Sense-Aid: A Framework for Enabling Network as a Service for Participatory Sensing

Anonymous Author(s)

Abstract

The rapid adoption of smartphones with different types of advanced sensors has led to an increasing trend in the usage of mobile crowdsensing applications, *e.g.*, to create hyperlocal weather maps. However, the high energy consumption of crowdsensing, chiefly due to expensive network communication, has been found to be detrimental to the wide-spread adoption. We propose a framework, called Sense-Aid, that can provide energy-efficient mobile crowdsensing service, coexisting with the cellular network. There are two key innovations in Sense-Aid beyond prior work (Piggyback Crowdsensing-Sensys13)—the middleware running on the cellular network edge to orchestrate multiple devices present in geographical proximity to suppress redundant data collection and communication. It understands the state of each device (radio state, battery state, etc.) to decide which ones should be selected for crowdsensing activities at any point in time. It also provides a simple programming abstraction to help with the development of crowdsensing applications. We show the benefit of Sense-Aid by conducting a user study consisting of 60 students in our campus, compared to a baseline periodic data collection method and Piggyback Crowdsensing. We find that energy saving is 93.3% for Sense-Aid compared with Piggyback Crowdsensing in a representative case which requires 2 devices to provide barometric values within an area of a circle whose radius is 1 kilometer and requires periodic data collection every 5 minutes for a 90-minute test. The selection algorithm of Sense-Aid also ensures reasonable fairness in the use of the different devices.

1 Introduction

Recent additions of advanced and highly sophisticated sensors to smartphones have led to an emerging category of devices which stand to push forward the *Internet of Things* (IoT) era. One area of IOT which has effectively made use of the sensing, computing, and networking capability of these mobile devices is *participatory sensing*. Participatory sensing, originally introduced in 2006 by Srivastava *et al.* [7], refers to tasks deployed on mobile devices to form interactive, participatory sensor networks that enable public and professional users to gather, analyze, and share local knowledge. There are already a large number of participatory sensing applications some of which are widely used. Examples include WeatherSignal [17], Pressurenet [11], which has joined Sunshine [2], air quality monitoring [10, 14, 21], noise pollution map [23], urban planning [6, 12, 16], road and traffic condition reporting [5, 18]. The possibilities for such applications are endless, limited only by our imagination and current technology. It turns out to be hundreds of thousands of downloads, e.g., about 100,000 downloads for Sunshine from Google Play store. However, current crowdsensing applications cause significant energy drain (as we show empirically later in this section) and this becomes a deterrent to participation.

Middleware'17,

DOI: 10.1145/nnnnnnn.nnnnnnn

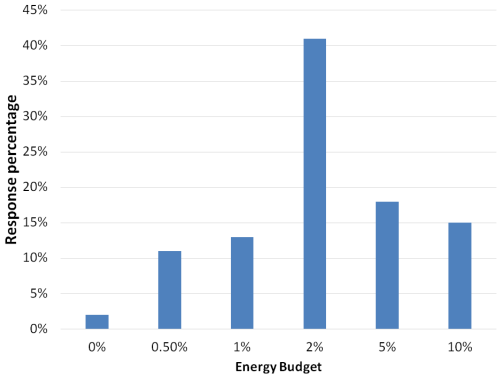

Figure 1. Energy usage expectations from 109 university students. Majority of the users are willing to spend up to 2% of energy in participatory crowdsensing activities.

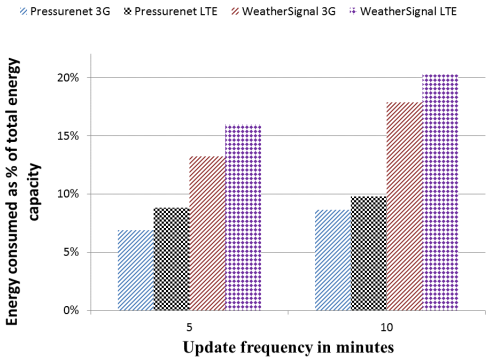

Figure 2. Power consumption case study for two popular weather-related crowdsensing apps. In both apps, the energy budget that the majority of our surveyed users is willing to spend on crowdsensing is exceeded. The duration of the experiment is 4 hours for the 5 minute measurement frequency and 8 hours for the 10 minute measurement frequency.

We did a survey that asked 109 people, from different universities in U.S., China, and India: *"At what battery cost level are you willing to take part in participatory sensing applications?"* Their responses are presented in Figure 1. We find that 41.4% of the people are willing to spend up to 2% of their phone's battery capacity for crowdsensing activities. None are willing to spend over 10% of their battery life.

To show that current popular crowdsensing applications cost more than this tolerable energy, we studied the power consumption of two popular crowdsensing applications, in which we can change the frequency of data uploads. Pressurenet measures barometric pressure and creates a pressure map and forecasts weather events like tornadoes. WeatherSignal [17] collects a wider variety of weather signals and magnetic field and overlays it on a map. We ran our tests on a Samsung Galaxy S4 smartphone. For each application, we varied the update frequency, while shutting down

all other applications. The results are shown in Figure 2. We can see that in all cases the energy consumption is more than what the majority of the users in our survey would want to pay for crowdsensing activities (2% of the battery capacity). Secondly, LTE energy consumption is higher than 3G and WeatherSignal is more energy hogging than Pressurenet, due no doubt to the richer data that it collects. A third observation is that even for light weight applications like Pressurenet that only uses barometer on the smartphone and does no other activities, in 4G LTE network, it will cost significant amount of energy (close to 10% in two experiments).

These observations lead us to the hunch that the number of people willing to be a part of crowdsensing activities would go up if the sensing applications were made more energy efficient.

How can crowdsensing be made resource conserving?

Looking deeply into the sources of high energy drain for crowdsensing, we find there are *3 primary opportunities* for resource-optimized crowdsensing. First, the network communication of the crowdsensing data from the device to the network can be optimized, such as, by piggybacking the crowdsensing data opportunistically on regular network communication from the device. This is beneficial because as is well shown from prior literature (*e.g.*, see Lagar-Cavilla *et al.* [19] and Huang *et al.* [15]), it costs far less energy to send a few extra bytes along with a current packet, than to create a whole new packet. Second, crowdsensing data can be collected and uploaded depending on the state of the device. For example, the typical 4G radio stack has the radio going into a low-power idle mode during dormant period and it is expensive, both power-wise and time-wise, to transition the radio from the idle state to an active state. According to the detailed study in [15], the power consumption in transitioning from idle to connected state in the LTE radio is about 1,300 mW compared to 11 mW in idle state. Therefore, a device can opportunistically upload crowdsensed information depending on its radio state and resource state (such as, remaining battery life). Third, consider the common case that to fuse crowdsensing data and obtain actionable knowledge, one needs only a certain amount of redundancy in the sensed readings in time and space. For example, to create a hyperlocal weather map, one needs pressure readings only about once in 5 minutes and from only about 2 devices in a 500 meters radius. Under the assumption that there is a dense presence of mobile devices taking part in the crowdsensing activity, as is likely to happen for example in an urban or a college campus environment, then an orchestrator can determine dynamically which devices to ask for crowdsensing data. This is in sharp contrast to existing crowdsensing frameworks [20, 28, 35, 37], which do not take a network-wide view and therefore cannot orchestrate individual devices. Instead, the processing is done on an individual device basis which loses out this important scope for resource optimization.

One previous approach, Piggyback CrowdSensing (PCS) [20], has looked at the issue of energy savings while performing crowdsensing activities. It leverages only the first of the three opportunities outlined above. It piggybacks sensing data onto applications that are active and using the radio to transmit sensing data. Examples of such applications include browsers, phone calls, maps, etc. Their method predicts when an app is going to be used and based on that, they decide whether to upload the sensing data now or wait to piggyback on normal traffic. However the fundamental limitation of this work is that it needs to predict application usage for each user and considering user diversity, this is difficult to do quickly.

Component	Power Consumption
Accelerometer	21mW
Gyroscope	130 mW
Barometer	110 mW
GPS	176 mW
Microphone	101 mW
Camera	>1000 mW
LTE Module	594 mW to 1700 mW

Table 1. List of power consumption of some components on Samsung Galaxy S4 (In the case of camera, the value of power consumption is dependent on what activity user is doing with camera.)

The system has to be trained and adapted separately to every single user, which imposes a burden on deployment. Their results show (Figure 8 in [20]) that even with 2 months of training, the accuracy of predicting app usage is around 40% (and saturated). For a more generous metric that the actual app that is used is one of the top 5 predicted apps, the accuracy is 60%. These show the difficulty of personalized smartphone usage prediction.

Our proposed solution

Previous work from Carrol *et al.* [8] has shown that the majority of power consumption can be attributed to the GSM module and the display. According to [33], we list some sensors' power consumption in Table 1. We posit that if the crowdsensing task were made aware of the state of the radio, it could do its job in a more energy efficient manner. In order to leverage the two other opportunities for optimized crowdsensing, we propose Sense-Aid that leverages the cellular network and creates a service resident in the network for participatory sensing. It provides middleware that runs on the mobile device and overlaid on existing elements at the edge of a cellular network. It also provides an application API for developers to develop crowdsensing applications. They can easily specify requirements for the crowdsensing activity from the system, such as, the degree of delay tolerance and the minimum spatial density of devices needed to get useful information.

