iOS crowd–sensing won't hurt a bit!: AWARE Framework and Sustainable Study Guideline for iOS Platform

Yuuki Nishiyama[1], Denzil Ferreira[2], Yusaku Eigen[3], Wataru Sasaki[3], Tadashi Okoshi[3], Jin Nakazawa[3], Anind K. Dey[4], and Kaoru Sezaki[1]

[1]The University of Tokyo, Japan
[2]University of Oulu, Finland
[3]Keio University, Japan
[4]University of Washington, USA

Abstract. The latest smartphones have advanced sensors that allow us to recognize human and environmental contexts. They operate primarily on Android and iOS, and can be used as sensing platforms for research in various fields owing to their ubiquity in society. Mobile sensing frameworks help to manage these sensors easily. However, Android and iOS are constructed following different policies, requiring developers and researchers to consider framework differences during research planning, application development, and data collection phases to ensure sustainable data collection. In particular, iOS imposes strict regulations on background data collection and application distribution. In this study, we design, implement, and evaluate a mobile sensing framework for iOS, namely *AWARE-iOS*, which is an iOS version of the *AWARE Framework*. Our performance evaluations and case studies measured over a duration of 288 h on four types of devices, show the risks of continuous data collection in the background and explore optimal practical sensor settings for improved data collection. Based on these results, we develop guidelines for sustainable data collection on iOS.

Keywords: Mobile Sensing Framework, Sustainable Sensing, Guideline, iOS, Data Collection Rate

1 Introduction

Mobile crowd sensing (MCS) is a research method to understand human activities on individual, group, and community levels using data collected by smartphones, which have become ubiquitous worldwide [1]. MCS-based research is being conducted on various scales and terms, including research in fields like computer science, social science, and public health [2–4].

The developments in mobile sensing frameworks [5–7] have accelerated these kinds of MCS research. Such frameworks are capable of dramatically reducing the developmental and maintenance costs of sensing software. For instance, Ferreira et al. developed the AWARE framework [5], which allows the collection of

hardware-based (e.g., accelerometer, GPS, and barometer), software-based (e.g., battery, screen, and network), and human-based (Experience Sampling Method (ESM)) data on the Android platform. In addition, AWARE is designed to manage large-scale human subject studies remotely through a web dashboard.

According to market research [8], Android systems account for 71.94% of the market share, and iOS accounts for 18.89% of the global market share. However, in certain countries (e.g., Japan, the United States of America, the United Kingdom), the market share of iOS is bigger than or comparable to that of Android. For example, in Japan, iOS accounts for 72.45% of the market share, compared to Android's 26.43%. If iOS accounts for more than 50% of the market share in a specific area, then collecting data from both iOS and Android in that region becomes imperative to expand data collection opportunities. Several mobile sensing tools [6, 7] have been developed for iOS, but planning and managing a sustainable MCS-based study using iOS remains a challenge.

In this paper, we propose a mobile crowd sensing framework for iOS (namely AWARE-iOS) based on the AWARE Framework [5] which is a mobile sensing framework for Android. AWARE-iOS allows us to collect sensor data sustainably with a few lines of code or through a published client application. Moreover, it provides options for controlling sensors, storage, remote server-connection, and web dashboard to optimize each research purpose. This proposed framework has already been used in various studies [9–11]. Through performance evaluations and case-studies of AWARE-iOS, we demonstrate the potential risks of in-the-wild mobile sensing on iOS and explore a configuration which allows sustainable data collection on iOS. Finally, based on these results, we propose a guideline for realizing sustainable MCS studies using iOS.

The contributions of this study are as follows:

- This study studies the regulations regarding background sensing on iOS, and demonstrates the advantages and disadvantages of iOS in the context of mobile sensing.
- It designs and develops an open-source MCS framework for iOS based on the regulations on iOS.
- It performs basic performance evaluations and in-the-wild studies, demonstrating the risks and modes of their prevention in background sensing on iOS.
- Based on these results, it proposes guidelines for a MCS study using iOS.

2 Mobile Crowd Sensing Studies and Frameworks

MCS-based research in Ubiquitous Computing, Human-Computer Interaction, and/or Public Health collect information regarding human activities using mobile devices to gain an understanding of activities in the daily lives of people. Rachuri et al. [4] developed a mobile sensing platform for social psychology studies that operated using mobile phones, called EmotionSense. Their developed system can detect individual emotions and verbal and proximal interactions between social group members from the sensors (e.g., microphone, GPS,

Table 1. Existing Mobile Sensing Frameworks for iOS

Name	OS	Client	Library	Server	Survey	OSS
AWARE [5]	**iOS**+Android	✓	✓	✓	✓	Apache 2.0
Sensus [7]	iOS+Android	✓		✓	✓	Apache 2.0
mEMA [21]	iOS+Android	✓		✓	✓	
SensingKit [6]	iOS+Android	✓	✓			LGPL-3.0
StudentLife [3]	iOS+Android	✓				

and accelerometer) on off-the-shelf smartphones. Similarly, StudentLife [3] measured hidden stress and strain in the lives of students based on data gathered by smartphones. In particular, they focused on detecting the day–to–day and week–by–week impacts of workload on stress, sleep, activity, mood, sociability, mental well–being, and academic performances of students. In SmartGPA [12], Wang et al., predicted the academic performance of participants based on the dataset of StudentLife. In addition to passive sensor data gathered by a smartphone, these research projects collect human subject data using a questionnaire on the smartphone. The questionnaire is called ESM and/or Ecological Momentary Assessment (EMA) [13–15]. Participants who have enrolled in a study record temporal thinking and their emotions at the same moment on memos or digital devices. For example, in StudentLife [3] project, they recorded participants' subjective data (e.g., stress-level, social-pressure, and sleep quality.) using EMA during their study.

Various mobile sensing frameworks for Android platform [4, 5, 16–20] have been proposed and used real studies. For instance, AWARE Framework [5] is an open-source mobile sensing framework, and that allows us to access hardware-, software-, and human-based sensor easily. AWARE is designed to handle a large–scale MCS study remotely through a web dashboard, flexibly extend or import the client and library for satisfying requirements of each study.

