Electronic Preprint for Journal of Information Processing Vol.27

Regular Paper

Estimation of Power Consumption of Each Application Considering Software Dependency in Android

Shun Kurihara[1,a)] Shoki Fukuda[1] Takeshi Kamiyama[2] Akira Fukuda[2]
Masato Oguchi[3] Saneyasu Yamaguchi[1]

Received: June 30, 2018, Accepted: November 14, 2018

Abstract: Some reports stated that the most important issue of smartphones is their large battery consumption. Information on the power consumption of each application is important for users and administrations of application distributing sites. Especially, information on power consumption of each application in the screen-off state is important because understanding the behavior of an application in the state is difficult. Naturally, the power consumption of a device increases and decreases by installing and uninstalling an application, respectively. However, the sizes of increase and decrease in power consumptions depend on the device. We think there are two types of dependencies, which are hardware and software dependencies. The hardware dependency is that the power consumption of an application depends on the hardware elements of the device. The software dependency is that the power consumption of an application depends on the other applications installed on the device. We then argue that consideration of these dependencies are essential for estimation of the power consumption of each application. In this paper, we focus on the software dependency and propose a method for estimating the size of increase and decrease in power consumptions of the device by installing and uninstalling an application considering software dependency. The proposed method monitors starts and ends of functions such as GPS usage and WakeLock, then estimates the parts of the power consumptions of each application separately. We estimate the GPS usage time and WakeLock time for evaluation of the proposed method and show that the proposed method can estimate these more accurately than the standard method of the Android operating system. Our evaluation demonstrated that the proposed method decreased the difference between the estimated and actual sizes of decreases in power consumption by 89% at most.

Keywords: Android, WakeLock, GPS, power consumption, software dependency

1. Introduction

Estimating the battery consumption of each application individually is essential for users to save on power consumption, which is the most important problem in recent smartphones [1], [2]. The Android operating system, which is one of the most popular smartphone operating systems and has the largest market share [3], has a function whereby the operating system can invoke an application at the specified time without user operation. This function allows applications to run in the screen-off state and consume battery power. In order to reduce battery consumption by uninstalling a non-important application, understanding the power consumption of each application is required. However, the sizes of the increased and decreased power consumption by installing and uninstalling an application depends on the device [4], [5]. As far as we know, there is no method for obtaining this information.

There are two types of dependencies, caused by hardware and software. Software dependency implies that the power consumption of an application depends on the other applications installed on the device.

[1] Kogakuin University, Shibuya, Tokyo 150–0011, Japan
[2] Kyushu University, Fukuoka 819–0395, Japan
[3] Ochanomizu University, Bunkyo, Tokyo 112–8610, Japan
[a)] sane@cc.kogakuin.ac.jp

The Android operating system estimates the power consumption of each application individually based on the monitored times of various factors such as WakeLock. In addition, it was demonstrated that execution of WakeLock and power consumption have a strong relationship [6]. Therefore, in order to accurately estimate the power consumption of each application, it is essential to improve the accuracy of the monitoring and estimating times considering the dependency. We propose a method for estimating the power consumption with consideration of software dependency [4], [5] only for applications which use GPS. The method was effective for applications using GPS. However, the effectiveness of other factors, i.e., WakeLock, CPU, and Wi-Fi, was not discussed.

We define the software dependency as follows. The size of decreased power consumption by uninstalling an application depends on the other applications that are installed in the device. We call this dependency software dependency. The size of power consumption decreased by uninstalling an application that utilizes a function, e.g., GPS, is large if the other applications do not utilize the function. It is small if some other applications utilize the function and the periods of usages overlap. In case of the number of application that uses a certain function at a certain period is changed from one to zero by uninstallation, the usage of the function at the period changes from *busy* to *idle* and the power

consumption decreases. In case of the number of applications that use the function changes from $n + 1$ to n such that $n > 0$, the usage of the function does not change, i.e., from *busy* to *busy*, and the consumption does not decrease.

In this paper, we focus on the software dependency and propose a method for estimating the sizes of increased and decreased power consumptions of a device by installing and uninstalling an application considering software dependency. We then estimate the GPS usage time and WakeLock time for evaluation of the proposed method and show that the proposed method can estimate these more accurately than the standard method of the Android operating system. We consider the software dependency with examples of GPS and WakeLock in this paper. It is mainly because these usages are important for discussing power consumption [4], [7]. This paper is an extended version of the previously published work of Refs. [4], [5], [7], [8].

The rest of this paper is organized as follows. Section 2 reviews the related studies. Section 3 explains power consumption in the screen-off state in Android and its estimation. Section 4 describes the proposed method for estimating power consumption with consideration of software dependency. Section 5 evaluates the proposed method. Section 6 discusses furthers improvement. Section 7 presents discussion. Finally, Section 8 concludes the paper.

2. Related Work

Power Consumption Estimation in Android: Many studies on power consumption estimation have been published. Corral et al. proposed a method for estimating power consumption using the information provided by the operating system [9]. Kaneda et al. modeled the power consumption of Wi-Fi devices. Motlhabi et al. estimated the power consumption of Wi-Fi devices [10]. Murmuria et al. also modeled the power consumption of Wi-Fi devices [11]. Friedman et al. discussed effective data transmission using Wi-Fi and showed that retransmission was ineffective with regard to power consumption [12]. In the work of Refs. [13], [15], methods for estimating the break-even time of disabling a network device were proposed. These studies did not provide a method for estimating the power consumption of each application.

The Android operating system estimates the power consumption of each application by summing the monitored times of GPS usage, WakeLock, Wi-Fi usage, CPU usage, and others. Therefore, improving the accuracy of monitoring times is essential to improving the estimation of the power consumption of each application. A method for improving the accuracy of monitoring the time of GPS usage was presented in the work of Ref. [4].

Static Estimation of Power Consumption: The following studies deal with the power consumption estimation of the static analysis of applications. Wu et al. tried to detect energy-drain defects in Android applications [16]. Singhai et al. proposed software-based methods for reducing the power consumption of a mobile device running an Android application [17]. Couto et al. presented a tool for monitoring and analyzing power consumption in the Android ecosystem [18]. The tool instrumented the source code of a given Android application and estimated the power consumed by the application. Bao et al. analyzed Android applications based on source code [19]. These studies are effective from the aspect of static analyses.

Dynamic Estimation of Power Consumption: Zhang et al. proposed an automated power model construction technique that used built-in battery voltage sensors and knowledge of battery discharge behavior to monitor power consumption while explicitly controlling the power management and activity states of individual components [20]. Mittal proposed a method that allowed developers to estimate the energy consumption for applications on their development workstations [21]. This method scaled the emulated resources including the processing speed and network characteristics to match the application behavior to that on a real mobile device.

Alessio et al. investigated the feasibility of constructing power-consumption-based sensors for the identification of security threats (for example, battery-drain attacks) on Android-based mobile devices [22]. Furusho et al. proposed a method for collecting essential data to profile the energy consumption of applications running on the Android operating system [23]. Their method analyzed the power consumption using a log collected during application usage of a particular user. In the work of Ref. [24], a method for monitoring application launching behavior was proposed. These studies are effective with regard to dynamic power consumption estimation. However, there is no method for estimation considering software dependency.

