below a certain threshold t. Thus each time after we rank the potential collaborators, we will only keep the t mobile devices that have a higher reliability level as the final collaborators.

4.4 Working schemes

There are 2 different kinds of work schemes in the *Circa* framework.

One server and multiple clients The server does not execute any task on its own. In this scheme, a single server device broadcasts its offloading requests to other mobile devices that are within the same region, e.g. such as processing the speech recognition task run by the smartwatch mentioned in Sect. 3. The offloaded task is then divided into multiple smaller subtasks, each of which is transmitted to one client device. The workload of each subtask is similar to one another to simplify the task allocation. In this case, there is one server and multiple clients. A mobile device acts as either a server or a client.

Multiple servers and multiple clients A node can be both a server and a client. In this scheme, all the mobile devices share the same goal to complete one computation-intensive task. Each of the devices provides its own original resource and takes its own responsibility to accomplish the given part of the computation, e.g., when processing the collaborative 3D modelling task mentioned in Sect. 3. In this case, there are multiple servers and multiple clients. A mobile device serves as both a server and a client at the same time.

5 Task allocation and priority assignment algorithms

5.1 Task allocation algorithms

In most mobile code offloading frameworks, in order to improve the performance of computation-intensive tasks, it is necessary to cut down the total completion time. Noted that the time for each collaborating device to receive its target subtasks may be different, because of varying availability levels and mobility patterns that may affect the Bluetooth signal strength. The execution time of each subtask on different clients may also vary, based on the computation ability of different devices. Similarly, the time for clients to transfer their subtasks back to the server devices also depends on their mobility patterns.

Our target is to reduce the total completion time of the whole task, by allocating the subtasks wisely among the most reliable devices with stronger computation abilities. We consider two different working schemes when designing the task allocation algorithm. One is the model with controllable contact knowledge and the other one is the model with uncontrollable contact knowledge. The task allocation for PNP-blocks in *Serenity* [20] also considers the design of algorithms in the context of several models with different contact knowledge and control channel availability knowledge, aimed at minimizing the job completion time. However, our design mainly focuses on the choosing the most reliable devices as collaborators according to the availability ranking list.

5.1.1 Task allocation with controllable contact knowledge

First of all, we consider a scheme with an ideal connection setting so that a device must stay in its current region until the subtask allocated to it is finished. In other words, the contact of collaborators is controllable, since a subtask is guaranteed to be completed after the allocation. This scenario achieves the best possible performance of the subtask allocation algorithm in our framework.

With a controllable future connection and contact information, we propose a task allocation algorithm that greedily picks out the most reliable clients among a group of potential collaborators of the server, which are chosen from the 1-hop neighbours of the server by Algorithm 1 after each update of the resource graph.

After each update of the potential collaborators of the server, the algorithm iteratively picks out the devices with the highest reliability ranking as the clients to receive the subtasks from the server. For each client chosen, the algorithm first examines whether the client is carrying a unfinished subtask with it. If yes, the execution of the subtask will be continued on the device. If not, a new subtask will be assigned to the device. Since the ideal network connection setting assumes that the clients will not disconnect with the server before the server receives the computation result of the subtask from the client, there will not be no subtasks abandoned during the offloading process. Once a certain subtask is delivered to a client, it is guaranteed to be completed and the result will be transferred back to the server.

5.1.2 Task allocation without controllable contact knowledge

Under the scenario that without any controllable contact, it is impossible to reserve a collaborator in advance for the whole task execution period. Thus we propose another algorithm, which allocates the subtasks in an opportunistic way, to solve this problem.

The general idea of this algorithm is that during the subtask dissemination process, the other intermediate nodes can also execute these subtasks. Since there is no controllable future contact knowledge, a client may leave its current area and disconnect with the server, even though the subtask is not finished. Thus, instead of assigning a subtask to a particular client device and let this client device execute the whole subtask until the computation is finished, in this algorithm, all the devices encountered by the client devices can share the responsibility to complete the subtask. The devices that previously serve as clients of the server but have disconnected with the server also have the chance to opportunistically disseminate the subtasks to other devices.

For example, we assume that in the beginning, the server will only consider the clients that are in its 1-hop neighbourhood as potential collaborators. However, during the dissemination and computation process, some old clients may leave the current region and no longer return the results back to the server. Meanwhile, there may be other devices that walk into the 1-hop neighbourhood of such clients. Actually, those non-working clients can still make their own contribution by transferring their unfinished subtask to the new arrivals that are connected to them. These new arrivals can keep the suspended jobs for them and start working on these jobs if they meet the server and connect to the server in the future. We demonstrate the task allocation design in Algorithm 4.

However, although we assume that all of the devices involved are collaborative and trustworthy, there will still be some extreme cases in this scenario.

Selfish devices Selfish devices refer to those devices that accept the request, but refuse to carry on the computation process or send the result back. Such situations may occur when there is a network disconnection. The clients chosen

by the server may leave their current working transmission regions owing to some emergency or just simply shut down their devices because they run out of battery. Token-based incentive mechanism [32] is proposed as a solution to this problem, which makes use of notional credit to pay off nodes for executing tasks, and can be applied as a precaution against selfish devices.

In the scenario of *Circa*, our precaution against selfish devices is to send the intermittent result to the encountered devices of the leaving clients. Once when a client leaves its server with an unfinished subtask, it acts as a server and looks for other clients in its one-hop neighbourhood that volunteers to take its task. After these clients accept the subtasks from the old collaborators, they execute the remaining parts of the subtask if they meet the original server and pass the computation result to the server.

However, it is still possible that after the departing client delivers the subtask to the other devices it encounters, these devices never meets the original server. Such situations may occur when some devices passively try to learn the computation results from others and refuse to return its own computational result. The drawback is that it will get a peak into the computation tasks and lead to delay of the task completion. In order to address this problem, we set a time tolerance threshold in both the server and these newly-joined clients. For a device serving as server, if the time duration that the server waits for the result of a certain subtask exceeds the time tolerance, the server will resend this subtask to another client. Moreover, for a device serving as server and client at the same time, if the duration between the time that it receives results from other clients and the time that it starts to offload results to other clients exceeds the time tolerance, this device will be excluded from the working list, and the subtasks that have been assigned to this client will be resent to another client. In Algorithm 4, we use time tolerance to control the working sequence of the subtasks.