Table 1 shows the iOS supported frameworks [3, 6, 7, 21] and functions of each framework. Though stable sensing is an important factor in a mobile sensing framework, these frameworks have not been satisfactorily evaluated the data collection performance in the real condition. SensingKit [6] has evaluated the battery consumption in simple sensor conditions, however, the performance might fluctuate by the sensor and device settings in the real situation. Moreover, Xiong et al. [7] conducted a case study using Sensus, however, the evaluation does not illustrate the stability of data collection during the study. While providing data loss risks and prevention methods assume helps us to plan and manage an MCS study, these risks are not clear and a guideline for sustainable MCS study using iOS platform does not exist.

3 Hindrances of Sustainable Mobile Crowd Sensing

A smartphone possessing multiple sensors is a powerful sensing tool to track the daily lives of people. At the same time, they are frequently used in daily life for multiple purposes such as web-browsing, emailing, gaming, camera, etc.

The Operating System (OS) tries to minimize resource usage for each application that is running as a background process to improve User Experience (UX), because each application on a smartphone uses shared resources like battery, CPU, and storage. On recent versions of OS, an application that does not follow the regulations of the OS is killed or suspended automatically by the OS. In particular, iOS imposes strict regulations on background sensing. This section describes the regulations of iOS regarding MCS-based studies.

3.1 OS Diversity

As a sensing platform, iOS and Android exhibit different characteristics, and they have been tabulated. While Android can access various sensors and distribute an application flexibly, the maintenance costs of the application on it are high because lots of devices are released every year from different manufacturers all over the world, and each Android OS is customized by the corresponding manufacturer.

On the other hand, iOS suffers from various limitations to accessing Application Programming Interfaces (APIs) and distribution methods of an iOS app are limited. However, the maintenance costs are low in its case because iOS devices are released exclusively by Apple. In addition, more than 90% of iOS users use a newer version of OS (iOS 12 or 13[1] on January 10, 2020) while only 38.7% of Android users use Android 8(Oreo) or 9(Pie)[2].

3.2 Resource Limitation

Minimizing resource usage is a common challenge on an off-the-shelf smartphone to improve UX. For example, Low-Power Mode on iOS reduces battery consumption by restricting CPU performance and background activities, especially network connections.

To evaluate storage consumption, we checked the available storage space for 54 undergraduate male students, who possess 16 GB (N = 10), 32 GB (N = 5), or 64 GB (N = 40) storage models, at Keio University, Japan in 2016. As recorded in Figure 1, 80% of 16 and 32 GB models had less than 1.5 GB free space. The data size of contents increase with time (e.g., Full HD to 4K video), and thus, in this context, available storage is also a significant risk for sustainable data collection.

In addition, an application is barred from using more than 80% of the CPU for more than 60 s while running as a background operation, according to Apple

[1] `https://developer.apple.com/support/app-store/`
[2] `https://developer.android.com/about/dashboards/index.html`

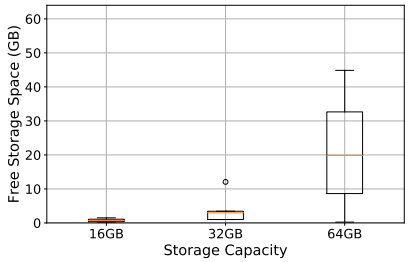


Fig. 1. Free storage space for each specification (16, 32, and 64 GB)

document [3]. If an application oversteps this regulation, the iOS shuts down the application process immediately.

3.3 Restricted Background Sensing on iOS

An iOS app that tracks human activities courteously in the daily lives of people needs to run even if the application is not running as a foreground process. As a lifecycle[4], apps on iOS may have one of the following five statuses: *Not Running*, *Inactive*, *Active*, *Background*, and *Suspended*. If an application is allowed to run in the background, it can continue to run after closing the app. On iOS, if the application serves a function from the following list, it can be run in the background.

- Location updates
- Remote Notifications – Voice over IP (VoIP)
- Audio, AirPlay, and Picture in Picture
- External accessory communication
- Uses Bluetooth LE accessories
- Acts as a Bluetooth LE Accessories
- Background Fetch
- Background Processing

"Location Updates" and "Remote Notifications" functions are commonly used in most applications, and, therefore, such apps easily pass the review by Apple and do not flout iOS regulations. Silent Push Notification (SPN) is a part of remote notification that can be used to send data in JSON format to smartphones from the server-side without any alert and sound to the recipient two or three times per hour[5]. The notification's priority is low, and the system does not guarantee its delivery.

[3] `https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/WorkLessInTheBackground.html`

[4] `https://developer.apple.com/documentation/uikit/app_and_environment/managing_your_app_s_life_cycle`

[5] `https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/pushing_background_updates_to_your_app`

Table 2. Application distribution methods

Method	Device Registration	Deployment Platform	Review by Apple	Estimated Review Time	Account Fee	Software Update	Private API	Study Scale	Build Expiration
AppStore	NO	AppStore	YES	1-7 days	99$	Automatic or Manual	NO	1~∞	Never
TestFlight (External)	NO	TestFlight or URL	YES	1-7 days	99$	Manual	NO	1~10000	90 days
Apple Developer Enterprise Program	NO	URL	NO	Immediate	299$	Manual	NO	1~100	Never
TestFlight (Internal)	YES	TestFlight	NO	Immediate	99$	Manual	NO	1~100	90 days
DeployGate (AdHoc)	YES	DeployGate	NO	Immediate	99$	Manual	YES	1~100	Never
Xcode	YES	Xcode (PC)	NO	Immediate	99$	Manual	YES	1~100	Never

3.4 Limitation of Application Distribution

Table 2 depicts distribution methods for an application to an iOS user. An application developer and researcher can choose the best way to provide an application for their study from the list.

During the developmental phase, a developed application can be installed on a device that is registered as an Apple Developer Account by Xcode (which is an IDE for Xcode). However, each developer account can register a maximum of only 100 devices. By using AdHoc distribution (like DeployGate), we can deploy the application by URL, but the iOS device is required to be registered to an Apple developer account before building the application. AppStore is a digital distribution platform for iOS applications managed by Apple. iOS devices can download and update applications through the platform. However, releasing an application on AppStore needs to pass a review by Apple. The review is conducted under AppStore review guideline[6], and generally takes a few days.