The middleware elements of Sense-Aid that execute on the device and the network cooperate to achieve the optimized crowdsensing. First, we leverage a subtle observation about the radio state of cellular network radios to further optimize individual crowdsensed data upload from a device. Previous work by Huang *et al.*[15] shows that the radio protocol for 4G cellular devices keeps the radio on for a *tail time* after the use of the radio by any application (about 11 seconds for 4G LTE radio stack). This tail time provides an ideal opportunity for crowdsensing data to be opportunistically sent out, without incurring much additional cost. Second, the network has access to information about the presence and location of devices at the granularity of a base station, while the device has local state information, such as, battery state. The network middleware orchestrates which devices are selected at any point in time to participate in the crowdsensing activity. The network middleware itself is distributed and resident among multiple edge elements of the cellular network, such as, the Radio Access Network (RAN) of the cellular network. The Sense-Aid middleware element intelligently schedules sensing and communication tasks on the mobile devices based on their location, radio state, current battery level, and total tolerable energy budget for crowdsensing tasks, while meeting the requirements of specific crowdsensing applications.

We evaluate Sense-Aid through a user study comprising 60 students on our campus who use their regular mobile devices with our app installed. We define various types of crowdsensing tasks, with differing requirements for timeliness and spatial density. We compare the resource consumption due to crowdsensing for these users using our solution Sense-Aid, the prior best solution *PCS*, and a baseline, *Periodic* which senses and communicates crowdsensing information with a fixed periodicity. We also evaluate the foreseeable use case where a user is concurrently participating in multiple crowdsensing activities and show that Sense-Aid is capable of handling that use case. We complement the user study with a trace-driven simulation, which lets us evaluate the behavior of Sense-Aid at a large scale. Based on our user study-based evaluation, Sense-Aid has energy savings of 42.4% to 81.4% over PCS and energy savings of 86.9% to 94.9% over Periodic under different crowdsensing tasks. We find that the energy benefit for Sense-Aid becomes more significant if more concurrent tasks are scheduled on a device or the frequency of communication between device and Sense-Aid server increases.

The paper makes the following contributions:

1. We propose a framework for developing crowdsensing apps that saves energy by using device information and network information to schedule which devices should be selected in sensing and uploading crowdsensing data at any point in time.
2. The framework takes away a significant part of the complexity of developing a participatory sensing application since the applications do not have to maintain a view of the entire environment or keep track of the devices and their locations.
3. We compare the benefits to the current state-of-the-art (*PCS*) and the current state-of-practice (*Periodic* sensing and communication) in user study and show that under the prerequisite of not harming crowdsensing data, the benefit of Sense-Aid is significant in all cases.
4. We demonstrate the benefit of Sense-Aid increases as more users adopt Sense-Aid through trace-based simulation on a real-world popular crowdsensing app (Pressurenet).

The paper is organized as follows. Section 2 talks about aspects of participatory sensing and how the radio usage impacts power consumption. Section 3 describes our framework's architecture and section 4 talks about the implementation details of it. The user study and simulation are evaluated in section 5. Section 6 talks about different angles regarding Sense-Aid. Section 7 looks at related work, and finally the paper concludes with a discussion on the possibilities and limitation of our framework.

2 Background

2.1 Participatory sensing

Participatory sensing is a mechanism of gathering data from individuals equipped with smartphones or some sort of mobile sensors. The data gathered is then mined in some manner to explore various aspects of our world. Participatory sensing applications have a broad range of categories. The most common categories of applications today are related to transportation, weather and environment, and health. For example, under transportation, the traffic pattern of individuals or vehicles can be mapped to measure the carbon footprint or to do urban planning. Examples apps are reported in [9] (for finding parking spaces) and [30] (for validating location information by using image data).

While participatory sensing has gained ground, there are still barriers to making it mainstream. Such barriers are cited in user studies where the users are queried about their attitude toward crowdsensing [34]. The major challenges for effective participatory sensing are how to meaningfully use the detailed crowdsensed data to provide higher level, easily consumable information, how to protect the privacy of the users collecting the data, how to validate the authenticity of the data, how to efficiently roll out a crowdsensing campaign, and how to minimize the resource drain on the mobile device participating in the crowdsensing campaign.

In this work, we address the last two challenges and we delve a little further into the complexity of developing crowdsensing apps. Our observations are based on a study of a host of crowdsensing apps, some of which are open sourced (Pressurenet, Ushahidi Platform, and NYTimes' Hive crowdsourcing platform) and many which are not (in which case we make indirect inferences, such as from the user interface features). The difficulty of deploying the app arises from the fact that a large number of geographically disparate users have to be orchestrated to collect meaningful data. The current crowdsensing applications need frequent measurements and have to do a lot of book-keeping to create a map of the available devices which is rather cumbersome. As a concrete example, om the popular crowdsensing app Pressurenet, 37% of the lines of code were devoted to such book-keeping activities (3,400 out of 9,000 LOC). Further, there does not exist any middleware that can be reused by different crowdsensing app vendors and less critically, no middleware on the mobile devices that can allow multiple crowdsensing apps to co-exist and leverage common functionality. With our middleware solution, we seek to address these two management challenges of crowdsensing deployment.

2.2 Cellular Network Background

We present a brief description of the 4G LTE network architecture, which is mainstream now and the various radio states that determine the energy consumption and available radio resources on the mobile devices. As shown in Figure 3, an LTE network consists of two major components: the Core Network (CN) (or Packet Core, the Level 1) and the Radio Access Network (RAN) (the Level 2). The mobile device, also called the User Equipment or UE (the Level 3), is connected to a physical base station (called evolved NodeB or eNodeB) in RAN. The core network is the backbone of the cellular network, which connects the RAN with the Internet (See [29] for the details of the LTE network).

The allocation of radio resources to mobile devices for wireless communications is governed by the Radio Resource Control (RRC) protocol [1]. 4G LTE networks have two RRC states—RRC_IDLE and RRC_CONNECTED as shown in Figure 4 (Figure 1 in [15]). When a UE in the RRC_IDLE state needs to initiate a wireless communication, it transits to the high power RRC_CONNECTED state. In this state promotion, tens of control messages are exchanged between the UE and the RAN for resource allocation. The actual user data communication happens in the RRC_CONNECTED state. The UE device remains in the high power RRC_CONNECTED state during the "radio tail" period the *T_tail* in Figure 4, after the last user data packet is transmitted or received. During tail tail time, there is no data communication between UE and cellular tower. After the tail time expires, the UE transits back to the low power RRC_IDLE state. For crowdsensing applications, the size of each

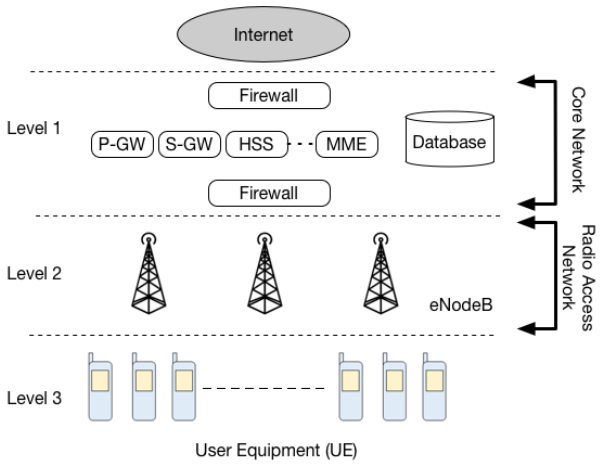

Figure 3. Overview of the 4G LTE architecture showing the network elements. The Core Network (CN) forms Level 1 while the Radio Access Network (RAN) forms Level 2. The User Endpoints (UEs) in Level 3 interact with the eNodeB's. Our solution is embedded between eNodeB and Core Network.

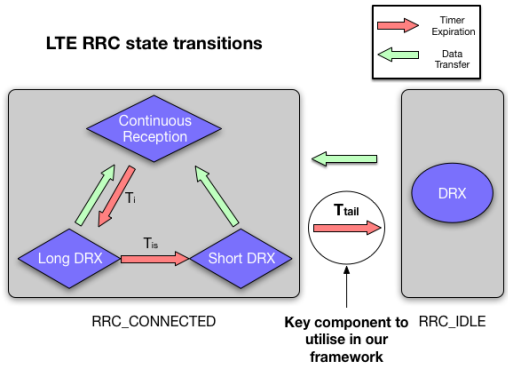

Figure 4. RRC state transitions in LTE. T_i, T_{is}, T_{tail} are timer variables to control state transitions. Sense-Aid will make use of T_{tail} to transmit crowdsensing data before smartphone's radio switches to RRC_IDLE. (Adapted from Figure 1 in [15])

transmitted data is usually small. For example in our user study, each data transmission is only 600 Bytes. However, for other regular network activities like web browsing, the exchanged data size is usually much larger. This observation, coupled with the characteristics of the RRC protocol mentioned above, lead to two design features of Sense-Aid—piggybacking crowdsensing data on top of regular application data (as *PCS* does, but we need to piggyback much less frequently), and using the tail timer opportunistically to send the crowdsensing data, and thus avoiding any expensive radio promotion from IDLE.

3 Sense-Aid Architecture

In this section we first describe the components of Sense-Aid. We first list some terminologies that might be helpful to understand.

