Energy Efficient Guidelines for iOS Core Location Framework

Abdul Ali Bangash*, Daniil Tiganov†, Karim Ali‡ and Abram Hindle§
Department of Computing Science, University of Alberta
Edmonton, AB, Canada
Email: *bangash@ualberta.ca, †tiganov@ualberta.ca, ‡karim.ali@ualberta.ca, §abram.hindle@ualberta.ca

2021 IEEE International Conference on Software Maintenance and Evolution (ICSME) | 978-1-6654-2882-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICSME52107.2021.00035

***Abstract*—Several types of apps require accessing user location, including map navigation, food ordering, and fitness tracking apps. To access user location, app developers use frameworks that the underlying platform provides to them. For the iOS platform, the Core Location framework enables developers to configure various services to obtain user location information. But how does a particular configuration affect the energy consumption of an app? The available Core Location framework documentation is insufficient to help developers reason about the tradeoff between choosing a particular configuration and energy consumption.**

In this paper, we present a set of guidelines that will help developers make an energy-efficient design choice while configuring the Core Location framework for their app. To achieve that, we have created microbenchmark configurations of the various services that the Core Location framework provides. We have then run several test-scenarios on these configurations to extract their energy profiles. To extract energy-efficient guidelines for developers, we have carefully examined those energy profile results. The guidelines show several configurations that not only reduce energy consumption but also access locations more frequently than other configurations. To evaluate those guidelines, we analyzed three real-world apps and a location service sample app provided by Apple. Our results show that the guidelines help reduce energy: 0.42% for a property search app, 10.59% for a weather app, 26.91% for a location utility app, and 11.37% for Apple's sample app. Additionally, our empirical evaluation shows that choosing an energy-hungry configuration can increase the energy consumption by up to a maximum of 23.97%. Our guidelines are effective on 3 real-world apps, and our methodology may be used to extract energy-efficient guidelines for frameworks other than the Core Location framework.

***Index Terms*—software energy consumption, developers guide, iOS development, smartphone apps**

I. Introduction

Energy consumption of smartphone applications (apps) is a rising concern for developers [1], [2]. An energy hungry app leads to unsatisfactory user experience, negative reviews, and even uninstallation of the app [2], [3]. To reduce energy consumption, developers look out to employ design choices and energy optimization techniques during development. In prior studies, correct design choice have proven to be more effective [4], [5] as compared to the energy optimization techniques [6], on real-world apps.

The situation is not much different for the iOS platform. Despite energy consumption being a major concern for iOS users [3], and that app developers prefer iOS over other development platforms [7], [8], the iOS platform, in general, suffers from lack of energy measurement frameworks and energy efficient guidelines for developers. Apple advises developers to focus on reducing the energy consumption of processing, networking, location, and graphics [9]. This advice is particularly important for location services, given the pervasive use of location information in various apps such as turn-by-turn map navigation, social networking, food and groceries ordering, carpooling, and vehicle hiring. Improper access of location information may prevent the device from going into sleep mode, which keeps the location hardware powered on.

To access user location, Apple provides iOS developers the Core Location framework with multiple location services: Standard Location Service, Significant Location Service, Visits Location Service, and Regional Monitoring Service. Each service is configurable to access user location with a certain frequency and accuracy. This frequency and accuracy comes at the cost of battery life through energy consumption. Battery life and accuracy is an engineering tradeoff a software developer needs to make while developing an app. Unfortunately, the current Apple developer's guide does not provide any information about such tradeoff [10]. It suggests choosing the most energy-efficient service but it does not provide details about which services are most energy-efficient:

"Always choose the most power-efficient service that serves the needs of your app. Disable location services when you do not need the location data offered by the service. For example, you might disable location services when your app is in the background and would not use that data otherwise."
–AppleInc [11] (Core Location Documentation)

To bridge this gap, we extract energy profiles of the location services, and to extract the profiles, we develop iGreenMiner, an automated energy measurement framework for iOS apps. We created 28 microbenchmarks, each representing a different configuration of a framework's location service. We use iGreenMiner to collect the energy profiles of each configuration and derive general guidelines for the developers from the collected energy profiles. Our analysis shows that, overall, energy-wise, *Visits Location Service* is the most efficient service, while *Standard Location Service* performs the worst. However, we find out that Standard Location Service can be configured with certain parameter values to consume the least amount of energy among all other configurations. Using the energy profiles, we have created a graph that describes the tradeoff between location access rate and energy consumption. Developers may use this graph as a guide to choose one

2576-3148/21/$31.00 ©2021 IEEE
DOI 10.1109/ICSME52107.2021.00035

location service over another. Finally, we have evaluated the effectiveness of these guidelines on three real-world apps as well as the sample app that Apple provides to its developers. On real-world apps, we were able to reduce up to 26.91% energy consumption and on sample app we reduced 11.37% energy consumption.

Our results have the potential to guide the developers in making informed, energy-efficient design choices before runtime for accessing user location in their app. We make the following contributions:

1) **Location framework benchmarks.** We created microbenchmarks with multiple configurations of the iOS Core Location framework. These configurations can be used to evaluate the effects of using different location services on app qualities such as energy consumption, execution speed, and user experience.
2) **Energy measurement framework.** We developed an automated iOS energy measurement framework, iGreenMiner, that developers can access through a web service.
3) **Recommendation guide.** We present guidelines to help developers make energy-efficient design choices while using the iOS Core Location framework. Our results provide evidence that developer's design choice can improve their app by reducing its energy consumption.

II. iOS Core Location Framework

The iOS Core Location framework provides five services:

- Standard and Significant Location Services: tracks changes in user location with configurable accuracy,
- Regional Monitoring Service: tracks user entry and exit from specific regions,
- Beacon Ranging Service: tracks presence near a beacon,
- Visits Location Service: tracks the location where a user has spent a significant amount of time, and
- Compass Headings Service: tracks user direction.

Some of these services are configurable with additional parameters. Developers may use these parameters to change the location accuracy and location access frequency. In this paper, we investigate how these parameter configurations affect energy consumption. We exclude the Beacon Ranging Service and Compass Headings Service from our investigation, because they do not report user location. Beacon Ranging Service detects a user's presence near a beacon and Compass Headings Service detects a user's movement direction.

a) Standard Location Service: This configurable service provides real-time user location. According to the iOS documentation [12], it is the most power-hungry service because it provides the most immediate location with highest accuracy. The documentation recommends using this service only when an app requires real-time location such as for navigation instructions or recording a user's hiking path.

Developers may configure this service through two parameters: `desiredAccuracy` and `distanceFilter`. The former may be set to `Best`, `HundredMeters`, `Kilometer`, `Navigation`, `ThreeKilometers`, or `NearestTenMeters`. The latter pauses the location service from fetching further locations unless the user movement exceeds the specified distance (in meters). By default, `desiredAccuracy` is set to `Best` and `distanceFilter` is set to `None` (i.e., the service updates the location for any user movement).