Malicious devices Malicious devices refer to the devices that distort the result. The data of the computation result transferred back by a malicious device may be incorrect. In a collaborative computing process, such incorrect results may lead to the failure of the whole task. To guarantee the correctness of the whole computation task, reputation-based trust [33], in which devices construct and share their reputation information, can be applied in the collaborating process. In our working scheme, if a server receives an incorrect result from a client, it will record the information of the client and penalize it in the reliability ranking in the next rounds.

Algorithm 2 Suspended Subtask Allocation

```
1: Procedure SUSPENDEDSUBTASKALLOCATION(𝕋) /* 𝕋 is the set of subtasks delivered by the server */ /* when the leaving collaborator carries a unfinished subtask, assign these suspended subtasks to its one-hop neighbours */
2: if 𝕋.leavingClientsList.size()!=0 then
3:     for all l ∈ 𝕋.leavingClientsList do
4:         if l.getSubtask() != NULL then
5:             𝕋.suspendedSubtasksList.add(l.getSubtask())
6:             𝕋.executingSubtasksList.add(l.getSubtask())
7:             ℕ ← ONEHOPNEIGHBOURS(l)
8:             for all n ∈ ℕ do
9:                 if n.getSubtask() == NULL then
10:                    n.assign(𝕋.getCurrentSubtask())
11:                end if
12:            end for
13:        end if
14:    end for
15: end if
16: end Procedure
```

Algorithm 3 Overtimed Subtask Allocation

```
1: Procedure OVERTIMEDSUBTASKALLOCATION(t) /* t is the subtask that are currently carried by the collaborator */
   /* if the result of the subtask carried has already been submitted, drop the subtask */
2: if SubtaskTmp ∈ 𝕋.finishedSubtasksList then
3:     v.assginSubtask(NULL)
4:     return TRUE
5: end if
   /* if the result belongs some previously suspended subtask, send the result to the server */
6: if SubtaskTmp ∈ 𝕋.suspendedSubtasksList then
7:     v.sendSubtaskResult();
8:     𝕋.suspendedSubtasksList.remove(SubtaskTmp)
9:     return TRUE
10: end if
11: end Procedure
```

Algorithm 4 Task Allocation without Controllable Contact

```
1: Procedure UNONTROLLABLECONTACTALLOCATION(𝕋, ℙ, tolenrance)
   /* 𝕋 is the set of subtasks delivered by the server, ℙ is the updated ranked set of potential collaborators of the server, tolenrance is the delay tolerance of unreceived subtask result */
2: 𝕋.stableClientsList ← ℙ
3: 𝕋.stableClientsList.retainAll(𝕋.oldClientsList)
4: 𝕋.leavingClientsList ← 𝕋.oldClientsList
5: 𝕋.leavingClientsList.removeAll(𝕋.stableClientsList)
6: 𝕋.newClientsList ← ℙ
7: 𝕋.newClientsList.removeAll(𝕋.stableClientsList)
8: SUSPENDEDSUBTASKALLOCATION(𝕋)
9: assignLoop:
10: for all v ∈ ℙ do
11:     SubtaskTmp ← v.getSubtask()
12:     if SubtaskTmp != NULL then
13:         if OVERTIMEDSUBTASKALLOCATION(SubtaskTmp) == TRUE then
14:             continue assignLoop
15:         end if
16:         v.executeSubtask();
17:         𝕋.executingSubtasksList.add(SubtaskTmp)
18:         if v.isFinished() then
19:             v.sendSubtaskResult();
20:             𝕋.finishedSubtask.add(v.getSubtask())
21:         end if
22:         continue
23:     else
24:         if 𝕋.finishedSubtask.size() == 𝕋.size() then
25:             break assignLoop
26:         end if
        /* assign the overtime subtask in the waiting list to the collaborator */
27:         for all t′ ∈ 𝕋.suspendedSubtasksList do
28:             if t′.waitingTime > tolerance then
29:                 v.assignSubtask(t′)
30:                 𝕋.suspendedSubtasksList.remove(t′)
31:                 𝕋.executingSubtasksList.add(t′)
32:                 continue assignLoop
33:             end if
34:         end for
35:         v.assignSubtask(𝕋.getNewSubtask())
36:         𝕋.executingSubtasksList.add(v.getSubtask())
37:     end if
38:     𝕋.oldClientsList.clear()
39: end for
40: end Procedure
```

Redundant results There will be situations where a server assigns the same subtask to two or more different clients and receives the results from both, which leads to redundant results. Such cases may happen when a selfish device returns and connects to the server. If this device has already delivered its subtask to another client and both this selfish device and this client successfully offloads their subtasks to the original server, there will be redundant results for this subtask. Similar problems will also occur when the server receives the computation result of a subtask that has already expired, i.e. its waiting time exceeds the time tolerance and has already been sent to other devices for further execution. There will be redundant results if the server also collects the output of the same subtask from the others

A precaution which helps to avoid redundant results is to record the reassigned subtasks and finished subtasks, so as to drop the redundant results. Algorithm 2 and Algorithm 3 are designed as a precaution against redundant results, by analysing and allocating the overtimed subtask and suspended subtasks wisely. If the subtask carried by a client is already in the finished subtask list, it will be automatically dropped by the client.

5.2 Priority scheduling algorithm

As mentioned in the previous subsection, the completion time for the whole task can be reduced by allocating the subtasks wisely to the clients of the server device. Furthermore, although each subtask is executed parallel on the clients, we should still consider the sending sequence of these subtasks. Which subtask should we assign to the device that come first in the reliability ranking list? Which should be assigned to the second one?

It is possible that some computationally intensive task is chronological. Some tasks can only be simultaneously allocated after other tasks are completed. For example, in the speech recognition application, those subtasks with posterior content in time should be executed earlier, since there will be information loss without the earlier content. For such chronological subtasks, those with posterior content should be assigned higher priority.

The task completion time can be reduced by assigning properties to each subtask according to their chronological

order in the original task. In the scenario of *Circa*, for a server device, we assign different priorities to the subtasks of its task. Subtasks with more descendants should have higher priorities.