- Crowdsensing task (interchangeably, Task) - It is given by crowdsensing application server as introduced in 2. Tasks are specified by multiple parameters and these parameters can be found in Table 2.
- Request - One task will generate multiple requests based on its task requirement. *e.g.*a task lasts for 60 minutes and requires sampling period of 10 minutes will generate 6 requests.
- Qualified devices - In the region that is specified by task, for the devices that have signed up to crowdsensing campaign and locate within that region, Sense-Aid server will evaluate these devices to check if any of them are capable of processing crowdsensing task. There two main reasons that a device becomes unqualified. One is that the device moves out of the region required by specific task. The second reason is that the reported crowdsensing data from one device is invalid or that device does not have the sensor required by task.
- Selected devices - This is the set of devices out of qualified devices, who are selected by server to process crowdsensing task. *Sense-Aid* server will only choose minimum number of devices that satisfy spatial density requirement (one parameter in Table 2).

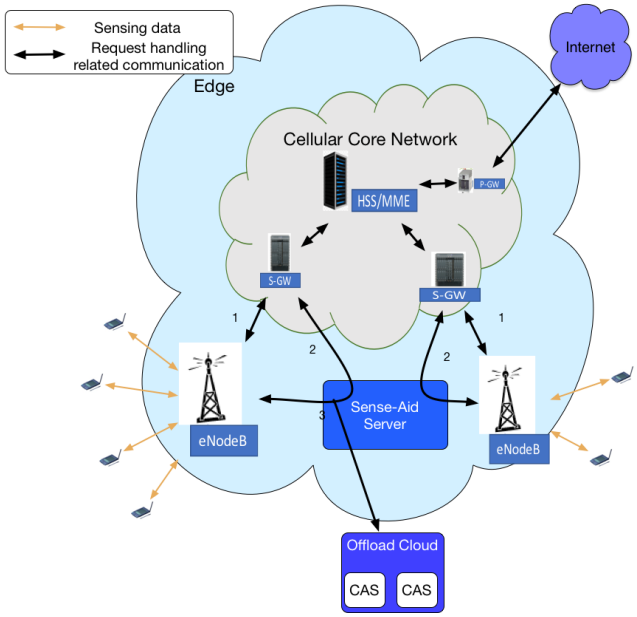

Figure 5. Sense-Aid deployed on a cellular LTE network, showing the three major components: Sense-Aid server, Sense-Aid clients, and Sense-Aid application servers. CAS: Crowdsensing application server. eNodeBs communicate with the cellular core network through path 2 if its incoming traffic from UEs includes crowdsensing data. Otherwise, eNodeBs use path 1.

The design of the framework can be broken down into three major portions as follows:

- Sense-Aid server: This is deployed on the edge of the cellular network and queries components in the edge as well as the core network. It provides the functions of Sense-Aid, namely, keeping track of location and state of the devices, handling tasks from the crowdsensing application server, and scheduling crowdsensing sensing tasks on the devices.
- Server-side library: This handles the communication with the crowdsensing application server, including the initial request for the crowdsensing task, dynamic changes to the task, and transmission of crowdsensing data. This component is deployed on the crowdsensing application server.

This component knows who to contact in the Sense-Aid server and how.

- Client-side library: This handles communication with the crowdsensing apps running on the device, including communicating the scheduling decision and transmission of crowdsensing data. This component is deployed on the mobile device. This component knows how to respond to a scheduling task from the Sense-Aid server.

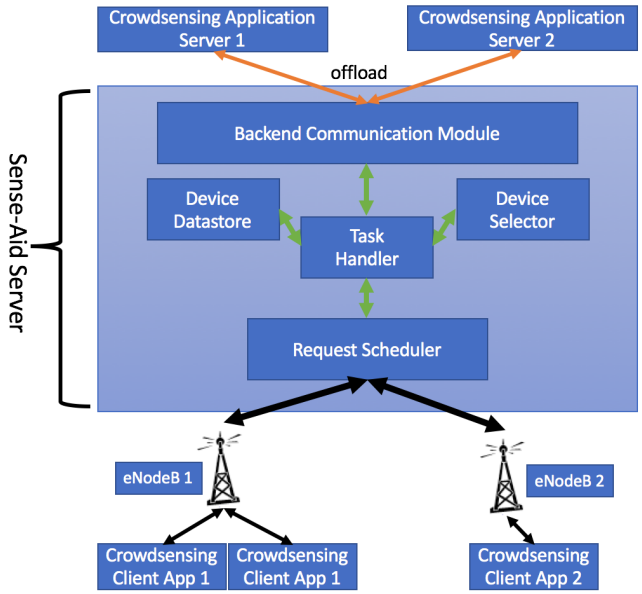

Figure 6. Components of Sense-Aid. Client sensing apps (at the bottom of the figure) are deployed on individual smartphones. Multiple crowdsensing servers can coexist in the infrastructure at the same time.

Figure 5 shows how Sense-Aid would be deployed on the current 4G LTE infrastructure. The eNodeBs are aware of the RRC states of the mobile devices as well as their location. The Sense-Aid server is deployed in the edge of the cellular network, logically between the eNodeB and the Core Network. Sense-Aid can retrieve the location and radio state information of mobile devices from the eNodeB. Notice that there are two connection paths from each eNodeB to S-GW in the core. If an eNodeB notices that some devices are participating in crowdsensing activities, it will pass its traffic through the Sense-Aid server (path 2). Sense-Aid server will offload crowdsensing traffic and then pass the rest of traffic to the S-GW on path 2. After the traffic offloading, Sense-Aid server performs the core algorithms related to our protocol on the crowdsensing traffic then sends the crowdsensing data to crowdsensing application server through path 3. On the other hand, if the eNodeB does not see any crowdsensing activity, it will use its traditional connection method (path 1). To ensure the normal functionality of the whole network, path 1 is the fail-safe path if Sense-Aid crashes to ensure the delivery of regular traffic.

3.1 Workflow

This section describes the workflow of a crowdsensing application which uses Sense-Aid server as middleware to interact with mobile devices. At the beginning there is a bootstrapping process where a user has to sign up to the crowdsensing campaign. User can specify the energy budget and the critical battery level. Once users have signed up for the campaign the crowdsensing application server sends a task to Sense-Aid server, which will generate multiple requests to the Sense-Aid server along with task parameters—the detailed parameters are shown in Table 2. Sense-Aid then looks up the cell towers in the specified area and obtains the list of *qualified devices*. It then runs the device selection algorithm to pick *selected devices*, ensuring that the selection is fair. Once devices have been selected Sense-Aid then schedules the crowdsensing tasks on the mobile devices. On receipt of crowdsensing data, it sends the data back to the crowdsensing application server. Algorithm 1 shows the basic workflow of Sense-Aid. Sense-Aid server maintains a run queue to store all crowdsensing requests from all crowdsensing application servers and a wait queue to store the requests which were popped out but have not timely been scheduled requests.

Algorithm 1 Sense-Aid server workflow

```
procedure Sense-Aid server
    request_select_thread (R, D): ▷ R - Request run queue; D - Devices datastore
    while S doerver has crowdsensing request to schedule.
        Pop request r from R.
        Get n requested devices for request r.
        Get N qualified devices in specific region of t.
        if n ≤ N then
            Run device selector and choose n devices.
            Schedule sensing requests
            Send data from devices to crowdsensing server.
        else
            Move t to wait queue
    wait_check_thread (W):    ▷ W - Requests stored in wait queue.
    Periodically check the request queue;
    if  thenw ← W is satisfiable now
        Move w to run queue R
```

3.2 Sense-Aid Server Design

This is the heart of Sense-Aid and does the majority of the work. The Sense-Aid server consists of the following entities. Logically, each of these entities is centralized. In its physical instantiation, each entity is distributed into multiple instances, which are resident at the edge of the cellular network. Each instance will be located spatially close to the mobile devices that are participating in that crowdsensing activity. This aspect of the design is key to high performance, *i.e.*, low latency, compared to an alternate design where the centralized server is located deep within the core network. Distribution however results in higher complexity and greater integration cost with an existing cellular network.

Task Datastore

The task datastore contains all the tasks for crowdsensing data that have been made to Sense-Aid server from the crowdsensing application servers. The task is typically of the form where it specifies what kind of data is needed (*i.e.*, from what sensors), at what time granularity (*e.g.*, every 1 minute), with what minimum spatial density (*e.g.*, at least 5 devices in one cell of the cellular network or at least 5 devices in multiple cells of the cellular network), and the duration for which the crowdsensing task should be active (*e.g.*, from August 24, 1971 till September 30, 1971). Table 2 provides the complete list of parameters that can be specified. The tasks can

be one-time in which case there is no duration or time granularity specified. Such tasks can be used to supplement data that is already being collected through a regular task mentioned above. We consciously use cell granularity for location rather than the finer granularity of GPS because all crowdsensing activities that we have profiled can be satisfied by this coarser granularity. Importantly, from an energy expenditure standpoint, this location information is already available at the eNodeB and thus, expensive GPS measurements are not needed.

Device Datastore

The device datastore contains relevant information pertaining to the mobile devices whose radio is currently active. For each device, Sense-Aid keeps track of the hash value of the International Mobile Station Equipment Identity (IMEI) code, remaining energy budget, current battery level, number of times a device has been selected for sensing, and a timestamp of most recent radio communication.