Given the lack of documentation about how these values affect energy consumption or location accuracy, developers might find it difficult to decide which value best suits their needs. In particular, Apple's documentation [12] provides a few guidelines for setting the value of `desiredAccuracy`. First, the developer should not set the accuracy level to `Best` or `NearestTenMeters` unless the app requires location updates every few meters. Second, in practice, the framework provides more accurate data than requested. For instance, accuracy `ThreeKilometers` provides an accuracy closer to one-hundred meters. This brief documentation of the configuration parameters does not help developers determine how these parameters may affect the energy consumption of their apps.

b) Significant Location Service: According to Apple's documentation [13], developers should use this service to get the current location and be notified only when a significant distance has been covered. This service does not offer any configuration parameters, and the distance accuracy is not mentioned in the documentation. The documentation states that Significant Location Service is the most energy-conservative service but provides the least location accuracy [13].

c) Visits Location Service: This service detects only noteworthy movements of a user [14]. The service is not intended to be used for real-time location data but rather to identify patterns in user location. The service has a parameter: `activityType`, which pauses location updates according to its type: `other`, `automotiveNavigation`, `fitness`, `otherNavigation`, and `airborne`. The `automotiveNavigation` option pauses the service location updates if it detects that the user in not traveling in an automotive. Similarly, `fitness` works for pedestrian activities where indoor positioning is disabled. The `otherNavigation` option works for navigation of boats, trains, or planes. There is no information available regarding `other` or `airborne` in documentation. However, the default `activityType` is `other`.

d) Regional Monitoring Service: This service is a low-energy alternative for a scenario where the app needs to identify the user presence within a certain geographical region [15]. A geographical region is a circle with a radius and a center point on the Earth's surface. To begin monitoring a region, the developer has to define a geographical region in their code. The service starts monitoring the user presence in the defined region immediately after the app starts.

Figure 1 provides a code example that shows how developers may initialize each location service to access user location. Lines 2–4 represent the usage of Standard Location Service, initializing `desiredAccuracy` to `Best` and `distanceFilter` to 2^{12} meters. Line 7 represents the usage of the Significant Location Service. Lines 10–14 represent the usage of Regional Monitoring Service. Line 10 defines the `maximumDistance` of the region, while Line 11 defines a

```
1  // Initializing Standard Location service
2  locationManager.desiredAccuracy =
       kCLLocationAccuracyBest;
3  locationManager.distanceFilter = 4096;
4  locationManager.startUpdatingLocation();
5
6  // Initializing Significant Location service
7  locationManager.
       startMonitoringSignfcntLocationChanges();
8
9  // Initializing Regional Monitoring service
10 let maxDistance = locationManager.
       maximumRegionMonitoringDistance;
11 let region = CLCircularRegion(center: center,
       radius: maxDistance, identifier:
       identifier);
12 region.notifyOnEntry = true;
13 region.notifyOnExit = false;
14 locationManager.startMonitoring(for: region);
15
16 // Initializing Visit Monitoring service
17 locationManager.startMonitoringVisits();
18 locationManager.activityType = CLActivityType.
       fitness;
```

Fig. 1: An example illustrating how to configure various services offered by the iOS Core Location framework.

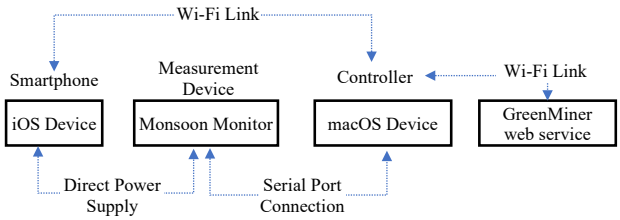


Fig. 2: The major components of iGreenMiner: an iOS device, measurement device, controller, and a web service. The dotted lines represent connections between these components.

region using a `radius` and a `center`. The `center` contains a latitude and a longitude. Line 14 initiates the user location monitoring for the specified `radius`. Lines 17–18 represent the usage of the Visits Location Service, initializing its parameter `activityType` with `fitness`.

III. Location Services Energy Profiling

To help developers find out how various location services and their parameters affect energy consumption, we have created microbenchmark configurations of the location services. To profile their energy consumption, we have developed iGreenMiner, an automated hardware-based energy measurement framework for iOS apps, accessible through a web service [1]. In this section, we present iGreenMiner, the location services microbenchmark configurations, and the test scenarios for evaluating these configurations.

A. Overview of iGreenMiner

The iGreenMiner framework extends GreenMiner [16], an Android-based energy measurement framework, to support the energy measurement of iOS apps. Figure 2 depicts the iGreenMiner infrastructure. iGreenMiner is a continuous testing suite that instruments an iOS device (currently supporting iPhone 11 running iOS version 13.4.1 and iPhone 6S running iOS version 9.0.1), runs multiple test cases on the device using Xcode 11.3 [17] that is an iOS app development platform, and measures the energy consumption of the entire device during a test run using a power monitor. iGreenMiner has a controller (a macOS device) that runs test scripts on the iOS device using Apple Script [18], automates the process of test case execution on the iOS device and collects energy measurement results. To measure energy consumption, iGreenMiner uses a Monsoon power monitor [19] (i.e., a current draw measurement instrument). Monsoon provides a direct voltage (4.2 Volts) to the device, measures the current draw of the running device in milliAmps (mA), with a sampling rate of 5 kHz. The power monitor then reports the current value with a timestamp to the controller. The controller records these power measurements, converts them into energy as Joules (J) with respect to time. It then uploads the computed measurements to a server for the results to be available at the GreenMiner web service.

To avoid non-deterministic and erroneous values, we take several precautions during our measurements. First, to avoid battery conditions and the battery aging effect, we substitute the measurement device battery with a constant direct supply of 4.2 Volts. Second, to deploy test cases on the device, we connect the controller to the measurement device through WiFi. This wireless connection helps to bypass the charging state of the measurement device, which usually turns on when the device is connected to the controller via a USB cable. Third, to synchronize the Monsoon monitor measurements with the test case execution time, we run a shell script that maps the energy measurement timestamps to the test case execution time.

