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

Heng Zhang, Nawanol Theera-Ampornpunt, He Wang, Saurabh Bagchi
Purdue University, West Lafayette, IN, 47907, USA,
[zhan2614,ntheeraa,hw,sbagchi]@purdue.edu

Rajesh K. Panta
AT&T Labs
rpanta@research.att.com

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 hyper-local 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.

CCS Concepts • **Networks → Network architectures**; **Network services**; *Network experimentation*;

Middleware '17, December 11–15, 2017, Las Vegas, NV, USA

ACM ISBN 978-1-4503-4720-4/17/12…$15.00
https://doi.org/10.1145/3135974.3135990

Keywords Mobile Crowdsensing, Smartphone Sensing, Mobile computing, Network architecture

ACM Reference format:
Heng Zhang, Nawanol Theera-Ampornpunt, He Wang, Saurabh Bagchi and Rajesh K. Panta. 2017. Sense-Aid: A Framework for Enabling Network as a Service for Participatory Sensing. In *Proceedings of Middleware '17, Las Vegas, NV, USA, December 11–15, 2017,* 13 pages.
https://doi.org/10.1145/3135974.3135990

1 Introduction

Recent additions of advanced and highly sophisticated sensors to smartphones have led to an emerging category of devices which pushes forward the *Internet of Things* (IoT) era. One area of IoT which has effectively made use of the sensing, computing, and networking capability of mobile devices is *participatory sensing*. Participatory sensing, originally introduced in 2006 by Srivastava *et al.* [4], 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 a large number of participatory sensing applications such as WeatherSignal [14], Pressurenet [8], which has joined Sunshine [28], air quality monitoring [7, 11, 18], noise pollution map [19], urban planning [3, 9, 13], road and traffic condition reporting [2, 15]. There are hundreds of thousands of downloads of these applications, 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.

We did a survey that asked 109 people, from different universities in the 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. 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 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 collects

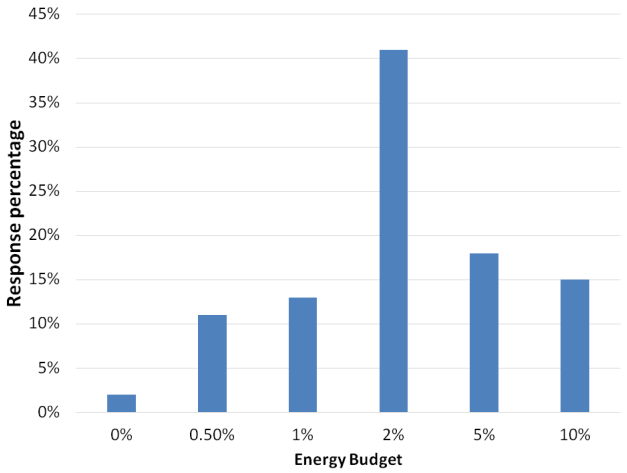

Figure 1. Energy usage expectations from 109 university students. Majority of the users are willing to spend up to 2% of energy in participatory crowdsensing activities.

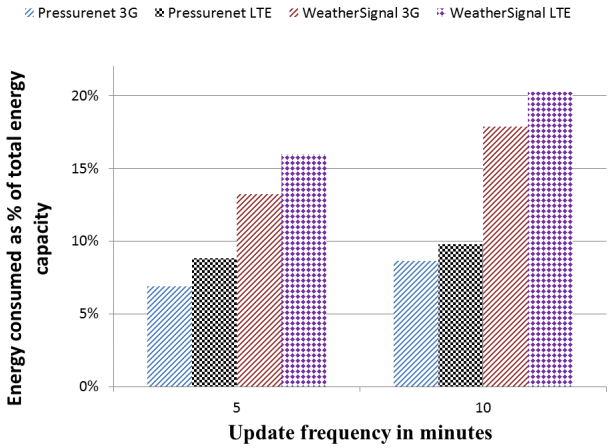

Figure 2. Power consumption case study for two popular weather-related crowdsensing apps. In both apps, the energy budget that the majority of our surveyed users is willing to spend on crowdsensing is exceeded. The duration of the experiment is 4 hours for the 5 minute measurement frequency and 8 hours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

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. In all cases the energy consumption is more than what the majority of the users would expect (2% of the battery capacity). Secondly, LTE energy consumption is higher than 3G and WeatherSignal is more energy hogging than Pressurenet, due 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 without any other activities, it will cost significant amount of energy (close to 10% in two experiments) in 4G LTE network,.