Algorithm 5 Priority Assignment

```
1: Procedure PRIORITYASSIGNMENT(𝕋)
   /* Step 1: Assign same priority to each subtask */
2: for t in 𝕋 do
3:     t.priority ← 𝕋.size()
4: end for
   /* Step 2: Sort the subtasks according to the chronological order
   of the whole task */
5: for t in 𝕋 do
6:     t.priority ← t.priority − t.ascendantsList.size()
7: end for
   /* Step 3: Sort the subtasks according to the number of their de-
   scendants */
8: for l in 𝕋.priorityList do
9:     for t′ in l do
10:        tmp ← l.descendantsList.getMax()-
           t′.descendantsList.size()
11:        t′.priority ← t′.priority -(tmp)
12:    end for
13: end for
   /* Step 4: Randomly assign different priorities to those that are
   with same priority */
14: for t in 𝕋 do
15:    for t′ in 𝕋 do
16:        if t.priority == t′.priority then
17:            t′.priority ← t′.priority −1
18:        end if
19:    end for
20: end for
21: end Procedure
```

The details of the priority assignment algorithm is shown in Algorithm 5. First of all, we assign the same priority to each subtask. Afterwards, we sort the subtasks according to the chronological order of the whole task. Then the algorithm will sort the subtasks according to the number of their children. In the last step, we randomly assign different priorities to those that have the same priority. A subtask will not be sent to a client for computing and processing until all the other subtasks with higher priorities have been sent.

6 Performance evaluation

6.1 Simulation setup

To evaluate the performance of the task allocation algorithms of our framework, we have carried out a simulation based on different mobility models. Under each mobility model, we configure the experimental settings such as the number of nodes, node properties and speed, all of which emulate the intermittent connectivity among nodes.

We use three mobility models to synthesize the connections among the mobile devices within a simulation area, including the Random Waypoint Model (RWP) [34] , the Gauss–Morkov Model (GM) [35] and the Reference Point Group Model (RPG) [36].

The simulation area is 100 m $\times$ 100 m. There are 30 nodes in each simulation area, representing 30 mobile devices. We use distance threshold, instead of contact trace, to determine the contact relationship of the nodes, since any two nodes can establish a connection with each other if they enter the transmission region of the same beacon. During the simulation process, if two devices set up a connection with each other with the assistance of iBeacons, there will be a link between the two nodes that represent these two devices in the figure, indicating that the two nodes are in contact with each other and are able to exchange data and subtasks. The contact range is set to 15 m here, since the efficient transmission region of a beacon is usually 15–20 m.

We have not adopted any contact traces in our mobility emulation modules, since with the assistance of iBeacon, any two devices within a certain beacon transmission region are capable of setting up a connection with each other.

In order to demonstrate how the *Circa* framework assists the collaborative task offloading and computation of the mobile device involved, we primarily compare the performance of executing the tasks locally on the device that act as a server with that of executing the divided subtasks on *Circa*. Every experiment is repeated 10 times to acquire the average values of the output results.

All the experiments presented are conducted on a MacBook Pro with 2.9 GHz Intel Core i5 processor and 8 GB 2133 MHz LPDDR3 memory. The simulation program is implemented using Python 2.7.

6.2 Performance evaluation

We compare the performance of the algorithms in this section. As mentioned in Sect. 4, the task of the server device is divided into multiple subtasks by a pre-process program. Then the subtasks will be distributed to the client devices for further processing. Finally, each client device will offload its execution result to the post-process program of the server device after the computation of its subtask is finished. Then the post-process program of the server device then assembles all the execution results it received and comes out with the final complete result. In each of our experiments, we randomly pick out one node in all the involved nodes in the simulation as the server device, then carry out the task allocation algorithms and the priority scheduling algorithms on this node. One important

assumption of our simulation is that all of the devices involved have the same objective, which is to complete the task.

Figures 3, 4 and 5 demonstrate the performance benefits of the two task allocation algorithms in *Circa*, compared to local execution. We examine the completion time of different numbers of subtasks which are offloaded to the clients for execution under different scenarios.

Figure 3 shows the simulation results of the Random Waypoint Model (RWP), while Fig. 4 demonstrates the results of the Gauss–Morkov Model (GM) and Fig. 5 examines the results of the Reference Point Group Model (RPG).

As shown in the figures, in all three mobility models, when the number of subtasks increases, the performance benefit of the task allocation algorithms become more significant. With a higher workload, *Circa* is able to cut down the total execution time of the input task by distributing the task among the nearby devices as collaborators wisely. Moreover, we can observe that the Predictable Allocation Algorithm consistently performs slightly better than the Unpredictable Allocation Algorithm, which indicates that the *Circa* framework achieves a better performance with the control channel and future contact information. Lastly, there are small difference between the simulation results of different mobility models. The experimental results of the Reference Point Group Model is the best, since in RPG, there are always groups of users travelling together, which implies that it is quite convenient for a server device to seek available and reliable collaborators, while the probability of connection failure is also quite small.

In order to further analyze the impact of different environments on the performance of the two task allocation algorithms, we also compare the completion time of the whole task with varying moving speed of the devices as well as the density of the devices. According to the experimental result shown in Fig. 6, when the speed increases from 0 to 5 m/s, the task completion time is cut down. Since the devices are moving freely in the fixed area, the probability that each device encounters available collaborators will become higher. However, when the speed increases from 5 to 10 m/s, the total execution time for the whole task does not decrease largely.

The next experiment demonstrates the impact of node density on the performance of the two task allocation algorithms. Figure 7 shows the comparison of the task completion time with different numbers of devices in a fixed area. Based on the observation of the experimental results, we can conclude that the task completion time will be shorter with increased device density. We can also observe that when the number of devices is larger than 40, the task completion time will remain constant even if the number of devices is increased.

We have also conducted another experiment to examine the impact brought by the different thresholds of the quantity of collaborators as well as the time delay tolerance. As demonstrated in Table 1, the total completion time of the whole task is rather long when the threshold of the number of collaborators is small. The task completion time decreases when the threshold is set to be larger, since the parallelism level of the whole task is increased, providing each of the subtasks the opportunity to run on different devices and save time.

However, if the threshold is larger than 6, the task completion time increases, since the probability of disconnection during the offloading and remote computation process is increased when more collaborators are involved. It leads to extra time for reconnection and transmission delay, hence the task completion time is proved to be longer with too many collaborators.