Software Dependency of Power Consumption: The software dependency of power consumption is discussed in Refs. [4], [6], [7], [8], [25], [26]. The works of Refs. [25], [26] focused on the relationship between broadcast intent [27], [28] and power consumption. Broadcast intent is issued by some applications and received by some other applications. Naturally, the number of issued broadcast intents and receiver applications are expected to depend on the installed applications. Broadcast intents are also issued in the screen-off state. Receiver applications are invoked at reception and consume power. This implies that power consumption may have a dependency on installed applications.

These works investigated the relationship between the number of receiving applications, power consumption per broadcast intent issue, and installed applications. These studies then showed that consumption per issue depended on the installed applications. In addition, these studies revealed that the number of issues was also dependent on applications installed in the device.

These works argued only that there existed a software dependency and did not propose something novel for improving power consumption estimation. In the work of Ref. [4], a method for estimating the power consumption considering software dependency was discussed.

This paper is based on previous works [4], [5]. However, these works did not present discussion on the power consumption caused by WakeLock. The work of Ref. [6] focused on the estimation of power consumption caused by WakeLock. However, the proposed method in this previous work did not take account of software dependency. In the work of Ref. [7], the method of Ref. [4] was applied to WakeLock function and the method was evaluated with benchmark applications. In the work of Ref. [8],

the method was evaluated with practical applications.

In the work of Ref. [29], smartphone usage models based on investigation of practical users usage, were introduced. The authors analyzed the feedback data from practical users and logs in practical used devices and showed the relationship between the trends and the users properties.

3. Power Consumption in Screen-off State in Android and its Estimation

3.1 Power Consumption in Screen-off State in Android Device

An Android device in the screen-off state goes into the sleep state without the user's operation. In the sleep state, CPU processing, display emitting, and communication are highly restricted, and power consumption is significantly decreased. The Android operating system can invoke an application using alarm and broadcast intent even in the screen-off state [6].

This restriction in the sleep state may prevent an application from working correctly [13], [15]. In order to avoid this problem, the Android operating system has a WakeLock function that forbids a device to go into the sleep state. For example, a movie application can keep its screen on during play with this function. The WakeLock function has several modes for choosing locked functions [14].

Naturally, the execution of WakeLock increases the power consumption of the device [6]. Furthermore, a user cannot observe the behavior of the application directly in the screen-off state. Thus, it is not easy for the user to monitor requests, and this removes an application that frequently executes WakeLock. To understand the battery consumption of each application, obtaining accurate information on the WakeLock time of each application is important.

3.2 Estimation of Power Consumption of Each Application in Android Operating System

The Android operating system has a standard method that monitors and manages the power consumption of each application. A user can access this function via the *Battery* function of the *Setting* application. We refer to this as *the standard method* in this paper. The Android operating system, including this standard application and the Android framework, monitors starts and ends of the usage of its devices such as its CPU and GPS, and executions of WakeLock. The standard application manages these times inside and then calculates the power consumption of each application.

An application can make a function, e.g., GPS and WakeLock, active or inactive by using APIs. It can be utilized while it is active. In a period of usage, its power consumption increases. In the case of GPS, the APIs are *acquire* and *release*, and the location information can be obtained while it is active.

The WakeLock time of each application is summed only in the screen-off state. Accurately monitoring these times is important for the accurate estimation of the power consumption of each application.

4. Proposed Method

In this section, we propose a method for estimating the time of usage of a function or a device of each application. The method monitors starts and ends of usage by each application and estimates the usage time after uninstalling an application on a basis of the monitored usage history considering overlapping.

The proposed method estimates the usage time of a target function after uninstalling each application as the following three steps.

(1) A user modifies the implementation of the Android operating system for monitoring starts and ends of usages of a target function.

(2) The user let the device run with the modified operating system with its applications. As a result, the *history* of starts and ends of the target function is obtained. A period in which one or more application(s) uses the target function is a period of usage. A period in which no application uses the function is a period of non-usage.

(3) The user removes the starts and ends by a target application from the history, we call this a *modified history*, for estimating the usage time of the target function after uninstalling the target application. Similar to the second step, the usage time of the target function can be predicted from this modified history.

This method modifies the implementation of the Android operating system in order to monitor starts and ends of usages of functions or devices. In a case of GPS, a monitoring function is inserted into the source code of implementations of *request* and *remove* of the *LocationManager*. In a case of WakeLock, the monitoring function is inserted into these of *acquire* and *release* of WakeLock. Monitoring of request and remove of GPS can be achieved by modifying `LocationManagerService.java` in Android operating system. Monitoring of WakeLock can be achieved by modifying `BatteryStatsImpl.java`. We monitored partial WakeLock. The Android operating system associates every application with a user ID in the Linux kernel. Thus, the application that issues a start or an end of a usage is easily identified by checking the process that issues it. All a user have to do is to obtain the time and the information of the process of each issue and store them, e.g. saving them to a file.

Figure 1 illustrates the proposed method. Figure 1 (a) is a sample of a recorded history. It contains the time and the application information. Figure 1 (b) is the visualized record. The function is used in the periods of 2–8, 10–12, and 14–20. The total usage time is 14. In this case, the proposed method estimates the usage time of the function after uninstalling each application as follows. The usage time after uninstalling application 1010 is obtained by removing the usages by application 1010 from the history. The modified history is shown in Fig. 1 (c). The estimated usage periods are 4–6, 10–12, and 16–18. The estimated total usage time is 6. The total usage time decreases by 8, i.e., from 14 to 6, by uninstalling the application 1010. The usage time after uninstalling application 1011 is 14 as shown in Fig. 1 (d). This means that uninstalling the application 1011 does not decrease the total usage time. The total usage time after uninstalling the application

1012 is 12 and the size of decrease is 2.

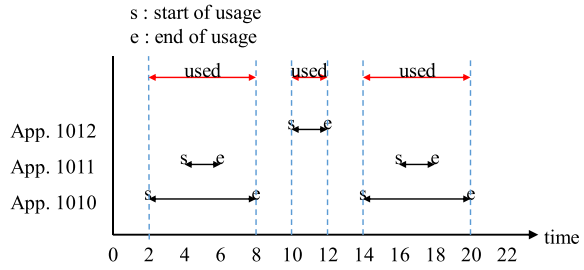
The letters "s" and "e" indicate *start* and *end* of usage, respectively. In the case of GPS, these are *request* and *remove* of GPS. In the case of WakeLock, these are *acquire* and *release* of WakeLock.

Unfortunately, the standard method of Android estimates the usage time after uninstalling without considering overlapping. If multiple applications share a usage, the time is divided by the number of the sharing applications and each divided time is allocated to every application. In the case of Fig. 1 (b), the usage time between 4 and 6 is shared by the applications 1010 and 1011. The usage time (2 s) is divided by the number of applications (two), and then the divided time (1 s) is allocated to both applications. Similarly, the 2 s of usage time between16 and 18 is divided into both applications. As a result, the standard method estimates the usage time of the application 1011 as 2 s. Then, the method predicts that the usage time decreases by 2 s, from 14 to 12, by uninstalling the application 1011. However, the usage time does not decrease by uninstalling the application 1011.