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.

Looking deeply into the sources of high energy drain for crowdsensing applications, we find there are *3 primary opportunities* for optimization. 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 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. According to the study by Huang *et al.* [12], the power consumption for smartphone using 4G LTE network, in transitioning cellular radio from idle state to connected state 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, 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 2 devices in a 500 meters radius circular area. Under a dense presence of mobile devices taking part in the crowdsensing activity, as is likely to happen in an urban or a college campus environment, an orchestrator can determine dynamically which devices to ask for crowdsensing data. This is in sharp contrast to existing crowdsensing frameworks [17, 24, 33, 34], which do not take a network-wide view and therefore cannot orchestrate individual devices.

One previous approach, Piggyback CrowdSensing (PCS) [17], 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 sensed data onto smartphone applications network packets (browsers, phone calls, maps, etc.) that are active and transmits them. Their method predicts app usage patterns and decide whether to upload the sensed data now or wait to piggyback on other normal traffic. However the fundamental limitations of PCS are the difficulty of accurate prediction and the system has to be trained and adapted separately to every single user imposing a burden on deployment. Their results show (Figure 8 from Lane *et al.* [17]) that even with 2 months of training, the saturated accuracy of predicting one app usage is around 40%. For top 5 apps usage prediction, the accuracy is 60%.

Our proposed solution

Previous work from Carrol *et al.* [5] has shown that the majority of power consumption on smartphone can be attributed to the GSM module and the display. From the post by Warden [30], the power consumption values for accelerometer, gyroscope, barometer, GPS, microphone, camera, and LTE Module on Samsung Galaxy S4 are 21 mW, 130 mW, 110 mW, 176 mW, 101 mW, >1000 mW, and 594 mW to 1700 mW respectively (Power consumption for camera depends on user's specific activity.) 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. Therefore, we propose Sense-Aid that leverages the cellular network and creates a service resident in the network for participatory sensing. It provides a middleware that runs on the mobile device and overlays on existing elements at the edge of a cellular network. It also provides an API for developers to develop crowdsensing applications. They can easily specify the requirements for the crowdsensing activity such as the type of sensor and the minimum spatial density of devices.

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.* [12] shows that the radio protocol for 4G cellular devices keeps the radio on for a *tail time* (about 11 seconds for 4G LTE radio stack) after the use of the radio by any application. This tail time provides an ideal opportunity for crowdsensing data to be opportunistically sent out. Second, the network has the information about the presence and location of devices at the granularity of a cellular tower, 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 Sense-Aid middleware elements intelligently schedule sensing and communication tasks on the mobile devices based on their locations, radio states, current battery levels, and total tolerable energy budgets for crowdsensing tasks, while meeting the requirements of the 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 energy 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. 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 applications that save 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.

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 is 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 which are mainly 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. Example apps are cited from Chen *et al.* [6] (for finding parking spaces) and from Talasila *et al.* [27] (for validating location information by using image data).

While participatory sensing has gained ground, there are still barriers to making it mainstream. 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 devices 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. The current crowdsensing applications need frequent measurements and have to do a lot of book-keeping jobs to create a map of the available devices, which is rather cumbersome. As a concrete example, in 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 functionalities. 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 work from Stefania *et al.* [26] for the details of the LTE network).

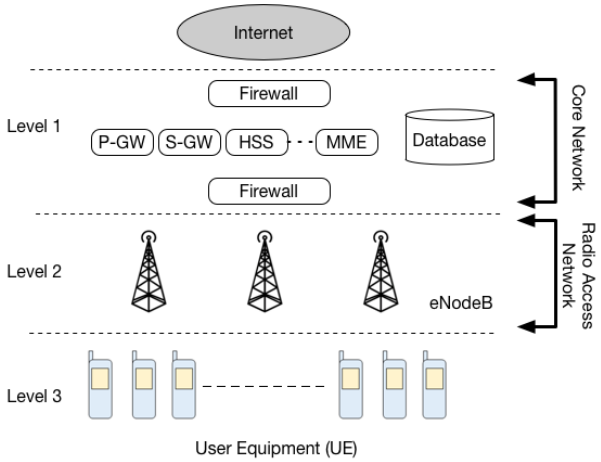

Figure 3. Overview of the 4G LTE architecture showing the network elements. The Core Network (CN) forms Level 1 while the Radio Access Network (RAN) forms Level 2. The User Endpoints (UEs) in Level 3 interact with the eNodeBs. Our solution is embedded between eNodeB and Core Network.