Task Handler

The Task handler module consists of two queues: run queue and wait queue. The run queue contains the list of tasks which can be satisfied right away based on the number of available devices and the characteristics of the tasks. The wait queue contains the list of tasks which cannot be satisfied right away. Both queues are sorted by the deadline of the task. The deadline is determined from the sampling duration or the start time and stop time specified by the task. In case the task is specified with a certain frequency, Sense-Aid automatically generates multiple tasks to be scheduled. For example, a task with sampling duration of an hour and a sampling frequency of every 5 minutes will result in the generation of 12 tasks. Each task is then pushed onto the run queue with appropriate deadline.

Parameter	Details
int sensor_type	Sensors are adapted from Android developer website We only use barometer in user study.
int sampling_period	Specifies the period between two consecutive samples. e.g. sampling_period of 5 secs implies one sample every 5 seconds.
int sampling_duration	Duration for which sensing needs to be done.
DateTime start_time	Time to start sensing
DateTime end_time	Time to end sensing
float area_radius	Given a specific location by crowdsensing task the radius determines the area for Sense-Aid server to select devices.
int spatial_density	Represents the number of devices required in the area specified above.
String device_type	This is an optional parameter to specify a particular device type. e.g. iPhone6 , LG G2 etc.

Table 2. Task parameters. One can either specify a sampling duration or a start and stop time. In case sampling duration is specified, the start time is set to current time.

Device Selector

Suppose the total number of devices available in the database at a given time instance is N and n is the number of requested devices for a given task. If

$$n > N$$

the task is deemed unsatisfiable and placed on the wait queue. However if

$$n <= N$$

then the device selector chooses the best n out of the N devices in the following way. Each device is assigned a score based on four factors — the energy consumed by the device for *all* crowdsensing tasks, counted since the beginning of some reasonable time interval, say the week; the number of times the device has been used for *all* crowdsensing tasks, again for the same time interval; the current remaining battery level of the device; and, the timestamp of most recent radio communication of the device. The device with a *lower score is preferred* over a device with a higher score. For simplicity, we use a linear combination of these four factors. Let us define the following variables.

- E_i : energy consumed by device i for the crowdsensing task.
- U_i : number of times device i has been used
- CBL_i : current battery level in percentage
- TTL_i : current timestamp - timestamp of most recent radio communication

The scoring function is defined as follows.

$$Score(i) = \alpha \cdot E_i + \beta \cdot U_i + \gamma \cdot (100 - CBL_i) + \phi \cdot TTL_i$$

Recall that the TTL value is important because it can be used to determine how much of the tail time is still left. A smaller value will increase the likelihood that the sensing task can be completed and the sensed value opportunistically sent across during the tail timer. Coefficients α, β, γ and ϕ are configurable and will determine the importance of each parameter in the selection criteria. There are also hard cutoffs for the first three criteria to constrain, Sense-Aid server never picks a device more than a certain number of times, when it has already expended a certain amount of energy for crowdsensing tasks, or when its battery is depleted beyond a level specified by the user.

Task Scheduler

Once the devices have been selected, the task scheduler schedules the crowdsensing tasks on the selected mobile devices. It communicates this to the device through the client-side library. Once the crowdsensing task is complete, the data is sent back by the mobile device to the crowdsensing application server, using the server-side library. This crowdsensing data still goes through the Sense-Aid server, rather than directly to the application server. This is to maintain user privacy by filtering out private information at Sense-Aid server. A second reason is that if a mobile device becomes unresponsive, then the Sense-Aid server can exclude it from future selections.

3.3 Crowdsensing Client Design

Developing crowdsensing client application is rather simple using the APIs provided by Sense-Aid client side library. The APIs are `register()`, `deregister()`, `update_preferences()`, `start_sensing()`, and `send_sense_data()`. Once users register at Sense-Aid server, Sense-Aid server will put them into Device Datastore. Users can choose their preferences (*e.g.*setting energy budget, choosing when to participate). Once a device is selected by Sense-Aid server, it will receive tasks from Sense-Aid server with what sensors to use and when to upload the crowdsensing data. The rest work for the client is only to sample the sensor and upload the value at the specified time. Client does not need to do GPS measurement since Sense-Aid

server knows its coarse-grained location from the eNodeBs and that location is adequate for crowdsensing tasks.

3.4 Crowdsensing Application Server Design

Another component is the crowdsensing application server, which develops the application that will make use of the crowdsensing data, such as for hyperlocal weather map or traffic conditions. Multiple crowdsensing application servers can interact with Sense-Aid and in fact the same mobile device can have multiple concurrent crowdsensing apps running on it. The crowdsensing application server uses an API defined by us to easily create task descriptions and push out these descriptions to the Sense-Aid server. The API calls supported are `task()` (to create a crowdsensing task and specify its properties), `update_task_param()` (to update parameters of an already created task), `delete_task()` (to remove a task from the system), and `receive_sensed_data()` (callback function invoked when there is crowdsensing data available for this server). With the help of Sense-Aid server, the goal is to simplify the development of a crowdsensing application and orchestrating requisite mobile devices.

3.5 Design for Scalability

A thread in the Sense-Aid server keeps updating the device datastore with the mobile devices that are available in a geographical region. To do this, it queries the eNodeBs in the background. A separate thread handles the tasks that are submitted by the crowdsensing application servers. It determines if an appropriate number of devices are available for satisfying the task. If it is, then it proceeds to schedule that task. The scalability of our infrastructure arises from the observation that the workload is embarrassingly parallel. If more number of mobile devices become participants in a crowdsensing campaign in a geographical region, then more server instances can be deployed in that geographical region. Each server instance handles a subset of the mobile devices and ranks their suitability for satisfying any crowdsensing task. Then in a relatively lightweight merge step, a single rank-ordered list of devices can be created. Equivalently, if the sensing task becomes more heavy-duty, say sampling period goes down from 5 minutes to 10 seconds, then more server instances can be spun up in the edge. Another form of scalability is that the cellular network provider adds more eNodeBs to handle extra demand on the network in a region. Then Sense-Aid server simply needs to have its database of available eNodeBs in a region updated so that it can query all of them for the Sense-Aid device datastore. One key to the scalability is to leverage already existing pieces of information that are available in various components of an existing 4G LTE cellular network, such as, the list of mobile devices and their radio state available at the eNodeBs. Leveraging this information means we do not have to exert extra network pressure to collect such information.

4 Implementation

This section describes the major aspects of the implementation of Sense-Aid. Since it is hard to actually deploy our framework on the actual cellular network, we had to deploy the core components of it (Figure 6) on a proxy server and we talk about these simplifications here. We set up two proxy servers on two 64 bit, 12 core machines with 16 GB of RAM and running Ubuntu 16.04 each. The simulation proxy server is to run the network trace simulation and the user study server is to run actual user study with 60 people on campus.

We implemented two variants of Sense-Aid, *Sense-Aid Basic* and *Sense-Aid Complete*. In *Sense-Aid Basic*, crowdsensing communications between device and Sense-Aid server typically happen in the cellular tail time and any communication, whether crowdsensing or regular, causes the tail timer to be reset. Thus, the length of time that the device is in the RRC_CONNECTED state gets extended leading to lower energy savings. If no change is made to the radio protocol stack, this will be the default behavior in the RRC protocol. In the case of *Sense-Aid Complete*, the tail timer is not reset when crowdsensing data is uploaded during the tail time. Thus, the radio will transition to the lower powered RRC_IDLE state as usual. Hence, *Sense-Aid Complete* will save more energy than *Sense-Aid Basic*. There are two reasons to present *Sense-Aid Basic* given that a better alternate solution exists. First, the RRC protocol is decided by the network carriers and they may not allow us to change the tail timer configuration by locking down the protocol stack. Second, the crowdsensing data may only become available close to the end of the tail time, thus not allowing enough time for the upload and necessitating an elongation of the tail time.

4.1 Device Datastore

We use a MySQL database for this with the device id, which is a hash of the phone's IMEI, as the primary key and the database is indexed by the geographical location. This enables us to do a fast lookup of available devices for a given location. Since ideally Sense-Aid is integrated with the cellular network as shown in Figure 5, the network is aware of the state of the device's radio and its approximate location without the device having to do anything about it. In our implementation, however, since we do not have information from the network, we use the GPS on smartphones to get the devices' location information. We also implemented a service thread to contact Sense-Aid server with information about the current battery level (dynamic information) and a hash of the IMEI and the energy budget (static information). This service sends a packet with the above dynamic information to the user study proxy server only when the radio tail time is found and has a very small network footprint. When we calculate the energy use for all three frameworks, we ignore energy consumption for control messages.

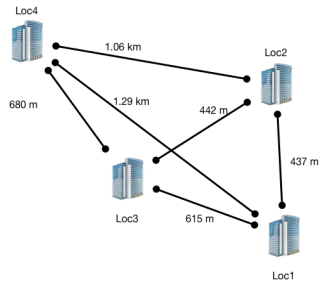

Figure 7. User study target buildings

4.2 Infer the tail time

In the user study, one challenge for the Sense-Aid client app is how to know the network tail time on the smartphone. In Android, there is no programming interface provide developers the state of the radio beyond the type of the network that the smartphone currently connects to *e.g.*Wifi, 3G, 4G LTE. Since we cannot directly know the LTE radio state on the smartphone, we have to find a way to infer it. We install an app called *tPacketCapture* [3] on the mobile device, which does not need the device to be rooted.