B. Microbenchmark Creation

To extract the location services energy profile, we have created a basic iOS app structure to access user location. Based on this structure, we have created multiple microbenchmark configurations. Each configuration implements a different location service with various parameter values. For Significant Location Service and Regional Monitoring Service, we have created only one configuration, because both of them do not have any parameters to configure. For Visits Location Service, we have created 6 configurations, each has a different value for the parameter `activityType`. For Standard Location Service, we have created 20 configurations, each has a different value for the parameters `desiredAccuracy` and `distanceFilter`. For each value for `desiredAccuracy`, we have selected 5 different values for `distanceFilter`: 2^4 meters, 2^8 meters,

[1] https://pizza.cs.ualberta.ca/gm/

TABLE I: The microbenchmark configurations that we have created for evaluation. Each configuration implements a different location service with different parameter values. S1–S20 represent configurations for Standard Location Service. V1–V6 represent configurations for Visits Location Service. SG and RM (not part of this table) represent the Significant Location Service and Regional Monitoring Service, respectively.

desiredAccuracy	distanceFilter			
	Best	Hundred Meters	Kilometer	Navigation
2^4	S1	S6	S11	S16
2^8	S2	S7	S12	S17
2^{12}	S3	S8	S13	S18
2^{16}	S4	S9	S14	S19
None	S5	S10	S15	S20

	activityType					
	airborne	automotiveNavigation	default	fitness	other	otherNavigation
Configuration	V1	V2	V3	V4	V5	V6

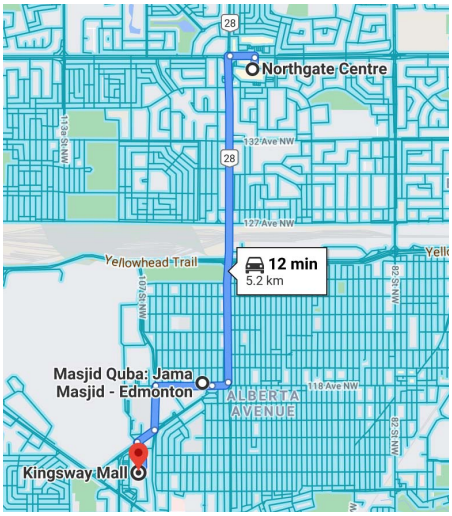


Fig. 3: A path for a user traveling from Northgate Centre to Kingsway Mall.

2^{12} meters, 2^{16} meters, and None (i.e., zero meters). Table I outlines all of the 28 configurations.

We do not consider the values of `desiredAccuracy` with `ThreeKilometers` and `NearestTenMeters`, because their functionality may be achieved with a combination of `desiredAccuracy` values set to `Kilometer` and `Best` with a `distanceFilter` set to `3000 meters` and `10 meters`, respectively. To accurately generate the energy profiles of the configurations, we run multiple test scenarios 5 times on each configuration, and compute the median energy measurement value as the representative of a configuration. We used median energy to mitigate the risk of outliers in the energy measurements that the Monsoon power monitor measures due to low frequency sample rate.

C. Measurement Process

To specifically measure the location service usage while the whole app is running, we first need to know how a location service works. A location service, as per its parameter configurations, initializes a handler that constantly tracks the movement of a user by using an accelerometer sensor and fetches the location by using GPS, cellular network, iOS beacon, or WiFi after every several meters. Since our goal is to evaluate the energy consumption of a location service usage and not its initialization, we exclude the handler initialization part from the energy measurements. To achieve that, we crop the energy measurement of the initialization of the handler through a time synchronization script and measure the energy consumption of movement tracking and location fetching only. To evaluate the configurations, we run two test scenarios on each of the configuration.

1) The first test scenario: measures the energy consumption of location fetching only, keeps the location stationary.
2) The second test scenario: measures the energy consumption of location fetching with user movement, simulates a user that travels from one point to another point on earth. Figure 3 shows the path, using Google Maps, where the user covers approximately 5.2 kilometers in 600 seconds from Northgate Centre to Kingsway Mall. We simulate the user location using Xcode 11.3 [17], an iOS app development platform.

To avoid erroneous energy measurements, we keep the screen brightness constant throughout the execution of a test case [20]. To log the number of locations accessed by a configuration (i.e., location access rate) during each test case execution, we instrument each configuration and re-execute the test cases on it separately. As a result, for analysis and comparison, we collected 140 non-instrumented and 140 instrumented energy measurements.

TABLE II: Energy consumption difference (p-values for Wilcoxon rank sum test) among the configurations of Standard Location Service for the desiredAccuracy parameter.

desiredAccuracy	Best	HundredMeters	Kilometer
HundredMeters	**0.048**	-	-
Kilometer	**0.048**	1.000	-
Navigation	1.000	**0.048**	**0.048**

TABLE III: Energy consumption difference (p-values for Wilcoxon rank sum test) among the configurations of Visits Location Service for the activityType parameter.

activityType	Airborne	AutoNav	Default	Fitness	Other
AutoNav	1.000	-	-	-	-
Default	1.000	1.000	-	-	-
Fitness	1.000	1.000	1.000	-	-
Other	1.000	1.000	1.000	1.000	-
OtherNav	1.000	1.000	1.000	1.000	1.000

D. Accurate Energy Measurement

To ensure accuracy throughout our empirical evaluation, we took 4 main measures.

First, the device could not be locked during a test case execution. This is a constraint of Xcode [17] that for a locked device, it loses connection with the device under test. As an alternative, to mimic the test environment of a real user, when the app is run in the background, we decrease the display brightness to the lowest point such that the screen becomes unreadable and consumes least energy [20].

Second, iGreenMiner with its current settings supports two iOS devices: iPhone 11 and iPhone 6S. To minimize the differences in device and iOS version-specific performance, we use only one device (iPhone 11) in our experiments.

Third, installation of the app and initialization of the location service handler is an overhead to our energy measurements. To avoid this overhead, we start measuring energy after the initialization of the app and the handler.

Finally, installing the same configurations and accessing the same location information multiple times can introduce performance differences due to cached app data. To avoid location data caching, we wipe out the app's data completely and install altering configurations each time prior to a test run.

IV. Which Location Service Should You Use?

To help developers understand the effect of using a particular location service on energy consumption, we have profiled all the energy consumption of all the configurations from Table I. We then evaluate those configurations through answering the following research questions:

- **RQ1:** Does using a particular location service affect the energy consumption of an app?
- **RQ2:** Which location service configuration may help developers reduce the energy consumption of their app?
- **RQ3:** Do the default location service configurations yield energy-efficient code?
- **RQ4:** Does the energy consumption of a location service change if it runs as a background service as opposed to a foreground service?
- **RQ5:** Which location service configurations perform best in terms of accessing most locations while consuming least energy?

A. Effect of Using A Particular Location Service (RQ1)

To evaluate the energy profiles of the configurations from Table I, we ran the second test scenario described in Section III-C on the configurations. Figure 4 represents the energy profiles of each configuration. Across all configurations, the minimum energy consumed by a configuration is 373 joules, and for the same purpose (location access), the maximum energy consumed by another configuration is 463 joules. This shows that if developers choose a particular service with certain parameter values, it can increase the energy consumption of their app up to 23.97%.