The allocation of radio resources to mobile devices for wireless communications is governed by the Radio Resource Control (RRC) protocol [21]. 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 after the last user data packet is transmitted or received. During 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 transmitted data is usually small (*e.g.* 600 bytes in our user study.) However, for other regular network activities like web browsing, the exchanged data size is much larger. This observation, coupled with the characteristics of the RRC protocol, 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 time opportunistically to send the crowdsensing data, and thus avoiding any expensive radio promotion from IDLE.

3 Sense-Aid Architecture

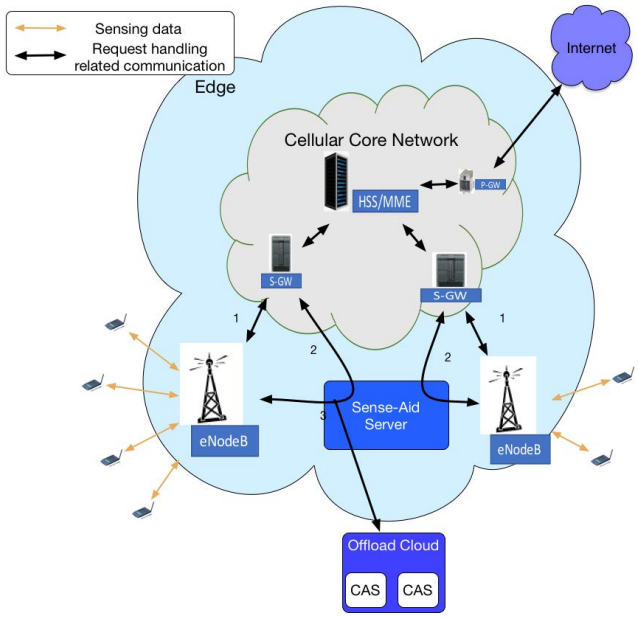

Figure 4. Sense-Aid deployed on a cellular LTE network, showing the three major components: Sense-Aid server, Sense-Aid clients, and crowdsensing application servers (CAS). eNodeBs communicate with the cellular core network through path 2 if its incoming traffic from UEs includes crowdsensing data. Otherwise, eNodeBs choose path 1.

In this section we 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. Tasks are specified by multiple parameters which can be found in Table 1.
- 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 the task, for the devices that have signed up to a crowdsensing campaign and locate within that region, Sense-Aid server will evaluate these devices to check if any of them are capable of processing this crowdsensing task. There are two main reasons that a device becomes unqualified. One is that the device moves out of the region required by the specific task. The second reason is that the submitted crowdsensing data from the device is invalid or the device does not have the sensor required by the task.
- Selected devices - This is the set of devices out of qualified devices, who are selected by server to process crowdsensing tasks. *Sense-Aid* server will only choose minimum number of devices that satisfy spatial density requirement (one parameter in Table 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 locations and states of the devices, handling tasks from the crowdsensing application server, and scheduling crowdsensing tasks on the devices.
- Crowdsensing 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.
- Crowdsensing 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 4 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.