4 AWARE Framework for iOS

AWARE Framework is an open-source mobile sensing framework for Android which has been developed by Ferreira et al. [5]. The AWARE Framework is composed of an AWARE-Server (hereafter referred to as *AWARE-Server*), which is a common LAMP (Linux+Apache+MySQL+PHP) server, and -Client (hereafter referred to as *AWARE-Android*), which is an Android application.

In this study, we design and implement an AWARE Framework for iOS, namely *AWARE-iOS*, that is compatible with *AWARE-Android* and *-Server*. Figure 2 presents an overview of *AWARE-iOS*. Its architecture is inspired by *AWARE-Android*. *AWARE-iOS* is composed of a Library (see Section 4.1) and a User Interface (UI) module (see Section 4.2) to improve reusability.

Compared to *AWARE-Android* [5], iOS does not allow connections to MQTT for a background process. Thus, instead of MQTT on Android, *AWARE-iOS* uses

[6] `https://developer.apple.com/app-store/review/guidelines/`

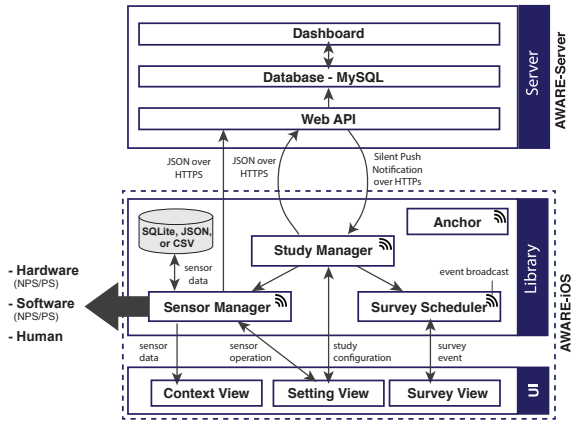


Fig. 2. Overview of AWARE-iOS

SPN[7] to transmit data (in JSON format) from the server-side. An SPN can be included as a payload whose maximum size is 4096 bytes[8].

4.1 Core Library

The Library module comprises three sub-modules to manage study, sensors, and survey schedules (see Section 4.1). This section describes these sub-modules with sample codes.

Table 3 depicts the supported sensors on AWARE-iOS. The number of supported sensors are less than that of AWARE-Android due to API limitations (e.g., light, proximity, and temperature), but all of the accessible sensors on iOS are supported, to the best of our knowledge [3, 6, 7]. The source code is written in *Objective-C* and supports iOS 10 or later. *AWARE-iOS* is published under Apache License 2.0 on GitHub[9], similar to *AWARE-Android*.

Library Instruction The core library is released on CocoaPods[10], which is a library management tool for Xcode project. Just to write `pod 'AWAREFramework'` into Podfile and run `pod install`, a developer and add AWARE-iOS into Xcode project. The developer can use the functions of AWARE-iOS by importing the library into the application by `import AWAREFramework`.

The supported sensors (as depicted in Table 3) can be categorized into the following two types:

[7] `https://github.com/tetujin/aware-push`
[8] `https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/generating_a_remote_notification`
[9] `https://github.com/tetujin/AWAREFramework-iOS`
[10] `https://cocoapods.org/`

Table 3. Supported sensors on AWARE-iOS comparing with -Android

Sensors	Support	Type	Source	Permission
Accelerometer	✓✓	Periodical	Hardware	NPS
Gravity	✓✓	Periodical	Hardware	NPS
Linear Accelerometer	✓✓	Periodical	Hardware	NPS
Gyroscope	✓✓	Periodical	Hardware	NPS
Rotation	✓✓	Periodical	Hardware	NPS
Magnetometer	✓✓	Periodical	Hardware	NPS
Wi-Fi	✓	Periodical	Hardware	NPS
Location	✓✓	Periodical	Hardware	PS
Barometer	✓✓	Periodical	Hardware	PS
Ambient Noise	✓✓	Periodical	Hardware	PS
Bluetooth	✓	Periodical	Hardware	PS
Telephony		Periodical	Hardware	—
Light		Periodical	Hardware	—
Temperature		Periodical	Hardware	—
Processor	✓✓	Periodical	Software	NPS
Timezone	✓✓	Periodical	Software	NPS
Activity Recognition	✓✓	Periodical	Software(Auto)	PS
Pedometer	✓✓	Periodical	Software(Auto)	PS
HealthKit *	✓✓	Periodical	Software(Auto)	PS
Weather	✓✓	Periodical	Software(Web API)	NPS
Fitbit	✓✓	Periodical	Software(Web API)	NPS
Proximity	✓	Event	Hardware	NPS
Screen	✓✓	Event	Software	NPS
Battery	✓✓	Event	Software	NPS
Communication	✓	Event	Software	NPS
Installations		Event	Software	—
Applications		Event	Software	—
Keyboard	✓	Event	Software	PS
ESM	✓	Event	Human	NPS

✓✓ Supported
✓ Partly Supported
* Only iOS

– *Non-permission-imposed Sensors (NPS)*: An *NPS* does not require permission for an app to access it. For example, accelerometer, gyroscope, Wi-Fi, and processor are *NPS* on iOS.
– *Permission-imposed Sensor (PS)*: A *PS* requires permission for an app to access it, and might need to describe the reason of requirement during the application review by Apple. On iOS, location, microphone, motion activity, Bluetooth, calendar, contact, and HealthKit are considered to be *PS*.

All applications on *AppStore* or *TestFlight* need to pass the review by Apple. Apple developer guideline demands minimizing the use of *PS* owing to security concerns, and therefore, a developer should focus on minimizing sensor usage. *PS* can be installed separately by using `subpod` function on CococoaPods as shown in Listing 1.1.

```
1 pod 'AWAREFramework'
2 pod 'AWAREFramework/Microphone'
3 pod 'AWAREFramework/MotionActivity'
4 pod 'AWAREFramework/Bluetooth'
5 pod 'AWAREFramework/Calendar'
6 pod 'AWAREFramework/Contact'
7 pod 'AWAREFramework/HealthKit'
```

Listing 1.1. Podfile

Anchor To collect data continuously, *AWARE-iOS* maintains a location sensor perpetually activated in the background as an anchor. The iOS location sensor has the following six levels of accuracy: *best*, *best for navigation*, *nearest ten meters*, *hundred meters*, *kilometer*, and *three kilometers*. By default, AWARE-iOS uses a location sensor that is accurate up to *three kilometers*. The battery impact of the low-accuracy location sensor is considered in our basic performance evaluation (see Section 5.2).