Meanwhile, the task completion time is also affected by the time delay tolerance, but not to a very large extent. It is also a little bit long when the time tolerance is set to be very short. It slightly decreases when the time delay tolerance is set to be longer. Nevertheless, when the time delay tolerance is set to be too long, i.e., longer than 40 s (more than two times longer than the subtask task completion time), it takes more time to finish the whole task. According to the experimental results, we should choose the proper value of the threshold of collaborator numbers as well as the time delay tolerance to improve the performance of the collaborative offloading.

The last experiment aims at exploring the importance of assigning priorities to the client devices. We adapt the subtask priority scheduling in Fig. 8 as a scheduling model of the experiment. Among the 100 subtasks that needed to

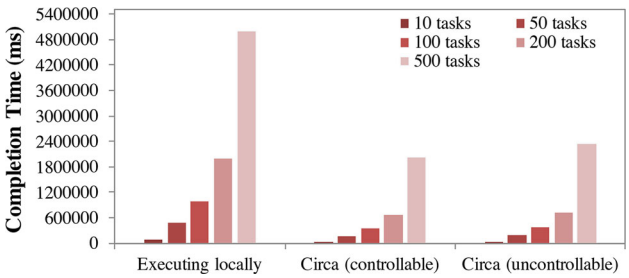


Fig. 3 A comparison of the performance benefits of *Circa*, with different amounts of subtask input. We use the Random Waypoint Model (RWP) here as the mobility model. The minimum velocity of each device is 0.05 m/s, the maximum velocity of each device is 0.2 m/s. The maximum waiting time is 300 s. We configure the simulation area to be 100 m $\times$ 100 m. There are 30 devices in the simulation area. The contact range is 15 m. The number of subtasks is 100. The subtask completion time is 15 s. The threshold of the quantity of collaborators is 5. The delay tolerance is 25 s

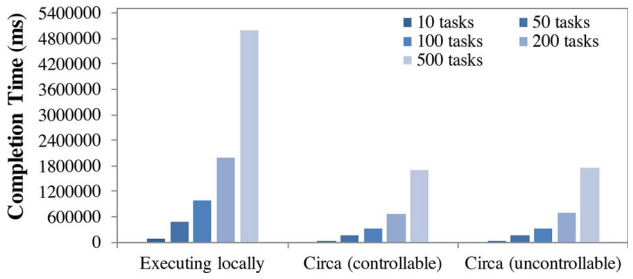


Fig. 4 A comparison of the performance benefits of *Circa*, with different amounts of subtasks input. We use the Gauss–Markov Model (GM) here as the mobility model. The mean velocity is 0.1 m/s. The tuning parameter used to vary the randomness, α, is 0.5, while the randomness variance is 0.1. We configure the simulation area to be 100 m $\times$ 100 m. There are 30 devices in the simulation area. The contact range is 15 m. The number of subtasks is 100. The subtask completion time is 15 s. The threshold of the quantity of collaborators is 5. The delay tolerance is 25 s

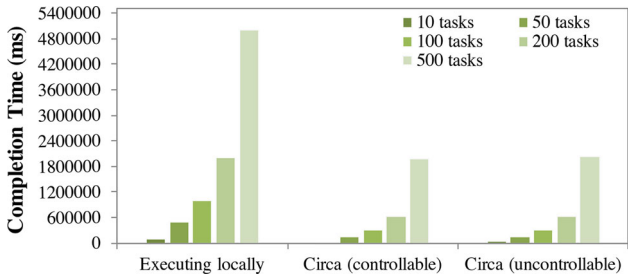


Fig. 5 A comparison of the performance benefits of *Circa*, with different amounts of subtasks input. We use the Reference Point Group Model (RPG) here as the mobility model. The minimum velocity of each device is 0.05 m/s, the maximum velocity of each device is 0.2 m/s.The value of aggregation is set to be 0.3. We configure the simulation area to be 100 m $\times$ 100 m. There are 30 devices in the simulation area. The contact range is 15 m. The number of subtasks is 100. The subtask completion time is 15 s. The threshold of the quantity of collaborators is 5. The delay tolerance is 25 s

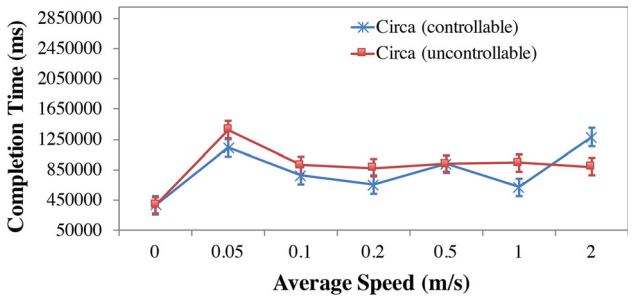


Fig. 6 The influence brought about by the device speed of *Circa*.We use the Gauss–Morkov Model (GM) here as the mobility model. We configure the simulation area to be 100 m $\times$ 100 m. There are 30 devices in the simulation area. The contact range is 15 m. The number of subtasks is 100. The subtask completion time is 15 s. The threshold of the quantity of collaborators is 5. The delay tolerance is 25 s

be processed, 20 of them belong to *A*; 10 of them belong to *B*; *C* has 20 subtasks; *D* has 20 subtasks; *E* has 30 subtasks and *F* has 0 subtask. Without the priority scheduling algorithm, all the subtasks are randomly distributed among the unordered collaborators. The priority scheduling algorithm, on the other hand, assigns higher priorities to *A*, *B*, *C* and *D*, while lower priorities are assigned to *E* and *F*. Those subtasks with higher priorities will be disseminated and processed earlier.

Figure 7 shows the completion time of the whole task with and without applying the priority scheduling algorithm before distributing the subtasks. According to the experimental results, the priority scheduling algorithm greatly improves the performance of the collaborative offloading in the *Circa* framework.

7 Implementation

7.1 Experiment setup

In this section, we evaluate the performance of *Circa* to validate its feasibility and efficacy. We first describe the hardware configurations, test applications and performance metrics used in our experiments. We then present and analyze the detailed experimental results under different scenarios.