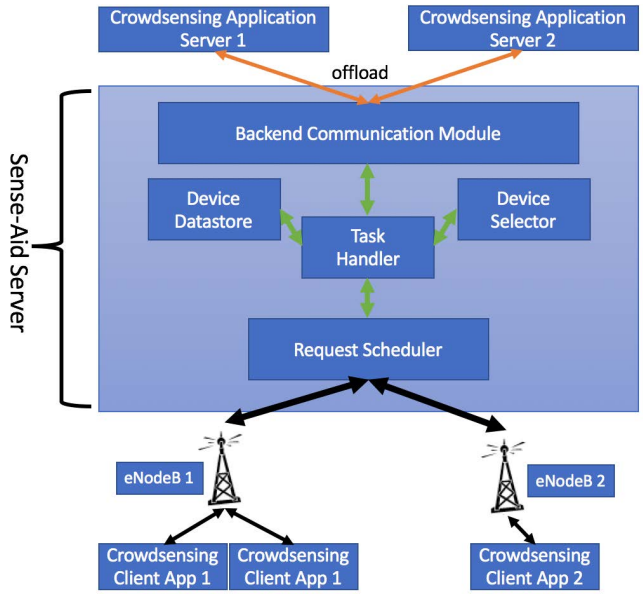

Figure 5. Components of Sense-Aid. Client sensing apps (at the bottom of the figure) are deployed on individual smartphones. Multiple crowdsensing servers can coexist in the infrastructure at the same time.

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 the 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 in the traffic, 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 server crashes.

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. Users 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 1. 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 framework. 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.