To find out the effect of each service parameter values on energy consumption, we have evaluated the configurations of Standard Location Service and Visits Location Service by applying pairwise Wilcoxon rank-sum test [21] on the energy measurements of these configurations. We keep α (significance level) as 0.05 and apply Bonferroni adjustment method to address the risk of type-I error. Table II and Table III shows the results of our test. For Standard Location Service, there is a statistically significant difference among the configurations for different values of the parameter `desiredAccuracy`. The bold values indicate $p < 0.05$. For Visits Location Service, varying values of the *activityType* parameter does not cause any significant difference.

Our results show that developers should be concerned about their choice of parameters within the Standard Location Service. However, in the case of the Visits Location Service, the choice of a configuration does not matter as there is no difference among the energy consumption of its configurations.

> Guideline 1 (G1): Developers should be conscious about choosing a location service configuration as choosing an energy hungry configuration can increase the energy consumption by up to 23.97%.

B. Effect of Configuring Location Services (RQ2)

To answer this question, we ranked all configurations based on the energy profiles that we have collected for RQ1. Across all configurations, collectively, Visits Location Service consumes the least energy followed by Regional Monitoring Service, Significant Location Service, and Standard Location Service. In particular, V1 and V5 from Visits Location Service and S12 from Standard Location Service, perform better than all other configurations while providing the same functionality.

> Guideline 2 (G2): Collectively, across all configurations, Visits Location Service is the most energy efficient choice for developers while Standard Location Service is the worst choice.

To identify the most energy-efficient configurations within Standard Location Service and Visits Location Service, we compare the energy profiles of the configurations of these

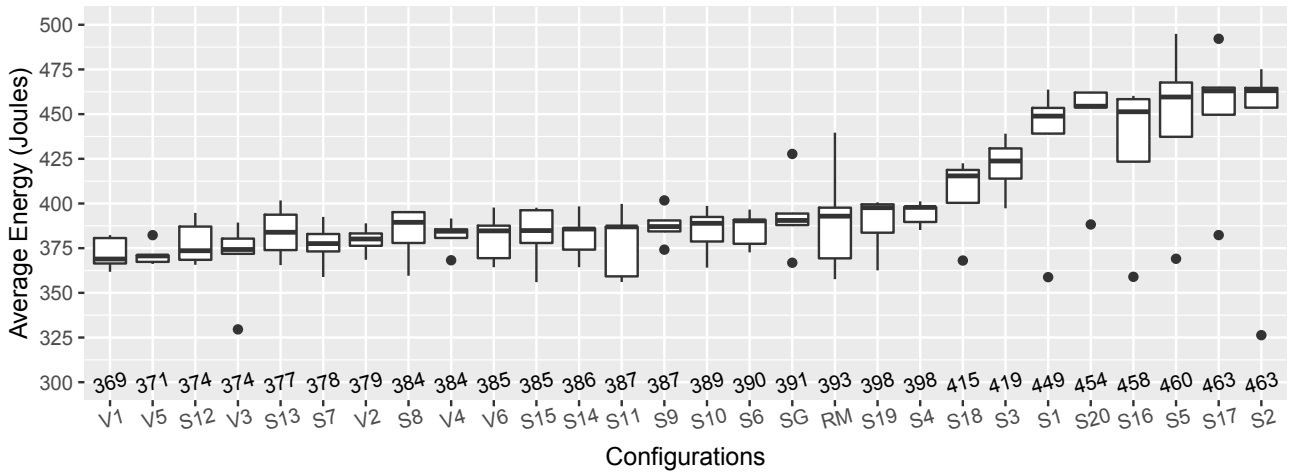


Fig. 4: The energy profile of each microbenchmark configuration from Table I. Values above the microbenchmark name along the x-axis represent the median energy consumption in Joules. Y-axis represents the average energy consumption in Joules.

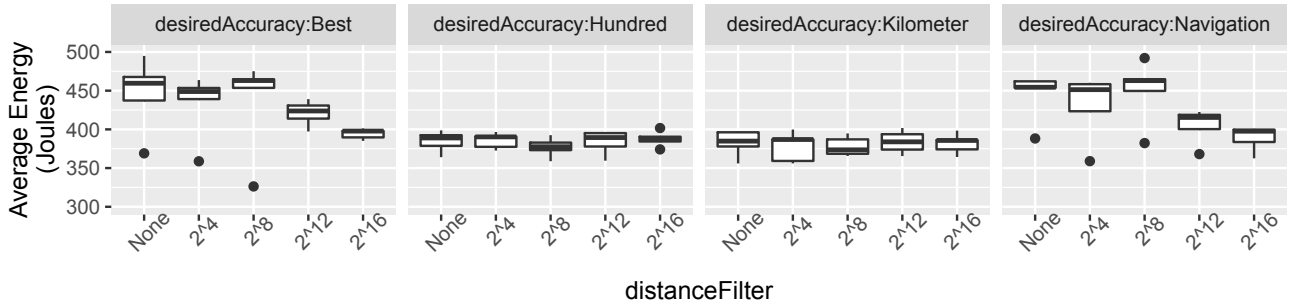


Fig. 5: The energy profile for each configuration of Standard Location Service.

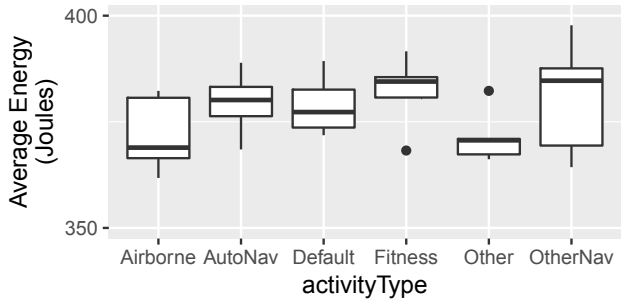


Fig. 6: The energy profile for each configuration of Visits Location Service.

services. Figure 5 shows the energy profile for each Standard Location Service configuration. Each box-plot represents the five energy measurement values collected for each configuration, and the x-axis represents the values of the parameter `distanceFilter`. Each boxed section represents the parameter `desiredAccuracy`. Considering the median energy profiles, the best configuration of Standard Location Service is when the value of `desiredAccuracy` is `Kilometer` and the `distanceFilter` value is 2^8 meters. On the other hand, this service performs the worst when the `desiredAccuracy` value is `Best` and the value of `distanceFilter` is 2^8 meters.

Figure 6 shows the energy profiles of each Visits Location Service configuration. For this service, developers may use any configuration, because there is no configuration that stands out in terms of energy efficiency.

> Guideline 3 (G3): The most energy-efficient configuration for Standard Location Service is to set `desiredAccuracy` to `Kilometer` and the `distanceFilter` value to 2^8 meters (configuration S12).