Sensor Manager Listing 1.2 depicts a sample code for using the accelerometer sensor in the background. During launch, the app (1) imports AWARE Framework into the project, and (2) requests permission for background sensing from the user. After authorization, `AWARECore` can be activated for background sensing (3). Each sensor can be initialized by (4). `AWARECore`, `AWAREStudy` and `AWARESensorManager` are designed as a singleton class; these classes can be accessed from anywhere in the app. `AWAREStudy` handles study configurations, such as a remote server URL, sensor activation if a study exists, and remote DB sync. In addition, (5) each sensor event can be handled by the `-setSensorEventHandler` method. (6) A sensor instance can be retained on the memory by adding it to `AWARESensorManager`, and can be accessed from any place in the app through the manager. The collected data are saved in SQLite, JSON, or CSV based local storage temporarily.

```
/// (1) import 'AWAREFramework'
import AWAREFramework
/// (2) request permission
let core    = AWARECore.shared()
core.requestPermissionForBackgroundSensing{ (status) in
    /// (3) activate AWARECore
    core.activate()
    /// (4) initialize sensor(s)
    let study = AWAREStudy.shared()
    let acc = Accelerometer(awareStudy: study, dbType: AwareDBTypeSQLite)
    /// (5) handle sensor events (option)
    acc.setSensorEventHandler { (sensor, data) in
      // YOUR CODE
    }
    acc.startSensor()
    /// (6) add the sensor(s) into the sensor manager
    AWARESensorManager.shared().add(acc)
}
```

Listing 1.2. A sample code for activating a sensor

Survey Manager As in the case of *AWARE-Android*, *AWARE-iOS* supports Mobile ESM as a survey function. On *AWARE-iOS*, the following types of survey are supported: *Text, Radio, Checkbox, Likert Scale, Quick Answer, Scale, Date-Time, PAM [22], Numeric, Web, Date, Time, Clock, Picture, Audio, and Video* (screenshots can be found here[11]).

[11] `https://github.com/tetujin/AWAREFramework-iOS/tree/master/Screenshots/esms`

Listing 1.3 depicts a sample code for setting a build-in survey with a *Radio* format. The `ESMScheduleManager` manages the entire ESM schedule that is instanced by `ESMScehdule`. An `ESMScehdule` instance possesses parameters for scheduling surveys as notification titles, expiration thresholds, times for notification, and survey items. The survey items support the 16 types of the surveys mentioned above. Moreover, `ESMScheduleManager` can be set up using a JSON file which includes configuration of survey schedule from a connected *AWARE-Server*.

```
let schedule = ESMSchedule()
schedule.notificationTitle   = "Notification Title"
schedule.expirationThreshold = 60
schedule.fireHours           = [9, 15, 21]

let radio = ESMItem(asRadioESMWithTrigger: "trigger", radioItems:
    ["A","B","C"])
radio.setTitle("Title")
radio.setInstructions("Instructions")
schdule.addESM(radio)

ESMScheduleManager.shared().add(schdule)
```

Listing 1.3. Generating an ESM schedule

In addition to the fixed schedule survey, *AWARE-iOS* can implement a context-based survey by using sensors and the `setSensorEventHandler` method. For example, *AWARE-iOS* is able to send a survey to the user when a user finishes a phone call.

Study Manager *AWARE-iOS* can upload sensor data to a connected remote cloud server if the user has signed up for a study. Studies can be managed on a dashboard on the *AWARE-Server* (accessible at `https://api.awareframework.com`). The *AWARE-Server* is also an Open Source Project (under Apache 2.0), and thus each researcher can host an AWARE server on the researcher's hosting server, such as Google Cloud Platform, Amazon Web Services, and Microsoft Azure. Listing 1.4 depicts a sample code for signing-up to a study by using a study URL, and for activating sensors with a study configuration on *AWARE-Server*

```
let url = "https://aware.server_url.com/STUDY_ID/PASS"
study.join(withURL: url) { (settings, studyState, error) in
    let manager = AWARESensorManager.shared()
    manager.addSensors(with: study)
    manager.startAllSensors()
}
```

Listing 1.4. Joining a study using a study URL

4.2 User Interface

The AWARE Client iOS is a sample application that serves as the user interface. Users access *AWARE-iOS* APIs through the application. The application is writ-

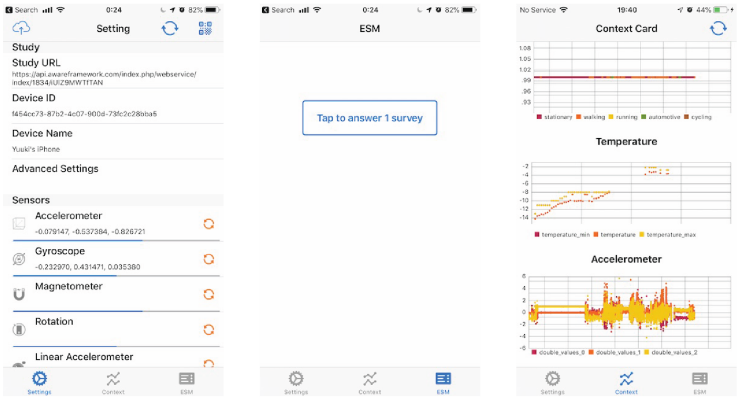


Fig. 3. Screenshots of the AWARE Client on iOS

ten in *Swift* and has been released on AppStore[12] and GitHub[13] under Apache License 2.0. As depicted in Figure 3, the application has three views: settings, ESM, and context cards. A user can modify study settings on the settings view. ESM displays a survey if the survey has been delivered to the app, and context cards show collected sensor data on a card-like view. These functions are changeable through the *AWARE-Server* and on the client.