Algorithm 1 Sense-Aid server workflow

```
1:  procedure Sense-Aid server
2:      request_select_thread (R, D):   ▷ R - Request run queue; D - Devices datastore
3:      While Server has crowdsensing request to schedule
4:      Pop request r from R.
5:      Get n requested devices for request r.
6:      Get N qualified devices in specific region of t.
7:      if n ≤ N then
8:          Run device selector and choose n devices.
9:          Schedule sensing requests
10:         Send data from devices to crowdsensing server.
11:     else
12:         Move t to wait queue
13:     wait_check_thread (W): ▷ W - Requests stored in wait queue.
14:     Periodically check the request queue;
15:     if then w ← W is satisfiable now
16:         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 sent 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 1 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 cellular tower 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 are 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 the device has been selected for sensing, and the timestamp of the 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 sampling period, Sense-Aid automatically generates multiple tasks to be scheduled. For example, a task with sampling duration of an hour and a sampling period 5 minutes will result in the generation of 12 tasks. Each task is then pushed onto the run queue with appropriate deadline.

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 *the past* crowdsensing tasks, counted since the beginning of some reasonable time interval, say the week; the number of times the device has been used for crowdsensing tasks, again for the same time interval; the current remaining battery level of the device; and the timestamp of the most

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 1. 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.

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. Variables for the four factors are defined as follows.

- 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 the 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 that the sensed value can be 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 that device 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 with the device through the crowdsensing client-side library. Once the crowdsensing task is complete, the data is sent back from the mobile device to the crowdsensing application server, using the crowdsensing server-side library. The 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 of the 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 crowdsensing 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 supported API calls are `task()` (to create a crowdsensing task and specify its properties), `update_task_param()` (to update parameters of an existing 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 development of a crowdsensing application and orchestrating requisite mobile devices is simplified.

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 5) on a proxy server and

we talk about these simplifications here. We set up a proxy server on 64 bit, 12 core machine with 16 GB of RAM and running Ubuntu 16.04 each. We implemented two variants of Sense-Aid, *Sense-Aid Basic* and *Sense-Aid Complete*. In *Sense-Aid Basic*, crowdsensing communication between device and Sense-Aid server happens 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 become available only when it is close to the end of the tail time, which does not allow enough time for the upload and necessitating an elongation of the tail time. For device location information, since we do not have information from the eNodeBs, we use the GPS on smartphones to get the devices' location information. We also implemented a service thread to contact Sense-Aid server with the information about current battery level (dynamic information) and a hash of the IMEI and the energy budget (static information). This service sends these control messages to the proxy server only when the radio tail time is found and thus has a very small network footprint. When we calculate the energy cost for all three frameworks in the user study, we ignore energy consumption for these control messages.

4.1 Infer the tail time

In the user study, one challenge for developing the Sense-Aid client app is how to know the network tail time on the smartphone. There is no programming interface providing developers the radio state information of the device. Since we cannot directly know the LTE radio state on the smartphone, we inferred it with the help of an smartphone app called *tPacketCapture* [25]. This app captures network packets and stores them to file system. In the Sense-Aid client app, we listen to the change of the file directory where *tPacketCapture* stores packet files. Whenever any changes happen to that file directory, we infer that some communication has happened and we reset the tail timer. We developed a tail-time testing app to connect to our server only at a network tail time when user is using the smartphone normally. Figure 6 shows the correctness of the inference of tail time. The X-axis is time in seconds and this data is output by AT&T's ARO tool [22]. 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, 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. Without reseting the tail timer when Sense-Aid traffic sent out during tail time, the total duration of tail time is about 11.5 secs.

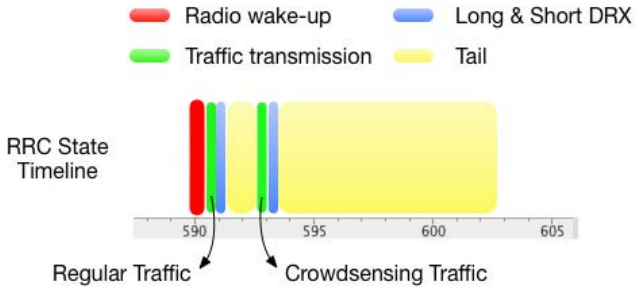