C. Effect of Default Configuration Parameters (RQ3)

If the default parameter values in the Standard Location Service and Visits Location Service are energy-efficient, developers will not have to worry about the parameter values of these location services. To evaluate the default configurations, we have executed the second test scenario described in Section III-C on the default configurations and microbenchmark configurations, and compared their energy profiles.

For Standard Location Service, the default values of `desiredAccuracy` and `distanceFilter` are `Best` and `None` (i.e., zero meters), respectively. This is the S5 configuration in our microbenchmarks. Observing the energy measurements of Standard Location Service, Figure 5 shows that the default parameter values are not the most energy optimal choice. In fact, the default setting (S5) performs the third worst within the 20 configurations of Standard Location Service. Also in terms of location access rate per second, the default configuration performs second to another configuration.

For Visits Location Service, the default value of its parameter `activityType` is `other`. As we have previously discussed, there is no difference across the energy consumption of Visits Location Service configurations. Therefore the default configuration for Visits Location Service is a safe option to use, in terms of energy consumption, for developers.

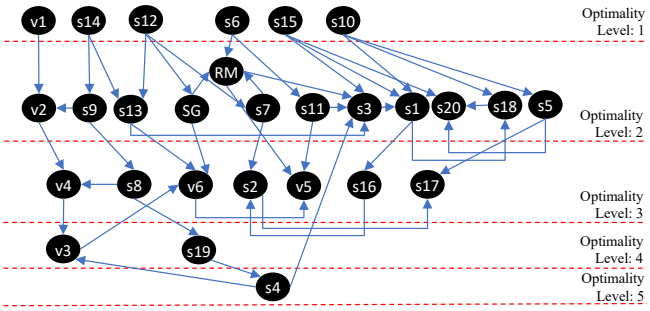
> Guideline 4 (G4): Developers should not use the default configuration of Standard Location Service, because it is not energy efficient. On the other hand, the default configuration for Visits Location Service is equally energy efficient to other configurations of the service.

D. Background vs Foreground Service (RQ4)

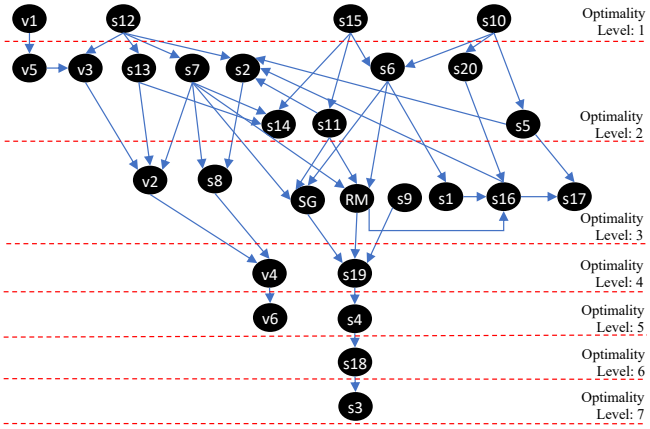
Depending on its requirements, an app may access location information while running in the background of the UI thread; for instance, a fitness app that continuously accesses locations while the phone is locked and inside the user's pocket. Contrarily, an app might access location while running at the foreground of the UI thread; for instance, a map navigation app continuously accessing locations while the phone is docked at the car's dashboard. The scope of our study is limited to apps that access locations in the background of the UI thread. However, for a brief comparison of the effects of accessing location in the background vs foreground, on energy, we re-execute the second scenario from Section III-C on our configurations in the foreground of the UI thread.

To compare the energy profiles of location access in the background of the UI thread against that in the foreground of the UI thread, we ran a paired Wilcoxon signed rank test between the two observations with Bonferroni adjustment applied to address the risk of type-I error. As a result, we get $p = 7.4e - 9$ for the difference in energy consumption, and $p = 0.1185$ for the difference in location access rate. This results implies that for α kept 0.05, there is a statistically significant difference among the observations for energy consumption, while there is no statistically significant difference among the observations for location access rate.

The average energy consumption for background services and foreground services is 402.07 joules and 1,255.20 joules, respectively. This overhead energy difference is due to several factors. The iOS operating system generally provides more resources to foreground tasks because they directly interact with the user [22]. Additionally, the screen consumes more energy for foreground services [20], because for background services the screen is dark, whereas for the foreground execution the screen is well lit with a white background. However, a detailed investigation is required to reason about the difference between the two observations, which is out of the scope of this study.



(a) Benchmarks ran at Foreground



(b) Benchmarks ran in Background

Fig. 7: The graph represents microbenchmarks which are better than the others when run at the background or foreground of a device. For example, an edge (line) from S12 to S13 means that S12 is better, it has lower energy cost and higher location access rate than S13. Optimality Level represents configurations that collectively dominate other configurations, e.g., Level 1 to Level 7 means most optimal to least optimal.

> Guideline 5 (G5): The energy consumption of accessing location information in the foreground of the UI thread is 3× higher than accessing it in the background of the UI thread.

E. Dominating Configurations (RQ5)

We have previously shown that, collectively, among all configurations, Visits Location Service is the best, whereas, Standard Location Service is the worst in terms of energy consumption. But what if Visits Location Service accesses the least number of locations, whereas, Standard Location Service accesses the most? If an app requires frequent location updates (e.g., for map navigation), then the app developer typically

aims for using a service that accesses the most number of locations while using the least energy.

To find out the location service that has the most number of location accesses with least amount of energy consumption, we execute the second test scenario described in Section III-C on the microbenchmark configurations twice: (1) in the background of the user-interface (UI) thread and (2) in the foreground of the UI thread.

To present the pareto-optimal solutions, we use a better-than-graph: a directed graph where a source node represents a better configuration than the target node. This graph helps developers know which location service configurations dominate others in terms of less energy consumption and high location access rate. Figure 7 depicts this better-than-graph. For accessing locations at the foreground of the UI thread, S6, S10, S12, S14, and S15 from Standard Location Service, as well as V1 from Visits Location Service are the most dominating solutions. While for accessing locations at the background of the UI thread S10, S12, and S15 from Standard Location Service, as well as V1 from Visits Location Service are the most dominating solutions.

Guideline 6 (G6): Developers can use our graph as a guide to choose a location service based on a trade-off between location access rate and energy consumption.

V. Discussion

A. Comparing Our Guidelines with the Apple iOS Energy Guidelines

We compare the results of our study with the official energy guidelines provided by Apple [10], [23], [24]. Although there is a lack of documentation for the location services, Apple does provide some general energy guidelines regarding these services. We provide a summary of the Apple guidelines (AG) and compared them to our own set of guidelines.