5 Evaluation and Case Studies

In this section, we evaluate (1) basic performance of SQLite, JSON and CSV-based local storage that is supported on *AWARE-iOS*, and (2) battery consumption of each sensor and configuration. Finally, (3) we conduct case studies to measure data collection rates and coverages in each case study.

5.1 Data Insertion and Fetching Performance

Motion sensors such as accelerometers, gyroscopes, and magnetic-fields are commonly used to detect human and mobile movement. While such sensors are implemented on various devices and are usable for multiple purposes, raw sensor data are collected at high frequencies (1—100 Hz), and this generates a significant amount of data on a smartphone.

Table 4 depicts the table format on the SQLite database on *AWARE-iOS*, which is a sample table format available on AWARE-Android. Accelerometer data are saved with a timestamp, device ID, values (including X, Y, and Z axes), and label. Table 5 depicts the estimated data sizes gathered by the accelerometer at 5 Hz and 50 Hz over an hour, day, week, and month (4 weeks). The total data size exceeds 1 GB in a week at 50 Hz and in a month at 5 Hz and 50 Hz.

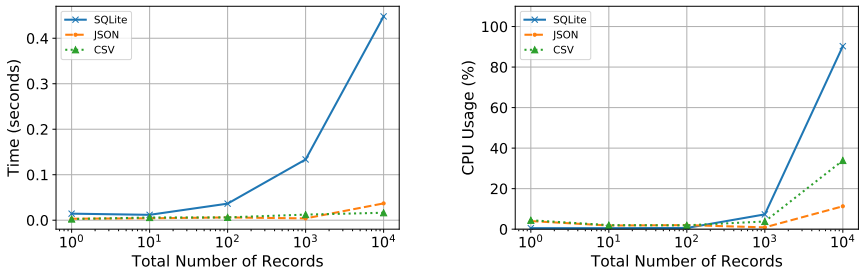
[12] `https://apps.apple.com/app/aware-client-v2/id1455986181`
[13] `https://github.com/tetujin/aware-client-ios-v2`

Table 4. Table format of Accelerometer sensor

Column	Type	Bytes
timestamp	Double	8
device_id	String	32
x	Double	8
y	Double	8
z	Double	8
accuracy	Int32	4
label	String	32

Table 5. Expected data size

Fre.	Term	Rows	Estimated Size
5Hz	Hour	18,000	1.72MB
	Day	432,000	41.20MB
	Week	3,024,000	288.39MB
	Month	12,096,000	1.13GB
50Hz	Hour	180,000	17.17MB
	Day	4,230,000	403.40MB
	Week	30,240,000	2.82GB
	Month	120,960,000	11.27GB



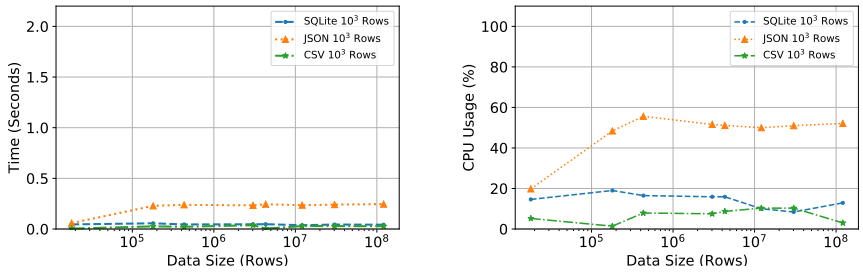
(a) Insert time (b) CPU usage

Fig. 4. Resource usage during a data insert operation

Insert Performance To evaluate insert performance, we measured the time taken and CPU usage during data insertion of different numbers of records (10^0, 10^1, 10^2, 10^3, and 10^4). Generally, accessing a number of files over a period of time influences battery consumption, and thus file access should be minimized. During this evaluation, we explore the best batch size based on CPU usage and execution times for inserting data into each type of local database (SQLite, JSON, and CSV). This evaluation is conducted on the iPhone 7 (iOS 13.2).

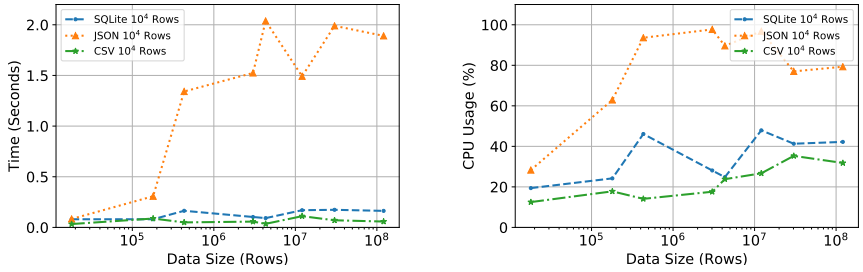
As depicted in Figure 4, data insertion time exceeds 10^3 units. CPU usage (depicted in Figure 4(b)) also exhibits an almost identical transition with time, except the insertion of 10^3 records per insertion. This result indicates that 10^3 is approximately the correct practical batch size for data insertion.

Fetch Performance As indicated by the data in Table 5, the amount of stored data continues to increase unless it is cleared. In this evaluation, we measure the time taken and the CPU usage during the process of fetching records from SQLite-, JSON-, and CSV- based storages. As a simulation, we prepared SQLite



(a) Fetch time (b) CPU usage

Fig. 5. Resource usage during a data (10^3 rows) fetch operation



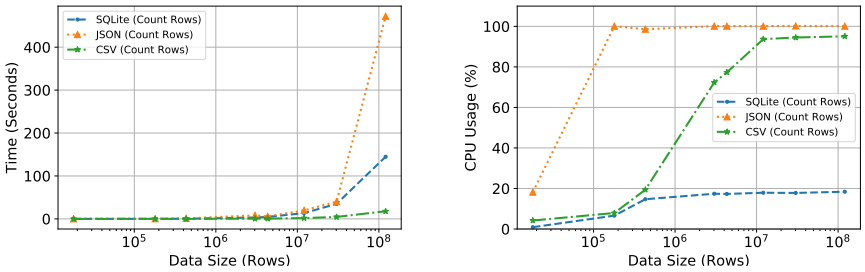
(a) Fetch time (b) CPU usage

Fig. 6. Resource usage during a data (10^4 rows) fetch operation

databases of multiple sizes based on the data presented in Table 5 and measured the performance of each. The fetch request is one of the following three requests: (1) fetch 10^3 records, (2) fetch 10^4 records, and (3) count number of stored records.