This app uses local VPN to capture network packets and stores packet files locally. In the Sense-Aid client app, we listen to the change of the file directory where *tPacketCapture* stores packet files. Whenever any change happens to that file directory, we infer that some communication has happened and we start the tail timer. To show a visualization of the radio state, we did an experiment where we developed a tail-time testing app to connect to our server only at a network tail time of smartphone while one person is using the smartphone normally. As shown in Figure 8, the X-axis is time in seconds and this data is output by AT&T's ARO tool [25]. The first chunk of tail time happening between 591 secs and 592.5 secs is due to the first regular packet traffic. At 592.5 secs, crowdsensing packets are ready and tail timer is inferred, then the crowdsensing packets are sent out. After a total of 120 ms of long and short DRX, the tail timer continues for about 10 secs and then the radio goes to idle at around 602.5 secs. In total, without reseting the tail timer when Sense-Aid traffic sent out during tail time, the tail time is about 11.5 secs.

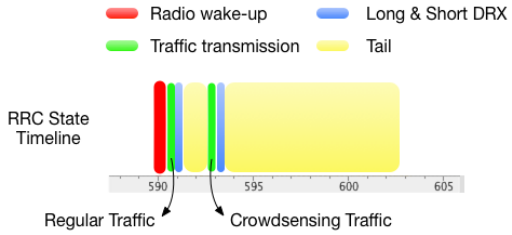

Figure 8. Visualizing network activity, crowdsensing activity, and LTE radio states.

5 Sense-Aid Evaluation

5.1 User Study Experiment Setup

The user study lasted for one week from Tuesday to Sunday. The tasks in the user study follow specifications in Table 2 and all tasks need barometer values from 4 location on our campus, *Student Union (Loc1), EE department (Loc2), CS department (Loc3), University Gym (Loc4).* Their geographic locations are shown in Figure 7.. We have conducted 4 experiments. In each experiment, we have three device sets and each of them have 20 devices carried by 20 students who are on campus. Each set corresponds to one framework and the devices are running the corresponding client application. We are comparing the energy efficiency of the following three frameworks:

1. *Periodic*: The client application does periodic sensing and periodic communication with the server.
2. *PCS* (Piggyback crowdsensing): The client application piggybacks the crowdsensing data to the *PCS* server based on prediction of the normal network usage on the device.
3. *Sense-Aid*: We consider both the Basic and the Complete versions of our framework.

As shown in Table 2, there are multiple parameters for a typical crowdsensing task. We varied only one parameter and fixed others in each experiment as summarized in Table 3. The table shows the summary results for the energy savings of Sense-Aid over Periodic and PCS. The energy saving metric is calculated as (Energy used in comparison framework-Energy used in Sense-Aid)/Energy used in comparison framework. The min and max are calculated over the range of values of the varying parameter. Within each experiment, one test refers to a particular setting of the varying parameter and is run for the duration specified in the table, *e.g.*, experiment 1 has

Experiment Number	Varying parameter	Default parameters	Energy savings 1: *Sense-Aid Basic/Periodic* 2: *Sense-Aid Complete/Periodic* 3: *Sense-Aid Basic/PCS* 4: *Sense-Aid Complete/PCS* Average (Min, Max)
Experiment 1	Area radius 100m, 200m, 300m, 600m 500m, 1000m	test duration = 1.5 hrs/test number of tasks/device =1 sampling period = 10mins spatial density = 2	1: 94.3%(88.7%, 98.3%); 2: 94.9%(90%, 98.5%); 3: 79% (65.9%, 92.5%); 4: 81.4% (68.6%, 93.3%) ;
Experiment 2	Sampling period (1, 5, 10)[min]	test duration = 2 hrs/test number of tasks/device = 1 spatial density=3 area radius = 500m	1: 86.6% (80.9%, 89.6%); 2: 88.1% (83.1%, 90.7%); 3: 42.1% (27,2%, 57.8%); 4: 48.3% (35.1%, 62.4%);
Experiment 3	Spatial_density (1, 2, 5, 7)	test duration = 1.5 hrs/test number of tasks/device = 1 sampling period = 5mins area radius = 1000m	1: 87.5% (76.9%, 96.2%); 2: 88.8% (79.4%, 96.7%); 3: 55.7% (17.4%, 88.2%); 4: 60.5% (26.3%, 89.5%);
Experiment 4	Number of tasks per device (3, 5, 10, 15)	test duration = 1.5 hrs/test sampling period = 5mins spatial density = 3 area radius = 500m	1: 85.3% (84.4%, 86.5%); 2: 86.9% (86.1%, 87.9%); 3: 35.4% (16.7%, 57.8%); 4: 42.4% (25.7%, 62.4%);

Table 3. Experimental parameter settings in user study and summary energy saving results.

6 tests. For easier comparison, we add the 2% battery threshold bar to all the energy cost figures, a number that users were comfortable spending on crowdsensing tasks according to our survey result from Section 1. This is calculated using a nominal 1800mAh, 3.82V battery and its value is 496 Joules.

5.2 Experiment 1: Impact of area radius

In Experiment 1, we vary the area radius, centered at each location, within which devices will be considered for the crowdsensing campaign. This experiment has three goals—determine how the number of qualified devices varies with the area radius, what is the energy saving of Sense-Aid (both variants) compared to the previous frameworks, and see if devices are chosen with some fairness.

We conducted this experiment at 4 different locations. The results are similar across these four locations and so we only present the result from CS department (Loc3). From Figure 9, we see that as we increase the area radius, the number of qualified devices expectedly increases since a larger area is now being considered centered at Loc3. The difference among *Periodic*, *PCS*, and *Sense-Aid* is due to the random mobility patterns of users and not due to any design feature of the frameworks. In Figure 10 we present the total energy cost for all devices. Since *Periodic* costs much more energy than the other two frameworks, to make the energy comparison obvious in the figure, we omit the bar for *Periodic* but the energy comparison with *Periodic* can be seen in Table 3. With the help of the device selection algorithm, Sense-Aid can save much energy especially when the spatial density requirement is less than the number of qualified devices. On average, the total energy saving is 79% over *PCS* for *Sense-Aid Basic* and 81.4% for *Sense-Aid Complete*. The minimum and maximum energy savings for *Sense-Aid Complete* over *PCS* are 68.6% and 93.3%. The beefit of Sense-Aid increases as the area radius increases. This is expected because when the area radius is larger, *PCS* will assign tasks to more devices and therefore the total energy increases. However, Sense-Aid will keep choosing the minimum number of devices required to satisfy the task's spatial density requirement. Selecting *all* qualified devices in Sense-Aid still saves energy compared to PCS and Periodic, respectively 54.5% and 60.3%. This shows that even without the global orchestration, Sense-Aid is effective because it triggers each device to upload crowdsensing data at an opportune time.

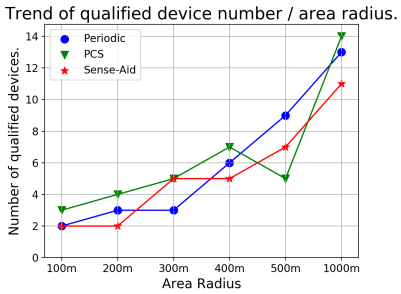

Figure 9. The number of qualified devices at the CS department for each value of area radius. The trends are similar in all 4 locations in experiment 4.

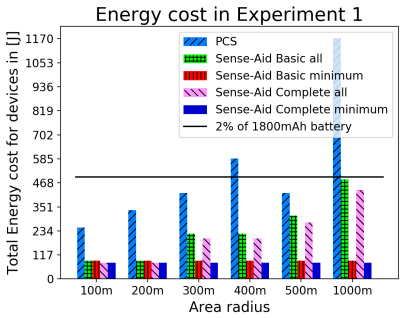

Figure 10. Total energy cost of crowdsensing summed across devices. Periodic is omitted as its energy cost is much higher than other setups. Both variants of Sense-Aid use significantly less energy than PCS.

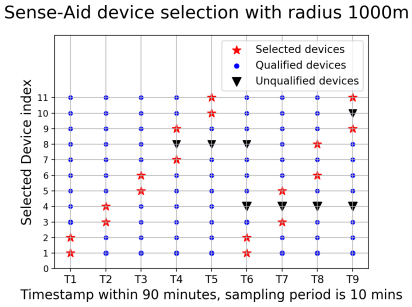

Figure 11. Visualization of Sense-Aid device selection algorithm. Sampling happens once every 10 minutes, and every time server requires data from two devices. Each device is selected either once or twice, showing that the selection is fair.

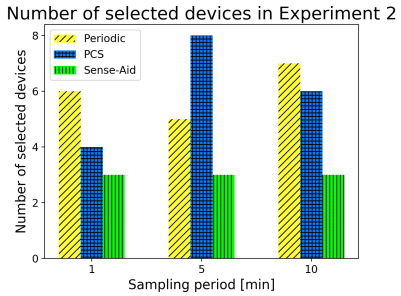

Figure 12. The number of selected devices for each test in Experiment 2. Each request requires minimum of 3 devices.