- **AG1:** Visits Location Service is the most energy-efficient choice when the app needs a user movement pattern instead of real-time locations. AG1 is recommended for a scenario when an app needs the location only if the user has spent significant amount of time at that location.
- **AG2:** Significant Location Service is the most power-efficient way when the app needs continuous live locations. AG2 is recommended for a scenario when real-time location updates are less frequently required.

Our results complement AG1, as we have previously shown that Visits Location Service is the most energy-efficient choice when compared to other location services (G2). However, contrary to AG2, we recommend that developers should prefer using some specific configurations of Standard Location Service and Visits Location Service over Significant Location Service. This is because these specific configurations are more energy-efficient and provide higher location access rate (G6).

- **AG3:** Standard Location Service uses significantly more power than other location services, but it delivers the most accurate and immediate location information.

Our results partially complement AG3. We discovered that Standard Location Service delivers the most frequent updates and is collectively the most energy intensive service (G4). However, we have shown that if Standard Location Service is configured with certain parameter values (S10, S12, S15), it is more energy efficient than Visits Location Service and Significant Location Service (G3).

- **AG4:** Developers should stop location services when the location is not needed anymore.

AG4 is not covered in our test scenarios, because our goal is to evaluate the energy consumption of the service itself when it is running. However, it is reasonable to believe that stopping a service will reduce energy.

- **AG5:** Setting Standard Location Services' `desiredAccuracy` to `Best` is the most energy-hungry decision for developers, but it provides the most number of locations at the same time.

Unlike AG5, our experiments have shown that the suggested setting performs second best (0.83 locations per second for background, 0.84 locations per second for foreground) to another setting where the `desiredAccuracy` is set to `Hundred` meters (0.905 locations per second for background, 0.89 locations per second for foreground). This alternate setting also consumes less energy than the proposed setting in AG5. A further in-depth investigation is required to find out why `Best`, with lower location access rate, consumes more energy than `Hundred` meters.

- **AG6:** Defer location updates from the services. Queue the locations and process them all at once sometime.

AG6 is an interesting guideline. Similar to prior work [4], the technique of queuing up processes has proven to reduce energy consumption due to less context switches. However, evaluating AG6 is out of the scope of our study.

B. Why Are There Energy Differences Among the Location Service Configurations?

The total energy consumption of a location service configuration is an aggregate of three costs: movement tracking sensors, service handler computation, and location fetching. Movement tracking sensors include the accelerometer sensor and it stays constantly active when the phone is in motion. Service handler computation involves the handler initialization and call frequency. The frequency of handler calls is configured by the `distanceFilter` parameter. Location fetching involves utilizing hardware components by accessing locations through a combination of nearby iOS beacons, network routers, cellular towers, and GPS.

Among these functionalities, in our study, we are interested in the location fetching part. To investigate the cost of location fetching alone, we execute the first test scenario from Section III-C on all configurations in the foreground of the UI thread. We investigate the location fetch energy cost alone by keeping the phone location stationary. A stationary location limits the usage of movement sensors and reduces the service handler cost that includes movement tracking.

To measure the difference among the benchmarks for this test scenario, we performed an unpaired Wilcoxon rank-sum test on the energy measurements. The results show that for $\alpha = 0.05$, and Bonferroni adjustment method applied, all resultant p-values are > 0.05, implying no statistically significant difference among the configurations in terms of energy consumption. This result shows that the energy difference observed among the configurations is mainly due to two reasons: **(1)** frequency of the location fetching handler calls and **(2)** sensor usage for movement tracking.

C. Do Our Guidelines Help Improve the Energy Consumption of Real-World Apps?

To evaluate the usefulness of our guidelines, we have evaluated them on real-world apps. We replace real-world apps' current location service implementation with an energy-optimal alternative according to the better-than-graph from Figure 7. To profile the energy consumption, we execute the second test scenario from Section III-C on the original app and its modified (energy-optimal alternative) version. To automatically detect the usage of a location service and its parameter values, we have extended SWAN [25], an iOS static analysis framework. Our extension searches for method calls and configuration parameters of any location service usage.

1) Selecting Real-World Apps: To find real-world apps that use the Core Location framework, we have explored the occurrence of code snippets on Github [2] using the search string "`locationManager.start`" [3]. This string represents the usage of location service in iOS apps. As a result, we found 155 apps that use multiple programming languages. Since Apple recommends Swift for development on iOS [26], and iGreenMiner supports Swift as well, we exclude the apps that do not use Swift as a programming language. This step leaves us with 40 apps that are academic assignments, course projects, and professional apps for Apple's app store.

Across these 40 apps, we applied the following exclusion criteria: Swift version older than 5, build failure, login credentials not provided, and that the app crashes. This filtration leaves us with three types of app categories: property search, weather, and location utility. We then picked one app from each category for this evaluation. In addition to real-world apps, we also consider Apple's provided sample code implementation of the location service [27]. This sample code demonstrates the usage of location service for developers.

2) Results:

a) Property Search App [28]: this app helps users find real state properties near their location. The app uses Standard Location Service with `desiredAccuracy` set to `NearestTenMeters`, and `distanceFilter` set to `None`. This setup is equivalent to setting `desiredAccuracy` to `Best` and `distanceFilter` to `10`. The closest representative setup for this configuration in our study is S1.

To optimize this app, we replaced the current app S1 implementation with S12 because Figure 7a suggests that S12 accesses more locations than S1 with less energy usage. As a result, with an energy-efficient alternative, we reduced the energy consumption by 0.42%. The impact is small since the app accesses the user location only once during the whole test scenario.

b) Weather App [29]: this app reports the weather of the current user location. The app continuously monitors user location and updates the weather at every location access. The app uses Standard Location Service with `desiredAccuracy` set to `HundredMeters` and `distanceFilter` set to `None`. This configuration represents S10 in our study.

To optimize this app, we replaced the current app S10 implementation with S12 because Figure 4 shows that it is the most energy efficient configuration for Standard Location Service. As a result, by replacing the app's actual location service implementation with an energy-efficient alternative, we reduced the energy consumption by 10.59%.

c) Location Utility App [30]: this sample utility app provides a user interface that continuously reports user location change with address, speed of movement, direction and altitude. It also performs geocoding, i.e., converting latitude/-longitude coordinates to a user-friendly description. The app uses Standard Location Service with `desiredAccuracy` set to `Best` and `distanceFilter` set to `None`. This configuration represents S5 in our study.

To optimize this app, we replaced the current app S5 implementation with S12 because according to Figure 7a, S12 accesses more locations than S5 with less energy usage. As a result, by replacing the app's actual location service implementation with an energy-efficient alternative, we reduced the energy consumption by 26.91%.

d) Apple Sample App [27]: this is a demonstration code that tracks user location changes and draw a trail over the map as the user moves. The app uses Standard Location Service with `desiredAccuracy` set to `Best` and `distanceFilter` set to `None`. This configuration represents S5 in our study.