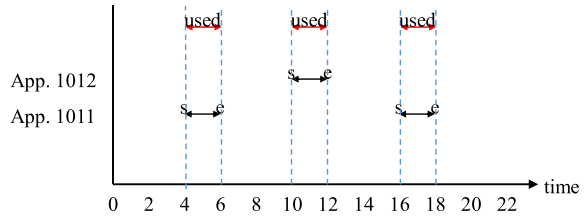
The further improvement considering the software dependency including system processes is discussed in Section 6.

time 2, start of usage by application 1010.
time 4, start of usage by application 1011.
time 6, end of usage by application 1011.
time 8, end of usage by application 1010.
time 10, start of usage by application 1012.
time 12, end of usage by application 1012.
time 14, start of usage by application 1010.
time 16, start of usage by application 1011.
time 18, end of usage by application 1011.
time 20, end of usage by application 1010.

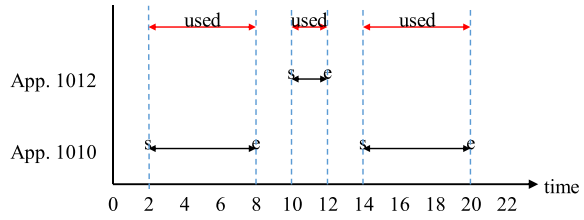
(a) a recorded history of starts and ends of usages



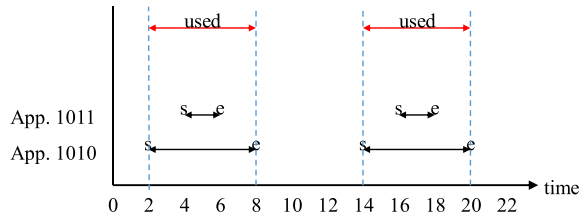
(b) visualized history



(c) visualized history (without App. 1010)



(d) visualized history (without App. 1011)



(e) visualized history (without App. 1012)

Fig. 1 Concurrent usages of functions by applications.

5. Evaluation

5.1 Benchmark GPS Application

Here, we evaluate the proposed method using a benchmark application that we implemented. The application sends *request* and *remove* of GPS for LocationManager at start and 2 h later, respectively. We performed four experiments, as shown in **Fig. 2**. In the case of Type O, we invoked no application and left the device untouched for 2 h. We then measured the power consumption and the GPS time. For Type A and Type B, we used applications A and B, respectively.

For Type AB, we used applications A and B simultaneously. Applications A and B are completely the same. The used device was a Nexus 7 (2013), which has the CPU Qualcomm Snapdragon S4 Pro 1.5 GHz, 2 GB of memory, and Android operating system 6.0.1 with a patch for the proposed method. The proposed method estimated the power consumption with an assumption that power consumption was proportional to the usage time in all the experiment in this section.

The experimental results are shown in **Figs. 3** to **6**. The vertical axis in Fig. 3 represents the actual size of the decrease in the battery of the device in 2 h. That in Fig. 4 represents the estimated size of the battery consumed by each application, which is provided by the standard method of the Android operating system.

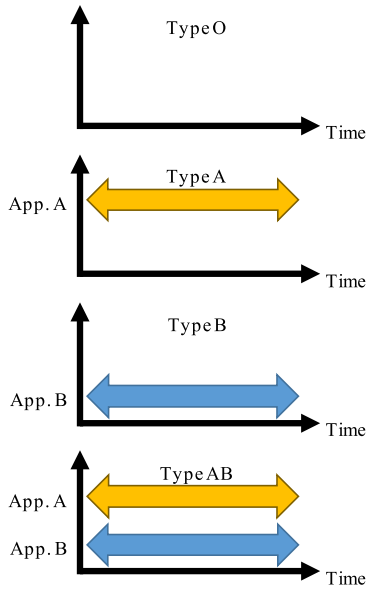


Fig. 2 The types of the experiments (benchmark application).

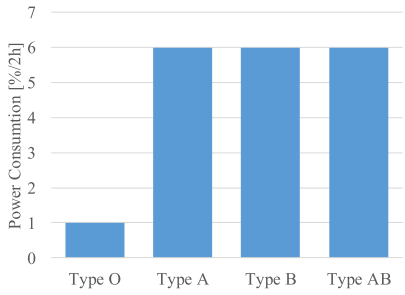


Fig. 3 Experimental results (power consumption, benchmark application).

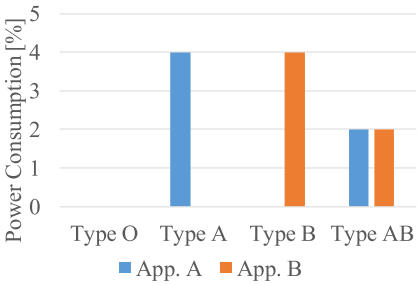


Fig. 4 Experimental results (power consumption of each application, benchmark application).

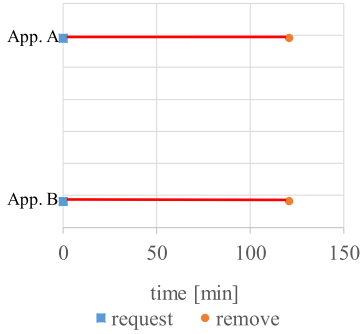


Fig. 5 Monitored GPS usage with the proposed method (benchmark application).

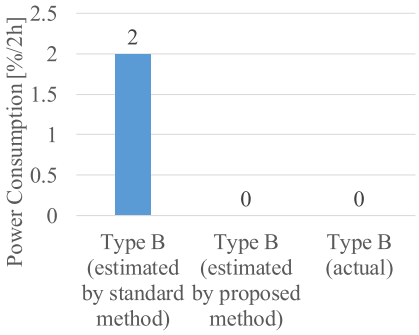


Fig. 6 Estimated and actual power consumption after uninstallation of application A (benchmark application).

Figure 5 depicts the periods of GPS usage by Applications A and B. Figure 6 shows the estimated and actual sizes of the decreased power consumption by uninstalling Application A from the Type AB. "Type B (estimated by standard method)" represents the estimated size by the standard method. The size is 2% as shown in Fig. 4. "Type B (estimated by proposed method)" represents the size estimated by the proposed method. The size is 0% because the GPS usage time will not be decreased by installing the Application A as shown in Fig. 5. "Type B (actual)" represents the measured size, which is the actual value. Therefore,

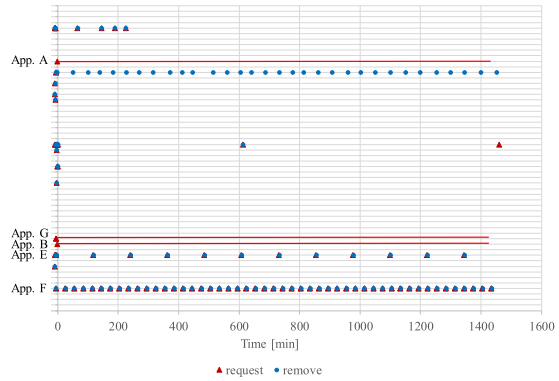


Fig. 7 Monitored GPS usage with the proposed method (practical applications).