As depicted in Figure 5(b) and 6(b), CPU usage is observed to increase sharply when the number of stored records exceeds 10^6. CPU usage decreases once the number exceeds 10^7, but, in this case, each fetch takes more than 10 s (as depicted in Figure 5(a) and 6(a)); 10^6 rows are almost equal with data of 3-days by 5 Hz or 0.5-day by 50 Hz.

As depicted in Figure 7, operations that read all data in the storage (e.g., counting the number of stored records) significantly affects time consumption and CPU usage. The time taken for the counting process for JSON and SQLite increased to more than 30 s when the size of stored data was greater than or equal to 10^7. However, the CPU usage for SQLite was less than 20% even when the size of stored data was increased. On the other hand, the CPU usage for JSON and CSV increased sharply with increase in the stored data size.



(a) Fetch time (b) CPU usage

Fig. 7. Resource usage during a counting operation

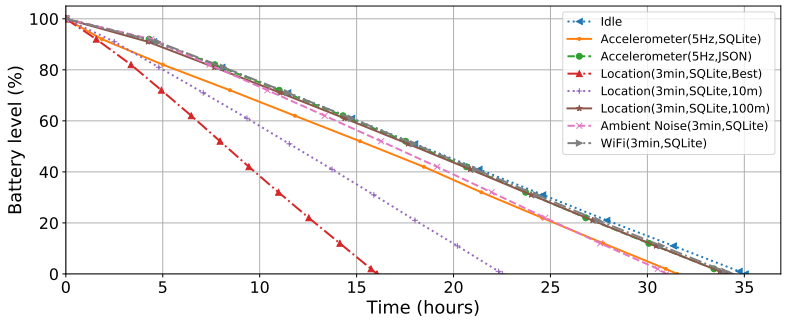


Fig. 8. Battery consumption of *AWARE-iOS*

5.2 Battery Impact

We measured the battery consumption while running *AWARE-iOS* on an iPhone X running iOS 12. All data were erased from the device and the manufacturer's default settings were restored[6]. No other third-party applications were installed or allowed to run in the background. The device was set to flight mode, and wifi, Bluetooth, and cellular connectivities were disabled. Finally, the background app refresh setting was turned off and the low power mode was turned on.

Figure 8 depicts the battery consumption of AWARE-iOS under the aforementioned conditions. In this experiment, we evaluated battery consumption of the library while using the four types of sensors (viz., accelerometer, location, Wi-Fi, and ambient noise from periodical hardware sensor, as recorded in Table 3), which use different storage systems (SQLite or JSON), sampling rates, and accuracies of sensors if required. In the idle condition, a smartphone can be run for a maximum of 35.05 h.

SQLite vs JSON Battery consumption of the accelerometer that used JSON-based storage was observed to be lesser than that using SQLite-based storage

Table 6. Device specification

Name	Device	OS	Storage
ideal (Ideal Condition)	iPhone 11	13.2	64GB
non-c (Non-control Condition)	iPhone 6 Plus	11.4.1	64GB
heavy (Heavy Use Condition)	iPhone 6 Plus	12.3	64GB
low-p (Low-Power Mode Condition)	iPhone 11	13.2	128GB

over nearly 5 h. As described in Section 5.1, the data insertion time corresponding to the JSON format is lower than that corresponding to SQLite, and the overall CPU usage is lower for the JSON format than for SQLite as well.

Effects of Location Accuracy We also compared battery consumptions during the use of three levels of accuracy for the location sensors — best for navigation (0 m), 10 m, and 100 m. The battery lives corresponding to the less accurate settings (34.05 and 22.52 h, respectively) were higher than that corresponding to the most accurate setting (16.05 h).

5.3 Case Study

As described in Section 3.3, iOS imposes several restrictions on apps that run continuous background sensing. In this section, we measure the data collection rates over three days in the following four cases.

BL A user does not interact with the data collection application during the term as a baseline.
ESM A user needs to open the application three times a day during the term.
SPN A user does not interact with the application. Instead, a remote-server sends a silent push notification to the device every 30 min for restart sensors.
ESM+SPN A user needs to carry out the tasks of ESM and SPN.

In all of these cases, we used AWARE Client iOS (described in detail in Section 4.2) with the following sensors: Accelerometer (5 Hz), location (accurate to 3 min and 100 m), weather (accurate to 10 min), pedometer (accurate to 10 min), screen, and battery. The collected data were saved in SQLite and synced with a remote server when the device had access to Wi-Fi and the battery was charged. The results corresponding to the four devices have been recorded in Table 6. The participants carried and used these devices in their daily lives. *ideal* represents the initial setting on the iPhone in which the AWARE Client iOS was installed. *low-p* represents the statistics measured when the phone was in Low-Power Mode during the entire term of investigation. *heavy* represents the case of an iPhone that received memory warnings more than 10 times a day. The memory warning appeared when the user used applications (e.g., YouTube, Instagram, and Camera) that consume significant memory.

Rate of Data Collection Status Figure 9 depicts the rates of data collection in each of the aforementioned cases. We defined the following three data collection statuses: `Comp`, `Incomp`, and `Loss`. For example, location data are collected every 3 min; thus, ideally, 20 records are collected in 1 h. In this case, records = 20 is categorized as `Comp`, 0<records<20 is categorized as `Incomp`, and records = 0 is categorized as `Loss`.

As depicted in Figure 9(a), corresponding to BL, though *ideal* and *non-c* almost exclusively exhibited the `Comp` status, more than half of the statuses corresponding to *heavy* and *low-p* were observed to be `Loss`. ESM, SPN, and ESM+SPN exhibited nearly 100% `Comp` in all devices except in *heavy* running ESM and SPN; 86.11% of the statuses in *heavy* with SPN condition were `Incomp`. The reason behind this is after receiving a silent push notification once every 30 min, the application was able to run for 30 s after which the device suspended it Thus, As recorded in Figure 5.3, data were collected only twice in 1 h on average.