Similar to the location utility app, we replaced the current app S5 implementation with S12. As a result, we reduced the energy consumption by 11.37%.

VI. Threats to Validity

We ran all our experiments on a single smartphone device that runs on a single operating system. This setup restricts the generalizability of our results to a single device/operating-system configuration. Due to the expensive hardware costs that entail iOS measurement, we evaluate our benchmarks on one of the latest available device running iOS version 13.4.1. Although our absolute energy measurements are dependent on a single device configuration, we believe that the relative energy difference between the location service configurations should stay the same. This hypothesis requires further investigation which is out of the scope of this paper. Another threat is that our results can not be generalized on iPhone versions other than 11, or on iOS versions other than iOS 13.4.1. Our methodology for evaluating the core location framework, however, will stay relevant.

[2] https://github.com

[3] https://github.com/search?q=locationManager.start&type=code

Instead of physically moving the smartphone, we simulate the user location. This is because moving for 10 minutes around the city five times for each benchmark, while having 28 benchmarks, requires 23 hours of traveling along with the iGreenMiner hardware, which is a prohibitively expensive process. However, physical movement in such an experiment is important because users use their phone in a heterogeneous network environment, an environment that causes several disconnections between WiFi and GPS. Moreover, cellular tower distance, surface elevation and signal range variation may affect energy consumption. For example, in an urban city, a user experiences frequent context switches as compared to a rural area. This is because urban cities often have more cellular towers than rural areas. Unfortunately while conducting the experiments, our mobility was restricted due to the Covid-19 pandemic situation. However, our experimental setup retains the relative energy difference among the location service configurations. After the movement restriction ban has been lifted, we plan to conduct a physical experiment to get more accurate measurements.

VII. Related Work

Recent studies have shown that developers are unaware of which parts of their code consumes energy, how to measure the energy consumption, and how their code changes may affect energy consumption [31], [1], [32], [33], [34]. To help developers, several researchers have proposed energy measurement techniques while some have helped developers through recommendation guides and energy optimization techniques.

A. Energy Measurement

To help developers measure or predict energy consumption, prior work offer tools such as GreenMiner [16], a hardware-based tool that developers use through an open-source web service to measure the energy consumption of their Android apps. Cruz et al. [35] highlight the issue of error in the current energy prediction models and propose an automated framework that helps developers measure energy in case the prediction model is inaccurate. However, to this date, the proposed framework has not been implemented yet. Matalonga et al. [36] present energy datasets for further analysis. Other researchers have developed energy prediction models [37], [38], [39], [40], [41] to help the developers quickly estimate the energy consumption of their code and relieve the burden of managing energy measurement hardware.

Unfortunately, all these measurement frameworks and prediction models are limited to the Android platform. Hence these measurement techniques can not be used to find design choice or energy optimization techniques for the iOS developers. Keeping this limitation in context, and to provide guidelines for the iOS Core Location framework, we extend the GreenMiner framework [16] to support the iOS platform.

B. Recommendation Guidelines

There have been several empirical studies to help developers be aware of the development practices that affect energy consumption. Manotas et al. [42] introduce a decision-support framework for developers for making energy-efficient choices while using Java Collections API. Similarly, Hasan et al. [5] identify the effect of various Java collection classes on energy consumption and provide guidelines to the developers to reduce energy. Chowdhury et al. [4] introduce an architectural design pattern that developers may follow to reduce energy consumption. In a recent study, Oliveira et al. [43] have shown that different development approaches have different impact on the energy consumption of Android apps.

C. Optimization Techniques

Pinto el al. [44] investigated refactoring techniques that developers can adapt to reduce energy consumption. Pambola et al. [45] have conducted an empirical study to find out how code smells affect smartphone apps energy consumption and which code smells are the most energy-hungry. Mazuera et al. [46] have performed an extensive study on code commits of both Android and iOS apps to extract performance bugs. The author then propose a taxonomy that developers should follow to build tools for detection and optimization of energy bugs. Couto et al. [47] have statically analyzed Android apps to detect energy-greedy patterns and propose an automated refactoring technique to eradicate the detected patterns.

Our work is closest to the investigation done by Schuler et al. [48], in terms of providing an energy recommendation guide for developers. Schuler et al. [48] evaluate the impact of third-party framework calls on energy consumption. Contrary to our work, that study considers all calls to frameworks collectively instead of an in-depth investigation of a single framework. The study also does not provide any recommendation guidelines to developers.

VIII. Conclusion

Developers use the Core Location framework for accessing user location in their apps. However, developers are unaware of how their design choice for a particular Core Location framework service's configuration affects energy consumption. To better help developers, we have provided a set of guidelines for making an energy-efficient choice regarding the Core Location framework.

Our guidelines show that for frequent location updates, three Standard Location Service configurations (S10, S12, S15) are the most energy efficient. While for less frequent location updates, Regional Monitoring Service is the most energy-efficient.

Upon implementing our guidelines on 3 real-world apps, each belonging to a different category, we reduce up to 26.91% energy consumption. Developers may use these guidelines to develop energy auto-tuning tools. Moreover, our methodology may be used to evaluate utility frameworks other than location services such as Core graphics, Core ML, Core Animation, and Core Bluetooth. Finally, our iGreenMiner framework provides a web service that developers may use to measure and monitor the energy consumption of their iOS apps.

REFERENCES

[1] G. Pinto, F. Castor, and Y. D. Liu, "Mining questions about software energy consumption," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 22–31.

[2] G. Pinto and F. Castor, "Energy efficiency: a new concern for app software developers," *Communications of the ACM*, vol. 60, no. 12, pp. 68–75, 2017.

[3] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users complain about?" *IEEE software*, vol. 32, no. 3, pp. 70–77, 2014.

[4] S. A. Chowdhury, A. Hindle, R. Kazman, T. Shuto, K. Matsui, and Y. Kamei, "Greenbundle: an empirical study on the energy impact of bundled processing," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1107–1118.

[5] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle, "Energy profiles of java collections classes," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 225–236.

[6] C. Sahin, L. Pollock, and J. Clause, "From benchmarks to real apps: Exploring the energy impacts of performance-directed changes," *Journal of Systems and Software*, vol. 117, pp. 307–316, 2016.

[7] A. Puvvala, A. Dutta, R. Roy, and P. Seetharaman, "Mobile app developers' platform choice model," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*. IEEE, 2016, pp. 5721–5730.

[8] M. Fitzgerald, "Android vs. ios app development: Which is the better choice for your business in 2019?" Oct 2019. [Online]. Available: https://bit.ly/3nbOCkS

[9] AppleInc., "Improving Battery Life and Performance," https://developer.apple.com/videos/play/wwdc2019/417/?time=405, 2019, [WWDC].