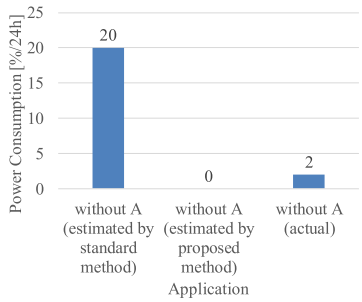


Fig. 8 Estimated and actual power consumption after uninstallation of application A (practical applications).

Table 1 Power consumption of application by standard method.

Application	Power Consumption [%/24h]
A	20
B	20
C	2
D	1
E	1
F	1

the actual decreasing power consumption is 0%. This indicates that the proposed method can estimate more accurately than the standard method in the case of this benchmark application.

5.2 Practical GPS Application

Here, we evaluate the proposed method with practical GPS applications. We use the top 50 applications in the application ranking in the category "Weather" in the Google Play Store on June 30, 2016.

Because five of those applications did not support the experimental device, 45 applications were installed on the device. We left the device untouched for 24 h and measured its power consumption. The used device was the same as that of the previous subsection.

Figures 7, **8**, and **Table 1** show the experimental results. Table 1 shows the ranking of the power consumption of each application reported by the standard method. Figure 7 shows the periods of GPS usage presented by the proposed method. Figure 8 shows the estimated and actual sizes of the decreased power consumption by uninstalling Application A, which is the top

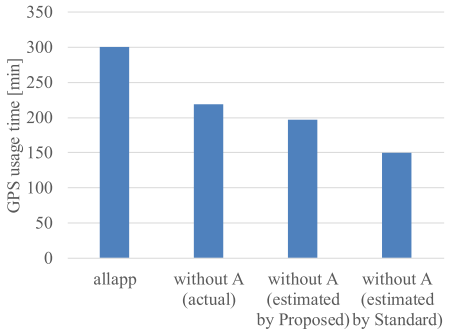


Fig. 9 Estimated and actual GPS time after uninstallation of application A (practical applications non-always GPS).

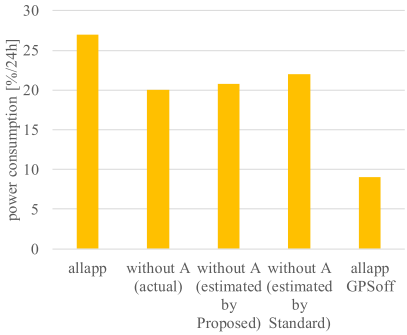


Fig. 10 Estimated and actual power consumption after uninstallation of application A (practical applications non-always GPS).

application in the ranking in Table 1. As shown in the table, the size of reduction estimated by the standard method is 20%. The total GPS time before uninstalling application A is 1,440 minutes as shown in Fig. 7. The total GPS time after uninstallation can be estimated from Fig. 7 by deleting the GPS usage of the application A and the estimated total GPS usage time is also 1,440 minutes. Thus, the size of reduction estimated by the proposed method is 0%. We actually uninstalled the application and measured the size of reduction. The values was 2%. The difference between the actual and estimated values decreased by 89%.

This result also demonstrates that the proposed method can estimate more accurately than the standard method.

5.3 Practical GPS Applications (Non-always GPS)

Here, we evaluate the proposed method using appllications that do not keep using GPS. In the evaluation in the previous subsection, two applications kept using GPS. In that case, it was easy for the proposed method to predict the time of GPS usage after uninstallation. We installed the top 50 applications in the free application ranking of the "Weather" category on July 1, 2016, in Google Play Store, except for five applications that the experimental terminal did not support and two applications that kept using GPS. Thus, 43 applications were installed. We left the terminal untouched for 24 h and measured the transition of the value of the remaining battery power. The used terminal was the same as that in Sections 5.1 and 5.2.

The power consumption ranking provided by the standard method is as follows. Applications A, B, C, and D consumed the battery by 5%, 2%, 1%, and 1%, respectively. **Figure 9** shows the estimated and actual GPS times before and after uninstallation. **Figure 10** shows the estimated and actual power consumptions.

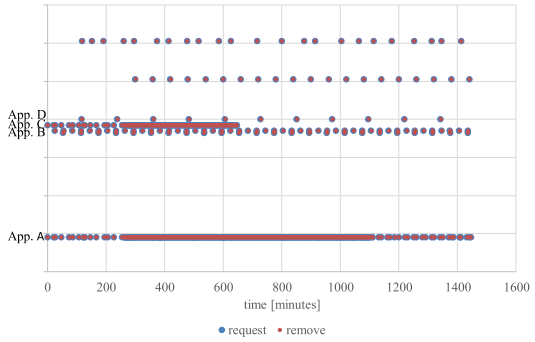


Fig. 11 GPS usage time for each application by proposed method.

The label *allapp* in the figure represents the experiments in which all 43 applications were installed, i.e., before uninstallation. The label *without A* indicates experiments with the 42 applications. The application A was uninstalled from all 43 applications, i.e., after uninstallation.

Application A is the top application in the ranking by the standard method. The label *estimated by proposed* represents the value estimated by the proposed method before uninstalling application A. The label *estimated by standard* indicates the value reported by the standard method. The label *allapp GPSoff* indicates the experiments with all 43 applications and GPS disabled. **Figure 11** shows the period of GPS usage of each application estimated by the proposed method. The *request* and *remove* labels in the figure indicate the times at which each application issued commands for starting and ending the usage of GPS.

The standard and proposed methods estimated the size of decrease in power consumption as 5% and 6.2%, respectively. The former is easily obtained. The ranking by the standard method implies that uninstallation of the application decreases the power consumption by 5%, as mentioned above. The latter is as follows. From Fig. 9, the proposed method estimates that the total GPS usage time in the device is 300 min, which was obtained from Fig. 11. In addition, the proposed method predicts the GPS usage time after uninstalling application A as 197 min. That is, the method predicts that uninstallation decreases the GPS time by 33%.

Comparing the values of *allapp* and *allapp GPSoff* indicates that the power consumed by the GPS is 18%. Uninstallation changes this from 18% into $18\% \times (197/300) = 11.8\%$. In other words, the proposed method predicts that uninstallation will decrease the consumption by 6.2%. Figure 10 shows that the actual size of the decreased power consumption was 7%. Therefore, we can conclude that the proposed method can predict the power consumption more accurately.

5.4 Benchmark WakeLock Application

Here, we evaluate the proposed method using a benchmark application that executes WakeLock.

The Android operating system manages the WakeLock time of each application in the standard *setting* application. We modified the source code of the application, which is an open-source application, so that it outputs the WakeLock times managed in it. We then assumed that uninstalling the application decreased the WakeLock time by this value. For example, if application A's

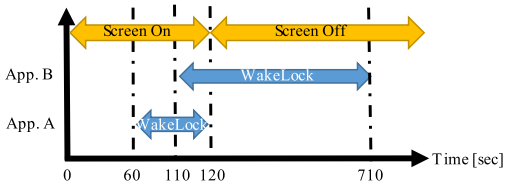


Fig. 12 The behavior of the applications for the evaluation.