Low power mode restricts network connections when applications are running in the background. Figure 9(b) depicts the number of records each hour corresponding to this mode. As is evident from the figure, if the low power mode is turned on, *low-p* loses weather information (collected by a Web API of OpenWeatherMap (`https://openweathermap.org/`), even if the application is running in the background.

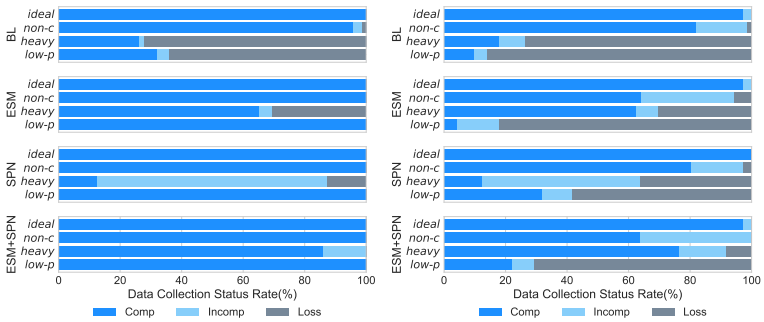
In all cases, the pedometer sensor (Figure 9(c)) collected ideal amounts of data. More recent versions of iPhone 5S and iOS devices run on Apple M-series coprocessors that collect, process, and store sensor data motions even if the iOS device is asleep. This aids the safe collection of data.

Coverage of Data Collection Each cell in Figure 5.3 depicts the total amount of location data collected each hour. *Ideal* and *non-c* phones had collected 100% of the location data under all four conditions. However, *heavy* and *low-c* devices had suspended the application after 12:00 hours on Day 1 in BL. However, in the cases of ESM and ESM+SPN, data collection was restarted after interaction with the app to take the provided survey.

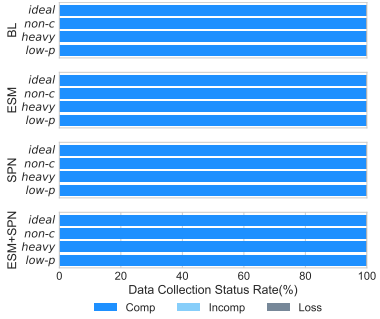
Memory Warnings Figure 5.3 depicts the total number of memory warnings during the study for each case. As is evident from 5.3, data collection had been suspended on *heavy* devices frequently. On the other hand, *ideal* devices did not suspend the application even once. In this sense, the user of a heavy iOS device runs the risk of reduced data collection rate.

6 Design Guideline for Sustainable MCS Studies on iOS

An MCS study involves the following phases: (1) research planning, (2) application development, (3) data collection, and data analytics. In this section, we illustrate a guideline for sustainable data collection by using iOS devices. Based on performance evaluation and case studies, we list recommendations corresponding to each phase for *AWARE-iOS*.



(a) Location sensor (b) Weather sensor



(c) Pedometer sensor

Fig. 9. Data collection status rates

1. Research planning phase
   (a) A researcher should make one or more hypotheses and attempt to collect sensor data as preliminary research to check the difference between Android and iOS using a preset client. (Section 3.1 and 4.2)
   (b) The number of sensors, durations of sensing intervals, and levels of accuracy on the sensing application should be optimized to reduce battery consumption. (Section 5.2)
   (c) An application distribution method that agrees with the research plan should be selected for smooth app distribution. (Section 3.4)
2. Application development phase
   (a) The number of *PS* should be minimized by using `subpod` for smooth app distribution. (Section 4.1)
   (b) The sensing application should present periodic opportunities to the user to open it, if possible. (Section 5.3)
   (c) To detecting suspension of the application, a researcher should send Silent Push Notifications to the participants' devices regularly during a study. (Section 5.3)
   (d) We recommend using a software-based sensor that is collected by motion co-processor automatically if you can replace it from other sensors. (Section 5.3)

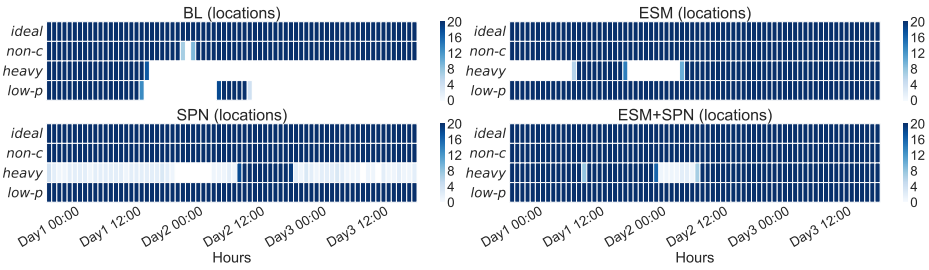


Fig. 10. Coverage of location data collection

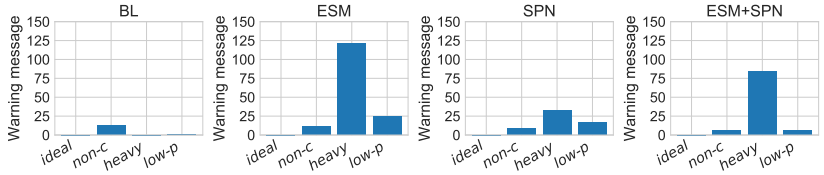


Fig. 11. Memory warnings

(e) The table size corresponding to SQLite should be less than 10^6 rows to reduce CPU impact and fetch speed. (Section 5.1)
(f) The batch size for data insertion should be less than 10^3 rows to reduce CPU usage. (Section 5.1)

3. Data collection phase
   (a) Sufficient free storage space should be obtained before beginning the study. In particular, it should be noted that iPhones have models with low storage capacities (e.g., 16 and 32 GB). (Section 3.2)
   (b) If possible, low power mode should be turned off during the study. (Section 5.3)

The performance evaluation in this study was conducted on limited types of iOS devices and OS. Therefore, if a device or OS of a different type is considered, its performance might be drastically different. Moreover, this study was performed over merely three days. The performance of AWARE-iOS needs to be evaluated over much longer durations to draw dependable conclusions regarding it. Moreover, we did not inspect the differences in settings on Anchor (see Section 4.1) in this study. Further, location sensors with high accuracy might consume battery at a faster rate, but their superior performance grants them a better data collection rate.