[10] AppleInc, "Getting Users Location," https://developer.apple.com/documentation/corelocation/getting_the_user_s_location, 2020, [Online; accessed 17-Feb-2021].

[11] AppleInc., "Core Location Framework," https://developer.apple.com/documentation/corelocation, 2020, [Online; accessed 17-Feb-2021].

[12] AppleInc, "Standard Location Service," https://developer.apple.com/documentation/corelocation/getting_the_user_s_location/using_the_standard_location_service, 2020, [Online; accessed 17-Feb-2021].

[13] AppleInc., "Significant Location Service," https://developer.apple.com/documentation/corelocation/getting_the_user_s_location/using_the_significant-change_location_service, 2020, [Online; accessed 17-Feb-2021].

[14] AppleInc, "Visit Location Service," https://developer.apple.com/documentation/corelocation/getting_the_user_s_location/using_the_visits_location_service, 2020, [Online; accessed 17-Feb-2021].

[15] AppleInc., "Region Monitoring Service," https://developer.apple.com/documentation/corelocation/monitoring_the_user_s_proximity_to_geographic_regions, 2020, [Online; accessed 17-Feb-2021].

[16] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky, "Greenminer: A hardware based mining software repositories software energy consumption framework," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 12–21.

[17] AppleInc., "Xcode 11.3 Release Notes," https://developer.apple.com/documentation/xcode-release-notes/xcode-11_3-release-notes, 2021, [XCode11.3].

[18] ——, "AppleScript Language Guide," https://developer.apple.com/library/archive/documentation/AppleScript/Conceptual/AppleScriptLangGuide/introduction/ASLR_intro.html, 2021, [Online; accessed 17-Feb-2021].

[19] MonsoonSolutions, "High Voltage Power Monitor," https://www.msoon.com/high-voltage-power-monitor, 2020, [Online; accessed 17-Feb-2021].

[20] M. Dong, Y.-S. K. Choi, and L. Zhong, "Power modeling of graphical user interfaces on oled displays," in *2009 46th ACM/IEEE Design Automation Conference*. IEEE, 2009, pp. 652–657.

[21] W. Haynes, *Wilcoxon Rank Sum Test*. New York, NY: Springer New York, 2013, pp. 2354–2355. [Online]. Available: https://doi.org/10.1007/978-1-4419-9863-7_1185

[22] AppleInc., "Avoid Extraneous Graphics and Animations," https://tinyurl.com/appleenergy, 2021, [Graphics Guide].

[23] AppleInc, "Energy Guide iOS," https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/LocationBestPractices.html\#//apple_ref/doc/uid/TP40015243-CH24, 2020, [Online; accessed 17-Feb-2021].

[24] AppleInc., "Location and Maps Programming Guide," https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/LocationAwarenessPG/CoreLocation/CoreLocation.html\#//apple_ref/doc/uid/TP40009497-CH2-SW1, 2020, [Online; accessed 17-Feb-2021].

[25] D. Tiganov, J. Cho, K. Ali, and J. Dolby, "Swan: A static analysis framework for swift," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1640–1644.

[26] AppleInc., "iOS 14 features reimagined," https://www.apple.com/ca/ios/ios-14/, 2021.

[27] AppleInc, "Breadcrumb: Using CoreLocation to track user movement," https://developer.apple.com/library/archive/samplecode/Breadcrumb, 2018.

[28] N. Jaswal, "Property Search App," https://github.com/nitinjaswal96/CapstoneProjectPS, 2018.

[29] Oluwakemi, "Weather Access App," https://github.com/kemi-mabel/MyWeatherApp, 2021.

[30] Sumandeep, "Location Utility App," https://github.com/Sumandeep2111/FinalProject-ios, 2020.

[31] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about the energy consumption of software?" *PeerJ PrePrints*, vol. 3, p. e886v2, 2015.

[32] C. Wilke, S. Richly, S. Götz, C. Piechnick, and U. Aßmann, "Energy consumption and efficiency in mobile apps: A user feedback study," in *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 134–141.

[33] C. Zhang, A. Hindle, and D. M. German, "The impact of user choice on energy consumption," *IEEE software*, vol. 31, no. 3, pp. 69–75, 2014.

[34] H. Malik, P. Zhao, and M. Godfrey, "Going green: An exploratory analysis of energy-related questions," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 418–421.

[35] L. Cruz and R. Abreu, "Emaas: Energy measurements as a service for mobile apps," in *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 2019, pp. 101–104.

[36] H. Matalonga, B. Cabral, F. Castor, M. Couto, R. Pereira, S. M. de Sousa, and J. P. Fernandes, "Greenhub farmer: real-world data for android energy mining," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 171–175.

[37] S. A. Chowdhury and A. Hindle, "Greenoracle: Estimating software energy consumption with energy measurement corpora," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016, pp. 49–60.

[38] S. Hao, D. Li, W. G. Halfond, and R. Govindan, "Estimating mobile app energy consumption using program analysis," in *2013 35th international conference on software engineering (ICSE)*. IEEE, 2013, pp. 92–101.

[39] A. Carroll, G. Heiser *et al.*, "An analysis of power consumption in a smartphone." in *USENIX annual technical conference*, vol. 14. Boston, MA, 2010, pp. 21–21.

[40] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 168–178.

[41] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, and M. Kandemir, "Using complete machine simulation for software power estimation: The softwatt approach," in *Proceedings Eighth International Symposium on High Performance Computer Architecture*. IEEE, 2002, pp. 141–150.

[42] I. Manotas, L. Pollock, and J. Clause, "Seeds: A software engineer's energy-optimization decision support framework," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 503–514.

[43] W. Oliveira, R. Oliveira, and F. Castor, "A study on the energy consumption of android app development approaches," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 42–52.

[44] G. Pinto, F. Soares-Neto, and F. Castor, "Refactoring for energy efficiency: A reflection on the state of the art," in *2015 IEEE/ACM 4th International Workshop on Green and Sustainable Software*. IEEE, 2015, pp. 29–35.

[45] F. Palomba, D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, "On the impact of code smells on the energy consumption of mobile apps," *Information and Software Technology*, vol. 105, pp. 43–55, 2019.
[46] A. Mazuera-Rozo, C. Trubiani, M. Linares-Vásquez, and G. Bavota, "Investigating types and survivability of performance bugs in mobile apps," *Empirical Software Engineering*, pp. 1–43, 2020.
[47] M. Couto, J. Saraiva, and J. P. Fernandes, "Energy refactorings for android in the large and in the wild," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 217–228.
[48] A. Schuler and G. Anderst-Kotsis, "Characterizing energy consumption of third-party api libraries using api utilization profiles," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–11.