WakeLock time that the standard application manages is w s, we assume that uninstalling application A decreases the total WakeLock time of the device by w s. Exactly writing, this value is not a value for prediction of GPS usage time but for estimation of the power consumption of each application. However, the Android operating system provides no information on GPS usage time. Thus, we compare this value with the GPS usage time predicted by the proposed method for discussion.

We also measured the actual decreased WakeLock time when uninstalling the application by comparing the total WakeLock times before and after uninstalling the application. The time of each measurement was 30 min. The display went to the screen-off state 2 min after being untouched. **Figure 12** illustrates the behaviors of the benchmark applications and the screen. Applications A and B are applications that we implemented for this evaluation. Both of these applications were invoked at almost the same time. These applications went into the background mode just after their invocation.

We define the time at which these applications went to the background state as time 0. Application A requested 60 s of WakeLock at time 60, i.e., WakeLock from 60 s to 120 s.

Application B requested 10 min of WakeLock at 110 s, i.e., WakeLock from 110 s to 710 s. The display went to the screen-off state at 120 s. Even though the screen went to the off state, the device and CPU stayed awake during WakeLock. The modes of all the WakeLocks are PARTIAL_WAKE_LOCK [14].

All applications installed on the device are Android Open Source Project (AOSP) standard applications. The used device is the same, i.e., the Nexus 7 (2013). The operating system is Android 6.0.1 with our modification described above. In order to obtain the measured WakeLock time of each application, we modified the implementation of the `calculateApp()` method and the `calculateRemaining()` method in `WakelockPowerCalculator.java` in the Android operating system. These methods are for calculating the battery consumption of each application caused by WakeLock.

The experimental results are depicted in **Fig. 13**. The results with the label $A + B$ (*measured*) shows the total WakeLock time in the device calculated in the Android operating system and reported for both applications A and B. That with $A + B$ (*estimated by proposed*) is the WakeLock time estimated by the proposed method according to the recorded starts and ends of WakeLock. That with A (*estimated by proposed*) shows the WakeLock time after uninstallation of application B as estimated by the proposed method. That with A (*estimated by standard*) shows the WakeLock time after uninstalling as estimated by the standard method. We obtained this from the report of the modified *setting* application. The data with label A (*actually*) represent the

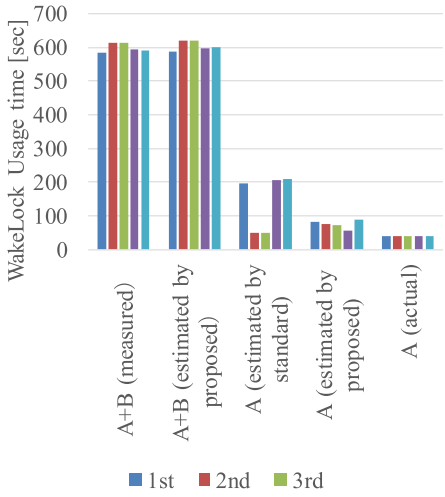


Fig. 13 The experimental results.

Table 2 Target applications (1/2).

Scenario 1	Scenario 2	Scenario 3
Life Style 1	Mail	Game 1
☆ Photo 1	SNS 1	SNS 2
	SNS 2	Photo 2
	☆ Security	Health
	Browser	☆ Security

Table 3 Target applications (2/2).

Scenario 4	Scenario 5	Scenario 6
SNS2	Mail	Life Style
SNS3	SNS 2	Life Style 3
☆ Mail	Game 2	Tool 3
Tool	Game 3	Tool 4
Health	Game 4	Browser
	communication	☆ Security
	Life Style 2	
	Tool	
	☆ Tool 2	
	Media	

actual WakeLock time afterward. This was obtained by actual measurement.

Comparing the results of A (*estimated by standard*) and A (*estimated by proposed*), we can see that the proposed method always provided fine estimated results, while the standard method did not always do so. The average difference between the actual time and the estimated time by the proposed method is 37 s, while that of the standard method is 104 s. From these results, we conclude that the proposed method improved the accuracy of the estimation of the Android operating system.

5.5 Practical WakeLock Applications

Here, we evaluate the proposed methods using sets of applications on a basis of practical application usage models [29]. We installed the applications in a set into the same device, which was Nexus 7, and left the device untouched 24 hours. Every application set is described in **Tables 2** and **3**. Both of the standard

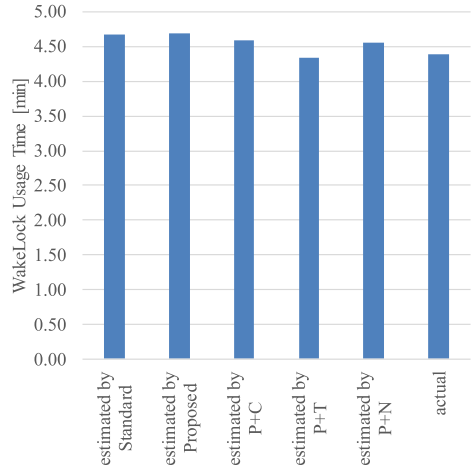


Fig. 14 Estimated and actual WakeLock times after uninstallation (Scenario 1).

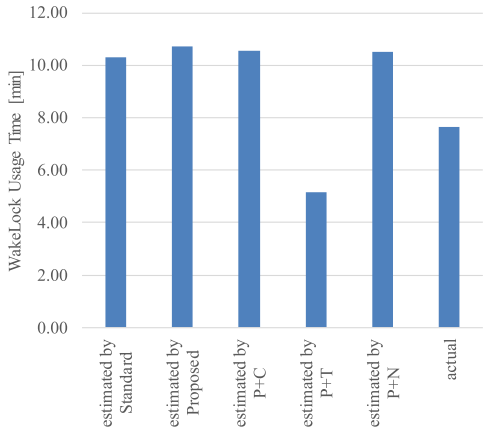


Fig. 15 Estimated and actual WakeLock times after uninstallation (Scenario 2).

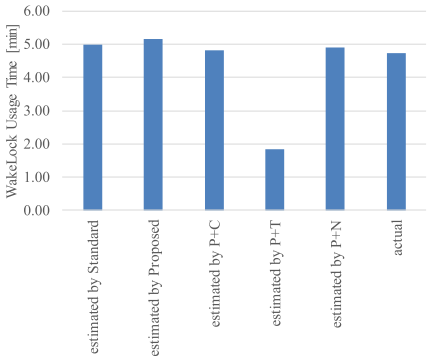


Fig. 16 Estimated and actual WakeLock times after uninstallation (Scenario 3).

and proposed methods estimated the WakeLock time after uninstalling the target application in the set. The target applications were the most battery-consuming applications reported by the standard *setting* application and are marked with star mark ☆ in the tables. We then measured the actual WakeLock times after uninstalling the application and compared the estimated and actual values.

The estimated and actual WakeLock times after uninstalling the target application are shown in **Figs. 14** to **19**. The data with the labels *estimated by proposed* and *estimated by standard* indicate the estimated values by the standard and proposed methods,

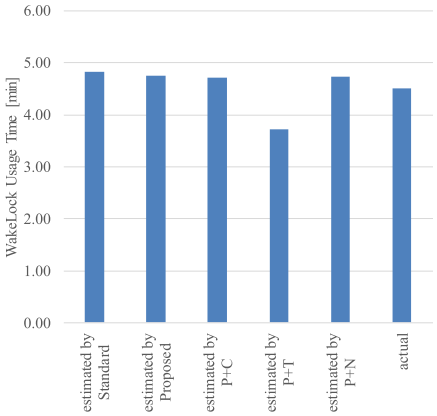


Fig. 17 Estimated and actual WakeLock times after uninstallation (Scenario 4).