Figure 6. Visualizing network activity, crowdsensing activity, and LTE radio states.

5 Sense-Aid Evaluation

5.1 User Study Experiment Setup

The user study lasted for 6 days from Tuesday to Sunday. The tasks in the user study follow specifications in Table 1 and all tasks need barometer values from 4 location on our campus, *Student Union, EE department , CS department, University Gym*. We have conducted 3 experiments. In each experiment, we have three device sets and each of them have 20 devices carried by 20 students on campus. Each set corresponds to one framework and the devices are running the corresponding client application. We are comparing the energy efficiencies 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 explained in Section 4.

As shown in Table 1, there are multiple parameters for a crowdsensing task. We varied only one parameter and fixed others in each experiment as summarized in Table 2.

Table 2 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 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 6 tests. For easier comparison,

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 100 m, 200 m, 300 m, 400 m 500 m, 1000 m	test duration = 1.5 hrs/test number of tasks/device =1 sampling period = 10 mins spatial density = 2	1: 94.3% (88.7%, 98.3%); 2: 94.9% (90.0%, 98.5%); 3: 79.0% (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 = 500 m	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	Number of tasks per device (3, 5, 10, 15)	test duration = 1.5 hrs/test sampling period = 5 mins spatial density = 3 area radius = 500 m	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 2. Experimental parameter settings in the user study and the summary of energy comparison results.

we add the 2% battery threshold bar to all the energy cost figures, a number that users were comfortable for spending on crowdsensing tasks according to our survey result from Section 1. This is calculated using a nominal 1800 mAh, 3.82 V battery and this threshold 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. From Figure 7, 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 CS department. The differences among *Periodic*, *PCS*, and *Sense-Aid* is due to the unknown mobility patterns of users and not due to any design feature of the frameworks. In Figure 8 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 2. With the help of the device selection algorithm, Sense-Aid can save much energy especially when the spatial density requirement is much 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 benefit 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, by 54.5% and 60.3%, respectively. 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.

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 9. 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 7, 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 10, 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 of devices from the set of qualified devices. The two other frameworks however pick all of the qualified devices for crowdsensing. Their variation with the sampling period is only due to the unknown mobility patterns of the users.

From Figure 11, 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 short 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

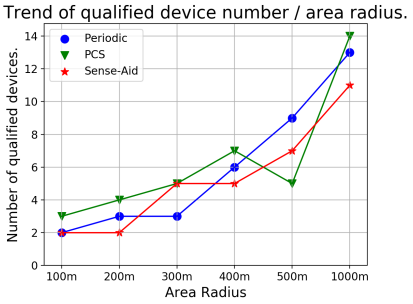

Figure 7. The number of qualified devices at the CS department for different of area radii.

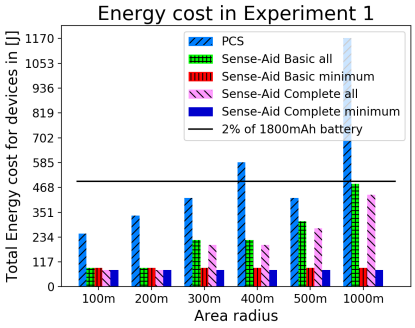

Figure 8. Total energy cost of crowdsensing summed across devices. Periodic is omitted as its energy cost is much higher than other setups. Both variants of Sense-Aid use significantly less energy than PCS.

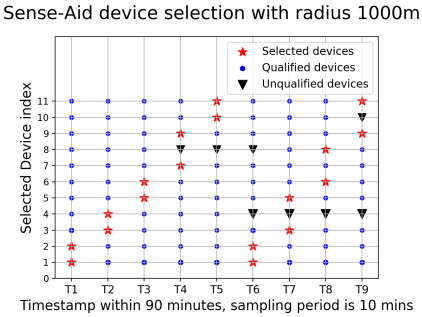

Figure 9. Visualization of Sense-Aid device selection algorithm. Sampling happens once every 10 minutes, and every time server requires data from two devices. Each device is selected either once or twice, showing that the selection is fair.

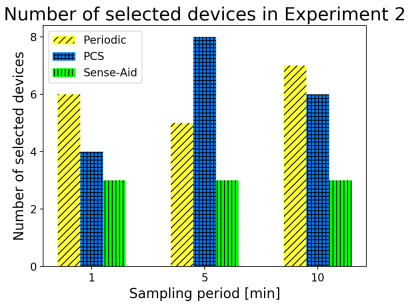

Figure 10. The number of selected devices for each test in Experiment 2. Each request requires minimum of 3 devices.

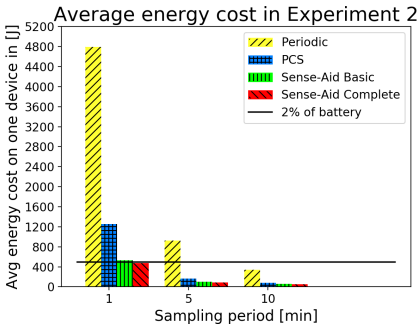

Figure 11. Average energy cost per device with varying sampling period in Experiment 2. *Sense-Aid Basic* and *Complete* cost relatively less energy over *PCS* when the sampling period is low.

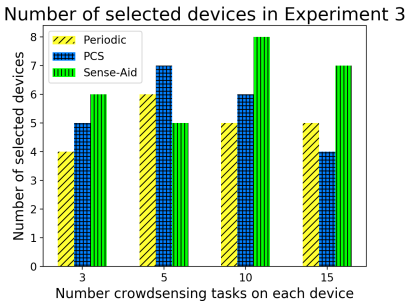

Figure 12. Number of selected devices when there are multiple crowdsensing tasks in Experiment 3. Periodic and PCS will choose all qualified devices while Sense-Aid will intelligently orchestrate among all qualified devices to choose the required number of devices.

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 to 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 number of concurrent tasks on one device

In Experiment 3, 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 12 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

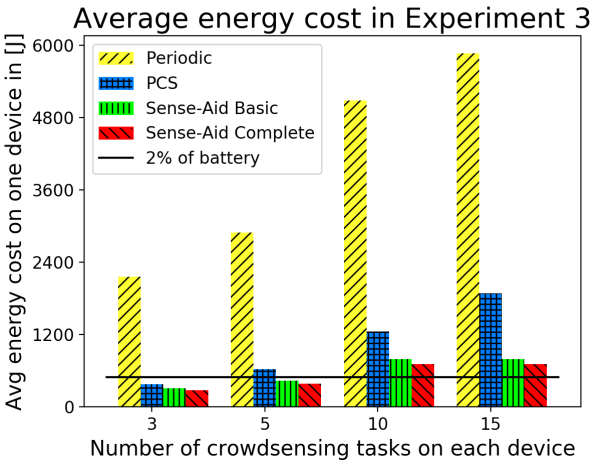

Figure 13. Average energy cost per device when there are multiple crowdsensing tasks in Experiment 3. Higher number of tasks results in higher energy costs for all setups. But *Sense-Aid Basic* and *Complete* cost relatively less energy over *PCS* and *Periodic* when many tasks exist in the system

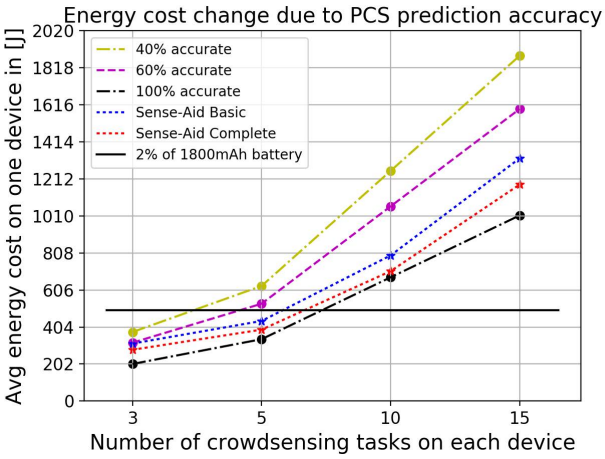

Figure 14. At different prediction accuracies of *PCS*, Sense-Aid over performs the best two practical cases and performs comparable with the ideal case.