Our prototype consists of three smart devices: an iPhone 5, an iPhone 4s and an iPad 3. The detailed information of the configurations for the tested devices are listed in Table 2.

We deploy 3 beacons bought from Estimote Inc. in our experiment environment, acting as the iBeacon transmitters in *Circa*. Each beacon has a square chip inside, which is s 32-bit ARM Cortex M0 CPU with 256 KB flash with a 2.4 GHz Bluetooth low-energy radio [29]. During the experiments, all of the beacons are attached to a plain wall in order to avoid possible signal distortion.

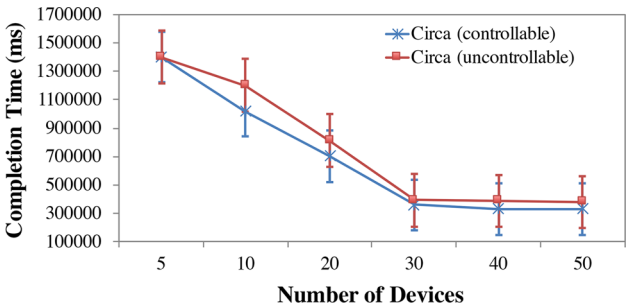


Fig. 7 The influence brought by device density of *Circa*.We use the Random Waypoint Model (RWP) here as the mobility model. We configure the simulation area to be 100 m $\times$ 100 m. The contact range is 15 m. The minimum velocity of each device is 0.05 m/s, the maximum velocity of each device is 0.2 m/s. The number of subtasks is 100. The subtask completion time is 15 s. The threshold of the quantity of collaborators is 5. The delay tolerance is 25 s

Table 1 A comparison of task completion time (measured in ms) with different thresholds of collaborator numbers and delay time tolerance

Delay tolerence (s)	Threhhold of the quality collaboration							
	2	3	4	5	6	7	8	9
15	820	548	413	357	352	372	372	388
20	805	546	411	362	352	370	372	390
25	817	543	407	359	345	372	385	390
30	819	546	403	359	351	372	390	405
35	805	545	416	367	362	376	387	413
40	810	543	405	361	359	376	390	414
45	805	546	409	355	358	390	405	425

We use the Gauss–Morkov Model (GM) here as the mobility model. We configure the simulation area to be 100 m × 100 m. There are 30 devices in the simulation area. The contact range is 15 m. The minimum velocity of each device is 0.05 m/s, the maximum velocity of each device is 0.2 m/s. The number of subtasks is 100. The subtask completion time is set to be 15 s

To test the practical performance of *Circa*, we develop a speech recognition application, which records a wav file of the speaker, then transmit the file to Google Speech API to obtain the corresponding text result. The application divides the wav file into several smaller files according to the quantity of selected collaborators.

We examine *Circa* under different scenarios in our experiment. In the first scenario "single device", the original device carries out the speech recognition task by itself. In the second scenario "2 collaborators" (1 server and 1 client), the original device switches to airplane mode and only turns on the Bluetooth for connection, which means that its WiFi and cellular network are both cut off. This original device then acts as a server. There is another device that is willing to act as a client to accept the task from the server. In other words, there are 2 collaborators, one serves as a server and the other serves as a client. In the third scenario "3 collaborators" (1 server and 2 clients), the original device still serves as a server, while there are two other devices acting as clients to accept the task offloaded from the server. The wav file from the server will be divided into two files with identical amounts of data, which will be offloaded to the 2 clients for speech recognition. In this situation, there are 3 collaborators in total.

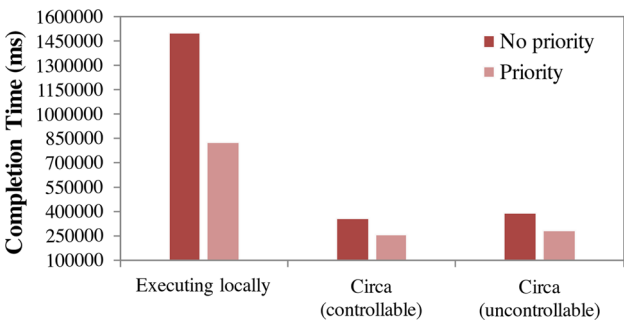


Fig. 8 The influence of priority assignment of *Circa*.We use the Random Waypoint Model (RWP) here as the mobility model. We configure the simulation area to be 100 m × 100 m. The number of nodes is 30. The contact range is 15 m. The minimum velocity of each device is 0.05 m/s, the maximum velocity of each device is 0.2 m/s. The number of subtasks is 100. The subtask completion time is 15 s. The threshold of the quantity of collaborators is 5. The delay tolerance is 25 s

In all the experiments, the iPhone 5 acts as the only server, while the iPhone 4s and iPad 3 act as potential clients. We measure the time duration of a whole task as an indicator of its performance, since the speech recognition task is time-sensitive. To quantify the delay incurred by offloading, we measure the time duration of a given task under different scenarios with varying task sizes, different numbers of obstacles, distances between devices and speed of devices. All of our measurements are performed under stable network conditions, with all the mobile devices running in standalone environments, in which all other applications and background tasks are shut off, with the screen on.

7.2 Performance analysis

The first experiment is conducted to compare the total execution time of the whole task with single device, 2 collaborators (1 server and 1 client), and 3 collaborators (1 server and 2 clients). As shown in Fig. 9, it takes a longer time to complete larger tasks. Compared to the original scheme with only one device, if there are 2 collaborators, there will be additional connection time, which is 5000 ms on average.

However, if more than two devices are involved in the task, the total execution time for the same task will be shorter than the local execution. The benefit of offloading becomes more significant when the task is larger. The main reason for this phenomenon is the degree of parallelism. With 3 collaborators, we can divide the task into two smaller pieces so that they can be processed at the same time, while the transmission delay is also around 5000 ms, remaining the same as that of 2 collaborators.

In the second experiment, of which the results are shown in Fig. 10, we measure the execution time of the task under

Table 2 Configurations of mobile devices tested

Devices	iPhone 5	iPhone 4s	iPad 3
Version	iOS 7.1.2	iOS 7.1.2	iOS 7.1.2
Capacity	27.9 GB	28.3 GB	27.8 GB
Avaliable	9.6 GB	21.8 GB	4.2 GB
Battery	15–20%	45–55%	90–100%

different environments with obstacles. The reason for conducting such experiments is that the Bluetooth signal could be easily affected by physical surroundings due to distraction or absorption. We use 1, 2, 3 and 4 cardboards as obstacles to block the signal from the beacon in different directions. The devices stand in a line, pointing in the same direction to the beacon. According to the result, only when we use 4 cardboards to block all 4 directions around the beacon, will the total execution time increase significantly due to minor connection latency.