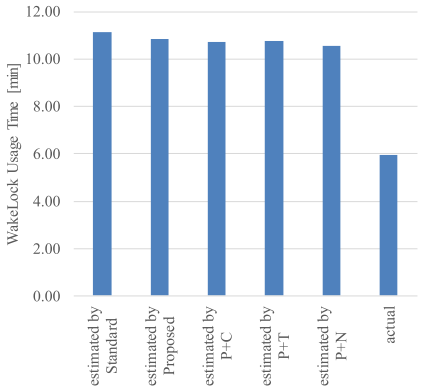


Fig. 18 Estimated and actual WakeLock times after uninstallation (Scenario 5).

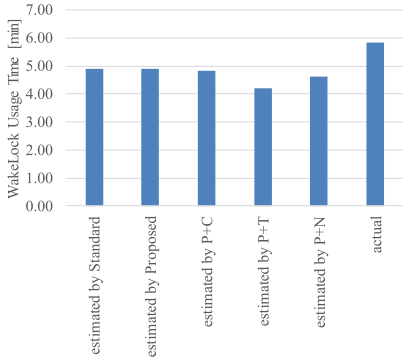


Fig. 19 Estimated and actual WakeLock times after uninstallation (Scenario 6).

respectively. The data with the label *actual* indicates the actual WakeLock time obtained by measuring. The other data, such as data with the label *estimated by* $P + C$ will be mentioned in the next section.

Comparing the estimated values by the standard and proposed methods, we can see that their accuracies are almost same. We will discuss further improvement in the next section. In the case of Scenario 6, the WakeLock time was increased by uninstalling the application. We think this result is contrary to intuition and will discuss it in the next section.

6. Further Improvement

6.1 Software Dependency of a System Process

First, we discuss the reason why the proposed method did not

Table 4 The time and times of WakeLock in 24 hours by GMS process application sets in Section 5.5 before and after uninstallation.

	Before uninstallation		After uninstallation	
	time [s]	times	time [s]	times
Scenario 1	238.1	3450	223.3	3252
Scenario 2	480.8	3925	289.7	3866
Scenario 3	259.5	3515	236.7	3139
Scenario 4	240.7	3274	214.2	3055
Scenario 5	377.4	4986	260.8	3646
Scenario 6	241.9	3356	284.2	3846

largely outperform the standard method in cases of the practical sets of applications. WakeLock is executed by not only application processes but also system processes. Especially, the Google Mobile Service (GMS) process executes WakeLock frequently. GMS process provides many essential applications, functions, and services of Google Play Services, such as Google location. An application can use these functions via GMS process and the process is invoked on receiving a request. The operating system recognizes that the GMS process issued WakeLock in case of WakeLock is executed by GMS process. However, many of WakeLock executions by the GMS process are caused by an application. The process rarely issues WakeLock without a request by an application. Therefore, uninstallation of an application removes or decreases WakeLocks by the application process and by the GMS process. In other words, the number of WakeLocks by the GMS process also has a dependency on installed software. We argue that accurate estimation of time of WakeLocks requires consideration of the software dependency of WakeLocks by GMS.

Table 4 shows the time and times of WakeLocks in 24 hours by the GMS process with application sets in Section 5.5 before and after uninstallation. The table implies that uninstallation of an application changes the time and times of WakeLocks by the GMS process. These results support our hypothesis that accurate estimation of WakeLock requires consideration of software dependency of WakeLock of the GMS process.

In the case of the Scenarios 1 to 5, the number and total time of WakeLocks by the GMS process decreased by uninstalling the application. These results are naturally understood. In the case of Scenario 6, those of WakeLocks increased by the uninstallation. This is contrary to intuition, as we described. In the following subsections, we propose methods for improving estimation of WakeLock considering software dependency of GMS.

6.2 Chaining

Here, we proposed a method for improving the estimation with *Chaining*. The method deletes the GMS's WakeLocks that started during a WakeLock execution by the target application from the WakeLock history in addition to the WakeLocks by the application. The estimated WakeLock times by this method are depicted in Figs. 14 to 19 as *estimated by* $P + C$. "P" and "C" mean "Proposed method" and "Chaining," respectively. P+C means "Proposed method and Chaining." Similar to the previous section, the proposed method estimated the power consumption with an assumption that power consumption was proportional to the usage time in all the experiment in this section. The improved method P+C estimated moderately more accurately than the standard method in the Scenarios 1, 3, 4, and 5.

6.3 Proportional Distribution by Time

In this subsection, we propose a method with proportional distribution by time. This method deletes the GMS's WakeLocks from the history with the probability p_t in addition to the WakeLocks by the target application. p_t is the ratio of the total time of WakeLocks by the uninstalled application divided by the total WakeLocks time by all the applications including GMS process. The estimated results by this methods are depicted in Figs. 14 to 19 as *estimated by* $P+T$. "T" means "Proportional distribution by Time." We can see that the improved method could estimate the time very similar to the actual one in the case of Scenario 1 and that its accuracy was worse than the other methods in the case of Scenario 3.

6.4 Proportional Distribution by Number

Here, we propose a method with proportional distribution by the number of WakeLocks. This method deletes the GMS's WakeLocks from the history with the probability p_n in addition to the WakeLocks by the target application. p_n is the ratio of the number of WakeLocks by the uninstalled application divided by the number of WakeLocks by all the applications including GMS process. The estimated results by this methods are depicted in Figs. 14 to 19 as *estimated by* $P + N$. "T" means "Proportional distribution by Number." We can see that the improved method could estimate the WakeLock time moderately more accurate in the Scenarios 1, 3, 4, and 5.

6.5 Discussion on Improvement

From the evaluation in Section 5 and Sections 6.2 to 6.4, we can say as follows. The proposed method in Section 4 can achieve better accuracy than the standard method in most cases. The improving methods in this section improve the accuracy more in most cases. Even though the best improving method depends on applications, these methods improved the accuracy of the proposed method in Section 4. Therefore, we can conclude that consideration of software dependency is effective.

However, these methods only predict the size of decrease in WakeLock times by uninstallation based on the start times of WakeLock or the statistical values. We think that monitoring requests to the GMS process from applications processes is effective for improving this accuracy. Understanding the relationship between WakeLocks by the GMS process and their causing applications must useful for choosing WakeLocks that will disappear after uninstallation.

6.6 Estimation for Installation

To predict increased GPS time and power consumption after installation of an application, an application behavior log should be included in the system before installation. Given this log, our system can estimate them without installation.

7. Discussion

In this section, we discuss applications of the proposed method. Naturally, accurate information, considering software dependency, on the sizes of increased and decreased power consumption by installing and uninstalling an application of each application is useful for saving on power consumption. A user can decide to uninstall or not an application with this information.