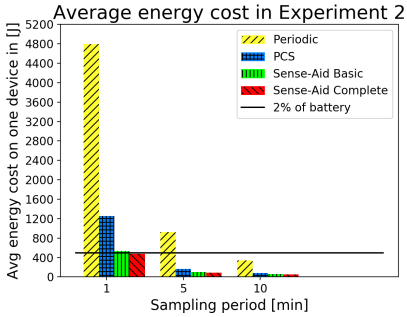

Figure 13. Average energy cost per device with varying sampling period in Experiment 2. *Sense-Aid Basic* and *Complete* cost relatively less energy over *PCS* when the sampling period is low.

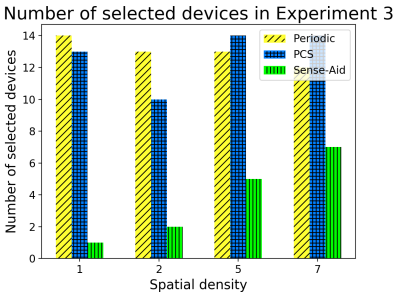

Figure 14. The number of selected devices with varying required spatial density of devices in Experiment 3.

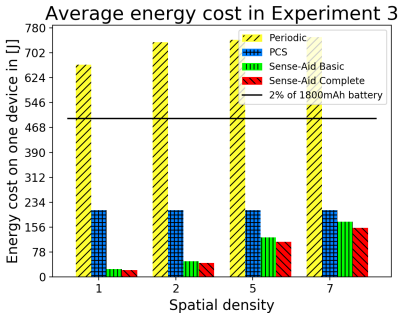

Figure 15. Average energy cost per device in Experiment 3. Increased spatial density affects only Sense-Aid's energy costs. However, even at the higher end, Sense-Aid still uses less energy than PCS and Periodic.

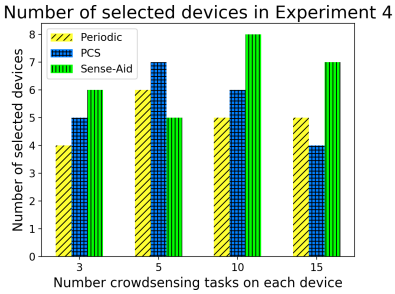

Figure 16. Number of selected devices when there are multiple crowdsensing tasks in Experiment 4. Each request requires a minimum of 3 devices. Periodic and PCS will choose whichever device that is qualified for all requests while Sense-Aid will intelligently orchestrate among all qualified devices to choose only 3 of them for one request.

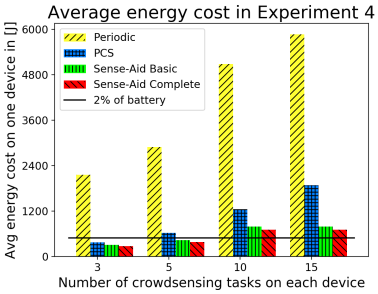

Figure 17. Average energy cost per device when there are multiple crowdsensing tasks in Experiment 4. Higher number of tasks results in higher energy costs for all setups. But *Sense-Aid Basic* and *Complete* can relatively less energy over *PCS* and *Periodic* when more tasks exist in the system

The significant energy efficiency in Sense-Aid is due in part to the fairness criterion in the device selector algorithm. To show how Sense-Aid is fairly selecting devices among qualified devices, we plot Figure 11. The data is based on 1000 meters area radius, 1 task at CS department, sampling period of 10 minutes, and spatial density of 2. As seen in Figure 9, there are 11 users in that region for Sense-Aid. Within the 90 minutes, the selection algorithm ran for 9 times. Now consider the devices selected at each execution of the device selector algorithm. At T1, devices 1 and 2 are selected at the initial run of device selector. At T2, the scores of devices 1 and 2 are higher than others (since they have already been selected, the fairness criterion), and it selects the next two devices that are qualified, devices 3 and 4. At T4, one would expect that Sense-Aid will select devices 7 and 8, but it actually selected devices 7 and 9.

We manually checked the control messages in our database and found out that at T4 device 8 is not within the 1000 m radius and therefore it became unqualified. At T8, device 8 comes back to the area and since by then, all the other 10 devices have been scheduled and this device has lowest score, device 8 is selected.

5.3 Experiment 2: Impact of sampling period

In Experiment 2, our goal is to see how the sampling period of a crowdsensing task affects the energy cost. As a form of root cause analysis, we analyze how many devices are selected for the crowdsensing task by each of the three frameworks.

As seen in Figure 12, every test for each framework has enough number of participants since all frameworks successfully selected devices equal in number or greater than the requirement (3 in this case). For Sense-Aid, the number of selected devices is exactly equal to the spatial density requirement, irrespective of the sampling period. This is due to the device selection algorithm, which orchestrates the devices in the neighborhood to select only the required number from the qualified set of devices. The two other frameworks however pick all of the qualified devices for crowdsensing. Their variation with the sampling period is only to random mobility patterns of the users.

From Figure 13, we see that with the increase of sampling period, the device-wise energy cost decreases since the task would require less network communication between the device and the server. At low sampling period where there are more connections than those in high sampling period, *Sense-Aid Basic* and *Complete* can save more energy than *PCS* and *Periodic*. On average, across the three tests, *Sense-Aid Basic* saves 42.1% and 86.6% over *PCS* and *Periodic* while *Sense-Aid Complete* saves 48.3% and 88.1% over *PCS* and *Periodic*, respectively. The minimum and maximum energy saving for *Sense-Aid Complete* over *PCS* are 35.1% and 62.4% as well as 83.1% and 90.7% over *Periodic*. When the crowdsensing task requires more frequent communication, the energy saving for Sense-Aid is more pronounced.

Another point of note is that even though we only set one task to each device, in the case of 1 minute sampling period, all frameworks go over the 2% battery threshold because the network activity for crowdsensing tasks is too frequent. Even here Sense-Aid has the lowest energy consumption.

5.4 Experiment 3: Impact of spatial density requirement

In Experiment 3, we vary the task's required spatial density of devices and observe the effect on energy cost. We set up the area radius to be 1000 meters to ensure enough number of qualified devices for each request.

As seen in Figure 14, with the increase in spatial density, Sense-Aid expectedly selects an increasing number of devices. For *Periodic* and *PCS*, they have do not have the selection intelligence, so they always choose all the qualified devices to be selected devices. Hence, there is no discernible pattern in the number of selected devices across the spatial density in these two frameworks.

Figure 15 shows the device-wise average energy consumption for the three frameworks. The average energy consumption for *Periodic* and *PCS* remains nearly unchanged but there are still user behavior randomness such as temporarily some selected devices moves out of required area. Sense-Aid server has intelligent device selection mechanism therefore only the required number of devices are selected for the crowdsensing task. Thus, the average energy cost on each device is lower than for the other two frameworks. In the test, there were 10 qualified devices for Sense-Aid. Under spatial density 1, the Sense-Aid server is orchestrating among those 10 devices to schedule the sensing task on a single device at a time. For a given device, the frequency with which it is scheduled to do a sensing task is 7X when the spatial density is 7 compared to 1. Therefore, the average energy cost is also 7X for Sense-Aid.

On average *Sense-Aid Basic* saves 55.7% and 87.5% in energy over *PCS* and *Periodic* while *Sense-Aid Complete* saves 60.5% and 88.8% over *PCS* and *Periodic*. The minimum and maximum energy savings for *Sense-Aid Complete* over *PCS* are 26.3% and 89.5% as well as 79.4% and 96.7% over *Periodic*. As we expected, the maximum benefit due to Sense-Aid occurs at low spatial density requirement (higher amount of overprovisioning in the other two frameworks) while the minimum benefit occurs at high spatial density requirement.

5.5 Experiment 4: Impact of number of concurrent tasks on one device

In Experiment 4, the number of concurrent crowdsensing tasks in the system is varied. The goal is to evaluate if Sense-Aid can handle more than one crowdsensing task active concurrently on the mobile devices that it orchestrates. This is in line with our vision that our service will enable crowdsensing campaigns to be easily rolled out and therefore there will be multiple tasks, possibly from different crowdsensing application providers.

Figure 16 shows the number of devices selected in each test. Notice that the number of selected devices in Sense-Aid is not flat any more, as in previous experiments. The reason is that there are multiple tasks going on concurrently and every task is an independent schedulable entity and requires 3 devices to participate. Since in the experiment, we do not have that many qualified devices to ensure one task per device, Sense-Aid server will choose from the limited number of qualified devices and will schedule multiple tasks on them.

Figure 17 shows the energy cost on one device. On average *Sense-Aid Basic* saves 35.4% and 85.3% over *PCS* and *Periodic* while *Sense-Aid Complete* saves 42.4% and 86.9% over *PCS* and *Periodic*, respectively. The minimum and maximum energy saving for *Sense-Aid Complete* over *PCS* are 25.7% and 62.4% as well as 86.1% and 87.9% over *Periodic*. As expected, the minimum benefit occurs at a small number of concurrent tasks, while the maximum benefit occurs with multiple crowdsensing tasks scheduled on the same device. This is because Sense-Aid can orchestrate better among the devices to schedule the multiple crowdsensing activities on them.