The third experiment examines the impact of the distances between devices during task execution. The distances shown in Fig. 11 refer to the vertical distances between every two devices. Based on the experimental results, we can conclude that the distances between the devices do not affect the offloading performance if the devices all stay in the broadcasting region of the same beacon. However, there will be additional connection latencies if the devices are too close to each other (less than 1m).

The fourth experiment compares the total execution time with varying device speeds during the offloading process. According to experimental data shown in Fig. 12, the speeds of the devices do not affect the processing time greatly, as long as the devices stay in the Bluetooth transmission region of each other.

Finally, as illustrated in Fig. 13, we have also evaluated the power consumption of each device during the offloading. We utilize the Energy Diagnostics in Open Developer Tool for iOS developers to measure the power consumption of the speech recognition application. For a certain size of task, we repeat the experiment for 15 times and use the average power consumption of each device as the final result. The experimental result indicates that with more collaborators, the average power consumption of each device will decrease. More power will be saved with more collaborators when the task becomes larger. The reason is that with more collaborators, the computational effort for the subtask conducted by each device is smaller, which leads to less power consumption.

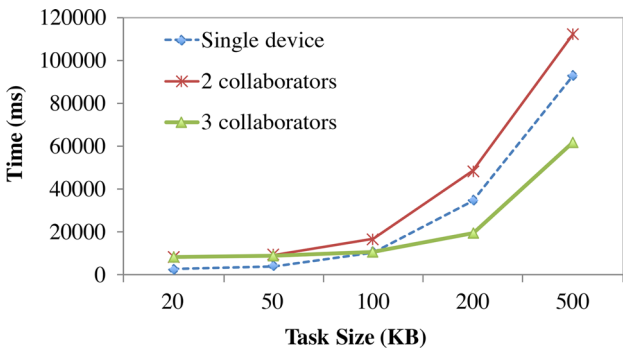


Fig. 9 The total execution time with different task sizes. Distance = 1 m. Speed = 0 m/s. No obstacle

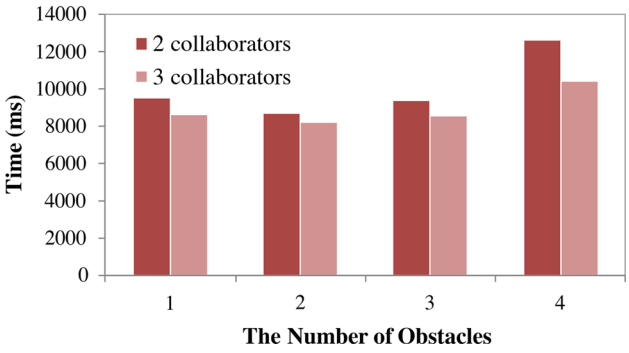


Fig. 10 The total execution time in different environment. Task size = 50 KB. Distance = 1 m. Speed = 0 m/s

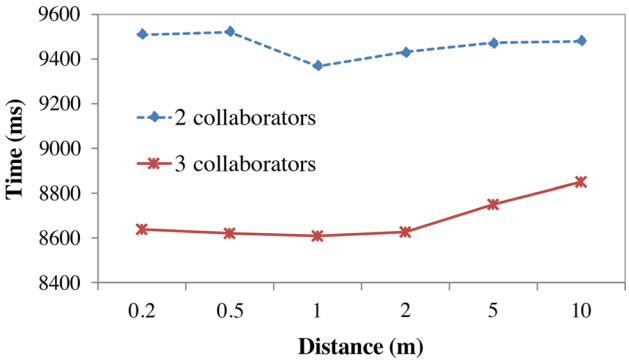


Fig. 11 The total execution time with varying distances between the devices. Task size = 50 KB. Speed = 0 m/s. No obstacle

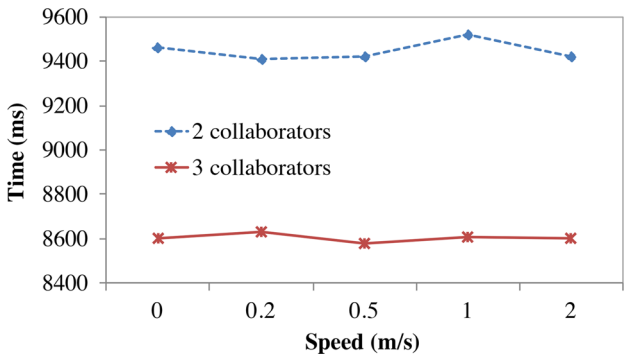


Fig. 12 The total execution time with varying device speeds. Task size = 50 KB. Distance = 1 m. No obstacle

We have also tested the situation where collaborators leave the region and the offloading process ends without prompt. When there are only two collaborators (1 server and 1 client) and the client leaves, the collaborative process just ends and the original device will be notified about the interruption. It can choose either to conduct the task by itself or to search for another possible collaborator. When there are three collaborators (1 server and 2 clients), if the server loses one of the clients, it simply offloads the unfinished task to another client.

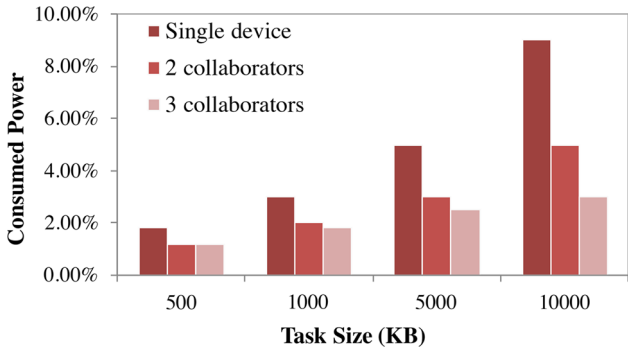


Fig. 13 The average power consumption of each device during the offloading, with different size of task. Distance = 1 m. No obstacle

8 Concluding remarks and future work

8.1 Conclusions

In this paper, we have designed and implemented *Circa*, a collaborative code offloading framework, leveraging the presence of iBeacon. Unlike the involvement of cloud servers in existing schemes, *Circa* is completely localized and requires no Internet connection. Devices within a local area can discover and help each other with the assistance of iBeacons as long as their Bluetooth functions work properly.