In addition, this information is profitable for managers of application distributing sites. As far as we know, no distributing site provides information on the power consumption of applications. One of the reasons is that they cannot determine its power consumption is large or small due to the software dependency. They can conclude that the consumption is large or small with the proposed method and the list of the installed application.

8. Conclusion

In this paper, we introduced power consumption's dependency on software and proposed a method for estimating the size of the decrease in power consumption by uninstalling each application considering this dependency. The method monitors starts and ends of usages of each function and estimates the power consumption after uninstalling an application based on this record. In addition, we proposed an improving method based on GMS's software dependency for estimating WakeLock times. Our evaluation demonstrated that the proposed method could estimate GPS and WakeLock times more accurately than the standard method of Android operating system for estimating power consumption. The proposed method decreased the difference between the actual and estimated sized of decreasing power consumption by 89% at most.

For future work, we plan to evaluate our method with a variety of applications and terminals.

Acknowledgments This work was supported by JSPS KAKENHI Grant Numbers 15H02696, 17K00109, and 18K11277. This work was supported by JST CREST Grant Number JPMJCR1503, Japan.

References

[1] Nihon Keizai Shimbun (2013) (in Japanese), available from ⟨http://www.nikkei.com/article/DGXNASFK2600W_W3A320C1000000/⟩.

[2] Nagata, K., Yamaguchi, S. and Ogawa, H.: A Power Saving Method with Consideration of Performance in Android Terminals, *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing* (*IEEE ATC 2012*), pp.578–585, DOI: 10.1109/UIC-ATC.2012.133 (2012).

[3] IDC, Smartphone OS Market Share, available from ⟨http://www.idc.com/promo/smartphone-market-share/os⟩.

[4] Kurihara, S., Fukuda, S., Hamanaka, S., Oguchi, M. and Yamaguchi, S.: Application power consumption estimation considering software dependency in Android, *Proc. 11th International Conference on Ubiquitous Information Management and Communication* (*IMCOM '17*), Article 86, 6 pages, ACM, DOI: 10.1145/3022227.3022312 (2018).

[5] Kurihara, S., Fukuda, S., Yamaguchi, S. and Oguchi, M.: Estimation of power consumption of each application based on software dependency in android, *2017 IEEE 6th Global Conference on Consumer Electronics* (*GCCE*), pp.1–2, DOI: 10.1109/GCCE.2017.8229436 (2017).

[6] Kurihara, S., Fukuda, S., Koyanagi, A., Kubota, A., Nakarai, A., Oguchi, M. and Yamaguchi, S.: A Study on Identifying Battery-Draining Android Applications in Screen-Off State, *2015 IEEE 4th Global Conference on Consumer Electronics* (*GCCE*), pp.603–604, DOI: 10.1109/GCCE.2015.7398682 (2015).

[7] Kurihara, S., Fukuda, S., Oguchi, M. and Yamaguch, S.: Estimation of Power Consumption of Each Application Caused by Device Lock Considering Software Dependency in Smartphones, *2017 5th International Symposium on Computing and Networking* (*CANDAR*), pp.560–564, DOI: 10.1109/CANDAR.2017.56 (2017).

[8] Kurihara, S., Fukuda, S., Kamiyama, T., Fukuda, A., Oguchi, M. and Yamaguchi, S.: Estimation of Power Consumption of Each Application Considering Software Dependency in Android and its Evaluation with Practical Applications, *CDS21* (2018) (in Japanese).

[9] Corral, L., Georgiev, A.B., Sillitti, A. and Succi, G.: A method for characterizing energy consumption in Android smartphones, *2nd International Workshop on Green and Sustainable Software* (*GREENS 2013*) (May 2013).

[10] Kaneda, Y., Okuhira, T., Ishihara, T., Hisazumi, K., Kamiyama, T. and Katagiri, M.: A run-time power analysis method using OS-observable parameters for mobile terminals, *Proc. ICESIT*, Vol.2010, No.1, p.39 (2010).

[11] Murmuria, R., Medsger, J. and Stavrou, A.: Mobile Application and Device Power Usage Measurements, *IEEE 6th International Conference on Software Security and Reliability* (*SERE 2012*) (June 2012).

[12] Friedman, R., Kogan, A. and Krivolapov, Y.: On Power and Throughput Tradeoffs of WiFi and Bluetooth in Smartphones, *IEEE Trans. Mobile Computing*, Vol.12, No.7, pp.1363–1376 (2013).

[13] Murakami, T., Kurihara, S., Fukuda, S., Oguchi, M. and Yamaguchi, S.: Saving power consumption of smartphones in the screen-off state with disabling the Wi-Fi, *2018 IEEE International Conference on Consumer Electronics* (*ICCE*), pp.1–6, DOI: 10.1109/ICCE.2018.8326265 (2018).

[14] PowerManager | Android Developers, available from ⟨https://developer.android.com/reference/android/os/PowerManager⟩ (accessed 2018-09-21).

[15] Murakami, T., Kamiyama, T., Fukuda, A., Oguchi, M. and Yamaguchi, S.: BET Estimation Accuracy on Intermittent Disabling Network Device for Saving Smartphones Power Consumption, *IEEE International Conference on Consumer Electronics* (*IEEE 2018 ICCE-TW*) (2018).

[16] Wu, H., Yang, S. and Rountev, A.: Static detection of energy defect patterns in Android applications, *Proc. 25th International Conference on Compiler Construction* (*CC 2016*), pp.185–195, ACM (2016).

[17] Singhai, A., Bose, J. and Yendeti, N.: Reducing power consumption in android applications, *2014 IEEE International Advance Computing Conference* (*IACC*), pp.668–673 (2014).

[18] Couto, M., Cunha, J., Fernandes, J.P., Pereira, R. and Saraiva, J.: GreenDroid: A tool for analysing power consumption in the android ecosystem, *2015 IEEE 13th International Scientific Conference on Informatics*, pp.73–78 (2015).

[19] Bao, L., Lo, D., Xia, X., Wang, X. and Tian, C.: How Android App Developers Manage Power Consumption? - An Empirical Study by Mining Power Management Commits, *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories* (*MSR*), pp.37–48 (2016).

[20] Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R.P., Mao, Z.M. and Yang, L.: Accurate online power estimation and automatic battery behavior based power model generation for smartphones, *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (*CODES+ISSS*), pp.105–114 (2010).

[21] Mittal, R., Kansal, A. and Chandra, R.: Empowering developers to estimate app energy consumption, *Proc. 18th Annual International Conference on Mobile Computing and Networking* (*Mobicom '12*), pp.317–328, ACM (2012).

[22] Alessio, M., Mauro, M. and Paolo, F.: Measuring and Estimating Power Consumption in Android to Support Energy-based Intrusion Detection, Security and High Performance Computing Systems, *Journal of Computer Security*, Vol.23, No.5, pp.611–637, DOI: 10.3233/JCS-150530 (2015).

[23] Furusho, H., Hisazumi, K., Kamiyama, T., Inamura, H., Nakanishi, T. and Fukuda, A.: Power Consumption Profiling Method based on Android Application Usage, Kim, K. (Ed.): Information Science and Applications, *Lecture Notes in Electrical Engineering*, Vol.339, Springer, Berlin, Heidelberg (2015).