5.6 Effect of PCS prediction accuracy

For the 4 experiments above, the assumption is that the App usage prediction accuracy in *PCS* is 40% which is the saturated accuracy observed in [20], Figure 8, for the prediction model using top 1 app to predict. At this accuracy, the energy cost of *PCS* is 41% higher than *Sense-Aid Basic* and 58% higher than *Sense-Aid Complete*. To clearly show the benefits achieved by Sense-Aid over *PCS*, we build the energy cost model for *PCS* under different prediction accuracies as shown in Figure 18. Ideally, when app prediction accuracy is 100%, *PCS* can achieve better performance than *Sense-Aid Basic* and *Sense-Aid Complete*. In this ideal case, *PCS* costs 75.8% energy of *Sense-Aid Basic* and 85% energy of *Sense-Aid Complete*. The implication of this is that purely local decision can be made at each device for uploading crowdsensing data. However, the challenge of building highly accurate, personalized model for usage of applications on a mobile device means that one will likely have

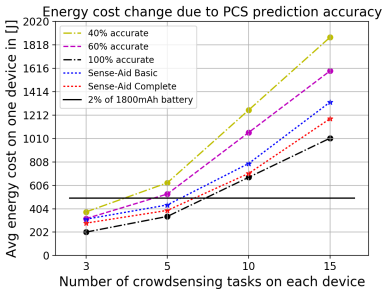

Figure 18. At different prediction accuracies of *PCS*, Sense-Aid over performs the best two practical cases and performs comparable with the ideal case.

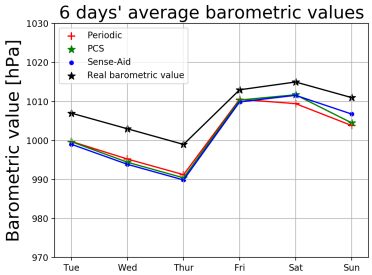

Figure 19. The barometric pressure values collected through crowdsensing in our user study follow the same trend as the actual value in our city. The three crowdsensing frameworks give almost identical results while Sense-Aid is the most energy efficient.

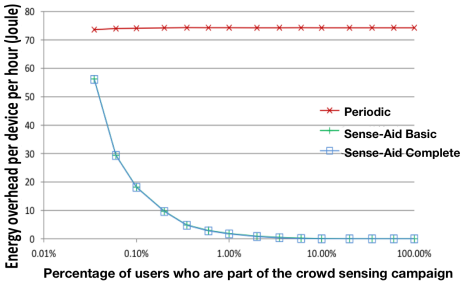

Figure 20. Energy overhead per device per hour of crowdsensing, with varying percentage of users who are part of the crowdsensing campaign. In the *Periodic* setup, all devices need to turn their radio on and take a GPS reading. Sense-Aid Basic and Complete are comparable in performance. The overhead reaches zero when at least 20% of the devices are signed on for the crowdsensing campaign.

to use both local and network information (as we do) to achieve energy-efficient crowdsensing.

5.7 Integrity of sensed data

After the discussion in previous sections, we have shown that *Sense-Aid* either *Basic* or *Complete* beats *PCS* and *baseline* in energy efficiency. However, an important factor in crowdsensing activities is the integrity of the collected data from devices. Therefore, if Sense-Aid can not ensure the data integrity, it makes no sense to achieve such high energy efficiency. Here, we evaluate if Sense-Aid maintains the data integrity despite its more energy-optimizing data collection strategy. Figure 19 presents the average sensed data (barometric pressure) on each day of the user study. We find that the values from each of the three crowdsensing frameworks is almost identical. Also, we look for the real weekly barometric value in our city from Wunderground [4], a weather forecast website. From Tuesday to Sunday, the trend of barometric value in our user study, whatever the frameworks we base on, is similar to that of the real value. The gap between real value and collected value on each day is because of distance between our campus and the measurement station used in Wunderground. They are 2.6 miles away from each other and roughly 20 meters difference in altitude. Therefore, this difference of 0.8% in the pressure reading can be expected.

5.8 Simulation: Energy evaluation for large number of devices

The simulation aims to measure the energy savings of Sense-Aid for a large number of devices, arising from the two features: device location awareness as well as radio state information. To achieve this, we run a simulation based on traces of real traffic. The traces come from a 3G UMTS network of a major U.S. cellular network provider. The simulated area is a 1 mile by 1 mile area in downtown San Francisco. This is a high density area for smartphone devices. To mimic reality that not all users will sign on to participate in any crowdsensing campaign, we vary the percentage of users who are part of the crowdsensing campaign ranging from 0.035% to 100%. The traces come from 3 - 7 pm of a randomly selected non-holiday weekday in November 2014. The traces report radio communication activity of each device individually, every 2 seconds. The number of devices whose radio is active at any given time point in the simulation is of the order of a hundred devices.

The simulation works as follows. Before simulation begins, each user present in the trace is classified as either part of the crowdsensing campaign or not, selected randomly based on the percentage of participating devices. For simplicity, at any point during simulation, each user's device is in one of the two states: radio on or radio off. A device's radio comes on as soon as there is a communication activity, and stays on for T_{tail}. The total duration that a device's radio is on as well as the number of GPS readings are recorded and used to compute the energy overhead of crowdsensing. The energy consumption of the radio is based on measurements by Huang *et al.*, while GPS's energy consumption is estimated at 1 Joule per location fix [22].

Tasks are generated every 10 minutes. Each task requires 10 devices in the area. We assume that the sensed quantity (pressure, etc.) does not vary much in the 1 sq. mi. area, and hence the task can ask for any 10 devices in the area, without needing to create a task for a smaller area. We compare three different setups: a) *Periodic*, b) *Sense-Aid Basic*, and c) *Sense-Aid Complete*. We omit *PCS* here because we cannot run the app prediction model using network traces. In *Periodic*, the framework sends every task to every device. Each device then takes a GPS reading and sends its location to the framework. The framework selects the required number of devices, which perform the sensing. We measure the energy *overhead*, *i.e.*, only the energy consumed due to the crowdsensing task.

Results of the simulations are shown in Figure 20. In the *Periodic* case, energy consumption is independent of the percentage of users who are in the campaign, since it only affects the number of sensing operations per device, which has negligible impact on overall consumption. In *Periodic*, GPS accounts for 8% of total energy consumption. The difference between Sense-Aid Basic and Sense-Aid Complete is negligible (the two curves are in fact visually overlapping), indicating that it is not necessary to modify the phone's operating system to get the full benefits of Sense-Aid. With higher number of users in the campaign, it is more likely that there are enough devices whose radio is already on, so the overhead becomes lower with Sense-Aid. The overhead reaches *zero* when at least 20% of the devices are signed on for the crowdsensing campaign. When there are not enough devices whose radio is already on, some devices need to be woken up and incur the 11-second radio tail time. However, even when the total number of devices

in the campaign is as low as 4 or 7 times the required number of devices, the energy savings due to Sense-Aid are still significant at 76% and 87% respectively, compared to *Periodic*.

6 Discussion

We anticipate that one model of deployment of Sense-Aid will be by the cellular network provider. The provider will deploy the different components of Sense-Aid respectively at the RAN and at the core network. Through this, Sense-Aid can leverage information that is already collected by the cellular network provider, such as the cell ID to which a device is associated, or information that can be readily inferred by the provider, such as the radio state of the device. The provider can provide this service either as a philanthropic activity since many crowdsensing apps have broad community benefit, or can charge the crowdsensing app development organization for the support services. As we have argued earlier in the paper (in Sections 1 and 4), this simplifies the development and deployment effort of the app developer because many book-keeping functions are taken care of and it reduces the energy demand from the client since it no longer has to take GPS measurements for tagging the crowdsensed data. An alternate business model is that this is provided as a service by a third-party provider, separate from the cellular network provider and the app development organization. Such a provider may (or may not) have a relationship with the cellular provider. If such a relationship exists, then the third-party service provider can deploy code close to (or at the) edge of the cellular network and thus observe easily the radio states of the different devices. An analogy can be drawn to CDN providers and internet backbone service providers.

If a user is persnickety about her privacy in participating in a crowdsensing campaign, then Sense-Aid should assuage some of her concers. As long as there is trust in the cellular network provider (and this is required in the current service model), then the Sense-Aid server should be trusted. No per-device data (such as, IMEI number) need be made visible to the crowdsensing application server.

7 Related Work

The concept of participatory sensing or crowdsensing has been around for several years since at least 2006 [32]. The popularity of smartphones and tablets, and the ease with which various types of sensor data can be collected from such mobile devices have led to the surge of mobile crowdsensing applications in recent years. One of the major concerns for crowdsensing applications is the energy consumption on the mobile devices that contribute sensor data to such applications. Sampling a sensor like GPS on the mobile device and transmitting the sensed data over the mobile network can consume significant amount of energy. The control plane overhead for setting up and tearing down the connection with the cellular network can also be expensive. This problem is especially severe for many crowdsensing applications, which transmit relatively small amounts of data in a periodic or event-driven manner, as explained by the radio energy characterization shown in [15, 26].

Attempts at making mobile crowdsensing energy-efficient got a major boost with Piggyback Crowdsensing (PCS) [20], which we have described and compared with in our evaluations. A key aspect of crowdsensing applications is to solve the scheduling question, which mobile device needs to sample and transmit which sensor data and at what time instant so that data collection deadlines specified by the user applications are met with high quality of data and low energy consumption? Existing systems make poor scheduling decisions because they use only the local information available on mobile devices or at best, some coarse-grained information available on the centralized servers. Some recent solutions have looked at selecting mobile devices to that some level of coverage of a sensed area is achieved [13, 36, 37]. In [37] for example, the system predicts the movement of users with crowdsensing mobile devices and selects a subset of users so that a geographical region is (probabilistically) covered. In all these works, the device selection is not done on a fine-grained basis—once a device is selected to participate in a crowdsensing task, it is expected to upload the sensed data, independent of its local state, such as, battery state or radio state. This has the obvious drawback that the sensing and consequent communication from a selected device may be quite energy inefficient. In contrast, Sense-Aid selects devices based on their location as well as their local state. Thus, Sense-Aid leverages the global view obtained from the cellular network and the local view available at the device to make better scheduling decisions.