Afterwards, a reliability analysis is conducted to select the mobile devices with sufficient computation resources and collaborative incentive to be collaborators for the current task. We also proposed two task allocation algorithms and one task priority scheduling algorithm to achieve the goal of minimizing local power consumption as well as cutting down the total completion time of the entire task.

We evaluate the performance benefits of this system with different mobility models. As a proof of concept, we have implemented a prototype to evaluate the feasibility and performance of *Circa*. Experimental results with realistic settings have shown that code offloading incurs negligible (if any) delay with two collaborators. Fortunately, with more than two collaborators, the performance is improved since total execution time is reduced, when compared to local execution of the same code.

8.2 Future work

As for further discussion, in order to improve the performance of the *Circa* framework, it is possible to involve a cloud server which connects to each device via WiFi or cellular network. The cloud server will be able to monitor the devices more efficiently and record the beacons in their vicinity. With a cloud server, the initiator devices (servers) no longer needs to spend extra battery or memory on the reliability analysis of their potential collaborators (clients). However, the benefits might offset the latency and energy consumption caused by long-range network connection to the cloud.

Another possible extension to *Circa* lies in the division of an offloaded task. Currently, the task is divided evenly into multiple subtasks for dissemination. Nevertheless, it would be more reasonable if the size of each divided part depends on the battery level and availability state of its targeted device. We leave a more sophisticated division algorithm for future work.

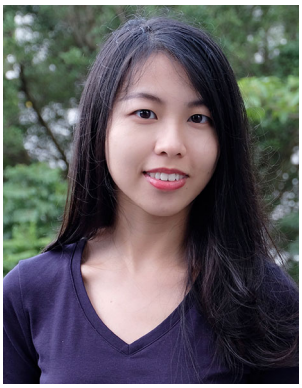
Furthermore, we will also complete our experimental evaluation of the prototype system by involving more devices and more challenging scenarios. We will also conduct more performance analysis in terms of execution time, energy, memory and processor consumption.

Acknowledgements This work is supported in part by RGC GRF Grants under the contracts 16211715 and 16206417, and an RGC CRF Grant under the contract C7036-15G.

References

1. Cuervo, E., Balasubramanian, A., Cho, D.-K., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010). Maui: Making smartphones last longer with code offload. In *Proceedings of ACM MobiSys* (pp. 49–62).
2. Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011). Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of ACM EuroSys* (pp. 301–314).
3. Kosta, S., Aucinas, A., Hui, P., Mortier, R., & Zhang, X. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proceedings of IEEE INFOCOM* (pp. 945–953).
4. iBeacons for developers. https://developer.apple.com/ibeacon/. Accessed 4 Sept 2018.
5. Gordon, M. S., Jamshidi, D. A., Mahlke, S. A., Mao, Z. M., & Chen, X. (2012). Comet: Code offload by migrating execution transparently. In *Proceedings of USENIX OSDI* (pp. 93–106).
6. Satyanarayanan, M., Bahl, P., Caceres, R., & Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE Transactions on Pervasive Computing*, *8*(4), 14–23.
7. Kemp, R., Palmer, N., Kielmann, T., & Bal, H. (2012). Cuckoo: A computation offloading framework for smartphones. In *Mobile computing, applications, and services* (pp. 59–79). Springer
8. Kao, Y.-H., Krishnamachari, B., Ra, M.-R., & Bai, F. (2017). Hermes: Latency optimal task assignment for resource-constrained mobile computing. *IEEE Transactions on Mobile Computing*, *16*, 3056–3069.
9. Chen, X., Jiao, L., Li, W., & Xiaoming, F. (2016). Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, *24*(5), 2795–2808.
10. Zhou, B., Dastjerdi, A. V., Calheiros, R. N., Srirama, S. N., & Buyya, R. (2015). A context sensitive offloading scheme for mobile cloud computing service. In *2015 IEEE 8th international conference on cloud computing (CLOUD)* (pp. 869–876). IEEE.
11. Flores, H., Hui, P., Nurmi, P., Lagerspetz, E., Tarkoma, S., Manner, J., et al. (2017). Evidence-aware mobile computational