[24] Nagata, K. and Yamaguchi, S.: An Android application launch analyzing system, *2012 8th International Conference on Computing Technology and Information Management* (*NCM and ICNIT*), pp.76–81 (2012).

[25] Hayakawa, A., Nakarai, A., Takemori, K., Yamaguchi, S. and Oguchi, M.: An Evaluation of Causal Association between the Issue of Broadcast-Intent and Applications for the Sake of Electric Power Saving of Android Terminals, *IC2014* (2014) (in Japanese).

[26] Nakamura, Y., Hayakawa, A., Takemori, K., Nakarai, A., Oguchi, M. and Yamaguchi, S.: A Study on Power Consumption Caused by

Broadcast Intent in Android, The Special Interest Group Technical Reports of IPSJ 2014-DBS-159 DBS, IPSJ (2014) (in Japanese).
[27] Enck, W., Ongtang, M. and McDaniel, P.: Understanding Android Security, *IEEE Security & Privacy*, Vol.7, No.1, pp.50–57, DOI: 10.1109/MSP.2009.26 (2009).
[28] Beresford, A.R., Rice, A., Skehin, N. and Sohan, R.: MockDroid: trading privacy for application functionality on smartphones, *Proc. 12th Workshop on Mobile Computing Systems and Applications* (*HotMobile '11*), pp.49–54, ACM, DOI: 10.1145/2184489.2184500 (2011).
[29] Kamiyama, T., Hisazumi, K., Inamura, H., Konishi, T., Ohta, K. and Fukuda, A.: Smartphone Usage Analysis Based on Actual-Use Survey, *Proc. 8th EAI International Conference on Mobile Computing, Applications and Services* (*MobiCASE'16*) (2016).

Appendix

Figures A·1 to **A·6** show the history of starts and ends of WakeLock of each application of the Scenarios 1 to 6 in Section 5.5, respectively.

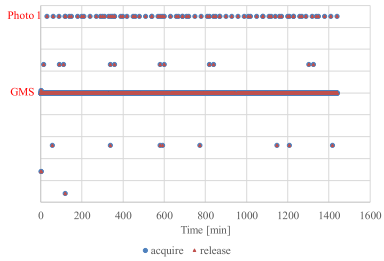


Fig. A·1 WakeLocks of each application (Scenario 1).

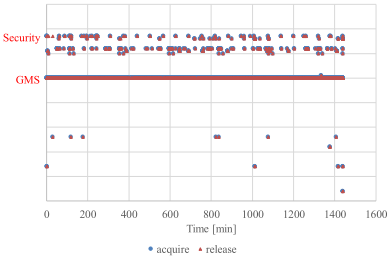


Fig. A·2 WakeLocks of each application (Scenario 2).

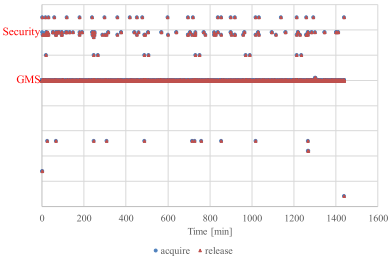


Fig. A·3 WakeLocks of each application (Scenario 3).

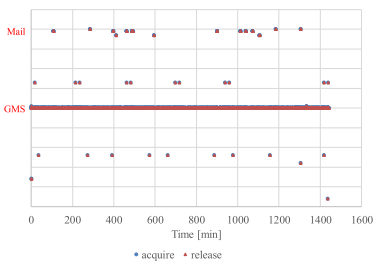


Fig. A·4 WakeLocks of each application (Scenario 4).

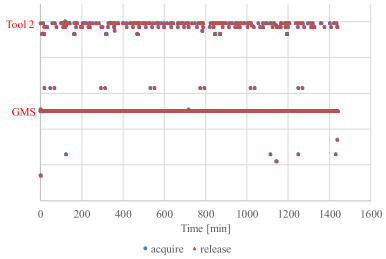


Fig. A·5 WakeLocks of each application (Scenario 5).

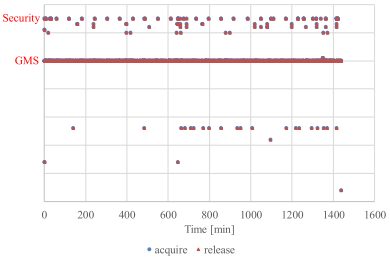


Fig. A·6 WakeLocks of each application (Scenario 6).

Shun Kurihara received his B.E. and M.E. degrees in Kogakuin University in 2016 and 2018, respectively.

Shoki Fukuda received his B.E. and M.E. degrees in Kogakuin University in 2016 and 2018, respectively.

Takeshi Kamiyama joined NTT DOCOMO, Inc. in 2006. His research interests include system research, especially energy-efficient design, on mobile device and distributed system. Before NTT DOCOMO, he received M.S. degree in University of Tokyo, Japan in 2006. Also, he was co-founder and CEO in e-jis, Inc. from 2003 to 2006. Currently, he is also a doctoral student in Kyushu University. He is a member of IPSJ.

Akira Fukuda received his B.Eng, M.Eng, and Ph.D. degrees in computer science and communication engineering from Kyushu University, Japan, in 1977, 1979, and 1985, respectively. From 1977 to 1981, he worked for the Nippon Telegraph and Telephone Corporation, where he engaged in research on performance evaluation of computer systems and the queueing theory. From 1981 to 1991 and from 1991 to 1993, he worked for the Department of Information Systems and the Department of Computer Science and Communication Engineering, Kyushu University, Japan, respectively. In 1994, he joined Nara Institute of Science and Technology, Japan, as a professor. Since 2001, 2008, and 2016, he has been a professor of Graduate School of Information Science and Electrical Engineering, and director of System LSI Research Center, and director of R&D Center for Smart Mobility, Kyushu Univ., Japan, respectively. Since 2015, he has been a distinguished professor of Kyushu Univ. His research interests include embedded systems, ubiquitous computing, system software (operating systems, compiler, and run-time systems), parallel and distributed systems, and performance evaluation. He is IPSJ fellow. He is a member of ACM, IEEE Computer Society, IEICE, IPSJ, and Operations Research Society of Japan.

Masato Oguchi received B.E. from Keio University, M.E. and Ph.D. from the University of Tokyo in 1990, 1992, and 1995 respectively. In 1995, he was a Researcher at the National Center for Science Information System (NACSIS), currently known as National Institute of Informatics (NII). From 1996 to 2000, he was a Research Fellow at the Institute of Industrial Science, University of Tokyo. From 2000 to 2003, he was an Associate Professor at the Research and Development Initiative, Chuo University. He joined Ochanomizu University as an Associate Professor in 2003 and becomes Professor since 2006. His areas of interest include network computing middleware, high-performance computing, and mobile networking. He is a member of IEEE, ACM, IEICE, and IPSJ.

Saneyasu Yamaguchi received Engineering Doctor's degree (Ph.D.) at Tokyo University in 2002. During 2002–2006, he stayed in Institute of Industrial Science, the University of Tokyo to study I/O processing. He now with Kogakuin University. Currently his researches focus on operating systems, virtualized systems, and storage system.