In terms of support for developing crowdsensing applications, there are some useful frameworks such as in [34], Twitch crowdsourcing [31], and Medusa [27]. Medusa has high-level abstractions for specifying the steps required to complete a crowdsensing task, and employs a distributed runtime system that coordinates the execution of these tasks between smartphones and a cluster on the cloud. Through implementing ten crowdsensing tasks on a prototype of Medusa, the authors show that Medusa task descriptions are two orders of magnitude smaller than standalone systems while keeping a low runtime overhead. However, Medusa does not deal with energy efficiency in crowdsensing. One aspect of mobile crowdsensing is collecting reliable data, which has been addressed in [24, 28]. This can be incorporated as another factor in our device selector algorithm.

8 Conclusion

We believe that there are going be two competing pulls on crowdsensing. First, the perceived necessity for crowdsensing will increase, as will the number of apps that provide crowdsensing functionality. Concurrently, users will have even more energy-hungry (non crowdsensing) apps executing on their mobile devices, while the battery capacity of the devices will not rise as quickly. Putting these two trends together, we believe that solutions that can opportunistically leverage existing traffic for sending crowdsensed data will become valuable. Our solution presented here and Piggyback Crowdsensing (PCS) fit in this requirement space.

Our framework presented here, called Sense-Aid, enables energy-efficient crowdsensing to the point where most tasks can be satisfied within the tolerable energy budget of our surveyed users. Sense-Aid leverages cooperation between the device and the cellular network to orchestrate the right set of devices at their right state to perform the crowdsensing action. It provides an API that eases the task of developing a crowdsensing application, by removing much of the book-keeping that we found is present in today's crowdsensing apps. In ongoing work, we are looking at scalability of our framework to large geographic regions, issues of consistency and failures in the data collection, and dynamic tasks that can alter their requirements based on received data.

References

[1] 3gpp lte. http://www.3gpp.org/LTE.

[2] Sunshine. http://https://thesunshine.co/.

[3] tpacketcapture. http://www.taosoftware.co.jp/en/android/packetcapture/.

[4] Wunderground. https://www.wunderground.com.

[5] Bhoraskar, R., Vankadhara, N., Raman, B., and Kulkarni, P. Wolverine: Traffic and road condition estimation using smartphone sensors. In *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on* (2012), IEEE, pp. 1–6.

[6] Brabham, D. C. Crowdsourcing the public participation process for planning projects. *Planning Theory 8*, 3 (2009), 242–262.

[7] Burke, J. A., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., and Srivastava, M. B. Participatory sensing. *Center for Embedded Network Sensing* (2006).

[8] Carroll, A., and Heiser, G. An analysis of power consumption in a smartphone. In *USENIX annual technical conference* (2010), pp. 1–14.

[9] Chen, X., Santos-Neto, E., and Ripeanu, M. Crowdsourcing for on-street smart parking. In *Proceedings of the second ACM international symposium on Design and analysis of intelligent vehicular networks and applications* (2012), ACM, pp. 1–8.

[10] Devarakonda, S., Sevusu, P., Liu, H., Liu, R., Iftode, L., and Nath, B. Real-time air quality monitoring through mobile sensing in metropolitan areas. In *Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing* (2013), ACM, p. 15.

[11] Gohring, N. App feeds scientists atmospheric data from thousands of smartphones., Feb. 2013.

[12] Greengard, S. Following the crowd. *Communications of the ACM 54*, 2 (2011), 20–22.

[13] Hachem, S., Pathak, A., and Issarny, V. Probabilistic registration for large-scale mobile participatory sensing. In *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on* (2013), IEEE, pp. 132–140.

[14] Hasenfratz, D., Saukh, O., Sturzenegger, S., and Thiele, L. Participatory air pollution monitoring using smartphones. *Mobile Sensing* (2012).

[15] Huang, J., Qian, F., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. A close examination of performance and power characteristics of 4g lte networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services* (2012), ACM, pp. 225–238.

[16] Hudson-Smith, A., Batty, M., Crooks, A., and Milton, R. Mapping for the masses accessing web 2.0 through crowdsourcing. *Social science computer review 27*, 4 (2009), 524–538.

[17] Johnston, S. Weathersignal: Big data meets forecasting — guest blog, scientific american blog network, Oct. 2013.

[18] Kanhere, S. S. Participatory sensing: Crowdsourcing data from mobile smartphones in urban spaces. In *Mobile Data Management (MDM), 2011 12th IEEE International Conference on* (2011), vol. 2, IEEE, pp. 3–6.

[19] Lagar-Cavilla, H. A., Joshi, K., Varshavsky, A., Bickford, J., and Parra, D. Traffic backfilling: subsidizing lunch for delay-tolerant applications in umts networks. *ACM SIGOPS Operating Systems Review 45*, 3 (2012), 77–81.

[20] Lane, N. D., Chon, Y., Zhou, L., Zhang, Y., Li, F., Kim, D., Ding, G., Zhao, F., and Cha, H. Piggyback crowdsensing (pcs): energy efficient crowdsourcing of mobile sensor data by exploiting smartphone app opportunities. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (Sensys)* (2013), ACM, pp. 1–14.

[21] Leonardi, C., Cappellotto, A., Caraviello, M., Lepri, B., and Antonelli, F. Secondnose: an air quality mobile crowdsensing system. In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational* (2014), ACM, pp. 1051–1054.

[22] Liu, J., Priyantha, B., Hart, T., Ramos, H. S., Loureiro, A. A., and Wang, Q. Energy efficient gps sensing with cloud offloading. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems* (2012), ACM, pp. 85–98.

[23] Maisonneuve, N., Stevens, M., and Ochab, B. Participatory noise pollution monitoring using mobile phones. *Information Polity 15*, 1 (2010), 51–71.

[24] Meng, C., Jiang, W., Li, Y., Gao, J., Su, L., Ding, H., and Cheng, Y. Truth discovery on crowd sensing of correlated entities. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems* (2015), ACM, pp. 169–182.

[25] Qian, F., Wang, Z., Gerber, A., Mao, Z., Sen, S., and Spatscheck, O. Demo: mobile application resource optimizer (aro). In *Proceedings of the 9th international conference on Mobile systems, applications, and services* (2011), ACM, pp. 369–370.

[26] Qian, F., Wang, Z., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. Characterizing radio resource allocation for 3g networks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (2010), ACM, pp. 137–150.

[27] Ra, M.-R., Liu, B., La Porta, T. F., and Govindan, R. Medusa: A programming framework for crowd-sensing applications. In *Proceedings of the 10th international conference on Mobile systems, applications, and services* (2012), ACM, pp. 337–350.

[28] Ren, J., Zhang, Y., Zhang, K., and Shen, X. S. Sacrm: Social aware crowdsourcing with reputation management in mobile sensing. *Computer Communications 65* (2015), 55–65.

[29] Stefania, S., Issam, T., and Matthew, B. Lte, the umts long term evolution: from theory to practice. *A John Wiley and Sons, Ltd 6* (2009), 136–144.

[30] Talasila, M., Curtmola, R., and Borcea, C. Improving location reliability in crowd sensed data with minimal efforts. In *Wireless and Mobile Networking Conference (WMNC), 2013 6th Joint IFIP* (2013), IEEE, pp. 1–8.

[31] Vaish, R., Wyngarden, K., Chen, J., Cheung, B., and Bernstein, M. S. Twitch crowdsourcing: crowd contributions in short bursts of time. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems* (2014), ACM, pp. 3645–3654.

[32] Von Ahn, L., Maurer, B., McMillen, C., Abraham, D., and Blum, M. recaptcha: Human-based character recognition via web security measures. *Science 321*, 5895 (2008), 1465–1468.

[33] Warden, P. Smartphone energy consumption. https://petewarden.com/2015/10/08/smartphone-energy-consumption/.

[34] Xiao, Y., Simoens, P., Pillai, P., Ha, K., and Satyanarayanan, M. Lowering the barriers to large-scale mobile crowdsensing. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications* (2013), ACM, pp. 1–9.

[35] Xiong, H., Zhang, D., Chen, G., Wang, L., and Gauthier, V. Crowdtasker: Maximizing coverage quality in piggyback crowdsensing under budget constraint. In *Pervasive Computing and Communications (PerCom), 2015 IEEE International Conference on* (2015), IEEE, pp. 55–62.

[36] Xiong, H., Zhang, D., Chen, G., Wang, L., Gauthier, V., and Barnes, L. E. icrowd: Near-optimal task allocation for piggyback crowdsensing. *IEEE Transactions on Mobile Computing 15*, 8 (2016), 2010–2022.

[37] Zhang, D., Xiong, H., Wang, L., and Chen, G. Crowdrecruiter: selecting participants for piggyback crowdsensing under probabilistic coverage constraint. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (2014), ACM, pp. 703–714.