7 Conclusion

MCS is a research method for understanding everyday human activities on the individual, group, and social levels using data collected through smartphones.

Even though Android is the most popular OS for smartphones globally, iOS is more popular in certain specific areas or among certain communities. Several MCS frameworks [5–7] for iOS have been proposed. However, planning and managing a sustainable MCS-based study with iOS remains a challenge, in spite of the strict rules that iOS enforces on collecting sensor data in the background.

In this paper, we propose an MCS framework for iOS, namely, *AWARE-iOS*, based on the AWARE framework for Android [5]. *AWARE-iOS* allows us to collect sensor data sustainably based on a few lines of code or by using a published client application, and grants us the opportunity to optimize each research purpose.

Our performance evaluations show that the battery consumption rates of devices depend on sensor settings and storage type. Moreover, case studies illustrate that *ideal-* and *non-control* devices can collect nearly all the data. However, when the device is on low power mode or is being used heavily, it runs the risk of data loss if silent-push notifications or periodical interactions are not employed. Finally, based on these results, we propose a guideline for creating a research plan and managing mobile sensing studies.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP18K11274. This is a post-peer-review, pre-copyedit version of an article published in Lecture Notes in Computer Science, vol 12203 (HCII 2020: Distributed, Ambient and Pervasive Interactions). The final authenticated version is available online at: `http://dx.doi.org/10.1007/978-3-030-50344-4_17`.

References

1. Center, P.R.: Smartphone ownership is growing rapidly around the world, but not always equally (feb 2019)
2. Lane, N., Miluzzo, E., Choudhury, T., Campbell, A., et al.: A survey of mobile phone sensing. IEEE Communications Magazine **48**(9) (sep 2010) 140–150
3. Wang, R., Chen, F., Chen, Z., Li, T., Campbell, A.T., et al.: StudentLife: assessing mental health, academic performance and behavioral trends of college students using smartphones. In: Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing. (2014) 3–14
4. Rachuri, K.K., Aucinas, A., et al.: EmotionSense: A Mobile Phones based Adaptive Platform for Experimental Social Psychology Research. Proceedings of the 12th ACM international conference on Ubiquitous computing - Ubicomp '10 (2010) 281
5. Ferreira, D., Kostakos, V., Dey, A.K.: AWARE: Mobile Context Instrumentation Framework. Frontiers in ICT **2** (apr 2015) 6
6. Katevas, K., Haddadi, H., Tokarchuk, L.: Sensing Kit: Evaluating the sensor power consumption in iOS devices. Proceedings - 12th International Conference on Intelligent Environments, IE 2016 (2016) 222–225
7. Xiong, H., Gerber, M.S., et al.: Sensus: A Cross-Platform, General-Purpose System for Mobile Crowdsensing in Human-Subject Studies. In: Proceedings of the 2016

ACM International Joint Conference on Pervasive and Ubiquitous Computing, NY, USA, ACM Press (2016) 415–426
8. Statcounter: Mobile Vendor Market Share. https://gs.statcounter.com/vendor-market-share/mobile/japan
9. Bae, S., Chung, T., Ferreira, D., Dey, A.K., Suffoletto, B.: Mobile phone sensors and supervised machine learning to identify alcohol use events in young adults: Implications for just-in-time adaptive interventions. Addictive Behaviors **83** (aug 2018) 42–47
10. Doryab, A., Villalba, D.K., Chikersal, P., Mankoff, J., Creswell, J.D., Dey, A.K., et al.: Identifying Behavioral Phenotypes of Loneliness and Social Isolation with Passive Sensing: Statistical Analysis, Data Mining and Machine Learning of Smartphone and Fitbit Data. JMIR mHealth and uHealth **7**(7) (jul 2019) e13209
11. Kuosmanen, E., Kan, V., Ferreira, D., et al.: Challenges of parkinson's disease: User experiences with stop. In: Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services. MobileHCI '19, NY, USA, Association for Computing Machinery (2019)
12. Wang, R., Harari, G., Campbell, A.T., et al.: Smartgpa: How smartphones can assess and predict academic performance of college students. In: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing. UbiComp '15, NY, USA, Association for Computing Machinery (2015) 295–306
13. Hormuth, S.E.: The sampling of experiences in situ. Journal of Personality **54**(1) (mar 1986) 262–293
14. Froehlich, J., Chen, M.Y., Consolvo, S., Harrison, B., Landay, J.A.: MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones (2007)
15. Shiffman, S., Stone, A.A., Hufford, M.R.: Ecological momentary assessment. Annual review of clinical psychology **4** (2008) 1–32
16. Aharony, N., Pan, W., Ip, C., Khayal, I., Pentland, A.: The social fMRI: measuring, understanding, and designing social mechanisms in the real world. In: Proceedings of the 13th international conference on Ubiquitous computing - UbiComp '11, NY, USA, ACM Press (2011) 445
17. Place, S., Nierenberg, A., Azarbayejani, A., et al.: Behavioral indicators on a mobile sensing platform predict clinically validated psychiatric symptoms of mood and anxiety disorders. J Med Internet Res **19**(3) (Mar 2017) e75
18. Wu, P., Zhu, J., Zhang, J.Y.: MobiSens: A versatile mobile sensing platform for real-world applications. Mobile Networks and Applications **18**(1) (2013) 60–80
19. Trossen, D., Pavel, D.: AIRS: A mobile sensing platform for lifestyle management research and applications. In Borcea, C., Bellavista, P., Giannelli, C., Magedanz, T., Schreiner, F., eds.: Mobile Wireless Middleware, Operating Systems, and Applications, Berlin, Heidelberg, Springer Berlin Heidelberg (2013) 1–15
20. Lane, N., Choudhury, T., Campbell, A., et al.: BeWell: A smartphone application to monitor, model and promote wellbeing. Proceedings of the 5th International ICST Conference on Pervasive Computing Technologies for Healthcare (01 2011)
21. ilumivu: mEMA. https://ilumivu.com/
22. Pollak, J.P., Adams, P., Gay, G.: PAM: a photographic affect meter for frequent, in situ measurement of affect. In: CHI. (2011)