offloading. *IEEE Transactions on Mobile Computing*, *17*, 1834–1850.
12. Doolan, D. C., Tabirca, S., Yang, L. T. (2008). MMPI: A message passing interface for the mobile environment. In *Proceedings of ACM conference on advances in mobile computing and multimedia* (pp. 317–321).
13. Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop distributed file system. In *Proceedings of IEEE symposium on mass storage systems and technologies (MSST)* (pp. 1–10).
14. Marinelli, E. E. (2009). Hyrax: Cloud computing on mobile devices using mapreduce. Technical report, DTIC Document.
15. Black, M., & Edgar, W. (2009). Exploring mobile devices as grid resources: Using an ×86 virtual machine to run boinc on an iphone. In *2009 10th IEEE/ACM international conference on grid computing* (pp. 9–16). IEEE.
16. Anderson, D. P. (2004). Boinc: A system for public-resource computing and storage. In *Fifth IEEE/ACM international workshop on grid computing, 2004. Proceedings* (pp. 4–10). IEEE.
17. Thain, D., Tannenbaum, T., & Livny, M. (2005). Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, *17*(2–4), 323–356.
18. Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., & Werthimer, D. (2002). Seti@ home: An experiment in public-resource computing. *Communications of the ACM*, *45*(11), 56–61.
19. Beberg, A. L., Ensign, D. L., Jayachandran, G., Khaliq, S., & Pande, V. S. (2009). Folding@ home: Lessons from eight years of volunteer distributed computing. In *IEEE international symposium on parallel and distributed processing, 2009. IPDPS 2009* (pp. 1–8). IEEE.
20. Shi, C., Lakafosis, V., Ammar, M. H., & Zegura, E. W. (2012). Serendipity: Enabling remote computing among intermittently connected mobile devices. In *Proceedings of the thirteenth ACM international symposium on mobile ad hoc networking and computing* (pp. 145–154). ACM.
21. Langford, T., Gu, Q., Rivera-Longoria, A., & Guirguis, M. (2013). Collaborative computing on-demand: Harnessing mobile devices in executing on-the-fly jobs. In *2013 IEEE 10th international conference on mobile ad-hoc and sensor systems (MASS)* (pp. 342–350). IEEE
22. Miluzzo, E., Cáceres, R., & Chen, Y.-F. (2012). Vision: mclouds-computing on clouds of mobile devices. In *Proceedings of the third ACM workshop on mobile cloud computing and services* (pp. 9–14). ACM.
23. Guirguis, M., Ogden, R., Song, Z., Thapa, S., & Gu, Q. (2011). Can you help me run these code segments on your mobile device? In *2011 IEEE global telecommunications conference (GLOBECOM 2011)* (pp. 1–5). IEEE.
24. Arslan, M. Y., Singh, I., Singh, S., Madhyastha, H. V., Sundaresan, K., & Krishnamurthy, S. V. (2012). Computing while charging: Building a distributed computing infrastructure using smartphones. In *Proceedings of the 8th international conference on emerging networking experiments and technologies* (pp. 193–204). ACM.
25. Sucipto, K., Chatzopoulos, D., Kosta, S., & Hui, P. (2017). Keep your nice friends close, but your rich friends closercomputation offloading using nfc. In *IEEE conference on computer communications INFOCOM 2017, IEEE* (pp. 1–9). IEEE.
26. Chen, X., & Zhang, J. (2017). When d2d meets cloud: Hybrid mobile task offloadings in fog computing. In *2017 IEEE international conference on communications (ICC)* (pp. 1–6). IEEE.
27. Cheng, Z., Li, P., Wang, J., & Guo, S. (2015). Just-in-time code offloading for wearable computing. *IEEE Transactions on Emerging Topics in Computing*, *3*(1), 74–83.
28. Lin, X., Jiang, J., Li, B., & Li, B. (2015). Circa: Offloading collaboratively in the same vicinity with ibeacons. In *2015 IEEE international conference on communications (ICC)* (pp. 3751–3756). IEEE.
29. Estimote Beacon API. https://developer.estimote.com. Accessed 4 Sept 2018.
30. Horovitz, A., Kim, K., LaMarche, J., & Mark, D. (2013). Peer-to-peer over bluetooth using game kit. In *More iOS6 development* (pp. 251–293). Springer.
31. Game Kit Programming Guide. https://developer.apple.com/documentation/gamekit. Accessed 4 Sept 2018.
32. Lu, R., Lin, X., Zhu, H., Shen, X. S., & Preiss, B. (2010). Pi: A practical incentive protocol for delay tolerant networks. *IEEE Transactions on Wireless Communications*, *9*(4), 1483–1493.
33. Buttyán, L., & Hubaux, J.-P. (2000). Enforcing service availability in mobile ad-hoc wans. In *Proceedings of the 1st ACM international symposium on mobile ad hoc networking and computing* (pp. 87–96). IEEE Press.
34. Saha, A. K., & Johnson, D. B. (2004). Modeling mobility for vehicular ad-hoc networks. In *Proceedings of the 1st ACM international workshop on vehicular ad hoc networks* (pp. 91–92). ACM.
35. Camp, T., Boleng, J., & Davies, V. (2002). A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing*, *2*(5), 483–502.
36. Hong, X., Gerla, M., Pei, G., & Chiang, C.-C. (1999). A group mobility model for ad hoc wireless networks. In *Proceedings of the 2nd ACM international workshop on modeling, analysis and simulation of wireless and mobile systems* (pp. 53–60). ACM.



Xueling Lin is currently a Ph.D. student at the Department of Computer Science and Engineering in Hong Kong University of Science and Technology. Before starting her Ph.D. journey in January 2017, she obtained her MPhil degree in Computer Science and Engineering from Hong Kong University of Science and Technology and received her Bachelor degree in Software Engineering from Sun Yat-sen University. Her current research interests include knowledge base refinement, data fusion, truth discovery, cloud computing and mobile computing.

Jingjie Jiang received her B.Eng. degree from the Department of Automation, Tsinghua University, China, in 2012, and the Ph.D. degree from the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology in 2017. She visited the Department of Electrical and Computer Engineering at the University of Toronto in 2015. She is now with Future Network Theory Lab, Huawei Technology. Her current research interests include datacenter network management, congestion control, big data scheduling, device-to-device communication and content distribution.

Calvin Hong Yi Li is currently an undergraduate student at Johns Hopkins University in Baltimore, Maryland, completing his B.S./B.A. degrees in Computer Science, Cognitive Science and Applied Math & Statistics. His research interests include cloud computing, mobile computing, deep learning and natural language processing.

Bo Li is a Chair professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, which he has been affiliated with since 1996. His works covered a wide spectrum of topics in computer networking and communications, more recently on datacenter networking, cloud computing, big data analytics, machine learning in cloud, content distribution in the Internet, and network control algorithms. He made pioneering contributions in the Internet video broadcast with a system called Coolstreaming, which was credited as the first large-scale Peer-to-Peer live video streaming system in the world. This work received the inaugural The Test-of-Time Paper Award from IEEE INFOCOM (2015). He has been an editor or a guest editor for over a two dozen journals and magazines, mostly in IEEE and ACM. He was the Co-TPC Chair for IEEE INFOCOM 2004. He received six Best Paper Awards from IEEE. He received the Young Investigator Award from Natural Science Foundation of China (NFSC) in 2005, the State Natural Science Award (2nd Class) in 2011. He received his Ph.D. in the Electrical and Computer Engineering from the University of Massachusetts at Amherst, and his B.Eng. (summa cum laude) in the Computer Science from Tsinghua University, Beijing, China.

Baochun Li received his B.Eng. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995 and his M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Professor. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include cloud computing, large-scale data processing, datacenter networking, and coding for distributed storage systems. Dr. Li has co-authored more than 350 research papers, with a total of over 16,000 citations, an H-index of 74 and an i10-index of 225, according to Google Scholar Citations. He was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award. He is a member of ACM and a Fellow of IEEE.