per device, Sense-Aid server will choose from the limited number of qualified devices and will schedule multiple tasks on them.

Figure 13 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.

5.5 Effect of PCS prediction accuracy

For the 3 experiments above, the assumption is that the app usage prediction accuracy of *PCS* is 40% which is the saturated accuracy observed in Lane *et al.* [17], 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 14. 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 to use both local and network information (as we do) to achieve energy-efficient crowdsensing.

6 Discussion

We anticipate that one model of deployment of Sense-Aid will be used by the cellular network provider in the coming age of IoT. Through this, Sense-Aid can leverage information that is already collected by the cellular network provider. 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. 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 context information (radio state, battery, location, and etc.) 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 concerns. 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 to be made visible to the crowdsensing application server.

There could exist some errors in Sense-Aid model. One possible source of errors that may crop up is the lack of synchronization among the client devices and the server infrastructure. However, we can use low-duty synchronization protocols such as [16] to avoid this source of error.

7 Related Work

The concept of participatory sensing or crowdsensing has been around for several years since at least 2006 [1]. One of the major concerns for crowdsensing applications is the energy consumption on the mobile devices. 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.

Attempts at making mobile crowdsensing energy-efficient got a major boost with Piggyback Crowdsensing (PCS) [17], 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 [10, 32, 34]. In the work from Zhang *et al.* [34] for example, the system predicts the movement of users 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. 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 the support for developing crowdsensing applications, there are some useful frameworks such as work from Xiao *et al.* [31], Twitch crowdsourcing [29], and Medusa [23]. 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. 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 Ren *et al.* [24] and Meng *et al.* [20]. 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 functions that we found are 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] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. 2008. recaptcha: Human-based character recognition via web security measures. *Science* 321, 5895 (2008), 1465–1468.

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

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

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

[5] A. Carroll and G. Heiser. 2010. An Analysis of Power Consumption in a Smartphone.. In *USENIX annual technical conference*. 1–14.

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

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

[8] N. Gohring. 2013. App Feeds Scientists Atmospheric Data from Thousands of Smartphones. (2013). http://www.technologyreview.com/news/510626/app-feeds-scientists-atmospheric-data-from-thousands-of-smartphones/

[9] S. Greengard. 2011. Following the crowd. *Commun. ACM* 54, 2 (2011), 20–22.

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

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

[12] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. 2012. 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*. ACM, 225–238.

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

[14] S. Johnston. 2013. WeatherSignal: Big Data Meets Forecasting | Guest Blog, Scientific American Blog Network. (Oct. 2013). http://blogs.scientificamerican.com/guest-blog/2013/10/11/weathersignal-big-data-meets-forecasting/

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

[16] J. Koo, R. K. Panta, S. Bagchi, and L. Montestruque. 2009. A tale of two synchronizing clocks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. ACM, 239–252.

[17] N. D. Lane, Y. Chon, L. Zhou, Y. Zhang, F. Li, D. Kim, G. Ding, F. Zhao, and H. Cha. 2013. 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)*. ACM, 1–14.

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

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

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

[21] M. Nohrborg. 2017. LTE. http://www.3gpp.org/LTE. (2017). Accessed: 2017-04.

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

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

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

[25] Tao Software. 2017. tPacketCapture. http://www.taosoftware.co.jp/en/android/packetcapture/. (2017). Accessed: 2017-04.

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

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

[28] Sunshine Technologies. 2017. Sunshine. https://thesunshine.co/. (2017). Accessed: 2017-04.

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

[30] P. Warden. 2015. Smartphone Energy Consumption. https://petewarden.com/2015/10/08/smartphone-energy-consumption/. (2015).

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

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

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

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